OPTIMISATION NOTES

A COMPILATION OF LECTURE NOTES
FROM GRADUATE-LEVEL
OPTIMISATION COURSES

WRITTEN BY

FABRICIO OLIVEIRA

ASSOCIATE PROFESSOR OF OPERATIONS RESEARCH
AALTO UNIVERSITY, SCHOOL OF SCIENCE

SOURCE CODE AVAILABLE AT
github.com/gamma-opt/optimisation-notes

Contents

I Linear optimisation **11**

1 Introduction **13**

1.1 What is optimisation? . . . 13

1.1.1 Mathematical programming and optimisation . . . 13

1.1.2 Types of mathematical optimisation models . . . 14

1.2 Linear programming applications . . . 15

1.2.1 Resource allocation . . . 15

1.2.2 Transportation problem . . . 17

1.2.3 Production planning (lot-sizing) . . . 19

1.3 The geometry of LPs - graphical method . . . 20

1.3.1 The graphical method . . . 20

1.3.2 Geometrical properties of LPs . . . 21

1.4 Exercises . . . 23

2 Basics of Linear Algebra **27**

2.1 Basics of linear problems . . . 27

2.1.1 Subspaces and bases . . . 28

2.1.2 Affine subspaces . . . 30

2.2 Convex polyhedral set . . . 31

2.2.1 Hyperplanes, half-spaces and polyhedral sets . . . 31

2.2.2 Convexity of polyhedral sets . . . 32

2.3 Extreme points, vertices, and basic feasible solutions . . . 34

2.4 Exercises . . . 39

3 Basis, Extreme Points and Optimality in Linear Programming **41**

3.1 Polyhedral sets in standard form . . . 41

3.1.1 The standard form of linear programming problems . . . 41

3.1.2 Forming bases for standard-form linear programming problems . . . 43

3.1.3 Adjacent basic solutions . . . 44

3.1.4 Redundancy and degeneracy . . . 45

3.2 Optimality of extreme points . . . 46
3.2.1 The existence of extreme points . . . 47
3.2.2 Finding optimal solutions . . . 48
3.2.3 Moving towards improved solutions . . . 50
3.2.4 Optimality conditions . . . 52
3.3 Exercises . . . 54

4 The simplex method **57**
4.1 Developing the simplex method . . . 57
4.1.1 Calculating step sizes . . . 57
4.1.2 Moving between adjacent bases . . . 58
4.1.3 A remark on degeneracy . . . 60
4.2 Implementing the simplex method . . . 61
4.2.1 Pivot or variable selection . . . 61
4.2.2 The revised simplex method . . . 62
4.2.3 Tableau representation . . . 64
4.2.4 Generating initial feasible solutions . . . 66
4.3 Column geometry of the simplex method . . . 68
4.4 Exercises . . . 72

5 Linear Programming Duality - Part I **75**
5.1 Formulating duals . . . 75
5.1.1 Motivation . . . 75
5.1.2 General form of duals . . . 76
5.2 Duality theory . . . 79
5.2.1 Weak duality . . . 79
5.2.2 Strong duality . . . 80
5.3 Geometric interpretation of duality . . . 81
5.3.1 Complementary slackness . . . 81
5.3.2 Dual feasibility and optimality . . . 82
5.4 Practical uses of duality . . . 83
5.4.1 Optimal dual variables as marginal costs . . . 83
5.4.2 The dual simplex method . . . 84
5.5 Exercises . . . 88

6 Linear Programming Duality - Part II **91**
6.1 Sensitivity analysis . . . 91
6.1.1 Adding a new variable . . . 92

6.1.2 Adding a new constraint . . . 93
6.1.3 Changing input data . . . 94
6.2 Cones and extreme rays . . . 96
6.2.1 Recession cones and extreme rays . . . 98
6.2.2 Unbounded problems . . . 98
6.2.3 Farkas' lemma . . . 100
6.3 Resolution theorem* . . . 101
6.4 Exercises . . . 103

7 Decomposition methods **105**
7.1 Large-scale problems . . . 105
7.2 Dantzig-Wolfe decomposition and column generation* . . . 107
7.2.1 Dantzig-Wolfe decomposition . . . 107
7.2.2 Delayed column generation . . . 110
7.3 Benders decomposition . . . 111
7.4 Exercises . . . 116

8 Integer programming models **119**
8.1 Types of integer programming problems . . . 119
8.2 (Mixed-)integer programming applications . . . 120
8.2.1 The assignment problem . . . 120
8.2.2 The knapsack problem . . . 121
8.2.3 The generalised assignment problem . . . 121
8.2.4 The set covering problem . . . 122
8.2.5 Travelling salesperson problem . . . 124
8.2.6 Uncapacitated facility location . . . 125
8.2.7 Uncapacitated lot-sizing . . . 127
8.3 Good formulations . . . 127
8.3.1 Comparing formulations . . . 128
8.4 Exercises . . . 130

9 Branch-and-bound method **133**
9.1 Optimality for integer programming problems . . . 133
9.2 Relaxations . . . 133
9.2.1 Linear programming relaxation . . . 135
9.2.2 Relaxation for combinatorial optimisation . . . 136
9.3 Branch-and-bound method . . . 137
9.3.1 Bounding in enumerative trees . . . 138

9.3.2 Linear-programming-based branch-and-bound . . . 139
9.4 Exercises . . . 144

10 Cutting-planes method **147**
10.1 Valid inequalities . . . 147
10.2 The Chvátal-Gomory procedure . . . 148
10.3 The cutting-plane method . . . 150
10.4 Gomory's fractional cutting-plane method . . . 151
10.5 Obtaining stronger inequalities . . . 154
10.5.1 Strong inequalities . . . 154
10.5.2 Strengthening 0-1 knapsack inequalities . . . 156
10.6 Exercises . . . 157

11 Mixed-integer programming solvers **161**
11.1 Modern mixed-integer linear programming solvers . . . 161
11.2 Presolving methods . . . 162
11.3 Cut generation . . . 165
11.3.1 Cut management: generation, selection and discarding . . . 166
11.4 Variable selection: branching strategy . . . 166
11.5 Node selection . . . 169
11.6 Primal heuristics . . . 170
11.6.1 Diving heuristics . . . 171
11.6.2 Local searches and large-neighbourhood searches . . . 172
11.7 Exercises . . . 175

II Nonlinear optimisation **177**

12 Introduction **179**
12.1 What is optimisation? . . . 179
12.1.1 Mathematical programming and optimisation . . . 179
12.1.2 Types of mathematical optimisation models . . . 180
12.2 Examples of applications . . . 181
12.2.1 Resource allocation and portfolio optimisation . . . 181
12.2.2 The pooling problem: refinery operations planning . . . 182
12.2.3 Robust optimisation . . . 183
12.2.4 Classification: support-vector machines . . . 185

13 Convex sets **189**

13.1 Convexity and optimisation . . . 189
13.2 Identifying convexity of sets . . . 189
13.2.1 Convexity-preserving set operations . . . 190
13.2.2 Examples of convex sets . . . 191
13.3 Convex hulls . . . 193
13.4 Closure and interior of sets . . . 194
13.4.1 Closure, interior and boundary of a set . . . 194
13.4.2 The Weierstrass theorem . . . 196
13.5 Separation and support of sets . . . 196
13.5.1 Hyperplanes and closest points . . . 197
13.5.2 Halfspaces and separation . . . 197
13.5.3 Farkas' theorem . . . 199
13.5.4 Supporting hyperplanes . . . 200

14 Convex functions **203**
14.1 Convexity in functions . . . 203
14.1.1 Example of convex functions . . . 203
14.1.2 Convex functions and their level sets . . . 204
14.1.3 Convex functions and their epigraphs . . . 205
14.2 Differentiability of functions . . . 206
14.2.1 Subgradients and supporting hyperplanes . . . 206
14.2.2 Differentiability and gradients for convex functions . . . 206
14.2.3 Second-order differentiability . . . 208
14.3 Quasiconvexity . . . 209

15 Unconstrained optimality conditions **213**
15.1 Recognising optimality . . . 213
15.2 The role of convexity in optimality conditions . . . 213
15.3 Optimality condition of convex problems . . . 215
15.3.1 Optimality conditions for unconstrained problems . . . 218

16 Unconstrained optimisation methods: part 1 **221**
16.1 A prototype of an optimisation method . . . 221
16.2 Line search methods . . . 221
16.2.1 Exact line searches . . . 222
16.2.2 Inexact line search . . . 226
16.3 Unconstrained optimisation methods . . . 227
16.3.1 Coordinate descent . . . 228

16.3.2 Gradient (descent) method . . . 228
16.3.3 Newton's method . . . 230

17 Unconstrained optimisation methods: part 2 235
17.1 Unconstrained optimisation methods . . . 235
17.1.1 Conjugate gradient method . . . 235
17.1.2 Quasi Newton: BFGS method . . . 240
17.2 Complexity, convergence and conditioning . . . 242
17.2.1 Complexity . . . 243
17.2.2 Convergence . . . 244
17.2.3 Conditioning . . . 245

18 Constrained optimality conditions 249
18.1 Optimality for constrained problems . . . 249
18.1.1 Inequality constrained problems . . . 250
18.2 Fritz-John conditions . . . 251
18.3 Karush-Kuhn-Tucker conditions . . . 252
18.4 Constraint qualification . . . 253

19 Lagrangian duality 257
19.1 The concept of relaxation . . . 257
19.2 Lagrangian dual problems . . . 258
19.2.1 Weak and strong duality . . . 258
19.2.2 Employing Lagrangian duality for solving optimisation problems . . . 264
19.2.3 Saddle point optimality and KKT conditions* . . . 265
19.3 Properties of Lagrangian functions . . . 267
19.3.1 The subgradient method . . . 268

20 Penalty methods 271
20.1 Penalty functions . . . 271
20.1.1 Geometric interpretation . . . 272
20.1.2 Penalty function methods . . . 273
20.2 Augmented Lagrangian method of multipliers . . . 275
20.2.1 Augmented Lagrangian method of multipliers . . . 277
20.2.2 Alternating direction method of multipliers - ADMM . . . 278

21 Barrier methods 279
21.1 Barrier functions . . . 279
21.2 The barrier method . . . 280

21.3 Interior point method for LP/QP problems 282
21.3.1 Primal/dual path-following interior point method 285

22 Primal methods **289**
22.1 The concept of feasible directions 289
22.2 Conditional gradient - the Frank-Wolfe method 289
22.3 Sequential quadratic programming 291
22.4 Generalised reduced gradient* 295
22.4.1 Wolfe's reduced gradient 295
22.4.2 Generalised reduced gradient method 297

Part I

Linear optimisation

Chapter 1

Introduction

1.1 What is optimisation?

Optimisation is one of these words that has many meanings, depending on the context you take as a reference. In the context of this book, optimisation refers to *mathematical optimisation*, which is a discipline of applied mathematics.

In mathematical optimisation, we build upon concepts and techniques from calculus, analysis, linear algebra, and other domains of mathematics to develop methods to find values for variables (or solutions) within a given domain that maximise (or minimise) the value of a function. In specific, we are trying to solve the following general problem:

$$\begin{aligned} &\text{min. } f(x) \\ &\text{s.t.: } x \in X. \end{aligned} \tag{1.1}$$

That is, we would like to find the solution x that *minimises* the value of the *objective function* f, such that (s.t.) x belongs to the *feasibility set* X. In a general sense, problems like this can be solved by employing the following strategy:

1. Analysing properties of the function $f(x)$ under specific domains and deriving the conditions that must be satisfied such that a point x is a candidate optimal point.
2. Applying numerical methods that iteratively searches for points satisfying these conditions.

This idea is central in several knowledge domains and often is described with area-specific nomenclature. Fields such as economics, engineering, statistics, machine learning and, perhaps more broadly, operations research, are intensive users and developers of optimisation theory and applications.

1.1.1 Mathematical programming and optimisation

Operations research and mathematical optimisation are somewhat intertwined, as they both were born around a similar circumstance.

I personally like to separate *mathematical programming* from (mathematical) *optimisation*. Mathematical programming is a modelling paradigm in which we rely on (very powerful, I might add) analogies to model *real-world* problems. In that, we look at a given decision problem considering that:

- *variables* represent *decisions*, as in a business decision or a course of action. Examples include setting the parameter of (e.g., prediction) model, production systems layouts, geometries of structures, topologies of networks, and so forth;

- *domain* represents business rules or *constraints*, representing logic relations, design or engineering limitations, requirements, and such;
- function is an *objective function* that provides a measure of solution quality.

With these in mind, we can represent the decision problem as a *mathematical programming model* of the form of (1.1) that can be solved using *optimisation* methods. From now on, we will refer to this specific class of models as mathematical optimisation models, or optimisation models for short. We will also use the term to *solve the problem* to refer to the task of finding optimal solutions to optimisation models.

This book mostly focuses on the optimisation techniques employed to find optimal solutions for these models. As we will see, depending on the nature of the functions f and g that are used to formulate the model, some methods might be more or less appropriate. Further complicating the issue, for models of a given nature, there might be alternative algorithms that can be employed and with no generalised consensus on whether one method is generally better performing than another, which is one of the aspects that make optimisation so exciting and multifaceted when it comes to alternative approaches. I hope that this makes more sense as we progress through the chapters.

1.1.2 Types of mathematical optimisation models

In general, the simpler the assumptions on the parts forming the optimisation model, the more efficient the methods to solve such problems.

Let us define some additional notation that we will use from now on. Consider a model in the general form

$$\begin{aligned} \text{min. } & f(x) \\ \text{s.t.: } & g_i(x) \leq 0,\ i = 1, \dots, m \\ & h_i(x) = 0,\ i = 1, \dots, l \\ & x \in X, \end{aligned}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is the objective function, $g : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a collection of m inequality constraints and $h : \mathbb{R}^n \mapsto \mathbb{R}^l$ is a collection of l equality constraints.

In fact, every inequality constraint can be represented by an equality constraint by making $h_i(x) = g_i(x) + x_{n+1}$ and augmenting the decision variable vector $x \in \mathbb{R}^n$ to include the slack variable x_{n+1}. However, since these constraints behave very differently from an algorithmic standpoint, we will explicitly represent both whenever necessary.

The most general types of models are the following. We also use this as an opportunity to define some (admittedly confusing) nomenclature from the field of operations research that we will be using in these notes.

1. *Unconstrained models:* in these, the set $X = \mathbb{R}^n$ and $m = l = 0$. These are prominent in, e.g., machine learning and statistics applications, where f represents a measure of model fitness or prediction error.
2. *Linear programming (LP):* presumes linear objective function $f(x) = c^\top x$ and affine constraints g and h, i.e., of the form $a_i^\top x - b_i$, with $a_i \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Normally, $X = \{x \in \mathbb{R}^n \mid x_j \geq 0, j = 1, \dots, n\}$ enforce that decision variables are constrained to be the non-negative orthant.

3. *Nonlinear programming (NLP):* some or all of the functions f, g, and h are nonlinear.

4. *Mixed-integer (linear) programming (MIP):* consists of an LP in which some (or all, being then simply integer programming) of the variables are constrained to be integers. In other words, $X \subseteq \mathbb{R}^k \times \mathbb{Z}^{n-k}$. Very frequently, the integer variables are constrained to be binary terms, i.e., $x_i \in \{0, 1\}$, for $i = 1, \ldots, n-k$ and are meant to represent true-or-false or yes-or-no conditions.

5. *Mixed-integer nonlinear programming (MINLP):* are the intersection of MIPs and NLPs.

Remark: notice that we use the vector notation $c^\top x = \sum_{j \in J} c_j x_j$, with $J = \{1, \ldots, N\}$. This is just a convenience for keeping the notation compact.

1.2 Linear programming applications

We will consider now a few examples of liner programming models with somewhat general structure. Many of these examples have features that can be combined into more general models.

1.2.1 Resource allocation

Most linear programming (LP) problems can be interpreted as a *resource allocation* problem. In that, we are interested in defining an optimal allocation of resources (i.e., a plan) that maximises return or minimise costs and satisfy allocation rules.

Specifically, let $I = \{1, \ldots, i, \ldots, M\}$ by a set of resources that can be combined to produce products in the set $J = \{1, \ldots, j, \ldots, N\}$. Assume that we are given a return per unit of product c_j, $\forall j \in J$, and a list of a_{ij}, $\forall i \in I, \forall j \in J$, describing which and how much of the resources $i \in I$ are required for making product $j \in J$. Assume that the availability of resource b_i, $\forall i \in I$, is known.

Our objective is to define the amounts x_j representing the production of $j \in J$. We would like to define those in a way that we optimise the resource allocation plan quality (in our case, maximise return from the production quantities x_j) while making sure the amount produced do not exceed the availability of resources. For that, we need to define:

The *objective function*, which measures the *quality* of a production plan. In this case, the total return for a given plan is given by:

$$\text{max.} \sum_{j \in J} c_j x_j \Rightarrow c^\top x,$$

where $c = [c_1, \ldots, c_N]^\top$ and $x = [x_1, \ldots, x_N]^\top$ are n-sized vectors. Notice that $c^\top x$ denotes the inner (or dot) product. The transpose sign $^\top$ is meant to reinforce that we see our vectors as column vectors, unless otherwise stated.

Next, we need to define *constraints* that state the conditions for a plan to be *valid*. In this context, a valid plan is a plan that does not utilise more than the amount of available resources b_i, $\forall i \in I$. This can be expressed as the collection (one for each $i \in I$) of affine (more often wrongly called, as we will too, linear) inequalities

$$\text{s.t.:} \sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I \Rightarrow Ax \leq b,$$

where a_{ij} are the components of the $M \times N$ matrix A and $b = [b_1, \ldots, b_M]^\top$. Furthermore, we also must require that $x_i \geq 0, \forall i \in I$.

Combining the above, we obtain the generic formulation that will be used throughout this text to represent linear programming models:

$$\text{max. } c^\top x \tag{1.2}$$
$$\text{s.t.: } Ax \leq b \tag{1.3}$$
$$x \geq 0. \tag{1.4}$$

Illustrative example: the paint factory problem

Let us work on a more specific example that will be useful for illustrating some important concepts related to the geometry of linear programming problems.

Let us consider a paint factory produces *exterior* and *interior paint* from raw materials *M1* and *M2*. The *maximum demand* for interior paint is 2 tons/day. Moreover, the amount of interior paint produced *cannot exceed* that of exterior paint by more than 1 ton/day.

Our goal is to determine the optimal paint production plan. Table 1.1 summarises the data to be considered. Notice the constraints that must be imposed to represent the daily availability of paint.

	material (ton)/paint (ton)		
	exterior paint	interior paint	daily availability (ton)
material M1	6	4	24
material M2	1	2	6
profit ($1000 /ton)	5	4	

Table 1.1: Paint factory problem data

The paint factory problem is an example of a resource allocation problem. Perhaps one aspect that is somewhat dissimilar is the constraint representing the production rules regarding the relative amounts of exterior and interior paint. Notice, however, that this type of constraint also has the same format as the more straightforward resource allocation constraints.

Let x_1 be the amount produced of exterior paints (in tons) and x_2 the amount of interior paints. The complete model that optimises the daily production plan of the paint factory is:

$$\text{max. } z = 5x_1 + 4x_2 \tag{1.5}$$
$$\text{s.t.: } 6x_1 + 4x_2 \leq 24 \tag{1.6}$$
$$x_1 + 2x_2 \leq 6 \tag{1.7}$$
$$x_2 - x_1 \leq 1 \tag{1.8}$$
$$x_2 \leq 2 \tag{1.9}$$
$$x_1, x_2 \geq 0 \tag{1.10}$$

Notice that paint factory model can also be *compactly represented* as in (1.2)–(1.4), where

$$c = [5, 4]^\top,\ x = [x_1, x_2]^\top,\ A = \begin{bmatrix} 6 & 4 \\ 1 & 2 \\ -1 & 1 \\ 0 & 1 \end{bmatrix},\ \text{and } b = [24, 6, 1, 2]^\top.$$

1.2.2 Transportation problem

Another important class of linear programming problems are those known as transportation problems. These problems are often modelled using the abstraction of graphs since they consider a network of nodes and arcs through which some flow must be optimised. Transportation problems have several important characteristics that can be exploited to design specialised algorithms, the so-called *transportation simplex* method. Although we will not discuss these methods in this text, the simplex method (and its variant, the dual simplex method) will be at the centre of our developments later on. Also, modern solvers have increasingly relegated transport simplex methods in their development, as dual simplex has consistently shown to perform similarly in the context of transportation problems, despite being a far more general method.

The problem can be summarised as follows. We would like to plan the production and distribution of a certain product, taking into account that the transportation cost is known (e.g., proportional to the distance travelled), the factories (or source nodes) have a capacity limit, and the clients (or demand nodes) have known demands. Figure 1.1 illustrates a small network with two factories, located in San Diego and Seattle, and three demand points, located in New York, Chicago, and Miami. Table 1.2 presents the data related to the problem.

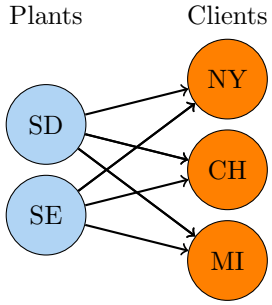


Figure 1.1: Schematic illustration of a network with two source nodes and three demand nodes

	Clients			
Factory	NY	Chicago	Miami	Capacity
Seattle	2.5	1.7	1.8	350
San Diego	3.5	1.8	1.4	600
Demands	325	300	275	-

Table 1.2: Problem data: unit transportation costs, demands and capacities

To formulate a linear programming model representing the transportation problem, let $i \in I = \{\text{Seattle}, \text{San Diego}\}$ be the index set representing factories. Similarly, let $j \in J = \{\text{New York}, \text{Chicago}, \text{Miami}\}$.

The decisions, in this case, are represented by x_{ij}, which represents the amount produced in factory i and sent to client j. Such a distribution plan can then be assessed by its total transportation cost, which is given by

$$\text{min. } z = 2.5x_{11} + 1.7x_{12} + 1.8x_{13} + 3.5x_{21} + 1.9x_{22} + 1.4x_{23}.$$

The total transportation cost can be more generally represented as

$$\text{min. } z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$$

where c_{ij} is the unit transportation cost from i to j. The problem has two types of constraints that must be observed, relating to the supply capacity and demand requirements. These can be stated as the following linear constraints

$$\begin{aligned}
&x_{11} + x_{12} + x_{13} \leq 350 \text{ (capacity limit in Seattle)} \\
&x_{21} + x_{22} + x_{23} \leq 600 \text{ (capacity limit in San Diego)} \\
&x_{11} + x_{21} \geq 325 \text{ (demand in New York)} \\
&x_{12} + x_{22} \geq 300 \text{ (demand in Chicago)} \\
&x_{13} + x_{23} \geq 275 \text{ (demand in Miami).}
\end{aligned}$$

These constraints can be expressed in the more compact form

$$\sum_{j \in J} x_{ij} \leq C_i, \ \forall i \in I \tag{1.11}$$

$$\sum_{i \in I} x_{ij} \geq D_j, \ \forall j \in J, \tag{1.12}$$

where C_i is the production capacity of factory i and D_j is the client demand j. Notice that the terms on the lefthand side in (1.11) accounts for the total production in each of the source nodes $i \in I$. Analogously, in constraint (1.12), the term on the left accounts for the total of the demand satisfied at the demand nodes $j \in J$.

Using an optimality argument, we can see that any solution for which, for any $j \in J$, $\sum_{i \in I} x_{ij} > D_j$ can be improved by making $\sum_{i \in I} x_{ij} = D_j$. This show that his constraint under these conditions will always be satisfied as an equality constraint instead and could be replaced like such.

The complete transportation model for the example above can be stated as

$$\begin{aligned}
\text{min. } z = {} & 2.5x_{11} + 1.7x_{12} + 1.8x_{13} + 3.5x_{21} + 1.9x_{22} + 1.4x_{23} \\
\text{s.t.: } & x_{11} + x_{12} + x_{13} \leq 350, \ x_{21} + x_{22} + x_{23} \leq 600 \\
& x_{11} + x_{21} \geq 325, \ x_{12} + x_{22} \geq 300, \ x_{13} + x_{23} \geq 275 \\
& x_{11}, \ldots, x_{23} \geq 0.
\end{aligned}$$

Or, more compactly, in the so-called *algebraic (or symbolic) form*

$$\begin{aligned}
\text{min. } z = {} & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\
\text{s.t.: } & \sum_{j \in J} x_{ij} \leq C_i, \ \forall i \in I \\
& \sum_{i \in I} x_{ij} \geq D_j, \ \forall j \in J \\
& x_{ij} \geq 0, \forall i \in I, \ \forall j \in J.
\end{aligned}$$

One interesting aspect to notice regarding algebraic forms is that they allow to represent the main structure of the model while being independent of the instance being considered. For example,

regardless of whether the instance would have 5 or 50 nodes, the algebraic formulation is the same, allowing for detaching the problem instance (in our case the 5 node network) from the model itself. Moreover, most computational tools for mathematical programming modelling (hereinafter referred to simply as modelling - like JuMP) empowers the user to define the optimisation model using this algebraic representation.

Algebraic forms are the main form in which we will specify optimisation models. This abstraction is a peculiar aspect of mathematical programming and is perhaps one of its main features, the fact that one must *formulate* models for each specific setting, which can be done in multiple ways and might have consequences for how well an algorithm performs computationally. Further in this text, we will discuss this point in more detail.

1.2.3 Production planning (lot-sizing)

Production planning problems, commonly referred to as lot-sizing problems in contexts related to industrial engineering, consider problems where a planning horizon is taken into consideration. Differently from the previous examples, lot-sizing problems allow for the consideration of a time flow aspect, in which production that takes place in the past can be "shifted" to a future point in time by means of inventories (i.e., stocks). Inventories are important because they allow for taking advantage of different prices at different time periods, circumventing production capacity limitations, or preparing against uncertainties in the future (e.g., uncertain demands).

The planning horizon is represented by a collection of chronologically ordered elements $t \in \{1, \ldots, T\}$ representing a set of uniformly-sized time periods (e.g., months or days). Then, let us define the decision variables p_t as the amount produced in period t and k_t the amount stored in period t, which is available for use in periods $t' > t$. These decisions are governed by two costs: P_t, $\forall t \in T$, representing the production cost in each time period t and the unit holding cost H, that is, how much it costs to hold one unit of product for one time period.

Our objective is to satisfy the demands D_t, $\forall t \in T$, at the minimum possible cost. Figure 1.2 provides a schematic representation of the process to be modelled. Notice that each node represents a material balance to be considered, that is, that at any period t, the total produced plus the amount held in inventory from the previous period $(t-1)$ must be the same as the amount used to satisfy the demand plus the amount held in inventory for the next period $(t+1)$.

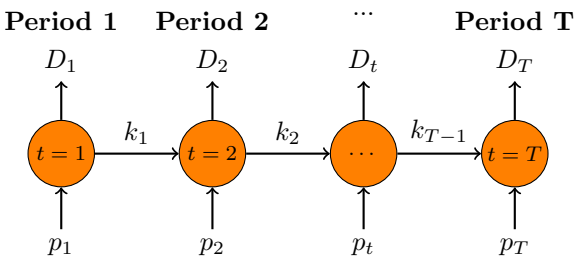


Figure 1.2: A schematic representation of the lot-sizing problem. Each node represents the material balance at each time period t.

The production planning problem can be formulated as

$$\begin{aligned} \text{min. } & \sum_{t \in T} [C_t p_t + H s_t] \\ \text{s.t.: } & p_t + k_{t-1} = D_t + k_t, \ \forall t \in T \\ & p_t, h_t \geq 0, \ \forall t \in T. \end{aligned}$$

A few points must be considered carefully when dealing with lot-sizing problems. First, one must carefully consider boundary condition, that is, what the model is deciding in time periods $t = T$ and what is the initial inventory (carried from $t = 0$). While the former will be seen by the model as the "end of the world" and thus will realise that optimal inventory levels at period $|T|$ must be zero, the latter might considerably influence how much production is needed during the planning horizon T. These must be observed and handled accordingly.

1.3 The geometry of LPs - graphical method

Let us now focus our attention to the geometry of linear programming (LP) models. As it will become evident later on, LP models have a very peculiar geometry that is exploited by one of the most widespread methods to solve them, the *simplex method*.

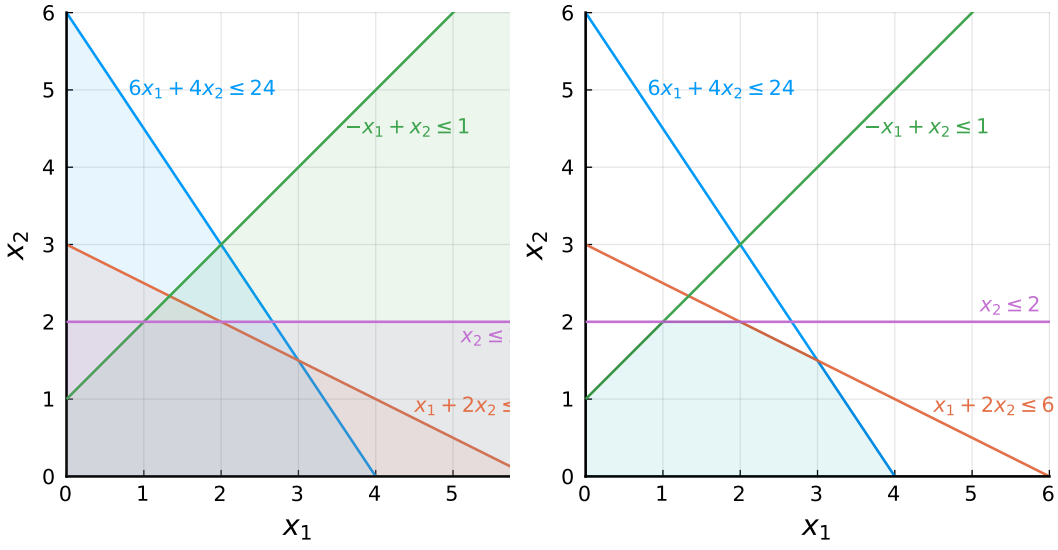
1.3.1 The graphical method

In order to create a geometric intuition, we will utilise a graphical representation of the resource allocation example (the paint factory problem). But first, recall the general LP formulation (1.2)–(1.4), where A is an $m \times n$ matrix, and b, c, and x have suitable dimensions. Let a_i be one of the m rows of A. Notice each constraint $a_i^\top x \leq b_i$ defines a closed half-space, with boundary defined by a hyperplane $a_i^\top x = b_i$, $\forall i \in I = \{1, \ldots, m\} \equiv [m]$ (we will return to these definitions in chapter 2; for now, just bear with me if these technical terms are unfamiliar to you). By plotting all of these closed half-spaces, we can see that their intersection will form the *feasible region* of the problem, that is, the (polyhedral) set of points that satisfy all constraints $Ax \leq b$. Figure 1.3 provides a graphical representation of the feasible region of the paint factory problem.

We can use this visual representation to find the optimal solution for the problem, that is, the point (x_1, x_2) within the feasible set such that the objective function value is maximal (recall that the paint factory problem is a maximisation problem). For that, we must consider how the objective function $z = c^\top x$ can be represented in the (x_1, x_2)-plane. Notice that the objective function forms a hyperplane in $(x_1, x_2, z) \subset \mathbb{R}^3$, of which we can plot level curves (i.e., projections) onto the (x_1, x_2)-plane. Figure 1.4a shows the plotting of three level curves, for $z = 5, 10$, and 15.

This observation provides us with a simple graphical method to find the optimal solution to linear problems. One must simply sweep the feasible region in the direction of the gradient $\nabla z = [\frac{\partial z}{\partial x_1}, \frac{\partial z}{\partial x_2}]^\top = [5, 4]^\top$ (or in its opposite direction, if minimising) until one last point (or edge) of contact remains, meaning that the whole of the feasible region is behind that furthermost level curve. Figure 1.4b illustrates the process of finding the optimal solution for the paint factory problem.

The graphical method is important because it allows noticing several key features that will be used later on when we analyse a method that can search for optimal solutions for LP problems. The first is related to the notion of active or inactive constraints. We say that a constraint is *active* if, at the



(a) (b)

Figure 1.3: The feasible region of the paint factory problem (in Figure 1.3b), represented as the intersection of the four closed-half spaces formed by each of the constraints (as shown in Figure 1.3a). Notice how the feasible region is a polyhedral set in $\mathbb{R}^2$, as there are two decision variables (x_1 and x_2).

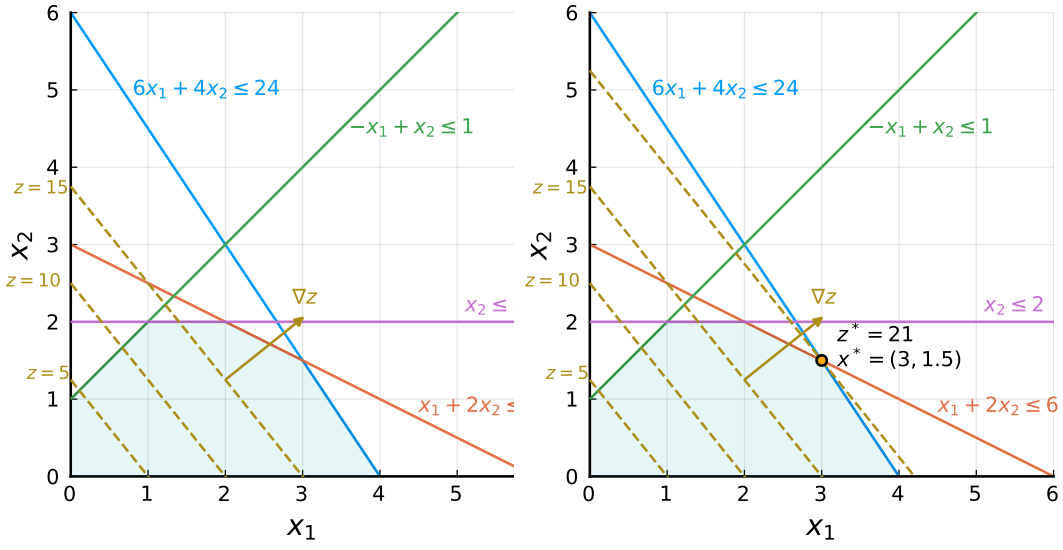
optimum, the constraint is satisfied as equality. For example, the constraints $6x_1 + 4x_2 \leq 24$ and $x_1 + 2x_2 \leq 6$ are active at the optimum $x^* = (3, 1.5)$, since $6(3) + 4(1.5) = 24$ and $3 + 2(1.5) = 6$. An active constraint indicates that the resource (or requirement) represented by that constraint is being fully depleted (or minimally satisfied).

Analogously, *inactive constraint* are constraints that are satisfied as strict inequalities at the optimum. For example, the constraint $-x_1 + x_2 \leq 1$ is inactive at the optimum, as $-(3) + 1.5 < 1$. In this case, an inactive constraint represents a resource (or requirement) that is not fully depleted (or is over-satisfied).

1.3.2 Geometrical properties of LPs

One striking feature concerning the geometry of LPs that becomes evident when we analyse the graphical method is that the number of candidate solutions is not infinite, but yet, only *a finite set* of points are potential candidates for optimal solution. This is because the process of sweeping in the direction of the gradient of the (linear) objective function will, in general, lead to a unique solution that must lie on a vertex of the polyhedral feasible set. The only exceptions are either when the gradient ∇z happens to be perpendicular to a facet of the polyhedral set (and in the direction of the sweeping) or in case the sweeping direction is not bounded by some of the facets of the polyhedral set. These exceptional cases will be discussed in more detail later on, but, as we will see, the observation still holds.

In the graphical example (i.e., in $\mathbb{R}^2$), notice how making $n = 2$ constraints active out of $m = 4$ constraints *forms a vertex*. However, not all vertices are feasible. This allows us to devise a mechanism to describe vertices by activating n of the m constraints at a time, in which we could



(a) (b)

Figure 1.4: Graphical representation of some of the level curves of the objective function $z = 5x_1 + 4x_2$. Notice that the constant gradient vector $\nabla z = (5, 4)^\top$ points to the direction in which the level curves increase in value. The optimal point is represented by $x^* = (3, 1.5)^\top$ with the furthermost level curve being that associated with the value $z^* = 21$

exhaustively test and select the best (i.e., that with the largest objective function value). The issue, however, is that the number of candidates increases *exponentially* with the number of constraints and variables of the problem, which indicates this would quickly become computationally infeasible. As we will see, it turns out that this search idea can be made surprisingly efficient and is, in fact, the underlying framework of the *simplex method*. However, there are indeed artificially engineered worst-case settings where the method does need to consider every single vertex.

The simplex method exploits the above idea to *heuristically* search for solutions by selecting n constraints to be active from the m constraints available. Starting from an initial selection of constraints to be active, it selects one inactive constraint to activate and one to deactivate in a way that improvement in the objective function can be observed while feasibility is maintained. This process repeats until no improvement can be observed. When such is the case, the *geometry* of the problem guarantees *(global) optimality*. In the following chapters, we will concentrate on defining algebraic objects that we will use to develop the simplex method.

1.4 Exercises

Exercise 1.1: Introduction to JuMP

Use the Julia package JuMP.jl to implement the problems below and find the optimal solution. For the two-dimensional problems, use Plots.jl to illustrate the feasible region and the optimal solution.

(a)

$$\begin{aligned} \text{max. } & x_1 + 2x_2 + 5x_3 \\ \text{s.t.: } & x_1 - x_2 - 3x_3 \geq 5 \\ & x_1 + 3x_2 - 7x_3 \leq 10 \\ & x_1 \leq 10 \\ & x_1, x_2 \geq 0. \end{aligned}$$

(b)

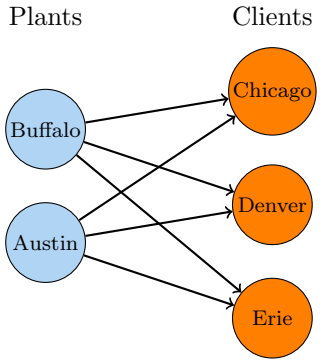
$$\begin{aligned} \text{max. } & 2x_1 + 4x_2 \\ \text{s.t.: } & x_1 + x_2 \leq 5 \\ & -x_1 + 3x_2 \leq 1 \\ & x_1 \leq 5 \\ & x_2 \leq 5 \\ & x_1, x_2 \geq 0. \end{aligned}$$

(c)

$$\begin{aligned} \text{min. } & -5x_1 + 10x_2 + x_3 + 2000x_4 \\ \text{s.t.: } & x_1 - x_2 \leq 1500 \\ & 4x_2 - x_3 \leq 5000x_4 \\ & x_1 + 3x_2 \geq 1000 \\ & x_1 \leq 10000 \\ & x_1, x_2 \in \mathbb{R}, x_3 \leq 0, x_4 \in \{0, 1\}. \end{aligned}$$

(d)

$$\begin{aligned} \text{max. } & 5x_1 + 3x_2 \\ \text{s.t.: } & x_1 + 5x_2 \leq 3 \\ & 3x_1 - x_2 \leq 5 \\ & x_1 \leq 2 \\ & x_2 \leq 30 \\ & x_1, x_2 \geq 0. \end{aligned}$$



	Clients			
Factory	Chicago	Denver	Eire	
Buffalo	4	9	8	25
Austin	10	7	9	600
Demands	15	12	13	-

Table 1.3: Problem data: unit transportation costs, demands and capacities

Exercise 1.2: Transportation problem [3]

Consider the following network, where the supplies from the Austin and Buffalo nodes need to meet the demands in Chicago, Denver, and Erie. The data for the problem is presented in table below.

Solve the transportation problem, finding the minimum cost transportation plan.

Exercise 1.3: Capacitated transportation

The Finnish company Next has a logistics problem in which used oils serve as raw material to form a class of renewable diesel. The supply chain team need to organise, to a minimal cost, the acquisition of two types of used oils (products p1 and p2) from three suppliers (supply nodes s1, s2, and s3) to feed the line of three of their used oils processing factories (demand nodes d1, d2, and d3). As the used oils are the byproduct of the supplier's core activities, the only requirement is that Next need to fetch the amount of oil and pay the transportation costs alone.

The oils have specific conditioning and handling requirements, so transportation costs vary between p1 and p2. Additionally, not all the routes (arcs) between suppliers and factories are available as some distances are not economically feasible. Table 1.4a shows the volume requirement of the two types of oil from each supply and demand node, and table 1.4b show the transportation costs for each oil type per L and the arc capacity. Arcs with "-" for costs are not available as transportation routes.

Find the optimal oil acquisition plan for Next, i.e., solve its transportation problem to the minimum cost.

node	p1	p2
s1 / d1	80 / 60	400 / 300
s2 / d2	200 / 100	1500 / 1000
s3 / d3	200 / 200	300 / 500

(a) Supply availability and demand per oil type [in L]

	d1 p1/p2 (cap)	d2 p1/p2 (cap)	d3 p1/p2 (cap)
s1	5/- (∞)	5/18 (300)	-/- (0)
s2	8/15 (300)	9/12 (700)	7/14 (600)
s3	-/- (0)	10/20 (∞)	8/- (∞)

(b) Arcs costs per oil type [in € per L] and arc capacities [in L]

Table 1.4: Supply chain data

Exercise 1.4: The farmer's problem [2]

Consider a farmer who produces wheat, corn, and sugar beets on his 500 acres of land. During the winter, the farmer wants to decide how much land to devote to each crop.

The farmer knows that at least 200 tons (T) of wheat and 240T of corn are needed for cattle feed. These amounts can be raised on the farm or bought from a wholesaler. Any production in excess of the feeding requirement would be sold. Over the last decade, mean selling prices have been $ 170 and $ 150 per ton of wheat and corn, respectively. The purchase prices are 40 % more than this due to the wholesaler's margin and transportation costs.

Another profitable crop is sugar beet, which he expects to sell at $36/T; however, the European Commission imposes a quota on sugar beet production. Any amount in excess of the quota can be sold only at $10/T. The farmer's quota for next year is 6000T.

Based on past experience, the farmer knows that the mean yield on his land is roughly 2.5T, 3T, and 20T per acre for wheat, corn, and sugar beets, respectively.

Based on the data, build up a model to help the farmer allocate the farming area to each crop and how much to sell/buy of wheat, corn, and sugar beets considering the following cases.

(a) The predictions are 100% accurate and the mean yields are the only realizations possible.

(b) There are three possible equiprobable scenarios (i.e, each one with a probability equal to $\frac{1}{3}$): a good, fair, and bad weather scenario. In the good weather, the yield is 20% better than the yield expected whereas in the bad weather scenario it is reduced 20% of the mean yield. In the regular weather scenario, the yield for each crop keeps the historical mean - 2.5T/acre, 3T/acre, and 20T/acre for wheat, corn, and sugar beets, respectively.

(c) What happens if we assume the same scenarios as item (b) but with probabilities 25%, 25%, and 50% for good, fair, and bad weather, respectively? How the production plan changes and why?

Exercise 1.5: Factory planning [4]

A factory makes seven products (PROD 1 to PROD 7) using the following machines: four grinders, two vertical drills, three horizontal drills, one borer and one planer. Each product yields a certain contribution to the profit (defined as $/unit selling price minus the cost of raw materials). These quantities (in $/unit) together with the unit production times (hours) required on each process are given in Table **??**. A dash indicates that a product does not require a process. There are also marketing demand limitations on each product each month. These are given in Table 1.6.

	PROD1	**PROD2**	**PROD3**	**PROD4**	**PROD5**	**PROD6**	**PROD7**
Profit	10	6	8	4	11	9	3
Grinding	1.5	2.1	–	–	0.9	0.6	1.5
Vert. drilling	0.3	0.6	–	0.9	–	1.8	–
Horiz. drilling	0.6	–	2.4	–	–	–	1.8
Boring	0.15	0.09	–	0.21	0.3	–	0.24
Planing	–	–	0.03	–	0.15	–	0.15

Table 1.5: Product yields

	PROD1	**PROD2**	**PROD3**	**PROD4**	**PROD5**	**PROD6**	**PROD7**
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	300	600	0	0	500	400	100
April	200	300	400	500	200	0	100
May	0	100	500	100	1000	300	0
June	500	500	100	300	1100	500	60

Table 1.6: Maximum demand

It is possible to store up to 100 of each product at a time at a cost of $0.5 per unit per month. There are no stocks at present, but it is desired to have a stock of 50 of each type of product at the end of June.

The factory works six days a week with two shifts of 8h each day. Assume that each month consists of only 24 working days. Also, there are no penalties for unmet demands. What is the factory's production plan (how much of which product to make and when) in order to maximise the total profit?

Chapter 2

Basics of Linear Algebra

2.1 Basics of linear problems

As we have seen in the previous chapter, the feasible region of a linear programming problem can be represented as

$$Ax \leq b, \tag{2.1}$$

where A is a $m \times n$ matrix, x is a n-dimensional column vector (or more compactly, $x \in \mathbb{R}^n$), and b is an m-dimensional column vector ($b \in \mathbb{R}^m$). Notice that $\leq$ is considered component-wise. Also, let $\mathbf{dim}(x)$ denote the dimension of vector x.

Before introducing the simplex method, let us first revisit a few key elements and operations that we will use in the process. The first of them is presented in Definition 2.1.

Definition 2.1 (Matrix inversion)**.** *Let A be a square $n \times n$ matrix. A^{-1} is the inverse matrix of A if it exists and $AA^{-1} = I$, where I is the $(n \times n)$ identity matrix.*

Matrix inversion is the "kingpin" of linear (and nonlinear) optimisation. As we will see later on, performing efficient matrix inversion operations (in reality, operations that are equivalent to matrix inversion but that can exploit the matrix structure to be made faster) is of utmost importance for developing a linear optimisation solver.

Another important concept is the notion of *linear independence*. We formally state when a collection of vectors is said to be linearly independent (or dependent) in Definition 2.2.

Definition 2.2 (Linearly independent vectors)**.** *The vectors $\{x_i\}_{i=1}^k \in \mathbb{R}^n$ are linearly dependent if there exist real numbers $\{a_i\}_{i=1}^k$ with $a_i \neq 0$ for at least one $i \in \{1, \ldots, k\}$ such that*

$$\sum_{i=1}^{k} a_i x_i = 0;$$

otherwise, $\{x_i\}_{i=1}^k$ are linearly independent.

In essence, for a collection of vectors to be linearly independent, it must be so that none of the vectors in the collection can be expressed as a linear combination (that is, multiplying the vectors by nonzero scalars and adding them) of the others. Analogously, they are said to be linearly dependent if one vector in the collection can be expressed as a linear combination of the others.

This is simpler to see in $\mathbb{R}^2$. Two vectors are linearly independent if one cannot obtain one by multiplying the other by a constant, which effectively means that they are not parallel. If the two vectors are not parallel, then one of them must have a component in a direction that the other

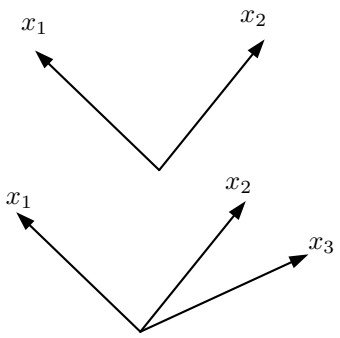


Figure 2.1: Linearly independent (top) and dependent (bottom) vectors in $\mathbb{R}^2$. Notice how, in the bottom picture, any of the vectors can be obtained by appropriate scaling and adding the other two

cannot achieve. The same idea can be generalised to any n-dimensional space. Also, that also implies why one can only have up to n independent vectors in $\mathbb{R}^n$. Figure 2.1 illustrates this idea.

Theorem 2.3 summarises results that we will utilise in the upcoming developments. These are classical results from linear algebra and are thus provided without proof.

Theorem 2.3 (Inverses, linear independence, and solving $Ax = b$)**.** *Let A be a $m \times m$ matrix. Then, the following statements are equivalent:*

1. *A is invertible*
2. *$A^\top$ is invertible*
3. *The determinant of A is nonzero*
4. *The rows of A are linearly independent*
5. *The columns of A are linearly independent*
6. *For every $b \in \mathbb{R}^m$, the linear system $Ax = b$ has a unique solution*
7. *There exists some $b \in \mathbb{R}^m$ such that $Ax = b$ has a unique solution.*

Notice that Theorem 2.3 establishes important relationships between the geometry of the matrix A (its rows and columns) and consequences it has to our ability to calculate its inverse A^{-1} and, consequently, solve the system $Ax = b$, to which the solution is obtained as $x = A^{-1}b$. Solving linear systems of equations will turn out to be the most important operation in the simplex method.

2.1.1 Subspaces and bases

Let us define some objects that we will frequently refer to. The first of them is the notion of a *subspace*. A subspace of $\mathbb{R}^n$ is a set comprising all linear combinations of its own elements. Specifically, if S is a subspace, then

$$S = \{ax + by : x, y \in S; a, b \in \mathbb{R}\}.$$

A related concept is the notion of a *span*. A span of a collection of vectors $\{x_i\}_{i=1}^k \in \mathbb{R}^n$ is the subspace of $\mathbb{R}^n$ formed by all linear combinations of such vectors, i.e.,

$$\mathbf{span}(x_1, \ldots, x_k) = \left\{ y = \sum_{i=1}^{k} a_i x_i : a_i \in \mathbb{R}, i \in \{1, \ldots, k\} \right\}.$$

Notice how the two concepts are related: the span of a collection of vectors forms subspace. Therefore, a subspace can be characterised by the collection of vectors whose span forms it. In other words, the span of a set of vectors is the subspace formed by all points we can represent by some linear combination of these vectors.

The missing part in this is the notion of a *basis*. A *basis* of the subspace $S \subseteq \mathbb{R}^n$ is a collection of vectors $\{x_i\}_{i=1}^k \in \mathbb{R}^n$ that are linearly independent such that $\mathbf{span}(x_1, \ldots, x_k) = S$.

Notice that a basis is a "minimal" set of vectors that form a subspace. You can think of it in light of the definition of linearly independent vectors (Definition 2.2); if a vector is linearly dependent to the others, it is not needed for characterising the subspace that the vectors span since it can be represented by a linear combination of the other vectors (and thus is in the subspace formed by the span of the other vectors).

The above leads us to some important realisations:

1. All bases of a given subspace S have the same dimension. Any extra vector would be linearly dependent to those vectors that span S. In that case, we say that the subspace has size (or dimension) k, the number of linearly independent vectors forming the basis of the subspace. We can overload the notation $\mathbf{dim}(S)$ to represent the dimension of the subspace S.

2. If the subspace $S \subset \mathbb{R}^n$ is formed by a basis of size $m < n$, we say that S is a proper subspace with $\mathbf{dim}(S) = m$, because it is not the whole $\mathbb{R}^n$ itself, but a space contained within $\mathbb{R}^n$. For example, two linearly independent vectors form (i.e., span) a hyperplane in $\mathbb{R}^3$; this hyperplane is a proper subspace since $\mathbf{dim}(S)m = 2 < 3 = n$.

3. If a proper subspace has dimension $m < n$, then it means that there are $n-m$ directions in $\mathbb{R}^n$ that are perpendicular to the subspace and to each other. That is, there are nonzero vectors a_i that are orthogonal to each other and to S. Or, equivalently, $a_i^\top x = 0$ for $i = n-m+1, ..., n$. Referring to the $\mathbb{R}^3$, if $m = 2$, then there is a third direction that is perpendicular to (or not in) S. Figure 2.2 can be used to illustrate this idea. Notice how one can find a vector, say x_3 that is perpendicular to S. This is because the whole space is $\mathbb{R}^3$, but S has dimension $m = 2$ (or $\mathbf{dim}(S) = 2$).

Theorem 2.4 builds upon the previous points to guarantee the existence of bases and propose a procedure to form them.

Theorem 2.4 (Forming bases from linearly independent vectors)**.** *Suppose that $S = \mathbf{span}(x_1, \ldots, x_k)$ has dimension $m \leq k$. Then*

1. *There exists a basis of S consisting of m of the vectors $x_1, \ldots, x_k$.*

2. *If $k' \leq m$ and $x_1, \ldots, x_{k'} \in S$ are linearly independent, we can form a basis for S by starting with $x_1, \ldots, x_{k'}$ and choosing $m - k'$ additional vectors from $x_1, \ldots, x_k$.*

Proof. Notice that, if every vector $x_{k'+1}, \ldots, x_k$ can be expressed as a linear combination of $x_1, \ldots x_{k'}$, then every vector in S is also a linear combination of $x_1, \ldots, x_{k'}$. Thus, $x_1, \ldots, x_{k'}$

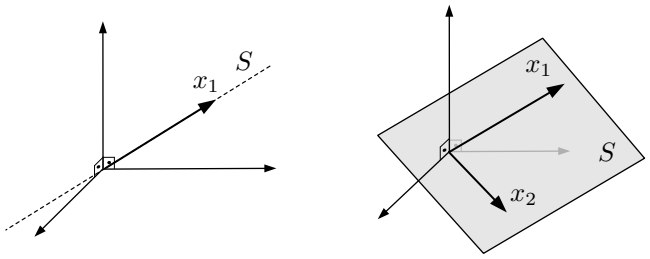


Figure 2.2: One- (left) and two-dimensional subspaces (right) in $\mathbb{R}^3$.

form a basis to S with $m = k'$. Otherwise, at least one of the vectors in $x_{k'+1}, \ldots, x_k$ is linearly independent from $x_1, \ldots, x_{k'}$. By picking one such vector, we now have $k' + 1$ of the vectors $x_{k'+1}, \ldots, x_k$ that are linearly independent. If we repeat this process $m - k'$ times, we end up with a basis for S. □

Our interest in subspaces and bases spans from (pun intended!) their usefulness in explaining how the simplex method works under a purely algebraic (as opposed to geometric) perspective. For now, we can use the opportunity to define some "famous" subspaces which will often appear in our derivations.

Let A be a $m \times n$ matrix as before. The *column space* of A consists of the subspace spanned by the n columns of A and has dimension m (recall that each column has as many components as the number of rows and is thus a m-dimensional vector). Likewise, the *row space* of A is the subspace in $\mathbb{R}^n$ spanned by the rows of A. Finally, the *null space* of A, often denoted as $\mathbf{null}(A) = \{x \in \mathbb{R}^n : Ax = 0\}$, consist of the vectors that are perpendicular to the row space of A.

One important notion related to those subspaces is their size. Both the row and the column space have the same size, which is the *rank* of A. If A is *full rank*, than it means that

$$\mathbf{rank}(A) = \min\{m, n\}.$$

Finally, the size of the null space of A is given $n - \mathbf{rank}(A)$, which is in line with Theorem 2.4.

2.1.2 Affine subspaces

A related concept is that of an *affine subspace*. Differently from linear subspaces (to which we have been referring to simply as subspaces), affine subspaces encode some form of translation, such as

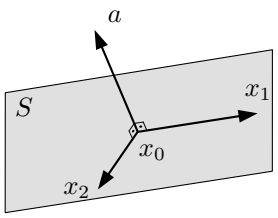
$$S = S_0 + x_0 = \{x + x_0 : x \in S_0\}.$$

Affine subspaces differ from linear subspaces because they do not contain the origin (recall that the definition of subspaces allows for a and b to be zero). Nevertheless, S has the *same dimension* as S_0.

Affine subspaces give a framework for representing linear programming problems algebraically. Specifically, let A be a $m \times n$ matrix with $m < n$ and b a m-dimensional vector. Then, let

$$S = \{x \in \mathbb{R}^n : Ax = b\}. \tag{2.2}$$

As we will see, the feasible set of any linear programming problem can be represented as an equality-constrained equivalent of the form of (2.2) by adding slack variables to the inequality



.

Figure 2.3: The affine subspace S generated by x_0 and $\mathbf{null}(a)$

constraints, meaning that we will always have that $m < n$. Now, assume that $x_0 \in \mathbb{R}^n$ is such that $Ax_0 = b$. Then, we have that

$$Ax = Ax_0 = b \Rightarrow A(x - x_0) = 0.$$

Thus, $x \in S$ if and only if the vector $(x - x_0)$ belongs to $\mathbf{null}(A)$, the nullspace of A. Notice that the feasible region S can be also defined as

$$S = \{x + x_0 : x \in \mathbf{null}(A)\},$$

being thus an affine subspace with dimension $n - m$, if A has m linearly independent rows (i.e., $\mathbf{rank}(A) = m$). This will have important implications in the way we can define multiple bases for S from the n vectors in the column space by choosing m to be removed and what this process means geometrically. Figure 2.3 illustrates this concept for a single-row matrix a. For multiple rows, one would see S as being represented by the intersection of multiple hyperplanes.

2.2 Convex polyhedral set

The feasible region of any linear programming problem is a convex polyhedral set, which we will simply refer to as a polyhedral set. That is because we are interested in polyhedral sets that are formed by an intersection of a finite number of half-spaces and can thus only be convex (as we will see in a moment), creating redundancy in our context but maybe some confusion overall.

2.2.1 Hyperplanes, half-spaces and polyhedral sets

Definition 2.5 formally states the structure that we refer to as polyhedral sets.

Definition 2.5 (Polyhedral set)**.** *A polyhedral set is a set that can be described as*

$$S = \{x \in \mathbb{R}^n : Ax \geq b\},$$

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

One important thing to notice is that polyhedral sets, as defined in Definition 2.5, as formed by the intersection multiple half-spaces. Specifically, let $\{a_i\}_{i=1}^m$ be the rows of A. Then, the set S can be described as

$$S = \left\{x \in \mathbb{R}^n : a_i^\top x \geq b_i, i = 1, \ldots, m\right\}, \tag{2.3}$$

which represents exactly the intersection of the half-spaces $a_i^\top x \geq b_i$. Furthermore, notice that the hyperplanes $a_i^\top x = b_i$, $\forall i \in \{1, \ldots, m\}$, are the boundaries of each hyperplane, and thus describe one of the facets of the polyhedral set. Figure 2.4 illustrates a hyperplane forming two half-spaces (also polyhedral sets) and how the intersection of five half-spaces form a (bounded) polyhedral set.

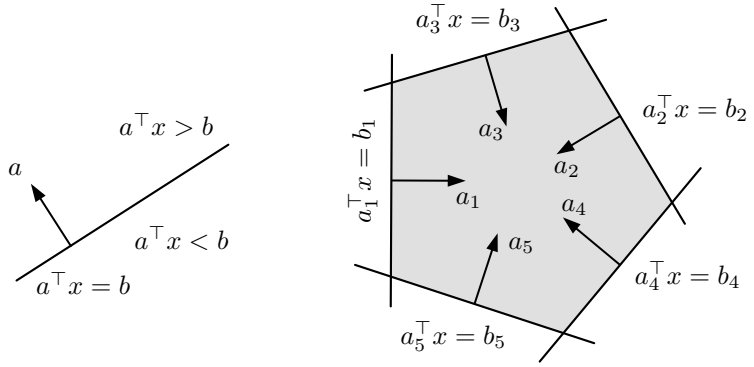


Figure 2.4: A hyperplane and its respective halfspaces (left) and the polyhedral set $\left\{x \in \mathbb{R}^2 : a_i^x \leq b_i, i = 1, \ldots, 5\right\}$ (right).

You might find authors referring to bounded polyhedral sets as polytopes. However, this is not used consistently across references, sometimes with switched meanings (for example, using polytope to refer to a set defined as in Definition 2.5 and using polyhedron to refer to a bounded version of S). In this text, we will only use the term polyhedral set to refer to sets defined as in Definition 2.5 and use the term bounded whenever applicable.

Also, it may be useful to formally define some elements in polyhedral sets. For that, let us consider a hyperplane $H = \left\{x \in \mathbb{R}n : a^\top x = b\right\}$. Now consider the set $F = H \cap S$. This set is known as a *face* of a polyhedral set. If the face F has dimension zero, then F is called a vertex. Analogously, if $\mathbf{dim}(F) = 1$, then F is called an edge. Finally, if $\mathbf{dim}(F) = dim(S) - 1$, then F is called a facet. Notice that in $\mathbb{R}^3$, facets and faces are the same, whenever the face is not an edge or a vertex.

2.2.2 Convexity of polyhedral sets

As will see in more detail in Part 2 of this book, convexity plays a crucial role in optimisation, being the "watershed" between easy and hard optimisation problems. One of the main reasons why we can solve challenging linear programming problems is due to the inherent convexity of polyhedral sets.

Let us first define the notion of convexity for sets, which is stated in Definition 2.6

Definition 2.6 (Convex set)**.** *A set $S \subseteq \mathbb{R}^n$ is convex if, for any $x_1, x_2 \in S$ and any $\lambda \in [0, 1]$, we have that $\overline{x} = \lambda x_1 + (1 - \lambda)x_2 \in S$.*

Definition 2.6 leads to a simple geometrical intuition: for a set to be convex, the line segment connecting any two points within the set must lie within the set. This is illustrated in Figure 2.5.

Associated with the notion of convex sets are two important elements we will refer to later, when we discuss linear problems that embed *integrality requirements*. The first is the notion of a convex combination, which is already contained in Definition 2.6, but can be generalised for an arbitrary number of points. The second consists of *convex hulls*, which are sets formed by combining the

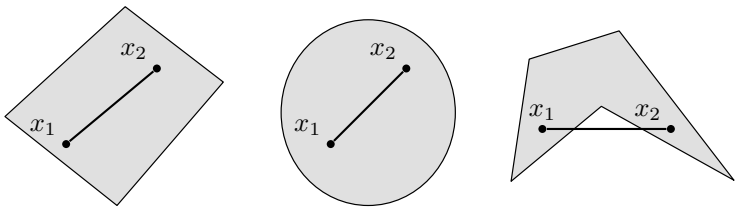


Figure 2.5: Two convex sets (left and middle) and one nonconvex set (right)

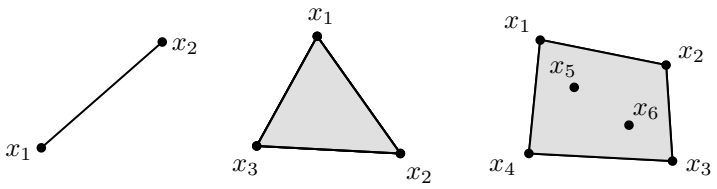


Figure 2.6: The convex hull of two points is the line segment connecting them (left); The convex hull of three (centre) and six (right) points in $\mathbb{R}^2$

convex combinations of all elements within a given set. As one might suspect, convex hulls are always convex sets, regardless whether the original set from which the points are drawn from is convex or not. These are formalised in Definition 2.7 and illustrated in Figure 2.6.

Definition 2.7 (Convex combinations and convex hulls)**.** *Let $x_1, \ldots, x_k \in \mathbb{R}^n$ and $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ such that $\lambda_i \geq 0$ for $i = 1, \ldots, k$ and $\sum_{i=1}^k \lambda_i = 1$. Then*

1. *$x = \sum_{i=1}^k \lambda_i x_i$ is a* convex combination *of $\{x_i\}_{i=1}^k \in \mathbb{R}^n$.*
2. *The* convex hull *of $\{x_i\}_{i=1}^k \in \mathbb{R}^n$, denoted* **conv**$(x_1, \ldots, x_k)$*, is the set of all convex combinations of $\{x_i\}_{i=1}^k \in \mathbb{R}^n$.*

We are now ready to state the result that guarantees the convexity of polyhedral sets of the form

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

Theorem 2.8 (Convexity of polyhedral sets)**.** *The following statements are true:*

1. *The intersection of convex sets is convex*
2. *Every polyhedral set is a convex set*
3. *A convex combination of a finite number of elements of a convex set also belongs to that set*
4. *The convex hull of a finite number of elements is a convex set.*

Proof. We provide the proof for each of the statements individually.

1. Let S_i, for $i \in I = \{1, \ldots, n\}$, be a collection of n convex sets and suppose that $x, y \in \bigcap_{i \in I} S_i$. Let $\lambda \in [0, 1]$. Since S_i are convex and $x, y \in S_i$ for all $i \in I$, $\lambda x + (1 - \lambda)y \in S_i$ for all $i \in I$ and, thus, $\lambda x + (1 - \lambda)y \in \bigcap_{i \in I} S_i$.

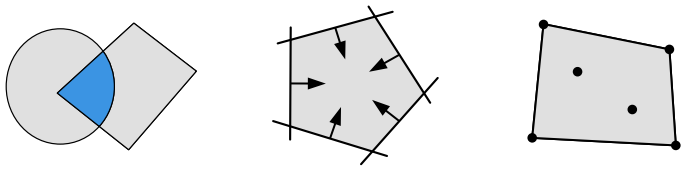


Figure 2.7: Illustration of statement 1 (left), 2 (centre), and 3 and 4 (right)

2. Let $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Let $x, y \in \mathbb{R}^n$, such that $a^\top x \geq b$ and $a^\top y \geq b$. Let $\lambda \in [0,1]$. Then $a^\top(\lambda x + (1-\lambda)y) \geq \lambda b + (1-\lambda)b = b$, showing that half-spaces are convex. The result follows from combining this with (1).

3. By induction. Let S be a convex set and assume that the convex combination of $x_1, \ldots, x_k \in S$ also belongs to S. Consider $k+1$ elements $x_1, \ldots, x_{k+1} \in S$ and $\lambda_1, \ldots, \lambda_{k+1}$ with $\lambda_i \in [0,1]$ for $i = 1, \ldots, k+1$ and $\sum_{i=1}^{k+1} \lambda_i = 1$ and $\lambda_{k+1} \neq 1$ (without loss of generality). Then

$$\sum_{i=1}^{k+1} \lambda_i x_i = \lambda_{k+1} x_{k+1} + (1-\lambda_{k+1}) \sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} x_i. \tag{2.4}$$

   Notice that $\sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} = 1$. Thus, using the induction hypothesis, $\sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} x_i \in S$. Considering that S is convex and using (2.4), we conclude that $\sum_{i=1}^{k+1} \lambda_{k+1} x_{k+1} \in S$, completing the induction.

4. Let $S = \mathbf{conv}(x_1, \ldots, x_k)$. Let $y = \sum_{i=1}^{k} \alpha_i x_i$ and $z = \sum_{i=1}^{k} \beta_i x_i$ be such that $y, z \in S$, $\alpha_i, \beta_i \geq 0$, and $\sum_{i=1}^{k} \alpha_i = \sum_{i=1}^{k} \beta_i = 1$. Let $\lambda \in [0,1]$. Then

$$\lambda y + (1-\lambda)z = \lambda \sum_{i=1}^{k} \alpha_i x_i + (1-\lambda) \sum_{i=1}^{k} \beta_i x_i = \sum_{i=1}^{k} (\lambda\alpha_i + (1-\lambda)\beta_i) x_i. \tag{2.5}$$

   Since $\sum_{i=1}^{k} \lambda\alpha_i + (1-\lambda)\beta_i = 1$ and $\lambda\alpha_i + (1-\lambda)\beta_i \geq 0$ for $i = 1, \ldots, k$, $\lambda y + (1-\lambda)z$ is a convex combination of $x_1, \ldots, x_k$ and, thus, $\lambda y + (1-\lambda)z \in S$, showing the convexity of S. □

Figure 2.7 illustrates some of the statements represented in the proof. For example, the intersection of the convex sets is always a convex set. One should notice however that the same does not apply to the union of convex sets. Notice that statement 2 proves that polyhedral sets as defined according to Definition 2.5 are convex. Finally the third figure on the right illustrates the convex hull of four points as a convex polyhedral set containing the lines connecting any two points within the set.

We will halt our discussion about convexity for now and return to it in deeper detail in Part 2. As it will become clearer then, the presence of convexity (which is a given in the context of linear programming, as we have just seen) is what allows us to conclude that the solutions returned by our optimisation algorithms are indeed optimal for the problem at hand.

2.3 Extreme points, vertices, and basic feasible solutions

Now we focus on the algebraic representation of the most relevant geometric elements in the optimisation of linear programming problems. As we have seen in the graphical example in the

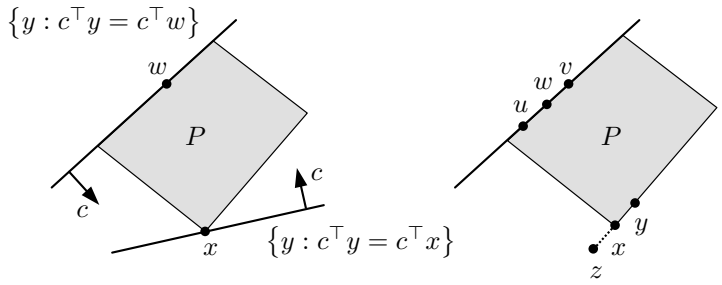


Figure 2.8: Representation of a vertex (left) and a extreme point (right)

previous chapter, the optimum of linear programming problems is generally located at the vertices of the feasible set. Furthermore, such vertices are formed by the intersection of n constraints (in a n-dimensional space, which comprises constraints that are active (or satisfied at the boundary of the half-space of said constraints).

First, let us formally define the notions of vertex and extreme point. Although in general these can refer to different objects, we will see that in the case of linear programming problems, if a point is a vertex, then it is an extreme point as well, being the converse also true.

Definition 2.9 (Vertex)**.** *Let P be a convex polyhedral set. The vector $x \in P$ is a vertex of P if there exists some c such that $c^\top x < c^\top y$ for all $y \in P$ with $y \neq x$.*

Definition 2.10 (Extreme points)**.** *Let P be a convex polyhedral set. The vector $x \in P$ is an extreme point of P if there are no two vectors $y, z \in P$ (different than x) such that $x = \lambda y + (1-\lambda)z$, for any $\lambda \in [0, 1]$.*

Figure 2.8 provides an illustration of the Definitions 2.9 and 2.10. Notice that the definition of a vertex involves an additional hyperplane that, once placed on a vertex point, strictly contains the whole polyhedral set in one of the half-spaces it defines, except for the vertex itself. On the other hand, the definition of an extreme point only relies on convex combinations of elements in the set itself.

Definition 2.9 also hints an important consequence for linear programming problems. As we seen from Theorem 2.8, P is convex, which guarantees that P is contained in the half-space $c^\top y > c^\top x$. This implies that $c^\top x \leq c^\top y, \forall y \in P$, which is precisely the condition that x must satisfy to be the minimum for the problem $\min._x \{c^\top x : x \in P\}$.

Now we focus on the description of active constraints from an algebraic standpoint. For that, let us first generalise our setting by considering all possible types of linear constraints. That is, let us consider the convex polyhedral set $P \subset \mathbb{R}^n$, formed by the set of inequalities and equalities:

$$a_i^\top x \geq b, i \in M_1,$$
$$a_i^\top x \leq b, i \in M_2,$$
$$a_i^\top x = b, i \in M_3.$$

Definition 2.11 (Active (or binding) constraints)**.** *If a vector $\overline{x}$ satisfies $a_i^\top \overline{x} = b_i$ for some $i \in M_1, M_2$, or M_3, we say that the corresponding constraints are active (or binding).*

Definition 2.11 formalises the notion of active constraints. This is illustrated in Figure 2.9, where the polyhedral set $P = \{x \in \mathbb{R}^3 : x_1 + x_2 + x_3 = 1, x_i \geq 0, i = 1, 2, 3\}$ is represented. Notice that,

while points A, B, C and D have 3 active constraints, E only has 2 active constraints ($x_2 = 0$ and $x_1 + x_2 + x_3 = 1$).

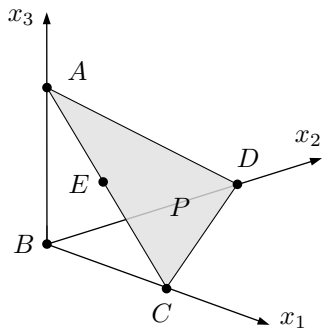


Figure 2.9: Representation of P in $\mathbb{R}^3$.

Theorem 2.12 sows a thread between having a collection of active constraints forming a vertex and being able to describe it as a basis of a subspace that is formed by the vectors a_i that form these constraints. This link is what will allow us to characterise vertices by their forming active constraints.

Theorem 2.12 (Properties of active constraints)**.** *Let $\overline{x} \in \mathbb{R}^n$ and $I = \left\{i \in M_1 \cup M_2 \cup M_3 \mid a_i^\top \overline{x} = b_i\right\}$. Then, the following are equivalent:*

1. *There exists n vectors in $\{a_i\}_{i \in I}$ that are linearly independent.*
2. *The $\mathbf{span}(\{a_i\}_{i \in I})$ spans $\mathbb{R}^n$. That is, every $x \in \mathbb{R}^n$ can be expressed as a linear combination of $\{a_i\}_{i \in I}$.*
3. *The system of equations $\left\{a_i^\top x = b_i\right\}_{i \in I}$ has a unique solution.*

Proof. Suppose that $\{a_i\}_{i \in I}$ spans $\mathbb{R}^n$, implying that the $\mathbf{span}(\{a_i\}_{i \in I})$ has dimension n. By Theorem 2.4 (part 1), n of these vectors form a basis for $\mathbb{R}^n$ and are, thus, linearly independent. Moreover, they must span $\mathbb{R}^n$ and therefore every $x \in \mathbb{R}^n$ can be expressed as a combination of $\{a_i\}_{i \in I}$. This connects (1) and (2).

Assume that the system of equations $\left\{a_i^\top x = b_i\right\}_{i \in I}$ has multiple solutions, say x_1 and x_2. Then, the nonzero vector $d = x_1 - x_2$ satisfies $a_i^\top d = 0$ for all $i \in I$. As d is orthogonal to every a_i, $i \in I$, d cannot be expressed as a combination of $\{a_i\}_{i \in I}$ and, thus, $\{a_i\}_{i \in I}$ do not span $\mathbb{R}^n$.

Conversely, if $\{a_i\}_{i \in I}$ do not span $\mathbb{R}^n$, choose $d \in \mathbb{R}^n$ that is orthogonal to $\mathbf{span}(\{a_i\}_{i \in I})$. If x satisfies $\left\{a_i^\top x = b_i\right\}_{i \in I}$, so does $\left\{a_i^\top (x + d) = b_i\right\}_{i \in I}$, thus yielding multiple solutions. This connects (2) and (3). □

Notice that Theorem 2.12 implies that there are (at least) n *active constraints (a_i)* that are *linearly independent* at $\overline{x}$. This is the reason why we will refer to $\overline{x}$, and any vertex-forming solution, as a *basic solution*, of which we will be interested in those that are feasible, i.e., that satisfy all constraints $i \in M_1 \cup M_2 \cup M_3$. Definition 2.13 provides a formal definition of these concepts.

Definition 2.13 (Basic feasible solution (BFS))**.** *Consider a convex polyhedral set $P \subset \mathbb{R}^n$ defined by linear equality and inequality constraints, and let $\overline{x} \in \mathbb{R}^n$.*

1. $\overline{x}$ *is a* basic solution *if*
   (a) *All equality constraints are active and,*
   (b) *Out of the constraints active at* $\overline{x}$*,* n *of them are linearly independent.*
2. *if* $\overline{x}$ *is a basic solution satisfying all constraints, we say* $\overline{x}$ *is a basic feasible solution.*

Figure 2.10 provides an illustration of the notion of basic solutions, and show how only a subset of the basic solutions are feasible. As one might infer, these will be the points of interest in out future developments, as these are the candidates for optimal solution.

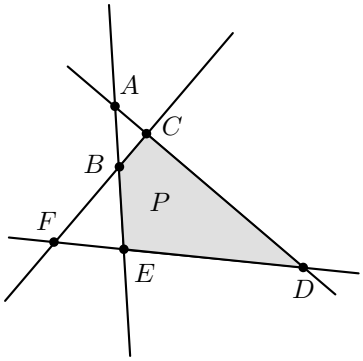


Figure 2.10: Points A to F are basic solutions; B,C,D, and E are BFS.

We finalise stating the main result of this chapter, which formally confirms the intuition we have developed so far. That is, for convex polyhedral sets, the notion of vertices and extreme points coincide, and these points can be represented as basic feasible solutions. This is precisely the link that allows for considering the feasible region of linear programming problems under a purely algebraic characterisation of the candidates for optimal solutions, those described uniquely by a subset of constraints of the problem that is assumed to be active.

Theorem 2.14 (BFS, extreme points and vertices)**.** *Let* $P \subset \mathbb{R}^n$ *be a convex polyhedral set and let* $\overline{x} \in P$*. Then, the following are equivalent*

$$\overline{x} \text{ is a vertex} \iff \overline{x} \text{ is an extreme point} \iff \overline{x} \text{ is a BFS.}$$

Proof. Let $P = \{x \in \mathbb{R}^n : a_i^\top x \geq b_i, i \in M_1, a_i^\top x = b_i, i \in M_2\}$, and $I = \left\{i \in M_1 \cup M_2 \mid a_i^\top x = b_i\right\}$.

1. (Vertex $\Rightarrow$ Extreme point) Suppose $\overline{x}$ is a vertex. Then, there exists some $c \in \mathbb{R}^n$ such that $c^\top \overline{x} < c^\top x$, for every $x \in P$ with $x \neq \overline{x}$ (cf. Definition 2.9). Take $y, z \in P$ with $y, z \neq \overline{x}$. Thus $c^\top \overline{x} < c^\top y$ and $c^\top \overline{x} < c^\top z$. For $\lambda \in [0,1]$, $c^\top \overline{x} < c^\top(\lambda y + (1-\lambda)z)$ implying that $\overline{x} \neq \lambda y + (1-\lambda)z$, and is thus an extreme point (cf. Definition 2.10).

2. (Extreme point $\Rightarrow$ BFS) suppose $\overline{x} \in P$ is not a BFS. Then, there are no n linearly independent vectors within $\{a_i\}_{i \in I}$. Thus $\{a_i\}_{i \in I}$ lie in a proper subspace of $\mathbb{R}^n$. Let the nonzero vector $d \in \mathbb{R}^n$ be such that $a_i^\top d = 0$, for all $i \in I$.

   Let $\epsilon > 0$, $y = \overline{x} + \epsilon d$, and $z = \overline{x} - \epsilon d$. Notice that $a_i^\top y = a_i^\top z = b_i$, for all $i \in I$. Moreover, for $i \neq I$, $a_i^\top x > b_i$ and, provided that ϵ is sufficiently small (such that $\epsilon |a_i^\top d| < a_i^\top \overline{x} - b_i$), we have that $a_i^\top x \geq b_i$ for all $i \in I$. Thus $y \in P$, and by a similar argument, $z \in P$. Now, by noticing that $\overline{x} = \frac{1}{2}y + \frac{1}{2}z$, we see that $\overline{x}$ is not an extreme point.

3. (BFS $\Rightarrow$ Vertex) Let $\overline{x}$ be a BFS. Define $c = \sum_{i \in I} a_i$. Then

$$c^\top \overline{x} = \sum_{i \in I} a_i^\top \overline{x} = \sum_{i \in I} b_i.$$

Also, for any $x \in P$, we have that

$$c^\top x = \sum_{i \in I} a_i^\top x \geq \sum_{i \in I} b_i,$$

since $a_i^\top x \geq b_i$ for $i \in M_1 \cup M_2$. Thus, for any $x \in P$, $c^\top \overline{x} \leq c^\top x$, making $\overline{x}$ a vertex (cf. Definition 2.9). $\square$

Some interesting insights emerge from the proof of Theorem 2.14, upon which we will build our next developments. Once the relationship between being a vertex/ extreme point and a BFS is made, it means that $\overline{x}$ can be recovered as the unique solution of a system of linear equations, these equations being the active constraints at that vertex. This means that the list of all optimal solution candidate points can be obtained by simply looking at all possible combinations of n active constraints, discarding those that are infeasible. This means that the number of candidates for optimal solution is *finite* and can be bounded by $\binom{m}{n}$, where $m = |M_1 \cup M_2|$.

2.4 Exercises

Exercise 2.1: Polyhedral sets [1]

Which of the following sets are polyhedral sets?

a) $\{(x, y) \in \mathbb{R}^2 \mid x\cos\theta + y\sin\theta \leq 1, \theta \in [0, \pi/2], x \geq 0, y \geq 0\}$

b) $\{x \in \mathbb{R} \mid x^2 - 8x + 15 \leq 0\}$

c) The empty set ($\emptyset$).

Exercise 2.2: Convexity of polyhedral sets

Prove the following theorem.

Theorem (Convexity of polyhedral sets)**.** The following convexity properties about convex sets can be said:

1. *The intersection of convex sets is convex*
2. *Every polyhedral set is a convex set*
3. *A convex combination of a finite number of elements of a convex set also belongs to that set*
4. *The convex hull of a finite number of elements is a convex set.*

Note: the proof of the theorem is proved in the notes. Use this as an opportunity to revisit the proof carefully, and try to take as many steps without consulting the text as you can. This is a great exercise to help you internalise the proof and its importance in the context of this book. I strongly advise against blindly memorising it, as I suspect you will never (in my courses, at least) be requested to recite the proof literally.

Exercise 2.3: Properties of active constraints

Let us consider the convex polyhedral set $P \subset \mathbb{R}^n$, formed by the set of equalities and inequalities:

$$\begin{aligned} a_i^\top x &\geq b, i \in M_1, \\ a_i^\top x &\leq b, i \in M_2, \\ a_i^\top x &= b, i \in M_3. \end{aligned}$$

Prove the following result.

Theorem (Properties of active constraints)**.** *Let $\overline{x} \in \mathbb{R}^n$ and $I = \left\{i \in M_1 \cup M_2 \cup M_3 \mid a_i^\top \overline{x} = b_i\right\}$. Then, the following are equivalent:*

1. *There exists n vectors in $\{a_i\}_{i\in I}$ that are linearly independent.*
2. *The* **span**$(\{a_i\}_{i\in I})$ *spans $\mathbb{R}^n$. That is, every $x \in \mathbb{R}^n$ can be expressed as a combination of $\{a_i\}_{i\in I}$.*
3. *The system of equations $\left\{a_i^\top x = b_i\right\}_{i\in I}$ has a unique solution.*

Note: see Exercise 2.2.

Exercise 2.4: Vertex, extreme points, and BFSs

Prove the following result.

Theorem (BFS, extreme points and vertices)**.** *Let $P \subset \mathbb{R}^n$ be a convex polyhedral set and let $\overline{x} \in P$. Then, the following are equivalent*

1. *$\overline{x}$ is a vertex;*
2. *$\overline{x}$ is a extreme point;*
3. *$\overline{x}$ is a BFS;*

Note: see Exercise 2.2.

Exercise 2.5: Binding constraints

Given the linear program defined by the system of inequalities below,

$$
\begin{aligned}
\text{max. } & 2x_1 + x_2 \\
\text{s.t.: } & 2x_1 + 2x_2 \leq 9 \\
& 2x_1 - x_2 \leq 3 \\
& x_1 - x_2 \leq 1 \\
& x_1 \leq 2.5 \\
& x_2 \leq 4 \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

Assess the following points relative to the polyhedron defined in $\mathbb{R}^2$ by this system and classify them as in (i) belonging to which active constraint(s), (ii) being a non-feasible/basic/basic feasible solution, and (iii) being an extreme point, vertex, or outside the polyhedron. Use Theorem 2.12 and Theorem 2.14 to check if your classification is correct.

a) $(1.5, 0)$

b) $(1, 0)$

c) $(2, 1)$

d) $(1.5, 3)$

Chapter 3

Basis, Extreme Points and Optimality in Linear Programming

3.1 Polyhedral sets in standard form

In the context of linear programming problems, we will often consider problems written in the so-called *standard form*. The standard form can be understood as posing the linear programming problem as an underdetermined system of equations (that is, with fewer equations than variables). Then, we will work on selecting a subset of the variables to be set to zero so that the number of remaining variables is the same as that of equations, making the system solvable.

A key point in this chapter will be devising how we relate this process of selecting variables with that of selecting a subset of active constraints (forming a vertex, as we seen in the previous chapter) that will eventually lead to an optimal solution.

3.1.1 The standard form of linear programming problems

First, let us formally define the notion of a standard-form polyhedral set. Let A is a $m \times n$ matrix and $b \in \mathbb{R}^m$. The *standard form* polyhedral set P is given by

$$P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}.$$

We assume that the m equality constraints are linearly independent, i.e., A is full (row) rank ($m \leq n$). We know that a basic solution can be obtained from a collection of n active constraints since the problem is defined in $\mathbb{R}^n$.

One important point is that *any* linear programming problem can be represented in the standard form. This is achieved utilising nonnegative *slack variables*. Thus, a feasibility set that is, say, originally represented as

$$P = \{x \in \mathbb{R}^n : A_1 x \leq b_1, A_2 x \geq b_2, x \geq 0\}$$

can be equivalently represented as a standard-form polyhedral set. For that, it must be modified to consider slack variables $s_1 \geq 0$ and $s_2 \geq 0$ such that

$$P = \left\{(x, s_1, s_2) \in \mathbb{R}^{(n+|b_1|+|b_2|)} : A_1 x + s_1 = b_1, A_2 x - s_2 = b_2, (x, s_1, s_2) \geq 0\right\},$$

where $|u|$ represents the cardinality of the vector u. Another transformation that may be required consists of imposing the condition $x \geq 0$. Let us assume that a polyhedral set P was such that (notice the absent nonnegativity condition)

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

It is a requirement for standard-form linear programs to have all variables to be assumed nonnegative. To achieve that in this case, we can simply include two auxiliary variables, say x^+ and x^-, with the same dimension as x, and reformulate P as

$$P = \left\{x^+, x^- \in \mathbb{R}^n : A(x^+ - x^-) = b, x^+, x^- \geq 0\right\}.$$

These transformations, as we will see, will be required for employing the simplex method to solve linear programming problems with inequality constraints and, inevitably, will always render standard form linear programming problems with more variables than constraints, or $m < n$.

The standard-form polyhedral set P always has, by definition, m active constraints because of its equality constraints. To reach the total of n active constraints, $n-m$ of the remaining constraints $x_i \geq 0$, $i = 1, \ldots, n$, must be activated, and this is achieved by selecting $n-m$ of those to be set as $x_i = 0$. These n active constraints (the original m plus the $n-m$ variables set to zero) form a basic solution, as we seen in the last chapter. If it happens that the m equalities can still be satisfied while the constraints $x_i \geq 0$, $i = 1, \ldots, n$, are satisfied, then we have a basic feasible solution (BFS). Theorem 3.1 summarises this process, guaranteeing that the setting of $n - m$ variables to zero will render a basic solution.

Theorem 3.1 (Linear independence and basic solutions)**.** *Consider the constraints $Ax = b$ and $x \geq 0$, and assume that A has m linearly independent (LI) rows $M = \{1, \ldots, m\}$. A vector $\overline{x} \in \mathbb{R}^n$ is a basic solution if and only if we have that $A\overline{x} = b$ and there exists indices $B(1), \ldots, B(m)$ such that*

(1) The columns $A_{B(1)}, \ldots, A_{B(m)}$ of A are LI

(2) If $j \neq B(1), \ldots, B(m)$, then $\overline{x}_j = 0$.

Proof. Assume that (1) and (2) are satisfied. Then the active constraints $\overline{x}_j = 0$ for $j \notin \{B(1), \ldots, B(m)\}$ and $Ax = b$ imply that

$$\sum_{i=1}^{m} A_{B(i)}\overline{x}_{B(i)} = \sum_{j=1}^{n} A_j\overline{x}_j = A\overline{x} = b.$$

Since the columns $\left\{A_{B(i)}\right\}_{i\in M}$ are LI, $\left\{\overline{x}_{B(i)}\right\}_{i\in M}$ are uniquely determined and thus $A\overline{x} = b$ has a unique solution, implying that $\overline{x}$ is a basic solution (cf. Theorem 2.14).

Conversely, assume that $\overline{x}$ is a basic solution. Let $\overline{x}_{B(1)}, \ldots, \overline{x}_{B(k)}$ be the nonzero components of $\overline{x}$. Thus, the system

$$\sum_{i=1}^{n} A_i\overline{x}_i = b \text{ and } \{\overline{x}_i = 0\}_{i\notin\{B(1),\ldots,B(k)\}}$$

has a unique solution, and so does $\sum_{i=1}^{k} A_{B(i)}\overline{x}_{B(i)} = b$, implying that the columns $A_{B(1)}, \ldots, A_{B(k)}$ are LI. Otherwise, there would be scalars $\lambda_1, \ldots, \lambda_k$, not all zeros, for which $\sum_{i=1}^{k} A_{B(i)}\lambda_i = 0$; this would imply that $\sum_{i=1}^{k} A_{B(i)}(\overline{x}_{B(i)} + \lambda_i) = b$, contradicting the uniqueness of $\overline{x}$. Since $A_{B(1)}, \ldots, A_{B(k)}$ are LI, $k \leq m$. Also, since A has m LI rows, it must have m LI columns spanning $\mathbb{R}^m$. Using Theorem 2.4, we can obtain $m - k$ additional columns $A_{B(k+1)}, \ldots, A_{B(m)}$ so that $A_{B(1)}, \ldots, A_{B(m)}$ are LI.

Finally, since $k \leq m$, $\{\overline{x}_j = 0\}_{i\notin\{B(1),\ldots,B(m)\}} \subset \{\overline{x}_j = 0\}_{i\notin\{B(1),\ldots,B(k)\}}$, satisfying (1) and (2). □

This proof highlights an important aspect in the process of generating basic solutions. Notice that once we set $n - m$ variables to be zero, the system of equations forming P becomes uniquely determined, i.e.,

$$\sum_{i=1}^{m} A_{B(i)}\overline{x}_{B(i)} = \sum_{j=1}^{n} A_j\overline{x}_j = A\overline{x} = b.$$

3.1.2 Forming bases for standard-form linear programming problems

Theorem 3.1 provides us with a way to develop a simple procedure to generate all basic solutions of a linear programming problem in standard form.

1. Choose m LI columns $A_{B(1)}, \ldots, A_{B(m)}$;
2. Let $x_j = 0$ for all $j \notin \{B(1), \ldots, B(m)\}$;
3. Solve the system $Ax = b$ to obtain $x_{B(1)}, \ldots, x_{B(m)}$.

You might have noticed that in the proof of Theorem 3.1, the focus shifted to the columns of A rather than its rows. The reason for that is because, when we think of solving the system $Ax = b$, what we are truly doing is finding a vector x representing the linear combination of the columns of A that yield the vector b. This creates an association between the columns of A and the components of x (i.e., the variables). Notice, however that the columns of A are not *the* variables per se, as they have dimension m (while $x \in \mathbb{R}^n$).

One important interpretation for Theorem 3.1 is that we will form bases for the column space of A by choosing m components to be nonzero in a vector of dimension n. Since $m < n$ by definition, A can only have rank $\mathbf{rank}(A) = \min m, n = m$, which happens to be the size of both the row space of A (as the rows are assumed LI). This, in turn, means that both the column and the row spaces have dimension m. Thus, these bases are bases for the column space of A. Finally, finding the vector x is the same as finding how the vector b can be expressed as a linear combination of that basis. Notice that this is always possible when the basis spans $\mathbb{R}^m$ (as we have m LI column vectors) and $b \in \mathbb{R}^m$.

You will notice that from here onwards, we will implicitly refer to the columns of A as variables (although we actually mean the weight associated with that column, represented by the respective component in the variable x). Then, when we say that we are setting some $(n-m)$ of the variables to be zero, it means that we are ignoring the respective columns of A (the mapping between variables and columns being their indices: x_1 referring to the first column, x_2 to the second, and so forth), while using the remainder to form a (unique) combination that yields the vector b, being the weights of this combination precisely the solution x, which in turn represent the coordinates in $\mathbb{R}^n$ of the vertex formed by the n (m equality constraints plus $n - m$ variables set to zero) active constraints.

As we will see, this procedure will be at the core of the simplex method. Since we will often refer to elements associated with this procedure, it will be useful to define some nomenclature.

We say that $B = \left\{A_{B(i)}\right\}_{i \in I_B}$ is a *basis* (or, perhaps more precisely, a basic matrix) with basic indices $I_B = \{B(1), \ldots, B(m)\}$. Consequently, we say that the variables x_j, for x_j, for $j \in I_B$, are *basic variables*. Somewhat analogously, we say that the variables chosen to be set to zero are the *nonbasic variables* x_j, for $j \in I_N$, where $I_N = J \setminus I_B$, with $J = \{1, \ldots, n\}$ being the indices of all variables (and all columns of A).

Notice that the basic matrix B is invertible, since its columns are LI (c.f. Theorem 2.4). For $x_B = (x_{B(1)}, \ldots, x_{B(m)})$, the *unique solution* for $Bx_B = b$ is

$$x_B = B^{-1}b, \text{ where } B = \begin{bmatrix} | & | & | \\ A_{B(1)} & \ldots & A_{B(m)} \\ | & | & | \end{bmatrix} \text{ and } x_B = \begin{bmatrix} x_{B(1)} \\ \vdots \\ x_{B(m)} \end{bmatrix}.$$

Let us consider the following numerical example. If we consider the following set P

$$P = \left\{ x \in \mathbb{R}^7 : \begin{array}{r} x_1 + x_2 + 2x_3 \leq 8 \\ x_2 + 6x_3 \leq 12 \\ x_1 \leq 4 \\ x_2 \leq 6 \\ x_1, x_2, x_3 \geq 0 \end{array} \right\}, \tag{3.1}$$

which can be written in the standard by adding slack variables $\{x_i\}_{i \in \{4,\ldots,7\}}$, yielding

$$P = \left\{ x \in \mathbb{R}^7 : \begin{array}{r} x_1 + x_2 + 2x_3 + x_4 = 8 \\ x_2 + 6x_3 + x_5 = 12 \\ x_1 + x_6 = 4 \\ x_2 + x_7 = 6 \\ x_1, \ldots, x_7 \geq 0 \end{array} \right\}. \tag{3.2}$$

The system $Ax = b$ can be represented as

$$\begin{bmatrix} 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 8 \\ 12 \\ 4 \\ 6 \end{bmatrix}.$$

Following our notation, we have that $m = 4$ and $n = 7$. The rows of A are LI, meaning that $\mathbf{rank}(A) = 4$[1]. We can make arbitrary selections of $n - m = 3$ variables to be set to zero (i.e., nonbasic) and calculate the value of the remaining (basic) variables. For example:

- Let $I_B = \{4, 5, 6, 7\}$; in that case $x_B = (8, 12, 4, 6)$ and $x = (0, 0, 0, 8, 12, 4, 6)$, which is a basic feasible solution (BFS), as $x \geq 0$.
- For $I_B = \{3, 5, 6, 7\}$, $x_B = (4, -12, 4, 6)$ and $x = (0, 0, 4, 0, -12, 4, 6)$, which is basic but not feasible, since $x_5 < 0$.

3.1.3 Adjacent basic solutions

Now that we know how a solution can be recovered, the next important concept that we need to define is how we, from one basic solution, move to an *adjacent* solution. This will be the mechanism that the simplex will utilise to move from one solution to the next in the search for the optimal solution.

Let us start formally defining the notion of an adjacent basic solution.

[1] You can see for yourself using Gaussian elimination or row reduction. Tip: do the elimination on the transpose $A^\top$ instead, recalling that $\mathbf{rank}(A) = \mathbf{rank}(A^\top)$.

Definition 3.2 (Adjacent basic solutions)**.** *Two basic solutions are adjacent if they share $n-1$ LI active constraints. Alternatively, two bases B_1 and B_2 are adjacent if all but one of their columns are the same.*

For example, consider the set polyhedral set P defined in (3.2). Our first BFS was defined by making $x_1 = x_2 = x_3 = 0$ (nonbasic index set $I_N = \{1, 2, 3\}$). Thus, our basis was $I_B = \{4, 5, 6, 7\}$. An adjacent basis was then formed, by replacing the basic variable x_4 with the nonbasic variable x_3, rendering the new (not feasible) basis $I_B = \{3, 5, 6, 7\}$.

Notice that the process of moving between adjacent basis has a simple geometrical interpretation. Since adjacent bases share all but one basic element, this means that the two must be connected by a line segment (in the case of the example, it would be the segment between $(0, 8)$ and $(4, 0)$, projected onto $(x_3, x_4) \in \mathbb{R}^2$, or equivalently, the line between $(0, 0, 0, 8, 12, 4, 6)$ and $(0, 0, 4, 0, -12, 4, 6)$ (Notice how the coordinate x_6 also changed values; this is necessary, so the movement is made along the edge of the polyhedral set. This will become clearer when we analyse the simplex method in further detail in Chapter 4.

3.1.4 Redundancy and degeneracy

An important underlying assumption in Theorem 3.1 is that the matrix A in the definition of the polyhedral set P is full (row) rank, that is, there are m linearly independent rows and thus m independent columns. Theorem 3.3 shows that this assumption can actually be made without loss of generality.

Theorem 3.3 (Redundant constraints)**.** *Let $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$, where A is $m \times n$ matrix with rows $\{a_i\}_{i \in M}$ and $M = \{1, \ldots, m\}$. Suppose that $\mathbf{rank}(A) = k < m$ and that the rows $a_{i_1}, \ldots, a_{i_k}$ are LI. Then P is the same set as $Q = \left\{x \in \mathbb{R}^n : a_{i_1}^\top x = b_{i_1}, \ldots, a_{i_k}^\top x = b_{i_k}, x \geq 0\right\}$.*

Proof. Assume, without loss of generality, that $i_1 = 1$ and $i_k = k$. Clearly, $P \subset Q$, since a solution satisfying the constraints forming P also satisfies those forming Q.

As $\mathbf{rank}(A) = k$, the rows $a_{i_1}, \ldots, a_{i_k}$ form a basis in the row space of A and any row a_i, $i \in M$, can be expressed as $a_i^\top = \sum_{j=1}^k \lambda_{ij} a_j^\top$ for $\lambda_{ij} \in \mathbb{R}$.

For $y \in Q$ and $i \in M$, we have $a_i^\top y = \sum_{j=1}^k \lambda_{ij} a_j^\top y = \sum_{j=1}^k \lambda_{ij} b_j = b_i$, which implies that $y \in P$ and that $Q \subset P$. Consequently, $P = Q$. □

Theorem 3.3 implies that any linear programming problem in standard form can be reduced to an equivalent problem with linearly independent constraints. It turns out that, in practice, most professional-grade solvers (i.e., software that implements solution methods and can be used to find optimal solutions to mathematical programming models) have *preprocessing* routines to remove redundant constraints. This means that the problem is automatically treated to become smaller by not incorporating unnecessary constraints.

Degeneracy is somewhat related to the notion of redundant constraints. We say that a given vertex is a *degenerate* basic solution if it is formed by the intersection of more than n active constraints (in $\mathbb{R}^n$). Effectively, this means that more than $n - m$ variables (i.e, some of the basic variables) are set to zero, which is the main way to identify a degenerate BFS. Figure 3.1 illustrates a case in which degeneracy is present.

Notice that, while in the figure on the left, the constraint causing degeneracy is redundant, that is not the case on the figure on the righthand side. That is, redundant constraints may cause degeneracy, but not all constraints causing degeneracy are redundant.

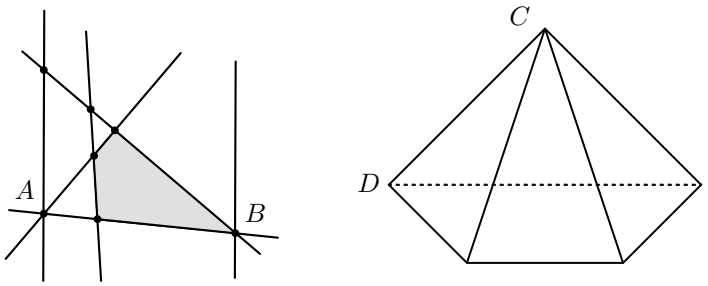


Figure 3.1: A is a degenerate basic solution, B and C are degenerate BFS, and D is a BFS.

In practice, degeneracy might cause issues related to the way we identify vertices. Because more than n active constraints form the vertex, and yet, we identify vertices by groups of n constraints to be active, it means that we might be have a collection of adjacent bases that, in fact, are representing the same vertex in space, meaning that we might be "stuck" for a while in the same position while scanning through adjacent bases. The numerical example below illustrates this phenomenon.

Let us consider again the same example as before.

$$\begin{bmatrix} 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 8 \\ 12 \\ 4 \\ 6 \end{bmatrix}$$

Observe the following,

- let $I_B = \{1, 2, 3, 7\}$; this implies that $x = (4, 0, 2, 0, 0, 0, 6)$. There are 4 zeros (instead of $n - m = 3$) in x, which indicates degeneracy.
- Now, let $= I_B = \{1, 3, 4, 7\}$. This also implies that $x = (4, 0, 2, 0, 0, 0, 6)$. The two bases are adjacent yet represent the same point in $\mathbb{R}^7$.

As we will see, there are mechanisms that prevent the simplex method from being stuck on such vertices forever, an issue that is referred to as *cycling*. One final point to observe about degeneracy is that it can be caused by the chosen representation of the problem. For example, consider the two equivalent sets:

$$P_1 = \{(x_1, x_2, x_3) : x_1 - x_2 = 0, x_1 + x_2 + 2x_3 = 2, x_1, x_3 \geq 0\} \text{ and } P_2 = P_1 \cap \{x_2 \geq 0\}.$$

The polyhedral set P_2 is equivalent to P_1 since $x_2 \geq 0$ is a redundant constraint. In that case, one can see that, while the point $(0, 0, 1)$ is not degenerate in P_1, it is in P_2, which illustrates the (weak but existent) relationship between redundancy and degeneracy. This is illustrated in Figure 3.2.

3.2 Optimality of extreme points

Now that we have discussed how to algebraic represent extreme points and have seen a simple mechanism to iterate among their adjacent neighbours, the final element missing for us to be able

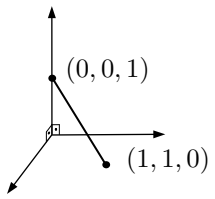


Figure 3.2: $(0, 0, 1)$ is degenerate if you add the constraint $x_2 \geq 0$.

to devise an optimisation method is to define the optimality conditions we wish to satisfy. In other words, we must define the conditions that, once satisfied, mean that we can stop the algorithm and declare the current solution optimal.

3.2.1 The existence of extreme points

First, let us define the condition that guarantees the existence of extreme points in a polyhedral set. Otherwise, there is no hope of finding an optimal solution.

Definition 3.4 (Existence of extreme point)**.** *A polyhedral set $P \subset \mathbb{R}^n$ contains a line if $P \neq \emptyset$ and there exists a nonzero vector $d \in \mathbb{R}^n$ such that $x + \lambda d \in P$ for all $\lambda \in \mathbb{R}$.*

Figure 3.3 illustrates the notion of containing a line and the existence of extreme points. Notice that if a set would have any "corner", then that would imply that a line is contained between the edges that form that corner and, therefore, that corner would be an extreme point.

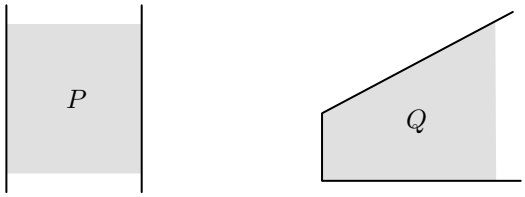


Figure 3.3: P contains a line (left) and Q does not contain a line (right)

We are now ready to pose the result that utilises Definition 3.4 to provide the conditions for the existence of extreme points.

Theorem 3.5 (Existence of extreme points)**.** *Let $P = \left\{x \in \mathbb{R}^n : a_i^\top x \geq b_i, i = 1, \ldots, m\right\} \neq \emptyset$ be a polyhedral set. Then the following are equivalent:*

1. *P has at least one extreme point;*
2. *P does not contain a line;*
3. *There exists n LI vectors within $\{a_i\}_{i=1}^m$.*

It turns out that linear programming problems in the standard form do not contain a line, meaning that they will always provide at least one extreme point (or a basic feasible solution). More generally, bounded polyhedral sets do not contain a line, and neither does the positive orthant.

We are now to state the result that proves the intuition we had when analysing the plots in Chapter 1, which states that if a polyhedral set has at least one extreme point and at least one optimal solution, then there must be an optimal solution that is an extreme point.

Theorem 3.6 (Optimality of extreme points)**.** *Let $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ be a polyhedral set and $c \in \mathbb{R}^n$. Consider the problem*

$$z = min. \left\{c^\top x : x \in P\right\}.$$

Suppose that P has at least one extreme point and that there exists an optimal solution. Then, there exists an optimal solution that is an extreme point of P.

Proof. Let $Q = \left\{x \in \mathbb{R}^n : Ax \geq b, c^\top x = z\right\}$ be the (nonempty) polyhedral set of all optimal solutions. Since $Q \subset P$ and P contains no line (cf. Theorem 3.5), Q contains no line either, and thus has an extreme point.

Let $\overline{x}$ be an extreme point of Q. By contradiction, assume that $\overline{x}$ is not an extreme point of P. Then, there exist $y \neq \overline{x}$, $w \neq \overline{x}$, and $\lambda \in [0, 1]$ such that $\overline{x} = \lambda y + (1 - \lambda)w$. Then, $c^\top \overline{x} = \lambda(c^\top y) + (1 - \lambda)c^\top w$. As $c^\top \overline{x} = z$ is optimal, we have that $z \leq c^\top y$ and $z \leq c^\top w$, and thus $z = c^\top y = c^\top w$.

Thus, $w \in Q$ and $y \in Q$, contradicting that $\overline{x}$ is an extreme point. Thus, $\overline{x}$ must be an extreme point and, since we established that $\overline{x} \in Q$, it is also optimal. □

Theorem 3.6 is posed in a somewhat general way, which might be a source for confusion. First, recall that in the example in Chapter 1, we considered the possibility of the objective function level curve associated with the optimal value to be parallel to one of the edges of the feasible region, meaning that instead of a single optimal solution (a vertex), we would observe a line segment containing an infinite number of optimal solutions, of which exactly two would be extreme points.

In a more general case (with $n > 2$) it might be so that a whole facet of optimal solutions is obtained. That is precisely the polyhedral set of all optimal solutions Q in the proof. Clearly, this polyhedral set will not contain a line and, therefore (cf. Theorem 3.5), have at least one extreme point.

Perhaps another important point is to notice that the result is posed assuming a minimisation problem, but it naturally holds for maximisation problems as well. In fact, maximising a function $f(x)$ is the same as minimising $-f(x)$, with the caveat that, although the optimal value x^* is the same in both cases, the optimal values are symmetric in sign (because of the additional minus we included in the problem being originally maximised).

This is important because we intend to design an algorithm that only inspects extreme points. This discussion guarantees that, even for the cases in which a whole set of optimal solution exists, some elements in that set will be extreme points anyway and thus identifiable by our method.

3.2.2 Finding optimal solutions

We now focus on the issue of being able to find and recognise extreme points as optimal solutions. In general, optimisation methods iterate the following steps:

1. Start from an initial (often feasible) solution;
2. Find a nearby solution with better value;
3. If none are available, return the best-known solution.

This very simple procedure happens to be the core idea of most optimisation methods. We will concentrate on how to identify directions of improvement and, as a consequence of their absence, how to identify optimality.

Starting from a point $x \in P$, we would like to move in a direction d that yields improvement. Definition 3.7 provides a formalisation of this idea.

Definition 3.7 (Feasible directions)**.** *Let $x \in P$, where $P \subset \mathbb{R}^n$ is a polyhedral set. A vector $d \in \mathbb{R}^n$ is a feasible direction at x if there exists $\theta > 0$ for which $x + \theta d \in P$.*

Figure 3.4 illustrates the concept. Notice that at extreme points, the relevant feasible directions are those along the edges of the polyhedral set since those are the directions that can lead to other extreme points.

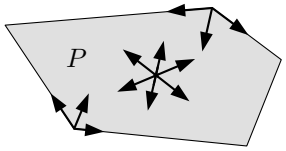


Figure 3.4: Feasible directions at different points of P

Let us now devise a way of identifying feasible directions algebraically. For that, let A be a $m \times n$ matrix, $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$. Consider the problem

$$\text{min. } \left\{c^\top x : Ax = b, x \geq 0\right\}.$$

Let x be a basic feasible solution (BFS) with basis $B = [A_{B(1)}, \ldots, A_{B(m)}]$. Recall that the basic variables x_B are given by

$$x_B = (x_{B(i)})_{i \in I_B} = B^{-1}b, \text{ with } I_B = \{B(1), \ldots, B(m)\} \subset J,$$

and that the remaining nonbasic variables x_N are such that $x_N = (x_j)_{j \in I_N} = 0$, with $I_N = J \setminus I_B$.

Moving to a neighbouring solution can be achieved by simply moving between adjacent basis, which can be accomplished without significant computational burden. This entails selecting a nonbasic variable x_j, $j \in I_N$, and increasing it to a positive value θ.

Equivalently, we can define a *feasible direction* $d = [d_N, d_B]$, where d_N represent the components associated with nonbasic variables and d_B those associated with basic variables and move from the point x to the point $x + \theta d$. The components d_N associated with the nonbasic variables are thus defined as

$$d = \begin{cases} d_j = 1 \\ d_{j'} = 0, \text{for all } j' \neq j, \end{cases}$$

with $j, j' \in I_N$. Notice that, geometrically, we are moving along a line in the dimension represented by the nonbasic variable x_j.

Now, feasibility might become an issue if we are not careful to retain feasibility conditions. To retain feasibility, we must observe that $A(x + \theta d) = b$, implying that $Ad = 0$. This allows us to define the components d_B of the direction vector d that is associated with the basic variables x_j, $j \in I_B$, since

$$0 = Ad = \sum_{j=1}^{n} A_j d_j = \sum_{i=1}^{m} A_{B(i)} d_{B(i)} + A_j = Bd_B + A_j$$

and thus $d_B = -B^{-1}A_j$ is the *basic direction* implied by the choice of the nonbasic variable x_j, $j \in I_N$, to become basic. The vector d_B can be thought of as the adjustments that must be made in the value of the other basic variables to accommodate the new variable becoming basic to retain feasibility.

Figure 3.5 provides a schematic representation of this process, showing how the change between adjacent basis can be seen as a movement between adjacent extreme points. Notice that it conveys a schematic representation of a $n = 5$ dimensional problem, in which we ignore all the m dimensions and concentrate on the $n-m$ dimensional projection of the feasibility set. This implies that the only constraints left are those associated with the nonnegativity of the variables $x \geq 0$, each associated with an edge of this alternative representation. Thus, when we set $n - m$ (nonbasic variables) to zero, we identify an associated extreme point. As $n - m = 2$, we can plot this alternative representation on $\mathbb{R}^2$.

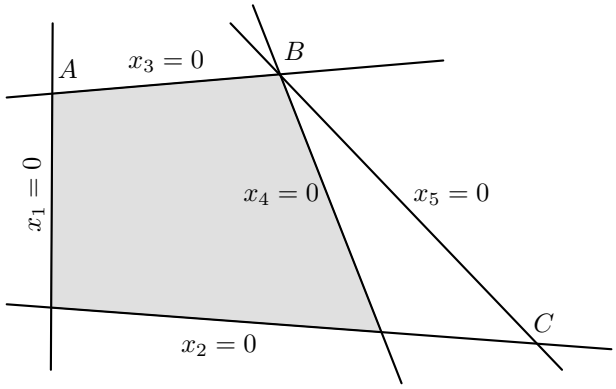


Figure 3.5: Example: $n = 5$ and $n - m = 2$. At A, $x_1 = x_3 = 0$ and $x_2, x_4, x_5 \geq 0$. Increasing x_1 while keeping x_3 zero leads to B. At B, suppose $I_N = \{3, 5\}$; by increasing x_3 while keeping x_5 zero would leads to C.

Clearly, overall feasibility, i.e., ensuring that $x \geq 0$ can only be retained if $\theta > 0$ is chosen appropriately small. This can be achieved if the following is observed:

1. All the other nonbasic variables remain valued at zero, that is, $x_{j'} = 0$ for $j' \in I_N \setminus \{j\}$.
2. if x is a *nondegenerate* extreme point, then all $x_B > 0$ and thus $x_B + \theta d_B \geq 0$ for appropriately small $\theta > 0$.
3. if x is a *degenerate* extreme point: $d_{B(i)}$ might not be feasible since, for some $B(i)$, if $d_{B(i)} < 0$ and $x_{B(i)} = 0$, any $\theta > 0$ will make $x_{B(i)} < 0$.

We will see later that we can devise a simply rule to define a value for θ that guarantees the above will be always observed. For now, we will put this discussion on hold and focus on the issue of how to guide the choice of which nonbasic variable x_j, $j \in I_N$, to select to become basic.

3.2.3 Moving towards improved solutions

A simple yet efficient way of deciding which nonbasic component $j \in I_N$ to make basic is to consider the immediate potential benefit that it would have for the objective function value.

Using objective function $c^\top x$, if we move along the feasible direction d as previously defined, we have that the objective function value changes by

$$c^\top d = c_B^\top d_B + c_j = c_j - c_B^\top B^{-1} A_j,$$

where $c_B = [c_{B(1)}, \ldots, c_{B(m)}]$. The quantity $c_j - c_B B^{-1} A_j$ can be used, for example, in a greedy fashion, meaning that we choose the nonbasic variable index $j \in I_N$ with greatest *potential of improvement*.

First, let us formally define this quantity, which is known as the *reduced cost*.

Definition 3.8 (Reduced cost)**.** *Let x be a basic solution associated with the basis B and let $c_B = [c_{B(1)}, \ldots, c_{B(m)}]$ be the objective function coefficients associated with the basis B. For each nonbasic variable x_j, with $j \in I_N$, we define the reduced cost $\overline{c}_j$ as*

$$\overline{c}_j = c_j - c_B^\top B^{-1} A_j.$$

The name reduced cost is motivated by the fact that it quantifies a cost change onto the reduced space of the basic variables. In fact, the reduced cost is calculating the change in the objective function caused by the increase in one unit of the nonbasic variable x_j elected to become basic (represented by the c_j component) and the associated change caused by the accommodation in the basic variable values to retain feasibility, as discussed in the previous section. Therefore, the reduced cost can be understood as a *marginal value* of change in the objective function value associated with each nonbasic variable.

Let us demonstrate this with a numerical example. Consider the following linear programming problem

$$\begin{aligned} \text{min. } & 2x_1 + 1x_2 + 3x_3 + 2x_4 \\ \text{s.t.: } & x_1 + x_2 + x_3 + x_4 = 2 \\ & 2x_1 + 3x_3 + 4x_4 = 2 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Let $I_B = \{1, 2\}$, yielding

$$B = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}.$$

Thus, $x_3 = x_4 = 0$ and $x = (1, 1, 0, 0)$. The basic direction d_B^3 for x_3 is given by

$$d_B^3 = -B^{-1} A_3 = -\begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} -3/2 \\ 1/2 \end{bmatrix}.$$

which gives $d^3 = (-3/2, 1/2, 1, 0)$. Analogously, for x_4, we have

$$d_B^4 = -B^{-1} A_4 = -\begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}.$$

The (reduced) cost of moving along the direction given by d^3 is

$$c^\top d^3 = (c_1, c_2)^\top d_B^3 + c_3 = ((-3/2)2 + (1/2)1) + 3 = 0.5$$

while moving along d^4 has a cost of

$$c^\top d^4 = (c_1, c_2)^\top d_B^4 + c_4 = ((-2)2 + (1)1) + 2 = -1.$$

Thus, between d^3 and d^4, d^4 is a better direction since its reduced cost indicates a reduction of 1 unit of the objective function per unit of x_4. In contrast, the reduced cost associated with d^3 indicates an increase of 0.5 units of the objective function per unit of x_3, indicating that this is a direction to be avoided as we are minimising the problem. Clearly, the willingness to choose $x_{j'}$, $j' \in I_N$ as the variable to become basic will depend on whether the scalar $c_{j'} - (c_j)^\top_{j \in I_B} d_B$ is negative (recall that we want to minimise the problem, so the smaller the total cost, the better). Another point is how large in module the reduced cost is. Recall that the reduced is, in fact, a measure of the marginal value associated with the increase in value of the nonbasic variable. Thus the larger (in module) it is, the quicker the objective function value will decrease per unit of increase of the nonbasic variable value. One interesting thing to notice is what is the reduced cost associated with basic variables. Recall that $B = [A_{B(1)}, \ldots, A_{B(m)}]$ and thus $B^{-1}[A_{B(1)}, \ldots, A_{B(m)}] = I$. Therefore $B^{-1}A_{B(i)}$ is the i^{th} column of I, denoted e_i, implying that

$$\overline{c}_{B(i)} = c_{B(i)} - c_B^\top B^{-1} A_{B(i)} = c_{B(i)} - c_B^\top e_i = c_{B(i)} - c_{B(i)} = 0.$$

3.2.4 Optimality conditions

Now that we have seen how to identify promising directions for improvement, we have incidentally developed a framework for identifying the optimality of a given basic feasible solution (BFS). That is, a BFS from which no direction of improvement can be found must be locally optimal. And, since local optimality implies global optimality in the presence of convexity (the convexity of linear programming problems was attested in Chapter 2; the global optimality in the presence of convexity is a result discussed in detail in Part II), we can declare this BFS as an optimal solution.

Theorem 3.9 establishes the optimality of a BFS from which no improving feasible direction can be found without explicitly relying on the notion of convexity.

Theorem 3.9 (Optimality conditions)**.** *Consider the problem P : min. $\{c^\top x : Ax = b, x \geq 0\}$. Let x be the BFS associated with a basis B and let $\overline{c}$ be the corresponding vector of reduced costs.*

(1) if $\overline{c} \geq 0$, then x is optimal.

(2) if x is optimal and nondegenerate, then $\overline{c} \geq 0$.

Proof. To prove (1), assume that $\overline{c}_j \geq 0$, let y be a feasible solution to P, and $d = y - x$. We have that $Ax = Ay = b$ and thus $Ad = 0$. Equivalently:

$$\begin{aligned} &Bd_B + \sum_{j \in I_N} A_j d_j = 0 \Rightarrow d_B = -\sum_{j \in I_N} B^{-1} A_j d_j, \text{ implying that} \\ &c^\top d = c_B^\top d_B + \sum_{j \in I_N} c_j d_j = \sum_{j \in I_N} (c_j - c_B^\top B^{-1} A_j) d_j = \sum_{j \in I_N} \overline{c}_j d_j \end{aligned} \tag{3.3}$$

We have that $x_j = 0$ and $y_j \geq 0$ for $j \in I_N$. Thus, $d_j \geq 0$ and $\overline{c}_j d_j \geq 0$ for $j \in I_N$, which implies that $c^\top d \geq 0$ (cf. (3.3)). Consequently,

$$c^\top d \geq 0 \Rightarrow c^\top (y - x) \geq 0 \Rightarrow c^\top y \geq c^\top x, \text{ i.e., } x \text{ is optimal.}$$

To prove (2) by contradiction, assume that x is optimal with $\overline{c}_j < 0$ for some $j \in I_N$. Thus, we could improve on x moving along this j^{th} direction d, contradicting the optimality of x. □

A couple of remarks are worth making at this point. First, notice that, in the presence of degeneracy, it might be that x is optimal with $\overline{c}_j < 0$ for some $j \in I_N$. Luckily, the simplex method manages to get around this issue in an effective manner, as we will see in the next chapter. Another point to notice is that, if $\overline{c}_j > 0$, $\forall j \in I_N$, then x is a *unique optimal*. Analogously, if $\overline{c} \geq 0$ with $c_j = 0$ for some $j \in I_N$, then it means that moving in that direction will cause no change in the objective function value, implying that both BFS are "equally optimal" and that the problem has multiple optimal solutions.

3.3 Exercises

Exercise 3.1: Properties of basic solutions

Prove the following theorem:

Theorem (Linear independence and basic solutions)**.** *Consider the constraints $Ax = b$ and $x \geq 0$, and assume that A has m LI rows $M = \{1, \ldots, m\}$. A vector $\overline{x} \in \mathbb{R}^n$ is a* basic solution *if and only if we have that $A\overline{x} = b$ and there exists indices $B(1), \ldots, B(m)$ such that*

(1) The columns $A_{B(1)}, \ldots, A_{B(m)}$ are LI
(2) If $j \neq B(1), \ldots, B(m)$, then $\overline{x}_j = 0$.

Note: the theorem is proved in the notes. Use this as an opportunity to revisit the proof carefully, and try to take as many steps without consulting the text as you can. This is a great exercise to help you internalise the proof and its importance in the context of the material. I strongly advise against blindly memorising it, as I suspect you will never (in my courses, at least) be requested to recite the proof literally.

Exercise 3.2: Basic solutions and extreme points

Consider the set $P = \{\mathbf{x} \in \mathbb{R}^2 : x_1 + x_2 \leq 6,\ x_2 \leq 3,\ x_1, x_2 \geq 0\}$.

(a) Enumerate all basic solutions, and identify those that are basic feasible solutions.

(b) Draw the feasible region, and identify the extreme point associated with each basic feasible solution.

(c) Consider a minimization problem with the cost vector $\mathbf{c}' = (c_1, c_2, c_3, c_4) = (-2, \frac{1}{2}, 0, 0)$. Compute the basic directions and the corresponding reduced costs of the nonbasic variables at the basic solution $\mathbf{x}' = (3, 3, 0, 0)$ with $\mathbf{x}'_B = (x_1, x_2)$ and $\mathbf{x}'_N = (x_3, x_4)$; either verify that $\mathbf{x}'$ is optimal, or move along a basic direction which leads to a better solution.

Exercise 3.3: Degeneracy - part 1

Given the linear program given below,

$$
\begin{array}{lrcl}
\max & 2x_1 + x_2 & & \\
\text{s.t.} & & & \\
& 2x_1 + 2x_2 & \leq & 9 \\
& 2x_1 - x_2 & \leq & 3 \\
& x_1 - x_2 & \leq & 1 \\
& 4x_1 - 3x_2 & \leq & 5 \\
& x_1 & \leq & 2.5 \\
& x_2 & \leq & 4 \\
& & & \\
& x_1, x_2 \geq 0 & &
\end{array}
$$

(a) Solve the problem and find the optimal point.

(b) What are the degenerated solutions and the bases forming them?

(c) Plot the constraints and degenerated points and determine why those are degenerated.

Exercise 3.4: Feasible direction

Let x be a point in a polyhedron $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. Show that a vector $d \in \mathbb{R}^n$ is a feasible direction at $x \in P$ if and only if $Ad = 0$ and $d_i \geq 0$ for all i for which $x_i = 0$. A feasible direction of P at point x is a vector $d \neq 0$ such that $x + \theta d \in P$ for some $\theta > 0$.

Exercise 3.5: Optimality of extreme points

Prove the following theorem.

Theorem (Optimality of extreme points)**.** *Let $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ be a polyhedral set and $c \in \mathbb{R}^n$. Consider the problem*

$$z = min. \left\{ c^\top x : x \in P \right\}.$$

Suppose that P has at least one extreme point and that there exists an optimal solution. Then, there exists an optimal solution that is an extreme point of P.

Note: see Exercise 3.1.

Chapter 4

The simplex method

4.1 Developing the simplex method

In Chapter 3, we discussed all the necessary theoretical aspects required to develop the simplex method. In this chapter, we will concentrate on operationalising the method from a computational standpoint.

4.1.1 Calculating step sizes

One discussion that we purposely delayed was that of how to define the value of the step size θ to be taken in the feasible direction d. Let $c \in \mathbb{R}^n$, A be a $m \times n$ full-rank matrix, b a nonnegative m-sized vector[1] and $J = \{1, \ldots, n\}$. Consider the linear programming problem P in the standard form

$$(P) : \text{min. } \left\{c^\top x : Ax = b, x \geq 0\right\}.$$

Building upon the elements we defined in Chapter 3, employing the simplex method to solve P consists of the following set of steps:

1. Start from a nondegenerate basic feasible solution (BFS)
2. Find a negative reduced cost component $\overline{c}_j$. If $\overline{c} \geq 0$, return the current solution.
3. Move along the feasible direction $d = (d_B, d_N)$, where $d_j = 1$, $d_{N\setminus\{j\}} = 0$ and $d_B = -B^{-1}A_j$.

Moving along the feasible direction d towards $x + \theta d$ (with scalar $\theta > 0$) makes $x_j > 0$ (i.e., $j \in I_N$ enter the basis) while reducing the objective value at a rate of $\overline{c}_j$. Thus, one should move as furthest as possible (say, take a step of length $\overline{\theta}$) along the direction d, which incurs on an objective value change of $\overline{\theta}(c^\top d) = \overline{\theta}\overline{c}_j$.

Moving as far along the feasible direction d as possible while observing that feasibility is retained is equivalent to setting $\overline{\theta}$ as

$$\overline{\theta} = \max\left\{\theta \geq 0 : A(x + \overline{\theta}d) = b,\ x + \overline{\theta}d \geq 0\right\}.$$

Recall that, by construction, the feasible direction d, we have that $Ad = 0$ and thus $A(x + \overline{\theta}d) = Ax = b$. Therefore, the only feasibility condition that can be violated when setting $\overline{\theta}$ too large is the nonnegativity of all variables, i.e., $x + \overline{\theta}d \geq 0$.

[1] notice that this can be assumed without loss of generality, by a trivial transformation that adds constants on both sides and, thus, does not change the constraint

To prevent this from being the case, all basic variables $i \in I_B$ for which the component in the basic direction vector d_B is negative must be guaranteed to retain

$$x_i + \overline{\theta} d_i \geq 0 \Rightarrow \overline{\theta} \leq -\frac{x_i}{d_i}.$$

Therefore, the maximum value $\overline{\theta}$ is that that can be increased until the first component of x_B turns zero. Or, more precisely put,

$$\overline{\theta} = \min_{i \in I_B : d_i < 0} \left\{ -\frac{x_{B(i)}}{d_{B(i)}} \right\}.$$

Notice that we only need to consider those basic variables with components d_i, $i \in I_B$, that are negative. This is because, if $d_i \geq 0$, then $x_i + \overline{\theta} d_i \geq 0$ holds for any value of $\overline{\theta} > 0$. This means that the constraints associated with these basic variables (referring to the representation in Figure 3.5) do not limit the increase in value of the select nonbasic variable. Notice that this can lead to a pathological case in which none of the constraints limits the increase in value of the nonbasic variable, which indicates that the problem has an unbounded direction of decrease for the objective function. In this case, we say that the problem is *unbounded*.

Another important point is the assumption of a nondegenerate BFS. The nondegeneracy of the BFS implies that $x_{B(i)} > 0$, $\forall i \in I_B$ and, thus, $\overline{\theta} > 0$. In the presence of degeneracy, one can infer that the definition of the step size $\overline{\theta}$ must be done more carefully.

Let us consider the numerical example we used in Chapter 3 with a generic objective function.

$$\begin{aligned} \text{min. } & c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 \\ \text{s.t.: } & x_1 + x_2 + x_3 + x_4 = 2 \\ & 2x_1 + 3x_3 + 4x_4 = 2 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Let $c = (2, 0, 0, 0)$ and $I_B = \{1, 2\}$. The reduced cost of the nonbasic variable x_3 is

$$\overline{c}_3 = c_3 - (c_1, c_2)^\top [-3/2, 1/2] = -3.$$

where $d_B = [-3/2, 1/2]$. As x_3 increases in value, only x_1 decreases, since $d_1 < 0$. Therefore, the largest $\overline{\theta}$ for which $x_1 \geq 0$ is $-(x_1/d_1) = 2/3$. Notice that this is precisely the value that makes $x_1 = 0$, i.e., nonbasic. The new basic variable is now $x_3 = 2/3$, and the new (adjacent, as we will see next) extreme point is

$$\overline{x} = x + \theta d = (1, 1, 0, 0) + (2/3)(-3/2, 1/2, 1, 0) = (0, 4/3, 2/3, 0).$$

4.1.2 Moving between adjacent bases

Once we have defined the optimal step size $\overline{\theta}$, we move to a new BFS $\overline{x}$. That new solution is such that, for the nonbasic variable $j \in I_N$ selected to be basic, we observe that $\overline{x}_j = \theta$. Now, let us define as l the index of the basic variable that first becomes zero, that is, the variable that defines the value of $\overline{\theta}$. More precisely put, let

$$l = \operatorname*{argmin}_{i \in I_B : d_i < 0} \left\{ -\frac{x_{B(i)}}{d_{B(i)}} \right\} \text{ and, thus, } \overline{\theta} = \left\{ -\frac{x_{B(l)}}{d_{B(l)}} \right\}. \tag{4.1}$$

By moving to the BFS $\overline{x}$ by making $\overline{x} = x + \overline{\theta}d$, we are in fact moving from the basis B to an adjacent basis $\overline{B}$, defined as

$$\overline{B} = \begin{cases} \overline{B}(i) = B(i), \text{ for } i \in I_B \setminus \{l\} \\ \overline{B}(i) = j, \text{ for } i = l. \end{cases}$$

Notice that the new basis $\overline{B}$ only has one pair of variables swapped between basic and nonbasic when compared against B. Analogously, the basic matrix associated with $\overline{B}$ is given by

$$\begin{bmatrix} | & & | & | & | & & | \\ A_{B(1)} & \dots & A_{B(l-1)} & A_j & A_{B(l+1)} & \dots & A_{B(m)} \\ | & & | & | & | & & | \end{bmatrix},$$

where the middle column representing that the column $A_{B(l)}$ has been replaced with A_j.

Theorem 4.1 provide the technical result that formalises our developments so far.

Theorem 4.1 (Adjacent bases)**.** *Let A_j be the column of the matrix A associated with the selected nonbasic variable index $j \in I_N$. And let l be defined as* (4.1)*, with $A_{B(i)}$ being its respective column in A. Then*

(1) The columns $A_{B(i)}$ and A_j are linearly independent. Thus, $\overline{B}$ is a basic matrix;

(2) The vector $\overline{x} = x + \overline{\theta}d$ is a BFS associated with $\overline{B}$.

Proof. We start by proving (1). By contradiction, assume that $\left\{A_{B(i)}\right\}_{i \in I_B \setminus \{l\}}$ and A_j are not linearly independent. Thus, there exist $\{\lambda_i\}_{i=1}^m$ (not all zeros) such that

$$\sum_{i=1}^{m} \lambda_i A_{\overline{B}(i)} = 0 \Rightarrow \sum_{i=1}^{m} \lambda_i B^{-1} A_{\overline{B}(i)} = 0,$$

making $B^{-1}A_{\overline{B}(i)}$ not linearly independent. However, $B^{-1}A_{\overline{B}(i)} = B^{-1}A_{B(i)}$ for $i \in I_B \setminus \{l\}$ and thus are all unit vectors e_i with the l^{th} component zero.

Now, $B^{-1}A_j = -d_B$, with component $d_{B(l)} \neq 0$, is linearly independent from $B^{-1}A_{B(i)} = B^{-1}A_{\overline{B}(i)}$. Thus, $\left\{A_{\overline{B}(i)}\right\}_{i \in I_{\overline{B}}} = \left\{A_{B(i)}\right\}_{i \in B \setminus \{l\}} \cup \{A_j\}$ are linearly independent, forming the contradiction.

Now we focus on proving (2). We have that $\overline{x} \geq 0$, $A\overline{x} = b$ and $\overline{x}_j = 0$ for $j \in I_{\overline{N}} = J \setminus I_{\overline{B}}$. This, combined with $\left\{\overline{B}(i)\right\}_{i \in I_{\overline{B}}}$ being linearly independent (cf. 1), completes the proof. □

We have finally compiled all the elements we need to state the simplex method pseudocode, which is presented in Algorithm 1. One minor detail in the presentation of the algorithm is the use of the auxiliary vector u. This allows for the precalculation of the components of $d_B = -B^{-1}A_j$ (notice the changed signed) to test for unboundedness, that is, the lack of a constraint (and associated basic variable) that can limit the increase of the chosen nonbasic variable.

The last missing element is proving that Algorithm 1 eventually converges to an optimal solution, should one exists. This is formally stated in Theorem 4.2.

Theorem 4.2 (Convergence of the simplex method)**.** *Assume that P has at least one feasible solution and that all BFS are nondegenerate. Then, the simplex method terminates after a finite number of iterations, in one of the following states:*

(1) The basis B and the associated BFS are optimal; or

(2) d is such that $Ad = 0$, $d \geq 0$, and $c^\top d < 0$, with optimal value $-\infty$.

Proof. If the condition in Line 3 of Algorithm 1 is not met, then B and associated BFS are optimal, c.f. Theorem 3.9. Otherwise, if Line 5 condition is met, then d is such that $Ad = 0$ and $d \geq 0$, implying that $x + \theta d \in P$ for all $\theta > 0$, and a value reduction $\theta\overline{c}$ of $-\infty$.

Finally, notice that $\overline{\theta} > 0$ step sizes are taken along d satisfy $c^\top d < 0$. Thus, the value of successive solutions is strictly decreasing and no BFS can be visited twice. As there is a finite number of BFS, the algorithm must eventually terminate. □

Algorithm 1 Simplex method

1: **initialise.** Initial basis B, associated BFS x, and reduced costs $\overline{c}$.
2: **while** $\overline{c}_j < 0$ for some $j \in N$ **do**
3: Choose some j for which $\overline{c_j} < 0$. Calculate $u = B^{-1}A_j$.
4: **if** $u \leq 0$ **then**
5: **return** $z = -\infty$.
6: **else**
7: $\overline{\theta} = \min_{i \in I_B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$ and $l = \operatorname{argmin}_{i \in B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$
8: Set $x_j = \overline{\theta}$ and $x_B = x - \overline{\theta}u$. Form new basis $B = B \setminus \{l\} \cup \{j\}$.
9: Calculate $\overline{c}_j = c_j - c_B^\top B^{-1} A_j$ for all $j \in N$.
10: **end if**
11: **end while**
12: **return** optimal basis B and optimal solution x.

4.1.3 A remark on degeneracy

We now return to the issue related to degeneracy. As we discussed earlier, degeneracy is an important pitfall for the simplex method. To recognise that the method arrived at a degenerate BFS, one must observe how the values of the basic variables are changing. If, for $\overline{\theta}$, more than one basic variable become zero at $\overline{x} = x + \overline{\theta}d$, then $\overline{B}$ degenerate.

Basically, if the current BFS is degenerate, $\overline{\theta} = 0$ can happen when $x_{B(l)} = 0$ and the component $d_{B(l)} < 0$. Notice that a step size of $\overline{\theta} = 0$ is the only option to prevent infeasibility in this case. Nevertheless, a new basis can still be defined by replacing $A_{B(l)}$ with A_j in B. However, $\overline{x} = x + \overline{\theta}d = x$. Sometimes, even though the method is effectively staying at the same extreme point, changing the basis on a degenerate solution might eventually expose a direction of improvement, a phenomenon that is called *stalling*. In an extreme case, it might be so that the selection of the next basic variable is such that the same extreme point is recovered over and over again, which is called *cycling*. The latter can be prevented by a specific technique for carefully selecting the variable that will enter the basis.

Figure 4.1 illustrates an example of stalling. In that, a generic problem with five variables is illustrated, with any given basis being formed by three variables. For example, at y, $I_B = 3, 4, 5$, with $x_3 > 0$, $x_4 > 0$, and $x_5 > 0$. Notice that there are multiple bases representing x. Suppose we have $I_B = \{1, 2, 3\}$. Notice that in this case, we have $x_2 = x_3 = 0$ even though x_2 and x_3 are basic variables. Now, suppose we perform one simplex iteration and move to the adjacent basis $I_B = \{1, 3, 4\}$). Even though the extreme point is the same (x), this new basis exposes

the possibility of moving to the nondegenerate basis $I_B = \{3, 4, 5\}$, i.e., y. Assuming that we are minimising, any direction with a negative component in the y-axis direction will represent an improvement in the objective function (notice the vector c, which is parallel to the y-axis and points upwards). Thus, the method is only stalled by the degeneracy of x but does not cycle.

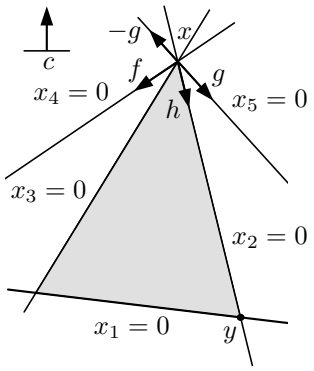


Figure 4.1: $I_N = \{4, 5\}$ for x; f ($x_5 > 0$) and g ($x_4 > 0$) are basic directions. Making $I_N = \{2, 5\}$ lead to new basic directions h ($x_4 > 0$) and $-g$ ($x_2 > 0$).

4.2 Implementing the simplex method

We now focus on some specific details related to alternative simplex method implementations. In a general sense, implementations of the simplex method vary in terms of how the selection of the nonbasic variables with $\bar{c}_j < 0$ that enters the basis is made. They also differ in the way the basic variable $l = \text{argmin}_{i \in B | d_i < 0} \left\{ -x_{B(i)}/d_{B(i)} \right\}$ to leave the basis in case of ties might be of interest, especially in the attempt of preventing cycling.

Another important aspect related to implementations of the simplex method is how matrices are represented, its consequences on memory utilisation, and how the operations related to matrix inversion are carried out.

4.2.1 Pivot or variable selection

The rules utilised for making choices regarding entering and leaving variables are generally referred to as *pivoting rules*. However, the term most commonly used to refer to the selection of nonbasic variables to enter the basis is *(simplex) pricing rules*.

- *Greedy selection* (or Dantzig's rule): choose x_j, $j \in I_N$, with largest $|\bar{c}_j|$. Prone to cycling.
- *Index-based order* (or Bland's rule): choose x_j, $j \in I_N$, with smallest j. It prevents cycling but is computationally inefficient.
- *Reduced cost pricing*: calculate $\overline{\theta}$ for all (or some) $j \in N$ and pick smallest $\overline{\theta}\bar{c}_j$. Calculating the actual observed change for all nonbasic variables is too computationally expensive. Partial pricing refers to the idea of only considering a subset of the nonbasic variables to calculate $\overline{\theta}\bar{c}_j$.

- *Devex*[2] and *steepest-edge*[3]: most commonly used by modern implementations of the simplex method, available in professional-grade solvers.

4.2.2 The revised simplex method

The central element in the simplex method is the calculation of the matrix $B^{-1}A_j$, from which the reduced cost vector $\bar{c}_j$, $j \in I_N$, the basic feasible direction vector d_B and the step size $\overline{\theta}$ can be easily computed.

First, let us consider a more natural way of implementing the simplex method so then we can point out how the method can be revised to be more computationally efficient. We will refer to this version as the "naive simplex". The main differences between the naive and its revised version will be how $B^{-1}A_j$ is computed and the amount of information being carried over between iterations.

A somewhat natural way to implement the simplex method would be to store the term $p^\top = c_B^\top B^{-1}$ in an auxiliary variable by solving the linear system $p^\top B = c_B^\top$. These terms are important, as we will see later, and they are often referred to as the *simplex multipliers*.

Once the simplex multipliers p are available, the reduced cost c_j associated with the nonbasic variable index $j \in I_N$ is simply

$$\bar{c}_j = c_j - p^\top A_j.$$

Once the column A_j is selected, we can then solve a second linear system $Bu = A_j$ to determine $u = B^{-1}A_j$.

The key observation that can yield computational savings is that we do not need to solve two linear systems. As one can notice, there is a common term between the two, the inverse of the basic matrix B^{-1}. If this matrix can be made available at the beginning of each iteration, then the terms $c_B^\top B^{-1}$ and $B^{-1}A_j$ can be easily and more cheaply (computationally) obtained.

For that to be possible, we need an efficient manner to update the matrix B^{-1} after each iteration. To see how that can be accomplished, recall that

$$\begin{aligned} B &= [A_{B(1)}, \ldots, A_{B(m)}], \text{ and} \\ \overline{B} &= [A_{B(1)}, \ldots, A_{B(l-1)}, A_j, A_{B(l+1)} \ldots, A_{B(m)}], \end{aligned}$$

where the l^{th} column $A_{B(l)}$ is precisely how the adjacent bases B and $\overline{B}$ differ, with A_j replacing $A_{B(l)}$ in $\overline{B}$.

We can devise an efficient manner to update B^{-1} into $\overline{B}^{-1}$ utilising *elementary row operations*. First, let us formally define the concept.

Definition 4.3 (Elementary row operations)**.** *Adding a constant multiple of one row to the same or another row is called an elementary row operation.*

Defining elementary row operations is the same of devising a matrix $Q = I + D_{ij}$ to premultiply B, where

$$D = \begin{cases} D_{ij} = \beta, \\ D_{i'j'} = 0 \text{ for all } i', j' \neq i, j. \end{cases}$$

[2] P. M. J. Harris (1973), Pivot Selection Methods in the Devex LP Code, Math. Prog., 57, 341–374.
[3] J. Forrest & D. Goldfarb (1992), Steepest-Edge Simplex Algorithms for LP, Math. Prog., 5, 1–28.

Calculating $QB = (I + D)B$ is the same as having the j^{th} row of B multiplied by a scalar β and then having the resulting j^{th} row added to the i^{th} row of B. Before we continue, let us utilise a numerical example to clarify this procedure. Let

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

and suppose we would like to multiply the third row by 2 and have it then added to the first row. That means that $D_{13} = 2$ and that $Q = I + D$ would be

$$Q = \begin{bmatrix} 1 & & 2 \\ & 1 & \\ & & 1 \end{bmatrix}$$

Then premultiplying B by Q yields

$$QB = \begin{bmatrix} 11 & 14 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

As a side note, we have that Q^{-1} exists since $\mathbf{det}(Q) = 1$. Furthermore, sequential elementary row operations $\{1, 2, \ldots, k\}$ can be represented as $Q = Q_1 Q_2, \ldots Q_k$.

Going back to the purpose of updating B^{-1} into $\overline{B}^{-1}$, notice the following. Since $B^{-1}B = I$, each term $B^{-1}A_{B(i)}$ is the i^{th} unit vector e_i (the i^{th} column of the identity matrix). That is,

$$B^{-1}\overline{B} = \begin{bmatrix} | & & | & | & | & & | \\ e_1 & \ldots & e_{l-1} & u & e_{l+1} & \ldots & e_m \\ | & & | & | & | & & | \end{bmatrix} = \begin{bmatrix} 1 & & u_1 & & \\ & \ddots & \vdots & & \\ & & u_l & & \\ & & \vdots & \ddots & \\ & & u_m & & 1 \end{bmatrix},$$

where $u = B^{-1}A_j$. We want to define an elementary row operation matrix Q such that $QB^{-1} = \overline{B}^{-1}$, or $QB^{-1}\overline{B} = I$. Therefore Q will be such that the elementary row operations turn $B^{-1}\overline{B}$ into an identity matrix, i.e., that turn the vector u into the unit vector e_l.

The main trick is that we do not need matrix multiplication to achieve it, saving considerably in computational resources. Instead, we can simply apply the elementary row operations, focusing only on the column u and the operations required to turn it into the unit vector e_l. This can be achieved by:

1. for each $i \neq l$, multiply the l^{th} row by $-\frac{u_i}{u_l}$ and add to the i^{th} row. This replaces u_i with zero for all $i \in I \setminus \{l\}$.

2. Divide the l^{th} row by u_l. This replaces u_l with one.

We can restate the simplex method in its revised form. This is presented in Algorithm 2.

Notice that in Algorithm 2, apart from the initialisation step, no linear systems are directly solved. Instead, elementary row operations (ERO) are performed, leading to computational savings.

Algorithm 2 Revised simplex method

initialise. Initial basis B, associated BFS x, and B^{-1} are available.
Calculate $p^\top = c_B^\top B^{-1}$ and $\overline{c}_j = c_j - p^\top A_j$ for $j \in N$.
while $\overline{c}_j < 0$ for some $j \in N$ **do**
    Choose some j for which $\overline{c_j} < 0$. Calculate $u = B^{-1}A_j$.
    if $u \leq 0$ **then**
        return $z = -\infty$.
    else
        $\overline{\theta} = \min_{i \in I_B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$ and $l = \operatorname{argmin}_{i \in B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$
        Set $x_j = \overline{\theta}$ and $x_B = x - \overline{\theta} u$.
        Form the matrix $[B^{-1} \,|\, u]$ and perform EROs to convert it to $[\overline{B}^{-1} \,|\, e_l]$.
        Make $B = B \setminus \{l\} \cup \{j\}$ and $B^{-1} = \overline{B}^{-1}$.
        Calculate $p^\top = c_B^\top B^{-1}$ and $\overline{c}_j = c_j - p^\top A_j$ for all $j \in I_N = J \setminus I_B$.
    end if
end while
return optimal basis B and optimal solution x.

The key feature of the revised simplex method is a matter of representation and, thus, memory allocation savings. Algorithm 2 only require keeping in memory a matrix of the form

$$\begin{bmatrix} p & | & p^\top b \\ B^{-1} & | & u \end{bmatrix}$$

which, after each series of elementary row operations, yield not only $\overline{B}^{-1}$ but also the updated simplex multipliers $\overline{p}$ and $\overline{p}^\top b = c_B^\top \overline{B}^{-1} b = c_B^\top \overline{x}_B$, which represents the objective function value of the new basic feasible solution $\overline{x} = [\overline{x}_B, \overline{x}_N]$. This bookkeeping savings will become obvious once we discuss the tabular (or non-revised) version of the simplex method.

Three main issues arise when considering the efficiency of implementations of the simplex method, namely, matrix (re)inversion, representation in memory, and the use of matrix decomposition strategies.

- *Reinversion:* localised updates have the side effect of accumulating truncation and round-off error. To correct this, solvers typically rely on periodically recalculating B^{-1}, which, although costly, can avoid numerical issues.
- *Representation:* A sparse representation of $Q_n = Q_1 Q_2 \dots Q_{k-1}$ can be kept instead of updating B^{-1}. Recall that $u = \overline{B}^{-1} A_j = Q B^{-1} A_j$. For larger problems, that means that a trade-off between memory allocation and the number of matrix-matrix multiplications.
- *Decomposition:* Decomposed (e.g., LU decomposition) forms of B are used to improve efficiency in storage and the solution of the linear systems to exploit the typical sparsity of linear programming problems.

4.2.3 Tableau representation

The tableau representation of the simplex method is useful as a concept presentation tool. It consists of a naive memory-intensive representation of the problem elements as they are iterated

between each basic feasible solution. However, it is a helpful representation from a pedagogical standpoint and will be useful for explaining further concepts in the upcoming chapters.

In contrast to the revised simplex method, instead of updating only B^{-1}, we consider the complete matrix

$$B^{-1}[A \mid b] = \left[B^{-1}A_1, \ldots, B^{-1}A_n \mid B^{-1}b\right].$$

Furthermore, we adjoint a row representing the reduced cost vector $\overline{c}^\top = c^\top - c_B^\top B^{-1}A$ and the negative of the objective function value for the current basis, $-c_B x_B = -c_B^\top B^{-1}b$, a row often referred to as the *zeroth row*. The reason why we consider the negative sign is that it allows for a simple updating rule for the zeroth row, by performing elementary row operations to make zero the j^{th} element associated with the nonbasic variable in B that becomes basic in $\overline{B}$.

The full tableau representation is given by

$$\begin{array}{|c|c|}\hline c^\top - c_B^\top B^{-1}A & -c_B^\top B^{-1}b \\ \hline B^{-1}A & B^{-1}b \\ \hline \end{array} \Rightarrow \begin{array}{|ccc|c|}\hline \overline{c}_1 & \cdots & \overline{c}_n & -c_B x_B \\ \hline & & & x_{B(1)} \\ B^{-1}A_1 & \cdots & B^{-1}A_n & \vdots \\ & & & x_{B(m)} \\ \hline \end{array}$$

In this representation, we say that the j^{th} column associated with the nonbasic variable to become basic is the *pivot column* u. Notice that, since the tableau exposes the reduced costs $\overline{c}_j$, $j \in I_N$, it allows for trivially applying the greedy pricing strategy (by simply choosing the variables with a negative reduced cost with the largest module).

The l^{th} row associated with the basic variable selected to leave the basis is the *pivot row*. Again, the tableau representation facilitates the calculation of the ratios used in choosing the basic variable l to leave the basis since it amounts to simply calculating the ratios between the elements on the rightmost column and those in the pivot column, disregarding those that present entries less or equal than zero and the zeroth row. The row with the minimal ratio will be the row associated with the current basic variable leaving the basis.

Once a pivot column and a pivot row have been defined, it is a matter of performing elemental row operations utilising the pivot row to turn the pivot column into the unit vector e_l and turn to zero the respective zeroth element (recall that basic variables have zero reduced costs). This is the same as using elementary row operations using the pivot row to turn all elements in the pivot column zero, except for the *pivot element* u_l, which is the intersection of the pivot row and the pivot column, that must be turned into one. The above highlights the main purpose of the tableau representation, which is to facilitate hand calculation.

Notice that, as we have seen before, performing elementary row operations to convert the pivot column u into e_l converts $B^{-1}[A \mid b]$ into $\overline{B}^{-1}[A \mid b]$. Analogously, turning the entry associated with the pivot column u in the zeroth row to zero converts $[c^\top - c_B^\top B^{-1}A \mid -c_B^\top B^{-1}b]$ into $[c^\top - c_{\overline{B}}^\top \overline{B}^{-1}A \mid -c_{\overline{B}}^\top \overline{B}^{-1}b]$.

Let s return to the paint factory example from section , which in its standard form can be written

as

$$\begin{aligned} \text{max. } & z = 5x_1 + 4x_2 && (4.2)\\ \text{s.t.: } & 6x_1 + 4x_2 + x_3 = 24 && (4.3)\\ & x_1 + 2x_2 + x_4 = 6 && (4.4)\\ & x_2 - x_1 + x_5 = 1 && (4.5)\\ & x_2 + x_6 = 2 && (4.6)\\ & x_1, \ldots, x_6 \geq 0. && (4.7) \end{aligned}$$

The sequence of tableaus for this problem is given by

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	-5	-4	0	0	0	0	0
x_3	**6**	4	1	0	0	0	24
x_4	1	2	0	1	0	0	6
x_5	-1	1	0	0	1	0	1
x_6	0	1	0	0	0	1	2

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	-2/3	5/6	0	0	0	20
x_1	1	2/3	1/6	0	0	0	4
x_4	0	**4/3**	-1/6	1	0	0	2
x_5	0	5/3	1/6	0	1	0	5
x_6	0	1	0	0	0	1	2

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	0	3/4	1/2	0	0	21
x_1	1	0	1/4	-1/2	0	0	3
x_2	0	1	-1/8	3/4	0	0	3/2
x_5	0	0	3/8	-5/4	1	0	5/2
x_6	0	0	1/8	-3/4	0	1	1/2

The bold terms in the tableau represent the pivot elements at each iteration, i.e., the intersection of the pivot column and row. From the last tableau, we see that the optimal solution is $x^* = (3, 3/2)$. Notice that we applied a change in signs in the objective function coefficients, turning it into a minimisation problem; also notice that this makes the values of the objective function in the RHS column appear as positive, although it should be negative as we are in fact minimising $-5x_1 - 4x_2$, for which $x^* = (3, 3/2)$ evaluates as -21. As we have seen, the tableau shows $-c_B x_B$, hence why the optimal tableau displays 21 in the first row of the RHS column.

4.2.4 Generating initial feasible solutions

We now consider the issue of converting general linear programming problems into the standard form they are assumed to be for the simplex method. As we mentioned before, problems with constraints of the form $Ax \leq b$ can be converted by simply adding nonnegative *slack variables* $s \geq 0$. In addition, we can trivially obtain an initial basic feasible solution (BFS) with $(x, s) = (0, b)$, with $B = I$, as

$$Ax \leq b \Rightarrow Ax + s = b.$$

Notice that this is equivalent to assuming all original problem variables (i.e., those that are not slack variables) to be initialised as zero (i.e., nonbasic) since this is a trivially available initial feasible solution. However, this method does not work for constraints of the form $Ax \geq b$, as in this case, the transformation would take the form

$$Ax \geq b \Rightarrow Ax - s = b \Rightarrow Ax - s + y = b.$$

Notice that making the respective slack variable basic would yield an initial value of $-b$ (recall that $b \geq 0$ can be assumed without loss of generality), making the basic solution not feasible.

For more general problems, however, this might not be possible since simply setting the original problem variables to zero might not yield a feasible solution that can be used as a BFS. To circumvent that, we rely on *artificial variables* to obtain a BFS.

Let $P : \text{min. } \{c^\top x : Ax = b, x \geq 0\}$, which can be achieved with appropriate transformation (i.e., adding nonnegative slack to the inequality constraints) and assumed (without loss of generality) to have $b \geq 0$. Then, finding a BFS for P amounts to finding a zero-valued optimal solution to the *auxiliary problem*

$$\begin{aligned} (AUX) : \text{min. } & \sum_{i=1}^{m} y_i \\ \text{s.t.: } & Ax + y = b \\ & x, y \geq 0. \end{aligned}$$

The auxiliary problem AUX is formed by including one artificial variable for each constraint in P, represented by the vector y of so-called *artificial variables*. Notice that the problem is represented in a somewhat compact notation, in which we assume that all slack variables used to convert inequalities into equalities have already been incorporated in the vector x and matrix A, with the artificial variables y playing the role of "slacks" in AUX that can be assumed to be basic and trivially yield an initial BFS for AUX. In principle, one does not need artificial variables for the rows in which there is a positive signed slack variable (i.e., an originally less-or-equal-than constraint), but this representation allows for compactness.

Solving AUX to optimality consists of trying to find a BFS in which the value of the artificial variables is zero since, in practice, the value of the artificial variables measures a degree of infeasibility of the current basis in the context of the original problem P. This means that a BFS in which the artificial variable plays no roles was found and can be used as an initial BFS for solving P. On the other hand, if the optimal for AUX is such that some of the artificial variables are nonzero, then this implies that there is no BFS for AUX in which the artificial variables are all zero, or, more specifically, there is no BFS for P, indicating that the problem P is *infeasible*.

Assuming that P is feasible and $\overline{y} = 0$, two scenarios can arise. The first is when the optimal basis B for AUX is composed only of columns A_j of the original matrix A, with no columns associated with the artificial variables. Then B can be used as an initial starting basis without any issues.

The second scenario is somewhat more complicated. Often, AUX is a degenerate problem and the optimal basis B may contain some of the artificial variables y. This then requires an additional preprocessing step, which consists of the following:

(1) Let $A_{B(1)}, \ldots, A_{B(k)}$ be the columns A in B, which are linearly independent. We know that from earlier (c.f. Theorem 2.4) that, being A full-rank, we can choose additional columns $A_{B(k+1)}, \ldots, A_{B(m)}$ that will span $\mathbb{R}^m$.

(2) Select the l^{th} artificial variable $y_l = 0$ and select a component j in the l^{th} row with nonzero $B^{-1}A_j$ and use elementary row operations to include A_j in the basis. Repeat this $m - k$ times.

The procedure is based on several ideas we have seen before. Since $\sum_{i=1}^{m} y_i$ is zero at the optimal, there must be a BFS in which the artificial variables are nonbasic (which is what (1) is referring to). Thus, step (2) can be repeated until a basis B is formed and includes none of the artificial variables.

Some interesting points are worth highlighting. First, notice that $B^{-1}A_{B(i)} = e_i$, $i = 1, \ldots, k$. Since $k < l$, the l^{th} component of each of these vectors is zero and will remain so after performing the elementary row operations. In turn, the l^{th} entry of $B^{-1}A_j$ is not zero, and thus A_j is linearly independent to $A_{B(1)}, \ldots, A_{B(k)}$.

However, it might be so that we find zero elements in the l^{th} row. Let g be the l^{th} row of $B^{-1}A$ (i.e., the entries in the tableau associated with the original problem variables). If g is the null vector, then g_l is zero, and the procedure fails. However, note that $g^\top A = 0 = g^\top Ax = g^\top b$, implying that $g^\top Ax = g^\top b$ is *redundant* can be removed altogether.

This process of generating initial BFS is often referred to as *Phase I* of the two-phase simplex method. *Phase II* consists of employing the simplex method as we developed it, utilising the BFS found in Phase I as a starting basis.

4.3 Column geometry of the simplex method

Let us try to develop a geometrical intuition on why it is so that the simplex method is remarkably efficient in practice. As we have seen in Theorem 4.2, although the simplex method is guaranteed to converge, the total number of steps the algorithm might need to take before convergence grows exponentially with the number of variables and constraints, since the number of steps depends on the number of vertices of the polyhedral set that represents the feasible region of the problem.

However, it turns out that, in practice, the simplex method typically requires $O(m)$ iterations (recall that m is the number of rows in the matrix A), being one of the reasons why it has experienced tremendous success and is by far one of the most mature and reliable methods when it comes to optimisation.

To develop a geometrical intuition on why this is the case, let us first consider an equivalently reformulated problem P:

$$\begin{aligned} P: \text{min. } & z \\ \text{s.t.: } & Ax = b \\ & c^\top x = z \\ & \sum_{j=1}^{n} x_j = 1 \\ & x \geq 0. \end{aligned}$$

In this reformulation, we make the objective function an auxiliary variable, so it can be easily represented on a real line at the expense of adding an additional constraint $c^\top x = z$. Furthermore, we normalise the decision variables so they add to one (notice that this implies a bounded feasible

set). Notice that problem P can be equivalently represented as

$$\begin{aligned} P: \min.\ & z \\ \text{s.t.:}\ & x_1 \begin{bmatrix} A_1 \\ c_1 \end{bmatrix} + x_2 \begin{bmatrix} A_2 \\ c_2 \end{bmatrix} + \cdots + x_n \begin{bmatrix} A_n \\ c_n \end{bmatrix} = \begin{bmatrix} b \\ z \end{bmatrix} \\ & \sum_{j=1}^{n} x_j = 1 \\ & x \geq 0. \end{aligned}$$

This second formulation exposes one interesting interpretation of the problem. Solving P is akin to finding a set of weights x that makes a convex combination (c.f. Definition 2.7) of the columns of A such that it constructs (or matches) b in a way that the resulting combination of the respective components of the vector c is minimised. Now, let us define some nomenclature that will be useful in what follows.

Definition 4.4 (k-dimensional simplex)**.** *A collection of vectors $y_1, \ldots, y_{k+1}$ are affinely independent if $k \leq n$ and $y_1 - y_{k+1}, \ldots, y_k - y_{k+1}$ are linearly independent. The convex hull of $k+1$ affinely independent vectors is a k-dimensional simplex.*

Definition 4.4 is precisely the inspiration for the name of the simplex method. We know that only $m+1$ components of x will be different than zero since that is the number of constraints we have and, thus, the size of a basis in this case. Thus, a BFS is formed by $m+1$ $(A_i, 1)$ columns, which in turn are associated with (A_i, c_i) basic points.

Figure 4.2 provides an illustration of the concept. In this, we have that $m = 2$, so each column A_j can be represented in a two-dimensional plane. Notice that a basis requires three points (A_i, c_i) and forms a 3-simplex. A BFS is a selection of three points (A_i, c_i) such that b, also illustrated in the picture, can be formed by a convex combination of the (A_i, c_i) forming the basis. This will be possible if b happens to be inside the 3-simplex formed by these points. For example, in Figure 4.2, the basis formed by columns $\{2, 3, 4\}$ is a BFS, while the basis $\{1, 2, 3\}$ is not.

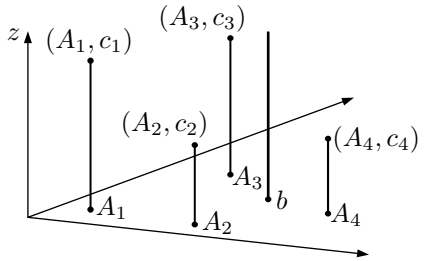


Figure 4.2: A solution x is a convex combinations of (A_i, c_i) such that $Ax = b$.

We now can add a third dimension to the analysis representing the value of z. For that, we will use Figure 4.3. As can be seen, each selection of basis creates a tilt in the three-dimensional simplex, such that the point b is met precisely at the high corresponding to its value in the z axis. This allows to compare bases according to their objective function value. And, since we are minimising, we would like to find the basis that has its respective simplex crossing b at the lowermost point.

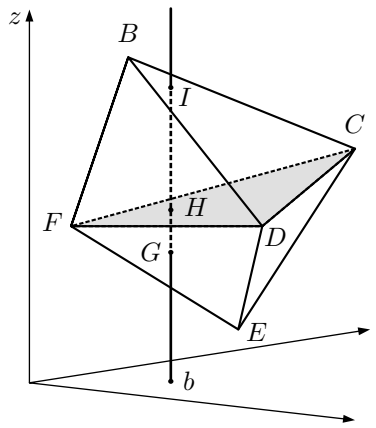


Figure 4.3: A solution x is a convex combinations of (A_i, c_i) such that $Ax = b$.

Notice that in Figure 4.3, although each facet is a basic simplex, only three are feasible (BCD, CDF, and DEF). We can also see what one iteration of the simplex method does under this geometrical interpretation. Moving between adjacent basis means that we are replacing one vertex (say, C) with another (say, E) considering the potential for decrease in value in the z axis (represented by the difference between points H and G onto the z axis). You can also see the notion of pivoting: since we are moving between adjacent bases, two successive simplexes share an edge in common, consequently, they pivot around that edge (think about the movement of the edge C moving to the point E while the edge $\overline{DF}$ remains fixed).

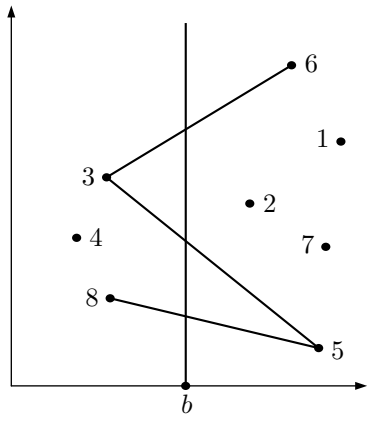


Figure 4.4: Pivots from initial basis $[A_3, A_6]$ to $[A_3, A_5]$ and to the optimal basis $[A_8, A_5]$

Now we are ready to provide an insight into why the simplex method is often so efficient. The main reason is associated with the ability that the method possesses of skipping bases in favour of those with most promising improvement. To see that, consider Figure 4.4, which is a 2-dimensional schematic projection of Figure 4.3. By using the reduced costs to guide the choice of the next basis, we tend to choose the steepest of the simplexes that can provide reductions in the objective function value, which has the side effect of allowing for skipping several basis that would have to be otherwise considered. This creates a "sweeping effect", in which allows the method to find optimal

solutions in fewer pivots than vertices. Clearly, this can be engineered to be prevented, as there are examples purposely constructed to force the method to consider every single vertex, but the situation illustrated in Figure 4.4 is by far the most common in practice.

4.4 Exercises

Exercise 4.1: Properties of the simplex algorithms

Consider the simplex method applied to a standard form minimization problem, and assume that the rows of the matrix A are linearly independent. For each of the statements that follow, give either a proof or a counter example.

(a) An iteration of the simplex method might change the feasible solution while leaving the cost unchanged.

(b) A variable that has just entered the basis cannot leave in the very next iteration.

(c) If there is a non-degenerate optimal basis, then there exists a unique optimal basis.

Exercise 4.2: The simplex method

Consider the problem

$$\begin{aligned} \text{max. } & 40x_1 + 60x_2 \\ \text{s.t.: } & 2x_1 + x_2 \leq 7 \\ & x_1 + x_2 \leq 4 \\ & x_1 + 3x_2 \leq 9 \\ & x_1, x_2 \geq 0. \end{aligned}$$

A feasible point for this problem is $(x_1, x_2) = (0, 3)$. Formulate the problem as a minimisation problem in standard form and verify whether or not this point is optimal. If not, solve the problem by using the simplex method.

Exercise 4.3: Solving a tableau

Consider a linear programming problem in standard form, described in terms of the following tableau:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
z	0	0	0	δ	3	γ	ξ	0
x_2	0	1	0	α	1	0	3	β
x_3	0	0	1	-2	2	η	-1	2
x_1	1	0	0	0	-1	2	1	3

The entries α, β, γ, δ, η and ξ in the tableau are unknown parameters. Furthermore, let B be the basis matrix corresponding to having x_2, x_3, and x_1 (in that order) be the basic variables. For each one of the following statements, find the ranges of values of the various parameters that will make the statement to be true.

(a) Phase II of the Simplex method can be applied using this as an initial tableau.

(b) The corresponding basic solution is feasible, but we do not have an optimal basis.

(c) The corresponding basic solution is feasible and the first Simplex iteration indicates that the optimal cost is $-\infty$.

(d) The corresponding basic solution is feasible, x_6 is a candidate for entering the basis, and when x_6 is the entering variable, x_3 leaves the basis.

(e) The corresponding basic solution is feasible, x_7 is a candidate for entering the basis, but if it does, the objective value remains unchanged.

Exercise 4.4: Two-phase simplex method

Solve the problem below using the two-phase simplex method. What is your conclusion about the feasibility of the problem? Verify your results by drawing the feasible region.

$$\begin{aligned} \text{max. } & 5x_1 + x_2 \\ \text{s.t.: } & 2x_1 + x_2 \geq 5 \\ & x_2 \geq 1 \\ & 2x_1 + 3x_2 \leq 12 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Chapter 5

Linear Programming Duality - Part I

5.1 Formulating duals

In this chapter, we will discuss the notion of duality in the context of linear programming problems. Duality can be understood as a toolbox of technical results that makes available a collection of techniques and features that can be exploited to both further understand characteristics of the optimal solution and to devise specialised methods for solving large-scale linear programming problems.

5.1.1 Motivation

Let us define the notation we will be using throughout the next chapters. As before, let $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and P be the standard form linear programming problem

$$\begin{aligned} (P): \text{min.} \quad & c^\top x \\ \text{s.t.:} \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

which we will refer to as the *primal* problem. In mathematical programming, we say that a constraint has been *relaxed* if it has been removed from the set of constraints. With that in mind, let us consider a *relaxed* version of P, where $Ax = b$ is replaced with a *violation penalty* term $p^\top(b - Ax)$. This lead to the following problem:

$$g(p) = \min_{x \geq 0} \left\{ c^\top x + p^\top (b - Ax) \right\},$$

which has the benefit of not having equality constraints explicitly represented, but only implicit by means of a penalty term. This term is used to penalise the infeasibility of the constraints in the attempt to steer the solution of the relaxed problem towards the solution to P. Recalling that our main objective is to solve P, we are interested in the values (or prices, as they are often called) for $p \in \mathbb{R}^m$ that make P and $g(p)$ equivalent.

Let $\overline{x}$ be the optimal solution to P. Notice that, for any $p \in \mathbb{R}^m$, we have that

$$g(p) = \min_{x \geq 0} \left\{ c^\top x + p^\top (b - Ax) \right\} \leq c^\top \overline{x} + p^\top (b - A\overline{x}) = c^\top \overline{x},$$

i.e., $g(p)$ is a *lower bound* on the optimal value $c^\top \overline{x}$. The lefthand-side inequality holds because, although $\overline{x}$ is optimal for P, it might not be optimal for $g(p)$ for an arbitrary vector p. The second inequality is a consequence of $\overline{x} \in P$, i.e., the feasibility of $\overline{x}$ implies that $A\overline{x} = b$.

We can use an optimisation-based approach to try to find an optimal lower bound, i.e., the tightest possible lower bound for P. This can be achieved by solving the *dual problem* D formulated as

$$(D) : \max_{p}. \; g(p).$$

Notice that D is an unconstrained problem to which a solution proves the *tightest* lower bound on P (say, at $\overline{p}$). Also, notice how the function $g(p) : \mathbb{R}^m \mapsto \mathbb{R}$ has embedded on its evaluation the solution of a linear programming problem with $x \in \mathbb{R}^n$ as decision variables for a fixed p, which is the argument given to the function g. This is a new concept at this point that often is a source of confusion.

We will proceed in this chapter developing the analytical framework that allows us to pose the key result in duality theory, which is that stating that

$$g(\overline{p}) = c^\top \overline{x}.$$

That is, we will next develop the results that guarantee the equivalence between primal and dual representations. This will be useful for interpreting properties associated with the optimal primal solution $\overline{x}$ from the associated optimal prices $\overline{p}$. Furthermore, we will see in later chapters that linear programming duality can be used as a framework for replacing constraints with equivalent representations, which is a useful procedure in many settings, including for developing alternative solution strategies also based in linear programming.

5.1.2 General form of duals

Now, let us focus on developing a formulation for dual problems that is based on linear programming as well. Using the definition of D, we notice that

$$\begin{aligned} g(p) = & \min_{x \geq 0}. \; \left\{ c^\top x + p^\top (b - Ax) \right\} \\ = & \; p^\top b + \min_{x \geq 0}. \; \left\{ c^\top x - p^\top Ax \right\} \\ = & \; p^\top b + \min_{x \geq 0}. \; \left\{ (c^\top - p^\top A)x \right\}. \end{aligned}$$

As $x \geq 0$, the rightmost problem can only be bounded if $(c^\top - p^\top A) \geq 0$. This gives us a linear constraint that can be used to enforce the existence of a solution for

$$\min_{x \geq 0}. \; \left\{ (c^\top - p^\top A)x \right\}.$$

With that in mind, we can equivalently reformulate D as

$$\begin{aligned} (D) : \max. \; & p^\top b \\ \text{s.t.:} \; & p^\top A \leq c^\top. \end{aligned}$$

Notice that D is a linear programming problem with m variables (one per constraint of the primal problem P) and n constraints (one per variable of P). As you might suspect, if you were to repeat the analysis, looking at D as the "primal" problem, you would end with a dual that is exactly P. For this to become more apparent, let us first define more generally the rules that dictate what kind of dual formulations are obtained for different types of primal problems in terms of their original (i.e., not in standard) form.

In the more general case, let P be defined as

$$\begin{aligned}(P): \text{min.} \quad & c^\top x \\ \text{s.t.:} \quad & Ax \geq b.\end{aligned}$$

Notice that the problem P can be equivalently reformulated as

$$\begin{aligned}(P): \text{min.} \quad & c^\top x \\ \text{s.t.:} \quad & Ax - s = b \\ & s \geq 0.\end{aligned}$$

Let us focus on the constraints in the reformulated version of P, which can be written as

$$[A \mid -I] \begin{bmatrix} x \\ s \end{bmatrix} = b.$$

We will apply the same procedure as before, being our constraint matrix $[A \mid -I]$ in place of A and $[x \mid s]^\top$ our vector of variables, in place of x. Using analogous arguments, we now require that $c^\top - p^\top A = 0$, so $g(p)$ is finite. Notice that this is a slight deviation from before, but in this case, we have that $x \in \mathbb{R}^n$, so $c^\top - p^\top A = 0$ is the only condition that allows the inner problem in $g(p)$ to have a finite solution. Then, we obtain the following conditions to be imposed to our dual linear programming formulation

$$\begin{aligned}& p^\top [A \mid -I] \leq [c^\top \mid 0^\top] \\ & c^\top - p^\top A = 0,\end{aligned}$$

Combining them all and redoing the previous steps for obtaining a dual formulation, we arrive at

$$\begin{aligned}(D): \text{max.} \quad & p^\top b \\ \text{s.t.:} \quad & p^\top A = c^\top \\ & p \geq 0.\end{aligned}$$

Notice how the change in the type of constraints in the primal problem P lead to additional nonnegative constraints in the dual variables p. Similarly, the absence of explicit nonnegativity constraints in the primal variables x lead to equality constraints in the dual problem D, as opposed to inequalities.

Table 5.1 provides a summary which allows one to identify the resulting formulation of the dual problem based on the primal formulation, in particular regarding its type (minimisation or maximisation), constraint types and variable domains.

For converting a minimisation primal problem into a (maximisation) dual, one must read the table from left to right. That is, the independent terms (b) become the objective function coefficients, greater or equal constraints become nonnegative variables, and so forth. However, if the primal problem is a maximisation problem, the table must be read from right to left. For example, in this case, less-or-equal-than constraints would become nonnegative variables instead, and so forth. It takes a little practice to familiarise yourself with this table, but it is a really useful resource to obtain dual formulations from primal problems.

One remark to be made at this point is that, as is hopefully clearer now, the conversion of primal problems into duals is symmetric, meaning that reapplying the rules in Table 5.1 would take you

Primal (dual)	**Dual (primal)**
minimise	maximise
Independent terms	Obj. function coef.
Obj. function coef.	Independent terms
i-th row of constraint coef.	i-th column of constraint coef.
i-th column of constraint coef.	i-th row of constraint coef.
Constraints	**Variables**
$\geq$	≥ 0
$\leq$	≤ 0
$=$	$\in \mathbb{R}$
Variables	**Constraints**
≥ 0	$\leq$
≤ 0	$\geq$
$\in \mathbb{R}$	$=$

Table 5.1: Primal-dual conversion table

from the obtained dual back to the original primal. This is a property of linear programming problems called being *self dual*. Another remark is that equivalent reformulations made in the primal lead to equivalent duals. Specifically, transformations that replace variables $x \in \mathbb{R}$ with $x^+ - x^-$, where $x^+, x^- \geq 0$, introduce nonnegative slack variables, or remove redundant constraints all lead to equivalent duals.

For example, recall that the dual formulation for the primal problem

$$\begin{aligned} (P): \min.\ & c^\top x \\ \text{s.t.:}\ & Ax \geq b \\ & x \in \mathbb{R}^n \end{aligned}$$

is given by

$$\begin{aligned} (D): \max.\ & p^\top b \\ \text{s.t.:}\ & p \geq 0 \\ & p^\top A = c^\top. \end{aligned}$$

Now suppose we equivalently reformulate the primal problem to become

$$\begin{aligned} (P'): \min.\ & c^\top x + 0^\top s \\ \text{s.t.:}\ & Ax - s = b \\ & x \in \mathbb{R}^n, s \geq 0. \end{aligned}$$

Then, using Table 5.1, we would obtain the following dual formulation, which is equivalent to D

$$\begin{aligned} (D'): \max.\ & p^\top b \\ \text{s.t.:}\ & p \in \mathbb{R}^m \\ & p^\top A = c^\top \\ & -p \leq 0. \end{aligned}$$

Analogously, suppose we were to equivalently reformulate P as

$$\begin{aligned}(P'') : \text{min. } & c^\top x^+ - c^\top x^- \\ \text{s.t.: } & Ax^+ - Ax^- \geq b \\ & x^+ \geq 0, x^- \geq 0.\end{aligned}$$

Then, the dual formulation for P'' would be

$$\begin{aligned}(D'') : \text{max. } & p^\top b \\ \text{s.t.: } & p \geq 0 \\ & p^\top A \leq c \\ & -p^\top A \leq -c^\top,\end{aligned}$$

which is also equivalent to D.

5.2 Duality theory

We will now develop the technical results associated with duality that will be the kingpin for its use as a framework for devising solution methods and interpreting optimal solution properties.

5.2.1 Weak duality

Weak duality is the property associated with the bounding nature of dual feasible solutions. This is stated in Theorem 5.1.

Theorem 5.1 (Weak duality)**.** *Let x be a feasible solution to $(P) : min.\ \left\{c^\top x : Ax = b, x \geq 0\right\}$ and p be a feasible solution to $(D) : max.\ \left\{p^\top b : p^\top A \leq c^\top\right\}$, the dual problem of P. Then $c^\top x \geq p^\top b$.*

Proof. Let $I = \{i\}_{i=1}^m$ and $J = \{j\}_{j=1}^n$. For any x and p, define

$$u_i = p_i(a_i^\top x - b_i) \text{ and } v_j = (c_j - p^\top A_j)x_j.$$

Notice that $u_i \geq 0$ for $i \in I$ and $v_j \geq 0$ for $j \in J$, since each pair of terms will have the same sign (you can see that from Table 5.1 and assuming x_j to be the dual variable associated with $p^\top A \leq c^\top$). Thus, we have that

$$0 \leq \sum_{i \in I} u_i + \sum_{j \in J} v_j = [p^\top Ax - p^\top b] + [c^\top x - p^\top Ax] = c^\top x - p^\top b. \qquad \square$$

Let us also pose some results that are direct consequences of Theorem 5.1, which are summarised in Corollary 5.2.

Corollary 5.2 (Consequences of weak duality)**.** *The following are immediate consequences of Theorem 5.1:*

(1) If the optimal value of P is $-\infty$ (i.e., P is unbounded), then D must be infeasible;

(2) if the optimal value of D is ∞ (i.e., D is unbounded), then P must be infeasible;

(3) let x and p be feasible to P and D, respectively. Suppose that $p^\top b = c^\top x$. Then x is optimal to P and p is optimal to D.

Proof. By contradiction, suppose that P has optimal value $-\infty$ and that D has a feasible solution p. By weak duality, $p^\top b \leq c^\top x = -\infty$, i.e., a contradiction. Part (2) follows a symmetric argument.

Part (3): let $\overline{x}$ be an alternative feasible solution to P. From weak duality, we have $c^\top \overline{x} \geq p^\top b = c^\top x$, which proves the optimality of x. The optimality of p follows a symmetric argument. □

Notice that Theorem 5.1 provides us with a bounding technique for any linear programming problem. That is, for a given pair of primal and dual feasible solutions, $\overline{x}$ and $\overline{p}$, respectively, we have that

$$\overline{p}^\top b \leq c^\top x^* \leq c^\top \overline{x},$$

where $c^\top x^*$ is the optimal objective function value.

Corollary 5.2 also provides an alternative way of identifying infeasibility by means of linear programming duals. One can always use the unboundedness of a given element of a primal-dual pair to state the infeasibility of the other element in the pair. That is, an unbounded dual (primal) implies an infeasible primal (dual). However, the reverse statement is not as conclusive. Specifically, an infeasible primal (dual) does not necessarily imply that the dual (primal) is unbounded, but does imply it to be *either* infeasible or unbounded.

5.2.2 Strong duality

This bounding property can also be used as a certificate of optimality, in case they match in value. This is precisely the notion of *strong duality*, a property that is inherent to linear programming problems. This is stated in Theorem 5.3.

Theorem 5.3 (Strong duality)**.** *If $(P) : min. \left\{c^\top x : Ax = b, x \geq 0\right\}$ has an optimal solution, so does its dual $(D) : max. \left\{p^\top b : p^\top A \leq c^\top\right\}$ and their respective optimal values are equal.*

Proof. Assume P is solved to optimality, with optimal solution x and basis B. Let $x_B = B^{-1}b$. At the optimal, the reduced costs are $c^\top - c_B^\top B^{-1}A \geq 0$. Let $p = c_B^\top B^{-1}$. We then have $p^\top A \leq c^\top$, which shows that p is feasible to D. Moreover,

$$p^\top b = c_B^\top B^{-1} b = c_B^\top x_B = c^\top x, \tag{5.1}$$

which, in turn, implies the optimality of p (cf. Corollary 5.2 (3)). □

The proof of Theorem 5.3 reveals something remarkable relating the simplex method and the dual variables p. As can be seen in (5.1), for any primal feasible solution x, an associated dual (not necessarily) feasible solution p can be immediately recovered. If the associated dual solution is also feasible, then Theorem 5.3 guarantees that optimality ensues.

This means that we can interchangeably solve either a primal or a dual form of a given problem, considering aspects related to convenience and computational ease. This is particularly useful in the context of the simplex method since the prices p are readily available as the algorithm progresses. In the next section, we will discuss several practical uses of this relationship in more detail. For now, let us halt this discussion for a moment and consider a geometrical interpretation of duality in the context of linear programming.

5.3 Geometric interpretation of duality

Linear programming duality has an interesting geometric interpretation that stems from the much more general framework of Lagrangian duality (of which linear programming duality is a special case) and its connection to optimality conditions, topics that will be further explored in Part II. For now, let us focus on how linear programming duality can be interpreted in the context of "balancing forces".

First, let $\overline{x}$ be the optimal solution of primal problem P in the form

$$\begin{aligned}(P): \min.\ & c^\top x \\ \text{s.t.: } & a_i^\top x \geq b_i,\ \forall i \in I.\end{aligned}$$

Imagine that there is a particle within the polyhedral set representing the feasible region of P and that this particle is subjected to a force represented by the vector $-c$. Notice that this is equivalent to minimising the function $z = c^\top x$ within the polyhedral set $\left\{a_i^\top x \geq b_i\right\}_{i\in I}$ representing the feasible set of P. Assuming that the feasible set of P is bounded in the direction $-c$, this particle will eventually come to a halt after hitting the "walls" of the feasible set, at a point where the pulling force $-c$ and the reaction of these walls reach an equilibrium. We can think of $\overline{x}$ as this stopping point. This is illustrated in Figure 5.1.

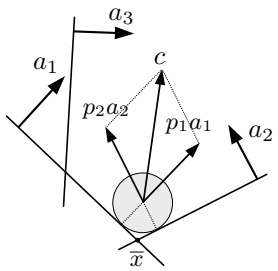


Figure 5.1: A geometric representation of duality for linear programming problems

We can then think of the dual variables p as the multipliers applied to the normal vectors associated with the hyperplanes (i.e., the walls) that are in contact with the particle to achieve this equilibrium. Hence, these multipliers p will be such that

$$c = \sum_{i\in I} p_i a_i,\ \text{for some } p_i \geq 0, i \in I,$$

which is precisely the dual feasibility condition (i.e., constraint) associated with the dual of P, given by

$$D: \max.\ \left\{p^\top b : p^\top A = c, p \geq 0\right\}.$$

And, dual feasibility, as we seen before, implies the optimality of $\overline{x}$.

5.3.1 Complementary slackness

One point that must be noticed is that, for the constraints that are not active at the optimal point $\overline{x}$ (i.e., the walls that are not exerting resistance to the particle at the equilibrium point),

the multipliers p must be set to zero. That is, we have that

$$p^\top b = \sum_{i \in I} p_i b_i = \sum_{i \in I} p_i (a_i^\top \overline{x}) = c^\top \overline{x},$$

which again implies the optimality of p (cf. Corollary 5.2 (3)). This geometrical insight leads to another key result for linear programming duality, which is the notion of *complementary slackness*.

Theorem 5.4 (Complementary slackness)**.** *Let x be a feasible solution for*

$$(P) : \textit{min.} \ \left\{ c^\top x : Ax = b, x \geq 0 \right\}$$

and p be a feasible solution for

$$(D) : \textit{max.} \ \left\{ p^\top b : p^\top A \leq c^\top \right\}.$$

The vectors x and p are optimal solutions to P and D, respectively, if and only if $p_i(a_i^\top x - b_i) = 0, \forall i \in I$, and $(c_j - p^\top A_j)x_j = 0$, $\forall j \in J$.

Proof. From the proof of Theorem 5.1 and with Theorem 5.3 holding, we have that

$$p_i(a_i^\top x - b_i) = 0, \forall i \in I, \text{ and } (c_j - p^\top A_j)x_j = 0, \ \forall j \in J.$$

In turn, if these hold, then x and p are optimal (cf. Corollary 5.2 (3)). □

For nondegenerate basic feasible solutions (BFS) (i.e., $x_j > 0$, $\forall j \in I_B$, where I_B is the set of basic variable indices), complementary slackness determines a *unique* dual solution. That is

$$(c_j - p^\top A_j)x_j = 0, \text{ which yields } c_j = p^\top A_j, \ \forall j \in I_B,$$

which has a unique solution $p^\top = c_B^\top B^{-1}$, as the columns A_j of be are assumed to be linearly independent. In the presence of degeneracy, this is not the case anymore, typically implying that a degenerate optimal BFS will have multiple associated feasible dual variables.

5.3.2 Dual feasibility and optimality

Combining what we have seen so far, the conditions for a *primal-dual pair* (x, p) to be optimal to their respective primal (P) and dual (D) problems are given by

$$a_i^\top x \geq b_i, \ \forall i \in I \qquad \text{(primal feasibility)} \tag{5.2}$$

$$p_i = 0, \ \forall i \notin I^0 \qquad \text{(complementary conditions)} \tag{5.3}$$

$$\sum_{i \in I} p_i^\top a_i = c \qquad \text{(dual feasibility I)} \tag{5.4}$$

$$p_i \geq 0, \qquad \text{(dual feasibility II)} \tag{5.5}$$

where $I^0 = \left\{ i \in I, a_i^\top x = b_i \right\}$ are the active constraints. From (5.2)–(5.5), we see that the optimality of the primal-dual pair has two main requirements. The first is that x must be (primal) feasible. The second, expressed as

$$\sum_{i \in I^0} p_i a_i = c, \ p_i \geq 0,$$

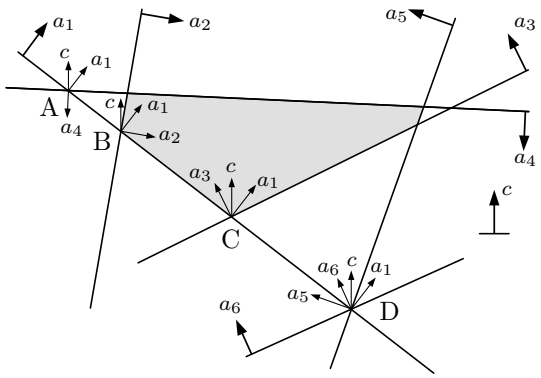


Figure 5.2: A is both primal and dual infeasible; B is primal feasible and dual infeasible; C is primal and dual feasible; D is degenerate.

is equivalent to requiring c to be expressed as a nonnegative linear combination (also known as a conic combination) of the active constraints. This has a nice geometrical interpretation: dual feasibility can be interpreted as having the vector c inside the "cone" formed by the normal vectors of the active constraints, which in turn is a necessary condition for the existence of an equilibrium, as described in Figure 5.1. Figure 5.2 illustrates this fact.

Notice how in Figure 5.2 neither points A or B are dual feasible, while C represents a dual feasible point, being thus the optimal for the problem depicted. One interesting point to notice is D. Although not feasible, it allows us to see an important effect that degeneracy may cause. Assume for a moment that D is feasible. Then, dual feasibility becomes dependent on the basis representing the vertex. That is, while the bases $I_B = \{1, 5\}$ and $I_B = \{1, 6\}$ are dual feasible, the basis $I_B = \{5, 6\}$ is not. As we will see, just as it is the case with the simplex method, the dual simplex method, which we will discuss in the next section, might be subject to stalling and cycling from the presence of primal degeneracy, which in turn may also leads to multiple dual feasible (primal optimal) solutions.

5.4 Practical uses of duality

We now consider practical uses for the properties we have discussed in the previous section. The most direct application of duality in linear programming problems is the interpretation of dual variable values as marginal values associated with constraints, with important economical implications.

Another important employment of duality consists of the development of an alternative variant of the simplex method, the *dual simplex method*, which is particularly useful in several contexts.

5.4.1 Optimal dual variables as marginal costs

The optimal dual variable values associated with an optimal BFS have an important practical interpretation. To see that, let $\overline{x}$ be a nondegenerate optimal BFS with corresponding basis B. Thus, $\overline{x}_B = B^{-1}b > 0$.

Now, assume that we cause a marginal perturbation on the vector b, represented by a vector d.

That is, assume that we have $B^{-1}(b+d) > 0$, noticing that nondegenerate feasibility holds.

Recall that the optimality condition $\bar{c} = c^\top - c_B^\top B^{-1}A \geq 0$ is not influenced by such a marginal perturbation. That is, for a small change d, the optimal basis (i.e., the selection of basic variables) is not disturbed. On the other hand, the optimal value of the basic variables is, and consequently, so is the optimal value, which becomes

$$c_B^\top B^{-1}(b+d) = p^\top(b+d).$$

Notice that $p^\top = c_B^\top B^{-1}$ is optimal for the (respective) dual problem. Thus, a change d causes a change of $p^\top d$ in the optimal value, meaning that the components p_i represent a *marginal value/cost* associated with the independent term b_i, for $i \in I$.

This has important implications in practice, as it allows for pricing the values of the resources associated with constraints. For example, suppose the dual value (or price) p_i is associated with a resource whose requirement is given by b_i. In that case, any opportunity to remove units of b_i for less than p_i should be seized since it costs p_i to satisfy any additional unit in the requirement b_i. A similar interpretation can be made in the context of less-or-equal-than constraints, in which p_i would indicate benefits (or losses, if $p_i > 0$) in increasing the availability of b_i.

For a numerical example, let us consider our paint factory problem once again.

The following tableau represents the optimal solution

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	0	3/4	1/2	0	0	21
x_1	1	0	1/4	-1/2	0	0	3
x_2	0	1	-1/8	3/4	0	0	3/2
x_5	0	0	3/8	-5/4	1	0	5/2
x_6	0	0	1/8	-3/4	0	1	1/2

where x_3 and x_4 were the slack variables associated with raw material M1 and M2, respectively. In this case, we have that

$$B^{-1} = \begin{bmatrix} 1/4 & -1/2 & 0 & 0 \\ -1/8 & 3/4 & 0 & 0 \\ 3/8 & -5/4 & 1 & 0 \\ 1/8 & -3/4 & 0 & 1 \end{bmatrix} \text{ and } p = c_B^\top B^{-1} = \begin{bmatrix} -5 \\ -4 \\ 0 \\ 0 \end{bmatrix}^\top \begin{bmatrix} 1/4 & -1/2 & 0 & 0 \\ -1/8 & 3/4 & 0 & 0 \\ 3/8 & -5/4 & 1 & 0 \\ 1/8 & -3/4 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -3/4 \\ -1/2 \\ 0 \\ 0 \end{bmatrix}$$

Notice that these are values of the entries in the z-row below the slack variables $x_3, \ldots, x_6$, except the minus sign. This is because the z-rows contain the entries for $c - p^\top B^{-1}$ and for all slack variables we have that $c_j = 0$, for $j = 3, \ldots, 6$. Also, recall that the paint factory problem is a maximisation problem, so p represents the decrease in the objective function value. In this, we see that removing one unit of M1 would decrease the objective function by 3/4 and by 1/2 if one unit of M2 is removed. Analogous, increasing M1 or M2 availability by one unit would increase the objective function value by 3/4 and 1/2, respectively.

5.4.2 The dual simplex method

In general, solution methods in mathematical programming can be either *primal methods*, in which primal feasibility of an initial solution is maintained while seeking for dual feasibility (i.e., primal

optimality); or *dual methods*, where dual feasibility is maintained while seeking for primal feasibility (i.e., dual optimality).

As we have seen in Chapter 4, the original (or primal) simplex method iterated from an initial basic feasible solution (BFS) until the optimality condition

$$\overline{c} = c^\top - c_B B^{-1} A \geq 0$$

was observed. Notice that this is precisely the dual feasibility condition $p^\top A \leq c$.

Being a dual method, the dual version of the simplex method, or the *dual simplex method*, considers conditions in reverse order. That is, it starts from an initial dual feasible solution and iterates in a manner that the primal feasibility condition $B^{-1}b \geq 0$ is *sought* for, while $\overline{c} \geq 0$, or equivalently, $p^\top A \leq c$, is maintained.

To achieve that, one must revise the pivoting of the primal simplex method such that the variable to leave the basis is some $i \in I_B$, with $x_{B(i)} < 0$, while the variable chosen to enter the basis is some $j \in I_N$, such that $\overline{c} \geq 0$ is maintained.

Consider the l^{th} simplex tableau row for which $x_{B(i)} < 0$ of the form $[v_1, \ldots, v_n, x_{B(i)}]$; i.e., v_j is the l^{th} component of $B^{-1}A_j$.

For each $j \in I_N$ for which $v_j < 0$, we pick

$$j' = \arg\min_{j \in I_N : v_j < 0} \frac{\overline{c}_j}{|v_j|}.$$

Pivoting is performed by employing elemental row operations to replace $A_{B(i)}$ with A_j in the basis. This implies that $\overline{c}_j \geq 0$ is maintained, since

$$\frac{\overline{c}_j}{|v_j|} \geq \frac{\overline{c}_{j'}}{|v_{j'}|} \Rightarrow \overline{c}_j - |v_j|\frac{\overline{c}_{j'}}{|v_{j'}|} \geq 0 \Rightarrow \overline{c}_j + v_j \frac{\overline{c}_{j'}}{|v_{j'}|} \geq 0, \, \forall j \in J.$$

Notice that it also justifies why we must only consider for entering the basis those variables for which $v_j < 0$. Analogously to the case in the primal simplex method, if we observe that $v_j \geq 0$ for all $j \in J$, then no limiting condition is imposed in terms the increase in the nonbasic variable (i.e., an unbounded dual, which, according to Corollary 5.2 (2), implies the original problem is infeasible).

Assuming that the dual is not unbounded, the termination of the dual simplex method is observed when $B^{-1}b \geq 0$ is achieved, and primal-dual optimal solutions have been found, with $x = (x_B, x_N) = (B^{-1}b, 0)$ (i.e., the primal solution) and $p = (p_B, p_N) = (0, c_B^\top B^{-1})$ (dual). Algorithm 3 presents a pseudocode for the dual simplex method.

To clarify some of the previous points, let us consider a numerical example. Consider the problem

$$\begin{aligned} \text{min. } & x_1 + x_2 \\ \text{s.t.: } & x_1 + 2x_2 \geq 2 \\ & x_1 \geq 1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

The first thing we must do is convert the greater-or-equal-than inequalities into less-or-equal-than inequalities and add the respective slack variables. This allows us to avoid the inclusion of artificial variables, which are not required anymore since we can allow for primal infeasibility. This leads to

Algorithm 3 Dual simplex method

initialise. Initial basis B and associated basic solution x.
while $x_B = B^{-1}b < 0$ for some component $i \in I_B$ **do**
    Choose some l for which $x_{B(l)} < 0$. Calculate $u = B^{-1}A_j$.
    if $u \geq 0$ **then**
        return $z = +\infty$.
    else
        Form new basis $B = B \setminus \{l\} \cup \{j'\}$ where $j' = \arg\min_{j \in I_N : u_j < 0} \frac{\overline{c}_j}{|u_j|}$
        Calculate $x_B = B^{-1}b$.
    end if
end while
return optimal basis B and optimal solution x.

the equivalent standard form problem

$$\begin{aligned} \text{min. } & x_1 + x_2 \\ \text{s.t.: } & -x_1 - 2x_2 + x_3 = -2 \\ & -x_1 + x_4 = -1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Below is the sequence of tableaus after applying the dual simplex method to solve the problem. The terms in bold font represent the pivot element (i.e., the intersection between the pivot row and pivot column).

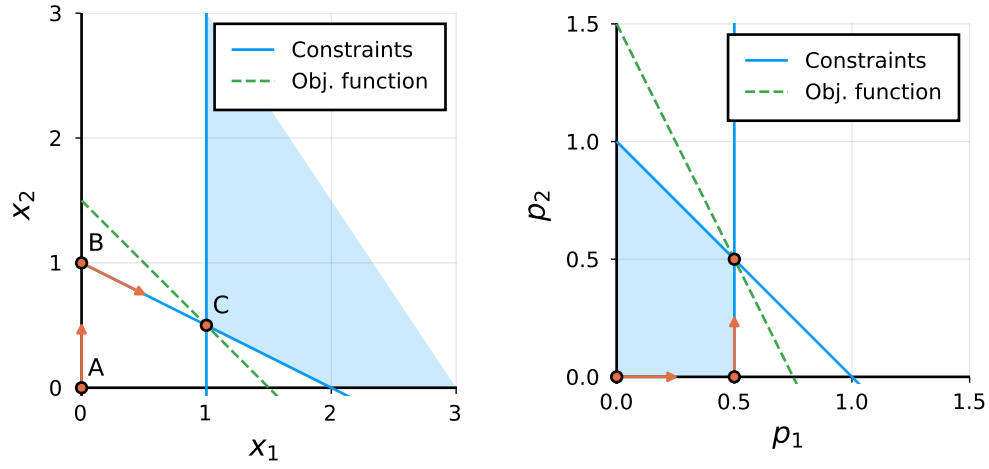
	x_1	x_2	x_3	x_4	RHS
z	1	1	0	0	0
x_3	-1	**-2**	1	0	-2
x_4	-1	0	0	1	-1

	x_1	x_2	x_3	x_4	RHS
z	1/2	0	1/2	0	-1
x_2	1/2	1	-1/2	0	1
x_4	**-1**	0	0	1	-1

	x_1	x_2	x_3	x_4	RHS
z	0	0	1/2	1/2	-3/2
x_2	0	1	-1/2	1/2	1/2
x_1	1	0	0	-1	1

Figure 5.3 illustrates the progress of the algorithm both in the primal (Figure 5.3a) and in the dual (Figure 5.3b) variable space. Notice how in the primal space the solution remains primal infeasible until a primal feasible solution is reached, that being the optimal for the problem. Also, notice that the coordinates of the dual variables can be extracted from the zeroth row of the simplex tableau.

Some interesting features related to the progress of the dual simplex algorithm are worth highlighting. First, notice that the objective function is monotonically increasing in this case, since



(a) The primal-variable space

(b) The dual-variable space

Figure 5.3: The progress of the dual simplex method in the primal and dual space.

$x_{B(l)} \frac{\bar{c}_{j'}}{|v_{j'}|}$ is added to $-c_B B^{-1} b$ and $x_{B(l)} < 0$, meaning that the dual cost increases (recall the convention of having a minus sign so that the zeroth row correctly represent the objective function value, given by the negative of the value displayed in the rightmost column). This illustrates the gradual loss of optimality in the search for (primal) feasibility. For a nondegenerate problem, this can also be used as an argument for eventual convergence since the dual objective value can only increase and is bounded by the primal optimal value. However, in the presence of dual degeneracy, that is, $\bar{c}_j = 0$ for some $j \in I_N$ in the optimal solution, the algorithm can suffer from cycling. As we have seen before, that is an indication that the primal problem has multiple optima.

The dual simplex method is often the best choice of algorithm, because it typically precludes the need for a Phase I type of method as it is often trivial to find initial dual feasible solutions (the origin, for example, is typically dual feasible in minimisation problems with nonnegative coefficients; similar trivial cases are also well known).

Moreover, dual simplex is the algorithm of choice for resolving a linear programming problem when after finding an optimal solution, you modify the feasible region. Turns out that this procedure is in the core of the methods used to solve integer programming problems, as well as in the Benders decomposition, both topics we will explore later on. The dual simplex method is also more successful than its primal counterpart in combinatorial optimisation problems, which are typically plagued with degeneracy. As we have seen, primal degeneracy simply means multiple dual optima, which are far less problematic under an algorithmic standpoint.

Most professional implementations of the simplex method use by default the dual simplex version. This has several computational reasons, in particular related to more effective Phase I and pricing methods for the dual counterpart.

5.5 Exercises

Exercise 5.1: Duality in the transportation problem

Recall Exercise 1.4 in which we solved a capacitated transportation problem. Answer the following questions based on the interpretation of the dual price. The tables with the arc capacities and supply/demand settings are once again presented in Tables 5.2a and 5.2b.

node	p1	p2
s1 / d1	80 / 60	400 / 300
s2 / d2	200 / 100	1500 / 1000
s3 / d3	200 / 200	300 / 500

(a) Supply availability and demand per oil type [in L]

	d1 p1/p2 (cap)	d2 p1/p2 (cap)	d3 p1/p2 (cap)
s1	5/- (∞)	5/18 (300)	-/- (0)
s2	8/15 (300)	9/12 (700)	7/14 (600)
s3	-/- (0)	10/20 (∞)	8/- (∞)

(b) Arc costs per oil type [in €per L] and arc capacity [in L]

Table 5.2: Supply chain data

(a) What would be the price the company would be willing to pay for increasing in one unit any of the supplies' availability?

(b) What would be the price the company would be willing to pay for increasing in one unit any of the arcs' capacity?

Exercise 5.2: Dual simplex

(a) Solve the problem below by using the dual simplex method. Report both the primal and dual optimal solutions x and p associated with the optimal basis.

(b) Write the dual formulation of the problem and use strong duality to verify that x and p are optimal.

$$\begin{aligned}
\text{min. } & 2x_1 + x_3 \\
\text{s.t.: } & -1/4x_1 - 1/2x_2 \leq -3/4 \\
& 8x_1 + 12x_2 \leq 20 \\
& x_1 + 1/2x_2 - x_3 \leq -1/2 \\
& 9x_1 + 3x_2 \geq -6 \\
& x_1, x_2, x_3 \geq 0.
\end{aligned}$$

Exercise 5.3: Unboundedness and duality

Consider the standard-form linear programming problem:

$$\begin{aligned}
(P): \text{min. } & c^\top x \\
\text{s.t.: } & Ax = b \\
& x \geq 0,
\end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Show that if P has a finite optimal solution, then the new problem $\overline{P}$ obtained from P by replacing the right-hand side vector b with another one $\bar{b} \in \mathbb{R}^m$ cannot be unbounded no matter what value the components of $\bar{b}$ can take.

Exercise 5.4: Dual in matrix form

Consider the linear programming problem:

$$
\begin{aligned}
(P): \min.\ & {c^1}^\top x^1 + {c^2}^\top x^2 + {c^3}^\top x^3 \\
\text{s.t.}\ & \\
& A^1x^1 + A^2x^2 + A^3x^3 \leq b^1 \quad (y^1) \\
& A^4x^1 + A^5x^2 + A^6x^3 \leq b^2 \quad (y^2) \\
& A^7x^1 + A^8x^2 + A^9x^3 \leq b^3 \quad (y^3) \\
& x^1 \leq 0 \\
& x^2 \geq 0 \\
& x^3 \in \mathbb{R}^{|x^3|},
\end{aligned}
$$

where $A^{1,\ldots,9}$ are matrices, $b^{1,\ldots,3}$, $c^{1,\ldots,3}$ are column vectors, and $y^{1,\ldots,3}$ are the dual variables associated to each constraint.

(a) Write the dual problem in matrix form.

(b) Compute the dual optimum for the case in which

$$
A^1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};\ A^2 = \begin{bmatrix} 5 & 1 \\ 0 & 0 \end{bmatrix};\ A^3 = \begin{bmatrix} 6 \\ 0 \end{bmatrix};\ A_4 = \begin{bmatrix} 1 & 1 \end{bmatrix};\ A^5 = \begin{bmatrix} 0 & 1 \end{bmatrix};\ A^6 = \begin{bmatrix} 1 \end{bmatrix};
$$

$$
A^7 = \begin{bmatrix} 0 & 2 \end{bmatrix};\ A^8 = \begin{bmatrix} 0 & 0 \end{bmatrix};\ A^9 = \begin{bmatrix} 3 \end{bmatrix};\ c^1 = \begin{bmatrix} 3 \\ 9 \end{bmatrix};\ c^2 = \begin{bmatrix} 4 \\ 2 \end{bmatrix};\ c^3 = \begin{bmatrix} 1 \end{bmatrix};\ b^1 = \begin{bmatrix} 5 \\ 10 \end{bmatrix};
$$

$$
b^2 = \begin{bmatrix} 3 \end{bmatrix};\ b^3 = \begin{bmatrix} 6 \end{bmatrix}.
$$

Exercise 5.5: Primal-dual conversion and complementary slackness

Recall the paint factory problem introduced in Section 4.2.3.

(a) Construct the dual of the problem below and solve both the original problem and its dual.

(b) Use complementary slackness to verify that the primal and dual solutions are optimal.

$$
\begin{aligned}
\max.\ & z = 5x_1 + 4x_2 \\
\text{s.t.:}\ & 6x_1 + 4x_2 \leq 24 \\
& x_1 + 2x_2 \leq 6 \\
& x_2 - x_1 \leq 1 \\
& x_2 \leq 2 \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

Chapter 6

Linear Programming Duality - Part II

6.1 Sensitivity analysis

We now consider further developments arising from the notion of duality in linear programming problems. We begin by focusing on the employment of the dual simplex method and the interpretation of the dual multipliers, as discussed in Chapter 5.

Specifically, assume that we have solved to optimality the problem P given as

$$\begin{aligned} P: \text{min. } & c^\top x \\ \text{s.t.: } & Ax = b \\ & x \geq 0. \end{aligned}$$

As we have seen in Chapter 4, the optimal solution $\overline{x}$ with associated basis B satisfies the following optimality conditions: it is a *basic feasible solution* and, therefore (i) $B^{-1}b \geq 0$; and (ii) all reduced costs are nonnegative, that is $c^\top - c_B^\top B^{-1}b \geq 0$.

We are interested in analysing aspects associated with the *stability* of the optimal solution $\overline{x}$ in terms of how it changes with the inclusion of new decision variables and constraints or in with changes in the input data. Both cases are somewhat motivated by the realisation that problems typically emerge from dynamic settings. Thus, one must assess how *stable* a given plan (represented by $\overline{x}$) is or how it can be adapted in face of changes in the original problem setting. This kind of analysis is generally referred to as *sensitivity analysis* in the context of linear programming.

First, we will consider the inclusion of new variables or new constraints *after* the optimal solution $\overline{x}$ is obtained. This setting represents, for example, the inclusion of a new product or a new production plant (referring to the context of resource allocation and transportation problems, as discussed in Chapter 1) or the consideration of additional constraints imposing new (or previously disregarded) requirements or conditions. The techniques we will consider here will also be relevant in the following chapters. We will then discuss specialised methods for large-scale problems and solution techniques for integer programming problems, both topics that heavily rely on the idea of iteratively incrementing linear programming problems with additional constraints (or variables).

The second group of cases relates to changes in the input data. When utilising linear programming models to optimise systems performance, one must bear in mind that there is inherent uncertainty associated with the input data. Be it due to measurement errors or a lack of complete knowledge about the future, one must accept that the input data of these models will, by definition, embed some measure of error. One way of taking this into account is to try to understand the consequences to the optimality of $\overline{x}$ in case of eventual changes in the input data, represented by the matrix A,

and the vectors c and b. We will achieve this by studying the ranges within which variations in these terms do not compromise the optimality of $\overline{x}$.

6.1.1 Adding a new variable

Let us consider that a new variable x_{n+1} with associated column (that is, respective objective function and constraint coefficients) (c_{n+1}, A_{n+1}) is added to P. This leads to a new augmented problem P' of the form

$$\begin{aligned} P' : \text{min. } & c^\top x + c_{n+1}x_{n+1} \\ \text{s.t.: } & Ax + A_{n+1}x_{n+1} = b \\ & x \geq 0, x_{n+1} \geq 0. \end{aligned}$$

We need to determine if, after the inclusion of this new variable, the current basis B is still optimal. If we make the newly added variable nonbasic yields the basic feasible solution (BFS) $x = (\overline{x}, 0)$. Moreover, we know that the optimality condition $c^\top - c_B^\top B^{-1} b \geq 0$ held before the inclusion of the variable, so we know that all the other reduced costs associated with the nonbasic variables $j \in I_N$ were nonnegative.

Therefore, the only check that needs to be done is whether the reduced cost associated with x_{n+1} also satisfies the optimality condition, i.e., if

$$\overline{c}_{n+1} = c_{n+1} - c_B^\top B^{-1} A_{n+1} \geq 0.$$

If the optimality condition is satisfied, the new variable does not change the optimal basis, and the solution $x = (\overline{x}, 0)$ is optimal. Otherwise, one must perform a new simplex iteration, using B as a starting BFS. Notice that in this case, primal feasibility is trivially satisfied, while dual feasibility is not observed (that is, $\overline{c}_{n+1} < 0$). Therefore, primal simplex can be employed, *warm started* by B. This is often a far more efficient strategy than resolving P' from scratch.

Let us consider a numerical example. Consider the problem

$$\begin{aligned} \text{min. } & -5x_1 - x_2 + 12x_3 \\ \text{s.t.: } & 3x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 3x_2 + x_4 = 16 \\ & x_1, \ldots, x_4 \geq 0. \end{aligned}$$

The tableau associated with its optimal solution is given by

	x_1	x_2	x_3	x_4	RHS
z	0	0	2	7	12
x_1	1	0	-3	2	2
x_2	0	1	5	-3	2

Suppose we include a variable x_5, for which $c_5 = -1$ and $A_5 = (1, 1)$. The modified problem then becomes

$$\begin{aligned} \text{min. } & -5x_1 - x_2 + 12x_3 - x_5 \\ \text{s.t.: } & 3x_1 + 2x_2 + x_3 + x_5 = 10 \\ & 5x_1 + 3x_2 + x_4 + x_5 = 16 \\ & x_1, \ldots, x_5 \geq 0. \end{aligned}$$

We have that the reduced cost of the new variable is given by $\bar{c}_5 = c_5 - c_B^\top B^{-1} A_5 = -4$ and $B^{-1}A_5 = (-1, 2)$. The tableau for the optimal basis B considering the new column associated with x_5 is thus

	x_1	x_2	x_3	x_4	x_5	RHS
z	0	0	2	7	-4	12
x_1	1	0	-3	2	-1	2
x_2	0	1	5	-3	2	2

Notice that this tableau now shows a primal feasible solution that is not optimal and can be further iterated using primal simplex.

6.1.2 Adding a new constraint

We now focus on the inclusion of additional constraints. Let us assume that a general constraint of the form ${a_{m+1}}^\top x \geq b_{m+1}$ is added to P. We assume it to be an inequality that has been applied to problem P, which was originally in the standard form, after it was solved.

The first thing to notice is that, if the optimal solution $\overline{x}$ to P satisfy ${a_{m+1}}^\top \overline{x} \geq b_{m+1}$, then nothing changes. Otherwise, we need to rewrite the new constraint accordingly by including a slack variable, obtaining

$$ {a_{m+1}}^\top x - x_{n+1} = b_{m+1}. $$

Notice that doing so changes the matrix A of the original problem P, which becomes

$$ \overline{A} = \begin{bmatrix} A & 0 \\ a_{m+1}^\top & -1 \end{bmatrix}. $$

We can reuse the optimal basis B to form a new basis $\overline{B}$ for the problem. This will have the form

$$ \overline{B} = \begin{bmatrix} B & 0 \\ a^\top & -1 \end{bmatrix}, $$

where a are the respective components of a_{m+1} associated with the columns from A that formed B. Now, since we have that ${\overline{B}}^{-1}\overline{B} = I$, we must have that

$$ {\overline{B}}^{-1} = \begin{bmatrix} B^{-1} & 0 \\ a^\top B^{-1} & -1 \end{bmatrix}. $$

Notice however, that the basic solution $(\overline{x}, {a_{m+1}}^\top \overline{x} - b_{m+1})$ associated with $\overline{B}$ is not feasible, since we assumed that $\overline{x}$ did not satisfy the newly added constraint, i.e., ${a_{m+1}}^\top \overline{x} < b_{m+1}$.

The reduced costs considering the new basis $\overline{B}$ then becomes

$$ [c^\top \;\; 0] - [c_B^\top \;\; 0] \begin{bmatrix} B^{-1} & 0 \\ a^\top B^{-1} & -1 \end{bmatrix} \begin{bmatrix} A & 0 \\ a_{m+1}^\top & -1 \end{bmatrix} = [c^\top - c_B^\top B^{-1} A \;\; 0]. $$

Notice that the new slack variable has a null component as a reduced cost, meaning that it does not violated dual feasibility conditions. Thus, after adding a constraint that makes $\overline{x}$ infeasible, we still have a dual feasible solution that can be immediately used by the dual simplex method, again allowing for warm starting the solution of the new problem.

To build an initial solution in terms of the tableau representation of the simplex method, we must simply add an additional row, which leads to a new tableau with the following structure

$$\overline{B}^{-1}A = \begin{bmatrix} B^{-1}A & 0 \\ a^\top B^{-1}A - a_{m+1}^\top & 1 \end{bmatrix}.$$

Let us consider a numerical example again. Consider the same problem as the previous example, but that we instead include the additional constraint $x_1 + x_2 \geq 5$, which is violated by the optimal solution $(2, 2, 0, 0)$. In this case, we have that $a_{m+1} = (1, 1, 0, 0)$ and $a^\top B^{-1}A - a_{m+1}^\top = [0, 0, 2, -1]$. This modified problem then looks like

$$\begin{aligned} \text{min. } & -5x_1 - x_2 + 12x_3 \\ \text{s.t.: } & 3x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 3x_2 + x_4 = 16 \\ & -x_1 - x_2 + x_5 = -5 \\ & x_1, \ldots, x_5 \geq 0 \end{aligned}$$

with associated tableau

	x_1	x_2	x_3	x_4	x_5	RHS
z	0	0	2	7	0	12
x_1	1	0	-3	2	0	2
x_2	0	1	5	-3	0	2
x_5	0	0	2	-1	1	-1

Notice that this tableau indicates that we have a dual feasible solution that is not primal feasible and thus suitable to be solved using dual simplex.

A final point to note is that these operations are related to each other in terms of equivalent primal-dual formulations. That is, consider dual of P, which is given by

$$\begin{aligned} D : \text{max. } & p^\top b \\ \text{s.t.: } & p^\top A \leq c. \end{aligned}$$

Then, adding a constraint of the form $p^\top A_{n+1} \leq c_{n+1}$ is equivalent to adding a variable to P, exactly as discussed in Section 6.1.1.

6.1.3 Changing input data

We now consider how changes in the input data can influence the optimality of a given basis. Specifically, we consider how to predict whether changes in the vector of independent terms b and objective coefficients c will affect the optimality of the problem. Notice that variations in the coefficient matrix A are left aside.

Changes in the vector b

Suppose that some component b_i changes and becomes $b_i + \delta$, with $\delta \in \mathbb{R}$. We are interested in the range for δ within which the basis B remains optimal.

First, we must notice that optimality conditions $\overline{c} = c^\top - c_B^\top B^{-1}A \geq 0$ are not directly affected by variation in the vector b. This means that the choice of variable indices $j \in I_B$ to form the basis B will, in principle, be *stable* unless the change in b_i is such that B is rendered *infeasible*. Thus, we need to study the conditions in which feasibility is retained, or, more specifically, if (recall the e_i is the vector of zeros except for the i^{th} component being 1)

$$B^{-1}(b + \delta e_i) \geq 0.$$

Let $g = (g_{1i}, \ldots, g_{mi})$ be the i^{th} column of B^{-1}. Thus

$$B^{-1}(b + \delta e_i) \geq 0 \Rightarrow x_B + \delta g \geq 0 \Rightarrow x_{B(j)} + \delta g_{ji} \geq 0, j = 1, \ldots, m.$$

Notice that this is equivalent to having δ within the range

$$\max_{j:g_{ji}>0} \left(-\frac{x_{B(j)}}{g_{ji}} \right) \leq \delta \leq \min_{j:g_{ji}<0} \left(-\frac{x_{B(j)}}{g_{ji}} \right).$$

In other words, changing b_i will incur changes in the value of the basic variables, and thus, we must determine the range within which all basic variables remain nonnegative (i.e., feasible).

Let us consider a numerical example. Once again, consider the problem from Section 6.1.1. The optimal tableau was given by

	x_1	x_2	x_3	x_4	RHS
z	0	0	2	7	12
x_1	1	0	-3	2	2
x_2	0	1	5	-3	2

Suppose that b_1 will change by δ in the constraint $3x_1 + 2x_2 + x_3 = 10$. Notice that the first column of B^{-1} can be directly extracted from the optimal tableau and is given by $(-3, 5)$. The optimal basis will remain feasible if $2 - 3\delta \geq 0$ and $2 + 5\delta \geq 0$, and thus $-2/5 \leq \delta \leq 2/3$.

Notice that this means that we can calculate the change in the objective function value as a function of $\delta \in [-2/5, 2/3]$. Within this range, the optimal cost changes as

$$c_B^\top(b + \delta e_i) = p^\top b + \delta p_i,$$

where $p^\top = c_B^\top B^{-1}$ is the optimal dual solution. In case the variation falls outside that range, this means that some of the basic variables will become negative. However, since the dual feasibility conditions are not affected by changes in b_i, one can still reutilise the basis B using dual simplex to find a new optimal solution.

Changes in the vector c

We now consider the case where variations are expected in the objective function coefficients. Suppose that some component c_j becomes $c_j + \delta$. In this case, optimality conditions become a concern. Two scenarios can occur. First, it might be that the changing coefficient is associated with a variable $j \in J$ that happens to be nonbasic ($j \in I_N$) in the optimal solution. In this case, we have that optimality will be retained as long as the nonbasic variable remains "not attractive", i.e., the reduced cost associated with j remains nonnegative. More precisely put, the basis B will remain optimal if

$$(c_j + \delta) - c_B B^{-1} A_j \geq 0 \Rightarrow \delta \geq -\overline{c}_j.$$

The second scenario concerns changes in variables that are basic in the optimal solution, i.e., $j \in I_B$. In that case, the optimality conditions are directly affected, meaning that we have to analyse the range of variation for δ within which the optimality conditions are maintained, i.e., the reduced costs remain nonnegative.

Let c_j is the coefficient of the l^{th} basic variable, that is $j = B(l)$. In this case, c_B becomes $c_B + \delta e_l$, meaning that all optimality conditions are simultaneously affected. Thus, we have to define a range for δ in which the condition

$$(c_B + \delta e_i)^\top B^{-1} A_i \leq c_i, \forall i \neq j$$

holds. Notice that we do not need to consider j since x_j is a basic variable, and thus, its reduced costs are assumed to remain zero.

Considering the tableau representation, we can use the l^{th} row and examine the conditions for which $\delta q_{li} \leq \overline{c}_i, \forall i \neq j$, where q_{li} is the l^{th} entry of $B^{-1}A_i$.

Let us once again consider the previous example, with optimal tableau

	x_1	x_2	x_3	x_4	RHS
z	0	0	2	7	12
x_1	1	0	-3	2	2
x_2	0	1	5	-3	2

First, let us consider variations in the objective function coefficients of variables x_3 and x_4. Since both variables are nonbasic in the optimal basis, the allowed variation for them is given by

$$\delta_3 \geq -\overline{c}_3 = -2 \text{ and } \delta_4 \geq -\overline{c}_4 = -7.$$

Two points to notice. First, notice that both intervals are one-sided. This means that one should only be concerned with variations that decrease the reduced cost value, since increases in their value can never cause any changes in the optimality conditions. Second, notice that the allowed variation is trivially the negative value of the reduced cost. For variations that turn the reduced costs negative, the current basis can be utilised as a starting point for the primal simplex.

Now, let us consider a variation in the basic variable x_1. Notice that in this case we have to analyse the impact in all reduced costs, with exception of x_1 itself. Using the tableau, we have that $q_l = [1, 0, -3, 2]$ and thus

$$\begin{aligned}
&\delta_1 q_{12} \leq \overline{c}_2 \Rightarrow 0 \leq 0 \\
&\delta_1 q_{13} \leq \overline{c}_3 \Rightarrow \delta_1 \geq -2/3 \\
&\delta_1 q_{14} \leq \overline{c}_4 \Rightarrow \delta_1 \leq 7/2,
\end{aligned}$$

implying that $-2/3 \leq \delta_1 \leq 7/2$. Like before, for a change outside this range, primal simplex can be readily employed.

6.2 Cones and extreme rays

We now change the course of our discussion towards some results that will be useful in identifying two non-ordinary situations when employing the simplex method: unboundedness and infeasibility. Typically, these are consequences of issues related to the data and/or with modelling assumptions and are challenging in that they prevent us from obtaining a solution from the model. As we will see, they also rely on duality in their derivations and are all connected by the notion of cones, which we formally give in Definition 6.1.

Definition 6.1 (Cones)**.** *A set $C \subset \mathbb{R}^n$ is a cone if $\lambda x \in C$ for all $\lambda \geq 0$ and all $x \in C$.*

A cone C can be understood as a set formed by the nonnegative scaling of a collection of vectors $x \in C$. Notice that it implies that $0 \in C$. Often, it will be the case that $0 \in C$ is an extreme point of C and in that case, we say that C is *pointed*. As one might suspect, in the context of linear programming, we will be mostly interested in a specific type of cone, those known are *polyhedral cones*. Polyhedral cones are sets of the form

$$P = \{x \in \mathbb{R}^n : Ax \geq 0\}.$$

Figure 6.1 illustrates a polyhedral cone in $\mathbb{R}^3$ formed by the intersection of three half-spaces.

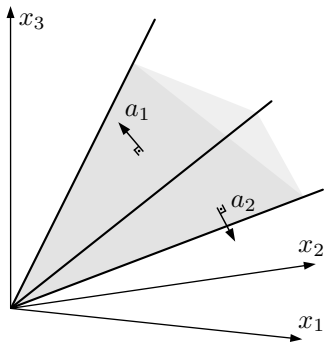


Figure 6.1: A polyhedral cone in $\mathbb{R}^3$ formed by 3 half-space

Some interesting properties can be immediately concluded regarding polyhedral cones. First, they are convex sets, since they are polyhedral sets (cf. Theorem 2.8). Also, the origin is an extreme point, and thus, polyhedral cones are always pointed. Furthermore, just like general polyhedral sets, a cone $C \in \mathbb{R}^n$ will always be associated with a collection of n linearly independent vectors. Corollary 6.2 summarises these points. Notice we pose it as a Corollary because these are immediate consequences of Theorem 3.5.

Corollary 6.2. *Let $C \subset \mathbb{R}^n$ be a polyhedral cone defined by constraints $\left\{a_i^\top x \geq 0\right\}_{i=1,\ldots,m}$. Then the following are equivalent*

1. *0 is an extreme point of C;*
2. *C does not contain a line;*
3. *There exists n vectors in $a_1, \ldots, a_m$ that are LI.*

Proof. Theorem 3.5 proof verbatim, with $P = C$. □

Notice that $0 \in C$ is the unique extreme point of the polyhedra cone C. To see that, let $0 \neq x \in C$, $x_1 = (1/2)x$ and $x_2 = (3/2)x$. Note that $x_1, x_2 \in C$, and $x \neq x_1 \neq x_2$. Setting $\lambda_1 = \lambda_2 = 1/2$, we have that $\lambda_1 x_1 + \lambda_2 x_2 = x$ and thus, x is not an extreme point (cf. Definition 2.10).

6.2.1 Recession cones and extreme rays

We now focus on a specific type of cone, called *recession cones*. In the context of linear optimisation, recession cones are useful for identifying directions of unboundedness. Let us first formally define the concept.

Definition 6.3 (Recession cone)**.** *Consider the polyhedral set $P = \{x \in \mathbb{R}^n : Ax \geq b\}$. The recession cone at $\overline{x} \in P$, denoted* $\mathbf{recc}(P)$*, is defined as*

$$\mathbf{recc}(P) = \{d \in \mathbb{R}^n : A(\overline{x} + \lambda d) \geq b, \lambda \geq 0\} \text{ or } \{d \in \mathbb{R}^n : Ad \geq 0\}.$$

Notice that the definition states that a recession cone comprises all directions d along which one can move from $\overline{x} \in P$ without ever leaving P. However, notice that the definition do not depend on $\overline{x}$, meaning that the recession cone is unique for the polyhedral set P, regardless of its "origin". Furthermore, notice that Definition 6.3 implies that recession cones of polyhedral sets are polyhedral cones.

We say that any directions $d \in \mathbf{recc}(P)$ is a *ray*. Thus, bounded polyhedra can be alternatively defined as polyhedral sets that do not contain rays.

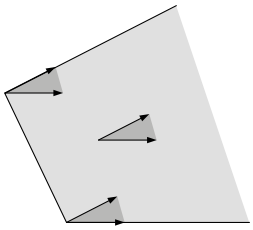


Figure 6.2: Representation of the recession cone of a polyhedral set

Figure 6.2 illustrates the concept of recession cones. Notice that it is purposely placed in several places to illustrate the independence of the point $\overline{x} \in P$.

Finally, the recession cone for a standard form polyhedral set $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ is given by

$$\mathbf{recc}(P) = \{d \in \mathbb{R}^n : Ad = 0, d \geq 0\}.$$

6.2.2 Unbounded problems

To identify unboundedness in linear programming problems, we must check for the existence of *extreme rays*. Extreme rays are analogous to extreme points, but defined with a "loose" degree of freedom. Definition 6.4 provides a technical definition of extreme rays.

Definition 6.4 (Extreme ray)**.** *Let $C \subset \mathbb{R}^n$ be a nonempty polyhedral cone. A nonzero $x \in C$ is an extreme ray if there are $n-1$ linearly independent active constraints at x.*

Notice that we are interested in extreme rays of the recession cone $\mathbf{recc}(P)$ of the polyhedral set P. However, it is typical to say that they are extreme rays of P. Figure 6.3 illustrates the concept of extreme rays in polyhedral cones.

Notice that, just like extreme points, the number of extreme rays is finite by definition. In fact, we say that two extreme rays are equivalent if they are positive multiples corresponding to the same $n-1$ linearly independent active constraints.

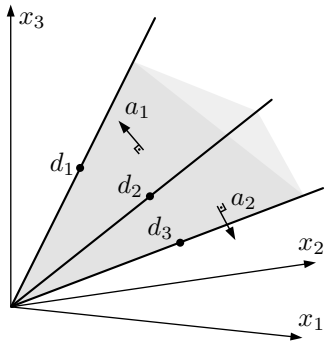


Figure 6.3: A polyhedral cone formed by the intersection of three half-spaces (the normal vector a_3 is perpendicular to the plane of the picture and cannot be seen). Directions d_1, d_2, and d_3 represent extreme rays.

The existence of extreme rays can be used to verify unboundedness in linear programming problems. The mere existence of extreme rays does not suffice since unboundedness is a consequence of the extreme ray being a direction of improvement for the objective function. To demonstrate this, let us first describe unboundedness in polyhedral cones, which we can then use to show the unboundedness in polyhedral sets.

Theorem 6.5 (Unboundedness in polyhedral cones)**.** *Let P : min. $\{c^\top x : x \in C\}$, with $C = \{a_i^\top x \geq 0, i = 1, \ldots, m\}$. The optimal value is equal to $-\infty$ if and only if some extreme ray $d \in C$ satisfies $c^\top d < 0$.*

Proof. If $c^\top d < 0$, then P is unbounded, since $c^\top x \to -\infty$ along d. Also, there exists some $x \in C$ for which $c^\top x < 0$ can be scaled to -1.

Let $P = \{x \in \mathbb{R}^n : a_i^\top x \geq 0, i = 1, \ldots, m, c^\top x = -1\}$. Since $0 \in C$, P has at least one extreme point $\{a_i\}_{i=1}^m$ and thus span $\mathbb{R}^n$ (cf. Theorem 3.5). Let d be one of those. As we have n linearly-independent active constraints at d, $n-1$ of the constraints $\{a_i^\top x \geq 0\}_{i=1}^m$ must be active (plus $c^\top x = -1$), and thus d is an extreme ray. □

We can now expand the result to general polyhedral sets.

Theorem 6.6 (Unboundedness in polyhedral sets)**.** *Let P : min. $\{c^\top x : x \in X\}$ with $X = \{x \in \mathbb{R}^n : Ax \geq b\}$ and assume that the feasible set has at least one extreme point. Then, the optimal value is $-\infty$ if and only if $c^\top d < 0$.*

Proof. As before, if $c^\top d < 0$, then P is unbounded, since $c^\top x \to -\infty$ along d. Now, let D : max. $\{p^\top b : p^\top A = c^\top, p \geq 0\}$ be the dual of P. Recall that, if P is unbounded, then D is infeasible, and so must be D^0 : max. $\{p^\top 0 : p^\top A = c^\top, p \geq 0\}$. This implies that the primal P^0 : min. $\{c^\top x : Ax \geq 0\}$ is unbounded (as 0 is feasible).

The existence of at least one extreme point for P implies that the rows $\{a_i\}_{i=1,\ldots,m}$ of A span $\mathbb{R}^n$ and $\mathbf{recc}(X) = \{x \in \mathbb{R}^n : Ax \geq 0\}$ is pointed. Thus, by Theorem 6.5 there exists d such that $c^\top d < 0$. □

We now focus on how this can be utilised in the context of the simplex method. It turns out that once unboundedness is identified in the simplex method, one can extract the extreme ray causing

the said unboundedness. In fact, most professional-grade solvers are capable of returning extreme (or unbounded) rays, which is helpful in the process of understanding the causes for unboundedness in the model. We will also see in the next chapter that these extreme rays are also used in the context of specialised solution methods.

To see that is possible, let P : min. $\left\{c^\top x : x \in X\right\}$ with $X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ and assume that, for a given basis B, we conclude that the optimal value is $-\infty$, that is, the problem is unbounded. In the context of the simplex method, this implies that we found a nonbasic variable x_j for which the reduced cost $\bar{c}_j < 0$ and the j^{th} column of $B^{-1}A_j$ has no positive coefficient. Nevertheless, we can still form the feasible direction $d = [d_B \; d_N]$ as before, with

$$d_B = -B^{-1}A_j \text{ and } d_N = \begin{cases} d_j = 1 \\ d_i = 0, \forall i \in I_N \setminus \{j\}. \end{cases}$$

This direction d is precisely an extreme ray for P. To see that, first, notice that $Ad = 0$ and $d \geq 0$, thus $d \in \mathbf{recc}(X)$. Moreover, there are $n-1$ active constraints at d: m in $Ad = 0$ and $n - m - 1$ in $d_i = 0$ for $i \in I_N \setminus \{j\}$. The last thing to notice is that $\bar{c}_j = c^\top d < 0$, which shows the unboundedness in the direction d.

6.2.3 Farkas' lemma

We now focus on the idea of generating certificates of infeasibility for linear programming problems. That is, we show that if a problem P is infeasible, then there is a structure that can be identified to certify the infeasibility. To see how this works, consider the two polyhedral sets

$$X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\} \text{ and}$$
$$Y = \left\{p \in \mathbb{R}^m : p^\top Ax \geq 0, p^\top b < 0\right\}.$$

If there exists any $p \in Y$, then there is no $x \in X$ for which $p^\top Ax = p^\top b$, (and in turn $Ax = b$), holds. Thus, X must be empty. Notice that this can be used to infer that a problem P with a feasibility set represented by X prior to solving P itself, by means of solving the *feasibility problem* of finding a vector $p \in Y$.

We now pose this relationship more formally via a result generally known as the Farkas' lemma.

Theorem 6.7 (Farkas' lemma)**.** *Let A be a $m \times n$ matrix and $b \in \mathbb{R}^m$. Then, exactly one of the following statements hold*

(1) There exists some $x \geq 0$ such that $Ax = b$;

(2) there exists some vector p such that $p^\top A \geq 0$, $p^\top b < 0$.

Proof. Assume that (1) is satisfied. If $p^\top A \geq 0$, then $p^\top b = p^\top Ax \geq 0$, which violates (2).

Now, consider the primal-dual pair P : min. $\left\{0^\top x : Ax = b, x \geq 0\right\}$ and D : max. $\left\{p^\top b : p^\top A \geq 0\right\}$. Being P infeasible, D must be unbounded (instead of infeasible) since $p = 0$ is feasible for D. Thus, $p^\top b < 0$ for some $p \neq 0$. □

The Farkas' lemma has a nice geometrical interpretation that represents the mutually exclusive relationship between the two sets. For that, notice that we can think of b as being a conic combination of the columns A_j of A, for some $x \geq 0$. If that cannot be the case, then there exists a hyperplane that separates b and the cone formed by the columns of A, $C = \{y \in \mathbb{R}^m : y = Ax\}$.

This is illustrated in Figure 6.4. Notice that the separation caused by such a hyperplane with normal vector p implies that $p^\top Ax \geq 0$ and $p^\top b < 0$, i.e., Ax and b are on the opposite sides of the hyperplane.

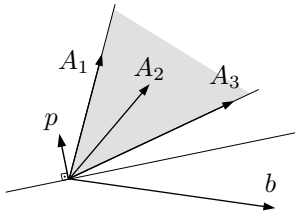


Figure 6.4: Since $b \notin X$, $p^\top x = 0$ separates them

6.3 Resolution theorem*

Lastly, we present a result that will be the foundation for our discussion in the next chapter. Specifically, we will see how we can use a presentation of the polyhedral set P purely based on extreme points and extreme rays to devise solution strategies that can be more efficient for large-scale linear programming problems.

For now, let us concentrate on this alternative representation. Basically, polyhedral sets in standard form can be represented in two manners: either by (i) a finite set of linear constraints; or (ii) by combinations of its extreme points and extreme rays.

Clearly, the first representation is far more practical than the second. That is, the second representation is an *explicit* representation that would require knowing beforehand each extreme point and extreme ray forming the polyhedral set. Notice that the first representation, which we have relied on so far, has extreme points and extreme rays only implicitly represented. However, we will see that this explicit representation has an important application in the devising of alternative solution methods for large-scale linear programming problems. This fundamental result is stated in Theorem 6.8.

Theorem 6.8 (Resolution theorem)**.** *Let $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ be a nonempty polyhedral set with at least one extreme point. Let $\{x_i\}_{i=1}^k$ be the set with all extreme points, and $\{w\}_{j=1}^r$ be the set of all extreme rays of P. Then $P = Q$, where*

$$Q = \left\{ \sum_{i=1}^{k} \lambda_i x^i + \sum_{j=1}^{r} \theta_j w^j : \lambda_i \geq 0,\ \theta_j \geq 0,\ \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$

Theorem 6.8 has an important consequence, as it states that bounded polyhedra, i.e., polyhedral sets that have no extreme rays, can be represented by the convex hull of its extreme points. For now, let us look at an example that illustrates the concept.

Consider the polyhedral set P given by

$$P = \{x_1 - x_2 \geq -2; x_1 + x_2 \geq 1, x_1, x_2 \geq 0\}.$$

The recession cone $C = \mathbf{recc}(P)$ is described by $d_1 - d_2 \geq 0$, $d_1 + d_2 \geq 0$ (from $Ad = 0$), and $d_1, d_2 \geq 0$, which can be simplified as

$$C = \left\{(d_1, d_2) \in \mathbb{R}^2 : 0 \leq d_2 \leq d_1\right\}.$$

We can then conclude that the two vectors $w^1 = (1, 1)$ and $w_2 = (1, 0)$ are extreme rays of P. Moreover, P has three extreme points: $x_1 = (0, 2)$, $x_2 = (0, 1)$, and $x_3 = (1, 0)$.

Figure 6.5 illustrates what is stated in Theorem 6.8. For example, a representation for the point $y = (2, 2) \in P$ is given by

$$y = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

that is, $y = x^2 + w^1 + w^2$. Notice however, that y could also be represented as

$$y = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{3}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

with then $y == \frac{1}{2}x^2 + \frac{1}{2}x^3 + \frac{3}{2}w^1$. Notice that this imply that the representation of each point is not unique.

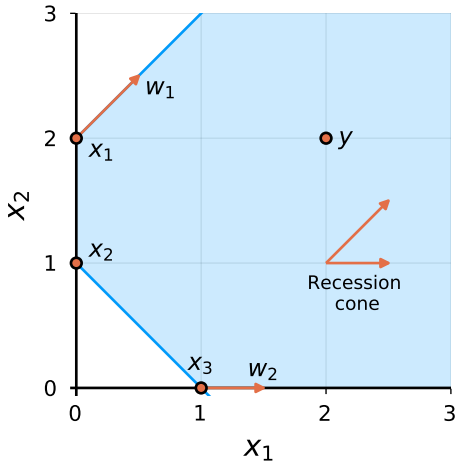


Figure 6.5: Example showing that every point of $P = \{x_1 - x_2 \geq -2; x_1 + x_2 \geq 1, x_1, x_2 \geq 0\}$ can be represented as a convex combination of its extreme point and a linear combination of its extreme rays

6.4 Exercises

Exercise 6.1: Sensitivity analysis in the RHS

Consider the following linear programming problem and its optimal tableau below:

$$\begin{aligned} \text{min. } & -2x_1 - x_2 + x_3 \\ \text{s.t.: } & x_1 + 2x_2 + x_3 \leq 8 \\ & -x_1 + x_2 - 2x_3 \leq 4 \\ & 3x_1 + x_2 \leq 10 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	0	1.2	0.2	0	0.6	-7.6
x_1	1	0	-0.2	-0.2	0	0.4	2.4
x_2	0	1	0.6	0.6	0	-0.2	2.8
x_5	0	0	-2.8	-0.8	1	0.6	3.6

(a) If you were to choose between increasing in 1 unit the right-hand side of any constraints, which one would you choose, and why? What is the effect of the increase on the optimal cost?

(b) Perform a sensitivity analysis on the model to discover what is the range of alteration in the RHS in which the same effect calculated in item (a) can be expected. *HINT*: JuMP (from version 0.21.6) includes the function "lp_sensitivity_report()" that you can use to help performing the analysis.

Exercise 6.2: Extreme points and extreme rays

(a) Let $P = \{(x_1, x_2) : x_1 - x_2 = 0,\ x_1 + x_2 = 0\}$. What are the extreme points and the extreme rays of P?

(b) Let $P = \{(x_1, x_2) : 4x_1 + 2x_2 \geq 0,\ 2x_1 + x_2 \leq 1\}$. What are the extreme points and the extreme rays of P?

(c) For the polyhedron of part (b), is it possible to express each one of its elements as a convex combination of its extreme points plus a nonnegative linear combination of its extreme rays? Is this compatible with the Resolution Theorem?

Exercise 6.3: From Farkas' lemma to duality

Use the Farkas' lemma to prove the duality theorem for a linear programming problem involving constraints of the form $a_i'x = b_i, a_i'x \geq b_i$, and nonegativity constraints for some of the variables x_j. *Hint*: Start by deriving the form of the set of feasible directions at an optimal solution.

Exercise 6.4: Adding a constraint

Consider the linear programming problem below with optimal basis $[x_1, x_2, x_5, x_6]$ and dual variables $p_1, \ldots, p_4$.

$$\begin{aligned} \text{max. } & 2x_1 + x_2 \\ \text{s.t.: } & 2x_1 + 2x_2 \leq 9 \quad (p_1) \\ & 2x_1 - x_2 \leq 3 \quad (p_2) \\ & x_1 \leq 3 \quad (p_3) \\ & x_2 \leq 4 \quad (p_4) \\ & x_1, x_2 \geq 0. \end{aligned}$$

(a) Find the primal and dual optimal solutions. *HINT*: You can use complementary slackness, once having the primal optimum, to find the dual optimal solution.

(b) Suppose we add a new constraint $6x_1 - x_2 \leq 6$, classify the primal and dual former optimal points stating if they: (i) remain optimal; (ii) remain feasible but not optimal; or (iii) become infeasible.

(c) Consider the new problem from item (b) and find the new dual optimal point through one dual simplex iteration. After that, find the primal optimum.

Chapter 7

Decomposition methods

7.1 Large-scale problems

In this chapter, we consider the notion of *decomposition*, which consists of a general term used in the context of mathematical programming to refer to solution methods that utilise some separability mechanism to more efficiently solve large-scale problems.

In general, decomposition methods are based on the premise that it is more efficient, under a computational standpoint, to repeatedly resolve a (collection of) smaller instances of a problem than to solve the full-scale original problem. More recently, with the widespread adoption of multithreaded processors and computing clusters with multiple nodes, decomposition methods have become attractive as parallelisation strategies, which can yield considerable computational savings.

There a mainly two-classes of decomposition methods. The first class utilises the explicit representation of polyhedral sets, as stated in Theorem 6.8, to iteratively reconstruct the full-scale problem, with the hope that the structure containing the optimal vertex will be successfully reconstructed before all of the problem itself is reconstructed. It turns out that this is the case in many applications, which is precisely the feature that render these methods very efficient in some contexts. This is the class of methods we are going to analyse in this chapter, first the Dantzig-Wolfe decomposition and related column generation, and then its equivalent dual method, generally known as Benders' decomposition.

The second class of methods utilises Lagrangian duality for obtaining separability. We will delay the presentation of this sort of approach to Part II, when we discuss Lagrangian duality under the more general context of nonlinear programming problems.

In either case, decomposition methods are designed in a way that they seek to break problems into easier parts by removing linking elements. Specifically, let

$$(P) : \text{min. } \left\{ c^\top x : x \in X \right\},$$

where $X = \bigcap_{k=1}^{K} X_k$, for some $K > 0$, and

$$X_k = \left\{ x^k \in \mathbb{R}^{n_k}_+ : D_k x_k = d_k \right\}, \forall k \in \{1, \ldots, K\}.$$

That is, X is the intersection of K standard-form polyhedral sets. Our objective is to devise a way to break into K separable parts that can be solved separately and recombined as a solution for P. In this case, this can be straightforwardly achieved by noticing that P can be equivalently stated as

$$\begin{array}{rllll} (P): \max._x & c_1^\top x_1 & +\cdots+ & c_K^\top x_K & \\ \text{s.t.:} & D_1 x_1 & & & = d_1 \\ & & \ddots & & \vdots \\ & & & D_K x_K & = d_K \\ & x_1, & \ldots, & x_K & \in \mathbb{R}^n_+ \end{array}$$

has a structure that immediately allows for separation. That is, P could be solved as K independent problems

$$P_k : \min. \ \left\{ c_k^\top x_k : x_k \in X_k \right\}$$

in parallel and then combine their individual solutions onto a solution for P, simply by making $\overline{x} = [x_k]_{k=1}^K$ and $c^\top \overline{x} = \sum_{i=1}^K c_k^\top \overline{x}_k$. Notice that, if we were to assume that the solution time scales linearly (it does not; it grows faster than linear) and $K = 10$, then solving P as K separated problems would be ten times faster (that is not true; there are bottlenecks and other considerations to take into account, but the point stands).

Unfortunately, *complicating structures* often compromise this natural separability, preventing one from being able to directly exploit this idea. Specifically, two types of complicating structures can be observed. The first is of the form of *complicating constraint*. That is, we observe that a constraint is such that it connects variables from (some of) the subsets X_k. In this case, we would notice that P has an additional constraint of the form

$$A_1 x_1 + \cdots + A_K x_K = b,$$

which precludes separability, since the problem structure becomes

$$\begin{array}{rllll} P': \max._x & c_1^\top x_1 & +\cdots+ & c_K^\top x_K & \\ \text{s.t.:} & A_1 x_1 & +\cdots+ & A_K x_K & = b \\ & D_1 x_1 & & & = d_1 \\ & & \ddots & & \vdots \\ & & & D_K x_K & = d_K \\ & x_1, & \ldots, & x_K & \in \mathbb{R}^n_+. \end{array}$$

The other type of complicating structure is the case in which the same set of decision variable is present in multiple constraints, or multiple subsets X_k. In this case, we observe that variables of a subproblem $k \in \{1, \ldots, K\}$ has nonzero coefficient in another subproblem $k' \neq k$, $k' \in \{1, \ldots, K\}$. Hence, problem P takes the form of

$$\begin{array}{rllllll} P'': \max._x & c_0^\top x_0 + & c_1^\top x_1 & +\cdots+ & c_K^\top x_K & & \\ \text{s.t.:} & A_1 x_0 + & D_1 x_1 & & & & = d_1 \\ & \vdots & & \ddots & & & \vdots \\ & A_K x_0 + & & & & D_K x_K & = d_K \\ & x_0, & x_1, & \ldots, & & x_K & \in \mathbb{R}^n_+. \end{array}$$

The challenging aspect is that, depending on the type of complicating structure, a specific method becomes more suitable. Therefore, being able to identify these structures is one of the key success factors in terms of the chosen method performance. As a general rule, problems with complicating constraints (as P') are suitable to be solved by a delayed variable generation method such as

column generation. Analogously, problem with complicating variables (P'') are better suited for employing delayed constraint generation methods such as Benders decomposition.

The development of professional-grade code employing decomposition methods is a somewhat recent occurrence. The commercial solver CPLEX offers a Benders decomposition implementation that requires the user to specify the separable structure. On the other hand, although there are some available frameworks for implementing column generation-based methods, these tend to be more ad hoc occurrences, yet often reaping impressive results.

7.2 Dantzig-Wolfe decomposition and column generation*

We start with the Dantzig-Wolfe decomposition, which consists of an alternative approach for reducing memory requirements when solving large-scale linear programming problems. Then, we show how this can be expanded further with the notion of delayed variable generation to yield a truly decomposed problem.

7.2.1 Dantzig-Wolfe decomposition

As before, let $P_k = \{x_k \geq 0 : D_k x_k = d\}$, with $P_k \neq \emptyset$ for $k \in \{1, \ldots, K\}$. Then, the problem P can be reformulated as:

$$\begin{aligned} \text{min.} \quad & \sum_{k=1}^{K} c_k^\top x_k \\ \text{s.t.:} \quad & \sum_{k=1}^{K} A_k x_k = b \\ & x_k \in P_k, \ \forall k \in \{1, \ldots, K\}. \end{aligned}$$

Notice that P has a complicating constraint structure, due to the constraints $\sum_{k=1}^{K} A_k x_k = b$. In order to devise a decomposition method for this setting, let us first assume that we have available for each of the sets P_k, $k \in \{1, \ldots, K\}$, (i) all extreme points, represented by x_k^j, $\forall j \in J_k$; and (ii) all extreme rays w_k^r, $\forall r \in R_k$. As one might suspect, this is in principle a demanding assumption, but one that we will be able to drop later on.

Using the Resolution theorem (Theorem 6.8), we know that any element of P_k can be represented as

$$x_k = \sum_{j \in J_k} \lambda_k^j x_k^j + \sum_{r \in R_k} \theta_k^r w_k^r, \tag{7.1}$$

where $\lambda_k^j \geq 0$, $\forall j \in J_k$, are the coefficients of the convex combination of extreme points, meaning that we also observe $\sum_{j \in J_k} \lambda_k^j = 1$, and $\theta_k^r \geq 0$, $\forall r \in R_k$, are the coefficients of the conic combination of the extreme rays.

Using the identity represented in (7.1), we can reformulate P onto the *main problem* P_M as follows.

$$
\begin{aligned}
(P_M): \min. \quad & \sum_{k=1}^{K}\left(\sum_{j\in J_k}\lambda_k^j c_k^\top x_k^j + \sum_{r\in R_k}\theta_k^r c_k^\top w_k^r\right) \\
\text{s.t.:} \quad & \sum_{k=1}^{K}\left(\sum_{j\in J_k}\lambda_k^j A_k x_k^j + \sum_{r\in R_k}\theta_k^r A_k w_k^r\right) = b \qquad (7.2)\\
& \sum_{j\in J_k}\lambda_k^j = 1,\ \forall k\in\{1,\dots,K\} \qquad (7.3)\\
& \lambda_k^j \geq 0, \theta_k^r \geq 0, \forall j\in J_k, r\in R_k, k\in\{1,\dots,K\}\,.
\end{aligned}
$$

Notice that (7.2) and (7.3) can be equivalently represented as

$$
\sum_{k\in K}\left(\sum_{j\in J_k}\lambda_k^j\begin{bmatrix}A_k x_k^j\\ e_k\end{bmatrix} + \sum_{r\in R_k}\theta_k^r\begin{bmatrix}A_k w_k^r\\ 0\end{bmatrix}\right) = \begin{bmatrix}b\\ 1\end{bmatrix},
$$

where e_k is the unit vector (i.e., with 1 in the k^{th} component, and 0 otherwise). Notice that P_M has as many variables as the number of extreme points and extreme rays of P, which is likely to be prohibitively large.

However, we can still solve it if we use a slightly modified version of the revised simplex method. To see that, let us consider that b is a m-dimensional vector. Then, a basis for P_M would be of size $m+K$, since we have the original m constraints plus one for each convex combination (arising from each subproblem $k\in K$). This means that we are effectively working with $(m+K)\times(m+K)$ matrices, i.e., the basic matrix B and its inverse B^{-1}. Another element we need is the vector of simplex multipliers p, which is a vector of dimension $(m+K)$.

The issue with the representation adopted in P_M arises when we are required to calculate the reduced costs of *all* the nonbasic variables, since this is the critical issue for its tractability. That is where the method provides a clever solution. To see that, notice that the vector p is formed by components $p^\top = (q, r_1, \dots, r_K)^\top$, where q represent the m dual variables associated with (7.2), and r_k, $\forall k\in\{1,\dots,K\}$, are the dual variables associated with (7.3).

The reduced costs associated with the extreme-point variables λ_k^j, $j\in J_K$, is given by

$$
c_k^\top x_k^j - [q^\top\ r_1\ \dots\ r_K]\begin{bmatrix}A_k x_k^j\\ e_k\end{bmatrix} = (c_k^\top - q^\top A_k)x_k^j - r_k. \qquad (7.4)
$$

Analogously, the reduced cost associated with extreme-ray variables θ_k^r, $r\in R_k$, is

$$
c_k^\top w_k^r - [q^\top\ r_1\ \dots\ r_K]\begin{bmatrix}A_k w_k^r\\ 0\end{bmatrix} = (c_k^\top - q^\top A_k)w_k^r. \qquad (7.5)
$$

The main difference is how we assess the reduced costs of the non-basic variables. Instead of explicitly calculating the reduced costs of all variables, we instead rely on an optimisation-based approach to consider them only *implicitly*. For that, we can use the subproblem

$$
\begin{aligned}
(S_k): \min_x \quad & \bar{c}_k = (c_k^\top - q^\top A_k)x_k \\
\text{s.t.:} \quad & x_k \in P_k,
\end{aligned}
$$

which can be solved in parallel for each subproblem $k \in \{1, \ldots, K\}$. The subproblem S_k is known as the *pricing problem*. For each subproblem $k = 1, \ldots, K$, we have the following cases.

We might observe that $\overline{c}_k = -\infty$. In this case, we have found an *extreme ray* w_k^r satisfying $(c_k^\top - q^\top A_k)w_k^r < 0$. Thus, the reduced cost of the associated extreme-ray variable θ_k^r is negative. If that is the case, we must generate the column

$$\begin{pmatrix} A_k w_k^r \\ 0 \end{pmatrix}$$

associated with θ_k^r and make it enter the basis.

Otherwise, being S_k bounded, i.e., $\overline{c}_k < \infty$, two other cases can occur. The first is the case in which $\overline{c}_k < r_k$. Therefore, we found an extreme point x_k^j satisfying $(c_k^\top - q^\top A_k)x_k^j - r_k < 0$. Thus, the reduced cost associated with the extreme-point variable λ_k^j is negative and, analogously, we must generate the column

$$\begin{pmatrix} A_k x_k^j \\ e_k \end{pmatrix}$$

associated with λ_k^j and make it enter the basis.

The last possible case is when we observe that $r_k < \overline{c}_k < \infty$. In this case, the pricing problem could not identify a beneficial variable to be made basic, and therefore there is not an extreme point or ray with negative reduced cost for subproblem k. If this condition holds for all $k = 1, dots, K$, then all necessary extreme points and rays to characterise the region where the optimal extreme point lies (or one of the extreme points, in the case of multiple solutions) have been found and the optimal solution can be recovered.

Algorithm 4 summarises the Dantzig-Wolfe method. The two most remarkable features of the method are (i) the fact that columns are not explicitly represented, but generated "on demand" and (ii) the fact that the pricing problem requires the solution of another linear programming problem. Analogously to the simplex method, it might be necessary to employ a "Phase 1" approach to obtain an initial basis to start the algorithm.

Under a theoretical standpoint, the Dantzig-Wolfe method is equally efficient as the revised simplex method. There are however two settings where the decomposition is most favourable. The first, consists of applications in which the pricing problem can be solved in a closed-form, without invoking a method to solve an additional linear programming subproblem. There are a few examples in which this happens to be the case and certainly many others yet to be discovered.

Secondly, the memory requirements of the Dantzig-Wolfe decomposition makes it an interesting approach for very large-scale problems. The original simplex method requires an amount of memory space that is $O((m + K \times m_0)^2)$, where m_0 is the number of rows of D_k, for $\forall k \in \{1, \ldots, K\}$. This is essentially the size of the inverse basic matrix B^{-1}. In contrast, the Dantzig-Wolfe reformulation requires $O((m + K)^2) + K \times O(m_0^2)$ of memory space, with the first term referring to the main problem inverse basic matrix and the second to the pricing problems basic matrices. For example, for a problem in which $m = m_0$ and much larger than, say, $K = 10$, this implies that the memory space required by the Dantzig-Wolfe reformulation is 100 times smaller, which can substantially enlarge the range of large-scale problems that can be solved for the same amount of computational resources available.

Algorithm 4 Dantzig-Wolfe decomposition

initialise. Let B be a BFS for P_M and set $l \leftarrow 0$.
repeat
  for $k \in \{1, \ldots, K\}$ **do**
    solve S_k and let $\bar{c}_k = \min_x \{S_k\}$
    if $\bar{c}_k = -\infty$ **then**
      obtain extreme ray w_k^r and make $R_k^l = R_k^l \cup \{w_k^r\}$.
      generate column $(A_k w_k^r, 0)$ to become basic.
    else if $\bar{c}_k < r_k < \infty$ **then**
      obtain extreme point x_k^j and make $J_k^l = J_k^l \cup \left\{x_k^j\right\}$.
      generate column $(A_k x_k^j, e_k)$ to become basic.
    end if
  end for
  select one of the generated columns to replace one of the columns of B and update B accordingly.
  $l \leftarrow l + 1$.
until $\bar{c}_k > r_k$ for all $k \in \{1, \ldots, K\}$
return B

7.2.2 Delayed column generation

The term column generation can also refer to a related, and perhaps more widely known, variant of the Dantzig-Wolfe decomposition. In that, the main problem P_M is also repeatedly solved, each time being incremented by an additional variable (or variables) associated with the column(s) identified with negative reduced costs in the pricing problem S_k, $k \in \{1, \ldots, K\}$. This is particularly useful for problems with exponentially increasing number of variables, or with a large number of variables associated with the complicating constraints (i.e., when m is a large number).

Algorithm 5 Column generation algorithm

1: **initialise.** Let $\tilde{X}_k \subset X_k$, for $k \in \{1, \ldots, K\}$, and set $l \leftarrow 0$.
2: **repeat**
3: solve P_M^l to obtain $\lambda^{*l} = (\lambda_1^{*l}, \ldots, \lambda_K^{*l})$ and duals $(q^{*l}, \{r_k^{*l}\}_{k=1}^K)$.
4: **for** $k \in \{1, \ldots, K\}$ **do**
5: solve the pricing problem

$$\overline{x}_k^{*l} \leftarrow \operatorname{argmin} \left\{c_k^\top x_k - q^{*l}(A_k x_k) - r_k^{*l} : x_k \in P_k\right\}.$$

6: **if** $\bar{c}_k = c_k^\top \overline{x}_k^{*l} - q^{*l}(A_k \overline{x}_k^{*l}) < r_k^{*l}$ **then** $\tilde{X}_k \leftarrow \tilde{X}_k \cup \left\{\overline{x}_k^{*l}\right\}$
7: **end if**
8: **end for**
9: $l \leftarrow l + 1$.
10: **until** $\bar{c}_k > r_k$ for all $k \in \{1, \ldots, K\}$
11: **return** λ^{*l}.

The (delayed) column generation method is presented in Algorithm 5. Notice in Line 6 the step that is generating new columns in the main problem P_M, represented in the statement $\tilde{X}_k \leftarrow \tilde{X}_k \cup \left\{\overline{x}_k^{*l}\right\}$. That is precisely when new variables λ_k^t are introduced in the P_M with coefficients represented by

the column

$$\begin{pmatrix} c_k \overline{x}_k^{*l} \\ A_k \overline{x}_k^{*l} \\ e_k \end{pmatrix}.$$

Notice that the unbounded case is not treated to simplify the pseudocode, but could be trivially adapted to return extreme rays to be used in P_M, like the previous variant presented in Algorithm 4. Also, notice that the method is assumed to be initialised with a collection of columns (i.e., extreme points) $\tilde{X}_k$, which can normally be obtained from inspection or using a heuristic method.

We finalise showing that the Dantzig-Wolfe and column generation methods can provide information related to its own convergence. This means that we have access to an optimality bound that can be used to monitor the convergence of the method and allow for a preemptive termination given an acceptable tolerance. This bounding property is stated in Theorem 7.1.

Theorem 7.1. *Suppose P is feasible with finite optimal value z. Let $\overline{z}$ be the optimal cost associated with P_M at a given iteration l of the Dantzig-Wolfe method. Also, let r_k be the dual variable associated with the convex combination of the k^{th} subproblem and z_k its optimal cost. Then*

$$z + \sum_{k=1}^{K} (z_k - r_k) \leq z \leq \overline{z}.$$

Proof. We know that $z \leq \overline{z}$, because a solution for P_M is primal feasible and thus feasible for P.

Now, consider the dual of P_M

$$\begin{aligned} (D_M): \quad \max. \quad & q^\top b + \sum_{k=1}^{K} r_k \\ \text{s.t.:} \quad & q^\top A_k x_k^j + r_k \leq c_k^\top x_k^j, \ \forall j \in J_k, \forall k \in \{1, \ldots, K\} \\ & q^\top A_k w_k^r \leq c_k^\top w_k^r, \ \forall r \in R_k, \forall k \in \{1, \ldots, K\} \end{aligned}$$

We know that strong duality holds, and thus $z = q^\top b + \sum_{k=1}^{K} r_k$ for dual variables $(q, r_1, \ldots, r_K)$.

Now, since z_k are finite, we have $\min_{j \in J_k}(c_k^\top x_k^j - q^\top A_k x_k^j) = z_k$ and $\min_{r \in R_k}(c_k^\top w_k^r - q^\top D_k w_k^r) \geq 0$, meaning that $(q, z_1, \ldots, z_K)$ is feasible to D_M. By weak duality, we have that

$$z \geq q^\top b + \sum_{k=1}^{K} z_k = q^\top b + \sum_{k=1}^{K} r_k + \sum_{k=1}^{K} (z_k - r_k) = z + \sum_{k=1}^{K} (z_k - r_k). \qquad \square$$

7.3 Benders decomposition

Benders decomposition is an alternative decomposition method, suitable for settings in which *complicating variables* are present. Differently than the Dantzig-Wolfe decomposition, the method presume from the get-go the employment of delayed constraint generation.

Benders decomposition has made significant inroads into practical applications. Not only it is extensively used in problems related to multi-period decision making under uncertainty, but it is also available in the commercial solver CPLEX to be used directly, only requiring the user to indicate (or annotate) which are the complicating variables.

Once again, let the problem P be defined as

$$\begin{aligned}(P): \min_{x,y}\ & c^\top x + \sum_{k=1}^{K} f_k^\top y_k \\ & Ax = b \\ & C_k x + D_k y_k = e_k,\ k \in \{1,\dots,K\} \\ & x \geq 0, y_k \geq 0,\ k \in \{1,\dots,K\}\,.\end{aligned}$$

Notice that this is equivalent to the problem P' presented in Section 7.1, but with a notation modified to make it easier to track how the terms are separated in the process. Again, we can see that P has a set of complicating variables x, which becomes obvious when we recast the problem as

$$\begin{array}{ccccccccccc} c^\top x & + & f_1^\top y_1 & + & f_2^\top y_2 & + & \dots & + & f_k^\top y_k & & \\ Ax & & & & & & & & & = & b \\ C_1 x & + & D_1 y_1 & & & & & & & = & e_1 \\ C_2 x & & & + & D_2 y_2 & & & & & = & e_2 \\ \vdots & & \vdots & & & & \ddots & & & & \vdots \\ C_K x & & & & & & & + & D_k y_k & = & e_K \\ x & & y_1 & & y_2 & & \dots & & y_k & \geq & 0 \end{array}$$

This structure is sometimes referred to as block-angular, referring to the initial block of columns on the left (as many as there are components in x) and the diagonal structure representing the elements associated with the variables y. In this case, notice that if the variable x were to be removed, or fixed to a value $x = \overline{x}$, the problem becomes separable in K independent parts

$$\begin{aligned}(S_k): \min_{y}\ & f_k^\top y_k \\ \text{s.t.: } & D_k y_k = e_k - C_k \overline{x} \\ & y_k \geq 0.\end{aligned}$$

Notice that these subproblems $k \in \{1,\dots,K\}$ can be solved in parallel and, in certain contexts, might even have analytical closed-form solutions. The part missing is the development of a coordination mechanism that would allow for iteratively update the solution $\overline{x}$ based on information emerging from the solution of the subproblems $k \in \{1,\dots,K\}$.

To see how that can be achieved, let us reformulate P as

$$\begin{aligned}(P_R): \min_{x}\ & c^\top x + \sum_{k=1}^{K} z_k(x) \\ \text{s.t.: } & Ax = b \\ & x \geq 0.\end{aligned}$$

where, for $k \in \{1,\dots,K\}$,

$$z_k(x) = \min_{y}\ \left\{ f_k^\top y_k : D_k y_k = e_k - C_k x \right\}.$$

Note that obtaining $z_k(x)$ requires solving S_k, which, in turn, depends on x. To get around this difficulty, we can rely on linear programming duality, combined, once again, with the resolution theorem (Theorem 6.8).

First, let us consider the dual formulation of the subproblems $k \in \{1, \ldots, K\}$, which is given by

$$\begin{aligned}(S_k^D) : z_k^D = \max.\ & p_k^\top (e_k - C_k x) \\ \text{s.t.: } & p_k^\top D_k \leq f_k.\end{aligned}$$

The main advantage of utilising the equivalent dual formulation is to "move" the original decision variable x to the objective function, a trick that will present its benefits shortly. Next, let us denote the feasibility set of S_k^D as

$$P_k = \left\{p : p^\top D_k \leq f_k\right\}, \forall k \in \{1, \ldots, K\}, \tag{7.6}$$

and assume that each $P_k \neq \emptyset$ with at least one extreme point. Relying on the resolution theorem, we can define the sets of extreme points of P_k, given by p_k^i, $i \in I_k$, and extreme rays w_k^r, $r \in R_k$.

As we assume that $P_k \neq \emptyset$, two cases can occur when we solve S_k^D, $k \in \{1, \ldots, K\}$. Either S_k^D is unbounded, meaning that the relative primal subproblem is infeasible, or S_k^D is bounded, meaning that $z_k^D < \infty$.

From the first case, we can use Theorem 6.6 to conclude that primal feasibility (or a bounded dual value $z_k^D < \infty$) can only be attained if and only if

$$(w_k^r)^\top (e_k - C_k x) \leq 0,\ \forall r \in R_k. \tag{7.7}$$

Furthermore, we know that if S_k^D has a solution, that must lie on a vertex of P_k. So, having the available the set of all extreme vertices p_k^i, $i \in I_k$, we have that if one can solve S_k^D, it can be equivalently represented as

$$(S_k^D) : z_k(x) = \max_{i \in I_k} (p_k^i)^\top (e_k - C_k x),$$

which can be equivalently reformulated as

$$\text{min.}\ \ \theta_k \tag{7.8}$$

$$\text{s.t.: } \theta_k \geq (p_k^i)^\top (e_k - C_k x),\ \forall i \in I_k. \tag{7.9}$$

Combining (7.7)–(7.9), we can reformulate P_R into a single-level equivalent form

$$\begin{aligned}(P_R) : \min_x.\ & c^\top x + \sum_{k=1}^{K} \theta_k \\ \text{s.t.: } & Ax = b \\ & (p_k^i)^\top (e_k - C_k x) \leq \theta_k,\ \forall i \in I_k, \forall k \in \{1, \ldots, K\} && (7.10) \\ & (w_k^r)^\top (e_k - C_k x) \leq 0,\ \forall r \in R_k, \forall k \in \{1, \ldots, K\} && (7.11) \\ & x \geq 0.\end{aligned}$$

Notice that, just like the reformulation used for the Dantzig-Wolfe method presented in Section 7.2, the formulation of P_R is of little practical use, since it requires the complete enumeration of (a typically prohibitive) number of extreme points and rays and is likely to be computationally intractable due to the large number of associated constraints. To address this issue, we can employ delayed constraint generation and iteratively generate only the constraints we observe to be violated.

By doing so, at a given iteration l, we have at hand a *relaxed main problem* P_M^l, which comprises only some of the extreme point and rays obtained until iteration l. The relaxed main problem can be stated as

$$\begin{aligned}(P_M^l) : z_{P_M}^l = \min_x \; & c^\top x + \sum_{k=1}^{K} \theta_k \\ \text{s.t.: } & Ax = b \\ & (p_k^i)^\top (e_k - C_k x) \leq \theta_k, \;\; \forall i \in I_k^l, \forall k \in \{1, \ldots, K\} \\ & (w_k^r)^\top (e_k - C_k x) \leq 0, \;\; \forall r \in R_k^l, \forall k \in \{1, \ldots, K\} \\ & x \geq 0,\end{aligned}$$

where $I_k^l \subseteq I_k$ represent subsets of extreme points p_k^i of P_k, and $R_k^l \subseteq R_k$ subsets of extreme rays w_k^r of P_k.

One can iteratively obtain these extreme points and rays from the subproblems S_k, $k \in \{1, \ldots, K\}$. To see that, let us fist define that, at iteration l, we solve the the main problem P_M^l and obtain a solution

$$\underset{x,\theta}{\operatorname{argmin}} \left\{ P_M^l \right\} = (\overline{x}^l, \overline{\theta}_1^l, \ldots \overline{\theta}_K^l).$$

We can then solve the subproblems S_k^l, $k \in \{1, \ldots, K\}$, for that fixed solution $\overline{x}^l$ and then observe if we can find additional constraints that were to be violated if they had been in the relaxed main problem in the first place. In other words, we can identify if the solution $\overline{x}^l$ allows for identifying additional extreme points p_k^i or extreme rays w_k^r of P_k that were not yet included in P_M^l.

To identify those, first recall that the subproblem (in its primal form), is given by

$$\begin{aligned}(S_k^l) : \text{min.} \;\; & f^\top y \\ \text{s.t.: } & D_k y_k = e_k - C_k \overline{x}^l \\ & y_k \geq 0.\end{aligned}$$

Then, two cases can lead to generating violated constraints that must be added to the relaxed primal problem to form P_M^{l+1}. The first is when S_k^l is feasible. In that case, a dual optimal basic feasible solution p_k^{il} is obtained. If $(p_k^{il})^\top (e_k - C_k \overline{x}^l) > \overline{\theta}_k^l$, then we can conclude that we just formed a violated constraint of the form of (7.10). The second case is when S_k^l is infeasible, then an extreme ray w_k^{rl} of P_k is available, such that $(w_k^{rl})^\top (e_k - C_k \overline{x}^l) > 0$, violating (7.11).

Notice that the above can also be accomplished by solving the dual subproblems S_k^D, $k \in \{1, \ldots, K\}$, instead. In that case, the extreme point p_k^{il} is immediately available and so are the extreme rays w_k^{rl} in case of unboundedness.

Algorithm 6 presents a pseudocode for the Benders decomposition. Notice that the method can benefit in terms of efficiency from the use of dual simplex, since we are iteratively adding violated constraints to the relaxed main problem P_M^l. Likewise, the subproblem S_k^l has only the independent terms being modified at each iteration and, in light of the discussion in Section 6.1.3, can also benefit from the use of dual simplex. Furthermore, the loop represented by Line 4 can be parallelised to provide further computational performance improvements.

Algorithm 6 Benders decomposition

initialise. Let $P_i^l = W_j^l = \emptyset$, for $k \in \{1, \ldots, K\}$, and set $l \leftarrow 0$.
repeat
  solve P_M^l to obtain $\left(\overline{x}^l, \left\{\overline{\theta}_k^l\right\}_{k=1}^K\right)$.
  for $k \in \{1, \ldots, K\}$ **do**
    solve S_k^l.
    if S_k^l is infeasible **then**
      obtain extreme ray w_j^k and make $W^l = W^l \cup \left\{w_j^k\right\}$.
    else
      obtain extreme point p_i^k and $P^l = P^l \cup \left\{p_i^k\right\}$
    end if
  end for
  $l = l + 1$.
until $(p_k^i)^\top (e_k - C_k \overline{x}) \leq \overline{\theta}_k, \ \forall k \in \{1, \ldots, K\}$
return $\left(\overline{x}^l, \left\{\overline{\theta}_k^l\right\}_{k=1}^K\right)$

Notice that the algorithm terminates if no violated constraint is found. This in practice implies that $(p_k^i)^\top (e_k - C_k \overline{x}) \leq \overline{\theta}_k$ for all $k \in \{1, \ldots, K\}$, and thus $(\overline{x}, \{\overline{\theta}_k\}_{k=1}^K)$ is optimal for P. In a way, if one consider the dual version subproblem, S_k^D, one can notice that it is acting as an implicit search for values of p_k^i that can make $(p_k^i)^\top (e_k - C_k \overline{x})$ larger than $\overline{\theta}_k$, meaning that the current solution $\overline{x}$ violates 7.10 and is thus not feasible to P_M^l.

Also, every time one solves P_M^l, a dual (lower for minimisation) bound $LB^l = z_{P_M}^l$ is obtained. This is simply because the relaxed main problem is a relaxation of the problem P, i.e., it contains less constraints than the original problem P. A primal (upper) bound can also be calculated at every iteration, which allows for keeping track of the progress of the algorithm in terms of convergence and preemptively terminate it at any arbitrary optimality tolerance. That can be achieved by setting

$$\begin{aligned} UB^l &= \min \left\{ UB^{l-1}, c^\top \overline{x}^l + \sum_{k=1}^K f^\top \overline{y}_k^l \right\} \\ &= \min \left\{ UB^{l-1}, z_{P_M}^l - \sum_{k=1}^K \overline{\theta}_k^l + \sum_{k=1}^K z_k^{Dl} \right\}, \end{aligned}$$

where $(\overline{x}^l, \left\{\overline{\theta}_k^l\right\}_{k=1}^K) = \text{argmin}_{x,\theta} \left\{P_M^l\right\}$, $\overline{y}_k^l = \text{argmin}_y \left\{S_k^l\right\}$, and z_k^{Dl} is the objective function value of the dual subproblem S_k^D at iteration l. Notice that, differently from the lower bound LB^l, there are no guarantees that the upper bound UB^l will decrease monotonically. Therefore, one must compare the bound obtained at a given iteration l using the solution $(\overline{x}^l, \overline{y}_1^l, \ldots, \overline{y}_K^l)$ against an incumbent (or best-so-far) bound UB^{l-1}.

7.4 Exercises

Exercise 7.1: Dantzig-Wolfe decomposition

Consider the following linear programming problem:

$$
\begin{array}{lrrrrrrr}
\text{min.} & & -x_{12} & & & -x_{22} & -x_{23} & \\
\text{s.t.:} & x_{11} & +x_{12} & +x_{13} & & & & = 20 \\
 & & & & x_{21} & +x_{22} & +x_{23} & = 20 \\
 & -x_{11} & & & -x_{21} & & & = \text{-}20 \\
 & & -x_{12} & & & -x_{22} & & = \text{-}10 \\
 & & & -x_{13} & & & -x_{23} & = \text{-}10 \\
 & x_{11} & & & & & +x_{23} & \leq 15 \\
 & x_{11,} & x_{12,} & x_{13,} & x_{21,} & x_{22,} & x_{23} & \geq 0
\end{array}
$$

We wish to solve this problem using Dantzig-Wolfe decomposition, where the constraint $x_{11}+x_{23} \leq 15$ is the only "coupling" constraint and the remaining constraints define a single subproblem.

(a) Consider the following two extreme points for the subproblem:

$$x^1 = (20, 0, 0, 0, 10, 10),$$

and

$$x^2 = (0, 10, 10, 20, 0, 0).$$

Construct a main problem in which x is constrained to be a convex combination of x^1 and x^2. Find the optimal primal and dual solutions for the main problem.

(b) Using the dual variables calculated in part a), formulate the subproblem and find its optimal solution.

(c) What is the reduced cost of the variable λ_3 associated with the extreme point x^3 obtained from solving the subproblem in part b)?

(d) Compute a lower bound on the optimal cost.

Exercise 7.2: Benders decomposition

Consider a wholesaler company planning to structure its supply chain to the retailers of a given product. The company needs to distribute the production from many suppliers to a collection of distribution points from which the retailers can collect as much product as they need for a certain period. By default, a pay-as-you-consume contract between wholesaler and retailers is signed and, therefore, the demand at each point is unknown at the moment of shipping. Consider there is no penalty for any unfulfilled demand and any excess must be discarded from one period to the other. The following parameters are given:

- B_i: production cost at supplier i
- C_i: production capacity at supplier i
- D_{js}: total orders from distribution point j in scenario s

- T_{ij}: transportation cost between i and j
- R_j: revenue for sale at distribution point j
- W_j: disposal cost at distribution point j

Let the variables be:

- p_i: production at supplier i
- t_{ij}: amount of products transported between i and j
- l_{js}: amount of products sold from the distribution point j in scenario s
- w_{js}: amount of products discarded from the distribution point j in scenario s
- r_j: amount pre-allocated in the distribution point j

The model for minimising the cost (considering revenue as a negative cost) is given below,

$$
\begin{aligned}
\text{min. } & \sum_{i \in I} B_i p_i + \sum_{i \in I, j \in J} T_{ij} t_{ij} + \sum_{s \in S} P_s \left(\sum_{j \in J} (-R_j l_{js} + W_j w_{js}) \right) \\
\text{s.t.: } & p_i \leq C_i, \ \forall i \in I \\
& p_i = \sum_{j \in J} t_{ij}, \ \forall i \in I \\
& r_j = \sum_{i \in I} t_{ij}, \ \forall j \in J \\
& r_j = l_{js} + w_{js}, \ \forall j \in J, \forall s \in S \\
& l_{js} \leq D_{js}, \ \forall j \in J, \forall s \in S \\
& p_i \geq 0, \ \forall i \in I \\
& r_j \geq 0, \ \forall j \in J \\
& t_{ij} \geq 0, \ \forall i \in I, \forall j \in J \\
& l_{js}, w_{js} \geq 0, \ \forall j \in J, \forall s \in S.
\end{aligned}
$$

Solve an instance of the wholesaler's distribution problem proposed using Benders decomposition.

Chapter 8

Integer programming models

8.1 Types of integer programming problems

In this chapter, we will consider problems in which we have the presence of integer variables. As we will see in the next chapters, the inclusion of integrality requirements impose further computational and theoretical challenges that we must overcome to be able to solve these problems. On the other hand, being able to consider integer variables allows for the modelling of far more complex and sophisticated systems, to an extent that integer programming problems, or more specifically, mixed-integer programming problems, are by far the most common in practice.

As we did in the first chapter, we will stat our discussion presenting general problems that have particular structures which can often be identified in larger and more complex settings. This will also be helpful in exemplifying how integer variables can be used to model particular features of optimisation problems.

Let us first specify what do we mean by an integer programming problem. Our starting point is a linear programming problem:

$$\begin{aligned} (\mathrm{P}) : \min.\ & c^\top x \\ \text{s.t.: } & Ax \leq b \\ & x \geq 0, \end{aligned}$$

where A is an $m \times n$ matrix, c an n-dimensional vector, b an m-dimensional vector, and x a vector of n decision variables.

We say that P is an integer programming problem if the variables x must take integer values, i.e., $x \in \mathbb{Z}^n$. If the variables are further constrained to be binary, i.e., $x \in \{0,1\}^n$, we say that it is a binary programming problem. Perhaps the most common setting is when only a subset of the variables are constrained to be integer or binary (say p of them), $x \in \mathbb{R}^{n-p} \times \mathbb{Z}^p$. This is what is referred to as *mixed-integer programming*, or MIP. The most common setting for integer programming problems is to have binary variables only, or a combination of binary and continuous variables.

One important distinction must be made. A closely related concept is that of combinatorial optimisation problems, which refer to problems of the form

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j : S \in \mathcal{F} \subseteq N \right\},$$

where c_j, $j \in N$, is a weight, and $\mathcal{F}$ is a family of feasible subsets. As the name suggests, in these problems we are trying to form combinations of elements such that a measure (i.e., an objective

function) is optimised. Integer programming happens to be an important framework for expressing combinatorial optimisation problems, though both integer programming and combinatorial optimisation expand further to other settings as well. To see this connection, let us define an *incidence vector* x^S of S such that

$$x_j^S = \begin{cases} 1, \text{if } j \in S \\ 0, \text{otherwise.} \end{cases}$$

Incidence vectors will permeate many of the (mixed-)integer programming formulations we will see. Notice that, once x^S is defined, the objective function simply becomes $\sum_{j \in N} c_j x_j$. Integer programming formulations are particularly suited for combinatorial optimisation problems when $\mathcal{F}$ can be represented by a collection of linear constraints.

8.2 (Mixed-)integer programming applications

We will consider now a few examples of integer and mixed-integer programming models with somewhat general structure. As we will see, many of these examples have features that can be combined into more general models.

8.2.1 The assignment problem

Consider the following setting. Assume that we must execute n jobs, which must be assigned to n distinct workers. Each job can be assigned to a worker only, and, analogously, each worker can only be assigned to one job. Assigning a worker i to a job j costs C_{ij}, which could measure, e.g., the time taken by worker i to execute job j. Our objective is to find a minimum cost assignment between workers and jobs. Figure 8.1a illustrates all possible worker-job assignments as arcs between nodes representing workers on the left and jobs on the right. Figure 8.1b represents one possible assignment.

Figure 8.1: An illustration of all potential assignments as a graph and an example of one possible assignment, with total cost $C_{12} + C_{31} + C_{24} + C_{43}$

To represent the problem, let $x_{ij} = 1$ if worker i is assigned to job j and 0, otherwise. Let $N = \{1, \ldots, n\}$ be a set of indices of workers and jobs (we can use the same set since they are of same number). The integer programming model that represent the assignment problem is given

by

$$
\begin{aligned}
(AP) : \text{min.} \quad & \sum_{i \in N} \sum_{j \in N} C_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{j \in N} x_{ij} = 1, \ \forall i \in N \\
& \sum_{i \in N} x_{ij} = 1, \ \forall j \in N \\
& x_{ij} \in \{0, 1\}, \ \forall i, \forall j \in N.
\end{aligned}
$$

Before we proceed, let us make a parallel to combinatorial problems. Clearly, the assignment problem is an example of a combinatorial problem, which can be posed by making N the set of all job-worker pairs (i, j), $S \in \mathcal{F}$ the (i, j) pairs in which i and j appear in exactly one pair, and, finally, x^S such that $x_{ij}, i, j = 1, \ldots, n$. Thus, the assignment problem is an example of a combinatorial optimisation problem that can be represented as an integer programming formulation.

8.2.2 The knapsack problem

The knapsack problem is another combinatorial optimisation problem that arise in several applications. Consider that we have a collection of n items from which we must make a selection. Each item has associated a cost A_i (e.g., weight) and our selection must be such that the total cost associated with the selection does not exceed a budget B (e.g., weight limit). Each item has also a value C_i associated, and our objective is to find a maximimum-valued selection selection of items that does not exceed the budget.

To model that, let us define $x_i = 1$, if item i is selected and 0, otherwise. Let $N = \{1, \ldots, n\}$ be the set of items. Then, the knapsack problem can be represented as the following integer programming problem

$$
\begin{aligned}
(KP) : \max_x \quad & \sum_{i=1}^{n} C_i x_i \\
\text{s.t.:} \quad & \sum_{i=1}^{n} A_i x_i \leq B \\
& x_i \in \{0, 1\}, \ \forall i \in N.
\end{aligned}
$$

Notice that the knapsack problems has variants in which an item can be selected a certain number of times, or that multiple knapsacks must be considered simultaneously, both being generalisations of KP.

Also, the knapsack problem is also a combinatorial optimisation problem, which can be stated by making N the set of all items $\{1, \ldots, n\}$, $S \in \mathcal{F}$ the subset of items with total cost not greater than B, and x^S such that $x_i, \forall i \in N$.

8.2.3 The generalised assignment problem

The generalised assignment problem (or GAP) is a generalisation of the assignment problem including a structure that resembles that of a knapsack problem. In this case, we consider the notion of bins, to each the items have to be assigned to. In this case, multiple items can be assigned to a

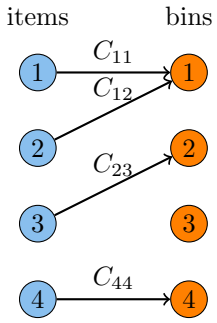


Figure 8.2: An example of a bin packing with total cost $C_{11} + C_{12} + C_{23} + C_{44}$

bin, or a bin might have no item assigned. In some context, this is problem is also known as the bin packing problem.

In this case, we would like to assign all of the m items to n bins, observing that the capacity B of each bin cannot be exceeded by the weights A_i of the items assigned to that bin. We know that assigning the item $i = 1, \ldots, m$ to the bin $j = 1, \ldots, n$ costs C_{ij}. Our objective is to obtain a minimum-cost bin assignment (or packing) that does not exceed any of the bin capacities. Figure 8.2 illustrates a possible assignment of items to bins. Notice that the number of total bins does not necessarily need to be the same the number of items.

To formulate the generalised assignment problem as an integer programming problem, let us define $x_{ij} = 1$, if item i is packed into bin j, and $x_{ij} = 0$ otherwise. Moreover, let $M = \{1, \ldots, m\}$ be the set of items and $N = \{1, \ldots, n\}$ be the set of bins. Then, the problem can be formulated as follows.

$$
\begin{aligned}
(GAP) : \min_{x} \quad & \sum_{i \in M} \sum_{j \in N} C_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{j=1}^{n} x_{ij} = 1, \forall i \in M \\
& \sum_{i=1}^{m} A_i x_{ij} \leq B_j, \forall j \in N \\
& x_{ij} \in \{0, 1\}, \forall i \in M, \forall j \in N.
\end{aligned}
$$

Hopefully, the parallel to the combinatorial optimisation problem version is clear at this point and is let for the reader as a thought exercise.

8.2.4 The set covering problem

Set covering problems relate to the location of infrastructure with the objective of covering demand points and, thus, a frequently recurring in settings where service centres such as fire brigades, hospitals, or police stations must be located to efficiently serve locations.

Let $M = \{1, \ldots, m\}$ be a set of regions that must be served by opening *service centres*. A centre can be opened at any of the $N = \{1, \ldots, n\}$ possible locations. If a centre is opened at location $J \in N$, then it serves (or covers) a subset of regions $S_j \subseteq M$ and has associated opening cost C_j.

Our objective is to decide where to open the centres so that all regions are served and the total opening cost is minimised.

Figure 8.3 illustrates an example of a set covering problem based on a fictitious map broken into regions. Each of the cells represents a region that must be covered, i.e., $M = \{1, \ldots, 20\}$. The blue cells represent regions that can have a centre opened, that is, $N = \{3, 4, 7, 11, 12, 14, 19\}$. Notice that $N \subset M$. In this case, we assume that if a centre is opened at a blue cell, then it can serve the respective cell and all adjacent cells. Therefore, we have, e.g., that $S_3 = \{1, 2, 3, 8\}$, $S_4 = \{2, 4, 5, 6, 7\}$, and so forth.

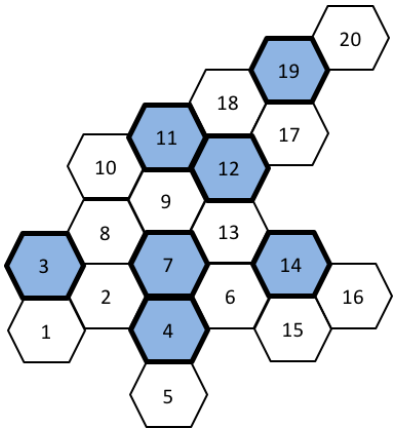


Figure 8.3: The hive map illustrating the set covering problem. Our objective is to cover all of the regions while minimising the total cost incurred by opening the centres at the blue cells

To model the set covering problem, and pretty much any other problem involving indexed subsets such as S_j, $\forall j \in N$, we need an auxiliary structure, often referred to as *0-1 incidence matrix* A such that

$$A = \begin{cases} A_{ij} = 1, \text{ if } i \in S_j, \\ A_{ij} = 0, \text{ otherwise.} \end{cases}$$

For example, referring to Figure 8.3, the first column of A would refer to $j = 3$ and would have nonzero values at rows 1, 2, 3, and 8.

We are now ready to pose the set covering problem as an integer programming problem. For that, let $x_j = 1$ if facility is opened at location j and $x_j = 0$, otherwise. In addition, let $M = \{1, \ldots, m\}$ be the set of regions to be served and $N = \{1, \ldots, n\}$ the set of candidate places to have a centre opened. Then, the set covering problem can be formulated as follows.

$$\begin{aligned} (SCP): \min_{x} \quad & \sum_{j \in N} C_j x_j \\ \text{s.t.:} \quad & \sum_{j \in N} A_{ij} x_j \geq 1, \forall i \in M \\ & x_j \in \{0, 1\}, \forall j \in N. \end{aligned}$$

As a final note, notice that this problem can also be posed as a combinatorial optimisation problem

of the form

$$\min_{T \subseteq N} \left\{ \sum_{j \in T} C_j : \bigcup_{j \in T} S_j = M \right\},$$

in which the locations $j \in T$ can be represented as an incidence vector, as before.

8.2.5 Travelling salesperson problem

The traveling salesperson problem (TSP) is one of the most famous combinatorial optimisation problems, perhaps due to its interesting mix of simplicity while being computationally challenging. Assume that we must visit a collection of n cities at most once, and return to our initial point, forming a so-called *tour*. When travelling from a city i to a city j, we incur in the cost C_{ij}, representing, for example, distance, or time. Our objective is to minimise the total cost of our tour. Notice that this is equivalent to finding the minimal cost permutation of $n-1$ cities, discarding that representing our starting and end point. Figure 8.4 illustrates a collection of cities and one possible tour.

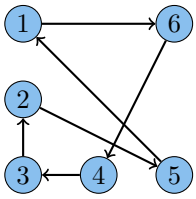


Figure 8.4: An example of a tour between the six cities

To pose the problem as an integer programming model, let us define $x_{ij} = 1$ if city j is visited directly after city i, $x_{ij} = 0$ otherwise. Let $N = \{1, \ldots, n\}$ be the set of cities. We assume that x_{ii} is not defined for $i \in N$. A naive model for the travelling salesperson problem would be

$$\begin{aligned}
(TSP) : \min_{x} \quad & \sum_{i \in N} \sum_{j \in N} C_{ij} x_{ij} \\
\text{s.t.:} \quad & \sum_{j \in N \setminus \{i\}} x_{ij} = 1, \forall i \in N \\
& \sum_{i \in N \setminus \{j\}} x_{ij} = 1, \forall j \in N \\
& x_{ij} \in \{0, 1\}, \forall i, \forall j \in N : i \neq j
\end{aligned}$$

First, notice that the formulation of TSP is exactly the same as that of the assignment problem, however, this formulation has an issue. Although it can guarantee that all cities are only visited once, it cannot enforce an important feature of the problem that is that the tour cannot present disconnections, i.e., contain *sub-tours*. In other words, the salesperson must physically visit from city to city in the tour, and cannot "teletransport" from one city to another. Figure 8.5 illustrate the concept of sub-tours.

In order to prevent subtours, we must include constraints that can enforce the full connectivity of the tour. There are mainly two types of such constraints. The first is called *cut-set constraints*

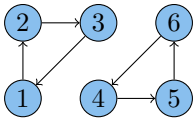


Figure 8.5: A feasible solution of for the naive TSP model. Notice the two sub-tours formed

and are defined as

$$\sum_{i \in S} \sum_{j \in N \setminus S} x_{ij} \geq 1, \forall S \subset N, 2 \leq |S| \leq n-1.$$

The cut-set constraints act by guaranteeing that among any subset of nodes $S \subseteq N$ there is always at least one arc (i, j) connecting one of the nodes in S and a node not in S.

An alternative type of constraint is called *subtour elimination* constraint and is of the form

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subset N, 2 \leq |S| \leq n-1.$$

Differently than the cutset constraints, the subtour elimination constraints prevent the cardinality of the nodes in each subset to match the cardinality of arcs within the same subset.

For example, consider the subtours illustrated in Figure 8.5 and assume that we would like to prevent the subtour formed by $S = \{1, 2, 3\}$. Then the cutset constraint would be of the form

$$x_{14} + x_{24} + x_{34} + x_{15} + x_{25} + x_{35} + x_{16} + x_{26} + x_{36} \geq 1$$

while the subtour elimination would be of the form

$$x_{12} + x_{13} + x_{21} + x_{23} + x_{31} + x_{32} \leq 2$$

There are some differences between these two constraints and typically cutset constraints are preferred for being stronger (we will discuss the notion of stronger constraints in the next chapters). In any case, either of them suffer from the same problem: the number of such constraints quickly becomes computationally prohibitive as the number of nodes increase. This is because one would have to generate a constraint to each possible combination of sizes 2 to $n - 1$.

A possible remedy to this consists of relying on delayed constraint generation. In this case, one can start from the naive formulation TSP and from the solution, observe whether there are torus formed. That being the case, only the constraints eliminating the observed subtours must be generated, and the problem can be warm-started. This procedure typically terminates far earlier than having all of the possible cutset or subtour elimination constraints generated.

8.2.6 Uncapacitated facility location

This is the first mixed-integer programming we consider. In this, we would like to design a network that can supply clients $i \in M$ by locating facilities among candidate locations $j \in N$. Openning a facility incurs in the fixed cost F_j, and serving a client $i \in M$ from a facility that has been located at $j \in N$ costs C_{ij}. Our objective is to design the most cost effective production and distribution network. That, is, we must decide where to locate facilities and how to serve clients (from these facilities) with minimum total (locating plus service) cost. Figure 8.6a illustrates an example of the problem with $M = \{1, \ldots, 8\}$ and $N = \{1, \ldots, 6\}$ and Figure 8.6b presents one possible

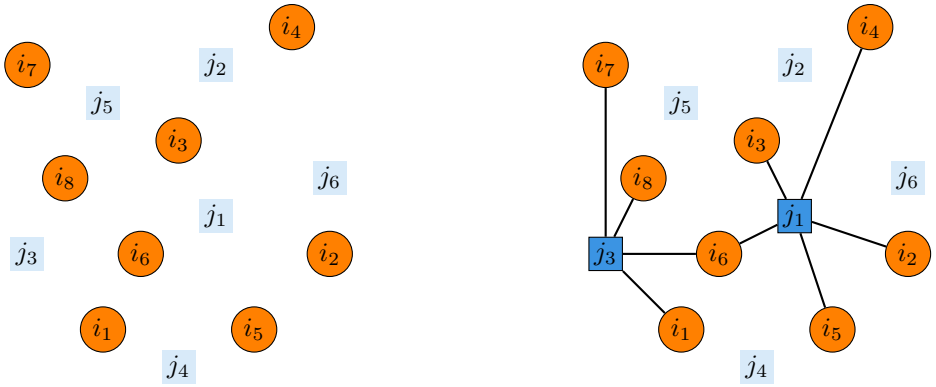


Figure 8.6: An illustration of the facility location problem an one possible solution with two facilities located

configuration with two facilities. The optimal number of facilities open and the client-facility association depends on the trade-offs between locating and service costs.

To formulate the problem as a mixed-integer programming problem, let us define x_{ij} as the fraction of the demand in $i \in M$ being served by a facility located at $j \in N$. In addition, we define the binary variable y_j such that $y_j = 1$, if a facility is located at $j \in N$ and 0, otherwise. With those, the uncapacitated facility location (or UFL) problem can be formulated as

$$(UFL): \min_{x,y} \quad \sum_{j \in N} F_j y_j + \sum_{i \in M} \sum_{j \in N} C_{ij} x_{ij} \tag{8.1}$$

$$\text{s.t.:} \quad \sum_{j \in N} x_{ij} = 1, \forall i \in M \tag{8.2}$$

$$\sum_{i \in M} x_{ij} \leq m y_j, \forall j \in N \tag{8.3}$$

$$x_{ij} \geq 0, \forall i \in M, \forall j \in N \tag{8.4}$$

$$y_j \in \{0, 1\}, \forall j \in N. \tag{8.5}$$

Some features of this model are worth highlighting. First, Notice that the absolute values associated with the demand at nodes $i \in M$ are somewhat implicitly represented in the cost parameter C_{ij}. This is an important modelling feature that allows the formulation to be not only stronger, but also more numerically favourable (avoiding large coefficients). Therefore, the demand is thought as being, at each node, 1 or 100%, and $0 \leq x_{ij} \leq 1$ represents the fraction of the demand at $i \in M$ being served by a facility eventually located at $j \in N$. Second, notice how the variable x_{ij} is only allowed to be greater than zero if the variable y_j is set to 1, due to (8.3). Notice that m, the number of clients, is acting as a maximum upper bound for the amount of demand being served from the facility j, which would be at most m when only one facility is located. That constraint is precisely the reason why the problem is called uncapacitated, since, in principle, there are no capacity limitations on how much demand is served from a facility.

Facility location problems are frequent in applications associated with supply chain planning problems, and can be specialised to a multitude of settings, including capacitated versions (both nodes and arcs), versions where the arcs must also be located (or built), having multiple echelons, and

so forth.

8.2.7 Uncapacitated lot-sizing

Another example of an important mixed-integer programming formulation is the uncapacited lot-sizing. In this, we would like to plan the production of a single product over a collection of time periods $T = \{1, \ldots, n\}$. We have encountered this problem before in Chapter 1, but now we consider the variation in which the production activity implies in a fixed cost F_t, representing, for example, a setup cost or the need for renting equipment. Once again, we incur on a cost C_t to produce one unit in period t and H_t is paid to store one unit of product from period t to period $t+1$.

Let us define $p_t \geq 0$ be the amount produced in period $t \in T$, and $s_t \geq 0$ the amount stored at the end of period $t \in T$. In addition, let $y_t \in \{0, 1\}$ indicate whether production occurs in period $t \in T$. Also, assume that M is a sufficiently large constraint. Then, the uncapacitated lot-sizing (ULS) can be formulated as

$$\begin{aligned}
(ULS): \min_{x,s,y} \quad & \sum_{t \in N} (F_t y_t + P_t p_t + H_t s_t) \\
\text{s.t.:} \quad & s_{t-1} + p_t = d_t + s_t, \forall t \in N \\
& p_t \leq M y_t, \forall t \in N \qquad (8.6) \\
& s_t, p_t \geq 0, \forall t \in N \\
& y_t \in \{0, 1\}, \forall t \in N.
\end{aligned}$$

Notice that the formulation of ULS is very similar to that seen in Chapter 1, with exception of the variable y_t, its associated fixed cost term $\sum_{t \in T} F_j y_j$ and the constraint (8.6). This constraint is precisely what renders the "uncapacitated" nomenclature, and is commonly known in the context of mixed-integer programming as *big-M constraints*. Notice that the constant M is playing the role of $+\infty$: it only really makes the constraint relevant when $y_t = 0$, so that $p_t \leq 0$ is enforced, making thus $p_t = 0$. However, this interpretation has to be taken carefully. Big-M constraints are known for being the cause of numerical issues and worsening the performance of mixed-integer programming solver methods. Thus, the value of M must be set such that it is the smallest value possible such that it does not artificially create a constraint. Finding these values are often challenging and instance dependent. In the capacitated case, M can trivially take the value of the production capacity.

8.3 Good formulations

In this section we discuss what makes a formulation a better formulation for a (mixed-)integer programming (MIP) problem. Just like it is the case with any mathematical programming application, there are potentially infinite possible ways to formulate the same problem. While in the case if linear programming (i.e., only continuous variables), alternative formulations typically do not lead to significant differences in terms of computational performance, the same is not true in the context of MIP. In fact, whether a MIP problem can be solved in a reasonable computational time often depend on having a good, or *strong*, formulation.

Therefore, it is fundamental that we can recognise which of alternatives formulations might yield a better computational performance. But first, we must be able to understand the source of these

differences. For that, the first thing to realise is that solution methods for MIP models rely on *successively solving linear programming models* called *linear programming (LP) relaxations*. How exactly this happens will be the subject of our next chapters. But for now, one can infer that a better formulation will be that that require the solution of less of such LP relaxations. And precisely, it so turns out that the number of LP relaxations that need to be solved (and, hence, performance) is strongly dependent on the *quality of the formulation*.

An LP relaxation simply consists of a version of the original MIP problem in which the integrality requirements are dropped. Most of the methods used to solve MIP models are based on LP relaxation.

There are several reasons why the employment of LP relaxations is a good strategy. First, we can solve LP problems efficiently. Secondly, the solution of the LP relaxation can be used to *reduce* the search space of the original MIP. However, simply rounding the solution of the LP relaxation will typically not lead to relevant solutions.

Let us illustrate the geometry of an integer programming model, such that the points we were discussing become more evident. Consider the problem

$$\begin{aligned}(P): \max_x.\ & x_1 + \frac{16}{25}x_2 \\ \text{s.t.:}\ & 50x_1 + 31x_2 \leq 250 \\ & 3x_1 + 31x_2 \geq -4 \\ & x_1, x_2 \in \mathbb{Z}_+.\end{aligned}$$

The feasible region of problem P is represented in Figure 8.7. First, notice how in this case the feasible region is not a polyhedral set anymore, but yet a collection of discrete points (represented in blue) that happen to be within the polyhedral set formed the linear constraints. This is one of its main complicating features, because the premise of convexity does not hold anymore. Another point can be noticed from Figure 8.7. Notice that rounding the solution obtained from the LP relaxation would in most cases lead to infeasible solutions, except when x_1 is rounded up and x_2 rounded down, which leads to the suboptimal solution $(2, 5)$. However, one can still graphically find the optimal solution using exactly the same procedure as that employed for linear programming problems, which would lead to the optimal integer solution $(5, 0)$.

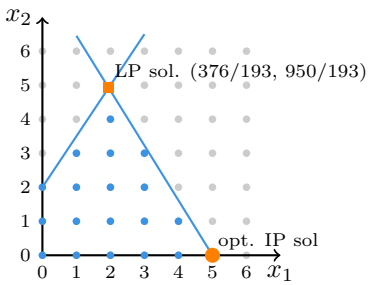


Figure 8.7: Graphical representation of the feasible region of the example

8.3.1 Comparing formulations

In order to be able to compare formulations, we require some specific definitions, including a precise definition of what is a *formulation*.

Definition 8.1. *A polyhedral set $P = \{x \in \mathbb{R}^{n+p} : Ax \leq b\}$ is a formulation for a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ if and only if $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.*

One aspect that can be noticed from Definition 8.1 is that the feasible region of an integer programming problem is a collection of points, represented by X. This is illustrated in Figure 8.8, where one can see three alternative formulations, P_1, P_2, and P_3 for the same set X.

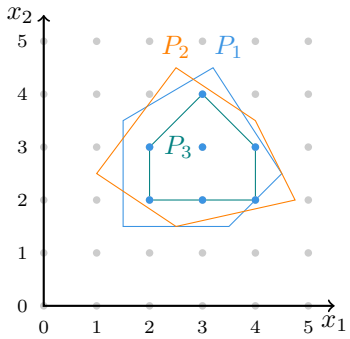


Figure 8.8: An illustration of three alternative formulations for X. Notice that P_3 is an ideal formulation, representing the convex hull of X.

The formulation P_3 has a special feature associated with it. Notice how all extreme points of P_3 belong to X. This has an important consequence. That implies that the solution of the original integer programming problem can be obtained by solving a single LP relaxation, since the solution of both problems is the same. This is precisely what characterises an *ideal* formulation, which is that leading to a minimal (i.e., only one) number of required LP relaxation solutions as solving an LP relaxation over an ideal P yields a solution $x \in X$ for any cost vector c. This will only be the case if the formulation P is the *convex hull* of X.

This is the case because of two important properties relating the set X and its convex hull $\mathbf{conv}(X)$. The first is that $\mathbf{conv}(X)$ is a polyhedral set and the second is that the extreme points of $\mathbf{conv}(X)$ belong to X. We summarise those two facts in Proposition 8.2, to which we will refer shortly. Notice that the proof for the proposition can be derived from Definition 2.7 and Theorem 2.8.

Proposition 8.2. $\mathbf{conv}(X)$ *is a polyhedral set and all its extreme points belong to* X*.*

If P is such that $P = \mathbf{conv}(X)$, the original problem

$$\text{min. } \left\{c^\top x : x \in X\right\},$$

can, in principle, be replaced with

$$\text{min. } \left\{c^\top x : x \in \mathbf{conv}(X)\right\},$$

where we have that

$$X = \left\{Ax \leq b, x \in \mathbb{Z}^n \times \mathbb{R}^p\right\} \text{ and } \mathbf{conv}(X) = \left\{Ax \leq b, x \in \mathbb{R}^{n+p}_+\right\}.$$

Unfortunately, this is often not the case. Typically, with exception of some specific cases, a description of $\mathbf{conv}(X)$ is not known and deriving it algorithmically require an exponential number or constraints. However, Proposition 8.2 allows us to define a structured way to compare formulations. This is summarised in Definition 8.3.

Definition 8.3. *Given a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^n$ and two formulations P_1 and P_2 for X, P_1 is a* better formulation *than P_2 if $P_1 \subset P_2$.*

Definition 8.3 gives us a framework to try to demonstrate that a given formulation is better than another. If we can show that $P_1 \subset P_2$, then, by definition (literally), P_1 is better formulation than P_2. Clearly, this is not a perfect framework, since, for example, it would not be useful for comparing P_1 and P_2 in Figure 8.8, and, in fact, there is nothing that can be said a priori about the two in terms of which will render the better performance. Often, in the context of MIP, this sort of analysis can only rely on careful computational experimentation.

A final point to make is that sometimes one must compare formulations of distinct dimensions, that is, with different number of variables. When that is the case, one can resort to projection, as a means to compare both formulations onto the same space of variables.

8.4 Exercises

Exercise 8.1: ULS formulations

Consider the following formulations $P^{(x,s,y)}_{\text{ULS-1}}$ and $P^{(w,y)}_{\text{ULS-2}}$ as relaxations for the ULS problem defined over $N = \{1, \ldots, n\}$ periods:

$$P^{(x,s,y)}_{\text{ULS-1}} = \left\{ \begin{array}{lll} (x,s,y): & s_{t-1} + x_t = d_t + s_t, & \forall t \in N \\ & x_t \leq M y_t, & \forall t \in N \\ & s_0 = 0, & \\ & s_t \geq 0, & \forall t \in N \\ & x_t \geq 0, & \forall t \in N \\ & 0 \leq y_t \leq 1, & \forall t \in N \end{array} \right\} \quad \left| \begin{array}{ll} x_t & \text{production in period } t \\ s_t & \text{stock in period } t \\ y_t & \text{setup in period } t \\ d_t & \text{demand in period } t \\ M & \text{maximum production} \\ & M = \sum_{t \in N} d_t \end{array} \right.$$

$$P^{(w,y)}_{\text{ULS-2}} = \left\{ \begin{array}{lll} (w,y): & \sum_{i=1}^{t} w_{it} = d_t, & \forall t \in N \\ & w_{it} \leq d_t y_i, & \forall i,t \in N : i \leq t \\ & w_{it} \geq 0, & \forall i,t \in N : i \leq t \\ & 0 \leq y_t \leq 1, & \forall t \in N \end{array} \right\} \quad \left| \begin{array}{ll} w_{it} & \text{production in period } i \\ & \text{to be used in period } t \\ y_t & \text{setup in period } t \end{array} \right.$$

(a) Use projection to show that $P^{(w,y)}_{\text{ULS-2}}$ is stronger than $P^{(x,s,y)}_{\text{ULS-1}}$, i.e, $P^{(w,y)}_{\text{ULS-2}} \subset P^{(x,s,y)}_{\text{ULS-1}}$.

Hint: First, construct an extended formulation $P^{(x,s,y,w)}_{\text{ULS-2}}$ by writing the variables x_t and s_t in terms of variables w_{it} and add them to $P^{(w,y)}_{\text{ULS-2}}$. Then, use projection to show that

$$\underset{x,s,y}{\mathbf{proj}} \left(P^{(x,s,y,w)}_{\text{ULS-2}} \right) \subseteq P^{(x,s,y)}_{\text{ULS-1}},$$

which is equivalent to $P^{(w,y)}_{\text{ULS-2}} \subseteq P^{(x,s,y)}_{\text{ULS-1}}$. Do this by verifying that each constraint of $P^{(x,s,y)}_{\text{ULS-1}}$ is satisfied by all $(x,s,y) \in P^{(x,s,y,w)}_{\text{ULS-2}}$. Finally, show that a solution $(\bar{x}, \bar{s}, \bar{y})$ with $\bar{x}_t = d_t$ and $\bar{y}_t = d_t/M$, for all $t \in N$, satisfies $(\bar{x}, \bar{s}, \bar{y}) \in P^{(x,s,y)}_{\text{ULS-1}} \setminus P^{(x,s,y,w)}_{\text{ULS-2}}$.

(b) The optimisation problems associated with the two ULS formulations are

$$
\begin{array}{lll}
\text{(ULS-1)} & \min\limits_{x,s,y} & \sum\limits_{t\in N}(f_t y_t + p_t x_t + q_t s_t) \\
& \text{s.t.:} & s_{t-1} + x_t = d_t + s_t, \qquad \forall t \in N \\
& & x_t \le M y_t, \qquad \forall t \in N \\
& & s_0 = 0, \\
& & s_t \ge 0, \qquad \forall t \in N \\
& & x_t \ge 0, \qquad \forall t \in N \\
& & 0 \le y_t \le 1, \qquad \forall t \in N
\end{array}
\quad\Bigg|\quad
\begin{array}{rl}
x_t & \text{production in period } t \\
s_t & \text{stock in period } t \\
y_t & \text{setup in period } t \\
d_t & \text{demand in period } t \\
M & \text{maximum production} \\
& M = \textstyle\sum_{t\in N} d_t
\end{array}
$$

$$
\begin{array}{lll}
\text{(ULS-2)} & \min\limits_{w,y} & \sum\limits_{t\in N}\left[f_t y_t + p_t \sum\limits_{i=t}^{n} w_{ti} + q_t \sum\limits_{i=1}^{t}\left(\sum\limits_{j=i}^{n} w_{ij} - d_i\right)\right] \\
& \text{s.t.:} & \sum\limits_{i=1}^{t} w_{it} = d_t, \qquad \forall t \in N \\
& & w_{it} \le d_t y_i, \qquad \forall i,t \in N : i \le t \\
& & w_{it} \ge 0, \qquad \forall i,t \in N : i \le t \\
& & 0 \le y_t \le 1, \qquad \forall t \in N
\end{array}
\quad\Bigg|\quad
\begin{array}{rl}
w_{it} & \text{production in period } i \\
& \text{to be used in period } t \\
y_t & \text{setup in period } t
\end{array}
$$

Consider a ULS problem instance over $N = \{1, \ldots, 6\}$ periods with demands $d = (6, 7, 4, 6, 3, 8)$, set-up costs $f = (12, 15, 30, 23, 19, 45)$, unit production costs $p = (3, 4, 3, 4, 4, 5)$, unit storage costs $q = (1, 1, 1, 1, 1, 1)$, and maximum production capacity $M = \sum_{i=1}^{6} d_j = 34$. Solve the problems ULS-1 and ULS-2 with Julia using JuMP to verify the result of part (a) computationally.

Exercise 8.2: TSP formulation - MTZ

Show that the following formulation P_{MTZ} is valid for the TSP defined on a directed graph $G = (N, A)$ with $N = \{1, \ldots, n\}$ cities and arcs $A = \{(i, j) : i, j \in N, i \ne j\}$ between cities.

$$
P_{MTZ} = \begin{cases}
\sum\limits_{j\in N\setminus\{i\}} x_{ij} = 1, & \forall i \in N \\
\sum\limits_{j\in N\setminus\{i\}} x_{ji} = 1, & \forall i \in N \\
u_i - u_j + (n-1)x_{ij} \le n-2, & \forall i,j \in N \setminus \{1\} : i \ne j \quad (*) \\
x_{ij} \in \{0,1\}, & \forall i,j \in N : i \ne j
\end{cases}
$$

where $x_{ij} = 1$ if city $j \in N$ is visited immediately after city $i \in N$, and $x_{ij} = 0$ otherwise. Constraints $(*)$ with the variables $u_i \in \mathbb{R}$ for all $i \in N$ are called *Miller-Tucker-Zemlin* (MTZ) subtour elimination constraints.

Hint: Formulation P_{MTZ} is otherwise similar to the formulation presented, except for the constraints $(*)$ which replace either the cutset constraints

$$\sum_{i\in S}\sum_{j\in N\setminus S} x_{ij} \ge 1, \ \forall S \subset N, S \ne \emptyset,$$

or the subtour elimination constraints

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1, \ \forall S \subset N, 2 \le |S| \le n-1,$$

which are used to prevent subtours in TSP solutions. Thus, you have to show that:

1. Constraints $(*)$ prevent subtours in any solution $x \in P_{MTZ}$.
2. Every TSP solution x (on the same graph G) satisfies the constraints $(*)$.

You can prove 1. by contradiction. First, assume that a solution $x \in P_{MTZ}$ has a subtour with k arcs $(i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i_1)$ and k nodes $\{i_1, \ldots, i_k\} \in N \setminus \{1\}$. Then, write the constraints $(*)$ for all arcs in this subtour and try to come up with a contradiction.

You can prove 2. by finding suitable values for each $u_2, \ldots, u_n$ that will satisfy the constraints $(*)$ for any TSP solution x. Recall that a TSP solution represents a *tour* that visits each of the $N = \{1, \ldots, n\}$ cities exactly once and returns to the starting city.

Exercise 8.3: TSP implementation

Solve a 15 node TSP instance using the formulation P_{MTZ} presented in Exercise 8.1. You can randomly generate city coordinates in x-y plane for all $N = \{1, \ldots, n\}$ cities. Letting c_{ij} denote the distance between cities i and j, the problem MTZ can be formulated as

$$
\begin{aligned}
(MTZ): \min_{x,u} \quad & \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} & & \\
\text{s.t.:} \quad & \sum_{j \in N \setminus \{i\}} x_{ij} = 1, & & \forall i \in N, \\
& \sum_{j \in N \setminus \{i\}} x_{ji} = 1, & & \forall i \in N, \\
& u_i - u_j + (n-1) x_{ij} \leq n - 2, & & \forall i, j \in N \setminus \{1\} : i \neq j, \\
& x_{ij} \in \{0, 1\}, & & \forall i, j \in N : i \neq j.
\end{aligned}
$$

Implement the model and solve the problem instance with Julia using JuMP.

Exercise 8.4: Scheduling problem

A set of $N = \{1, \ldots, n\}$ jobs must be carried out on a single machine that can do only one job at a time. Each job $j \in N$ takes p_j hours to complete. Given job weights w_j for all $j \in N$, in what order should the jobs be carried out so as to minimise the weighted sum of their start times? Formulate this scheduling problem as a mixed integer program.

Chapter 9

Branch-and-bound method

9.1 Optimality for integer programming problems

We will now discuss a method to solve mixed-integer programming problems that relies on the successive solution of linear programming relaxations. Although there are several methods that can be employed to solve combinatorial optimisation problems, most of them are not capable of providing optimality guarantees for the solution obtained (e.g., are heuristics or metaheuristics) or do not exploit the availability of a (linear) mathematical programming formulation. To date, the most widespread method that is capable of both is something that is generally known as a *branch-and-cut* method.

Branch-and-cut methods are composed of a combination of multiple parts, including, among other techniques, a branch-and-bound approach, cutting planes, and heuristics as well. In the next chapters we will focus on each of these parts individually, starting with the branch-and-bound method.

9.2 Relaxations

Before we present the method itself, let us discuss the more general concept of *relaxation*. We have visited the concept somewhat informally before, but now we will concentrate on a more concrete definition.

Consider an integer programming problem of the form

$$z = \min_{x} \left\{ c^\top x : x \in X \subseteq \mathbb{Z}^n \right\}.$$

To prove that a given solution x^* is optimal, we must rely on the notion of *bounding*. That is, we must provide a pair of upper and lower bounds that are as close (or tight) as possible. In the occasion that these bounds are the same, and thus match the value of $z = c^\top x^*$, we have available a certificate of optimality for x^*. This concept must be familiar to you. We already used a similar arguments in Chapter 5, when we introduced the notion of dual bounds.

Most methods that can prove optimality work by bounding an optimal solution. In this context, bounding means to construct an increasing sequence of lower bounds

$$\underline{z}_1 < \underline{z}_2 < \cdots < \underline{z}_s \leq z$$

and a decreasing sequence of upper bounds

$$\overline{z}_1 > \overline{z}_2 > \cdots > \overline{z}_t \geq z$$

to obtain as tight as possible lower ($\underline{z} \leq z$) and upper ($\overline{z} \geq z$) bounds. Notice that the process can be arbitrarily stopped when $\overline{z}_t - \underline{z}_s \leq \epsilon$, where s and t are some positive integers and $\epsilon > 0$ is a predefined (suitably small) tolerance. The term ϵ represents an *absolute optimality gap*, meaning that one can guarantee that the optimal value is at most greater than $\underline{z}$ by ϵ units and at most smaller than $\overline{z}$ by ϵ units. In other words, the optimal value must be either $\underline{z}$, $\overline{z}$, or a value in between.

This framework immediately poses the key challenge of deriving such bounds efficiently. It turns out that this is a challenge that goes beyond the context of mixed-integer programming problems. In fact, we have already seen this idea of bounding in Chapter 7, when we discussed decomposition methods, which also generate lower and upper bounds during their execution.

Regardless of the context, bounds are typically of two types: *primal* bounds, which are bounds obtained by evaluating a *feasible* solution (i..e, that satisfy primal feasibility conditions); and *dual* bounds, which are typically attained when a primal feasibility is allowed to be violated so that a dual feasible solution is obtained. In the context of minimisation, primal bounds are upper bounds (to be minimised), while dual bounds are lower bounds (to be maximised). Clearly, in the case of maximisation, the reverse holds.

Primal bounds can be obtained by means of a feasible solution. For example, one can heuristically assemble a solution that is feasible by construction. On the other hand, dual bounds are typically obtained by means of solving a *relaxation* of the original problem. We are ready now to provide Definition 9.1, which formally states the notion of a relaxation.

Definition 9.1 (Relaxation)**.** *A problem*

$$(RP) : z_{RP} = min. \ \left\{\overline{c}^\top x : x \in \overline{X} \subseteq \mathbb{R}^n\right\}$$

is a relaxation *of problem*

$$(P) : z = min. \ \left\{c^\top x : x \in X \subseteq \mathbb{R}^n\right\}$$

if $X \subseteq \overline{X}$*, and* $\overline{c}^\top x \leq c^\top x$*,* $\forall x \in X$*.*

Definition 9.1 provides an interesting insight related to relaxations: they typically comprise an expansion of the feasible region, possibly combined with an objective function bounding. Thus, two main strategies to obtain relaxations are to enlarge the feasible set by dropping constraints and replacing the objective function with another of same or smaller value. One might notice at this point that we have used a very similar argumentation setting to define linear programming duals in Chapter 5. We will return to the relationship between relaxations and Lagrangian duality in a more general setting in Part II, when we discuss the notion of Lagrangian relaxation.

Clearly, for relaxations to be useful in the context of solving mixed-integer programming problems, they have to be easier to solve than the original problem. That being the case, we can then rely on two important properties that relaxations have, which are crucial for using them as a means to generate dual bounds. These are summarised in Proposition 9.2 and 9.3.

Proposition 9.2. *If RP is a relaxation of P, then* z_{RP} *is a dual bound for* z*.*

Proof. For any optimal solution x^* of P, we have that $x^* \in X \subseteq \overline{X}$, which implies that $x^* \in \overline{X}$. Thus, we have that

$$z = c^\top x^* \geq \overline{c}^\top x^* \geq z_{RP}. \qquad \square$$

The first inequality is due to Definition 9.1 and the second is because x^* is simply a feasible solution, but not necessarily the optimal, for z_{RP}.

Proposition 9.3. *The following statements are true:*

1. *If a relaxation RP is infeasible, then P is infeasible.*
2. *Let x^* be an optimal solution for RP. If $x^* \in X$ and $\overline{c}^\top x^* = c^\top x^*$, then x^* is an optimal solution for P.*

Proof. To prove (1), simply notice that if $\overline{X} = \emptyset$, then $X = \emptyset$. To show (2), notice that as $x^* \in X$, $z \leq c^\top x^* = \overline{c}^\top x^* = z_{RP}$. From Proposition 9.2, we have $z \geq z_{RP}$. Thus, $z = z_{RP}$. □

9.2.1 Linear programming relaxation

In the context of solving (mixed-)integer programming problems, we will rely on the notion of linear programming (LP) relaxations. We have briefly discussed the idea in the previous chapter, but, for the sake of precision, let us first define what we man by the term.

Definition 9.4 (Linear programming (LP) relaxation)**.** *The LP relaxation of an integer programming problem min. $\left\{c^\top x : x \in P \cap \mathbb{Z}^n\right\}$ with $P = \left\{x \in \mathbb{R}^n_+ : Ax \leq b\right\}$ is the linear programming problem min. $\left\{c^\top x : x \in P\right\}$.*

Notice that an LP relaxation is indeed a relaxation, since we are enlarging the feasible region by dropping the integrality requirements while maintaining the same objective function (cf. Definition 9.1).

Let us consider a numerical example. Consider the integer programming problem

$$\begin{aligned} z = \max_x. \quad & 4x_1 - x_2 \\ \text{s.t.:} \quad & 7x_1 - 2x_2 \leq 14 \\ & x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x \in \mathbb{Z}^2_+. \end{aligned}$$

A dual (upper; notice the maximisation) bound for z can be obtained by solving its LP relaxation, which yields the bound $z_{LP} = 8.42 \geq z$. A primal (lower) bound can be obtained by choosing any of the feasible solutions (e.g., $(2, 1)$, to which $z = 7 \leq z$). This is illustrated in Figure 9.1.

We can now briefly return to the discussion about better (or stronger) formulations for integer programming problems. Stronger formulations are characterised by those that wield stronger relaxations, or, specifically, that yield relaxations that are guaranteed to provide better (or tighter) dual bounds. This is formalised in Proposition 9.5.

Proposition 9.5. *Let P_1 and P_2 be formulations of the integer programming problem*

$$\min_x. \ \left\{c^\top x : x \in X\right\} \ \text{with } X = P_1 \cap \mathbb{Z}^n = P_2 \cap \mathbb{Z}^n.$$

Assume P_1 is a better formulation than P_2 (i.e., $P_1 \subset P_2$). Let $z^i_{LP} = \min. \left\{c^\top x : x \in P_i\right\}$ for $i = 1, 2$. Then $z^1_{LP} \geq z^2_{LP}$ for any cost vector c.

Proof. Apply Proposition 9.2 by noticing that P_1 is a relaxation of P_2. □

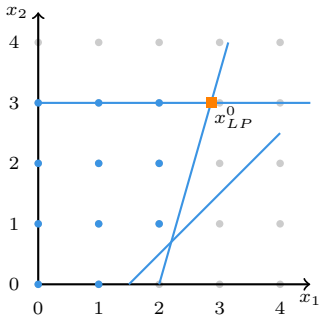


Figure 9.1: The feasible region of the example (represented by the blue dots) and the solution of the LP relaxation, with objective function value $z_{LP} = 8.42$

9.2.2 Relaxation for combinatorial optimisation

There is another important type of relaxation that is often exploited in the context of combinatorial optimisation. Specifically, a relaxation for a combinatorial optimisation problem that is also a combinatorial optimisation problem is called a *combinatorial relaxation*.

Efficient algorithms are known for some combinatorial optimisation problems, and this can be exploited in a solution method for problem to which the combinatorial relaxation happens to be one of such problems.

Let us illustrate the concept with a couple of example. Consider the travelling salesperson problem (TSP). Recall that, without considering the tour elimination constraints, we recover the assignment problem. It so turns out that the assignment problem can be solved efficiently (for example, using the so-called the Hungarian method) and thus can be used as a relaxation for the TSP, i.e.,

$$z_{TSP} = \min_{T \subseteq A} \left\{ \sum_{(i,j) \in T} c_{ij} : T \text{ forms a tour} \right\} \geq$$
$$z_{AP} = \min_{T \subseteq A} \left\{ \sum_{(i,j) \in T} c_{ij} : T \text{ forms an assignment} \right\}.$$

Still relating to the TSP, one can obtain even stronger combinatorial relaxations using *1-trees* for symmetric TSP. Let us first define some elements. Consider an undirected graph $G = (V, E)$ with edge (or arcs) weights c_e for $e \in E$. The objective is to find a minimum weight tour.

Now, notice the following: (i) a tour contains exactly two edges adjacent to the origin node (say, node 1) and a path through nodes $\{2, \ldots, |V|\}$; (ii) a tour is a special case of a *(spanning) tree*, which is any subset of edges that covers (or touch at least once) all nodes $v \in V$.

We can now define what is a 1-tree. A *1-tree* is a subgraph consisting of two edges adjacent to node 1 plus the edges of a tree on nodes $\{2, \ldots, |V|\}$. Clearly, every tour is a 1-tree with the additional requirement (or constraint) that every node has exactly two incident edges. Thus, the problem of finding minimal 1-trees is a relaxation for the problem of finding optimal tours. Figure 9.2 illustrates a 1-tree for an instance with eight nodes.

Once again, it so turns out that several efficient algorithms are known for forming minimal spanning

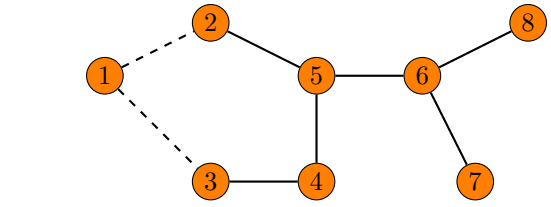


Figure 9.2: An example of a 1-tree considering eight nodes

trees, which can be efficiently utilised as a relaxation for the symmetric TSP, that is

$$z_{STSP} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a tour} \right\} \geq$$

$$z_{1-TREE} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a 1-tree} \right\}.$$

9.3 Branch-and-bound method

Relaxations play a major role in solving mixed-integer programming and/or combinatorial optimisation problems. However, they are only part of the framework (specifically, the bounding part of the branch-and-bound method). One still needs to be able to, from the solution of said relaxations, be able to construct a solution to the original problem.

Branch-and-bound is an algorithmic strategy that is far broader than mathematical programming and optimisation. In essence, it consists of a *divide-and-conquer* strategy, in which we first break an original problem into smaller and manageable (or solvable) parts and then recombine the solution of these parts into a solution for the original problem.

Specifically, let

$$P : z = \max_{x} \left\{ c^\top x : x \ \in S \right\}.$$

The working principle behind this strategy is based on the principle formalised in Proposition 9.6.

Proposition 9.6. *Let* $K = \{1, \ldots, |K|\}$ *and* $\bigcup_{k \in K} S_k = S$ *be a decomposition of* S*. Let* $z^k = \max_x \left\{ c^\top x : x \in S_k \right\}, \forall k \in K$*. Then*

$$z = \max_{k \in K} \left\{ z^k \right\}.$$

Notice the use of the word *decomposition* in Proposition 9.6. Indeed, the principle is philosophically the same, and this connection will be exploited in later chapters when we discuss in more detail the available technology for solving mixed-integer programming problems.

Now, one challenging aspect related with divide-and-conquer approaches is that, in order to be able to find a solution, one might need to repeat several times the strategy based on Proposition 9.6, leading to a multi-layered collection of subproblems. To address this issue, such methods typically rely on tree structures called *enumerative trees*, which are simply a representation that allows for keeping track the relationship (represented by branches) between subproblems (represented by nodes).

Figure 9.3 represents an enumerative tree for a generic problem $S \subseteq \{0,1\}^3$ in which one must define the value of a three-dimensional binary variable. The subproblems are formed by, at each level, fixing one of the components to zero or one, forming then two subproblems. Any strategy to form subproblems that generate two subproblems (or children) is called a *binary branching*.

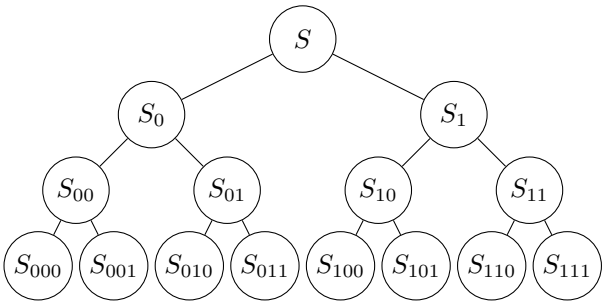


Figure 9.3: A enumeration tree using binary branching for a problem with three binary variables

Specifically, notice that, at the highest level, we have

$$S = S_0 \cup S_1 = \{x \in S : x_1 = 0\} \cup \{x \in S : x_1 = 1\},$$

which renders two subproblems. Then, each subproblem is again decomposed each into two children, such that

$$S_i = S_{i0} \cup S_{i1} = \{x \in S : x_1 = i, x_2 = 0\} \cup \{x \in S : x_1 = i, x_2 = 1\}.$$

Finally, once all of the variables are fixed, we arrive at what is called the *leaves* of the tree. These are such that they cannot be further divided, since they immediately yield a candidate solution for the original problem.

$$S_{ij} = S_{ij0} \cup S_{ij1} = \{x \in S : x_1 = i, x_2 = j, x_3 = 0\} \cup \{x \in S : x_1 = i, x_2 = j, x_3 = 1\}$$

Notice that applying Proposition 9.6, we can recover an optimal solution to the problem.

9.3.1 Bounding in enumerative trees

As you may suspect, the strategy of enumerating all possible solutions will quickly become computationally intractable and will most likely not be feasible for mixed-integer programming problems, or any relevant combinatorial optimisation for that matter. That is precisely when the notion of bounding comes to the spotlight: by possessing bound information on our original problem, we might be able to dismiss branches (or prune, in keeping with our tree analogy) from being searched, and hopefully find a solution without the need to exhaustively explore the enumeration tree.

The main principle behind the pruning of branches in enumerative search trees is summarised in Proposition 9.7.

Proposition 9.7. *Consider the problem P and let $S = \bigcup_{k \in K} S_k$ be a decomposition of S into smaller sets. Let $z^k = max._x \left\{c^\top x : x \in S_k\right\}$ for $k \in K$, and let $\overline{z}^k$ ($\underline{z}^k$) be an upper (lower) bound on z^k. Then $\overline{z} = \max_{k \in K} \left\{\overline{z}^k\right\}$ and $\underline{z} = \max_{k \in K} \left\{\underline{z}^k\right\}$.*

First, notice tat P is a maximisation problem, for which an upper bound is a dual bound, obtained from a relaxation, and a lower bound is a primal bound, obtained from a feasible solution.

Proposition 9.7 states that the best known primal (lower) bound can be applied *globally* to all of the subproblems S_k, $k \in K$. On the other hand, dual (upper) bounds can only be considered valid locally, since only the worst of the upper bounds can be guaranteed to hold globally.

Pruning branches is made possible by combining relaxations and global primal bounds. If, at any moment of the search, the solution of a relaxation of S_k is observed to be *worse* than a known global primal bound, then any further branching from that point onwards would be fruitless, since no solution found from that subproblem could be better than the relaxation for S_k. Specifically, we have that

$$\underline{z} \geq \overline{z}_k \geq \overline{z}_{k'}, \ \forall k' \text{ that is descendent of } k.$$

9.3.2 Linear-programming-based branch-and-bound

Branch-and-bound is the general nomenclature given to methods that operate based on solving relaxations of subproblems and using bounding information to preemptively prune branches in the enumerative search tree.

The characteristics that define a branch-and-bound method are thus the relaxation being solved (or how bounding is performed), and how subproblems are generated (how branching is performed). In the specific context of (mixed-)integer programming problems, bounding is performed utilising linear programming (LP) relaxations.

In regards to branching, we employ the following strategy utilising the information from the solution of the LP relaxation. At a given subproblem S_k, suppose we have an optimal solution with a fractional component $x_j^{*k} = \overline{x}_j \notin \mathbb{Z}^1$. We can then *branch* S_k into the following subproblems:

$$S_{k1} = S_k \cap \{x : x_j \leq \lfloor \overline{x_j} \rfloor\} \text{ and } S_{k2} = S_k \cap \{x : x_j \geq \lceil \overline{x_j} \rceil\}.$$

Notice that this implies that each of the subproblems will be disjunct (i.e., with no intersection) and have one additional constraint that eliminates the fractional part of the component around the solution of the LP relaxation.

Bounding can occur in three distinct ways. The first case is when the solution of the LP relaxation happens to be integer and, therefore, optimal for the subproblem itself. In this case, no further exploration along that subproblem is necessary and we say that the node has been *pruned by optimality*.

Figure 9.4 illustrates the process of pruning by optimality. Each box denotes a subproblem, with the interval denoting known lower (primal) and upper (dual) bounds for the problem and x denoting the solution for the LP relaxation of the subproblem. In Figure 9.4, we see a pruning that is caused because a solution to the original (integer) subproblem has been identified by solving its (LP) relaxation, akin to the leaves in the enumerative tree represented in Figure 9.3. This can be concluded because the solution of the LP relaxation of subproblem S_1 is integer.

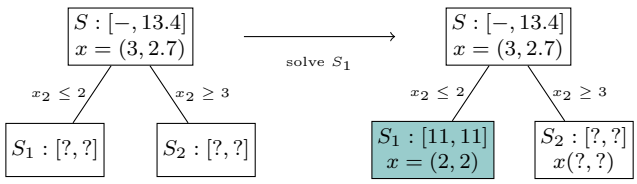


Figure 9.4: An example of pruning by optimality. Since the solution of LP relaxation of subproblem S_1 is integer, $x = (2, 2)$ must be optimal for S_1

Another type of pruning takes place when known global (primal) bounds can be used to prevent further exploration of a branch in the enumeration tree. Continuing the example in Figure 9.4, notice that the global lower (primal) bound $\underline{z} = 11$ becomes available and can be transmitted to all subproblems. Now suppose we solve the LP relaxation of S_2 and obtain the optimal value of $\overline{z}_2 = 9.7$. Notice that we are precisely in the situation described in Section 9.3.1. That is, the nodes descending from S_2 (i.e., their descendants) can only yield solutions that have objective function value worse than the dual bound of S_2, which, in turn, is worse than a known global primal (lower) bound. Thus, any further exploration among the descendants of S_2 would be fruitless in terms of yielding better solutions and can be pruned. This is known as *pruning by bound*, and is illustrated in Figure 9.5.

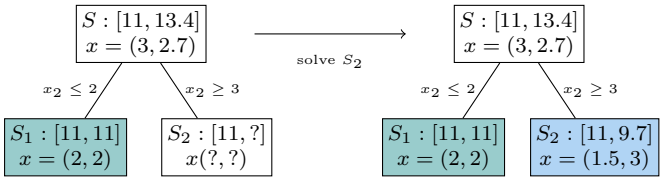


Figure 9.5: An example of pruning by bound. Notice that the newly found global bound holds for all subproblems. After solving the LP relaxation of S_2, we notice that $\overline{z}_2 \leq \underline{z}$, which renders the pruning.

The third type of pruning is called *pruning by infeasibility*, which takes place whenever the branching constraint added to the subproblem renders its relaxation infeasible, implying that the subproblem itself is infeasible (cf. Proposition 9.3)

Algorithm 7 presents a pseudocode for an LP-based branch-and-bound method. Notice that the algorithm keeps a list $\mathcal{L}$ of subproblems to be solved and requires that a certain rule to select which subproblem is solved next to be employed. This subproblem selection (often referred to as *search strategy*) can have considerable impacts on the performance of the method. Similarly, in case multiple components are found to be fractional, one must be chosen. Defining such *branching priorities* also has consequences to performance. We will discuss these in more depth later on.

Also, recall that we have seen how to efficiently resolve a linear programming problem from an optimal basis once we include an additional constraint (in Chapter 6). It so turns out that an efficient dual simplex method is the kingpin of an efficient branch-and-bound method for (mixed)-integer programming problems.

Finally, although we developed the method in the context of integer programming problems, the method can be readily applied to mixed-integer programming problems, with the only difference being that the branch-and-bound steps are only applied to the integer variables while the continuous variables are naturally taken care of in the solution of the LP relaxations.

Let us finalise presenting a numerical example of the employment of branch-and-bound method to solve an integer programming problem. Consider the problem:

$$\begin{aligned} \max_{x} \; z =& 4x_1 - x_2 \\ \text{s.t.: } & 7x_1 - 2x_2 \leq 14 \\ & x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x \in \mathbb{Z}^2_+ \end{aligned}$$

We start by solving its LP relaxation, as represented in Figure 9.6. We obtain the solution $x_{LP} =$

Algorithm 7 LP-relaxation-based branch-and-bound

1: **initialise.** $\mathcal{L} \leftarrow \{S\}$, $\underline{z} \leftarrow -\infty$, $\overline{x} \leftarrow -\infty$
2: **while** $\mathcal{L} \neq \emptyset$ **do**
3: select problem S_i from $\mathcal{L}$. $\mathcal{L} \leftarrow \mathcal{L} \setminus \{S_i\}$.
4: solve LP relaxation of S_i over P_i, obtaining z^i_{LP} and x^i_{LP}. $\overline{z}^i \leftarrow z^i_{LP}$.
5: **if** $S_i = \emptyset$ **then** return to step 2.
6: **else if** $\overline{z}^i \leq \underline{z}$ **then** return to step 2.
7: **else if** $x^i_{LP} \in \mathbb{Z}^n$ **then** $\underline{z} \leftarrow \max\left\{\underline{z}, \overline{z}^i\right\}$, $\overline{x} \leftarrow x^i_{LP}$; and return to step 2
8: **end if**
9: select a fractional component x_j and create subproblems S_{i1} and S_{i2} with formulations P_{i1} and P_{i2}, respectively, such that

$$P_{i1} = P_i \cup \{x_j \leq \lfloor \overline{x_j} \rfloor\} \text{ and } P_{i2} = P_i \cup \{x_j \geq \lceil \overline{x_j} \rceil\}.$$

10: $\mathcal{L} \leftarrow \mathcal{L} \cup \{S_{i1}, S_{i2}\}$.
11: **end while**
12: **return** $(\overline{x}, \underline{z})$.

$(20/7, 3)$ with objective value of $z = 59/7$. As the first component of x is fractional, we can generate subproblems by branching the node into subproblems S_1 and S_2, where

$$S_1 = S \cap \{x : x_1 \leq 2\}$$
$$S_2 = S \cap \{x : x_1 \geq 3\}.$$

The current enumerative (or branch-and-bound) tree representation is depicted in Figure 9.7.

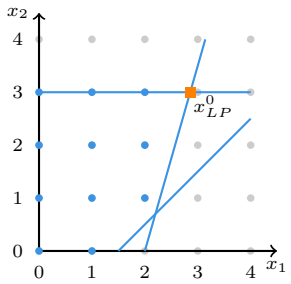


Figure 9.6: LP relaxation of the problem S

Suppose we arbitrarily choose to solve the relaxation of S_1 next. Notice that this subproblem consists of the problem S, with the added constraint $x_1 \leq 2$. The feasible region and solution of the LP relaxation of S_2 is depicted in Figure 9.8. Since we again obtain a new fractional solution $x^1_{LP} = (2, 1/2)$, we must branch on the second component, forming the subproblems

$$S_{11} = S_1 \cap \{x : x_2 = 0\}$$
$$S_{12} = S_1 \cap \{x : x_2 \geq 1\}.$$

Notice that, at this point, our list of active subproblems is formed by $\mathcal{L} = \{S_1, S_{11}, S_{12}\}$. Our current branch-and-bound tree is represented in Figure 9.9.

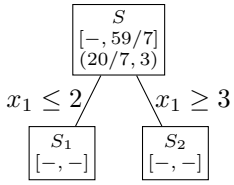


Figure 9.7: The branch-and-bound tree after branching S onto S_1 and S_2

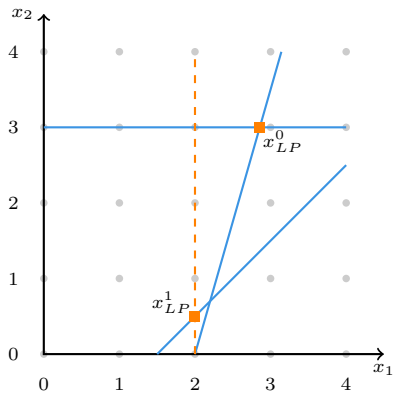


Figure 9.8: LP relaxation of subproblem S_1

Suppose we arbitrarily choose to first solve S_2. Once can see that this would render an infeasible subproblem, since the constraint $x_2 \geq 3$ does not intersect with the original feasible region and, thus, S_2 can be pruned by infeasibility.

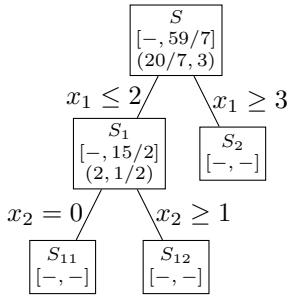


Figure 9.9: The branch-and-bound tree after branching S_1 onto S_{11} and S_{12}

Next, we choose to solve the LP relaxation of S_{12}, which yields an integer solution $x^{12}_{LP} = (2, 1)$. Therefore, an optimal solution for S_{12} was found, meaning that a global primal (lower) bound has been found and can be transmitted to the whole branch-and-bound tree. Solving S_{11} next, we obtain the solution $x^{11}_{LP} = (3/2, 0)$ with optimal value $z = 6$. Since a better global primal (lower) bound is known, we can prune S_{11} by bound. As there are no further nodes to be explored, the solution for the original problem is the best (and, in this case, the single) integer solution found in the process (cf. Proposition 9.6), $x^* = (2, 1)$, $z^* = 7$. Figure 9.10 illustrates the feasible region of the subproblems and their respective optimal solutions, while Figure 9.11 presents the final branch-and-bound tree with all branches pruned.

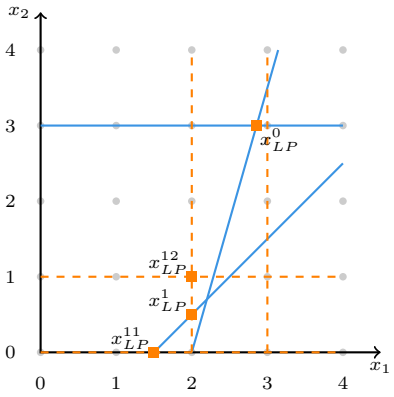


Figure 9.10: LP relaxations of all subproblems. Notice that S_{11} and S_{12} includes the constraints $x_1 \leq 2$ from the parent node S_1

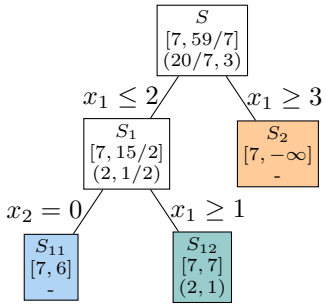


Figure 9.11: The final branch-and-bound tree

Notice that in this example, the order in which we solved the subproblems was crucial for pruning by bound the subproblem S_{11}, only possible because we happened to solve the LP relaxation of S_{12} first and that happened to yield a feasible solution and associated primal bound. This illustrates an important aspect associated with the branch and bound method: having good feasible solutions available early on in the process increases the likelihood of performing more pruning by bound, which is highly desirable in terms of computational savings (and thus, performance). We will discuss in more detail the impacts of different search strategies later on when we consider this and other aspects involved in the implementation of mixed-integer programming solvers.

9.4 Exercises

Problem 9.1: Uncapacitated Facility Location (UFL)

(a) Let $N = \{1, \ldots, n\}$ be a set of potential facilities and $M = \{1, \ldots, m\}$ a set of clients. Let $y_j = 1$ if facility j is opened, and $y_j = 0$ otherwise. Moreover, let x_{ij} be the fraction of client i's demand satisfied from facility j. The UFL can be formulated as the mixed-integer problem:

$$\begin{aligned}
(\text{UFL-W}) : \min_{x,y} \quad & \sum_{j \in N} f_j y_j + \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} && && (9.1)\\
\text{s.t.:} \quad & \sum_{j \in N} x_{ij} = 1, && \forall i \in M, && (9.2)\\
& \sum_{i \in M} x_{ij} \leq m y_j, && \forall j \in N, && (9.3)\\
& x_{ij} \geq 0, && \forall i \in M, \forall j \in N, && (9.4)\\
& y_j \in \{0, 1\}, && \forall j \in N, && (9.5)
\end{aligned}$$

where f_j is the cost of opening facility j, and c_{ij} is the cost of satisfying client i's demand from facility j. Consider an instance of the UFL with opening costs $f = (4, 3, 4, 4, 7)$ and client costs

$$(c_{ij}) = \begin{pmatrix} 12 & 13 & 6 & 0 & 1 \\ 8 & 4 & 9 & 1 & 2 \\ 2 & 6 & 6 & 0 & 1 \\ 3 & 5 & 2 & 1 & 8 \\ 8 & 0 & 5 & 10 & 8 \\ 2 & 0 & 3 & 4 & 1 \end{pmatrix}$$

Implement (the model) and solve the problem with Julia using JuMP.

(b) An alternative formulation of the UFL is of the form

$$\begin{aligned}
(\text{UFL-S}) : \min_{x,y} \quad & \sum_{j \in N} f_j y_j + \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} && && (9.6)\\
\text{s.t.:} \quad & \sum_{j \in N} x_{ij} = 1, && \forall i \in M, && (9.7)\\
& x_{ij} \leq y_j, && \forall i \in M, \forall j \in N, && (9.8)\\
& x_{ij} \geq 0, && \forall i \in M, \forall j \in N, && (9.9)\\
& y_j \in \{0, 1\}, && \forall j \in N. && (9.10)
\end{aligned}$$

Linear programming (LP) relaxations of these problems can be obtained by relaxing the binary constraints $y_j \in \{0, 1\}$ to $0 \leq y_j \leq 1$ for all $j \in N$. For the same instance as in part (a), solve the LP relaxations of UFL-W and UFL-S and compare the optimal costs of the LP relaxations against the optimal integer cost obtained in part (a).

Problem 9.2: LP-based branch-and-bound method

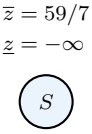
Consider the following IP problem and its standard form

$$\begin{array}{lll} \text{(IP)} & z = \max & 4x_1 - x_2 \\ & \text{s.t.:} & 7x_1 - 2x_2 \leq 14 \\ & & x_2 \leq 3 \\ & & 2x_1 - 2x_2 \leq 3 \\ & & x_1, x_2 \in \mathbb{Z}_+ \end{array} \qquad \begin{array}{lll} \text{(IP)} & \text{z} = & \max \quad 4x_1 - x_2 \\ & \text{s.t.:} & 7x_1 + 2x_2 + x_3 = 14 \\ & & x_2 + x_4 = 3 \\ & & 2x_1 - 2x_2 + x_5 = 3 \\ & & x_1, \ldots, x_5 \in \mathbb{Z}_+ \end{array}$$

Solve the IP problem by LP-based branch and bound, i.e., use LP relaxations to compute dual (upper) bounds. Use Dual Simplex to efficiently solve the subproblem of each node starting from the optimal basis of the previous node. Recall that the LP relaxation of IP is obtained by relaxing the variables $x_1, \ldots, x_5 \in \mathbb{Z}_+$ to $x_1, \ldots, x_5 \geq 0$.

Hint: The initial dual bound $\overline{z}$ is obtained by solving the LP relaxation of IP at the root node S. Let $[\underline{z}, \overline{z}]$ be the lower and upper bounds of each node. The optimal tableau and the initial Branch & Bound tree with only the root node S is shown below.

	-z	x_1	x_2	x_3	x_4	x_5
	-59/7	0	0	-4/7	-1/7	0
$x_1 =$	20/7	1	0	1/7	2/7	0
$x_2 =$	3	0	1	0	1	0
$x_5 =$	23/7	0	0	-2/7	10/7	1



You can proceed by branching on the fractional variable x_1 and imposing either $x_1 \leq 2$ or $x \geq 3$. This creates two new subproblems $S_1 = S \cap \{x_1 \leq 2\}$ and $S_2 = S \cap \{x_1 \geq 3\}$ in the branch-and-bound tree that can be solved efficiently using the dual simplex method, starting from the optimal tableau of S shown above, by first adding the new constraint $x_1 \leq 2$ for S_1 or $x_1 \geq 3$ for S_2 to the optimal tableau. The dual simplex method can be applied immediately if the new constraint is always written in terms of non-basic variables before adding it to the tableau as a new row, possibly multiplying the constraint by -1 if needed.

Problem 9.3: Employing the branch-and-bound method graphically

Consider the following integer programming problem IP:

$$\begin{array}{lll} (IP): \max. & z = & x_1 + 2x_2 \\ & \text{s.t.:} & -3x_1 + 4x_2 \leq 4 \\ & & 3x_1 + 2x_2 \leq 11 \\ & & 2x_1 - x_2 \leq 5 \\ & & x_1, x_2 \in \mathbb{Z}_+ \end{array}$$

Plot (or draw) the feasible region of the linear programming (LP) relaxation of the problem IP, then solve the problems using the figure. Recall that the LP relaxation of IP is obtained by

replacing the integrality constraints $x_1, x_2 \in \mathbb{Z}_+$ by linear nonnegativity $x_1, x_2 \geq 0$ and upper bounds corresponding to the upper bounds of the integer variables ($x_1, x_2 \leq 1$ for binary variables).

(a) What is the optimal cost z_{LP} of the LP relaxation of the problem IP? What is the optimal cost z of the problem IP?

(b) Draw the border of the convex hull of the feasible solutions of the problem IP. Recall that the convex hull represents the *ideal* formulation for the problem IP.

(c) Solve the problem IP by LP-relaxation based branch-and-bound. You can solve the LP relaxations at each node of the branch-and-bound tree graphically. Start the branch-and-bound procedure without any primal bound.

Chapter 10

Cutting-planes method

10.1 Valid inequalities

In this chapter, we will discuss the idea of generating and adding constraints to improve a formulation of a (possibly mixed-)integer programming problem. This idea can either be implemented in a priori setting, for example before employing the branch-and-bound method, or as the solution method itself. These constraints are often called *valid inequalities* or *cuts*, though the latter is typically used in the context of cutting-planes methods.

Let us start by defining the integer programming problem

$$(IP) : \max_{x} \ \left\{c^\top x : x \in X\right\}$$

where $X = P \cap \mathbb{Z}^n$ and $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

The idea of using constraints to solve IP is founded in the following observations. We know that $\mathbf{conv}(X)$ is a (convex) polyhedral set (cf. Definition 2.7) and, being so, there exists a finite set of inequalities $\tilde{A}x \leq \tilde{b}$ such that

$$\mathbf{conv}(X) = \left\{x \in \mathbb{R}^n : \tilde{A}x \leq \tilde{b}, x \geq 0\right\}$$

Furthermore, if we had available $\tilde{A}x \leq \tilde{b}$, then we could solve IP by solving its linear programming relaxation.

Cutting-plane methods are based on the idea of iteratively approximating the set of inequalities $\tilde{A}x \leq \tilde{b}$ by adding constraints to the formulation P of IP. These constraints are called *valid inequalities*, a term we define more precisely in Definition 10.1.

Definition 10.1 (Valid inequality)**.** *An inequality $\pi^\top x \leq \pi_0$ is valid for $X \subset \mathbb{R}^n$ if $\pi^\top x \leq \pi_0$ for all $x \in X$.*

Notice that the condition for an inequality to be valid is that it does not remove any of the point in the original integer set X. In light of the idea of gradually approximating $\mathbf{conv}(X)$, one can infer that good valid inequalities are those that can "cut off" some of the area defined by the polyhedral set P, but without removing any of the points in X. This is precisely where the name *cut* comes from. Figure 10.1 illustrates the process of adding a valid inequality to a formulation P. Notice how the inequality expose one of the facets of the convex hull of X. Cuts like such are called "facet-defining" and are the strongest types of cuts one can generate. We will postpone the discussion of stronger cuts to later in this chapter.

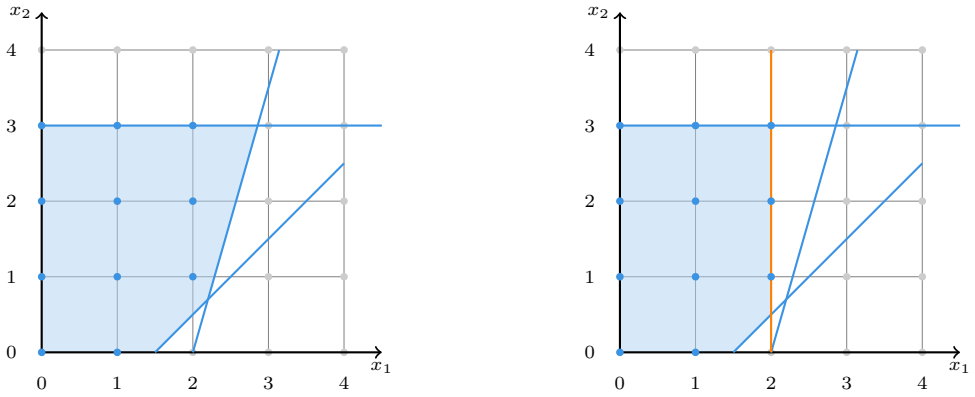


Figure 10.1: Illustration of a valid inequality being added to a formulation P. Notice how the inequality cuts of a portion of the polyhedral set P while not removing any of the feasible points X (represented by the dots)

10.2 The Chvátal-Gomory procedure

To develop a systematic procedure for generating valid inequalities in the context of a solution method for integer programming problems, we will rely on a two step procedure. First, a cut that is valid for the polyhedral set P is (somewhat automatically) generated, and then it is made valid for the integer set X by a simple rounding procedure. Before we proceed, let us define the notion of valid inequalities in the context of linear programming problems.

Proposition 10.2 (Valid inequalities for polyhedral sets)**.** *An inequality $\pi^\top x \leq \pi_0$ is valid for $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$, if and only if $P \neq \emptyset$ and there exists $u \geq 0$ such that $u^\top A \geq \pi$ and $u^\top b \leq \pi_0$.*

Proof. We can use linear programming duality to prove this statement. First, notice that, if the proposition holds, for $u \geq 0$ and $x \in P$, we have

$$\begin{aligned} Ax &\leq b \\ u^\top Ax &\leq u^\top b \\ \pi^\top x \leq u^\top Ax &\leq u^\top b \leq \pi_0, \end{aligned}$$

and, thus, it implies the validity of the cut, i.e., that $\pi^\top x \leq \pi_0, \forall x \in P$. Now, consider the primal problem

$$\begin{aligned} \max.\ & \pi^\top x \\ \text{s.t.:}\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

and its dual

$$\begin{aligned} \min.\ & u^\top b \\ \text{s.t.:}\ & u^\top A \geq \pi \\ & u \geq 0. \end{aligned}$$

Thus, $u^\top A \geq \pi$ can be seen as a consequence of dual feasibility, which is guaranteed to hold for some u since $\pi^\top x$ is bounded. Furthermore, strong duality gives $u^\top b = \pi^\top x \leq \pi_0$, completing the proof. □

One thing to notice is that valid cuts in the context of polyhedral sets are somewhat redundant, since, by definition, they do not alter the polyhedral set in any means. However, the concept can be combined with a simple yet powerful way of generating valid inequalities for integer set by using rounding. This is stated in Proposition 10.3.

Proposition 10.3 (Valid inequalities for integer sets)**.** *Let $X = \left\{y \in \mathbb{Z}^1 : y \leq b\right\}$. The inequality $y \leq \lfloor b \rfloor$ is valid for X.*

The proof of Proposition 10.3 is somewhat straightforward and left as a thought exercise.

We can combine Propositions 10.2 and 10.3 into a single procedure to automatically generate valid inequalities. Let us start with a numerical example. Consider the set $X = P \cap \mathbb{Z}^n$ where P is defined by

$$P = \left\{x \in \mathbb{R}^2_+ : 7x_1 - 2x_2 \leq 14, x_2 \leq 3, 2x_1 - 2x_2 \leq 3\right\}.$$

First, let $u = \left[\frac{2}{7}, \frac{37}{63}, 0\right]$, which, for now, we can assume that they were arbitrarily chosen. We can then combine the constraints in P (the $Ax \leq b$ in Proposition 10.2) forming the constraint (equivalent to $u^\top Ax \leq u^\top b$)

$$2x_1 + \frac{1}{63}x_2 \leq \frac{121}{21}.$$

Now, notice that the constraint would remain valid for P if we simply *round down* the coefficients on the lefthand side (as $x \geq 0$ and all coefficients are positive). This would lead to the new constraint (notice that this yields a vector π in Proposition 10.2)

$$2x_1 + 0x_2 \leq \frac{121}{21}.$$

Finally, we can invoke Proposition 10.3 to generate a cut valid for X. This can be achieved by simply rounding down the righthand side (yielding π_0), obtaining

$$2x_1 + 0x_2 \leq 5,$$

which is valid for X, but not for P. Notice that, apart from the vector of weights u used to combine the constraints, everything else in the procedure of generating the valid inequality for X is automated. This procedure is known as the *Chvátal-gomory procedure* and can be formalised as follows.

Definition 10.4 (Chvátal-Gomory procedure)**.** *Consider the integer set $X = P \cap \mathbb{Z}^n$ where $P = \left\{x \in \mathbb{R}^n_+ : Ax \leq b\right\}$, A is an $m \times n$ matrix with columns $\{A_1, \ldots, A_n\}$ and $u \in \mathbb{R}^m_+$.*

The Chvátal-Gomory procedure consists of the following set of steps to generate valid inequalities for X:

1. *$\sum_{j=1}^n u^\top A_j x_j \leq u^\top b$ is valid for P as $u \geq 0$;*
2. *$\sum_{j=1}^n \lfloor u^\top A_j \rfloor x_j \leq u^\top b$ is valid for P as $x \geq 0$;*
3. *$\sum_{j=1}^n \lfloor u^\top A_j \rfloor x_j \leq \lfloor u^\top b \rfloor$ is valid for X as LHS is integer.*

Perhaps the most striking result in the theory of integer programming is that *every* valid inequality for an integer set X can be obtained by employing the Chvátal-gomory procedure a number of times. This is formalised in Theorem 10.5.

Theorem 10.5. *Every valid inequality for X can be obtained by applying the Chvátal-Gomory procedure a finite number of times.*

10.3 The cutting-plane method

Let us now consider how one could use valid inequalities to devise a solution method. The working paradigm behind a cutting-plane method is the *separation principle*.

The separation principle states that, given an integer set $X = P \cap \mathbb{Z}^n$ if a solution $x \notin X$, then there exists a hyperplane $\pi^\top x \leq \pi_0$ separating x and X[1]. As one might infer, the challenge is how one can generate such pairs (π, π_0). This is precisely what is called the *separation problem*, in the context of integer programming.

In general, these valid inequalities are generated from a family of inequalities $\mathcal{F}$, which are related to each other by properties related to, e.g., problem structure or nature of the inequality itself. One way of thinking about it is to see the family of inequalities $\mathcal{F}$ as a means to dictate, to some extent, how the selection of weights u in the Chvátal-Gomory procedure is defined.

In any case, in possession of a family of inequalities and a method to solve the separation problem, we can pose a cutting-plane methods in general terms. This is stated in Algorithm 8.

Algorithm 8 Cutting-plane algorithm

initialise. let $\mathcal{F} \subseteq \left\{(\pi, \pi_0) : \pi^\top x \leq \pi_0 \text{ is valid for } X\right\}$. $k \leftarrow 0$.
while $x^k_{LP} \notin \mathbb{Z}^n$ **do**
    solve the LP relaxation over P, obtaining the optimal objective value z^k_{LP} and optimal solution x^k_{LP}.
    if $x^k_{LP} \notin \mathbb{Z}^n$ **then** find $(\pi^k, \pi^k_0) \in \mathcal{F}$ such that ${\pi^k}^\top x > \pi^k_0$.
    else
        return (x^k_{LP}, z^k_{LP}).
    end if
    $P \leftarrow P \cup \left\{{\pi^k}^\top x \leq \pi^k_0\right\}$. $k \leftarrow k + 1$.
end while
return (x^k_{LP}, z^k_{LP}).

Notice that we have already discussed a cutting-plane method before in Chapter 7, when we presented the Benders decomposition. In that case, the optimality and feasibility cuts form the family of valid inequalities $\mathcal{F}$, while the separation problem was the subproblem responsible to find the cuts that were violated by the current main problem solution.

Like it was the case in Benders decomposition, the motivation of cutting-plane algorithms lies in the belief that one a few of all $|\mathcal{F}|$ inequalities (assuming $\mathcal{F}$ is finite, which might not be necessarily the case) are necessary, circumventing the computationally prohibitive need of generating all possible inequalities from $\mathcal{F}$.

There are some other complicating aspects that must be observed when dealing with cutting-plane algorithms. First, it might be so that a given family of valid inequalities $\mathcal{F}$ is not sufficient to

[1] the separation principle is a consequence of the separation theorem (Theorem 13.10) in Part II.

expose the optimal solution $x \in X$, which might be the case, for example, if $\mathcal{F}$ cannot fully describe $\mathbf{conv}(X)$ or if the separation problem is unsolvable. In that case, the algorithm will terminate with a solution for the LP relaxation tat is not integer, i.e., $x^k_{LP} \notin \mathbb{Z}^n$.

However, failing to converge to an integer solution is not a complete failure since, in the process, we have improved the formulation P (cf. Definition 8.3). In fact, this idea plays a major role in professional-grade implementations of mixed-integer programming solvers, as we will see later.

10.4 Gomory's fractional cutting-plane method

One important cutting-plane method that is guaranteed to converge (in theory) to an integer solution is the Gomory's fractional cutting plane method. Basically, the method consists of exploiting the Chvátal-Gomory procedure (cf .Definition 10.4) to be the family of cuts generated, while solving separation problem becomes simply the process of rounding to be applied to solutions of LP relaxations.

Specifically, consider the integer programming problem

$$(P) : \max_x \ \left\{c^\top x : x \in X\right\}$$

where $X = \left\{x \in \mathbb{Z}^n_+ : Ax = b\right\}$. Recall that the optimal solution of the LP relaxation is characterised by a basis B formed by columns of the matrix A, i.e.,

$$A = [B \mid N] \text{ and } x = (x_B, x_N),$$

where x_B are the basic components of the solution and $x_N = 0$ the nonbasic components. The matrix N if formed by columns of A associated with the nonbasic variables x_N.

As we have discussed in Chapter 3, the system of equation $Ax = b$ can be written as

$$Bx_B + Nx_N = b \text{ or } x_B + B^{-1}Nx_N = B^{-1}b,$$

which is equivalent to $B^{-1}Ax = B^{-1}b$. Now, let $\overline{a}_{ij}$ be the element in row i and column j in $B^{-1}A$, and let $\overline{a}_{i0} = (B^{-1}b)_i$ be the i-th component of $B^{-1}b$. With that, we can represent the set of feasible solutions X as

$$\begin{aligned} x_{B(i)} + \sum_{j \in I_N} \overline{a}_{ij} x_j = \overline{a}_{i0}, \forall i \in I \\ x_j \in \mathbb{Z}_+, \forall j \in J, \end{aligned}$$

where $I = \{1, \ldots, m\}$, $J = \{1, \ldots, n\}$, $I_B \subset J$ are the indices of basic variables and $I_N = J \setminus I_B$ the indices of nonbasic variables. Notice that, at this point, we are simply recasting P by performing permutations of columns, since basic feasible solutions for the LP relaxation do not necessarily translate into a feasible solution for X.

However, assume we solve the LP relaxation of the integer programming problem P and obtain an optimal solution $x = (x_B, x_N)$ with associated optimal basis B. If x is fractional, then it means that $\overline{a}_{i0}$ is fractional for some i.

From any of the rows i with fractional $\overline{a}_{i0}$, we can derive a valid inequality using the Chavátal-Gomory procedure. These inequalities, commonly referred to as *CG cuts*, take the form

$$x_{B(i)} + \sum_{j \in I_N} \lfloor \overline{a}_{ij} \rfloor x_j \leq \lfloor \overline{a}_{i0} \rfloor. \tag{10.1}$$

As this is thought to be used in conjunction with the simplex method, we must be able to state (10.1) in terms of the nonbasic variables x_j, $\forall j \in I_N$. To do so, we can replace $x_{B(i)} = \overline{a}_{i0} - \sum_{j \in I_N} \overline{a}_{ij} x_j$, obtaining

$$\sum_{j \in I_N} (\overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor) x_j \geq (\overline{a}_{i0} - \lfloor \overline{a}_{i0} \rfloor),$$

which, by defining $f_{ij} = \overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor$, can be written in the more conventional form

$$\sum_{j \in I_N} f_{ij} x_j \geq f_{i0}. \tag{10.2}$$

In the form of (10.2), this inequality is referred to as the Gomory (fractional) cut.

Notice that the inequality (10.2) is not satisfied by the optimal solution of the LP relaxation, since $x_j = 0, \forall j \in I_N$ and $f_{i0} > 0$. Therefore, this indicates that a cutting-plane method using this idea benefits from the employment of dual simplex, in line with the discussion in Section 6.1.2.

Let us present an numerical example illustrating the employment of Gomory's fractional cutting plane algorithm for solving the following integer programming problem

$$\begin{aligned} z = \max_x. \; & 4x_1 - x_2 \\ \text{s.t.:} \; & 7x_1 - 2x_2 \leq 14 \\ & x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x_1, x_2 \in \mathbb{Z}_+ \end{aligned}$$

Figure 10.2 illustrates the feasible region of the problem and indicates the solution of the its LP relaxation. Considering the tableau representation of the optimal basis for the LP relaxation, we have

x_1	x_2	x_3	x_4	x_5	RHS
0	0	-4/7	-1/7	0	59/7
1	0	1/7	2/7	0	20/7
0	1	0	1	0	3
0	0	-2/7	10/7	1	23/7

Notice that the tableau indicate that the component x_1 in the optimal solution is fractional. Thus, we can choose that rom to generate a Gomory cut. This will lead to the new constraint (with added respective slack variable $s \geq 0$.

$$\frac{1}{7} x_3 + \frac{2}{7} x_4 - s = \frac{6}{7}.$$

We can proceed to add this new constraint onto the problem, effectively adding an additional row to the tableau. After multiplying it by -1 (so we have s as a basic variable complementing the augmented basis), we obtain the new tableau

x_1	x_2	x_3	x_4	x_5	s	RHS
0	0	-4/7	-1/7	0	0	59/7
1	0	1/7	2/7	0	0	20/7
0	1	0	1	0	0	3
0	0	-2/7	10/7	1	0	23/7
0	0	-1/7	-2/7	0	1	-6/7

Notice that the solution remains dual feasible, which indicates the suitability of the dual simplex method. Applying the dual simplex method leads to the optimal tableau

x_1	x_2	x_3	x_4	x_5	s	RHS
0	0	0	0	-1/2	-3	15/2
1	0	0	0	0	1	2
0	1	0	0	-1/2	1	1/2
0	0	1	0	-1	-5	1
0	0	0	1	1/2	-1	5/2

Notice that we still have a fractional component, this time associated with x_2. We proceed in an analogous fashion, first generating the Gomory cut and adding the slack variable $t \geq 0$, thus obtaining

$$\frac{1}{2}x_5 - t = \frac{1}{2}.$$

Then, adding it to the previous tableau and employing the dual simplex again, leads to the optimal tableau

x_1	x_2	x_3	x_4	x_5	s	t	RHS
0	0	0	0	0	-4	-1	7
1	0	0	0	0	1	0	2
0	1	0	0	0	0	-1	1
0	0	1	0	0	-7	-2	2
0	0	0	1	0	0	1	2
0	0	0	0	1	-2	-2	1

Notice that now all variables are integer, and thus, an optimal for solution for the original integer programming problem was found.

Some points are worth noticing. First, notice that, at the optimum, all variables, including the slacks, are integer. This is a consequence of having the Gomory cuts active at the optimal solution since (10.1), and consequently (10.2), have both the left and righthand sides integer. Also, notice that at each iteration the problem increases in size, due to the new constraint being added, which implies that the basis also increases in size. Thought this is an issue also in branch-and-bound method, it can be a more prominent computational issue in the context of cutting-plane methods.

We can also interpret the progress of the algorithm in graphical terms. First of all, notice that we can express the cuts in terms of the original variables (x_1, x_2) by noticing that the original formulation gives $x_3 = 14 - 7x_1 + x_2$ and $x_4 = 3 - x_2$. Substituting x_3 and x_4 in the cut $\frac{1}{7}x_3 + \frac{2}{7}x_4 - s = \frac{6}{7}$ gives $x_1 \leq 2$. More generally, we have that cuts can be expressed using the original problem variables, as stated in Proposition 10.6.

Proposition 10.6. *Let β be the row l of B^{-1} selected to generate the cut, and let $q_i = \beta_i - \lfloor \beta_i \rfloor$, $i \in \{1, \ldots, m\}$. Then the cut $\sum_{j \in I_N} f_{lj} x_j \geq f_{l0}$, written in terms of the original variables, is the Chvátal-Gomory inequality*

$$\sum_{j=1}^{n} \lfloor qA_j \rfloor x_j \leq \lfloor qb \rfloor.$$

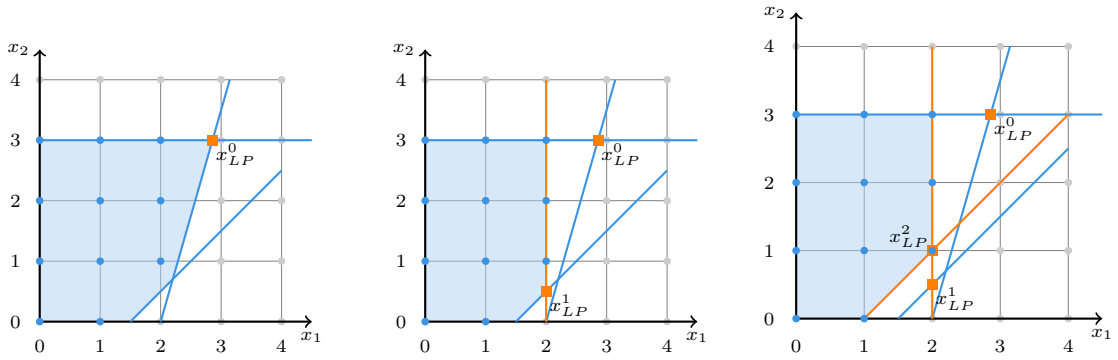


Figure 10.2: Feasible region of the LP relaxation (polyhedral set) and of the integer programming problem (blue dots) at each of three iterations taken to solve the integer programming problem. The inequalities in orange represent the Gomory cut added at each iteration

10.5 Obtaining stronger inequalities

We finalise this chapter discussing the notion of strong inequalities and providing an example on how they can be made stronger in some cases by means of an external process.

10.5.1 Strong inequalities

The notion of strong inequalities arises from the notion of stronger formulations, cf. Definition 8.3. That is, we say that an inequality is strong in terms of its relative quality in describing the convex hull of integer set.

However, in this context we are interested in comparing two alternative inequalities to decide which is stronger. For that, we can rely on the notions of *dominance* and related *redundancy* of inequalities.

Consider two valid inequalities $\pi x \leq \pi_0$ and $\mu x \leq \mu_0$ that are valid for a polyhedral set $P = \left\{x \in \mathbb{R}^n_+ : Ax \leq b\right\}$. Definition 10.7 formalises the notion of dominance.

Definition 10.7 (Dominance)**.** *The inequality $\pi x \leq \pi_0$* dominates *$\mu x \leq \mu_0$ if these exists $u > 0$ such that $\pi \geq u\mu$, $\pi_0 \leq u\mu_0$, and $(\pi, \pi_0) \neq (u\mu, u\mu_0)$.*

Let us illustrate the concept of dominance with a numerical example. Consider the inequalities $2x_1 + 4x_2 \leq 9$ and $x_1 + 3x_2 \leq 4$, which are valid for $P = \mathbf{conv}(X)$, where

$$X = \{(0,0), (0,1), (0,2), (0,3), (0,4), (1,0), (1,1)\}.$$

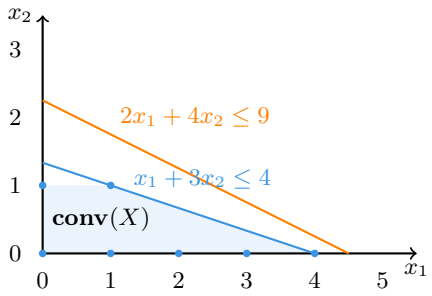


Figure 10.3: Illustration of dominance between constraints. Notice that $x_1 + 3x_2 \leq 4$ dominates $2x_1 + 4x_2 \leq 9$ and is thus stronger

Notice that, if we consider $u = 1/2$, we have that, for any $x = (x_1, x_2)$, that $1x_1 + 3x_2 \geq u(2x_1 + 4x_2) = x_1 + 2x_2$ and that $4 \leq 9u = 9/2$. Thus, we say that $x_1 + 3x_2 \leq 4$ dominates $2x_1 + 4x_2 \leq 9$. Figure 10.3 illustrates the two inequalities. Notice that $x_1 + 3x_2 \leq 4$ is a stronger inequality, since it is more efficient in representing the convex hull of X than $2x_1 + 4x_2 \leq 9$.

Another related concept is the notion of redundancy. Clearly, the presence of two constraints in which one dominates the other, the dominated constraint is also redundant and can be safely removed from the formulation of a polyhedral set P. However, in some cases one might not be able to identify redundant constraints simply because no constraint is clearly dominated by another.

Even then, there might be a way to identify weak (or redundant) constraints by combining two or more constraints that then form a dominating constraint. This is formalised in Definition 10.8.

Definition 10.8 (Redundancy)**.** *The inequality $\pi x \leq \pi_0$ is* redundant *for P if there exists $k \geq 1$ valid inequalities $\pi^i x \leq \pi_0^i$ and $k \geq 1$ vectors $u_i > 0$, for $i \in \{1, \ldots, k\}$, such that $\left(\sum_{i=1}^k u_i \pi^i\right) x \leq \sum_{i=1}^k u_i \pi_0^i$ dominates $\pi x \leq \pi_0$.*

Once again, let us illustrate the concept with a numerical example. Consider we generate the inequality $5x_1 - 2x_2 \leq 6$, which is valid for the polyhedral set

$$P = \left\{x \in \mathbb{R}^2_+ : 6x_1 - x_2 \leq 9, 9x_1 - 5x_2 \leq 6\right\}.$$

The inequality $5x_1 - 2x_2 \leq 6$ is not dominated by any of the inequalities forming P. However, if we set $u_i = (\frac{1}{3}, \frac{1}{3})$, we obtain $5x_1 - 2x_2 \leq 5$, which in turn dominates $5x_1 - 2x_2 \leq 6$. Thus, we can conclude that the generated inequality is redundant and does not improve the formulation of P. This is illustrated in Figure 10.4.

As one might realise, checking whether a newly generated inequality improves the current formulation is a demanding task, as it requires finding the correct set of coefficients u for all constraints currently forming the polyhedral set P. Nonetheless, the notions of redundancy and dominance can be used to guide procedures that generate or improve existing inequalities. Let us discuss one of such procedures, in the context of 0-1 knapsack inequalities.

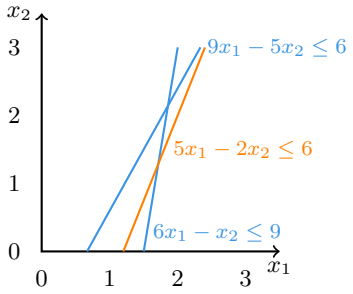


Figure 10.4: Illustration of a redundant inequality. Notice how the inequality $5x_1 - 2x_2 \leq 6$ (in orange) does not dominate any of the other inequalities

10.5.2 Strengthening 0-1 knapsack inequalities

Let us consider the family of constraints known as knapsack constraints and see how they can be strengthened. For that, let us first define the knapsack set

$$X = \left\{ x \in \{0,1\}^n : \sum_{j=1}^{n} a_j x_j \leq b \right\}.$$

We assume that $a_j \geq 0$, $j \in N = \{1, \ldots, n\}$, and $b > 0$. Let us start by defining the notion of a *minimal cover*.

Definition 10.9 (minimal cover)**.** *A set $C \subseteq N$ is a cover if $\sum_{j \in C} a_j > b$. A cover C is minimal if $C \setminus \{j\}$ for all $j \in C$ is not a cover.*

Notice that a cover C refers to any selection of items that exceed the budget b of the constraint and this selection is said to be a minimal cover if, upon removing of any item of the selection, the constraint becomes satisfied. This logic allows us to design a way to generate valid inequalities using covers. This is the main result in Proposition 10.10.

Proposition 10.10. *If $C \subseteq N$ is a cover for X, then a valid cover inequality for X is*

$$\sum_{j \in C} x_j \leq |C| - 1.$$

Proof. Let $R = \left\{ j \in N : x_j^R = 1 \right\}$, for $x^R \in X$. If $\sum_{j \in C} x_j^R > |C| - 1$, then $|R \cap C| = |C|$ and $C \subseteq R$. Thus, $\sum_{j \in N} a_j x_j^R = \sum_{j \in R} a_j > b$, which violates the inequality and implies that $x^R \notin X$. □

The usefulness of Proposition 10.10 becomes evident if C is a minimal cover. Let us consider a numerical example to illustrate this. Consider the knapsack set

$$X = \left\{ x \in \{0,1\}^7 : 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19 \right\}.$$

The following are minimal cover inequalities for X.

$$x_1 + x_2 + x_3 \leq 2$$
$$x_1 + x_2 + x_6 \leq 2$$
$$x_1 + x_5 + x_6 \leq 2$$
$$x_3 + x_4 + x_5 + x_6 \leq 3$$

Notice that we would obtain a non-minimal cover C' by adding an inequality $x_i \leq 1$, $i \in C' \setminus C$, to a minimal cover inequality of C, which makes the cover inequality somewhat redundant. For example, adding $x_7 \leq 1$ to a minimal cover inequality $x_1 + x_2 + x_3 \leq 2$ of $C = \{1, 2, 3\}$ yields a non-minimal cover $C' = \{1, 2, 3, 7\}$ with inequality $x_1 + x_2 + x_3 + x_7 \leq 3$.

There is a simple way to strengthen cover inequalities, using the the notion of *extended* cover inequalities. One can extend a cover inequality by expanding the set C with elements that have a coefficient a_j, $j \in N \setminus C$ greater or equal than all coefficients a_i, $i \in C$. This guarantees that a swap between elements must happen for the inequality to be feasible, meaning that the righthand side of the constraint remains $|C| - 1$. This is summarised in Proposition 10.11.

Proposition 10.11. *If C is a cover for X, the extended cover inequality*

$$\sum_{j \in E(C)} x_j \leq |C| - 1$$

with $E(C) = C \cup \{j \in N : a_j \geq a_i, \forall i \in C\}$ is valid for X.

We leave the proof as a thought exercise. Let us however illustrate this using the previous numerical example. For $C = \{3, 4, 5, 6\}$, $E(C) = \{1, 2, 3, 4, 5, 6\}$, yielding the inequality

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$$

which is stronger than $x_3 + x_4 + x_5 + x_6 \leq 3$ as the former dominates the latter, cf. Definition 10.7.

10.6 Exercises

Exercise 10.1: Chvátal-Gomory (C-G) procedure

Consider the set $X = P \cap \mathbb{Z}^n$ where $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ and in which A is an $m \times n$ matrix with columns $\{A_1, \ldots, A_n\}$. Let $u \in \mathbb{R}^m$ with $u \geq 0$. The Chvátal-Gomory (C-G) procedure to construct valid inequalities for X uses the following 3 steps:

1. $\sum_{j=1}^{n} uA_j x_j \leq ub$ is valid for P, as $u \geq 0$ and $\sum_{j=1}^{n} A_j x_j \leq b$.

2. $\sum_{j=1}^{n} \lfloor uA_j \rfloor x_j \leq ub$ is valid for P, as $x \geq 0$.

3. $\sum_{j=1}^{n} \lfloor uA_j \rfloor x_j \leq \lfloor ub \rfloor$ is valid for X, as any $x \in X$ is integer and thus $\sum_{j=1}^{n} \lfloor uA_j \rfloor x_j$ is integer.

Show that every valid inequality for X can be obtained by applying the Chvátal-Gomory procedure a finite number of times.

Hint: We show this for the 0-1 case. Thus, let $P = \{x \in \mathbb{R}^n : Ax \leq b, 0 \leq x \leq 1\}$, $X = P \cap \mathbb{Z}^n$, and suppose that $\pi x \leq \pi_0$ with $\pi, \pi_0 \in \mathbb{Z}$ is a valid inequality for X. We show that $\pi x \leq \pi_0$ can be obtained by applying Chvátal-Gomory procedure a finite number of times. We do this in parts by proving the following claims **C1**, **C2**, **C3**, **C4**, and **C5**.

C1. An inequality $\pi x \leq \pi_0 + t$ with $t \in \mathbb{Z}_+$ is valid for P.

C2. For a large enough $M \in \mathbb{Z}_+$, the inequality

$$\pi x \leq \pi_0 + M\Big(\sum_{j \in N^0} x_j + \sum_{j \in N^1} (1 - x_j)\Big) \tag{10.3}$$

is valid for P for every partition (N^0, N^1) of N.

C3. If $\pi x \leq \pi_0 + \tau + 1$ is a valid inequality for X with $\tau \in \mathbb{Z}_+$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in N^0} x_j + \sum_{j \in N^1} (1 - x_j) \tag{10.4}$$

is also a valid inequality for X for every partition (N^0, N^1) of N.

C4. If

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0 \cup \{p\}} x_j + \sum_{j \in T^1} (1 - x_j) \tag{10.5}$$

and

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1 \cup \{p\}} (1 - x_j) \tag{10.6}$$

are valid inequalities for X, where $\tau \in \mathbb{Z}_+$ and (T^0, T^1) is any partition of $\{1, \ldots, p-1\}$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1} (1 - x_j) \tag{10.7}$$

is also a valid inequality for X.

C5. If

$$\pi x \leq \pi_0 + \tau + 1 \tag{10.8}$$

is a valid inequality for X with $\tau \in \mathbb{Z}_+$, then

$$\pi x \leq \pi_0 + \tau \tag{10.9}$$

is also a valid inequality for X.

Finally, after proving the claims **C1** - **C5**, if we start with $\tau = t - 1 \in \mathbb{Z}_+$ and successively apply **C5** for $\tau = t-1, \ldots, 0$, turning each valid inequality (10.8) of X into a new one (10.9), it leads to the inequality $\pi x \leq \pi_0$ which is valid for X. This shows that every valid inequality $\pi x \leq \pi_0$ of X with $\pi, \pi_0 \in \mathbb{Z}_+$ can be obtained by applying the C-G procedure a finite number of times.

Exercise 10.2: Chvátal-Gomory (C-G) procedure example

(a) Consider the set $X = \{\mathbf{x} \in \mathbb{B}^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2\}$. Derive the following inequalities as C-G inequalities:

(*i*) $x_2 + x_4 \geq 1$

(*ii*) $x_1 \leq x_2$

(b) Consider the set $X = \{x \in \mathbb{B}^4 : x_i + x_j \leq 1 \text{ for all } i, j \in \{1, \ldots, 4\} : i \neq j\}$. Derive the clique inequalities $x_1 + x_2 + x_3 \leq 1$ and $x_1 + x_2 + x_3 + x_4 \leq 1$ as C-G inequalities.

Exercise 10.3: Cuts from the simplex tableau

Consider the integer programming problem IP:

$$\begin{aligned} (IP) \quad \max_{x_1,x_2} \quad & 2x_1 + 5x_2 \\ \text{s.t.:} \quad & 4x_1 + x_2 \leq 28 \\ & x_1 + 4x_2 \leq 27 \\ & x_1 - x_2 \leq 1 \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

The LP relaxation of the problem IP is obtained by relaxing the integrality constraints $x_1, x_2 \in \mathbb{Z}_+$ to $x_1 \geq 0$ and $x_2 \geq 0$. The LP relaxation of IP in standard form is the problem LP:

$$\begin{aligned} (LP) \quad \max_{x_1,x_2} \quad & 2x_1 + 5x_2 \\ \text{s.t.:} \quad & 4x_1 + x_2 + x_3 = 28 \\ & x_1 + 4x_2 + x_4 = 27 \\ & x_1 - x_2 + x_5 = 1 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

The optimal Simplex tableau after solving the problem LP with primal Simplex is

x_1	x_2	x_3	x_4	x_5	RHS
0	0	-1/5	-6/5	0	-38
1	0	4/15	-1/15	0	17/3
0	1	-1/15	4/15	0	16/3
0	0	-1/3	1/3	1	2/3

(a) Derive two fractional Gomory cuts from the rows of x_1 and x_5, and express them in terms of the original variables x_1 and x_2.

(b) Derive the same cuts as in part (a) as Chvátal-Gomory cuts. *Hint:* Use Proposition 5 from Lecture 9. Recall that the bottom-right part of the tableau corresponds to $B^{-1}A$, where B^{-1} is the inverse of the optimal basis matrix and A is the original constraint matrix. You can thus obtain the matrix B^{-1} from the optimal Simplex tableau, since the last three columns of A form an identity matrix.

Exercise 10.4: More Gomory cuts

Consider the following integer programming problem IP:

$$\begin{aligned}(IP): \max.\ z = \ & x_1 + 2x_2 \\ \text{s.t.:}\ & -3x_1 + 4x_2 \leq 4 \\ & 3x_1 + 2x_2 \leq 11 \\ & 2x_1 - x_2 \leq 5 \\ & x_1, x_2 \in \mathbb{Z}_+\end{aligned}$$

Solve the problem by adding Gomory cuts to the LP relaxation until you find an integer solution.

Exercise 10.5. The cover separation problem

Consider the 0-1 knapsack set :

$$X = \left\{ x \in \{0,1\}^7 : 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19 \right\}$$

and a solution $\bar{x} = (0, 2/3, 0, 1, 1, 1, 1)$ to its LP relaxation. Find a cover inequality cutting out (violated by) the fractional solution $\bar{x}$.

Chapter 11

Mixed-integer programming solvers

11.1 Modern mixed-integer linear programming solvers

In this chapter, we will discuss some of the numerous features that are shared by most professional-grade implementation of mixed-integer programming (MIP) solver. As it will become clear, MIP solvers are formed by an intricate collection of techniques that have been developed through the last few decades. The continuous improvement and development of new such techniques have enabled performance improvements beyond purely hardware performance progress. In fact, this is a very lively and exciting research area, with new features being proposed and incorporated in these solvers with frequent new releases of these tools.

The main difference between MIP solver implementations is which "tricks" and techniques they have implemented. In some cases, these are not disclosed in full detail, since the high-performing solvers are commercial products subject to trade secrets. Luckily, some open-source (such as CBC and HiGHS) and free-to-use alternatives (such as SCIP) have been made available, but they are not up to par with commercial implementations in terms of performance as yet.

We will focus on describing the most important techniques forming a professional-grade MIP solver implementation. Most MIP solvers allow for significant tuning and on-off toggling of these techniques. Therefore, knowing the most important techniques and how they work can be beneficial in configuring MIP solvers to your own needs.

Most MIP solvers implement a method that is called *branch-and-cut* which consists of a combination of the linear-programming (LP)-based branch-and-bound method (as described in Chapter 9) and a cutting-plane method (as described in Chapter 10) that is employed at the root note (or the first subproblem LP relaxation) and possibly at later nodes as well. Figure 11.1 illustrates the typical flowchart of a MIP solver algorithm.

The first phase consists of a preprocessing phase called *presolve*. In that, the problem formulation is analysed to check whether redundant constraints or loose variables can be trivially removed. In addition, more sophisticated techniques can be employed to try to infer the optimal value of some variables via logic or to tighten their bounds. For simple enough problems, the presolve might be capable of returning an optimal solution or a certificate that the problem is infeasible or unbounded.

Then, the main solution loop starts, similarly to what we have described in Chapter 9 when discussing the branch-and-bound method. A node selection method is employed and the LP relaxation is solved. Then, branching is applied and the process continues until an optimal solution has been found.

The main difference however relates to the extra *Cuts* and *Heuristics* phases. Together with the

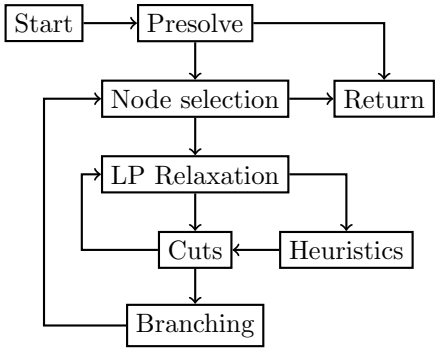


Figure 11.1: The flowchart of a typical MIP solver. The nodes represent phases of the algorithm

presolve, this is likely the phases that most differ between implementations of MIP solvers. The cut phase consists of the employment of a cutting-plane method onto the current LP relaxation with the aim of either obtaining an integer solution (and thus pruning the branch by optimality) or strengthening the formulation of the LP relaxation, as discussed in Chapter 10. Each solver will have their own family of cuts that is used in this phase, and typically a collection of them are used simultaneously. The heuristics phase is used in combination to try to obtain primal feasible solutions from the LP relaxations (possibly augmented by cuts) so primal bounds (integer and feasible solution values) can be obtained and broadcasted to the whole search tree, hopefully fostering pruning by bound.

In what follows, we will discuss the main techniques in each of these phases.

11.2 Presolving methods

Presolving (or preprocessing) methods are methods typically employed before the start of the branch-and-cut method. These methods have three main goals: (i) reducing the problem size by fixing variables and eliminating constraints; (ii) strengthening the LP relaxation by identifying bounds and coefficients that can be tightened; and (iii) exploiting integrality to improve formulation and identify problem structures (e.g., knapsack or assignment structures)

Detecting infeasibility and redundancy

Many techniques used in the preprocessing phase relies on the notion of constraint activity. Consider the constraint $a^\top x \leq b$, with $x \in \mathbb{R}^n$ as a decision variable vector, $l \leq x \leq u$, $b \in \mathbb{R}$, where (a, b, l, u) are given. We say that the *minimum activity* of the constraint is given by

$$\alpha_{\min} = \min\left\{a^\top x : l \leq x \leq u\right\} = \sum_{j:a_j>0} a_j l_j + \sum_{j:a_j<0} a_j u_j.$$

Analogously, the *maximum activity* of a constraint is given by

$$\alpha_{\max} = \max\left\{a^\top x : l \leq x \leq u\right\} = \sum_{j:a_j>0} a_j u_j + \sum_{j:a_j<0} a_j l_j$$

Notice that the constraint activity is simply capturing what is the minimum and maximum (respectively) values the left-hand side of $a^\top x \leq b$ could assume. This constraint activity can be used

in number of ways. For example, if there is a constraint for which $\alpha_{\min} > b$, then the problem is trivially *infeasible*. On the other hand, if one observes that $\alpha_{\max} \leq b$ for a given constraint, then the constraint can be safely removed since it is guaranteed to be redundant.

Bound tightening

Another important presolving method is bound tightening, which, as the name suggests, tries to tighten lower and upper bounds of variables, thus strengthening the LP relaxation formulation. There are alternative ways that this can be done, and they typically trade off how much tightening can be observed and how much computational effort they require.

One simple way of employing bound tightening is by noticing the following. Assume, for simplicity, that $a_j > 0 \; \forall j \in J$. Then, we have that

$$\alpha_{\min} = \sum_{j \in J} a_j l_j = a^\top l \leq a^\top x \leq b,$$

where $l \in \mathbb{R}^{|J|}$. From the second inequality, we obtain

$$a_j x_j \leq b - a^\top x + a_j x_j.$$

Let $a^\top = (\overline{a}, a_j)^\top$, $x = (\overline{x}, x_j)$ and $l = (\overline{l}, l_j)$, then we have $\alpha_{\min} = a^\top l = \overline{a}^\top \overline{l} + a_j l_j$ and $a^\top x = \overline{a}^\top \overline{x} + a_j x_j$. By the definition of *minimum activity*, we know that $\overline{\alpha}_{\min} = \overline{a}^\top \overline{l} \leq \overline{a}^\top \overline{x}$, which is equivalent to

$$\alpha_{\min} - a_j l_j \leq a^\top x - a_j x_j.$$

This can be reformulated as

$$b - a^\top x + a_j x_j \leq b - \alpha_{\min} + a_j l_j.$$

Combining this result with what was obtained from the second inequality, we have that

$$a_j x_j \leq b - a^\top x + a_j x_j \leq b - \alpha_{\min} + a_j l_j,$$

from which we can extract the bound

$$x_j \leq \frac{b - \alpha_{\min} + a_j l_j}{a_j},$$

that is, an upper bound for x_j. The procedure can be analogously adapted to obtain a lower bound as well. Furthermore, rounding can be employed in the presence of integer variables.

Another common bound tightening technique consists of solving a linear programming subproblem for each variable variable x_j, $j = 1, \ldots, n$. Let

$$IP : \text{min. } \left\{ c^\top x : x \in X = P \cap \mathbb{Z}^n \right\}$$

where P is a polyhedral set. Then, optimal solution value of the subproblem

$$LP_{x_j} : \text{min. } \{ x_j : x \in P \}$$

provides a lower bound for x_j that considers all possible constraints at once. Analogously, solving LP_{x_j} as a maximisation problem yields an upper bound. Though this can be done somewhat efficiently, this clearly has steeper computational requirements.

Coefficient tightening

Differently from bound tightening, coefficient tightening techniques aim at improving the strength of existing constraints. The simplest form consists of the following. Let $a_j > 0$, $x_j \in \{0, 1\}$, such that $\alpha_{\max} - a_j < b$. If such coefficients are identified, then the constraint

$$a_j x_j + \sum_{j':j' \neq j} a_{j'} x_{j'} \leq b$$

can be modified to become

$$(\alpha_{\max} - b)x_j + \sum_{j':j' \neq j} a_{j'} x_{j'} \leq (\alpha_{\max} - a_j).$$

Notice that the modified constraint is valid for the original integer set, while dominating the original constraint (cf. Definition 10.7 since, on the lefthand side, we have that $\alpha_{\max} - b < a_j$ and on the righthand side wee have $\alpha_{\max} - a_j < b$).

Other methods

There are a wide range of methods employed as preprocessing, and they vary greatly among different solvers, and even different modelling languages. Thus, compiling an exhaustive list is no trivial feat. Some other common methods that are employed include:

- **Merge of parallel rows and columns**: methods implemented to identify pairs of rows (constraints) and columns (variables) with constant proportionality coefficient (i.e., are linearly dependent) and merge them into a single entity, thus reducing the size of the model.
- **Domination tests between constraints**: heuristics that test whether domination between selected constraints can be asserted so that some constraints can be deemed redundant and removed.
- **Clique merging**: a clique is a subset of vertices of graph that are fully connected. Assume that $x_j \in \{0, 1\}$ for $j \in \{1, 2, 3\}$ and that the three constraints hold:

  $$x_1 + x_2 \leq 1$$
  $$x_1 + x_3 \leq 1$$
  $$x_2 + x_3 \leq 1.$$

  Then, one can think of these constraints forming a clique between the imaginary nodes 1, 2, and 3, which renders the clique cut $x_1 + x_2 + x_3 \leq 1$. Many other ideas using this graph representation of implication constraints, known as *conflict graphs* are implemented in presolvers.
- **Greatest common denominator (GCD) reduction**: We can use the GCD of the coefficients $a = [a_1, \ldots, a_n]$ to generate or tighten inequalities. Let $\mathbf{gcd}(a)$ be the GCD of all coefficients a_j in a. Then we can generate the valid inequality

  $$\sum_{j=1}^{n} \frac{a_j}{\mathbf{gcd}(a)} x_j \leq \left\lfloor \frac{b}{\mathbf{gcd}(a)} \right\rfloor.$$

Some final remarks are worth making. Most solvers might, at some point in the process of solving the MIP refer to something called *restart*, which consists of reapplying some or all of the techniques associated with the preprocessing phase after a few iterations of the branch-and-cut process. This can be beneficial since in the solution process new constraints are generated (cuts) which might lead to new opportunities for reducing the problem or further tightening bounds.

In addition, conflict graphs can contain information that can be exploited in a new round of preprocessing and transmitted across all search tree, a process known as propagation. Conflict graphs and propagation are techniques originally devised for constraint programming and satisfiability (SAT) problems, but have made considerable inroads in MIP solvers as well.

11.3 Cut generation

One major component of MIP solvers is its cut generation procedures. In practice, MIP solvers implement what is called *branch-and-cut*, which is a method formed by the amalgamation of branch-and-bound (as we seen in Chapter 9) and cutting-plane methods (Chapter 10). In fact, the combination of both methods is arguably the most important methodological development that ultimately rendered MIP solvers reliable to be used in practice in many applications.

There is an interesting symbiotic relationship emerging from the combination of the two methods. In general, cutting planes are useful in tightening the existing formulation but often fail in being able to generate all the cuts required to expose an integer optimal solution. On the other hand, branch-and-bound might require considerable computational efforts to adequately expose the integer optimal solutions of all subproblems, but is guaranteed to converge in finite time (even if this time is not feasible in a practical sense). Thus, the combination of both allows for a method that is far more likely to terminate within reasonable computation time.

However, some care must be considered when generating cuts in branch-and-cut settings, since it can quickly increase the dimension of the subproblems, which would lead to amplified consequences to the performance of the branch-and-bound part. This is mitigated with a feature called *cut pool*, which consists of a way to make sure that only selected cuts, e.g., cuts that are violated by the solution of the relaxation, are considered in the subproblem. Such cuts are chosen using what is referred to as cut selection routines.

Furthermore, most professional-grade solvers allow for the definition of user cuts, which are user-defined cuts that are strongly problem specific, but give strong customisation powers to advanced users. As it turns out, the types of cuts available is one of the main differentiators between solvers and the search for cuts that are both efficient and generic is an active research and development direction.

The most common types of cuts utilised include fractional (or Gomory) cuts, of the form

$$\sum_{j \in I_N} f_{ij} x_j \geq f_{i0}, \text{ where } f_{ij} = \overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor$$

and Chvátal-Gomory cuts for pure integer constraints, which are dependent on the heuristic or process used to define the values of the multipliers u in

$$\sum_{j=1}^{n} \lfloor uA_j \rfloor x_j \leq \lfloor ub \rfloor.$$

For example, zero-half cuts (with $u \in [0, 1/2]$) and mod-k cuts ($u \in \{0, 1/k, \ldots, (k-1)/k\}$) are available in most solvers. Other cuts such as knapsack (or cover) inequalities, mixed-integer round-

ing, clique as also common and, although it can be shown to be related to the Chvátal-Gomory procedure, are generated by means of heuristics (normally referring to a cut generation procedure).

11.3.1 Cut management: generation, selection and discarding

With the numerous possibilities of cuts to be generated, it becomes clear that one must be mindful of the total of cuts generated and its associated trade-offs. These trade-offs are managed taking into account which cuts to generate and which cuts to select from the cut pool to be added to the problem. In particular, in case of the generation and selection of cuts, this can often be influenced by users, by selecting how "aggressively" cuts are generated.

For example, although dual simplex is perfectly suited for resolving the linear programming (LP) relaxation after the addition of cuts, if multiple cuts are added at once, it might mean that resolving the LP relaxation from the previous dual optimal basis is not much more beneficial than resolving the LP relaxation from scratch. Moreover, adding cuts increases the size of the basis in the problem, which in turn increases the size of the basic matrix B and its inverse $B^{-}1$, thus gradually increasing the time per iteration of the method. Lastly, some cuts might require the solution of a demanding separation problem, e.g., lifted cover inequalities. Although these can provide considerable improvements to the formulation, the computational effort required for generating them might dampen the performance improvement they incur.

Selecting which cuts would be the most efficient is not an easy task. Ideally, we would like to chose the minimal number of cuts that has the most impact in terms of improving the LP relaxation. Cleary, we can only try to achieve this via proxies. Solvers normally associate a scoring to each cut taking into account a collection of criteria and these scores are then used in the process of selecting the cuts (their cut selection routines), e.g., selecting a fraction of the top scorer cuts or discarding those that the score is below a certain threshold.

One criteria for trivially discarding cuts is numerical stability. Cuts with too large or too small coefficients are prone to cause numerical issues due to the matrix decomposition methods employed. Therefore, these can be easily disregarded in the selection process.

An important proxy for efficiency is the notion of *depth* of the cut. That is, a cut can have its depth measured by the distance between the hyperplane that forms the cut and the solution of the LP relaxation. The larger this distance, the "deeper" the cut is cutting through the LP relaxation, which could potentially mean that the cut is more efficient.

Another important proxy is *orthogonality*. Cuts that are pairwise orthogonal with other cuts are likely to be more effective. This is easy to see if you think of the extreme case of zero orthogonality, or the cuts being parallel, clearly meaning that one of the cuts is dominated by the other. Orthogonality can also be measured against the objective function, in which case we are interested in cuts that are almost parallel (but not exactly, as this would lead to numerical issues) to the objective function since those are more likely to cause improvement in the dual bound (LP relaxation optimal) value.

11.4 Variable selection: branching strategy

As we discussed in Chapter 9, some decisions in terms of selecting variables to branch and subproblems to solve can have a great impact in the total number of subproblems solved in a branch-and-bound method. Variable selection is still a topic under intensive research, with newer ideas only recently being made available in the main solvers.

Variable selection, commonly referred to as branching strategies in most MIP solver implementations, refer to the decision of which of the currently fractional variables should be chosen to generate subproblems. There are basic three main methods most commonly used, which we discuss next. Furthermore, most MIP solvers allow for the user to set priority weights to the variables, which defines priority orders for variable selection. These can be useful for example when the user knows that the problem possess a dependence structure between variables (e.g., location and allocation variables, where allocation can only happen if the location decision is made) that the solver cannot infer automatically.

Maximum infeasibility

The first branching strategy, sometimes called *maximum infeasibility*, consists of choosing the variable with the fractional part as close to 0.5 as possible, or, more precisely select the variables $j \in \{1, \dots, n\}$ as

$$\arg \max_{j \in \{1,\dots,n\}} \min \{f_j, 1 - f_j\}$$

where $f_j = x_j - \lfloor x_j \rfloor$. This in effect is trying to reduce as most as possible the infeasibility of the LP relaxation solution, which in turn would more quickly lead to a feasible (i.e., integer) solution. A analogous form, called *minimum infeasibility* is often available, and as the name suggests, focus on selecting the variables that are closer to be integer valued.

Strong branching

Strong branching can be understood as an explicit look-ahead strategy. That is, to decide which variable to branch on, the method performs branching on all possible variables, and choose that provides the best improvement on the dual (LP relaxation) bound. Specifically, for each fractional variable, we can x_j, solve the LP relaxations corresponding to branching options $x_j \leq \lfloor x^{\text{LP}} \rfloor$ and $x_j \geq \lceil x^{\text{LP}} \rceil$ and then choose the fractional variable x_j that leads to subproblems the best LP relaxation objective value.

As you might suspect, there is a trade-off related to the observed reduction in the number of nodes explored, given by the more prominent improvement of the dual bound, and how computationally intensive is the method. There are however ideas that can exploit this trade off more efficiently. First, the solution of the subproblems might yield information related to infeasibility and pruning by limit, which can be used in favour of the method.

Another idea is to limit the number of simplex iterations performed when solving the subproblems associated with each branching option. This allows for using approximate solution of the subproblems and potential savings in computational efforts. Some solvers offer a parameter that allow the user to set this iteration limit value.

Pseudo-cost branching

Pseudo-cost branching relies on the idea of using past information from the search process to estimate gains from branching on a specific variable. Because of this reliance on past information, the method tends to be more reliable later in the search tree, where more information has been accumulated on the impact of choosing a variable for branching.

These improvement estimates are the so called *pseudo-costs*, which compile an estimate on much the dual (LP relaxation) bound has improved per fractional unit of the variable that has been

reduced. More specifically, let

$$f_j^- = x_j^{\text{LP}} - \lfloor x_j^{\text{LP}} \rfloor \text{ and } f_j^+ = \lceil x_j^{\text{LP}} \rceil - x_j^{\text{LP}}. \tag{11.1}$$

Then, we can define the quantities Ψ_j^- and Ψ_j^+ to be the average improvement in the dual bound observed per fractional unit reduced and increased, respectively, whenever the variable x_j has been selected for branching, i.e, for each branching direction. Notice that this requires that several subproblems to be solved for a reliable estimate to be available.

Then, we can define the quantities

$$\Delta_j^- = f_j^- \Psi_j^- \text{ and } \Delta_j^+ = f_j^+ \Psi_j^+ \tag{11.2}$$

which represent the estimated change to be observed when selecting the variable x_j for branching, based on the current fractional parts f_j^+ and f_j^-. In effect, these are considered in a branching score, with the branching variable being selected as, for example,

$$j = \underset{j=1,\ldots,n}{\text{argmax}} \left\{ \alpha \min \left\{ \Delta_j^-, \Delta_j^+ \right\} + (1-\alpha) \max \left\{ \Delta_j^-, \Delta_j^+ \right\} \right\}.$$

where $\alpha \in [0, 1]$. Setting the value of α trades off two aspects. Assume a maximisation problem. Then, setting α closer to zero will slow down *degradation*, which refers to the decrease of the upper bound (notice that the dual bound is decreasing and thus, Δ^+ and Δ^- are negative). This strategy improves the chances of finding a good feasible solution on the given branch, and, in turn, potentially fostering pruning by bound. In contrast, setting α closer to one increases the rate of decrease (improvement) of the dual bound, which can be helpful for fostering pruning once a good global primal bound is available. Some solvers allow for considering alternatives branching score functions.

As one might suspect, it might take several iterations before reliable estimates Ψ^+ and Ψ^- are available. The issue with unreliable pseudo-costs can be alleviated with the use of a hybrid strategy known as *reliability* branching[1]. In that, variables that are deemed unreliable for not having been selected for branching a minimum number of times $\eta \in [4, 8]$, have strong branching employed instead.

GUB branching

Constraints of the form

$$\sum_{j=1}^{k} x_j = 1$$

are referred to as special ordered sets 1 (or SOS1) which, under the assumption that $x_j \in \{0, 1\}$, $\forall j \in \{1, \ldots, k\}$ implies that only one variable can take value different than zero. Notice you may have SOS1 sets involving continuous variables, which, in turn would require the use of binary variables to be modelled appropriately.

Branching on these variables might lead to unbalanced branch-and-bound trees. This is because the branch in which x_j is set to a value different than zero, immediately define the other variables to be zero, leading to an early pruning by optimality or infeasibility. In turn, unbalanced trees are undesirable since they preclude the possibility of parallelisation and might lead to issues related to searches that focuses on finding leaf notes quickly.

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin (2005), Branching rules revisited, Operations Research Letters

To remediate this, the idea of using a generalised upper bound is employed, leading to what is referred to as GUB branching (with some authors referring to this as SOS1 branching). A generalised upper bound is an upper bound imposed on the sum of several variables. In GUB branching, branching for binary variables is imposed considering the following rule:

$$S_1 = S \cap \{x : x_{j_i} = 0, \forall i \in \{1, \ldots, r\}\}$$
$$S_2 = S \cap \{x : x_{j_i} = 0, \forall i \in \{r+1, \ldots, k\}\}$$

where $r = \text{argmax}_{t \in \{1,\ldots,k\}} \sum_{j=1}^{t} x_j \leq 0.5$. Notice the upper bounding on the some of the variables, from which the name stems. That is, only a subset of the variables are forced to be zero, while several others are left unconstrained, which favours more balanced search trees. As a final remark, constraint of the form of

$$\sum_{j=1}^{k} x_j \leq 1$$

can also benefit of the use of GUB branching, with the term GUB being perhaps better suited in this case.

11.5 Node selection

We now focus on the strategies associated with selecting the next subproblem to be solved. As we have seen in Chapter 9, the order in which the subproblem's LP relaxations are solved can have a major impact on the total number of nodes explored, and, consequently, on the efficiency of the method. The alternative strategies for node selection typically trade off the following

1. Focus on finding primal feasible solutions earlier, to foster pruning by bound.

2. Alternatively, focus on improving the dual bound faster, hoping that once an integer solution is found, more pruning by bound is possible.

3. Increase *ramp-up*, which means increase the number of unsolved nodes in the list of subproblems so that these might be solved in parallel. For that, the nodes must be created, and the faster they are opened, the earlier parallelisation can benefit the search.

4. Minimise computational effort by minimising the overhead associated with changing the subproblems to be solved. That means that children nodes are preferred over other nodes, in a way to minimise the changes needed to assemble a starting basis for the dual simplex method.

You might notice that points 1 and 2 are conflicting, since while the former would benefit from a breadth-focused search (that is, having wider trees earlier is preferable than deeper trees) the latter would benefit from searches that dive deeply in the tree searching from leaf nodes. Points 3 and 4 pose exactly the same dilemma: the former benefits from breadth-focusing searches while the latter benefits from depth-focusing searches.

The main strategies for node selection are, to a large extent, ideas to emphasise each or a combination of the above.

Depth-first search (DFS) and breadth-first search (BFS)

A depth-first search focus on diving down in the search tree, prioritising nodes at lower levels. It has the effect of increasing the chances of finding leaves, and potentially primal feasible solutions, earlier. Furthermore, because the problems being successively solved are very similar, with the exception of one additional branching constraint, the dual simplex methods can be efficiently restarted and often fewer iterations are needed to find the optimal of the children subproblem relaxation. On the other hand, as a consequence of the way the search is carried, it is slower in populating the list of subproblems.

In contrast, breadth-first search gives priorities to nodes higher levels in the tree, ultimately causing a horizontal spread of the search tree. This has as consequence a faster improvement of the dual bound, at the expense of potentially delaying the generation of primal feasible solutions. This also generates more subproblems quickly, fostering diversification (more subproblems and potentially more information to be re-utilised in repeated rounds of preprocessing) and opportunities for parallelisation.

Best bound

Best bound consists of the strategy of choosing the next node to be that with the best dual (LP relaxation) bound. It leads to a breadth-first search pattern, but with flexibility to allow potentially good nodes that are in deeper levels of the tree to be selected.

Ultimately, this strategy foster a faster improvement of the dual bound, but with a higher overhead on the set up of the subproblems, since they can be quite different than each other in terms of its constraints. One way to mitigate this overhead is perform *diving* sporadically, which consists of, after choosing a node by best bound, temporarily switching to a DFS search for a few iterations.

Best estimate or best projection

Uses a strategy similar to that employing pseudo-costs to choose which variable to branch on. However, instead of focusing on objective function values, it uses estimates of the node progress towards feasibility but relative to its bound degradation.

To see how this works, assume that the parent node has been solved, and a dual bound z_D is available. Now, using our estimates in (11.2), we can calculate an estimate of the potential primal feasible solution value E, given by

$$E = z_D + \sum_{j=1}^{n} \min\left\{\Delta_j^-, \Delta_j^+\right\}.$$

The expression E is an estimate of what is the best possible value an integer solution could have if it were to be generated by rounding the LP relaxation solution. These estimates can also take into account the feasibility per se, trying to estimate feasibility probabilities considering known feasible solutions and how fractional the subproblem LP relaxation solution is.

11.6 Primal heuristics

The last element of MIP solvers we must consider are primal heuristics. The term "primal" refers to the fact that these are methods geared towards either (i) building primal feasible solutions normally

from a solution obtained from a relaxation; or (ii), geared towards improving on previously known primal solutions. The former strategy is often referred to as *constructive* heuristics, while the latter are called *improvement* heuristics.

The name heuristic refers to the fact that these are methods that are not guided by any optimality certificates per se, but yet by performing local (or nearby, according to a given metric of solution difference) improvements repeatedly.

Primal heuristics play three main important roles in MIP solver algorithms. First, they are employed in the preprocessing phase to verify whether the model can be proved to be feasible, by constructing a primal feasible solution. Second, constructive heuristics are very powerful in generating feasible solutions during the branch-and-bound phase, meaning that it can make primal feasible solutions available *before* they are found at leaf nodes when pruning by optimality (i.e., with integer LP relaxation solution), and therefore fostering early pruning by bound. Lastly, heuristics are a powerful way to obtain reasonably (often considerably) good solutions, which in practical cases, might be sufficient given computational or time limitations and precision requirements.

11.6.1 Diving heuristics

Diving heuristics are used in combination with node selection strategies that search in breadth (instead of depth). In simple terms, it consists of performing a local depth search at the node being considered with no or very little backtracking, with the hope of reusing the subproblem structure while searching for primal feasible solutions. The main difference is that the subproblems are generated in an alternative tree in which branching based on rounding and fixing instead of the standard branching we have seen in Chapter 9.

Once the heuristic terminates, the structure is discarded, but the solution, if found, is kept. Notice that the diving can also be preemptively aborted if it either renders an infeasible subproblem or if it leads to a relaxation with worse bound than a known primal bound from an incumbent solution. Another common termination criteria consists in limiting the total number of LP iterations solving the subproblems or the total number of subproblems solved.

The most common types of rounding employed in diving heuristics include *fractional diving*, in which the variable selected for rounding is simply that with the smallest fractional component, i.e., x_j is chosen, such that the index j is given by

$$\underset{i=1,\dots,n}{\operatorname{argmax}} \left\{ \min \left\{ x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i \right\} \right\}.$$

Another common idea consists of selecting the variables to be rounded by considering a reference solution, which often is an incumbent primal feasible solution. This *guided dive* is then performed by choosing the variable with the smallest fractional value when compared against this reference solution.

A third idea consists of taking into account the number of *locks* associated with the variable. The locks refer to the number of constraints that are potentially made infeasible by rounding up or down a variable. This potential infeasibility stems from taking into count the coefficient of the variable, the type of constraint constraint and whether rounding it up or down can potentially cause infeasibility. This is referred to as *coefficient diving*.

11.6.2 Local searches and large-neighbourhood searches

Local and large-neighbourhood searches are, contrary to most diving heuristics, improvement heuristics. In these, a reference solution is used to to search for new solutions within its *neighbourhood*. This is simply the idea of limiting any solutions found to share a certain number of elements (for example, have the same values in some components) with the reference solution. We say that a solution is a k-neighbour solution if they share k components.

Local search and large-neighbourhood search simply differ in terms of scope; the former allows for only localised change while the latter considers wider possibilities for divergence from the reference solution.

Some of search are seldom used and often turned off by default in professional-grade solvers. They tend to be expensive under a computational standpoint because they require considerable extra work. Most of them are based on fixing and/or relaxing integrality of a number of variables, adding extra constraints and/ or changing objective functions, and resolving which can be considerably onerous. Let us present the most common heuristics found in professional grade solvers.

Relaxation-induced neighbourhood search (RINS) and Relaxation-enforced neighbourhood search

The relaxation-induced neighbourhood search (or RINS) is possibly the most common improvement heuristic available in modern implementations[2]. The heuristic tries to find solutions that present a balance between proximity of the current LP solution, hoping this would improve the solution quality, and proximity to an incumbent (i.e., best-known primal feasible) solution, emphasising feasibility.

In a nutshell, the method consists of the following. After solving the LP relaxation of a node, suppose we obtain the solution x^{LP}. Also, assume we have at hand an incumbent solution $\overline{x}$. Then, we form an auxiliary MIP problem in which we fix all variables coinciding between x^{LP} and $\overline{x}$. This can be achieved by including the constraints

$$x_j = \overline{x}_j,\ \forall j \in \{1, \ldots, n\} : \overline{x}_j = x^{\mathrm{LP}},$$

which, in effect fix these variables to integer values and remove them from the problem, as they can be converted to parameters (or input data). Notice that this constraints the feasible space to be in the (potentially large) neighbourhood of the incumbent solution. In later iterations, when more of the components of the relaxation solutions x^{LP} are integer, this becomes a more local search, with less degrees of freedom. Finally, this additional MIP is solved and, in case an optimal solution is found, a new incumbent solution might become available.

In contrast, relaxation-enforced neighbourhood search (or RENS) is a constructive heuristics, which has not yet seem a wider introduction in commercial-grade solvers, though it is available in CBC and SCIP[3].

The main differences between RINS and RENS are the fact that no incumbent solution is considered (hence the dropping of the term "induced") but rather the LP relaxation solution x^{LP} fully defines the neighbourhood (explaining the name "enforced").

Once again, let us assume we obtain the solution x^{LP}. And again, we fix all integer valued variables

[2] Emilie Danna, Edward Rothberg, and Claude Le Pape (2005), Exploring relaxation induced neighborhoods to improve MIP solutions, Mathematical Programming

[3] Timo Berthold (2014), RENS: The optimal rounding, Mathematical Programming Computation

in x^{LP}, forming a large neighbourhood

$$x_j = x_j^{\text{LP}},\ \forall j \in \{1, \ldots, n\} : x_j^{\text{LP}} \in \mathbb{Z}.$$

One key difference is how the remaining variables are treated. For those components that are fractional, the following bounds are imposed

$$\lfloor x_j \rfloor \leq x_j \leq \lceil x_j \rceil,\ \forall j \in \{1, \ldots, n\} : x_j^{\text{LP}} \notin \mathbb{Z}.$$

Notice that his in effect makes the neighbourhood considerably smaller around the solution x^{LP}. Then, the MIP subproblem with all these additional constraints is solved and a new incumbent solution may be found.

Local branching

The idea of local branching is to allow the search to be performed in a neighbourhood of controlled size, which is achieved by the use of an L_1-norm[4]. The size of the neighbourhood is controlled by a divergence parameter Δ, which in the case of binary variables, amounts to being the Humming distance between the variable vectors.

In its most simple form, it can be seen as the following idea. From an incumbent solution $\overline{x}$, one can generate and impose the following neighbourhood inducing constraint

$$\sum_{j=1}^{n} |x_j - \overline{x}_j| \leq \Delta$$

and then solve the resulting MIP.

The original use of local branching (as the name suggests) as proposed is to use this constraint directly to form a branching rule in an auxiliary tree search. However, most solvers use it by means of subproblems as described above.

Feasibility pump

Feasibility pump is a constructive heuristic that, contrary to the previous heuristics, has made inroads in most professional-grade solvers and if often employed by default. The focus is exclusively trying to find a first primal feasible solution. The idea consists of, from the LP relaxation solution x^{LP}, performing alternate steps of rounding and solving a projection step, which happens to be a LP problem.

Starting from the x^{LP}, the method starts by simply rounding the components to obtain an integer solution $\overline{x}$. If this rounded solution is feasible, the algorithm terminates. Otherwise, we perform a projection step by replacing the LP relaxation objective function with

$$f^{\text{aux}}(x) = \sum_{j=1}^{n} |x_j - \overline{x}_j|$$

and resolving it. This is called a projection because it is effectively finding a point in the feasible region of the LP relaxation that is the closest to the integer solution $\overline{x}$. This new solution x^{LP} is once again rounded and the process repeats until a feasible integer solution is found.

[4]Matteo Fischetti and Andrea Lodi (2003), Local branching, Mathematical Programming

It is known that feasibility pump can suffer from cycling, that is, repeatedly findingthe same x^{LP} and $\overline{x}$ solutions. This can be alleviated by performing random perturbations on some of the components of $\overline{x}$.

Feasibility pump is an extremely powerful method and plays a central role in many professional-grade solvers. It is also useful in the context of mixed-integer nonlinear programming models. More recently, variants have been developed[5], taking into account the quality of the projection (i.e., taking into account also the original objective function) and discussing theoretical properties of the methods and its convergence guarantees.

[5]Timo Berthold, Andrea Lodi, and Domenico Salvagnin (2018), Ten years of feasibility pump, and counting, EURO Journal on Computational Optimization

11.7 Exercises

Problem 11.1: Preprocessing and primal heuristics

(a) *Tightening bounds and redundant constraints*

Consider the LP below,

$$\begin{array}{lrcrcrcr} \max & 2x_1 & + & x_2 & - & x_3 & & \\ \text{s.t.} & 5x_1 & - & 2x_2 & + & 8x_3 & \leq & 15 \\ & 8x_1 & + & 3x_2 & - & x_3 & \leq & 9 \\ & x_1 & + & x_2 & + & x_3 & \leq & 6 \\ & 0 \leq x_1 \leq 3 & & & & & & \\ & 0 \leq x_2 \leq 1 & & & & & & \\ & x_3 \geq 1 & & & & & & , \end{array}$$

derive tightened bounds for variables x_1 and x_3 from the first constraint and eliminate redundant constraints after that.

(b) *Primal heuristics (RINS)*

Consider the formulation UFL-W:

$$\begin{aligned} (\text{UFL-W}): \quad & \min_{x,y} \sum_{j\in N} f_j y_j + \sum_{i\in M}\sum_{j\in N} c_{ij}x_{ij} && && (11.3)\\ & \text{s.t.:} && && \\ & \sum_{j\in N} x_{ij} = 1, && \forall i \in M, && (11.4)\\ & \sum_{i\in M} x_{ij} \leq m y_j, && \forall j \in N, && (11.5)\\ & x_{ij} \geq 0, && \forall i \in M, \forall j \in N, && (11.6)\\ & y_j \in \{0,1\}, && \forall j \in N,, && (11.7) \end{aligned}$$

where f_j is the cost of opening facility j, and c_{ij} is the cost of satisfying client i's demand from facility j. Consider an instance of the UFL with opening costs $f = (21, 16, 30, 24, 11)$ and client costs

$$(c_{ij}) = \begin{pmatrix} 6 & 9 & 3 & 4 & 12 \\ 1 & 2 & 4 & 9 & 2 \\ 15 & 2 & 6 & 3 & 18 \\ 9 & 23 & 4 & 8 & 1 \\ 7 & 11 & 2 & 5 & 14 \\ 4 & 3 & 10 & 11 & 3 \end{pmatrix}$$

In the UFL problem, the facility production capacities are assumed to be large, and there is no budget constraint on how many facilities can be built. The problem thus has a feasible solution if at least one facility is opened. We choose an initial feasible solution $\bar{y} = (1, 0, 0, 0, 0)$. Try to improve the solution by using relaxation-induced neighbourhood search (RINS) and construct a feasible solution using relaxation-enforced neighbourhood search (RENS).

Problem 11.2: Vehicle routing problem

Consider a centralised depot, from which deliveries are supposed to be made from. The delivers have to be made to a number of clients, and each client has a specific demand to be received. Some assumptions we will consider:

- The deliveries have to be made by vehicles that are of limited capacity;
- Multiple routes can be taken, but only a single vehicle is assigned to a route;
- We assume that the number of vehicles is not a limitation

Our objective is to define optimal routes such that the total distance traveled is minimised. We assume that the total distance is a proxy for the operation cost in this case. The structural elements and parameters that define the problem are described below:

- n is the total number of clients
- N is the **set** of clients, with $N = \{2, \ldots, n + 1\}$
- V is the set of **nodes**, representing a depot (node 1) and the clients (nodes $i \in N$). Thus $V = \{1\} \cup N$
- A is a set of **arcs**, with $A = \{(i, j) \in V \times V : i \neq j\}$
- $C_{i,j}$ - cost of travelling via arc $(i, j) \in A$
- Q - vehicle capacity in units
- D_i - amount that has to be delivered to customer $i \in N$, in units

Formulate the LP to solve the problem and solve it using JuMP, use the instance found in the notebook for session 11. After solving, test different solver parameters and compare the time needed to solve the problem.

Part II

Nonlinear optimisation

Chapter 12

Introduction

12.1 What is optimisation?

An optimisation is one of these words that has many meanings, depending on the context you take as a reference. In the context of this course, optimisation refers to *mathematical optimisation*, which is a discipline of applied mathematics.

In mathematical optimisation, we build upon concepts and techniques from calculus, analysis, linear algebra, and other domains of mathematics to develop methods that allow us finding values for variables within a given domain that maximise (or minimise) the value of a function. In specific, we are trying to solve the following general problem:

$$\begin{aligned} &\min f(x) \\ &\text{s.t.} x \in X. \end{aligned} \tag{12.1}$$

In a general sense, these problems can be solved by employing the following strategy:

1. Analysing properties of functions under specific domains and deriving the conditions that must be satisfied such that a point x is a candidate optimal point.

2. Applying numerical methods that iteratively searches for points satisfying these conditions.

This idea is central in several domains of knowledge, and very often are defined under area-specific nomenclature. Fields such as economics, engineering, statistics, machine learning and, perhaps more broadly, operations research, are intensive users and developers of optimisation theory and applications.

12.1.1 Mathematical programming and optimisation

Operations research and mathematical optimisation are somewhat intertwined, as they both were born around a similar circumstance.

I like to separate *mathematical programming* from (mathematical) *optimisation*. Mathematical programming is a modelling paradigm, in which we rely on (very powerful, I might add) analogies to model *real-world* problems. In that, we look at a given decision problem considering that

- *variables* represent *decisions*, as in a business decision or a course of action. Examples include setting the parameter of (e.g., prediction) model, production systems layouts, geometries of structures, topologies of networks, and so forth;

- *domain* represents business rules or *constraints*, representing logic relations, design or engineering limitations, requirements, and such;
- function is an *objective function* that provides a measure of solution quality.

With these in mind, we can represent the decision problem as a *mathematical programming model* of the form of (12.1) that can be solved using *optimisation* methods. From now on, we will refer to this specific class of models as mathematical optimisation models, or optimisation models for short. We will also use the term to *solve the problem* to refer to the task of finding optimal solutions to optimisation models.

This course is mostly focused on the optimisation techniques employed to find optimal solutions for these models. As we will see, depending on the nature of the functions f and g that are used to formulate the model, some methods might be more or less appropriate. Further complicating the issue, for models of a given nature, there might be alternative algorithms that can be employed and with no generalised consense whether one method is generally better performing than another.

12.1.2 Types of mathematical optimisation models

In general, the simpler the assumptions on the parts forming the optimisation model, the more efficient are the methods to solve such problems.

Let us define some additional notation that we will use from now on. Consider a model in the general form

$$\begin{aligned} \text{min. } & f(x) \\ \text{s.t.: } & g_i(x) \leq 0, i = 1, \dots, m \\ & h_i(x) = 0, i = 1, \dots, l \\ & x \in X, \end{aligned}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is the objective function, $g : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a collection of m inequality constraints and $h : \mathbb{R}^n \mapsto \mathbb{R}^l$ is a collection of l equality constraints.

Remark: in fact, every inequality constraint can be represented by an equality constraint by making $h_i(x) = g_i(x) + x_{n+1}$ and augmenting the decision variable vector $x \in \mathbb{R}^n$ to include the slack variable x_{n+1}. However, since these constraints are of very different nature, we will explicitly represent both whenever necessary.

The most general types of models are the following. We also use this as an opportunity to define some (admittedly confusing) nomenclature from the field of operations research that we will be using in these notes.

1. *Unconstrained models:* in these, the set $X = \mathbb{R}^n$ and $m = l = 0$. These are prominent in, e.g., machine learning and statistics applications, where f represents a measure of model fitness or prediction error.
2. *Linear programming (LP):* presumes linear objective function. $f(x) = c^\top x$ and constraints g and h affine, i.e., of the form $a_i^\top x - b_i$, with $a_i \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Normally, $X = \{x \in \mathbb{R}^n \mid x_j \geq 0, j = 1, \dots, n\}$ enforce that decision variables are constrained to be the non-negative orthant.
3. *Nonlinear programming (NLP):* some or all of the functions f, g, and h are nonlinear.

4. *Mixed-integer (linear) programming (MIP):* consists of an LP in which some (or all) of the variables are constrained to be integers. In other words, $X \subseteq \mathbb{R}^k \times \mathbb{Z}^{n-k}$. Very frequently, the integer variables are binary terms, i.e., $x_i \in \{0, 1\}$, for $i = 1, \ldots, n-k$ and are meant to represent true-or-false or yes-or-no conditions.

5. *Mixed-integer nonlinear programming (MINLP):* are the intersection of MIPs and NLPs.

Remark: notice that we use the vector notation $c^\top x = \sum_{j \in J} c_j x_j$, with $J = \{1, \ldots, N\}$. This is just a convenience for keeping the notation compact.

12.2 Examples of applications

We now discuss a few examples to illustrate the nature of the problems to which we will develop solution methods and their applicability to real-world contexts.

12.2.1 Resource allocation and portfolio optimisation

In a general sense, any mathematical optimisation model is an instantiation of the *resource allocation problem*. A resource allocation problem consists of how to design an optimal allocation of resources to tasks, such that a given outcome is optimised.

Classical examples typically include production planning settings, in which raw materials or labour resources are inputted into a system and a collection of products, a production plan, results from this allocation. The objective is to find the best production plan, that is, a plan with the maximum profit or minimum cost. Resource allocation problems can also appear in a less obvious setting, where the resources can be the capacity of transmission lines in an energy generation planning setting, for example.

Let $i \in I = \{1, \ldots, M\}$ be a collection of resources and $j \in J = \{1, \ldots, N\}$ be a collection of products. Suppose that, to produce one unit of product j, a quantity a_{ij} of resource i is required. Assume that the total availability of resource i is b_i and that the return per unit of product j is c_j.

Let x_j be the decision variable representing total of product j produced. The resource allocation problem can be modelled as

$$\text{max.} \quad \sum_{j \in J} c_j x_j \tag{12.2}$$

$$\text{s.t.:} \quad \sum_{j \in J} a_{ij} x_j \leq b_i, \ \forall i \in I \tag{12.3}$$

$$x_j \geq 0, \ \forall j \in J. \tag{12.4}$$

Equation (12.2) represents the objective function, in which we maximise the total return obtained from a given production plan. Equation (12.3) quantify the resource requirements for a given production plan and enforce that such a requirement does not exceed the resource availability. Finally, constraint (12.4) defines the domain of the decision variables.

Notice that, as posed, the resource allocation problem is linear. This is perhaps the most basic, and also most diffused setting for optimisation models for which very reliable and mature technology is available. In this course, we will concentrate on methods that can solve variants of this model in which the objective function and/or the constraints are required to include nonlinear terms.

One classic variant of resource allocation that include nonlinear terms is the *portfolio optimisation problem*. In this, we assume that a collection of assets $j \in J = \{1, \ldots, N\}$ are available for investment. In this case, capital is the single (actual) resource to be considered. Each asset has random return R_j, with an expected value $\mathbb{E}[R_j] = \mu_j$. Also, the covariance between two assets $i, j \in J$ is given by $\sigma_{ij} = \mathbb{E}[(R_i - \mu_i)(R_j - \mu_j)]$, which can be denoted as the covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_{11} & \ldots & \sigma_{1N} \\ \vdots & \ddots & \vdots \\ \sigma_{N1} & \ldots & \sigma_{NN} \end{bmatrix}$$

Markowitz (1952) proposed using $x^\top \Sigma x$ as a risk measure that captures the variability in the return of the assets. Given the above, the optimisation model that provides the investment portfolio with the least risk, given a minimum requirement ϵ in terms of expected returns is given by

$$\text{min.} \quad x^\top \Sigma x \tag{12.5}$$

$$\text{s.t.:} \ \mu^\top x \geq \epsilon \tag{12.6}$$

$$0 \leq x_j \leq 1, \ \forall j \in J. \tag{12.7}$$

Objective function (12.5) represents the portfolio risk to be minimised, while constraint (12.6) enforces that the expected return must be at least ϵ. Notice that ϵ can be seen as a resource that has to be (at least) completely depleted, if one wants to do a parallel with the resource allocation structure discussed early. Constraint (12.7) defined the domain of the decision variables. Notice how the problem is posed in a scaled form, where $x_j \in [0, 1]$ represents a percentage of a hypothetical available capital for investment.

In this example, the problem is nonlinear due to the quadratic nature of the objective function $x^\top \Sigma x = \sum_{i,j \in J} \sigma_{ij} x_i x_j$. As we will see later on, there are efficient methods that can be employed to solve quadratic problems like this.

12.2.2 The pooling problem: refinery operations planning

The *pooling problem* is another example of a resource allocation problem that naturally presents nonlinear constraints. In this case, the production depends on *mixing operations*, known as pooling, to obtain certain product specification for a given property.

As an illustration, suppose that products $j \in J = \{1, \ldots, N\}$ are produced by mixing byproducts $i \in I_j \subseteq I = \{1, \ldots, M\}$. Assume that the qualities of byproducts q_i are known and that there is no reaction between byproducts. Each product is required to have a property value q_j within an acceptable range $[\underline{q}_j, \overline{q}_j]$ to be classified as product j. In this case, mass and property balances are calculated as

$$x_j = \sum_{i \in I_j} x_i, \ \forall j \in J \tag{12.8}$$

$$q_j = \frac{\sum_{i \in I_j} q_i x_i}{x_j}, \ \forall j \in J. \tag{12.9}$$

These can then incorporated into the resource allocation problem accordingly. One key aspect associated with pooling problem formulations is that the property balances represented by (12.9) define *nonconvex* feasibility regions. As we will see later, convexity is a powerful property that allows for developing efficient solution methods and its absence typically compromises computational performance and tractability in general.

12.2.3 Robust optimisation

Robust optimisation is a subarea of mathematical programming concerned with models that support decision-making under *uncertainty*. In specific, the idea is to devise a formulation mechanism that can guarantee feasibility of the optimal solution in face of variability, ultimately taking a risk-averse standpoint.

Consider the resource allocation problem from Section 12.2.1. Now, suppose that the parameters $\tilde{a}_i \in \mathbb{R}^N$ associated with a given constraint $i \in I = \{1, \ldots, M\}$ are uncertain with a unknown probability distribution. The resource allocation problem can then be formulated as

$$\begin{aligned} \max. \quad & c^\top x \\ \text{s.t.:} \quad & \tilde{a}_i^\top x \leq b_i, \ \forall i \in I \\ & x_j \geq 0, \ \forall j \in J. \end{aligned}$$

Let us assume that the only information available are observations $\hat{a}_i$, from which we can estimate a nominal value $\bar{a}_i$. This is illustrated in Figure 12.1, in which 100 random observations are generated for $\tilde{a}_i = [\tilde{a}_{i1}, \tilde{a}_{i2}]$ with $\tilde{a}_{i1} \sim \text{Normal}(10, 2)$ and $\tilde{a}_{i2} \sim \text{Normal}(5, 3)$ for a single constraint $i \in I$. The nominal values are assumed to have coordinates given by the average values used in the Normal distributions. Our objective is to develop a model that incorporates a given level of protection in

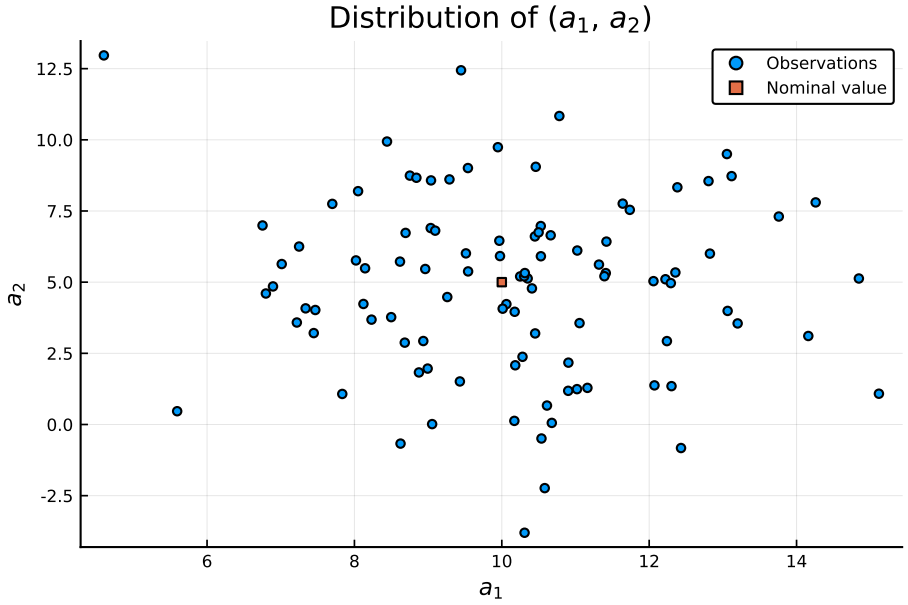


Figure 12.1: One hundred random realisations for $\tilde{a}_i$.

terms of feasibility guarantees. That is, we would like to develop a model that provides solutions that are guaranteed to remain feasible if the realisation of $\tilde{a}_i$ falls within an *uncertainty set* ϵ_i of size controlled by the parameter Γ_i. The idea is that the bigger the uncertainty set ϵ_i, the more robust is the solution, which typically comes at the expense of accepting solutions with expected worse performance.

The tractability of robust optimisation models depends on the geometry of the uncertainty set

employed. Let us assume in what follows that

$$\epsilon_i = \{\overline{a}_i + P_i u \mid ||u||_2 \leq \Gamma_i\} \tag{12.10}$$

is an ellipsoid with the characteristic matrix P_i (i.e., its eigenvalues show how the ellipsoid extends in every direction from $\overline{a}_i$) and Γ_i employs a scaling of the ellipsoid size.

Remark: an alternative (perhaps more frequent) characterisation of an ellipsoid $\epsilon \subset \mathbb{R}^n$ centred at $\overline{x}$ is given by $\epsilon = \left\{x \in \mathbb{R}^n \mid (x - \overline{x})^\top A(x - \overline{x}) = 1\right\}$. By making $A = P^{-2}$, we recover the representation in (12.10).

We can now formulate the *robust counterpart*, which consists of a risk-averse version of the original resource allocation problem. In that, we try to anticipate the worst possible outcome and make decisions that are both optimal and guarantee feasibility in this worst-case sense. This standpoint translates into the following optimisation model.

$$\begin{aligned} \max.\ & c^\top x \\ \text{s.t.:}\ & \max_{a_i \in \epsilon_i}\ \left\{a_i^\top x\right\} \leq b_i,\ \forall i \in I \\ & x_j \geq 0, \forall j \in J. \end{aligned} \tag{12.11}$$

Notice how the constraint (12.11) has an embedded optimisation problem, turning into a *bi-level optimisation* problem. This highlights the issue associated with tractability, since solving the whole problem strongly depends on deriving tractable equivalent reformulations.

Assuming that the uncertainty set ϵ_i is an ellipsoid, the following result holds.

$$\max_{a_i \in \epsilon_i} \left\{a_i^\top x\right\} = \overline{a}_i^\top x + \max_u \left\{u^\top P_i x : ||u||_2 \leq \Gamma_i\right\} \tag{12.12}$$

$$= \overline{a}_i^\top x + \Gamma_i ||P_i x||_2. \tag{12.13}$$

In (12.12), we recast the inner problem in terms of the ellipsoidal uncertainty set, ultimately meaning that we recast the inner maximisation problem in terms of variable u. Since the only constraint is $||u||_2 \leq \Gamma_i$, in (12.13) we can derive a closed form for the inner optimisation problem.

With the closed form derived in (12.13), we can reformulate the original bi-level problem as a tractable single-level problem of the following form

$$\begin{aligned} \max.\ & c^\top x \\ \text{s.t.:}\ & \overline{a}_i^\top x + \Gamma_i ||P_i x||_2 \leq b_i,\ \forall i \in I \\ & x_j \geq 0,\ \forall j \in J. \end{aligned} \tag{12.14}$$

Notice how the term $\Gamma_i ||P_i^\top x||_2$ creates a buffer for constraint (12.14), ultimately preventing the complete depletion of the resource. Clearly, this will lead to a suboptimal solution when compared to the original deterministic at the expense of providing protection against deviations in coefficients a_i. This difference is often referred to as the *price of robustness*.

In Figure 12.2, we show the ellipsoidal sets for two levels of Γ_i for a single constraint i. We define

$$\epsilon_i = \left\{ \begin{bmatrix} 10 \\ 5 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \right\} \tag{12.15}$$

using the average and standard deviation of the original distributions that generated the observations. We plot the ellipsoids for $\Gamma_1 = 1$ and $\Gamma_2 = 1.5$, illustrating how the protection level increases as Γ increases. This can be inferred since the uncertainty set covers more of the observations and the formulation is such that feasibility is guaranteed for any observation within the uncertainty set.

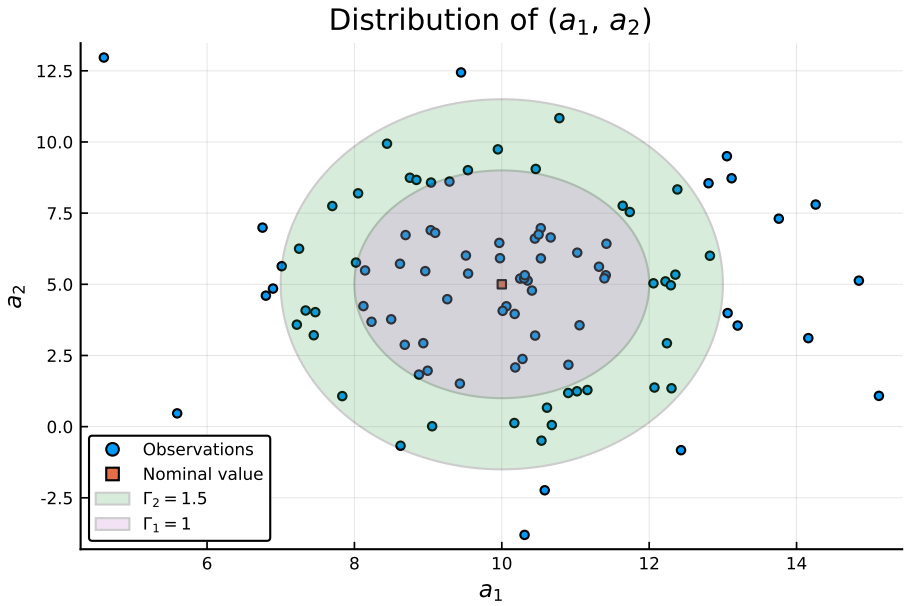


Figure 12.2: One hundred random realisations for $\tilde{a}_i$.

12.2.4 Classification: support-vector machines

This is an example in which the resource allocation structure within the optimisation model is not as obvious. Suppose we are given a data set $D \in \mathbb{R}^n$ with $|D| = N + M$ that can be divided into two disjunct sets $I^- = \{x_1, \ldots, x_N\}$ and $I^+ = \{x_1, \ldots, x_M\}$.

Each element in D is an observation of a given set of n features with values represented by a $x \in \mathbb{R}^n$ that has been classified as belonging to set I^- and I^+. Because of the availability of labelled data, classification is said to be ane xample of supervised learning in the field of machine learning.

Figure 12.3 illustrates this situation for $n = 2$, in which the orange dots represent points classified as belonging to I^- (negative observations) and the blue dots represent points classified as belonging to I^+ (positive observations).

Our task is to obtain a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ from a given family of functions that is capable to, given an observed set of features $\hat{x}$, classify whether it belongs to I^- or I^+. In other words, we want to calibrate f such that

$$f(x_i) < 0, \ \forall x_i \in I^-, \text{ and } f(x_i) > 0, \ \forall x_i \in I^+. \tag{12.16}$$

This function would then act as a classifier that could be employed to any new observation $\hat{x}$ made. If f is presumed to be an affine function of the form $f(x) = a^\top x - b$, then we obtain a *linear classifier*.

Our objective is to obtain $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that misclassification error is minimised. Let us

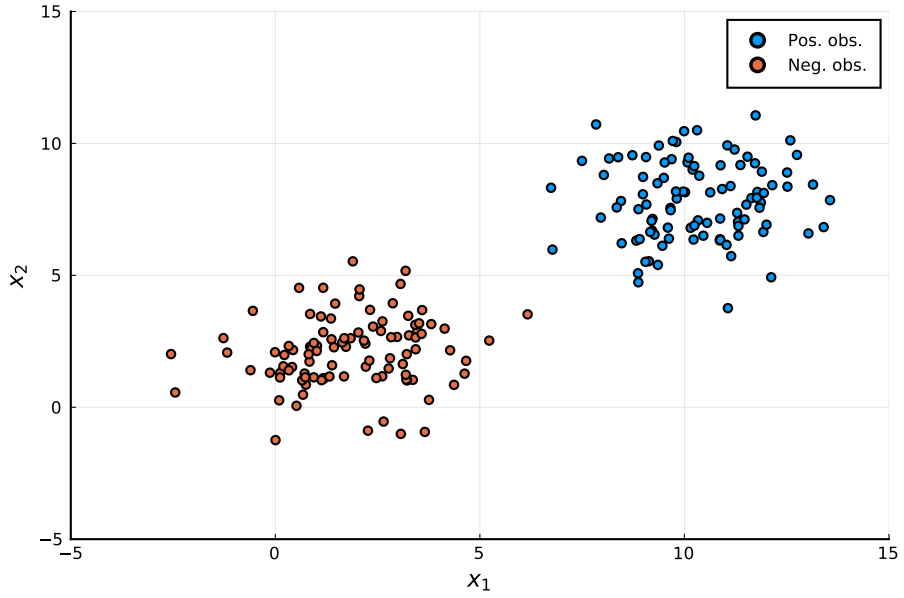


Figure 12.3: Two hundred observations for x_i classified to belong to I^- (orange) or I^+ (blue).

define the error measure as

$$e^-(x_i \in I^-; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \leq 0, \\ a^\top x_i - b, \text{ if } a^\top x_i - b > 0. \end{cases}$$

$$e^+(x_i \in I^+; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \geq 0, \\ b - a^\top x_i, \text{ if } a^\top x_i - b < 0. \end{cases}$$

Using this error measure, we can define constraints that capture deviation on each measure by means of nonnegative slack variables. Let $u_i \geq 0$ for $i = 1, \ldots, N$ and $v_i \geq 0$ for $i = 1, \ldots, M$ be slack variables that measure the *misclassification error* for $x_i \in I^-$ and $x_i \in I^+$, respectively.

The optimisation problem that finds optimal parameters a and b can be stated as

$$\text{min.} \quad \sum_{i=1}^{M} u_i + \sum_{i=1}^{N} v_i \tag{12.17}$$

$$\text{s.t.:} \; a^\top x_i - b - u_i \leq 0, i = 1, \ldots, M \tag{12.18}$$

$$a^\top x_i - b + v_i \geq 0, i = 1, \ldots, N \tag{12.19}$$

$$||a||_2 = 1 \tag{12.20}$$

$$u_i \geq 0, i = 1, \ldots, N \tag{12.21}$$

$$v_i \geq 0, i = 1, \ldots, M \tag{12.22}$$

$$a \in \mathbb{R}^n, b \in \mathbb{R}. \tag{12.23}$$

The objective function (12.17) accumulates the total misclassification error. Constraint (12.18) allows for capturing the misclassification error for each $x_i \in I^-$. Notice that $u_i = \max\left\{0, a^\top x_i - b\right\} = e^-(x_i \in I^-; a, b)$. Likewise, constraint (12.19) guarantees that $v_i = e^+(x_i \in I^+; a, b)$. To avoid

trivial solutions in which $(a, b) = (0, 0)$, the normalisation constraint $||a||_2 = 1$ is imposed in constraint (12.20), which turns the model nonlinear.

Solving the model (12.17)–(12.23) provides optimal (a, b) which translates into the classifier represented as the green line in Figure 12.4.

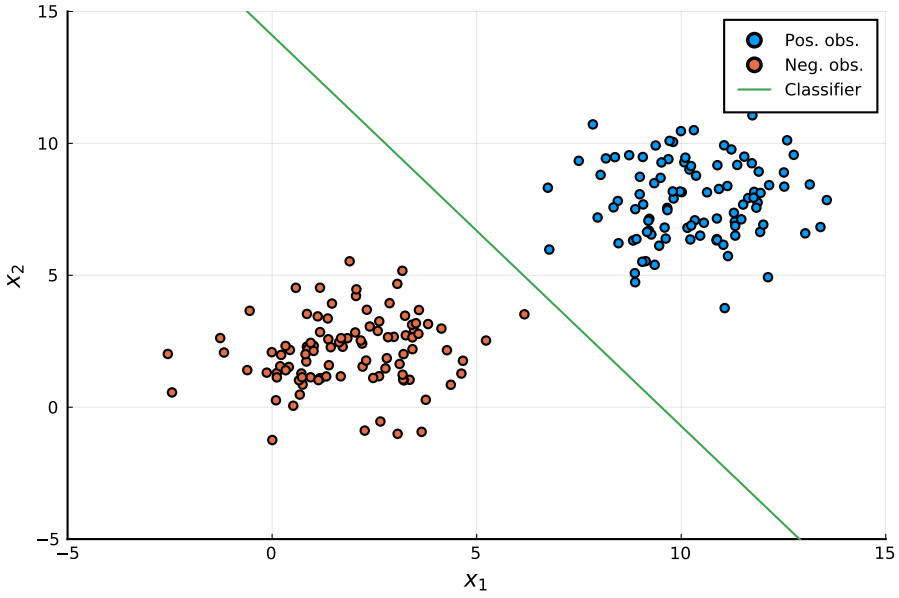


Figure 12.4: Two hundred observations for x_i classified to belong to I^- (orange) or I^+ (blue) with a classifier (green).

A variant referred to as *robust classifier* penalises not only the the misclassification error, but also the observations within a given slab $S = \left\{x \in \mathbb{R}^n \mid -1 \leq a^\top x - b \leq 1\right\}$. Notice that, being the two lines defined by $f^-(x) : a^\top x - b = -1$ and $f^+(x) : a^\top x - b = +1$, the distance between the two hyperplanes is given by $\frac{2}{||a||_2}$.

Accordingly, we redefine our error measures as follows.

$$e^-(x_i \in I^-; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \leq -1, \\ |a^\top x_i - b|, \text{ if } a^\top x_i - b > -1. \end{cases}$$

$$e^+(x_i \in I^+; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \geq 1, \\ |b - a^\top x_i|, \text{ if } a^\top x_i - b < 1. \end{cases}$$

By doing so, a penalty is applied not only to those points that were misclassified but also to those points correctly classified that happen to be inside the slab S. To define an optimal robust classifier, one must trade off the size of the slab, which is inversely proportional to $||a||$, and the

total of observations that fall in the slab S. The formulation for the robust classifier then becomes

$$\text{min.} \quad \sum_{i=1}^{M} u_i + \sum_{i=1}^{N} v_i + \gamma ||a||_2^2 \tag{12.24}$$

$$\text{s.t.:} \; a^\top x_i - b - u_i \leq -1, \; i = 1, \ldots, M \tag{12.25}$$

$$a^\top x_i - b + v_i \geq 1, \; i = 1, \ldots, N \tag{12.26}$$

$$u_i \geq 0, i = 1, \ldots, N \tag{12.27}$$

$$v_i \geq 0, i = 1, \ldots, M \tag{12.28}$$

$$a \in \mathbb{R}^n, b \in \mathbb{R}. \tag{12.29}$$

In objective function (12.24), the errors accumulated in variables u_i, $i = 1, \ldots, N$ and v_i, $i = 1, \ldots, M$ and the squared norm $||a||_2^2$ are considered simultaneously. The term γ is a scalar used to impose an emphasis on minimising the norm $||a||_2$ and incentivising a larger slab S (recall that the slab is large for smaller $||a||_2$). The squared norm $||a||_2^2$ is considered instead as a means to recover differentiability, as the norm $||a||_2$ is not differentiable. Later on, we will see how beneficial it is for optimisation methods to be able to assume differentiability. Moreover, note how in constraints (12.25) and (12.26) u and v also accumulate penalties for correctly classified x_i that happen to be between the slab S, that is, that have term $a^\top x - b$ larger/ smaller than -1/ +1. Figure 12.5 shows a robust classifier an arbitrary value of γ.

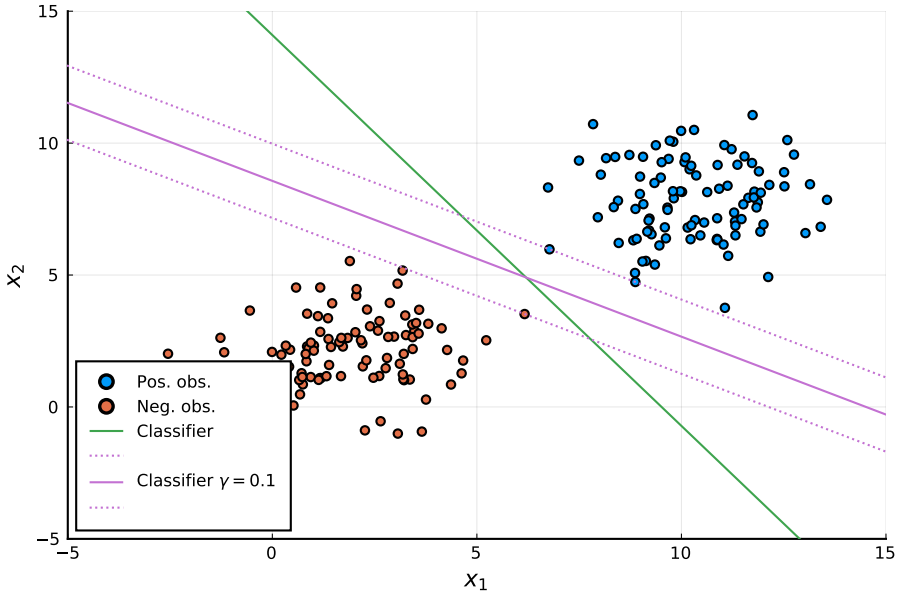


Figure 12.5: Two hundred observations for x_i classified to belong to I^- (orange) or I^+ (blue).

Remark: robust classifiers are known in the machine learning literature as *support vector machines*, where the support vectors are the observations that support the slab.

Chapter 13

Convex sets

13.1 Convexity and optimisation

Convexity is perhaps the most important property that the elements forming an optimisation problem can present. Paraphrasing Tyrrell Rockafellar:

> ... in fact, the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity.

The importance of convexity will become clear later in the course. In a nutshell, the existence of convexity allows us to infer global properties of a solution (i.e., that holds for all of its domain) by considering exclusively local information (such as gradients, for example). This is critical in the context of optimisation, since most of the methods we know to perform well in practice are designed to find solutions that satisfy local optimality conditions. Once convexity is attested, one can then guarantee that these local solutions are in fact globally optimal without exhaustively exploring the solution space.

For a problem of the form

$$\begin{aligned}(P): \quad & \text{min. } f(x) \\ & \text{s.t.: } x \in X\end{aligned}$$

to be convex, we need to verify whether f is a *convex function* and X is a *convex set*. If both statements hold true, we can conclude that P is a *convex problem*. We start looking into how to identify convex sets, since we can use the convexity of sets to infer the convexity of functions.

13.2 Identifying convexity of sets

Before we formally define convex sets, let us first look at the idea of *combinations*. For that, let $S \subseteq \mathbb{R}^n$ be a set and $x_j \in S$ for $j = 1, \ldots, k$ be a collection of vectors (i.e., n-dimensional "points") belonging to S. Then, we have that:

- A *linear combination* of x_j for $j = 1, \ldots, k$ is the set

$$\left\{ x \in \mathbb{R}^n : \sum_{j=1}^{k} \lambda_j x_j, \ \lambda_j \in \mathbb{R} \text{ for } j = 1, \ldots, k \right\}. \tag{13.1}$$

- An *affine combination* is a linear combination, with the additional constraint that $\sum_{j=1}^{k} \lambda_j = 1$. That is,

$$\left\{ x \in \mathbb{R}^n : \sum_{j=1}^{k} \lambda_j x_j,\ \sum_{j=1}^{k} \lambda_j = 1,\ \lambda_j \in \mathbb{R} \text{ for } j = 1, \ldots, k \right\}. \tag{13.2}$$

- A *conic combination* is a linear combination with the additional condition that $\lambda_j \geq 0$ for $j = 1, \ldots, k$.

$$\left\{ x \in \mathbb{R}^n : \sum_{j=1}^{k} \lambda_j x_j,\ \lambda_j \geq 0 \text{ for } j = 1, \ldots, k \right\}. \tag{13.3}$$

- And finally, a *convex combination* is the intersection between an affine and a conic combinations, implying that $\lambda_j \in [0, 1]$.

$$\left\{ x \in \mathbb{R}^n : \sum_{j=1}^{k} \lambda_j x_j,\ \sum_{j=1}^{k} \lambda_j = 1,\ \lambda_j \geq 0 \text{ for } j = 1, \ldots, k \right\}. \tag{13.4}$$

We say that a set is convex if it contains all points formed by the convex combination of any pair of points in this set. This is equivalent to saying that the set contains the line segment between any two points belonging to the set.

Definition 13.1 (Convex sets)**.** *A set $S \subseteq \mathbb{R}^n$ is said to be convex if $\overline{x} = \sum_{j=1}^{k} \lambda_j x_j$ belongs to S, where $\sum_{j=1}^{k} \lambda_j = 1$, $\lambda_j \geq 0$ and $x_j \in S$ for $j = 1, \ldots, k$.*

Definition 13.1 is useful as it allows for showing that some set operations preserve convexity.

13.2.1 Convexity-preserving set operations

Lemma 13.2 (Convexity-preserving operations)**.** *Let S_1 and S_2 be convex sets in $\mathbb{R}^n$. Then, the sets resulting from the following operations are also convex.*

1. *Intersection: $S = S_1 \cap S_2$;*
2. *Minkowski addition: $S = S_1 + S_2 = \{x_1 + x_2 : x_1 \in S_1, x_2 \in S_2\}$;*
3. *Minkowski difference: $S = S_1 - S_2 = \{x_1 - x_2 : x_1 \in S_1, x_2 \in S_2\}$;*
4. *Affine transformation: $S = \{Ax + b : x \in S_1\}$.*

Figures 13.1 and 13.2 illustrate the concept behind some of these set operations. Showing that the sets resulting from the operations in Lemma 13.2 are convex typically entails showing that convex combinations of elements in the resulting set S also belong to S_1 and S_2.

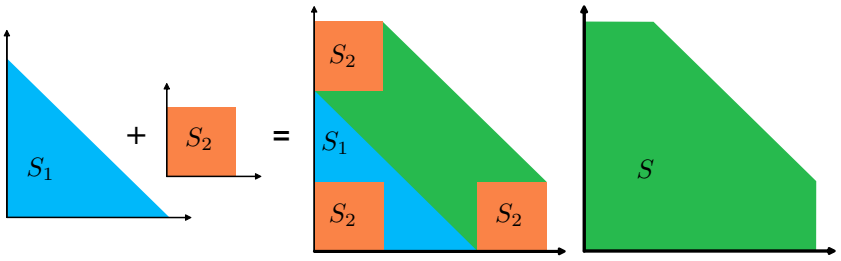


Figure 13.1: Minkowski sum of two convex sets.

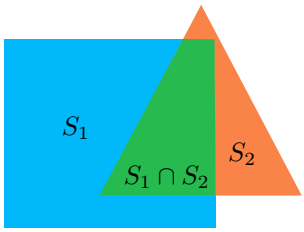


Figure 13.2: Intersection of two convex sets.

13.2.2 Examples of convex sets

There are several familiar sets that are known to be convex. Having the knowledge that these sets are convex is useful as a building block for determining the convexity of more complicated sets.

Some important examples of convex sets include:

- Them empty set $\emptyset$, any singleton $\{\overline{x}\}$ and the whole space $\mathbb{R}^n$;
- halfspaces: $S = \left\{x : p^\top x \leq \alpha\right\} \subset \mathbb{R}^n$;
- hyperplanes: $H = \left\{x : p^\top x = \alpha\right\} \subset \mathbb{R}^n$, where $p \neq 0^n$ is a normal vector and $\alpha \in \mathbb{R}$ is a scalar. Notice that H can be equivalently represented as $H = \left\{x \in \mathbb{R}^n : p^\top (x - \overline{x}) = 0\right\}$ for $\overline{x} \in H$;
- polyhedral sets: $P = \{x : Ax \leq b\} \subset \mathbb{R}^n$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$;
- norm-induced sets (balls): $B = \{x : ||x - \overline{x}|| \leq \alpha\} \subseteq \mathbb{R}^n$, where $||\cdot||$ is any norm and α a scalar;
- norm cones: $C = \left\{(x, \alpha) \in \mathbb{R}^{n+1} : ||x|| \leq \alpha\right\}$;

For example, let us consider the polyhedral set $P = \{x \in \mathbb{R}^n : Ax \leq b\} \subset \mathbb{R}^n$ with A being a $m \times n$ matrix. Notice that S is the intersection of a collection of half-spaces $H_i = \left\{x \in \mathbb{R}^n : a_i^\top x \leq b_i\right\}$, where a_i are vectors from the rows of the matrix A and b_i are the components of the column vector b. We know that H_i are convex sets, thus $P = \cap_{i=1}^m H_i$ is also convex, as the intersection of sets is a convexity-preserving set operation.

Hyperplanes and halfspaces

Hyperplanes and halfspaces will play a central role in the developments we will see in our course. Therefore, let us take a moment and discuss some important aspects related these convex sets. First, notice that, geometrically, a hyperplane $H \subset \mathbb{R}^n$ can be interpreted as the set of points with a *constant* inner product to a given vector $p \in \mathbb{R}^n$, while $\overline{x}$ determines the offset of the hyperplane from the origin. That is,

$$H = \left\{x : p^\top (x - \overline{x}) = 0\right\} \equiv \overline{x} + p^\perp,$$

where $p^\perp$ is the orthogonal complement of p, i.e., the set of vectors orthogonal to p, which is given by $\left\{x \in \mathbb{R}^n : p^\top x = 0\right\}$.

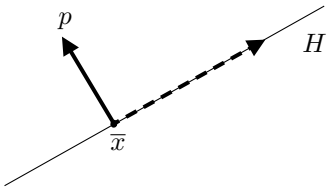


Figure 13.3: A hyperplane $H = \left\{x \in \mathbb{R}^n : p^\top (x - \overline{x}) = 0\right\}$ with normal vector p displaced to $\overline{x}$.

Analogously, a halfspaces can be represented as $S = \left\{x \in \mathbb{R}^n : p^\top (x - \overline{x}) \leq 0\right\}$ where $p^\top \overline{x} = \alpha$ is the hyperplane that forms the boundary of the halfspace. This definition suggests a simple geometrical interpretation: the halfspace S consists of $\overline{x}$ plus any vector with an obtuse or right angle (i.e., greater of equal to 90°) with the outward normal vector p.

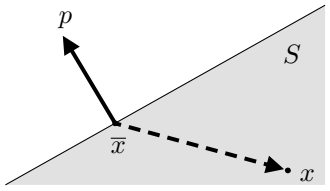


Figure 13.4: A halfspace $S = \left\{x \in \mathbb{R}^n : p^\top (x - \overline{x}) \leq 0\right\}$ defined by the same hyperplane H. Notice how the vectors p (or $p - \overline{x}$, which is fundamentally the same vector but translated to $\overline{x}$) and $x - \overline{x}$ form angles greater or equal than 90°.

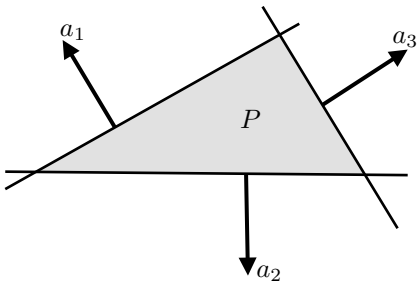


Figure 13.5: A polyhedron P formed by the intersection of three halfspace. Each hyperplane $H_i = \left\{x \in \mathbb{R}^n : a_i^\top x \leq b_i\right\}$, for $i = 1, 2, 3$, has a normal vector a_i, and has an offset from the origin b_i (which cannot be seen once project on a 2-dimensional plane as in the picture).

Norm balls and norm cones

An Euclidean ball (or simply ball) of radius ϵ in $\mathbb{R}^n$ has the form

$$B(\overline{x}, r) = \{x \in \mathbb{R}^n : ||x - \overline{x}||_2 \leq \epsilon\} \equiv \left\{x \in \mathbb{R}^n : (x - \overline{x})^\top (x - \overline{x}) \leq \epsilon^2\right\}$$

As one might suspect, balls are convex, which can be proved by noting that

$$\begin{aligned} ||\lambda x_1 + (1 - \lambda)x_2 - \overline{x}||_2 &= ||\lambda(x_1 - \overline{x}) + (1 - \lambda)(x_2 - \overline{x})||_2 \\ &\leq \lambda||x_1 - \overline{x}||_2 + +(1 - \lambda)||x_2 - \overline{x}||_2 \leq \epsilon. \end{aligned}$$

Notice that between the first and the second line, we use the triangle inequality, which states that $||x + y|| \leq ||x|| + ||y||$ for any two vectors x and y and any norm (including the Euclidean norm).

Euclidean balls are a special case of norm balls, which are defined as $B(\overline{x}, r) = \{x \in \mathbb{R}^n : ||x - \overline{x}|| \leq \epsilon\}$ where $||\ \cdot\ ||$ is any norm on $\mathbb{R}^n$.

A related set is the norm cone, defined as $C(x, \alpha) = \left\{(x, \alpha) \in \mathbb{R}^{n+1} : ||x|| \leq \alpha\right\}$, where α is a scalar. For example, the second-order cone (also known as the ice cream cone or Lorentz cone) is the norm cone for the Euclidean norm.

Remark. *Norm induced sets (balls or cones) are convex for any norm* $||x||_p = (\sum_{i=1}^n x_i^p)^{\frac{1}{p}}$ *for* $x \in \mathbb{R}^n$ *and* $p \geq 1$.

13.3 Convex hulls

A *convex hull* of a set S, denoted $\mathbf{conv}(S)$ is the set formed by all convex combinations of all points in S. As the name suggests, $\mathbf{conv}(S)$ is a convex set, regardless of S being or not convex.

Another interpretation for $\mathbf{conv}(S)$ is to think of it as the tightest enveloping (convex) set that contains S. Notice that, if S is convex, then $S = \mathbf{conv}(S)$. Formally, convex hulls are defined as follows.

Definition 13.3 (Convex hull of a set)**.** *Let* $S \subseteq \mathbb{R}^n$ *be an arbitrary set. The convex hull of* S*, denoted by* $\mathbf{conv}(S)$*, is the collection of all convex combinations of* S*. That is, for* $x_j \in S$*, with* $j = 1, \ldots, k$*,* $x \in \mathbf{conv}(S)$ *if and only if*

$$x = \sum_{j=1}^k \lambda_j x_j : \sum_{j=1}^k \lambda_j = 1,\ \lambda_j \geq 0,\ \textit{for } j = 1, \ldots, k.$$

From Definition 13.3, one can show that the convex hull $\mathbf{conv}(S)$ can also be defined as the intersection of all convex sets containing S. Perhaps the easiest way to visualise this is to think of the infinitely many half-space containing S and their intersection, which can only be S. Figure 13.6 illustrates the convex hull $\mathbf{conv}(S)$ of an nonconvex set S.

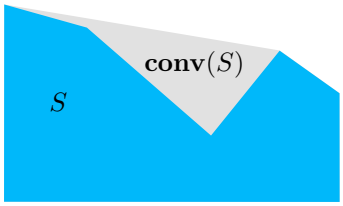


Figure 13.6: Example of an arbitrary set S (in solid blue) and its convex hull $\mathbf{conv}(S)$ (combined blue and grey areas).

The notion of convex hulls is a powerful tool in optimisation. One important application is using $\mathbf{conv}(S)$ to obtain approximations for a nonconvex S that can be exploited to solve an optimisation problem with constraint set defined by S. This is the underpinning technique in many important optimisation methods for such as branch-and-bound-based methods for nonconvex problems and decomposition methods (i.e., methods that solve large problems by breaking it into smaller parts that are presumably easier to solve).

In specific, let us consider the convex hull of a finite collection of discrete points. Some of these sets are so important in optimisation that they have their own names.

Definition 13.4. *Let $S = \{x_1, \ldots, x_{n+1}\} \subset \mathbb{R}^n$. Then $\mathbf{conv}(S)$ is called a* polytope. *If $x_1, \ldots, x_{n+1}$ are affinely independent (i.e., $x_2 - x_1, \ldots, x_{n+1} - x_1$ are linearly independent) then $\mathbf{conv}(S)$ is called a* simplex *with vertices $x_1, \ldots, x_{n+1}$.*

13.4 Closure and interior of sets

Many of the set-related results we will see in this course depends on the characteristics of the set itself. Often, assuming properties such as closedness or compactness considerably ease technical derivations.

13.4.1 Closure, interior and boundary of a set

Let us define some properties that will be useful in this course. For that, we will use an ϵ-neighbourhood of $x \in \mathbb{R}^n$ (which is a norm ball of radius ϵ centred in x) defined as

$$N_\epsilon(x) = \{y : ||y - x|| < \epsilon\}.$$

Let $S \subseteq \mathbb{R}^n$ be an arbitrary set. We can use N_ϵ to define the following elements related to S.

1. *Closure of S*: the set defined by the closure of S, denoted $\mathbf{clo}(S)$, is defined as

$$\mathbf{clo}(S) = \{x \in \mathbb{R}^n : S \cap N_\epsilon(x) \neq \emptyset \text{ for every } \epsilon > 0\}.$$

   Notice that the closure might contain points that do not belong to S. We say that a set is *closed* if $S = \mathbf{clo}(S)$, that is, the set itself is its own closure.

2. *Interior of S*: the interior of S, denoted $\mathbf{int}(S)$, is the set

$$\mathbf{int}\, S = \{x \in S : N_\epsilon(x) \subset S \text{ for some } \epsilon > 0\}.$$

   If S is the same as its own interior, then we say that S is *open*. Some authors say that S is solid if it has a nonempty interior (that is, $\mathbf{int}(S) \neq \emptyset$. Notice that the interior of S is a subset of S, that is $\mathbf{int}(S) \subseteq S$.

3. *Boundary of S*: the boundary of S, denoted $\mathbf{bou}(S)$ is the collection of points defined by

$$\mathbf{bou}(S) = \{x \in \mathbb{R}^n : N_\epsilon(x) \text{ contains some } x_i \in S \text{ and some } x_j \notin S \text{ for every } \epsilon > 0\}.$$

   We say that S is bounded if exists $N_\epsilon(x)$, $x \in \mathbb{R}^n$, for some $\epsilon > 0$ such that $S \subset N_\epsilon(x)$.

We say that a set is *compact* if it is both *closed* and *bounded*. Compact sets appear very frequently in real-world applications of optimisation, since typically one can assume the existence of bounds for decision variables (such as nonnegativity or maximum physical bounds or, at an extreme case, smallest/ largest computational constants). Another frequent example of bounded set is the convex hull of a collection of discrete points, which is called by some authors *polytopes* (effectively bounded polyhedral sets).

Let us consider the following example. Let $S = \left\{(x_1, x_2) \in \mathbb{R}^n : x_1^2 + x_2^2 \leq 1\right\}$. Then, we have that:

1. $\mathbf{clo}(S) = \left\{(x_1, x_2) \in \mathbb{R}^n : x_1^2 + x_2^2 \leq 1\right\}$. Since $S = \mathbf{clo}(S)$, S is closed.
2. $\mathbf{int}(S) = \left\{(x_1, x_2) \in \mathbb{R}^n : x_1^2 + x_2^2 < 1\right\}$.
3. $\mathbf{bou}(S) = \left\{(x_1, x_2) \in \mathbb{R}^n : x_1^2 + x_2^2 = 1\right\}$. Notice that it makes S bounded.
4. S is compact, since it is closed and bounded.

Notice that, if S is closed, then $\mathbf{bou}(S) \subset S$. That is, its boundary is part of the set itself. Moreover, it can be shown that $\mathbf{clo}(S) = \mathbf{bou}(S) \cup S$ is the smallest closed set containing S.

In case S is convex, one can infer the convexity of the interior $\mathbf{int}(S)$ and its closure $\mathbf{clo}(S)$. The following theorem summarises this result.

Theorem 13.5. *Let $S \subseteq \mathbb{R}^n$ be a convex set with $\mathbf{int}(S) \neq \emptyset$. Let $x_1 \in \mathbf{clo}(S)$ and $x_2 \in \mathbf{int}(S)$. Then $x = \lambda x_1 + (1 - \lambda)x_2 \in \mathbf{int}(S)$ for all $\lambda \in (0, 1)$.*

Theorem 13.5 is useful for inferring the convexity of the elements related to S. We summarise the key results in the following corollary.

Corollary 13.6. *Let S be a convex set with $\mathbf{int}(S) \neq \emptyset$. Then*

1. $\mathbf{int}(S)$ *is convex;*
2. $\mathbf{clo}(S)$ *is convex;*
3. $\mathbf{clo}(\mathbf{int}(S)) = \mathbf{clo}(S)$*;*
4. $\mathbf{int}(\mathbf{clo}(S)) = \mathbf{int}(S)$.

13.4.2 The Weierstrass theorem

The Weierstrass theorem is a result that guarantees the existence of optimal solutions for optimisation problems. To make it more precise, let

$$(P): \ z = \text{min. } \{f(x) : x \in S\}$$

be our optimisation problem. If an optimal solution x^* exists, then $f(x^*) \leq f(x)$ for all $x \in S$ and $z = f(x^*) = \min\{f(x) : x \in S\}$.

Notice the difference between min. (an abbreviation for minimise) and the operator min. The first is meant to represent the problem of minimising the function f in the domain S, while min is shorthand for minimum, in this case z, assuming that it is attainable.

It might be that an optimal solution is not attainable, but a bound can be obtained for the optimal solution value. The greatest lower bound for z is its *infimum* (or *supremum* for maximisation problems), denoted by inf. That is, if $z = \inf\{f(x) : x \in S\}$, then $z \leq f(x)$ for all $x \in S$ and there is no $\overline{z} > z$ such that $\overline{z} \leq f(x)$ for all $x \in S$. We might sometimes use the notation

$$(P): \ z = \inf\{f(x) : x \in S\}$$

to represent optimisation problems for which one cannot be sure whether an optimal solution is attainable. The Weierstrass theorem describes the situations in which those minimums (or maximums) are guaranteed to be attained, which is the case whenever S is compact.

Theorem 13.7 (Weierstrass theorem)**.** *Let $S \neq \emptyset$ be a compact set, and let $f : S \to \mathbb{R}$ be continuous on S. Then there exists a solution $\overline{x} \in S$ to min. $\{f(x) : x \in S\}$.*

Figure 13.7 illustrates three examples. In the first (on the left) the domain $[a, b]$ is compact, and thus the minimum of f is attained at b. In the other two, $[a, b)$ is open and therefore, Weierstrass theorem does not hold. In the middle example, one can obtain $\inf f$, which is not the case for the last example on the right.

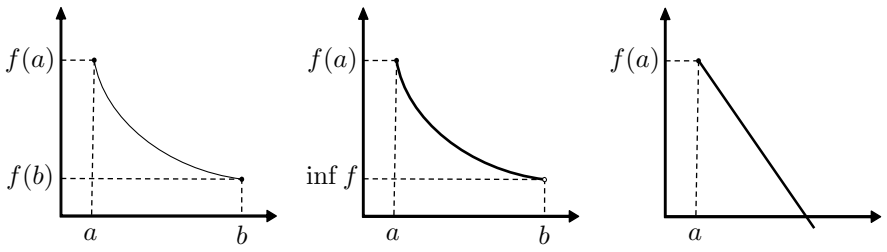


Figure 13.7: Examples of attainable minimum (left) and infimum (centre) and an example where neither are attainable (right).

13.5 Separation and support of sets

The concepts of *separation* and *support* of sets are key for establishing optimality conditions later in this course. We are interested in mechanisms that allow one to infer whether there exists hyperplanes separating points from sets (or sets from sets). We will also be interested in means to, given a point $x \notin S$, find the closest to point not belonging to S.

13.5.1 Hyperplanes and closest points

We start with how to identify closest points to sets.

Theorem 13.8 (Closest-point theorem)**.** *Let $S \neq \emptyset$ be a closed convex set in $\mathbb{R}^n$ and $y \notin S$. Then, there exists a unique point $\overline{x} \in S$ with minimum distance from y. In addition, $\overline{x}$ is the minimising point if and only if*

$$(y - \overline{x})^\top (x - \overline{x}) \leq 0, \text{ for all } x \in S$$

Simply put, if S is a closed convex set, then $\overline{x} \in S$ will be the closest point to $y \notin S$ if the vector $y - \overline{x}$ is such that if it forms an angle that is greater or equal than $90°$ with all other vectors $x - \overline{x}$ for $x \in S$. Figure 13.8 illustrates this logic.

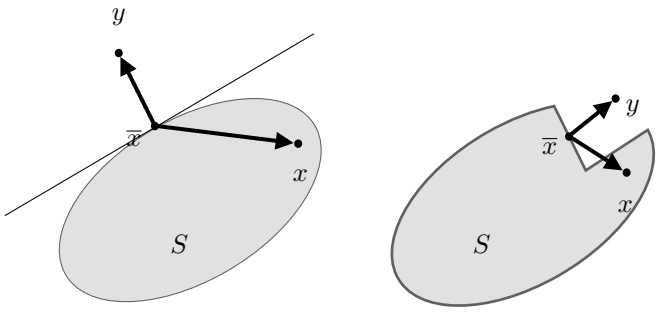


Figure 13.8: Closest-point theorem for a closed convex set (on the left). On the right, an illustration on how the absence of convexity invalidates the result.

Notice that S lies in the half-space $(y - \overline{x})^\top (x - \overline{x}) \leq 0$ defined by the hyperplane $p^\top (x - \overline{x}) = 0$ with normal vector $p = (y - \overline{x})$. We will next revise the concepts of half-spaces and hyperplanes, since they will play a central role in the derivations in this course.

13.5.2 Halfspaces and separation

We can use halfspaces to build the concept of separation. Let us start recalling that a hyperplane $H = \{x : p^\top x = \alpha\}$ with normal vector $p \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ defines two half-spaces $H^+ = \{x : p^\top x \geq \alpha\}$ and $H^- = \{x : p^\top x \leq \alpha\}$. Figure 13.9 illustrates the concept. Notice how the vector p lies in the half-space H^+.

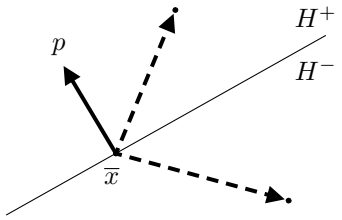


Figure 13.9: Normal vectors, hyperplane and halfspaces

Any hyperplane H can be defined in reference to a point $\overline{x} \in H$ by noticing that

$$p^\top(x - \overline{x}) = p^\top x - p^\top \overline{x} = \alpha - \alpha = 0.$$

From that, the half-spaces defined by H can be equivalently stated as $H^+ = \{x : p^\top(x - \overline{x}) \geq 0\}$ and $H^- = \{x : p^\top(x - \overline{x}) \leq 0\}$.

We can now define the separation of convex sets.

Definition 13.9. *Let S_1 and S_2 be nonempty sets in $\mathbb{R}^n$. The hyperplane $H = \{x : p^\top x = \alpha\}$ is said to* separate *S_1 and S_2 if $p^\top x \geq \alpha$ for each $x \in S_1$ and $p^\top x \leq \alpha$ for each $x \in S_2$. In addition, the following apply:*

1. **Proper separation:** *$S_1 \cup S_2 \not\subset H$;*
2. **Strict separation:** *$p^\top x < \alpha$ for each $x \in S_1$ and $p^\top x > \alpha$ for each $x \in S_2$;*
3. **Strong separation:** *$p^\top x \geq \alpha + \epsilon$ for some $\epsilon > 0$ and $x \in S_1$, and $p^\top x \leq \alpha$ for each $x \in S_2$.*

Figure 13.10 illustrates the three types of separation in Definition 13.9. On the left, proper separation is illustrated, which is obtained by any hyperplane that does not contain both S_1 and S_2, but that might contain points from either or both. In the middle, sets S_1 and S_2 belong to two distinct half-spaces in a strict sense. On the right, strict separation holds with an additional margin $\epsilon > 0$, which is defined as strong separation.

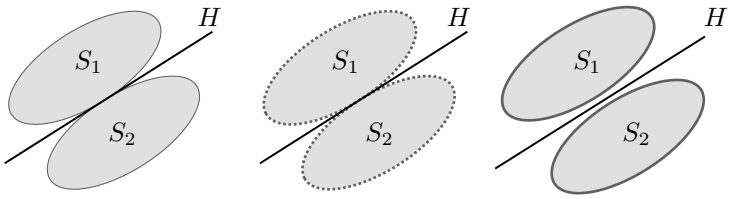


Figure 13.10: Three types of separation between S_1 and S_2.

A powerful yet simple result that we will use later is that, for a closed convex set S, there always exists a hyperplane separating S and a point y that does not belong to S.

Theorem 13.10 (Separation theorem)**.** *Let $S \neq \emptyset$ be a closed convex set in $\mathbb{R}^n$ and $y \notin S$. Then, there exists a nonzero vector $p \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ such that $p^\top x \leq \alpha$ for each $x \in S$ and $p^\top y > \alpha$.*

Proof. Theorem 13.8 guarantees the existence of a unique minimising $\overline{x} \in S$ such that $(y - \overline{x})^\top(x - \overline{x}) \leq 0$ for each $x \in S$. Let $p = (y - \overline{x}) \neq 0$ and $\alpha = \overline{x}^\top(y - \overline{x}) = p^\top \overline{x}$. Then we get $p^\top x \leq \alpha$ for each $x \in S$, while $p^\top y - \alpha = (y - \overline{x})^\top(y - \overline{x}) = ||y - \overline{x}||^2 > 0$. □

This is the first proof we look at in these notes, and the reason for that is its importance in many of the results we will discuss further. The proof first looks at the problem of finding a minimum distance point as an optimisation problem and uses the Weierstrass theorem (our Theorem 13.8 is a consequence of the Weierstrass theorem stated in Theorem 13.7) to guarantee that such a $\overline{x}$ exists. Being a minimum distance point, we know from Theorem 13.8 that $(y - \overline{x})^\top(x - \overline{x}) \leq 0$ holds. Now by defining p and α as in the proof, one might notice that

$$(y - \overline{x})^\top(x - \overline{x}) \leq 0 \;\Leftrightarrow\; (y - \overline{x})^\top x \leq (y - \overline{x})^\top \overline{x} \;\Leftrightarrow\; p^\top x \leq p^\top \overline{x} = \alpha.$$

The inequality $p^\top y > \alpha$ is demonstrated to hold in the final part by noticing that

$$\begin{aligned} p^\top y - \alpha &= (y - \overline{x})^\top y - \overline{x}^\top (y - \overline{x}) \\ &= y^\top (y - \overline{x}) - \overline{x}^\top (y - \overline{x}) \\ &= (y - \overline{x})^\top (y - \overline{x}) = ||y - \overline{x}||^2 > 0. \end{aligned}$$

Theorem **??** has interesting consequences. For example, one can apply it to every point in the boundary $\mathbf{bou}(S)$ to show that S is formed by the intersection of all half-spaces containing S.

Another interesting result is the existence of strong separation. If $y \notin \mathbf{clo}(\mathbf{conv}(S))$, then one can show that strong separation between y and S exists since there will surely be a distance $\epsilon > 0$ between y and S.

13.5.3 Farkas' theorem

Farkas' theorem plays a central role in deriving optimality conditions. It can assume several alternative forms, which are typically referred to as Farkas' lemmas. In essence, the Farkas' theorem is used to demonstrate that a given system of linear equations has a solution if and only if a related system can be shown to have no solutions and vice-versa.

Theorem 13.11. *Let A be an $m \times n$ matrix and c be an n-vector. Then exactly one of the following two systems has a solution:*

$$\begin{aligned} &(1): \ Ax \leq 0, \ c^\top x > 0, \ x \in \mathbb{R}^n \\ &(2): \ A^\top y = c, \ y \geq 0, \ y \in \mathbb{R}^m. \end{aligned}$$

Proof. Suppose (2) has a solution. Let x be such that $Ax \leq 0$. Then $c^\top x = (A^\top y)^\top x = y^\top Ax \leq 0$. Hence, (1) has no solution.

Next, suppose (2) has no solution. Let $S = \left\{x \in \mathbb{R}^n : x = A^\top y, \ y \geq 0\right\}$. Notice that S is closed and convex and that $c \notin S$. By Theorem **??**, there exists $p \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ such that $p^\top c > \alpha$ and $p^\top x \leq \alpha$ for $x \in S$.

As $0 \in S$, $\alpha \geq 0$ and $p^\top c > 0$. Also, $\alpha \geq p^\top A^\top y = y^\top Ap$ for $y \geq 0$. This implies that $Ap \leq 0$, and thus p satisfies (1). □

The first part of the proof shows that, if we assume that system (2) has a solution, than $c^\top x > 0$ cannot hold for $y \geq 0$. The second part uses the separation theorem (Theorem **??**) to show that c can be seen as a point not belonging to the closed convex set S for which there is a separation hyperplane and that the existence of such plane implies that system (1) must hold. The set S is closed and convex since it is a conic combination of rows a_i, for $i = 1, \ldots, m$. Using the $0 \in S$, one can show that $\alpha \geq 0$. The last part uses the identity $p^\top A^\top = (Ap)^\top$ and the fact that $(Ap)^\top y = y^\top Ap$. Notice that, since y can be arbitrarily large and α is a constant, $y^\top Ap \leq \alpha$ can only hold if $y^\top Ap \leq 0$, requiring that $p \leq 0$ since $y \geq 0$ from the definition of S.

Farkas' theorem has an interesting geometrical interpretation arising from this proof, as illustrated in Figure 13.11. Consider the cone C formed by the rows of A

$$C = \left\{ c \in \mathbb{R}^n : c_j = \sum_{i=1}^{m} a_{ij} y_i, \ j = 1, \ldots, n, \ y_i \geq 0, \ i = 1, \ldots, m \right\}$$

The *polar cone* of C, denoted C^0, is formed by the all vectors having angles of 90° or more with vectors in C. That is,

$$C^0 = \{x : Ax \leq 0\}.$$

Notice that (1) has a solution if the intersection between the polar cone C^0 and the positive (H^+ as defined earlier) half-space $H^+ = \left\{x \in \mathbb{R}^n : c^\top x > 0\right\}$ is not empty. If (2) has a solution, as in the beginning of the proof, then $c \in C$ and the intersection $C^0 \cap H^+ = \emptyset$. Now, if (2) does not have a solution, that is, $c \notin C$, then one can see that $C^0 \cap H^+$ cannot be empty, meaning that (1) has a solution.

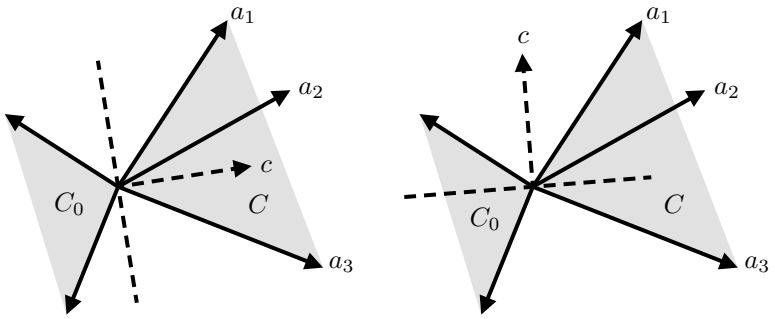


Figure 13.11: Geometrical illustration of the Farkas' theorem. On the left, system (2) has a solution, while on the right, system (1) has a solution

13.5.4 Supporting hyperplanes

There is an important connection between the existence of hyperplanes that support a whole set and optimality conditions of points. Let us first define supporting hyperplanes.

Definition 13.12 (Supporting hyperplane)**.** *Let $S \neq \emptyset$ be a set in $\mathbb{R}^n$, and let $\overline{x} \in \mathbf{bou}(S)$. $H = \left\{x \in \mathbb{R}^n : p^\top(x - \overline{x}) = 0\right\}$ is a supporting hyperplane of S at $\overline{x}$ if either $S \subseteq H^+$ (i.e., $p^\top(x - \overline{x}) \geq 0$ for $x \in S$) or $S \subseteq H^-$.*

Figure 13.12 illustrates the concept of supporting hyperplanes. Notice that supporting hyperplanes might not be unique, with the geometry of the set S playing an important role in that matter.

Let us define the function $f(x) = p^\top x$ with $x \in S$. One can see that the optimal solution $\overline{x}$ given by

$$\overline{x} = \operatorname*{argmax}_{x \in S} f(x)$$

is a point $x \in S$ for which p is a supporting hyperplane. A simple geometric analogy is to think that the f increases value as one moves in the direction of p. The constraint $x \in S$ will eventually prevent the movement further from S and this last contact point is precisely $\overline{x}$. This is a useful concept for optimising problem using gradients of functions, as we will discuss later in the course.

One characteristic that convex sets present that will be of great importance when establishing optimality conditions is the existence of supporting hyperplanes at every boundary point.

Theorem 13.13 (Support of convex sets)**.** *Let $S \neq \emptyset$ be a convex set in $\mathbb{R}^n$, and let $\overline{x} \in \mathbf{bou}(S)$. Then there exists $p \neq 0$ such that $p^\top(x - \overline{x}) \leq 0$ for each $x \in \mathbf{clo}(S)$.*

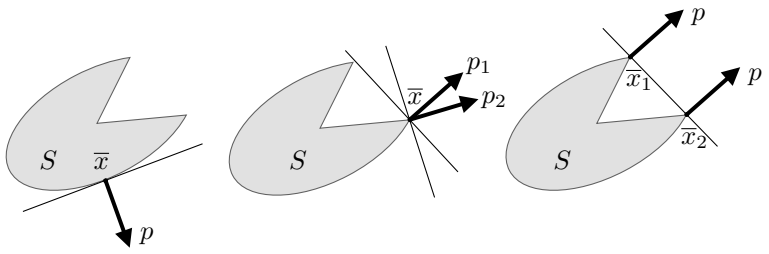


Figure 13.12: Supporting hyperplanes for an arbitrary set. Notice how a single point might have multiple supporting planes (middle) or different points might have the same supporting hyperplane (right)

The proof follows immediately from Theorem **??**, without explicitly considering a point $y \notin S$ and by noticing that $\mathbf{bou}(S) \subset \mathbf{clo}(S)$. Figure 13.13 provides an illustration of the theorem.

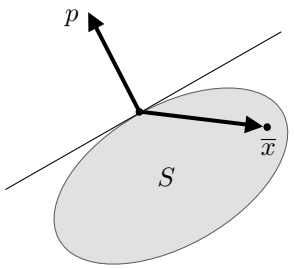


Figure 13.13: Supporting hyperplanes for convex sets. Notice how every boundary point has at least one supporting hyperplane

Chapter 14

Convex functions

14.1 Convexity in functions

Now we turn our attention to identifying the convexity of functions. Consider the general problem

$$\begin{aligned}(P): \quad \text{min. } & f(x) \\ \text{s.t.: } & g(x) \leq 0 \\ & x \in X\end{aligned}$$

with $f : \mathbb{R}^n \mapsto \mathbb{R}$, $g : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $X \subseteq \mathbb{R}^n$. Assuming X is a convex set, the next step towards attesting that (P) is a convex problem is to check whether f and g are convex. It is important to emphasise (perhaps redundantly at this point) how crucial is for us to be able to attest the convexity (P), since it allows us to generalise local optimality results to the whole domain of the problem.

The convexity of functions has a different definition than that used to define convex sets.

Definition 14.1 (Convexity of a function I)**.** *Let $f : S \mapsto \mathbb{R}$ where $S \subseteq \mathbb{R}^n$ is a nonempty convex set. The function f is said to be* convex *on S if*

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

for each $x_1, x_2 \in S$ and for each $\lambda \in [0, 1]$.

Very often, we use the term convex to loosely refer to concave functions, which must be done with caution. In fact, if f is convex, than $-f$ is concave and we say that (P) is a convex problem even if f is concave and we seek to maximise f instead. Also, linear functions are both convex and concave.

We say that a convex function is *strictly convex* if the inequality holds strictly in Definition 14.1 for each $\lambda \in (0, 1)$ (notice the open interval instead). In practice, it means that the function is guaranteed to not present flatness around its minimum (or maximum, for concave functions).

14.1.1 Example of convex functions

Some examples of convex function are:

1. $f(x) = a^\top x + b$;
2. $f(x) = e^x$;

3. $f(x) = x^p$ on $\mathbb{R}_+$ for $p \leq 0$ or $p \geq 1$; concave for $0 \leq p \leq 1$.
4. $f(x) = ||x||_p$ (p-norm);
5. $f(x) = -\log x$ and negative entropy $f(x) = -x \log x$ are concave;
6. $f(x) = \max\{x_1, \ldots, x_n\}$.

Knowing that these common functions are convex is helpful for identifying convexity in more complex functions formed by *composition*. By knowing that an operation between functions preserves convexity, we can infer the convexity of more complicated functions. The following are convexity preserving operations.

1. Let $f_1, \ldots, f_k : \mathbb{R}^n \mapsto \mathbb{R}$ be convex. Then these are convex:
   - $f(x) = \sum_{j=1}^k \alpha_j f_j(x)$ where $\alpha_j > 0$ for $j = 1, \ldots, k$;
   - $f(x) = \max\{f_1(x), \ldots, f_k(x)\}$;
2. $f(x) = \frac{1}{g(x)}$ on S, where $g : \mathbb{R}^n \mapsto \mathbb{R}$ is concave and $S = \{x : g(x) > 0\}$;
3. $f(x) = g(h(x))$, where $g : \mathbb{R} \mapsto \mathbb{R}$ is a nondecreasing convex function and $h : \mathbb{R}^n \mapsto \mathbb{R}$ is convex.
4. $f(x) = g(h(x))$, where $g : \mathbb{R}^m \mapsto \mathbb{R}$ is convex and $h : \mathbb{R}^n \mapsto \mathbb{R}^m$ is affine: $h(x) = Ax + b$ with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

14.1.2 Convex functions and their level sets

There is a strong connection between convexity of sets and the convexity of functions. Let us first consider *level sets*, which is one type of set spawned by functions.

Definition 14.2 (Lower level set)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty set. The lower level set of $f : \mathbb{R}^n \mapsto \mathbb{R}$ for $\alpha \in \mathbb{R}$ is given by*

$$S_\alpha = \{x \in S : f(x) \leq \alpha\}.$$

Figure 14.1 illustrates the lower level sets of two functions. The lower level set S_α can be seem as the projection of the function image onto the domain for a given level α.

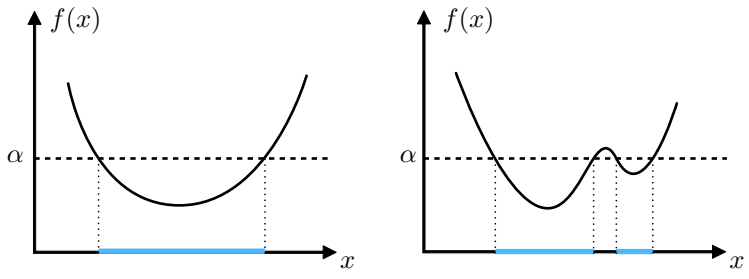


Figure 14.1: The lower level sets S_α (in blue) of two functions, given a value of α. Notice the nonconvexity of the level set of the nonconvex function (on the right)

Notice that, for convex functions, no discontinuity can be observed, making S_α convex. Lemma 14.3 states this property.

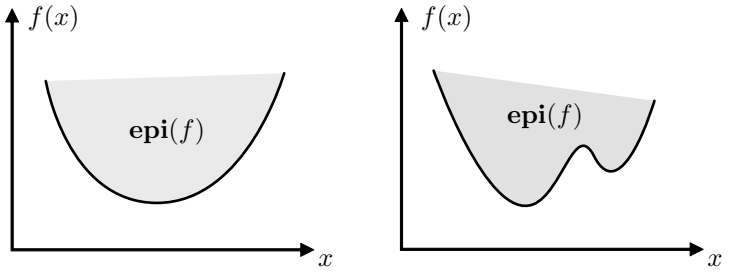


Figure 14.2: The epigraph **epi**()f of a convex function is a convex set (in grey on the left).

Lemma 14.3. *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$ a convex function. Then, any level set S_α with $\alpha \in \mathbb{R}$ is convex.*

Proof. Let $x_1, x_2 \in S_\alpha$. Thus, $x_1, x_2 \in S$ with $f(x_1) \leq \alpha$ and $f(x_2) \leq \alpha$. Let $\lambda \in (0, 1)$ and $x = \lambda x_1 + (1 - \lambda)x_2$. Since S is convex, we have that $x \in S$. Now, by the convexity of f, we have

$$f(x) \leq \lambda f(x_1) + (1 - \lambda)f(x_2) \leq \lambda\alpha + (1 - \lambda)\alpha = \alpha$$

and thus $x \in S_\alpha$. □

Remark: notice that a convex lower level set does not necessarily mean that the function is convex. In fact, as we will see later, there are nonconvex functions that have convex level sets (the so-called quasiconvex functions).

14.1.3 Convex functions and their epigraphs

Epigraphs, on the other hand, can be used to show the convexity of functions. Let us first formally define epigraphs.

Definition 14.4 (Ephigraph)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty set and $f : S \mapsto \mathbb{R}$. The epigraph of f is*

$$\mathbf{epi}(f) = \{(x, y) : x \in S, y \in \mathbb{R}, y \geq f(x)\} \subseteq \mathbb{R}^{n+1}$$

Figure 14.2 illustrates the epigraphs of two functions. Notice that the second function (on the right) is not convex, and nor is its epigraph. In fact, we can use the convexity of epigraphs (and the technical results associated with the convexity of sets) to show the convexity of functions.

Theorem 14.5 (Convex epigraphs)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$. Then f is convex if and only if* **epi**(f) *is a convex set.*

Proof. First, suppose f is convex and let $(x_1, y_1), (x_2, y_2) \in \mathbf{epi}(f)$ for $\lambda \in (0, 1)$. Then

$$\lambda y_1 + (1 - \lambda)y_2 \geq \lambda f(x_1) + (1 - \lambda)f(x_2) \geq f(\lambda x_1 + (1 - \lambda)x_2).$$

As $\lambda x_1 + (1 - \lambda)x_2 \in S$, $(\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2) \in \mathbf{epi}(f)$.

Conversely, suppose **epi**(f) is convex. Therefore $x_1, x_2 \in S$: $(x_1, f(x_1)) \in \mathbf{epi}(f)$, $(x_2, f(x_2)) \in \mathbf{epi}(f)$ and $(\lambda x_1 + (1-\lambda)x_2, \lambda f(x_1) + (1-\lambda)f(x_2)) \in \mathbf{epi}(f)$ for $\lambda \in (0, 1)$, implying that $\lambda f(x_1) + (1 - \lambda)f(x_2) \geq f(\lambda x_1 + (1 - \lambda)x_2)$. □

The proof starts with the implication "if f is convex, then $\mathbf{epi}(f)$ is convex". For that, it assumes that f is convex and use the convexity of f to show that any convex combination of x_1,x_2 in S will also be in the $\mathbf{epi}(f)$, which is the definition of a convex set.

To prove the implication "if $\mathbf{epi}(f)$ is convex, then f is convex", we define a convex combination of points in $\mathbf{epi}(f)$ and use the definition of $\mathbf{epi}(f)$ to show that f is convex by setting $y = \lambda f(x_1) + (1-\lambda)f(x_2)$ and $x = \lambda x_1 + (1-\lambda)x_2$.

14.2 Differentiability of functions

14.2.1 Subgradients and supporting hyperplanes

Subgradients can be understood as supporting hyperplanes at the boundary of function epigraphs. They can be seem as first-order local approximations of the function, which is often helpful information for optimisation methods when searching for directions of improvement.

Definition 14.6 (Subgradients)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$ a convex function. Then $\xi \in \mathbb{R}^n$ is a* subgradient *of f at $\overline{x} \in S$ if*

$$f(x) \geq f(\overline{x}) + \xi^\top (x - \overline{x}). \tag{14.1}$$

Inequality (14.1) is called the *subgradient inequality* and is going to be useful in several contexts later int his course. The set of subgradients ξ of f at $\overline{x}$ is the *subdifferential*

$$\partial_f(\overline{x}) = \left\{\xi \in \mathbb{R}^n : f(x) \geq f(\overline{x}) + \xi^\top (x - \overline{x})\right\}.$$

Every convex function $f : S \mapsto \mathbb{R}$ has at least one subgradient at any point $\overline{x}$ in the interior of the convex set S. Requiring that $\overline{x} \in \mathbf{int}(S)$ allows us to disregard boundary points of f where $\partial(\overline{x})$ might be empty. Theorem 14.7 presents this result.

Theorem 14.7. *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$ a convex function. Then for all $\overline{x} \in \mathbf{int}(S)$, there exists $\xi \in \mathbb{R}^n$ such that*

$$H = \left\{(x, y) : y = f(\overline{x}) + \xi^\top (x - \overline{x})\right\}$$

supports $\mathbf{epi}(f)$ at $(\overline{x}, f(\overline{x}))$. In particular,

$$f(x) \geq f(\overline{x}) + \xi^\top (x - \overline{x}), \forall x \in S.$$

The proof consists of directly applying Theorem 14.5 and then using the support of convex sets theorem (Theorem 14 in Lecture 2) to show that the subgradient inequality holds.

14.2.2 Differentiability and gradients for convex functions

Let us first define differentiability of a function.

Definition 14.8. *Let $S \subseteq \mathbb{R}^n$ be a nonempty set. The function $f : S \mapsto \mathbb{R}$ is differentiable at $\overline{x} \in \mathbf{int}(S)$ if there exists a vector $\nabla f(\overline{x})$, called a gradient vector, and a function $\alpha : \mathbb{R}^n \mapsto \mathbb{R}$ such that*

$$f(x) = f(\overline{x}) + \nabla f(\overline{x})^\top (x - \overline{x}) + ||x - \overline{x}||\alpha(\overline{x}; x - \overline{x})$$

where $\lim_{x \mapsto \overline{x}} \alpha(\overline{x}; x - \overline{x}) = 0$. If this is the case for all $\overline{x} \in \mathbf{int}(S)$, we say that the function is differentiable in S.

Notice that this definition is based on the existence of first-order (Taylor series) expansion, with an error term α. This definition is useful as it highlights the requirement that $\nabla f(\overline{x})$ exists and is unique at $\overline{x}$ since the gradient is given by $\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_i}\right]_i = 1, \ldots, n$.

If f is differentiable in S, then its subdifferential $\partial(x)$ is a singleton (a set with a single element) for all $x \in S$. This is shown in Lemma 14.9

Lemma 14.9. *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$ a convex function. Suppose that f is differentiable at $\overline{x} \in \mathbf{int}(S)$. Then $\partial_f(\overline{x}) = \{\nabla f(\overline{x})\}$, i.e., the subdifferential $\partial_f(\overline{x})$ is a singleton with $\nabla f(\overline{x})$ as its unique element.*

Proof. From Theorem 14.7, $\partial f(\overline{x}) \neq \emptyset$. Moreover, combining the existence of a subgradient ξ and differentiability of f at $\overline{x}$, we obtain:

$$f(\overline{x} + \lambda d) \geq f(\overline{x}) + \lambda \xi^\top d \tag{14.2}$$

$$f(\overline{x} + \lambda d) = f(\overline{x}) + \lambda \nabla f(\overline{x})^\top d + \lambda ||d|| \alpha(\overline{x}; \lambda d) \tag{14.3}$$

Subtracting (14.3) from (14.2), we get $0 \geq \lambda(\xi - \nabla f(\overline{x}))^\top d - \lambda ||d|| \alpha(\overline{x}; \lambda d)$. Dividing by $\lambda > 0$ and letting $\lambda \mapsto 0^+$, we obtain $(\xi - \nabla f(\overline{x}))^\top d \leq 0$. Now, by setting $d = \xi - \nabla f(\overline{x})$, it becomes clear that $\xi = \nabla f(\overline{x})$. □

Notice that in the proof we use $\overline{x} + \lambda d$ to indicate that x is in direction d, scaled by $\lambda > 0$. The fact that $\partial_f(x)$ is a singleton comes from the uniqueness of the solution for $(\xi - \nabla f(\overline{x}))^\top (\xi - \nabla f(\overline{x})) = 0$.

Figure 19.9 illustrates subdifferential sets for three distinct points of a piecewise linear function. The picture schematically represents a multidimensional space x as a one-dimensional projection (you can imagine this picture as being a section in one of the x dimensions). For the points in which the function is not differentiable, the subdifferential set contains an infinite number of subgradients. At points in which the function is differentiable (any mid-segment point) the subgradient is unique (a gradient) and the subdifferential is a singleton.

If $f : S \mapsto \mathbb{R}$ is a convex differentiable function, then Theorem 14.7 can be combined with Lemma 14.9 to express the one of the most powerful results relating f and its affine (first-order) approximation at $\overline{x}$.

Theorem 14.10 (Convexity of a function II)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex open set, and let $f : S \mapsto \mathbb{R}$ be differentiable on S. The function f is convex if and only if for any $\overline{x} \in S$, we have*

$$f(x) \geq f(\overline{x}) + \nabla f(\overline{x})^\top (x - \overline{x}), \ \forall x \in S.$$

The proof for this theorem follows from the proof for Theorem 14.7 to obtain the subgradient inequality and then use Lemma 14.9 to replace the subgradient with the gradient. To see how the opposite direction (subgradient inequality holing implying the convexity of f), one should proceed as follows.

1. Take x_1 and x_2 from S. The convexity of S implies that $\lambda x_1 + (1 - \lambda) x_2$ is also in S.

2. Assume that the subgradient exists, and therefore the two relations hold:

$$f(x_1) \geq f(\lambda x_1 + (1 - \lambda) x_2) + (1 - \lambda) \xi^\top (x_1 - x_2) \tag{14.4}$$

$$f(x_2) \geq f(\lambda x_1 + (1 - \lambda) x_2) + \lambda \xi^\top (x_2 - x_1) \tag{14.5}$$

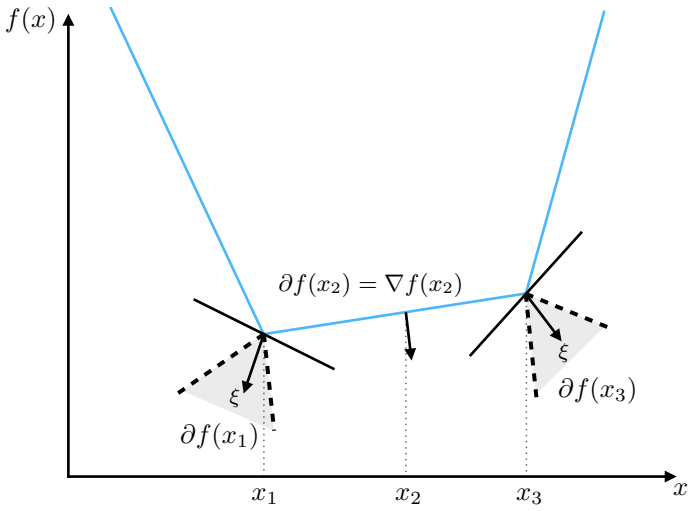


Figure 14.3: A representation of the subdifferential (in grey) for nondifferentiable (x_1 and x_3) and differentiable (x_2) points

3. Multiply (14.4) by λ, (14.5) by $(1-\lambda)$, and add them together. One will then obtain

$$\lambda f(x_1) + (1-\lambda) f(x_2) \geq f(\lambda x_1 + (1-\lambda) x_2),$$

which is implies convexity.

14.2.3 Second-order differentiability

We say that a function is *twice-differentiable* if it has a second-order Taylor expansion. Having second-order expansions can be useful in that it allows for encoding curvature information in the approximation, which is characterised by the *Hessian*, and to verify convexity (or strict convexity) by testing for semi-definiteness (positive definiteness).

Let $f_{ij}(\overline{x}) = \frac{\partial^2 f(\overline{x})}{\partial x_i \partial x_j}$. Recall that the Hessian matrix $H(\overline{x})$ at $\overline{x}$ is given by

$$H(\overline{x}) = \begin{bmatrix} f_{11}(\overline{x}) & \dots & f_{1n}(\overline{x}) \\ \vdots & \ddots & \vdots \\ f_{n1}(\overline{x}) & \dots & f_{nn}(\overline{x}) \end{bmatrix}$$

Second-order differentiability can be defined as follows.

Definition 14.11 (Second-order differentiability)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty set, and let $f : S \mapsto \mathbb{R}$. Then f is twice differentiable at $\overline{x} \in \mathbf{int}(S)$ if there exists a vector $\nabla f(\overline{x}) \in \mathbb{R}^n$, an $n \times n$ symmetric matrix $H(\overline{x})$ (the Hessian), and a function $\alpha : \mathbb{R}^n \mapsto \mathbb{R}$ such that*

$$f(x) = f(\overline{x}) + \nabla f(\overline{x})^\top (x - \overline{x}) + \frac{1}{2}(x - \overline{x})^\top H(\overline{x})(x - \overline{x}) + ||x - \overline{x}||^2 \alpha(\overline{x}; x - \overline{x})$$

where $\lim_{x \mapsto \overline{x}} \alpha(\overline{x}; x - \overline{x}) = 0$. If this is the case for all $\overline{x} \in S$, we say that the function is twice differentiable in S.

We say that $H(\overline{x})$ is *positive semi-definite* if $x^\top H(\overline{x})x \geq 0$ for $x \in \mathbb{R}^n$. Having a positive semi-definite Hessian for all $x \in S$ implies that the function is convex in S.

Theorem 14.12. *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex open set, and let $f : S \mapsto \mathbb{R}$ be twice differentiable on S. Then f is convex if and only if the Hessian matrix is positive semidefinite (PSD) at each point in S.*

Proof. Suppose f is convex and let $\overline{x} \in S$. Since S is open, $\overline{x} + \lambda x \in S$ for a small enough $|\lambda| \neq 0$. From Theorem 14.10 and twice differentiability of f, we have

$$f(\overline{x} + \lambda x) \geq f(\overline{x}) + \lambda \nabla f(\overline{x})^\top x \tag{14.6}$$

$$f(\overline{x} + \lambda x) = f(\overline{x}) + \lambda \nabla f(\overline{x})^\top x + \frac{1}{2}\lambda^2 x^\top H(\overline{x})x + \lambda^2 ||x||^2 \alpha(\overline{x}; \lambda x) \tag{14.7}$$

Subtracting (14.6) from (14.7), we get $\frac{1}{2}\lambda^2 x^\top H(\overline{x})x + \lambda^2 ||x||^2 \alpha(\overline{x}; \lambda x) \geq 0$. Diving by $\lambda^2 > 0$ and letting $\lambda \to 0$, it follows that $x^\top H(\overline{x})x \geq 0$.

Conversely, assume that $H(\overline{x})$ is PSD for all $\overline{x} \in S$. Using the mean value theorem and second-order expansion, one can show that

$$f(x) = f(\overline{x}) + \nabla f(\overline{x})^\top (x - \overline{x}) + \frac{1}{2}(x - \overline{x})^\top H(\hat{x})(x - \overline{x})$$

where $\hat{x} = \lambda \overline{x} + (1 - \lambda)x$ for $\lambda \in (0, 1)$. Note that $\hat{x} \in S$ and $H(\hat{x})$ is positive semidefinite by assumption. Thus $(x - \overline{x})^\top H(\hat{x})(x - \overline{x}) \geq 0$, implying $f(x) = f(\overline{x}) + \nabla f(\overline{x})^\top (x - \overline{x}) \geq 0$. □

The proof uses a trick we have seen before. First, we assume convexity and use the definition of convexity provided by Theorem 14.10 combined with an alternative definition for $(x - \overline{x})$ to show that $x^\top H(\overline{x})x \geq 0$. That is, instead of using the reference points x and $\overline{x}$, we incorporate a step size λ from $\overline{x}$ in the direction of x.

To show the other direction of implication, that is, that $x^\top H(\overline{x})x \geq 0$ implies convexity, we use the *mean value theorem*. The mean value theorem states that there must exist a point $\hat{x}$ between x and $\overline{x}$ for which the second order approximation is exact. From these, we can derive the definition of convexity, as in Theorem 14.10.

Checking for positive semi-definiteness can be done efficiently using appropriate computational algebra method, though it can be computationally expensive. It involves calculating the eigenvalues of $H(\overline{x})$ and testing whether they are all nonnegative (positive), which implies the positive semi-definiteness (definiteness) of $H(\overline{x})$. Some nonlinear solvers are capable of returning warning messages (or errors even) pointing out lack of convexity by testing (under a certain threshold) for positive semi-definiteness.

14.3 Quasiconvexity

Quasiconvexity can be seem as the generalisation of convexity to functions that are not convex, but share similar properties that allow for defining global optimality conditions. One class of these functions are named *quasiconvex*. Let us first technically define quasiconvex functions.

Definition 14.13 (quasiconvex functions)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$. Function f is quasiconvex if, for each $x_1, x_2 \in S$ and $\lambda \in (0, 1)$, we have*

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \max\{f(x_1), f(x_2)\}. \tag{14.8}$$

We say that, if f is quasiconvex, then $-f$ is quasiconcave. Also, functions that are both quasiconvex and quasiconcave are called *quasilinear*. Quasiconvex functions are also called *unimodal*.

Figure 14.4 illustrates a quasiconvex function. Notice that, for any pair of points x_1 and x_2 in the domain of f, the graph of the function is always below the maximum between $f(x_1)$ and $f(x_2)$. This is precisely what renders convex the lower level sets of quasiconvex functions. Notice that, on the other hand, the epigraph $epi(f)$ is not a convex set.

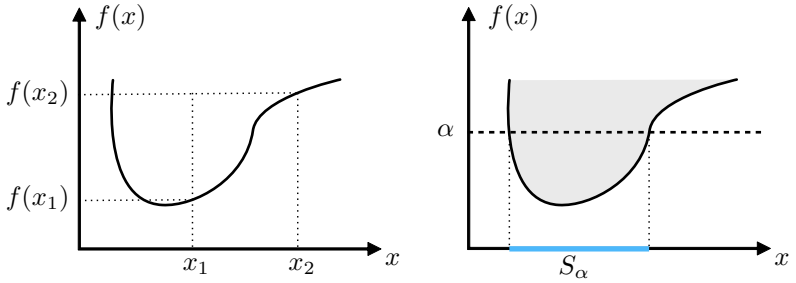


Figure 14.4: A quasiconvex function with its epigraph (in grey) and lower level set (in blue).

Examples of quasiconvex functions include:

1. $f(x) = \sqrt{|x|}$ in $\mathbb{R}$
2. $f(x) = \log x$ is quasilinear for $x > 0$
3. $f(x) = \inf\{z \in \mathbb{Z} : z \geq x\}$ is quasilinear
4. $f(x_1, x_2) = x_1 x_2$ is quasiconcave on $S = \{(x_1, x_2) \in \mathbb{R}^2 : x_1, x_2 > 0\}$

An important property of quasiconvex functions is that their level sets are convex.

Theorem 14.14. *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$. Function f is quasiconvex if and only if $S_\alpha = \{x \in S : f(x) \leq \alpha\}$ is convex for all $\alpha \in \mathbb{R}$.*

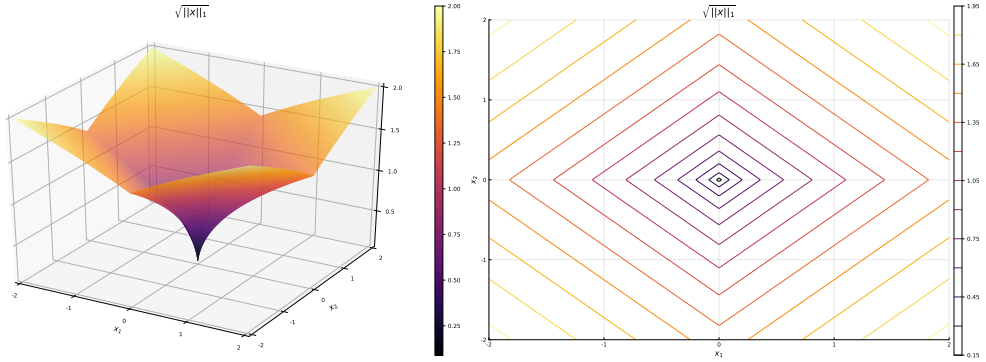
Proof. Suppose f is quasiconvex and let $x_1, x_2 \in S_\alpha$. Thus, $x_1, x_2 \in S$ and $\max\{f(x_1), f(x_2)\} \leq \alpha$. Let $x = \lambda x_1 + (1-\lambda)x_2$ for $\lambda \in (0,1)$. As S is convex, $x \in S$. By quasiconvexity of f, $f(x) \leq \max\{f(x_1), f(x_2)\} \leq \alpha$. Hence, $x \in S_\alpha$ and S_α is convex.

Conversely, assume that S_α is convex for $\alpha \in \mathbb{R}$. Let $x_1, x_2 \in S$, and let $x = \lambda x_1 + (1-\lambda)x_2$ for $\lambda \in (0,1)$. Note that, for $\alpha = \max\{f(x_1), f(x_2)\}$, we have $x_1, x_2 \in S_\alpha$. The convexity of S_α implies that $x \in S_\alpha$, and thus $f(x) \leq \alpha = \max\{f(x_1), f(x_2)\}$, which implies that f is quasiconvex. □

The proof relies on the convexity of the domain S to show that a convex combination from point in the level set S_α also belongs to S_α. To show the other way around, we simply need to define $\alpha = \max\{f(x_1), f(x_2)\}$ to see that a convex level set S_α implies that f is quasiconvex.

Quasiconvex functions have an interesting first-order condition that arises from the convexity of its level sets.

Theorem 14.15. *Let $S \subseteq \mathbb{R}^n$ be a nonempty open convex set, and let $f : S \mapsto \mathbb{R}$ be differentiable on S. Then f is quasiconvex if and only if, for $x_1, x_2 \in S$ and $f(x_1) \leq f(x_2)$, $\nabla f(x_2)^\top (x_1 - x_2) \leq 0$.*



(a) surface plot (b) level curves

Figure 14.5: Surface plot and level curves for $f(x) = \sqrt{||x||_1}$

The condition in Theorem 14.15 is in fact sufficient for global optimality if one can show that f is in fact *strictly quasiconvex*, that is when (14.8) holds strictly. Figure 14.5a and 14.5b show an example of a strict quasiconvex function and its level curves, illustrating that, despite the lack of convexity, the level sets are convex.

Strictly quasiconvex functions is a subset of a more general class of functions named *pseudoconvex*, for which the conditions in Theorem 14.15 are sufficient for global optimality.

Chapter 15

Unconstrained optimality conditions

15.1 Recognising optimality

We now turn our focus to recognising whether a given point satisfy necessary and/or sufficient conditions for optimality. Even though these conditions can be used to test if a candidate point is optimal for a problem, its most important use is serving as a framework for directing solution methods in their search for optimal solutions.

Before we proceed, let us define the terminology we will use to refer to solutions. Let $f : \mathbb{R}^n \mapsto \mathbb{R}$. Consider the problem (P) : min. $\{f(x) : x \in S\}$.

1. a *feasible solution* is any solution $\overline{x} \in S$;
2. a *local optimal solution* is a feasible solution $\overline{x} \in S$ that has a neighbourhood $N_\epsilon(\overline{x}) = \{x : ||x - \overline{x}|| \leq \epsilon\}$ for some $\epsilon > 0$ such that $f(\overline{x}) \leq f(x)$ for each $x \in S \cap N_\epsilon(\overline{x})$.
3. a *global optimal solution* is a feasible solution $\overline{x} \in S$ with $f(\overline{x}) \leq f(x)$ for all $x \in S$. Or alternatively, is a local optimal solution for which $S \subseteq N_\epsilon(\overline{x})$.

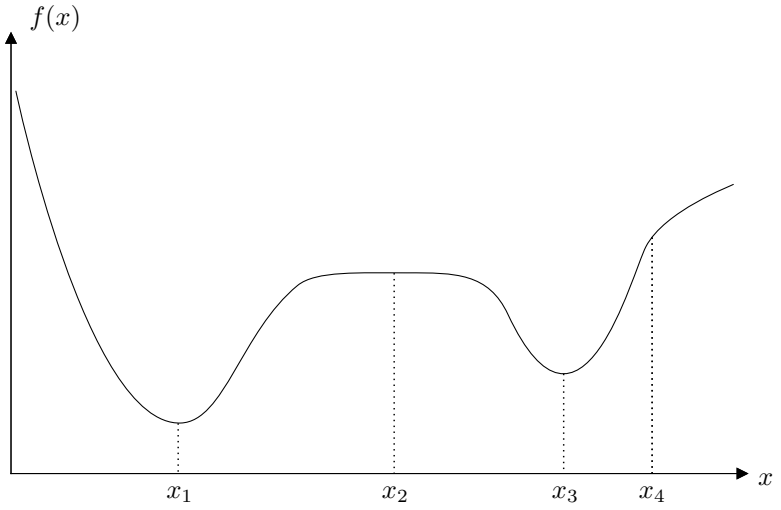
Figure 15.1 illustrates the concepts above. Solution x_1 is an unconstrained global minimum, but is not a feasible solution considering the feasibility set S. Solution x_2 is a local optima, for any neighbourhood $N_\epsilon(x_2)$ only encompassing the points within the same plateau. Solution x_3 is a local optimum, while x_4 is neither a local or a global optimum in the unconstrained case, but is a global maximum in the constrained case. Finally, x_5 is the global minimum in the constrained case.

15.2 The role of convexity in optimality conditions

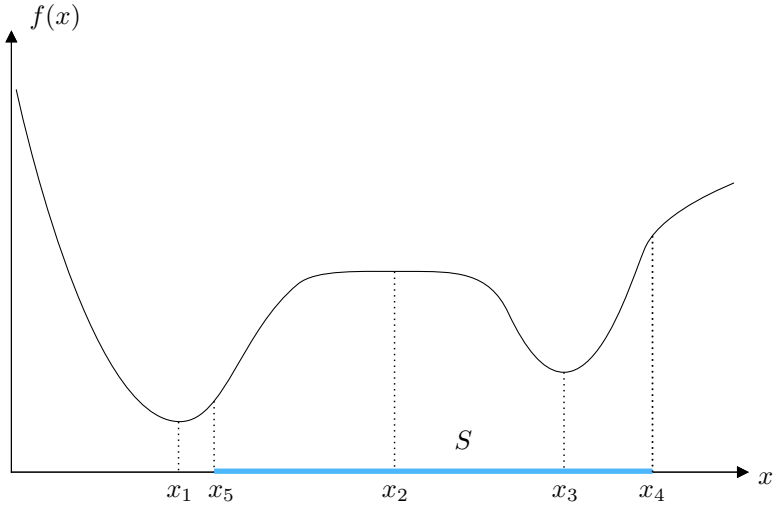
We can now state what is possibly the most important result in optimisation. In a nutshell, this results allows one promote local optimality to global optimality in the presence of convexity.

Theorem 15.1 (global optimality of convex problems)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : S \mapsto \mathbb{R}$ convex on S. Consider the problem (P) : min. $\{f(x) : x \in S\}$. Suppose $\overline{x}$ is a local optimal solution to P. Then $\overline{x}$ is a global optimal solution.*

Proof. Since $\overline{x}$ is a local optimal solution, there exists $N_\epsilon(\overline{x})$ such that, for each $x \in S \cap N_\epsilon(\overline{x})$, $f(\overline{x}) \leq f(x)$. By contradiction, suppose $\overline{x}$ is not a global optimal solution. Then, there exists a



(a) Unconstrained optimisation problem



(b) Constrained optimisation problem

Figure 15.1: Points of interest in optimisation. Points x_1, x_2 and x_3 are local optima in the unconstrained problem. Once a constraint set S is imposed, x_4 and x_5 become points of interest and x_1 becomes infeasible.

solution $\hat{x} \in S$ so that $f(\hat{x}) < f(\overline{x})$. Now, for any $\lambda \in [0,1]$, the convexity of f implies:

$$f(\lambda\hat{x} + (1-\lambda)\overline{x}) \leq \lambda f(\hat{x}) + (1-\lambda)f(\overline{x}) < \lambda f(\overline{x}) + (1-\lambda)f(\overline{x}) = f(\overline{x})$$

However, for $\lambda > 0$ sufficiently small, $\lambda\hat{x} + (1-\lambda)\overline{x} \in S \cap N_\epsilon(\overline{x})$ due to the convexity of S, which contradicts the local optimality of $\overline{x}$. Thus, $\overline{x}$ is a global optimum. □

The proof is built using contradiction. That is, we show that for a solution to be a local optimum in a convex problem, not being a global solution contradicts its local optimality, originally true by assumption. This is achieved using the convexity of f and showing that the convex combination between hypothetical better solution $\hat{x}$ and $\overline{x}$ would have to be both in $N_\epsilon(\overline{x})$ and better than $\overline{x}$, contradicting the local optimality of $\overline{x}$.

15.3 Optimality condition of convex problems

We first look at optimality conditions in a general sense to then translate the concept to unconstrained and constrained problems specifically. Taking this more general standpoint is also helpful to understand how these can be specialised in the absence of a closed domain or in the presence of differentiability. We assume convexity for now, and later we will discuss further the consequences of the absence of convexity. Note that unconstrained problems have convex feasibility set (i.e., the whole $\mathbb{R}^n$), and thus what follows can be generalised to unconstrained optimisation problems.

Theorem 15.2 (optimality condition for convex problems)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty convex set and $f : \mathbb{R}^n \mapsto \mathbb{R}$ convex on S. Consider the problem (P) : min. $\{f(x) : x \in S\}$. Then, $\overline{x} \in S$ is an optimal solution to (P) if and only if f has a subgradient ξ at $\overline{x}$ such that $\xi^\top(x - \overline{x}) \geq 0$ for all $x \in S$.*

Proof. Suppose that $\xi^\top(x - \overline{x}) \geq 0$ for all $x \in S$, where ξ is a subgradient of f at $\overline{x}$. By convexity of f, we have, for all $x \in S$

$$f(x) \geq f(\overline{x}) + \xi^\top(x - \overline{x}) \geq f(\overline{x})$$

and hence $\overline{x}$ is optimal.

Conversely, suppose that $\overline{x}$ is a global optimal for P. Construct the sets:

$$\Lambda_1 = \{(x - \overline{x}, y) : x \in \mathbb{R}^n,\ y > f(x) - f(\overline{x})\}$$
$$\Lambda_2 = \{(x - \overline{x}, y) : x \in S,\ y \leq 0\}$$

Note that Λ_1 and Λ_2 are convex. By optimality of $\overline{x}$, $\Lambda_1 \cap \Lambda_2 = \emptyset$. Using the *separation theorem*, there exists a hyperplane defined by $(\xi_0, \mu) \neq 0$ and α that separates Λ_1 and Λ_2:

$$\xi_0^\top(x - \overline{x}) + \mu y \leq \alpha,\ \forall x \in \mathbb{R}^n,\ y > f(x) - f(\overline{x}) \tag{15.1}$$
$$\xi_0^\top(x - \overline{x}) + \mu y \geq \alpha,\ \forall x \in S,\ y \leq 0. \tag{15.2}$$

Letting $x = \overline{x}$ and $y = 0$ in (15.2), we get $\alpha \leq 0$. Next, letting $x = \overline{x}$ and $y = \epsilon > 0$ in (15.1), we obtain $\alpha \geq \mu\epsilon$. As this holds for any $\epsilon > 0$, we must have $\mu \leq 0$ and $\alpha \geq 0$, the latter implying $\alpha = 0$.

If $\mu = 0$, we get from (15.1) that $\xi_0^\top(x - \overline{x}) \leq 0$ for all $x \in \mathbb{R}^n$. Now, by letting $x = \overline{x} + \xi_0$, it follows that $\xi_0^\top(x - \overline{x}) = ||\xi_0||^2 \leq 0$, and thus $\xi_0 = 0$. Since $(\xi_0, \mu) \neq 0$, we must have $\mu < 0$.

Dividing (15.1) and (15.2) by $-\mu$ and denoting $\xi = \frac{-\xi_0}{\mu}$, we obtain:

$$\xi^\top(x - \overline{x}) \leq y, \ \forall x \in \mathbb{R}^n, \ y > f(x) - f(\overline{x}) \tag{15.3}$$

$$\xi^\top(x - \overline{x}) \geq y, \ \forall x \in S, \ y \leq 0 \tag{15.4}$$

Letting $y = 0$ in (15.4), we get $\xi^\top(x - \overline{x}) \geq 0$ for all $x \in S$. From (15.3), we can see that $y > f(x) - f(\overline{x})$ and $y \geq \xi^\top(x - \overline{x})$. Thus, $f(x) - f(\overline{x}) \geq \xi^\top(x - \overline{x})$, which is the *subgradient inequality*. Thus, ξ is a subgradient at $\overline{x}$ with $\xi^\top(x - \overline{x}) \geq 0$ for all $x \in S$. □

In the first part of the proof, we use the definition of convexity based on the subgradient inequality to show that $\xi^\top(x - \overline{x}) \geq 0$ implies that $f(\overline{x}) \leq f(x)$ for all $x \in S$. The second part of the proof uses the separation theorem in a creative way to show that the subgradient inequality must hold if $\overline{x}$ is optimal. This is achieved by using the two sets Λ_1 and Λ_2. Notice that, $\overline{x}$ being optimal implies that $y > f(x) - f(\overline{x}) \geq 0$, which leads to the conclusion that $\Lambda_1 \cap \Lambda_2 = \emptyset$, demonstrating the existence of a separating hyperplane between them, as shown in (15.1) and (15.2). We can show that α in those has to be 0 by noticing that $\mu\epsilon \leq 0$ must hold for $\epsilon > 0$ to be a bounded constant.

The second part is dedicated to show that $\mu < 0$, so we can divide (15.1) and (15.2) by μ to obtain the subgradient inequality as we have seen. We show that by contradiction, since $\mu = 0$ would imply $\xi_0 = 0$, which disagrees with existence of a $(\xi, \mu) \neq 0$ in the separation theorem. Finally, as $y > f(x) - f(\overline{x})$ and $y \geq \xi^\top(x - \overline{x})$, for any given y, we have that $f(x) - f(\overline{x}) \geq \xi^\top(x - \overline{x})$ [1], which leads to the subgradient inequality.

Notice that this result provides necessary and sufficient conditions for optimality for convex problems. These conditions can be extended to the unconstrained case as well, which is presented in Corollary 15.3.

Corollary 15.3 (optimality in open sets)**.** *Under the conditions of Theorem 15.2, if S is open, $\overline{x}$ is an optimal solution to P if and only if $0 \in \partial f(\overline{x})$.*

Proof. From Theorem 15.2, $\overline{x}$ is optimal if and only if ξ is a subgradient at $\overline{x}$ with $\xi^\top(x - \overline{x}) \geq 0$ for all $x \in S$. Since S is open, $x = \overline{x} - \lambda\xi \in S$ for some $\lambda > 0$, and thus $-\lambda||\xi||^2 \geq 0$, implying $\xi = 0$. □

Notice that, if S is open, then the only way to attain the condition $\xi^\top(x - \overline{x}) \geq 0$ is if $\xi = 0$ itself. This is particularly relevant in the context of nondifferentiable functions, as we will see later. Another important corollary is the classic optimality condition $\nabla f(\overline{x}) = 0$, which we state below for completeness.

Corollary 15.4 (optimality for differentiable functions)**.** *Suppose that $S \subseteq \mathbb{R}^n$ is a nonempty convex set and $f : S \to \mathbb{R}$ a differentiable convex function on S. Then $\overline{x} \in S$ is optimal if and only if $\nabla f(\overline{x})^\top(x - \overline{x}) \geq 0$ for all $x \in S$. Moreover, if S is open, then $\overline{x}$ is optimal if and only if $\nabla f(\overline{x}) = 0$.*

The proof for Corollary 15.4 is the same as Theorem 15.2 under a setting where $\partial(x) = \{\nabla f(x)\}$ due to the differentiability of f.

Let us consider two examples. First, consider the problem

[1] Notice that, on the line of nonnegative reals, for a same y, $f(x) - f(\overline{x})$ is always on the 'right side' of $\xi^\top(x - \overline{x})$ because it is an open interval.

$$
\begin{aligned}
\text{min.} \quad & \left(x_1 - \frac{3}{2}\right)^2 + (x_2 - 5)^2 \\
\text{s.t.:} \quad & -x_1 + x_2 \leq 2 \\
& 2x_1 + 3x_2 \leq 11 \\
& x_1 \geq 0 \\
& x_2 \geq 0
\end{aligned}
$$

Figure 15.2 presents a plot of the feasible region S, which is form by the intersection of the two halfspaces, and the level curves of the objective function, with some of the values indicated in the curves. Notice that that this is a convex problem.

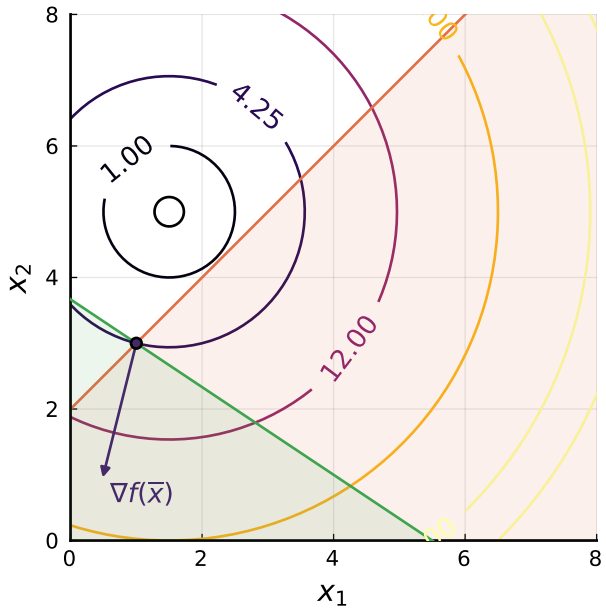


Figure 15.2: Example 1

The arrow shows the gradient $\nabla f(\overline{x})$ at $\overline{x} = (1, 3)$. Notice that this point is special since at that point, no vector $x - \overline{x}$ can be found forming an angle greater than 90° with $\nabla f(\overline{x})$, that is $\nabla f(\overline{x})^\top (x - \overline{x}) \geq 0$ for any $x \in S$, which means that $\overline{x}$ is optimal. Since the problem is convex, that is in fact the global optimum for this problem.

Figure 15.3 shows a similar situation, but now with one of the constraints being nonlinear. Notice that of the two points highlighted ((1,2) in orange and (2,1) in purple), the optimality condition only holds for (2,1). For example, for $x = (2, 1)$ and $\overline{x} = (1, 2)$ the vector $x - \overline{x}$ forms a angle greater than 90° with the gradient of f at $\overline{x}$, $\nabla f(\overline{x})$, and thus the condition $\nabla f(\overline{x})^(x - \overline{x}) \geq 0$ does not hold for all S. The condition does hold for $\overline{x} = (2, 1)$, as can be seen in Figure 15.3.

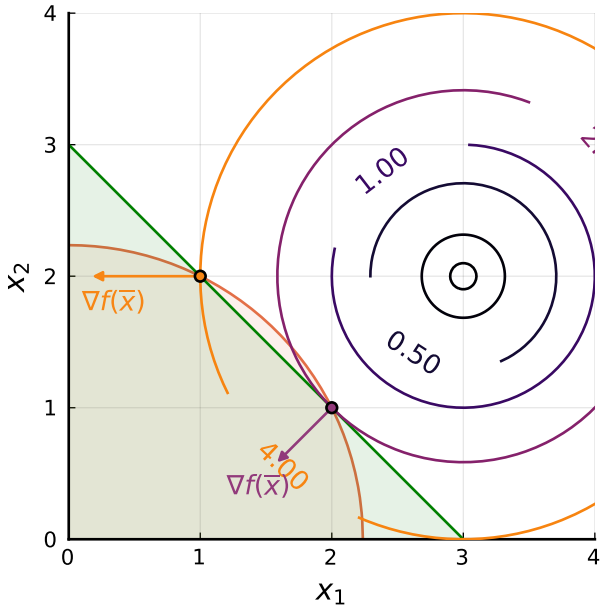


Figure 15.3: Example 2

A geometrical interpretation of the optimality condition $\xi^\top(x-\overline{x}) \geq 0$ is as follows. If there exists a subgradient ξ (or a gradient $\nabla f(\overline{x})$ if f is differentiable) that serves as a separating hyperplane between the level curve of f at $\overline{x}$ and the feasible region S, then there can be no feasible point further into the lower level set defined by that level curve. Ultimately, this means that there is no feasible point with smaller objective function value to be found. This is why the separation theorem from Lecture 2 plays an important role here, since it can be used to state that the feasible options have been exhausted in terms of potential directions of decrease of objective function value.

15.3.1 Optimality conditions for unconstrained problems

We have developed most of the concepts required to state optimality conditions for unconstrained optimisation problems, as presented in Corollaries 15.3 and 15.4. We now take an alternative route in which we do not take into account the feasibility set, but only the differentiability of f. This will be useful as it will allow us to momentarily depart from the assumption of convexity, which was used to state Theorem 15.2.

First-order optimality conditions

Let us start defining what it means to be a *descent direction*.

Theorem 15.5 (descent direction)**.** *Suppose $f : \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable at $\overline{x}$. If there is d such that $\nabla f(\overline{x})^\top d < 0$, there exists $\delta > 0$ such that $f(\overline{x} + \lambda d) < f(\overline{x})$ for each $\lambda \in (0, \delta)$, so that d is a descent direction of f at $\overline{x}$.*

Proof. By differentiability of f at $\overline{x}$, we have that

$$\frac{f(\overline{x}+\lambda d)-f(\overline{x})}{\lambda} = \nabla f(\overline{x})^\top d + ||d||\alpha(\overline{x};\lambda d).$$

Since $\nabla f(\overline{x})^\top d < 0$ and $\alpha(\overline{x};\lambda d) \to 0$ when $\lambda \to 0$ for some $\lambda \in (0,\delta)$, we must have $f(\overline{x}+\lambda d) - f(\overline{x}) < 0$. □

The proof uses the first-order expansion around $\overline{x}$ to show that, f being differentiable, the condition $\nabla f(\overline{x})^\top d < 0$ implies that $f(\overline{x}+\lambda d) < f(\overline{x})$, or put in words, that a step in the direction d decreases the objective function value.

We can derive the first-order optimality condition in Corollary 15.4 as a consequence from Theorem 15.5. Notice, however, that since convexity is not assumed, all we can say is that this condition is necessary (but not sufficient) for local optimality.

Corollary 15.6 (first-order necessary condition)**.** *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable at $\overline{x}$. If $\overline{x}$ is a local minimum, then $\nabla f(\overline{x}) = 0$.*

Proof. By contradiction, suppose that $\nabla f(\overline{x}) \neq 0$. Letting $d = -\nabla f(\overline{x})$, we have that $\nabla f(\overline{x})^\top d = -||\nabla f(\overline{x})||^2 < 0$. By Theorem 15.5, there exists a $\delta > 0$ such that $f(\overline{x}+\lambda d) < f(\overline{x})$ for all $\lambda \in (0,\delta)$, thus contradicting the local optimality of $\overline{x}$. □

Notice that Corollary 15.6 only holds in one direction. The proof uses contradiction once again, where we assume local optimality of $\overline{x}$ and show that having $\nabla f(\overline{x}) \neq 0$ contradicts the local optimality of $\overline{x}$, our initial assumption. To do that, we simply show that having any descent direction d (we use $-\nabla f(\overline{x})$ since in this setting it is guaranteed to exist as $\nabla f(\overline{x}) \neq 0$) would mean that small step λ can reduce the objective function value, contradicting the local optimality of $\overline{x}$.

Second-order optimality conditions

We now derive necessary conditions for local optimality of $\overline{x}$ based on second-order differentiability. As we will see, it requires that the Hessian $H(\overline{x})$ of $f(x)$ at $\overline{x}$ is positive semidefinite.

Theorem 15.7 (second-order necessary condition)**.** *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable at $\overline{x}$. If $\overline{x}$ is a local minimum, then $H(\overline{x})$ is positive semidefinite.*

Proof. Take an arbitrary direction d. As f is twice differentiable, we have:

$$f(\overline{x}+\lambda d) = f(\overline{x}) + \lambda \nabla f(\overline{x})^\top d + \frac{1}{2}\lambda^2 d^\top H(\overline{x}) d + \lambda^2 ||d||^2 \alpha(\overline{x};\lambda d)$$

since $\overline{x}$ is a local minimum, Corollary 15.6 implies that $\nabla f(\overline{x}) = 0$ and $f(\overline{x}+\lambda d) \geq f(\overline{x})$.

Rearranging terms and dividing by $\lambda^2 > 0$ we obtain

$$\frac{f(\overline{x}+\lambda d)-f(\overline{x})}{\lambda^2} = \frac{1}{2} d^\top H(\overline{x}) d + ||d||^2 \alpha(\overline{x};\lambda d).$$

Since $\alpha(\overline{x};\lambda d) \to 0$ as $\lambda \to 0$, we have that $d^\top H(\overline{x}) d \geq 0$. □

The second-order conditions can be used to attest local optimality of $\overline{x}$. In the case where $H(\overline{x})$ is positive definite, then this second order condition becomes *sufficient* for local optimality, since it implies that the function is 'locally convex' for a small enough neighbourhood $N_\epsilon(\overline{x})$.

In case f is convex, then the first-order condition $\nabla f(x) = 0$ becomes also sufficient for attesting the global optimality of $\overline{x}$. Recall that f is convex if and only if $H(x)$ is positive semidefinite for all $x \in \mathbb{R}^n$, meaning that in this case the second-order necessary conditions are also satisfied at $\overline{x}$.

Theorem 15.8. *Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be convex. Then $\overline{x}$ is a global minimum if and only if $\nabla f(\overline{x}) = 0$.*

Proof. From Corollary 15.6, if $\overline{x}$ is a global minimum, then $\nabla f(\overline{x}) = 0$. Now, since f is convex, we have that

$$f(x) \geq f(\overline{x}) + \nabla f(\overline{x})^\top (x - \overline{x})$$

Notice that $\nabla f(\overline{x}) = 0$ implies that $\nabla f(\overline{x})^\top (x - \overline{x}) = 0$ for each $x \in \mathbb{R}^n$, thus implying that $f(\overline{x}) \leq f(x)$ for all $x \in \mathbb{R}^n$. □

Chapter 16

Unconstrained optimisation methods: part 1

16.1 A prototype of an optimisation method

Most, if not all, optimisation methods are based on the conceptual notion of successively obtaining *directions* of potential improvement and suitable *step sizes* in this direction, until a convergence or termination criterion (collectively called stopping criteria) is satisfied.

Considering what we have seen so far, we have now the concepts required for describing several unconstrained optimisation methods. We start by posing a conceptual optimisation algorithm in a pseudocode structure. This will be helpful in identifying the elements that differentiate the methods we will discuss.

Algorithm 9 Conceptual optimisation algorithm

1: **initialise.** iteration count $k = 0$, starting point x_0
2: **while** stopping criteria are not met **do**
3: compute direction d_k
4: compute step size λ_k
5: $x_{k+1} = x_k + \lambda_k d_k$
6: $k = k + 1$
7: **end while**
8: **return** x_k.

Algorithm 22 has two main elements, namely the computation of the direction d_k and the step size λ_k at each iteration k. In what follows, we present some univariate optimisation methods that can be employed to calculate step sizes λ_k. These methods are commonly referred to as *line search methods*.

16.2 Line search methods

Finding an optimal step size λ_k is in itself an optimisation problem. The name line search refers to the fact that it consists of a unidimensional search as $\lambda_k \in \mathbb{R}$.

Suppose that $f : \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable. We define the unidimensional function $\theta : \mathbb{R} \mapsto \mathbb{R}$ as

$$\theta(\lambda) = f(x + \lambda d).$$

Assuming differentiability, we can use the first-order necessary condition $\theta'(\lambda) = 0$ to obtain optimal values for the step size λ. This means solving the system

$$\theta'(\lambda) = d^\top \nabla f(x + \lambda d) = 0$$

which might pose challenges. First, $d^\top \nabla f(x + \lambda d)$ is often nonlinear in λ, with optimal solutions not trivially resting at boundary points for an explicit domain of λ. Moreover, recall that $\theta'(\lambda) = 0$ is not a sufficient condition for optimality in general, unless properties such as convexity can be inferred.

In what follows, we assume that strict quasiconvexity holds and therefore $\theta'(\lambda) = 0$ becomes necessary and sufficient for optimality. In some contexts, unidimensional strictly quasiconvex functions are called *unimodal*.

Theorem 16.1 establishes the mechanism underpinning line search methods. In that, we use the assumption that the function has a unique minimum (a consequence of being strictly quasiconvex) to successively reduce the search space until the optimal is contained in a sufficiently small interval l within an acceptable tolerance.

Theorem 16.1 (Line search reduction)**.** *Let $\theta : \mathbb{R} \to \mathbb{R}$ be strictly quasiconvex over the interval $[a, b]$, and let $\lambda, \mu \in [a, b]$ such that $\lambda < \mu$. If $\theta(\lambda) > \theta(\mu)$, then $\theta(z) \geq \theta(\mu)$ for all $z \in [a, \lambda]$. If $\theta(\lambda) \leq \theta(\mu)$, then $\theta(z) \geq \theta(\lambda)$ for all $z \in [\mu, b]$.*

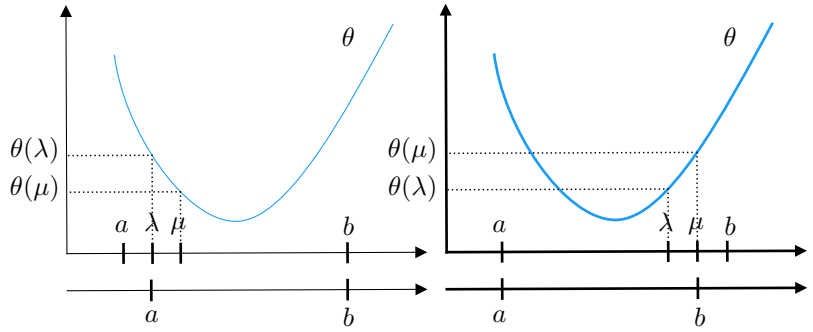


Figure 16.1: Applying Theorem 16.1 allows to iteratively reduce the search space.

Figure 16.1 provides an illustration of Theorem 16.1. The line below the x-axis illustrates how the search space can be reduced between two successive iterations. In fact, most line search methods will iteratively reduce the search interval (represented by $[a, b]$) until the interval is sufficiently small to be considered "a point" (i.e., is smaller than a set threshold l).

Line searches are *exact* when optimal step sizes λ_k^* are calculated at each iteration k, and inexact when arbitrarily good approximations for λ_k^* are used instead. As we will see, there is a trade-off between the number iterations required for convergence and the time taken per iteration that must be taken into account when choosing between exact and inexact line searches.

16.2.1 Exact line searches

Exact methods are designed to return the optimal step value λ^* within a pre-specified tolerance l. In practice, it means that these methods return an interval $[a_k, b_k]$ such that $b_k - a_k \leq l$.

Uniform search

The uniform search consists of breaking the search domain $[a, b]$ into N slices of uniform size $\delta = \frac{|b-a|}{N}$. This leads to a one-dimensional grid with grid points $a_n = a_0 + n\delta, n = 0 \dots N$ where $a_0 = a$ and $a_N = b$. We can then set $\hat{\lambda}$ to be

$$\hat{\lambda} = \arg \min_{i=0,\dots,n} f(a_i)$$

From Theorem 16.1, we know that the optimal step size $\lambda^* \in [\hat{\lambda} - \delta, \hat{\lambda} + \delta]$. The process can then be repeated, by making $a = \hat{\lambda} - \delta$ and $b = \hat{\lambda} + \delta$ (see Figure 16.2). until $|a - b|$ is less than a prespecified tolerance l. Without enough repetition of the search, the uniform search becomes an inexact search.

This type of search is particularly useful when setting values for hyperparameters in algorithms (that is, user defined parameters that influence the behaviour of the algorithm) of performing any sort of search in a grid structure. One concept related to this type of search is what is known as the *coarse-to-fine approach*. Coarse-to-fine approaches use sequences of increasingly fine approximations (i.e., gradually increasing n) to obtain computational savings in terms of function evaluations. In fact, the number of function evaluations a line search method executes is one of the indicators of its efficiency.

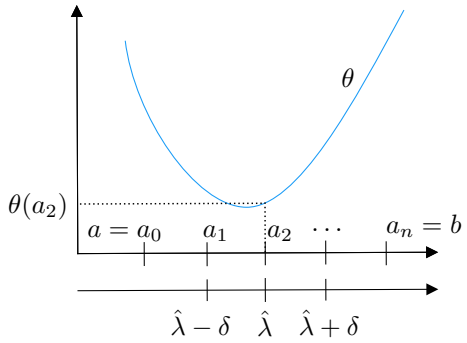


Figure 16.2: Grid search with 5 points; Note that $\theta(a_2) = \min_{i=0,\dots,n} \theta(a_i)$.

Dichotomous search

The *dichotomous search* is an example of a sequential line search method, in which evaluations of the function θ at a current iteration k are reused in the next iteration $k + 1$ to minimise the number of function evaluations and thus improve performance.

The word dichotomous refer to the mutually exclusive parts that the search interval $[a, b]$ is divided at each iteration. We start by defining a distance margin ϵ and defining two reference points $\lambda = \frac{a+b}{2} - \epsilon$ and $\mu = \frac{a+b}{2} + \epsilon$. Using the function values $\theta(\lambda)$ and $\theta(\mu)$, we proceed as follows.

1. If $\theta(\lambda) < \theta(\mu)$, then *move to the left* by making $a_{k+1} = a_k$ and $b_{k+1} = \mu_k$;
2. Otherwise, if $\theta(\lambda) > \theta(\mu)$, then *move to the right* by making $a_{k+1} = \lambda_k$ and $b_{k+1} = b_k$.

Notice that, the assumption of strict quasiconvexity implies that $\theta(\lambda) = \theta(\mu)$ cannot occur, but in a more general setting one must make sure a criterion for resolving the tie. Once the new search

interval $[a_{k+1}, b_{k+1}]$ is updated, new reference points λ_{k+1} and μ_{k+1} are calculated and the process is repeated until $|a-b| \leq l$. The method is summarised in Algorithm 23. Notice that, at any given iteration k, one can calculate what will be the size $|a_{k+1} - b_{k+1}|$, given by

$$b_{k+1} - a_{k+1} = \frac{1}{2^k}(b_0 - a_0) + 2\epsilon\left(1 - \frac{1}{2^k}\right).$$

This is useful in that it allows predicting the number of iterations Algorithm 23 will require before convergence. Figure 16.3 illustrates the process for two distinct functions. Notice that the employment of the central point $\frac{a+b}{2}$ as the reference to define the points λ and μ turns the method robust in terms of interval reduction at each iteration.

Algorithm 10 Dichotomous search

initialise. distance margin $\epsilon > 0$, tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
while $b_k - a_k > l$ **do**
  $\lambda_k = \frac{a_k+b_k}{2} - \epsilon$, $\mu_k = \frac{a_k+b_k}{2} + \epsilon$
  if $\theta(\lambda_k) < \theta(\mu_k)$ **then**
    $a_{k+1} = a_k$, $b_{k+1} = \mu_k$
  else
    $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$
  end if
  $k = k + 1$
end while
return $\overline{\lambda} = \frac{a_k+b_k}{2}$

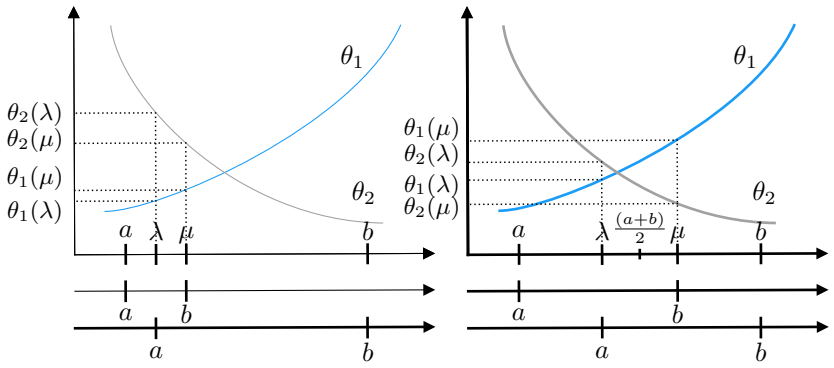


Figure 16.3: Using the midpoint $(a + b)/2$ and Theorem 16.1 to reduce the search space.

Golden section search*

The *golden section search* is named after the *golden ratio* $\varphi = \frac{1+\sqrt{5}}{2}$, of which the inverse is used as the ratio of reduction for the search interval $[a, b]$ at each iteration.

Consider that, once again, we rely on two reference points λ_k and μ_k. The method is a consequence of imposing two requirements for the line search:

1. the reduction in the search interval should not depend on whether $\theta(\lambda_k) > \theta(\mu_k)$ or vice-versa.

2. at each iteration, we perform a single function evaluation, thus making $\lambda_{k+1} = \mu_k$ if $\theta(\lambda_k) > \theta(\mu_k)$ or vice-versa.

From requirement 1, we can infer that $b_{k+1} - a_{k+1} = b_k - \lambda_k = \mu_k - a_k$ is required. To find the interval reduction rate $\alpha \in (0, 1)$ that would allow so, we define $\mu_k = a_k + \alpha(b_k - a_k)$ and, consequently, $\lambda_k = a_k + (1 - \alpha)(b_k - a_k)$. Notice that this makes $b_{k+1} - a_{k+1} = \alpha(b_k - a_k)$.

Notice the following. Suppose that $\theta(\lambda_k) > \theta(\mu_k)$ at iteration k. We then make $a_{k+1} = \lambda_k$ and $b_{k+1} = b_k$, a "movement to the right". From requirement 2, we also make $\lambda_{k+1} = \mu_k$ so that $\theta(\lambda_{k+1}) = \theta(\mu_k)$, avoiding a function evaluation.

From the above, we can calculate the ratio α that would allow the method to work. Notice that

$$\begin{aligned}
&\lambda_{k+1} = \mu_k \\
&a_{k+1} + (1 - \alpha)(b_{k+1} - a_{k+1}) = \mu_k \\
&(1 - \alpha)[\alpha(b_k - a_k)] = \mu_k - \lambda_k \\
&(\alpha - \alpha^2)(b_k - a_k) = a_k + \alpha(b_k - a_k) - [a_k + (1 - \alpha)(b_k - \alpha_k)] \\
&\alpha^2 + \alpha - 1 = 0
\end{aligned}$$

to which $\alpha = \frac{2}{1+\sqrt{5}} = 0.618... = \frac{1}{\varphi}$ is the positive solution. Clearly, the same result is obtained if one consider $\theta(\lambda_k) < \theta(\mu_k)$. Algorithm 24 summarises the golden section search. Notice that at each iteration, only a single additional function evaluation is required.

Algorithm 11 Golden section search

initialise. tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $\alpha = 0.618$, $k = 0$
$\lambda_k = a_k + (1 - \alpha)(b_k - a_k)$, $\mu_k = a_k + \alpha(b_k - a_k)$
while $b_k - a_k > l$ **do**
    if $\theta(\lambda_k) > \theta(\mu_k)$ **then**
        $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$, $\lambda_{k+1} = \mu_k$, and
        $\mu_{k+1} = a_{k+1} + \alpha(b_{k+1} - a_{k+1})$. Calculate $\theta(\mu_{k+1})$
    else
        $a_{k+1} = a_k$, $b_{k+1} = \mu_k$, $\mu_{k+1} = \lambda_k$, and
        $\lambda_{k+1} = a_{k+1} + (1 - \alpha)(b_{k+1} - a_{k+1})$. Calculate $\theta(\lambda_{k+1})$
    end if
    $k \leftarrow k + 1$
end while
return $\overline{\lambda} = \frac{a_k + b_k}{2}$

Comparing the above method for a given accuracy l, the required number of function evaluations is:

$$\min \left\{ n : \begin{array}{l} \text{uniform: } n \geq \frac{b_1 - a_1}{l/2} - 1 \\ \text{dichotomous: } (1/2)^{n/2} \leq \frac{l}{b_1 - a_1} \\ \text{golden section: } (0.618)^{n-1} \leq \frac{l}{b_1 - a_1} \end{array} \right\}$$

For example: suppose we set $[a, b] = [-10, 10]$ and $l = 10^{-6}$. Then the number of iterations required for convergence is

- uniform: $n = 4 \times 10^6$;

- dichotomous: $n = 49$;
- golden section: $n = 36$.

A variant of the golden section method uses Fibonacci numbers to define the ratio of interval reduction. Despite being marginally more efficient in terms of function evaluations, the overhead of calculating Fibonacci numbers has to be taken into account.

Bisection search

Differently form the previous methods, the bisection search relies on derivative information to infer whether how the search interval should be reduced. For that, we assume that $\theta(\lambda)$ is differentiable and convex.

We proceed as follows. If $\theta'(\lambda_k) = 0$, then λ_k is a minimiser. Otherwise

1. if $\theta'(\lambda_k) > 0$, then, for $\lambda > \lambda_k$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) > 0$, which implies $\theta(\lambda) \geq \theta(\lambda_k)$ since θ is convex. Therefore, the new search interval becomes $[a_{k+1}, b_{k+1}] = [a_k, \lambda_k]$.
2. if $\theta'(\lambda_k) < 0$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) > 0$ (and thus $\theta(\lambda) \geq \theta(\lambda_k)$) for $\lambda < \lambda_k$. Thus, the new search interval becomes $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$.

As in the dichotomous search, we set $\lambda_k = \frac{1}{2}(b_k + a_k)$, which provides robust guarantees of search interval reduction. Notice that the dichotomous search can be seen as a bisection search in which the derivative information is estimated using the difference of function evaluation at two distinct points. Algorithm 12 summarises the bisection method.

Algorithm 12 Bisection method

initialise. tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
while $b_k - a_k > l$ **do**
    $\lambda_k = \frac{(b_k + a_k)}{2}$ and evaluate $\theta'(\lambda_k)$
    if $\theta'(\lambda_k) = 0$ **then** return λ_k
    else if $\theta'(\lambda_k) > 0$ **then**
        $a_{k+1} = a_k$, $b_{k+1} = \lambda_k$
    else
        $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$
    end if
    $k \leftarrow k + 1$
end while
return $\overline{\lambda} = \frac{a_k + b_k}{2}$

16.2.2 Inexact line search

Often, it is worth sacrificing optimality of the step size λ^k for the overall efficiency of the solution method in terms of solution time.

There are several heuristics that can be employed to define step sizes and their performance are related to how the directions d_k are defined in Algorithm 22. Next, we present the *Armijo rule*, arguably the most used technique to obtain step sizes in efficient implementations of optimisation methods.

Armijo rule

The Armijo rule is a condition that is tested to decide whether a current step size $\overline{\lambda}$ is acceptable or not. The step size $\overline{\lambda}$ is considered acceptable if

$$f(x + d\overline{\lambda}) - f(x) \leq \alpha\overline{\lambda}\nabla f(x)^\top d.$$

One way of understanding the Armijo rule is to look at what it means in terms of the function $\theta(\lambda) = f(x + \lambda d)$. Notice that, at $\lambda = 0$, the Armijo rule becomes

$$\begin{aligned} \theta(\overline{\lambda}) - \theta(0) &\leq \alpha\overline{\lambda}\theta'(0) \\ \theta(\overline{\lambda}) &\leq \theta(0) + \alpha\overline{\lambda}\theta'(0). \end{aligned} \tag{16.1}$$

That is, $\theta(\overline{\lambda})$ has to be less than the deflected linear extrapolation of θ at $\lambda = 0$. The deflection is given by the pre-specified parameter α. In case $\overline{\lambda}$ does not satisfy the test in (16.1), $\overline{\lambda}$ is reduced by a factor $\beta \in (0, 1)$ until the test in (16.1) is satisfied.

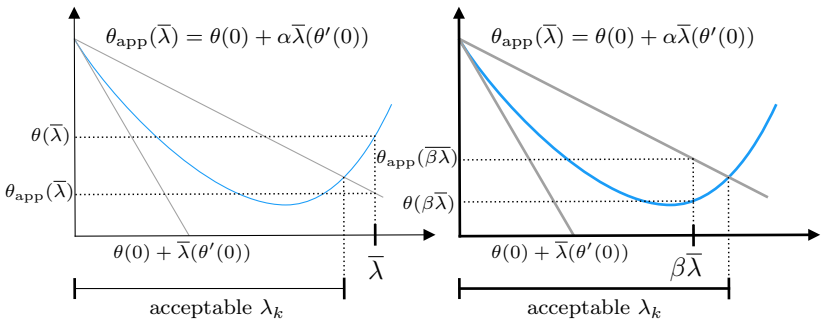


Figure 16.4: At first $\lambda_0 = \overline{\lambda}$ is not acceptable; after reducing the step size to $\lambda_1 = \beta\overline{\lambda}$, it enters the acceptable range where $\theta(\lambda_k) \leq \theta_{\text{app}}(\lambda_k) = \theta(0) + \alpha\lambda_k(\theta'(0))$.

In Figure 16.4, we can see the acceptable region for the Armijo test. At first, $\overline{\lambda}$ does not satisfy the condition (16.1), being then reduced to $\beta\overline{\lambda}$, which, in turn, satisfies (16.1). In this case, λ_k would have been set $\beta\overline{\lambda}$. Suitable values for α are within $(0, 0.5]$ and for β are within $(0, 1)$, trading of precision (higher values) and number of tests before acceptance (lower values).

The Armijo rule is called *backtracking* in some contexts, due to the successive reduction of the step size caused by the factor $\beta \in (0, 1)$. Some variants might also include rules that prevent the step size from becoming too small, such as $\theta(\delta\overline{\lambda}) \geq \theta(0) + \alpha\delta\overline{\lambda}\theta'(0)$, with $\delta > 1$.

16.3 Unconstrained optimisation methods

We now focus on developing methods that can be employed to optimise $f : \mathbb{R}^n \mapsto \mathbb{R}$. We start with coordinate descent method, which is derivative free, to then discuss the gradient method and Newton's method. In essence, the main difference between the three methods is how the directions d_k in Algorithm 22 are determined. Also, all of these methods rely on line searches to define optimal step sizes, which can be any of the methods seen before or any other unidimensional optimisation method.

16.3.1 Coordinate descent

The *cordinate descent method* relies on a simple yet powerful idea. By focusing on one coordinate at the time, the method trivially derives directions d having $d_i = 1$ for coordinate i and $d_{j \neq i} = 0$ otherwise. As one would suspect, the order in which the coordinates are selected influences the performance of the algorithm. Some known variants include:

1. **Cyclic:** coordinates are considered in order $1, \ldots, n$;
2. **Double-sweep:** swap the coordinate order at each iteration;
3. **Gauss-Southwell:** choose components with largest $\frac{\partial f(x)}{\partial x_i}$;
4. **Stochastic:** coordinates are selected at random.

Algorithm 13 summarises the general structure of the coordinate descent method. Notice that the for-loop starting in Line 3 uses the cyclic variant of the coordinate descent method.

Algorithm 13 Coordinate descent method (cyclic)

1: **initialise.** tolerance $\epsilon > 0$, initial point x^0, iteration count $k = 0$
2: **while** $||x^{k+1} - x^k|| > \epsilon$ **do**
3: **for** $j = 1, \ldots n$ **do**
4: $d = \{d_i = 1, \text{ if } i = j; d_i = 0, \text{ if } i \neq j\}$
5: $\overline{\lambda}_j = \text{argmin}_{\lambda \in \mathbb{R}} \left\{ f(x_j^k + \lambda d_j) \right\}$
6: $x_j^{k+1} = x_j^k + \overline{\lambda}_j d_j$
7: **end for**
8: $k = k + 1$
9: **end while**
10: **return** x^k

Figure 16.5 shows the progress of the algorithm when applied to solve

$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$

using the golden section method as line search.

The coordinate descent method is the strategy employed in several other methods, such as the *Gauss-Seidel* method for solving linear system of equations, which is why some references refer to each of these iterations as Gauss-Seidel steps. Also, when a collection of coordinates is used to derive a direction, the term *block coordinate descent* is used, though a method for deriving directions for each block is still necessary, for example the gradient method presented next.

16.3.2 Gradient (descent) method

The *gradient descent* uses the function gradients as the search direction d. Before we present the method, let us present a result that justifies the use of gradients to derive search directions.

Lemma 16.2. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable at $x \in \mathbb{R}^n$ and $\nabla f(x) \neq 0$. Then $\overline{d} = -\frac{\nabla f(x)}{||\nabla f(x)||}$ is the direction of steepest descent of f at x.*

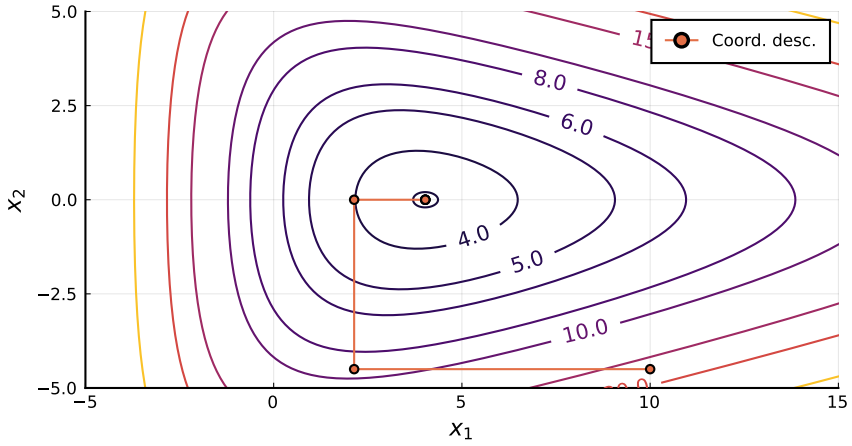


Figure 16.5: Coordinate descent method applied to f. Convergence is observed in 4 steps for a tolerance $\epsilon = 10^{-5}$

Proof. From differentiability of f, we have

$$f'(x;d) = \lim_{\lambda \to 0^+} \frac{f(x+\lambda d) - f(x)}{\lambda} = \nabla f(x)^\top d.$$

Thus, $\overline{d} = \text{argmin}_{||d|| \leq 1} \left\{ \nabla f(x)^\top d \right\} = -\frac{\nabla f(x)}{||\nabla f(x)||}$ □

In the proof, we use the differentiability to define a directional derivative for f at direction d, that is, the change in the value of f by a move of size $\lambda > 0$ in the direction d, which is given by $\nabla f(x)^\top d$. If we minimise this term in d for $||d||_2 \leq 1$, we observe that d is a vector of length one that has the opposite direction of $\nabla f(x)$, thus $d = -\frac{\nabla f(\overline{x})}{||\nabla f(\overline{x})||}$.

That provides us with the insight that we can use $\nabla f(\overline{x})$ to derive (potentially good) directions for optimising f. Notice that the direction employed is the opposite direction of the gradient for minimisation problems, being the opposite in case of maximisation. That is the reason why the gradient method is called the *steepest descent* method in some references, though gradient and steepest descent might refer to different methods in specific contexts.

Using the gradient $\nabla f(\overline{x})$ is also a convenience as it allows for the definition of a straightforward convergence condition. Notice that, if $\nabla f(\overline{x}) = 0$, then the algorithm stalls, as $x_{k+1} = x_k + \lambda_k d_k = x_k$. In other words, the algorithm converges to points $x \in \mathbb{R}^n$ that satisfy the first-order necessary conditions $\nabla f(\overline{x}) = 0$.

The gradient method has many known variants that try to mitigate issues associated with the poor convergence caused by the natural 'zigzagging' behaviour of the algorithm (see, for example the gradient method *with momentum* and the *Nesterov* method).

There are also variants that only consider the partial derivatives of some (and not all) of the dimensions $i = 1, \ldots, n$ forming *blocks* of coordinates at each iteration. If these blocks are randomly formed, these methods are known as *stochastic gradient* methods.

In Algorithm 14 we provide a pseudocode for the gradient method. In Line 2, the stopping condition for the while-loop is equivalent of testing $\nabla f(\overline{x}) = 0$ for a tolerance ϵ.

Algorithm 14 Gradient method

initialise. tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$.
while $||\nabla f(x_k)|| > \epsilon$ **do**
  $d = -\frac{\nabla f(x_k)}{||\nabla f(\overline{x})||}$
  $\overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
  $x_{k+1} = x_k + \overline{\lambda} d_j$
  $k = k + 1$
end while
return x_k.

Figure 16.6 presents the progress of the gradient method using exact (bisection) and inexact (Armijo rule with $\alpha = 0.1$ and $\beta = 0.7$) line searches. As can be expected, when an inexact line search is employed, the method overshoots slightly some of the steps, taking a few more iterations to converge.

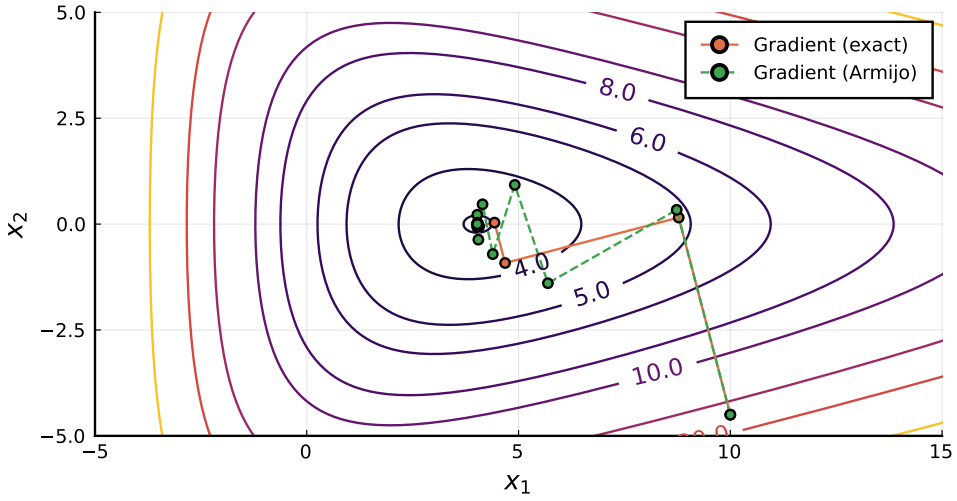


Figure 16.6: Gradient method applied to f. Convergence is observed in 10 steps using exact line search and 19 using Armijo's rule (for $\epsilon = 10^{-5}$)

16.3.3 Newton's method

One can think of gradient methods as using first-order information to derive directions of improvement, while *Newton's method* consists of a step forward also incorporating second-order information. This can be shown to produce better convergence properties, but at the expense of the extra computational burden incurred by calculating and manipulating Hessian matrices.

The idea of the Newton's method is the following. Consider the second-order approximation of f at x_k, which is given by

$$q(x) = f(x_k) + \nabla f(x_k)^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k)$$

The method uses as direction d that of the extremum of the quadratic approximation at x_k, which can be obtained from the first-order condition $\nabla q(x) = 0$. This renders

$$\nabla q(x) = \nabla f(x_k) + H(x_k)(x - x_k) = 0. \tag{16.2}$$

Assuming that $H^{-1}(x_k)$ exists, we can use (16.2) to obtain the following update rule, which is known as the *Newton step*

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k) \tag{16.3}$$

Notice that the "pure" Newton's method has embedded in the direction of the step, its length (i.e., the step size) as well. In practice, the method uses $d = -H^{-1}(x_k)\nabla f(x_k)$ as a direction combined with a line search to obtain optimal step sizes and prevent divergence (that is, converge to $-\infty$) in cases where the second-order approximation might lead to divergence. Fixing $\lambda = 1$ renders the natural Newton's method, as derived in (16.3). The Newton's method can also be seen as employing Newton-Raphson method to solve the system of equations that describe the first order conditions of the quadratic approximation at x_k.

Figure 16.7 shows the calculation of direction $d = -H^{-1}(x_k)\nabla f(x_k)$ for the first iteration of the Newton's method. Notice that the direction is the same as the that of the minimum of the quadratic approximation $q(x)$ at x_k. The employment of a line search allows for overshooting the exact minimum, making the search more efficient.

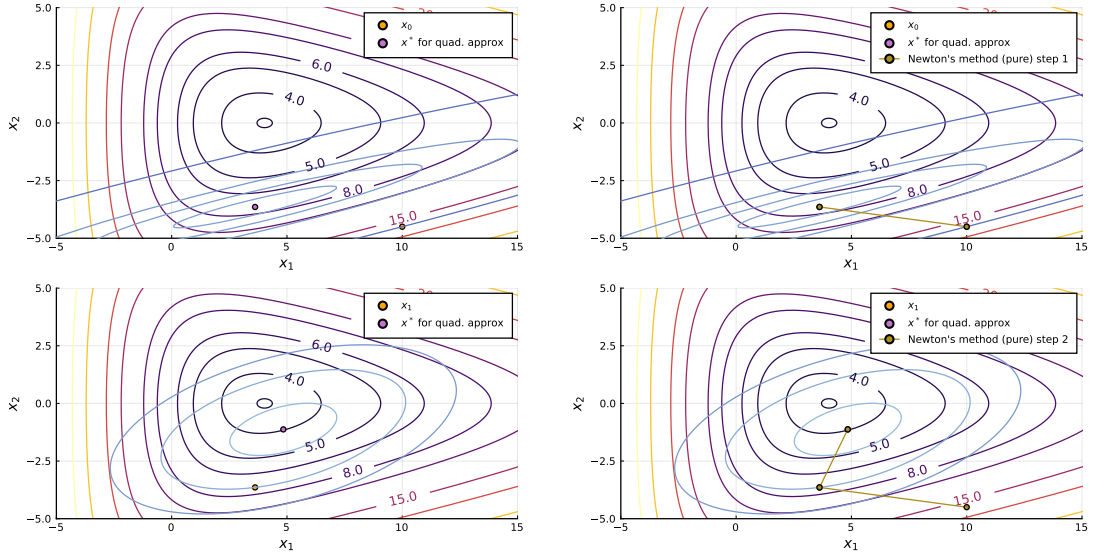


Figure 16.7: The calculation of the direction $d = x^* - x_0$ in the first two iterations of the Newton's method with step size λ fixed to 1 (the pure Newton's method, in left to right, top to bottom order). Notice in blue the level curves of the quadratic approximation of the function at the current point x_k and how it improves from one iteration to the next.

The Newton's method might diverge if the initial point is too far from the optimal and fixed step sizes (such as $\lambda = 1$) are used, since the quadratic approximation minimum and the actual function minimum can become drastically and increasingly disparate. Levenberg-Marquardt method and other trust-region-based variants address convergence issues of the Newton's method. As a general

rule, combining the method with an exact line search of a criteria for step-size acceptance that require improvement (such as employing the Armijo rule for defining the step sizes) if often sufficient for guaranteed convergence. Figure 16.8 compares the convergence of the pure Newton's method and the method employing an exact line search.

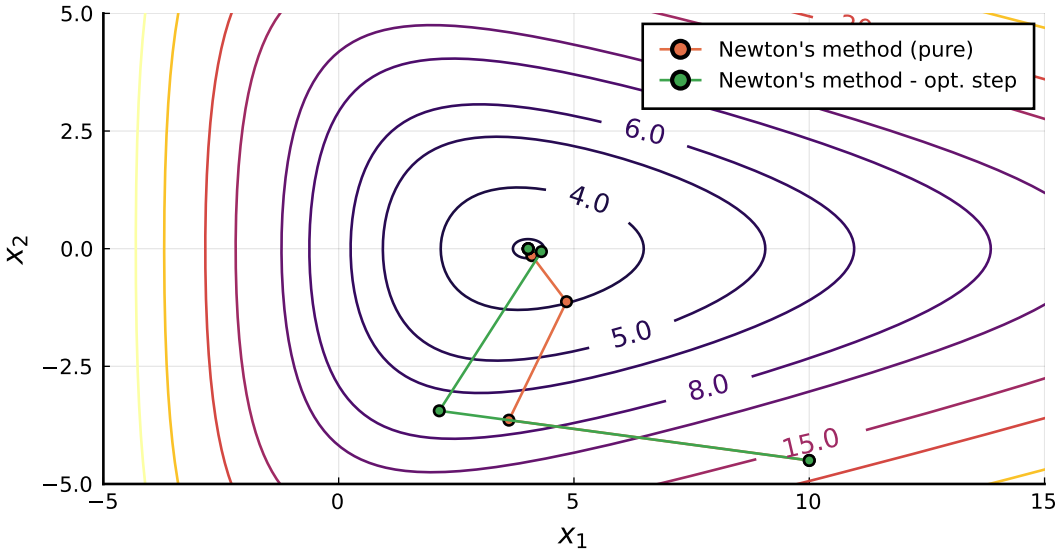


Figure 16.8: A comparison of the trajectory of both Newton's method variants. Notice that in the method using the exact line search, wile the direction $d = x^* - x_0$ is utilised, the step size is larger in the first iteration.

Algorithm 15 presents a pseudocode for the Newton's method. Notice that in Line 3, an inversion operation is required. One might be cautious about this operation, since as $\nabla f(x_k)$ tends to zero, the Hessian $H(x_k)$ tends to become singular, potentially causing numerical instabilities.

Algorithm 15 Newton's method

1: **initialise.** tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$.
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $\quad d = -H^{-1}(x_k)\nabla f(x_k)$
4: $\quad \overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
5: $\quad x_{k+1} = x_k + \overline{\lambda} d$
6: $\quad k = k + 1$
7: **end while**
8: **return** x_k

Figure 16.9 shows the progression of the Newton's method for f with exact and inexact line searches.

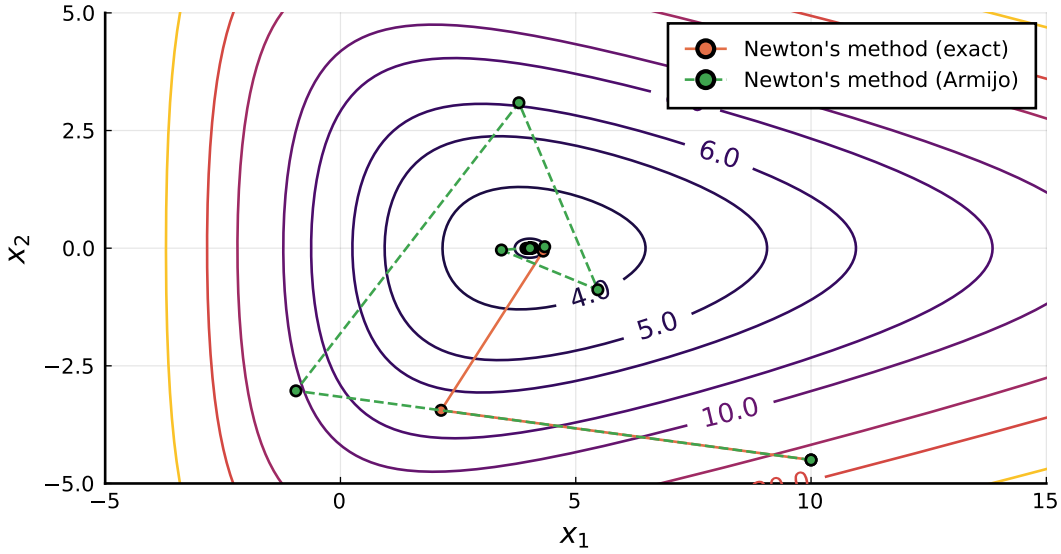


Figure 16.9: Newton's method applied to f. Convergence is observed in 4 steps using exact line search and 27 using Armijo's rule ($\epsilon = 10^{-5}$)

Chapter 17

Unconstrained optimisation methods: part 2

17.1 Unconstrained optimisation methods

We will now discuss to variants of the gradient and Newton methods that try to exploit the computational simplicity of gradient methods while encoding of curvature information as the Newton's method, but without explicitly relying on second-order derivatives (i.e., Hessian matrices).

17.1.1 Conjugate gradient method

The conjugate gradient method use the notion of *conjugacy* to guide the search for optimal solutions. The original motivation for the method comes from quadratic problems, in which one can use conjugacy to separate the search for the optimum of $f : \mathbb{R}^n \mapsto \mathbb{R}$ into n exact steps.

The concept of conjugacy

Let us first define the concept of conjugacy.

Definition 17.1. *Let H be an $n \times n$ symmetric matrix. The vectors $d_1, \ldots, d_n$ are called (H-)conjugate if they are linearly independent and $d_i^\top H d_j = 0$, for all $i, j = 1, \ldots, n$ such that $i \neq j$.*

Notice that H-conjugacy (or simply conjugacy) is a generalisation of orthogonality under the linear transformation imposed by the matrix H. Notice that orthogonal vectors are H-conjugate for $H = I$. Figure 17.1 illustrate the notion of conjugacy between two vectors d_1 and d_2 that are H-conjugate, being H the Hessian of the underlying quadratic function. Notice how it allows one to generate, from direction d_1, a direction d_2 that, if used in combination with an exact line search, would take us to the centre of the curve.

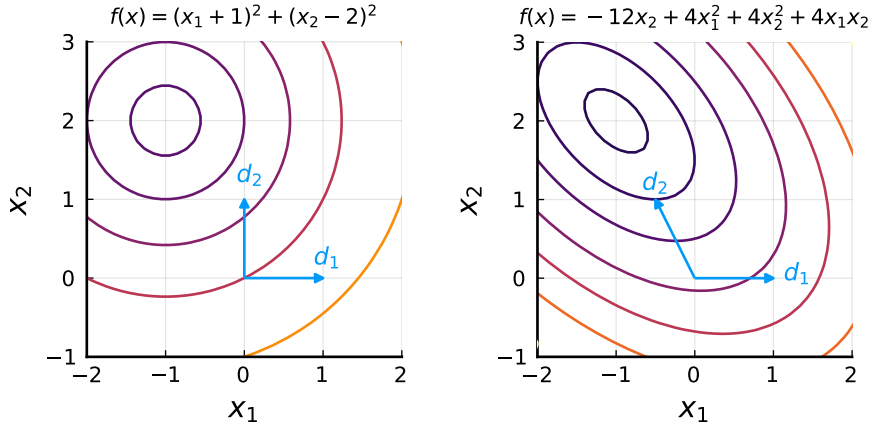


Figure 17.1: d_1 and d_2 are H-conjugates; on the left, $H = I$.

One can use H-conjugate directions to find optimal solutions for the quadratic function $f(x) = c^\top x + \frac{1}{2}x^\top Hx$, where H is a symmetric matrix. Suppose we know directions $d_1, \ldots, d_n$ that are H-conjugate. Then, given an initial point x_0, any point x can be described as $x = x_0 + \sum_{j=1}^n \lambda_j d_j$. We can then reformulate $f(x)$ as a function of the step size λ, i.e.,

$$\begin{aligned} f(x) = F(\lambda) &= c^\top(x_0 + \sum_{j=1}^n \lambda_j d_j) + \frac{1}{2}(x_0 + \sum_{j=1}^n \lambda_j d_j)^\top H(x_0 + \sum_{j=1}^n \lambda_j d_j) \\ &= \sum_{j=1}^n \left[c^\top(x_0 + \lambda_j d_j) + \frac{1}{2}(x_0 + \lambda_j d_j)^\top H(x_0 + \lambda_j d_j)\right]. \end{aligned}$$

This reformulation exposes an important properties that having conjugate directions $d_1, \ldots, d_n$ allows us to explore: separability. Notice that $F(\lambda) = \sum_{j=1}^n F_j(\lambda_j)$, where $F_j(\lambda_j)$ is given by

$$F_j(\lambda_j) = c^\top(x_0 + \lambda_j d_j) + \frac{1}{2}(x_0 + \lambda_j d_j)^\top H(x_0 + \lambda_j d_j),$$

and is, ultimately, a consequence of the linear independence of the conjugate directions. Assuming that H is positive definite, and thus that first-order conditions are necessary and sufficient for optimality, we can then calculate optimal $\overline{\lambda}_j$ for $j = 1, \ldots, n$ as

$$\begin{aligned} F_j'(\lambda_j) &= 0 \\ c^\top d_j + x_0^\top H d_j + \lambda_j d_j^\top H d_j &= 0 \\ \overline{\lambda}_j &= -\frac{c^\top d_j + x_0^\top H d_j}{d_j^\top H d_j}, \text{ for all } j = 1, \ldots, n. \end{aligned}$$

This result can be used to devise an iterative method that can obtain optimal solution for quadratic functions in exactly n iterations. From an initial point x_0 and a collection of H-conjugate directions $d_1, \ldots, d_n$, the method consists of the successively executing the following step

$$x_k = x_{k-1} + \lambda_k d_k, \text{ where } \lambda_k = -\frac{c^\top d_k + x_{k-1}^\top H d_k}{d_k^\top H d_k}$$

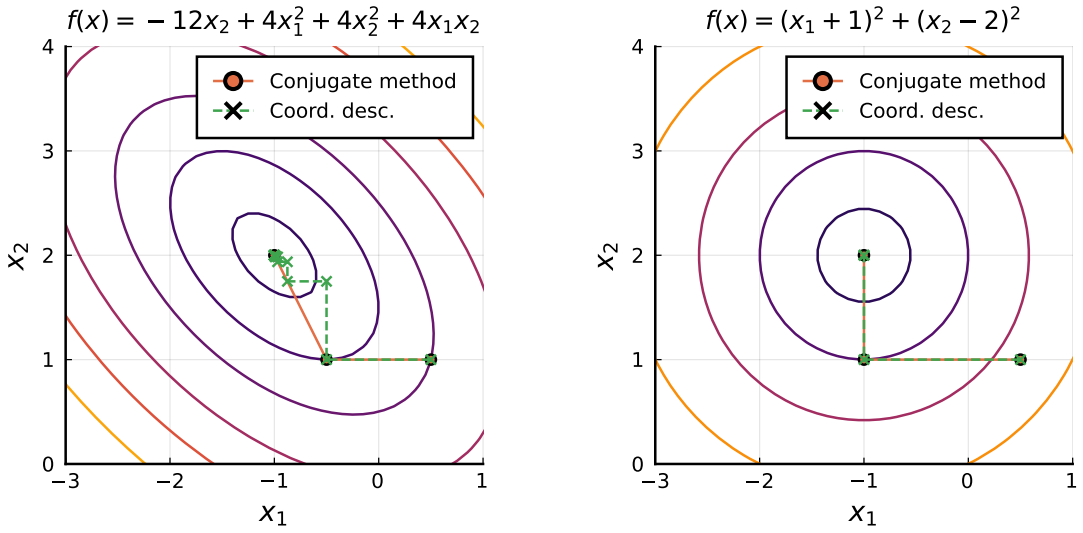


Figure 17.2: Optimising f with the conjugate method and coordinate descent (left). For $H = I$, both methods coincide (right)

Notice the resemblance this method hold with the coordinate descent method. In case $H = I$, then the coordinate directions given by $d_i = 1$ and $d_{j\neq i} = 0$ are H-conjugate and thus, the coordinate descent method converges in two iterations. Figure 17.2 illustrates this behaviour. Notice that, on the left, the conjugate method converges in exactly two iterations, while coordinate descent takes several steps before finding the minimum. On the right, both methods become equivalent, since, when $H = I$, the coordinate directions become also conjugate to each other.

Generating conjugate directions

The missing part at this point is how one can generate H-conjugate directions. This can be done efficiently using an adaptation of the *Gram-Schmidt* procedure, typically employed to generate orthonormal bases.

We intend to build a collection of conjugate directions $d_0, \ldots, d_{n-1}$, which can be achieved provided that we have a collection of linearly independent vectors $\xi_0, \ldots, \xi_{n-1}$.

The method proceed as follows.

1. First, set $d_0 = \xi_0$ as a starting step.

2. At a given iteration $k+1$, we need to set the coefficients α^i_{k+1} such that d_{k+1} is H-conjugate to $d_0, \ldots, d_k$ and formed by adding ξ_{k+1} to a linear combination of $d_0, \ldots, d_k$, that is

$$d_{k+1} = \xi_{k+1} + \sum_{l=0}^{k} \alpha^l_{k+1} d_l.$$

3. To obtain H-conjugacy one must observe that, for each $i = 0, \ldots, k$,

$$d_{k+1}{}^\top H d_i = \xi_{k+1}{}^\top H d_i + \left(\sum_{l=0}^{k} \alpha_{k+1}^l d_l \right)^\top H d_i = 0.$$

Due to the H-conjugacy, $d_l^\top H d_k = 0$ for all $l \neq k$. Thus the value of α_{k+1} is

$$\alpha_{k+1}^i = \frac{-\xi_{k+1}^\top H d_i}{d_i^\top H d_i}, \text{ for } i = 0, \ldots, k. \tag{17.1}$$

Gradients and conjugate directions

The next piece required for developing a method that could exploit conjugacy is the definition of what collection of linearly independent vectors $\xi_0, \ldots, \xi_{n-1}$ could be used to generate conjugate directions. In the setting of developing an unconstrained optimisation method, the gradients $\nabla f(x_k)$ can play this part, which is the key result in Theorem 17.2.

Theorem 17.2. *Let $f(x) = c^\top x + \frac{1}{2} x^\top H x$, where H is an $n \times n$ symmetric matrix. Let $d_1, \ldots, d_n$ be H-conjugate, and let x_0 be an arbitrary starting point. Let λ_j be the optimal solution to $F_j(\lambda_j) = f(x_0 + \lambda_j d_j)$ for all $j = 1, \ldots, n$. Then, for $k = 1, \ldots, n$ we must have:*

1. *x_{k+1} is optimal to min. $\{f(x) : x - x_0 \in L(d_1, \ldots, d_k)\}$ where $L(d_1, \ldots, d_k) = \left\{ \sum_{j=1}^{k} \mu_j d_j : \mu_j \in \mathbb{R}, j = 1, \ldots, k \right\}$;*
2. *$\nabla f(x_{k+1})^\top d_j = 0$, for all $j = 1, \ldots, k$;*

The proof of this theorem is based on the idea that, for a given collection of conjugate directions $d_0, \ldots, d_k$, x_k will be optimal in the space spanned by the conjugate directions $d_0, \ldots, d_k$, meaning that the partial derivatives of $F(\lambda)$ for these directions is zero. This phenomena is sometimes called *the expanding manifold property*, since at each iteration $L(d_0, \ldots, d_k)$ expands in one independent (conjugate) direction at the time. To verify the second point, notice that the optimality condition for $\lambda_j \in \arg\min \{F_j(\lambda_j)\}$ is $d_j^\top \nabla f(x_0 + \lambda d_j) = 0$.

Conjugate gradient method

We have now all parts required for describing the *conjugate gradient method*. The method uses the gradients $\nabla f(x_k)$ as linearly independent vectors to generate conjugate directions, which are then used as search directions d_k.

In specific, the method operates generating a sequence of iterates

$$x_{k+1} = x_k + \lambda_k d_k,$$

where $d_0 = -\nabla f(x_0)$. Given a current iterate x_{k+1} with $-\nabla f(x_{k+1}) \neq 0$, we use Gram-Schmidth procedure, in particular (17.1), to generate a conjugate direction d_{k+1} by making the linearly independent vector $\xi_{k+1} = \nabla f(x_{k+1})$. Thus, we obtain

$$d_{k+1} = -\nabla f(x_{k+1}) + \alpha_k d_k, \text{ with } \alpha_k = \frac{\nabla f(x_{k+1})^\top H d_k}{d_k^\top H d_k}. \tag{17.2}$$

Notice that, since $\nabla f(x_{k+1}) - \nabla f(x_k) = H(x_{k+1} - x_k) = \lambda_k H d_k$ and $d_k = -\nabla f(x_k) + \alpha_{k-1} d_{k-1}$, α_k can be simplified to be

$$\begin{aligned}\alpha_k &= \frac{\nabla f(x_{k+1})^\top H d_k}{d_k^\top H d_k} \\ &= \frac{\nabla f(x_{k+1})^\top (\nabla f(x_{k+1}) - \nabla f(x_k))}{(-\nabla f(x_k) + \alpha_{k-1} d_{k-1})^\top (\nabla f(x_{k+1}) - \nabla f(x_k))} \\ &= \frac{||\nabla f(x_{k+1})||^2}{||\nabla f(x_k)||^2},\end{aligned}$$

where the last relation follows from Theorem 17.2. Algorithm 23 summarises the conjugate gradient method.

Algorithm 16 Conjugate gradient method

1: **initialise.** tolerance $\epsilon > 0$, initial point x_0, direction $d_0 = -\nabla f(x_0)$, $k = 1$
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $y_0 = x_{k-1}$
4: $d_0 = -\nabla f(y_0)$
5: **for** $j = 1, \ldots, n$ **do**
6: $\overline{\lambda}_j = \text{argmin}_{\lambda \geq 0} \{f(y_{j-1} + \lambda d_{j-1})\}$
7: $y_j = y_{j-1} + \overline{\lambda}_j d_{j-1}$
8: $d_j = -\nabla f(y_j) + \alpha_j d_{j-1}$, where $\alpha_j = \frac{||\nabla f(y_j)||^2}{||\nabla f(y_{j-1})||^2}$.
9: **end for**
10: $x_k = y_n$, $k = k + 1$
11: **end while**
12: **return** x_k.

The conjugate gradient method using $\alpha_k = \frac{||\nabla f(x_{k+1})||^2}{||\nabla f(x_k)||^2}$ is due to Fletcher and Reeves. An alternative version of the method uses

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

which is known for having better numerical properties for solving problems that are not quadratic.

Figure 17.3 illustrates the behaviour of the conjugate gradient method when applied to solve $f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$ using both exact and inexact line searches.

If $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a quadratic function, then the method is guaranteed to converge in exactly n iterations. However, the method can be applied to any differentiable function f, in which setting the method behaves as successively solving quadratic approximations of f, in a similar fashion to that of Newton's method, but without requiring second-order (Hessian) information, which is the most demanding aspect associated with Newton's method. When employed to non-quadratic functions, the process of obtaining conjugate directions is restarted at the current point x_k after n steps (represented in the loop staring in Line 5 in Algorithm 23).

Equation (17.2) exposes an important property of the conjugate gradient method. In general, the employment of second-order terms is helpful for the optimisation method because it encodes *curvature information* on the definition of the search direction. The conjugate gradient method is also capable of encoding curvature information, not by using Hessians, but by weighting the current

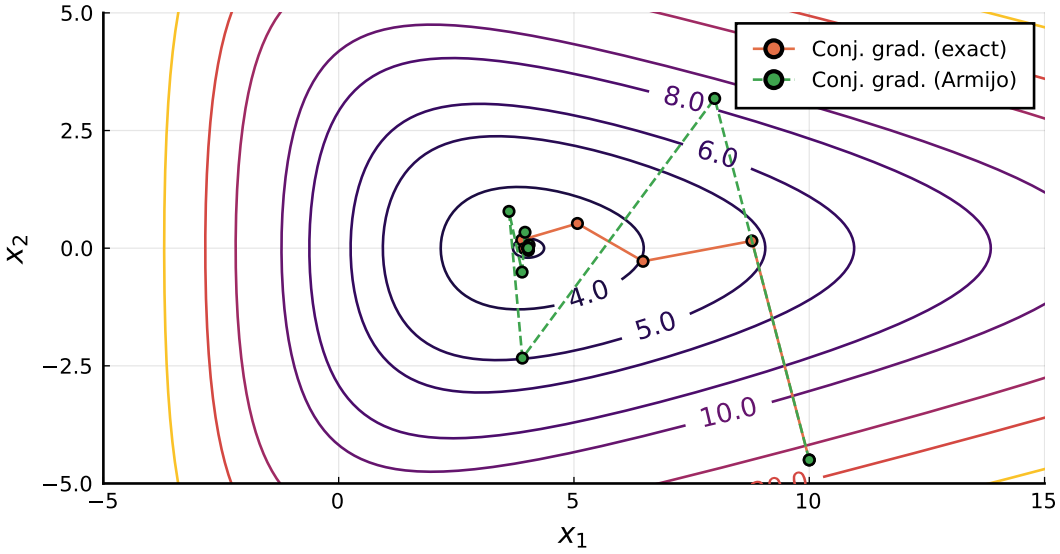


Figure 17.3: Conjugate gradient method applied to f. Convergence is observed in 24 steps using exact line search and 28 using Armijo's rule ($\epsilon = 10^{-6}$)

direction (given by the gradient) $-\nabla f(x_{k+1})$ and the previous direction $\alpha_k d_k$, which naturally compensates for the curvature encoded in the original matrix H (which is the Hessian of the quadratic approximation).

17.1.2 Quasi Newton: BFGS method

Quasi-Newton methods is a term referring to methods that use approximations for the inverse of the Hessian of f at $\overline{x}$, $H^{-1}(\overline{x})$, that do not explicitly require second-order information (i.e., Hessians) neither expensive inversion operations.

In quasi-Newton methods, we consider the search direction $d_k = -D_k \nabla f(x_k)$, where D_k acts as the approximation for the inverse Hessian $H^{-1}(\overline{x})$. To compute D_k, we use local curvature information, in the attempt to approximate second-order derivatives. For that, let us define the terms

$$\begin{aligned} p_k &= \lambda_k d_k = x_{k+1} - x_k \\ q_k &= \nabla f(x_{k+1}) - \nabla f(x_k) = H(x_{k+1} - x_k) = H p_k. \end{aligned}$$

Starting from an initial guess D_0, quasi-Newton methods progress by successively updating $D_{k+1} = D_k + C_k$, with C_k being such that it only uses the information in p_k and q_k and that, after n updates, D_n converges to H^{-1}.

For that to be the case, we require that p_j, $j = 1, \ldots, k$ are eigenvectors of $D_{k+1}H$ with unit eigenvalue, that is

$$D_{k+1} H p_j = p_j, \text{ for } j = 1, \ldots, k. \tag{17.3}$$

This condition guarantees that, at the last iteration, $D_n = H^{-1}$. To see that, first, notice the

following from (17.3).

$$\begin{aligned}
&D_{k+1}Hp_j = p_j,\ j = 1,\dots,k\\
&D_{k+1}q_j = p_j,\ j = 1,\dots,k\\
&D_kq_j + C_kq_j = p_j\ j = 1,\dots,k\\
&p_j = D_kHp_j + C_kq_j = p_j + C_kq_j,\ j = 1,\dots,k-1,
\end{aligned}$$

which implies that $C_kq_j = 0$ for $j = 1,\dots,k-1$.

Now, for $j = k$, we require that

$$\begin{aligned}
&D_{k+1}q_k = p_k\\
&D_kq_k + C_kq_k = p_k\\
&(D_k + C_k)q_k = p_k
\end{aligned}$$

This last condition allows, after n iterations, to recover

$$D_n = [p_0,\dots,p_{n-1}][q_0,\dots,q_{n-1}]^{-1} = H(x_n) \tag{17.4}$$

Condition (17.4) is called the *secant condition* as a reference to the approximation to the second-order derivative. Another way of understanding the role this condition has is by noticing the following.

$$\begin{aligned}
D_{k+1}q_k &= p_k\\
D_{k+1}(\nabla f(x_{k+1}) - \nabla f(x_k)) &= x_{k+1} - x_k\\
\nabla f(x_{k+1}) &= \nabla f(x_k) + D_{k+1}^{-1}(x_{k+1} - x_k),
\end{aligned} \tag{17.5}$$

where D_{k+1}^{-1} can be seen as an approximation to the Hessian H, just as D_{k+1} is an approximation to H^{-1}. Now, consider the second-order approximation of f at x_k

$$q(x) = f(x_k) + \nabla f(x_k)^\top(x - x_k) + \frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k).$$

We can now notice the resemblance the condition (17.5) holds with

$$\nabla q(x) = \nabla f(x_k) + H(x_k)^\top(x - x_k) = 0.$$

In other words, at each iteration, the updates are made such that the optimality conditions in terms of the quadratic expansion remains valid.

The *Davidon-Fletcher-Powell* (DFP) is one classical quasi-Newton method available. It employs updates of the form

$$D_{k+1} = D_k + C^{DFP} = D_k + \frac{p_kp_k^\top}{p_k^\top q_k} - \frac{D_kq_kq_k^\top D_k}{q_k^\top D_kq_k}$$

We can verify that C^{DFP} satisfies conditions (17.3) and (17.4). For that, notice that

(1) $C^{DFP}q_j = C^{DFP}Hp_j$

$= \frac{p_kp_k^\top Hp_j}{p_k^\top q_k} - \frac{D_kq_kp_k^\top HD_kHp_j}{q_k^\top D_kq_k} = 0,$ for $j = 1,\dots,k-1;$

(2) $C^{DFP}q_k = \frac{p_k p_k^\top q_k}{p_k^\top q_k} - \frac{D_k q_k q_k^\top D_k q_k}{q_k^\top D_k q_k} = p_k - D_k q_k.$

The main difference between available quasi-Newton methods is the nature of the matrix C employed in the updates. Over the years, several ideas emerged in terms of generating updates that satisfied the above properties. The most widely used quasi-Newton method is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS), which has been widely shown to have remarkable practical performance. BFGS is part of the Broyden family of updates, given by

$$C^B = C^{DFP} + \phi \frac{\tau_j v_k v_k^\top}{p_k^\top q_k},$$

where $v_k = p_k - \left(\frac{1}{\tau_k}\right) D_k q_k$, $\tau_k = \frac{q_j^\top D_k q_k}{p_k^\top q_k}$, and $\phi \in (0, 1)$. The extra term in the Broyden family of updates is designed to help with mitigating numerical difficulties from near-singular approximations.

It can be shown that all updates from the Broyden family also satisfy the quasi-Newton conditions (17.3) and (17.4). The BFGS update is obtained for $\phi = 1$, which renders

$$C_k^{BFGS} = \frac{p_k p_k^\top}{p_k^\top q_k}\left(1 + \frac{q_k^\top D_k q_k}{p_k^\top q_k}\right) - \frac{D_k q_k p_k^\top + p_k q_k^\top D_k}{p_k^\top q_k}.$$

The BFGS method is often presented explicitly approximating the Hessian H instead of its inverse, which is useful when using specialised linear algebra packages that rely on the "backslash" operator to solve linear systems of equations. Let B_k be the current approximation of H. Then $D_{k+1} = B_{k+1}^{-1} = (B_k + \overline{C}_k^{BFGS})^{-1}$, with

$$\overline{C}_k^{BFGS} = \frac{q_k q_k^\top}{q_k^\top p_k} - \frac{B_k p_k p_k^\top B_k}{p_k^\top B_k p_k}.$$

The update for the inverse Hessian H^{-1} can then be obtained using the *Sherman-Morrison formula*.

Figure 17.4 illustrates the behaviour of the BFGS method when applied to solve

$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$

using both exact and inexact line searches. Notice how the combination of imprecisions both in the calculation of H^{-1} and in the line search turns the search noisy. This combination (BFGS combined with Armijo rule) is, however, widely used in efficient implementations of several nonlinear optimisation methods.

A variant of BFGS, called the *limited memory* BFGS (l-BFGS) utilises efficient implementations that do not require storing the whole approximation for the Hessian, but only a few most recent p_k and q_k vectors.

17.2 Complexity, convergence and conditioning

Several aspects must be considered when analysing the performance of algorithms under a given setting and, in each, a multitude of theoretical results that can be used to understand, even if to some extent, the performance of a given optimisation method.

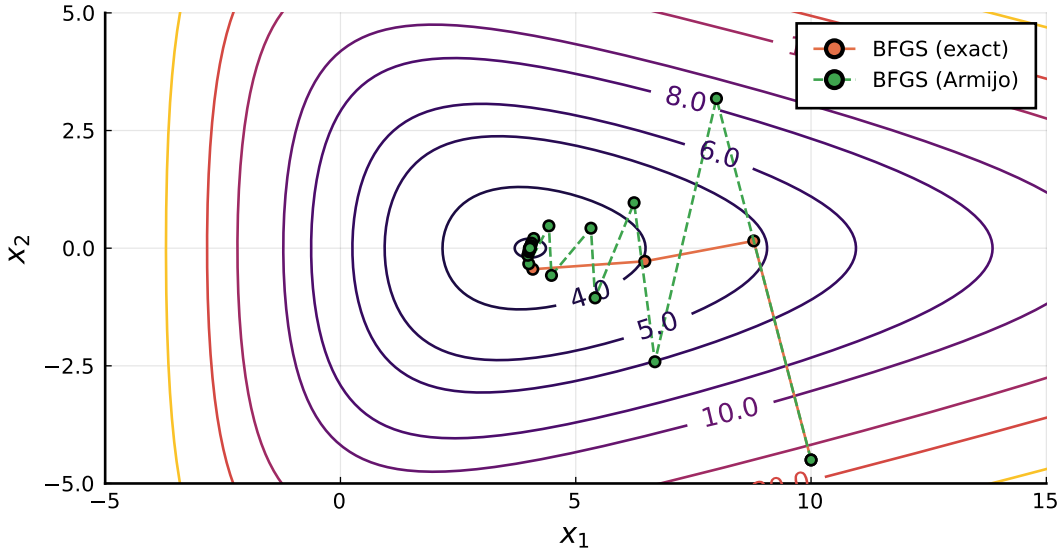


Figure 17.4: BFGS method applied to f. Convergence is observed in 11 steps using exact line search and 36 using Armijo's rule ($\epsilon = 10^{-6}$)

We focus on three key properties that one should be aware when employing the methods we have seen to solve optimisation problems. The first two, *complexity* and *convergence* refer to the algorithm itself, but often involve considerations related to the function being optimised. *Conditioning*, on the other hand, is a characteristic exclusively related to the problem at hand. Knowing how the "three C's" can influence the performance of an optimisation problem is central in making good choices in terms of which optimisation method to employ.

17.2.1 Complexity

Algorithm complexity analysis is a discipline from computer science that focus on deriving worst-case guarantees in terms of the number of computational steps required for an algorithm to converge, given an input of known size. For that, we use the following definition to identify efficient, generally referred to as *polynomial*, algorithms.

Definition 17.3 (Polynomial algorithms)**.** *Given a problem P, a problem instance $X \in P$ with length $L(X)$ in binary representation, and an algorithm A that solves X, let $f_A(X)$ be the number of* elementary calculations *required to run A on X. Then, the running time of A on X is proportional to*

$$f_A^*(n) = \sup_X \{f_A(X) : L(X) = n\}.$$

Algorithm A is polynomial *for a problem P if $f_A^*(n) = O(n^p)$ for some integer p.*

Notice that this sort of analysis only render bounds on the worst-case performance. Though it can be informative under a general setting, there are several well known examples in that experimental practice does not correlate with the complexity analysis. One famous example is the simplex method for linear optimisation problems, which despite not being a polynomial algorithm, presents widely-demonstrated reliable (polynomial-like) performance.

17.2.2 Convergence

In the context of optimisation, *local analysis* is typically more informative regarding to the behaviour of optimisation methods. This analysis tend to disregard initial steps further from the initial points and concentrate on the behaviour of the sequence $\{x_k\}$ to a unique point $\overline{x}$.

The convergence is analysed by means of *rates of convergence* associated with *error functions* $e : \mathbb{R}^n \mapsto \mathbb{R}$ such that $e(x) \geq 0$. Typical choices for e include:

- $e(x) = ||x - \overline{x}||$;
- $e(x) = |f(x) - f(\overline{x})|$.

The sequence $\{e(x_k)\}$ is then compared to the geometric progression β^k, with $k = 1, 2, \ldots$, and $\beta \in (0, 1)$. We say that a method presents *linear convergence* if exists $q > 0$ and $\beta \in (0, 1)$ such that $e(x) \leq q\beta^k$ for all k. An alternative way of posing this result is stating that

$$\lim_{k\to\infty} \sup \frac{e(x_{k+1})}{e(x_k)} \leq \beta.$$

We say that an optimisation method converges superlinearly if the rate of convergence tends to zero. That is, if exists $\beta \in (0, 1)$, $q > 0$ and $p > 1$ such that $e(x_k) \leq q\beta^{p^k}$ for all k. For $k = 2$, we say that the method presents quadratic convergence. Any p-order convergence is obtained if

$$\lim_{k\to\infty} \sup \frac{e(x_{k+1})}{e(x_k)^p} < \infty, \text{ which is true if } \lim_{k\to\infty} \sup \frac{e(x_{k+1})}{e(x_k)} = 0.$$

Linear convergence is the most typical convergence rate for nonlinear optimisation methods, which is satisfactory if β is not too close to one. Certain methods are capable of achieving superlinear convergence for certain problems, being Newton's method an important example.

In light of what we discussed, let us analyse the convergence rate of some of the methods earlier discussed. We start by posing the convergence of gradient methods.

Theorem 17.4 (Convergence of the gradient method)**.** *Let $f(x) = \frac{1}{2}x^\top Hx$ where H is a positive definite symmetric matrix. Suppose $f(x)$ is minimised with the gradient method using an exact line search. Let $\underline{\lambda} = \min_{i=1,\ldots,n} \lambda_i$ and $\overline{\lambda} = \max_{i=1,\ldots,n} \lambda_i$, where λ_i are eigenvalues of H. Then, for all k,*

$$\frac{f(x_{k+1})}{f(x_k)} \leq \left(\frac{\overline{\lambda} - \underline{\lambda}}{\overline{\lambda} + \underline{\lambda}}\right)^2$$

Theorem 17.4 implies that, under certain assumptions, the gradient methods present *linear convergence*. Moreover, this result shows that the convergence rate is *dependent* on the scaling of the function, since it depends on the ratio of eigenvalues of H, which in turn can be modified by scaling f. This results exposes an important shortcoming that gradient methods present: the dependence on the *conditioning* of the problem, which we will discuss shortly. Moreover, this result can be extended to incorporate functions other than quadratic and also inexact line searches.

The convergence of Newton's method is also of interest since, under specific circumstances, it presents a quadratic convergence rate. Theorem 17.5 summarises these conditions.

Theorem 17.5 (Convergence of Newton's method - general case)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}^n$ be differentiable, $\overline{x}$ such that $g(\overline{x}) = 0$, and let $\{e(x_k)\} = \{||x_k - \overline{x}||\}$. Moreover, let $N_\delta(\overline{x}) = \{x : ||x - \overline{x}|| \leq \delta\}$ for some $\delta > 0$. Then*

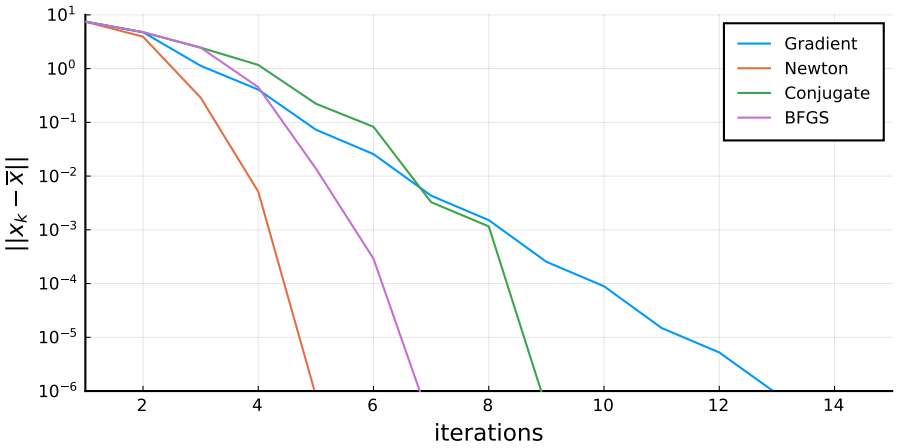


Figure 17.5: Convergence comparison for the four methods

1. *There exists* $\delta > 0$ *such that if* $x_0 \in N_\delta(\overline{x})$*, the sequence* $\{x_k\}$ *with* $x_{k+1} = x_k - (\nabla g(x_k)^\top)^{-1} g(x_k)$ *belongs to* $N_\delta(\overline{x})$ *and converges to* $\overline{x}$*, while* $\{e(x_k)\}$ *converges superlinearly.*

2. *If for some* $L > 0$*,* $M > 0$*, and for all* $x, y \in N_\delta(\overline{x})$*,* $\lambda \in (0, \delta]$

$$||\nabla g(x) - \nabla g(y)|| \leq L||x - y|| \quad \textit{and} \quad ||(\nabla g(x_k)^\top)^{-1}|| \leq M,$$

then, if $x_0 \in N_\delta(\overline{x})$*, we have for* $k = 0, 1, \ldots$

$$||x_{k+1} - \overline{x}|| \leq \frac{LM}{2}||x_k - \overline{x}||^2.$$

If $\frac{LM\delta}{2} < 1$ *and* $x_0 \in N_\delta(\overline{x})$*,* $\{e(x_k)\}$ *converges quadratically.*

Notice that the convergence of the method is analysed in two distinct phases. In the first phase, referred to as 'damped' phase, superlinear convergence is observed within the neighbourhood $N_\delta(\overline{x})$ defined by δ. The second phase is where quadratic convergence is observed and it happens when $\delta < \frac{2}{LM}$, which in practice can only be interpreted as small enough, as the constants L (the Lipschitz constant) and M (a finite bound for the norm of the Hessian) cannot be easily estimated in practical applications.

However, it is interesting to notice that the convergence result for Newton's method do not depend on the scaling of the problem, like the gradient method. This property, called *affine invariance* is one of the greatest features that Newton's method possess.

Figure 17.5 compare the convergence of four methods presented considering $f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$, employing exact line search and using $e(x) = ||x_k - \overline{x}||$. Notice how the quadratic convergence of Newton's method compare with the linear convergence of the gradients method. The other two, conjugate gradients and BFGS, present superlinear convergence.

17.2.3 Conditioning

The *condition number* of a symmetric matrix is given by

$$\kappa = ||A||_2||A^{-1}||_2 = \frac{\max_{i=1,\ldots,n}\{\lambda_i\}}{\min_{i=1,\ldots,n}\{\lambda_i\}} = \frac{\overline{\lambda}}{\underline{\lambda}}$$

The condition number κ is an important measure in optimisation, since it can be used to predict how badly scaled a problem might be. Large κ values mean that numerical errors will be amplified after repeated iterations, in particular matrix inversions.

Roughly speaking, having $\kappa \geq 10^k$ means that at each iteration, k digits of accuracy are lost. As general rule, one would prefer smaller κ numbers, but good values are entirely problem dependent.

One way of understanding the role that the conditioning number κ has is to think the role that the eigenvalues of the Hessian have in the shape of the level curves of quadratic approximations of a general function $f : \mathbb{R}^n \mapsto \mathbb{R}$. First, let us consider the Hessian $H(x)$ at a given point $x \in \mathbb{R}^n$ is the identity matrix I, for which all eigenvalues are 1 and eigenvectors are e_i, $i = 1, \ldots, n$, where e_i is the vector with component 1 in the position i and zero everywhere else. This means that in the direction of the n-eigenvectors, the ellipsoid formed by the level curves (specifically, the lower level sets) of f stretch by the same magnitude and, therefore, the level curves of the quadratic approximation are in fact a circle. Now, suppose that for one of the dimensions i of the matrix $H(x)$, we have one of the eigenvalues greater than 1. What we would see is that the level curves of the quadratic approximation will be more stretched in that dimension i than in the others. The reason for that is because the Hessian plays a role akin to that of a characteristic matrix in an ellipsoid (specifically due to the second order term $\frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k)$ in the quadratic approximation).

Thus, larger κ will mean that the ratio between the eigenvalues is larger, which in turn implies that there is eccentricity in the lower level sets (i.e., the lower level sets are far wider in one direction than in others), which ultimately implies that first-order methods struggle since often the gradients often point to directions that only show descent for small step sizes.

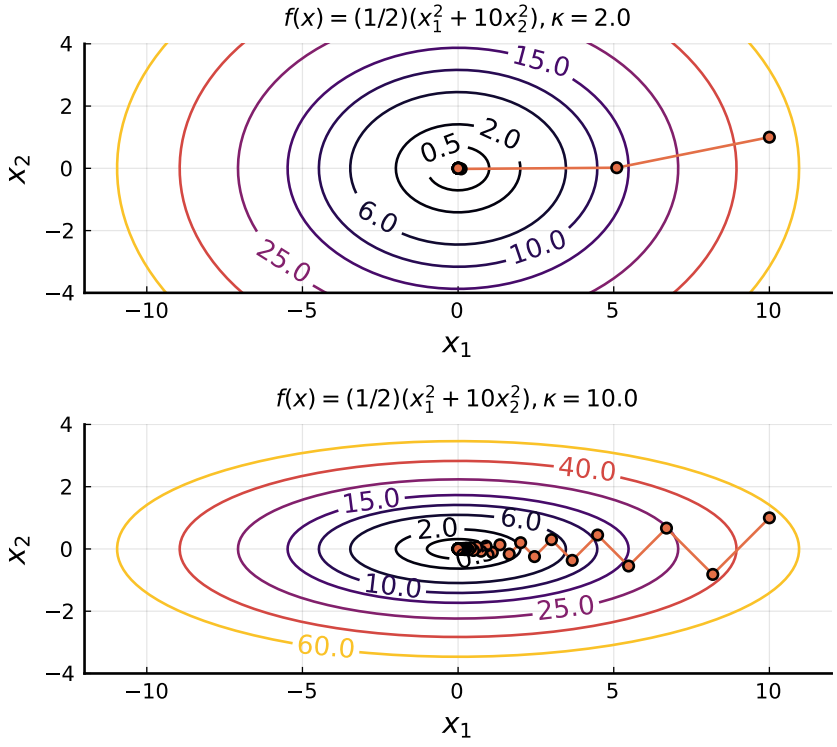


Figure 17.6: The gradient method with exact line search for different κ.

Figure 17.6 illustrates the effect of different condition numbers on the performance of the gradient method. As can be seen, the method require more iterations for higher conditioning numbers, in accordance to the convergence result presented in Theorem 17.4.

Chapter 18

Constrained optimality conditions

18.1 Optimality for constrained problems

We now investigate how to derive optimality conditions for the problem

$$(P): \quad \text{min. } \{f(x) : x \in S\}.$$

In particular, we are interested in understanding the role that the feasibility set S has on the optimality conditions of constrained optimisation problems in the form of P. Let us first define two geometric elements that we will use to derive the optimality conditions for P.

Definition 18.1 (cone of feasible directions)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty set, and let $\overline{x} \in \mathbf{clo}(S)$. The* cone of feasible directions *D at $\overline{x} \in S$ is given by*

$$D = \{d : d \neq 0, \text{ and } \overline{x} + \lambda d \in S \text{ for all } \lambda \in (0, \delta) \text{ for some } \delta > 0\}.$$

Definition 18.2 (cone of descent directions)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty set, $f : \mathbb{R}^n \to \mathbb{R}$, and $\overline{x} \in \mathbf{clo}(S)$. The* cone of improving (i.e., descent) directions *F at $\overline{x} \in S$ is*

$$F = \{d : f(\overline{x} + \lambda d) < f(\overline{x}) \text{ for all } \lambda \in (0, \delta) \text{ for some } \delta > 0\}.$$

These cones are geometrical descriptions of the regions that, from a given point $\overline{x}$, one can obtain feasible (D) and improving (F) solutions. This is useful in that it allows to express the optimality conditions for $\overline{x}$ as observing that $F \cap D = \emptyset$ holds. In other words, $\overline{x}$ is optimal if there exists no feasible direction that can provide improvement in the objective function value.

Although having a geometrical representation of such sets can be useful in solidifying the conditions for which a feasible solution is also optimal, we need to derive an *algebraic* representation of such sets that can be used in computations. To reach that objective, let us start by defining an algebraic representation for F. For that, let us assume that $f : S \subset \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable. Recall that d is a descent direction at $\overline{x}$ if $\nabla f(\overline{x})^\top d < 0$. Thus, we can define the set F_0

$$F_0 = \left\{d : \nabla f(\overline{x})^\top d < 0\right\}$$

as an algebraic representation for F. Notice that F_0 is an open half-space formed by the hyperplane with normal $\nabla f(\overline{x})$. Figure 18.1 illustrates the condition $F_0 \cap D = \emptyset$. Theorem 18.3 establishes that the condition $F_0 \cap D = \emptyset$ is necessary for optimality in constrained optimisation problems.

Theorem 18.3 (geometric necessary condition)**.** *Let $S \subseteq \mathbb{R}^n$ be a nonempty set, and let $f : S \to \mathbb{R}$ be differentiable at $\overline{x} \in S$. If $\overline{x}$ is a local optimal solution to*

$$(P): \quad \textit{min. } \{f(x) : x \in S\},$$

then $F_0 \cap D = \emptyset$, where $F_0 = \left\{d : \nabla f(\overline{x})^\top d < 0\right\}$ and D is the cone of feasible directions.

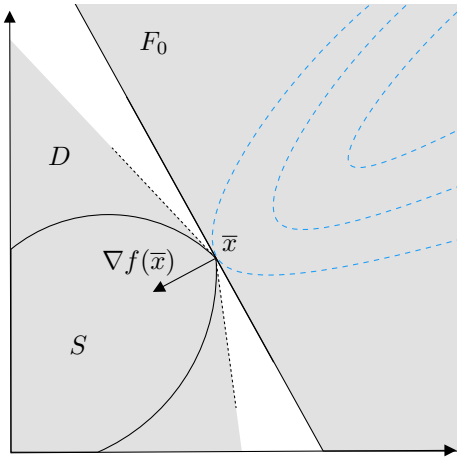


Figure 18.1: Illustration of the cones F_0 and D for the optimal point $\overline{x}$. Notice that D is an open set.

The proof for this theorem consists of using the separation theorem to show that $F_0 \cap D = \emptyset$ implies that the first-order optimality condition $\nabla f(\overline{x})^\top d \geq 0$ holds.

As discussed earlier (in Lecture 4), in the presence of convex, this conditions becomes sufficient for optimality. Moreover, if f is strictly convex, then $F = F_0$. If f is linear, it might be worth considering $F_0' = \left\{d \neq 0 : \nabla f(\overline{x})^\top d \leq 0\right\}$ to allow for considering orthogonal directions.

18.1.1 Inequality constrained problems

In mathematical programming applications, the feasibility set S is typically expressed by a set of inequalities. Let us redefine P as

$$\begin{aligned}(P): \quad \text{min.} \quad & f(x) \\ \text{s.t.:} \quad & g_i(x) \leq 0, \ i = 1, \ldots, m \\ & x \in X,\end{aligned}$$

where $g_i : \mathbb{R}^n \mapsto \mathbb{R}$, are differentiable functions for $i = 1, \ldots, m$ and $X \subset \mathbb{R}$ is an nonempty open set. The differentiability of g_i, $i = 1, \ldots, m$, allows for the definition of a proxy for D using the gradients of the binding constraints $i \in I = \{i : g_i(\overline{x}) = 0\}$ at $\overline{x}$. This set, denoted by G_0, is defined as

$$G_0 = \left\{d : \nabla g_i(\overline{x})^\top d < 0, i \in I\right\}.$$

The use of G_0 is a convenient algebraic representation, since it can be shown that $G_0 \subseteq D$, which is stated in Lemma 18.4. As $F_0 \cap D = \emptyset$ must hold for a local optimal solution $\overline{x} \in S$, it follows that $F_0 \cap G_0 = \emptyset$ must also hold.

Lemma 18.4. *Let $S = \{x \in X : g_i(x) \leq 0$ for all $i = 1, \ldots, m\}$, where $X \subset \mathbb{R}^n$ is a nonempty open set and $g_i : \mathbb{R}^n \to \mathbb{R}$ a differentiable function for all $i = 1, \ldots, m$. For a feasible point $\overline{x} \in S$, let $I = \{i : g_i(\overline{x}) = 0\}$ be the index set of the binding (or active) constraints. Let*

$$G_0 = \left\{d : \nabla g_i(\overline{x})^\top d < 0, i \in I\right\}$$

Then $G_0 \subseteq D$, where D is the cone of feasible directions.

In settings in which g_i is affine for some $i \in I$, it might be worth considering $G_0' = \left\{d \neq 0 : \nabla g_i(\overline{x})^\top d \leq 0, i \in I\right\}$ so that orthogonal feasible directions can also be represented. Notice that in this case $D \subseteq G_0'$.

18.2 Fritz-John conditions

The Fritz-John conditions are the algebraic conditions that must be met for $F_0 \cap G_0 = \emptyset$ to hold. These algebraic conditions are convenient as they only involve the gradients of the binding constraints and they can be verified computationally.

Theorem 18.5 (Fritz-John necessary conditions)**.** *Let $X \subseteq \mathbb{R}^n$ be a nonempty open set, and let $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}$ be differentiable for all $i = 1, \ldots, m$. Additionally, let $\overline{x}$ be feasible and $I = \{i : g_i(\overline{x}) = 0\}$. If $\overline{x}$ solves P locally, there exist scalars u_i, $i \in \{0\} \cup I$, such that*

$$\begin{aligned} &u_0 \nabla f(\overline{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\overline{x}) = 0 \\ &u_i g_i(\overline{x}) = 0, \ i = 1, \ldots, m \\ &u_i \geq 0, \ i = 0, \ldots, m \\ &u = (u_0, \ldots, u_m) \neq 0 \end{aligned}$$

Proof. Since $\overline{x}$ solves P locally, Theorem 18.3 guarantees that there is no d such that $\nabla f(\overline{x})^\top d < 0$ and $\nabla g_i(x)^\top d < 0$ for each $i \in I$. Let A be the matrix whose rows are $\nabla f(\overline{x})^\top$ and $\nabla g_i(\overline{x})^\top$ for $i \in I$.

Using Farkas' theorem, we can show that if $Ad < 0$ is inconsistent, then there exists nonzero $p \geq 0$ such that $A^\top p = 0$. Letting $p = (u_0, u_{i_1}, \ldots, u_{i_{|I|}})$ for $I = \left\{i_1, \ldots, i_{|I|}\right\}$ and making $u_i = 0$ for $i \neq I$, the result follows. □

The proof considers that, if $\overline{x}$ is optimal, then $f(\overline{x})^\top d \geq 0$ holds and a matrix A formed by

$$A = \begin{bmatrix} \nabla f(\overline{x}) \\ \nabla g_{i_1}(\overline{x}) \\ \vdots \\ \nabla g_{i_{|I|}}(\overline{x}) \end{bmatrix}$$

with $I = \left\{i_1, \ldots, i_{|I|}\right\}$, will violate $Ad < 0$. This is used with a variant of Farkas' theorem (known as the Gordan's theorem) to show that the alternative system $A^\top p = 0$, with $p \geq 0$ holds, which, by setting $p = [u_0, u_{i_1}, \ldots, u_{i_{|I|}}]$ and enforcing that the remainder of the gradients $\nabla g_i(\overline{x})$, for $i \notin I$, are removed by setting $u_i = 0$, which leads precisely to the Fritz-John conditions.

The multipliers u_i, for $i = 0, \ldots, m$, are named Lagrangian multipliers due to the connection with Lagrangian duality, as we will see later. Also, notice that for nonbinding constraints ($g_i(\overline{x}) < 0$ for $i \notin I$), u_i must be zero to form the Fritz-John conditions. This condition is named complementary slackness.

The Fritz-John conditions are unfortunately too weak, which is a problematic issue in some rather common settings. A point $\overline{x}$ satisfies the Fritz-John conditions if and only if $F_0 \cap G_0 = \emptyset$, which is trivially satisfied when $G_0 = \emptyset$.

For example, the Fritz-John conditions are trivially satisfied for points where some of the gradient vanishes (i.e., $\nabla f(\overline{x}) = 0$ or $\nabla g_i(\overline{x}) = 0$ for some $i = 1, \ldots, m$). Sets with no relative interior in the immediate vicinity of $\overline{x}$ also satisfy Fritz-John conditions.

An interesting case is for problems with equality constraints, as illustrated in Figure 18.2. In general, if the additional regularity condition that the gradients $\nabla g_i(\overline{x})$ are linearly independent does not hold, $\overline{x}$ trivially satisfies the Fritz-John conditions.

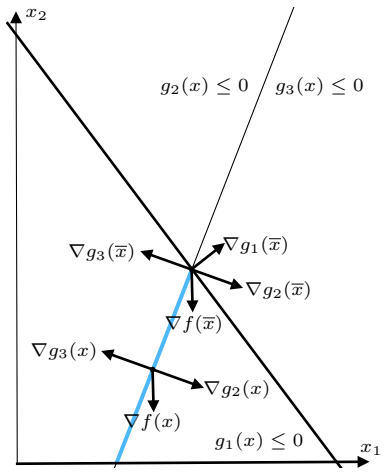


Figure 18.2: All points in the blue segment satisfy FJ conditions, including the minimum $\overline{x}$.

18.3 Karush-Kuhn-Tucker conditions

The Karush-Kuhn-Tucker (KKT) conditions can be understood as the Frizt-John conditions with an extra requirement of regularity for $\overline{x} \in S$. This regularity requirement is called *constraint qualification* and, in a general sense, are meant to prevent the trivial case $G_0 = \emptyset$, making thus the optimality conditions stronger (i.e., more stringent).

This is achieved by making $u_0 = 1$ in Theorem 18.5, which ultimately implies that the gradients $\nabla g_i(\overline{x})$ for $i \in I$ must be linearly independent. This condition is called *linearly independent constraint qualification* (LICQ) and is one of several known constraints qualifications that can be used to guarantee regularity of $\overline{x} \in S$.

Theorem 18.6 establishes the KKT conditions as necessary for local optimality of $\overline{x}$ assuming that LICQ holds. For notational simplicity, let us assume for now that

$$(P): \quad \min. \ \{f(x) : g_i(x) \leq 0, i = 1, \ldots, m, x \in X\}.$$

Theorem 18.6 (Karush-Kuhn-Tucker necessary conditions)**.** *Let $X \subseteq \mathbb{R}^n$ be a nonempty open set, and let $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}$ be differentiable for all $i = 1, \ldots, m$. Additionally, for a feasible $\overline{x}$, let $I = \{i : g_i(\overline{x}) = 0\}$ and suppose that $\nabla g_i(\overline{x})$ are linearly independent for all $i \in I$. If*

$\overline{x}$ solves P locally, there exist scalars u_i for $i \in I$ such that

$$\begin{aligned}
&\nabla f(\overline{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\overline{x}) = 0 \\
&u_i g_i(\overline{x}) = 0, \ i = 1, \ldots, m \\
&u_i \geq 0, \ i = 1, \ldots, m
\end{aligned}$$

Proof. By Theorem 18.5, there exists nonzero $(\hat{u}_i)$ for $i \in \{0\} \cup I$ such that

$$\begin{aligned}
&\hat{u}_0 \nabla f(\overline{x}) + \sum_{i=1}^{m} \hat{u}_i \nabla g_i(\overline{x}) = 0 \\
&\hat{u}_i \geq 0, \ i = 0, \ldots, m
\end{aligned}$$

Note that $\hat{u}_0 > 0$, as the linear independence of $\nabla g_i(\overline{x})$ for all $i \in I$ implies that $\sum_{i=1}^{m} \hat{u}_i \nabla g_i(\overline{x}) \neq 0$. Now, let $u_i = \hat{u}_i / u_0$ for each $i \in I$ and $u_i = 0$ for all $i \notin I$. □

The proof builds upon the Fritz-John conditions, which under the assumption that the gradients of the active constraints $\nabla g_i(\overline{x})$ for $i \in I$ are independent, the multipliers $\hat{u}_i$ can be rescaled so that $u_0 = 1$.

The general conditions including both inequality and equality constraints are posed as follows. Notice that the Lagrange multipliers v_i associated with the equality constraints $h(\overline{x}) = 0$ for $i = 1, \ldots, l$ are irrestricted in sign and the complementary slackness condition is not explicitly stated, since it holds redundantly. These can be obtained by replacing equality constraints $h(x) = 0$ with two equivalent inequalities $h_-(x) \leq 0$ and $-h_+(x) \leq 0$ and writing the conditions in Theorem 18.6. Also, notice that, in the absence of constraints, the KKT conditions reduce to the unconstrained first-order condition $\nabla f(\overline{x}) = 0$.

$$\begin{aligned}
&\nabla f(\overline{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\overline{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\overline{x}) = 0 && \text{(dual feasibility 1)} \\
&u_i g_i(\overline{x}) = 0, \qquad i = 1, \ldots, m && \text{(complementary slackness)} \\
&\overline{x} \in X, \ g_i(\overline{x}) \leq 0, \ i = 1, \ldots, m && \text{(primal feasibility)} \\
&h_i(x) = 0, \qquad i = 1, \ldots, l \\
&u_i \geq 0, \qquad i = 1, \ldots, m && \text{(dual feasibility 2)}
\end{aligned}$$

The KKT conditions can be interpreted geometrically as follows. Consider the cone spanned by the active constraints at $\overline{x}$, defined as $N(\overline{x}) = \left\{ \sum_{i \in I} u_i \nabla g_i(\overline{x}) : u_i \geq 0 \right\}$. A solution $\overline{x}$ will then satisfy the KKT conditions if $-\nabla f(\overline{x}) \in N(\overline{x})$, which is equivalent to $-\nabla f(\overline{x}) = \sum_{i=1}^{m} u_i \nabla g_i(\overline{x})$. Figure 18.3 illustrates this condition.

18.4 Constraint qualification

Constraint qualification is a technical condition that needs to be assessed in the context of nonlinear optimisation problems. As we rely on an algebraic description of the set of directions G_0 that serves as proxy for D, it is important to be sure that the former is indeed a reliable description of the latter.

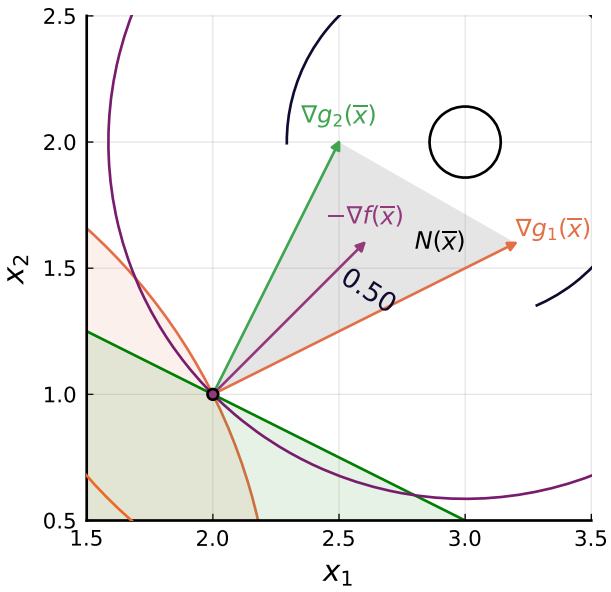


Figure 18.3: Graphical illustration of the KKT conditions at the optimal point $\overline{x}$

In specific, constraint qualification can be seen as a certification that the geometry of the feasible region and gradient information obtained from the constraints that forms it are related at an optimal solution. Remind that gradients can only provide a *first-order* approximation of the feasible region, which might lead to mismatches. This is typically the case when the feasible region has cusps, or a single feasible points.

Constraint qualification can be seen as certificates for proper relationships between the set of feasible directions

$$G_0' = \left\{ d \neq 0 : \nabla g_i(\overline{x})^\top d \leq 0, i \in I \right\}$$

and the cone of tangents (or tangent cone)

$$T = \left\{ d : \ d = \lim_{k \to \infty} \lambda_k (x_k - \overline{x}), \lim_{k \to \infty} x_k = \overline{x}, x_k \in S, \lambda_k > 0, \forall k \right\} \tag{18.1}$$

with $S = \{g_i(x) \leq 0, \ i = 1, \ldots, m; h(x) = 0, \ i = 1, \ldots, l; x \in X\}$.

The cone of tangents is a cone representing all directions in which the feasible region allow for an arbitrarily small movement from the point $\overline{x}$ while retaining feasibility. As the name suggests, it is normally formed by the lines that are tangent to S at $\overline{x}$. Note that, however, if the point is in the interior of $S \subseteq \mathbb{R}^n$, then $T = \mathbb{R}^n$.

One way of interpreting the cone of tangents as defined in (18.1) is the following: consider a sequence of feasible points $x \in S$ in any trajectory you like, but in a way that the sequence converges to $\overline{x}$. Then, take the last (in a limit sense, since $k \to \infty$) x_k and consider this direction from which x_k came onto $\overline{x}$. The collection of all these directions from all possible trajectories is what forms the cone of tangents.

Constraint qualification holds when $T = G_0'$ holds for $\overline{x}$, a condition named *Abadie's constraint*

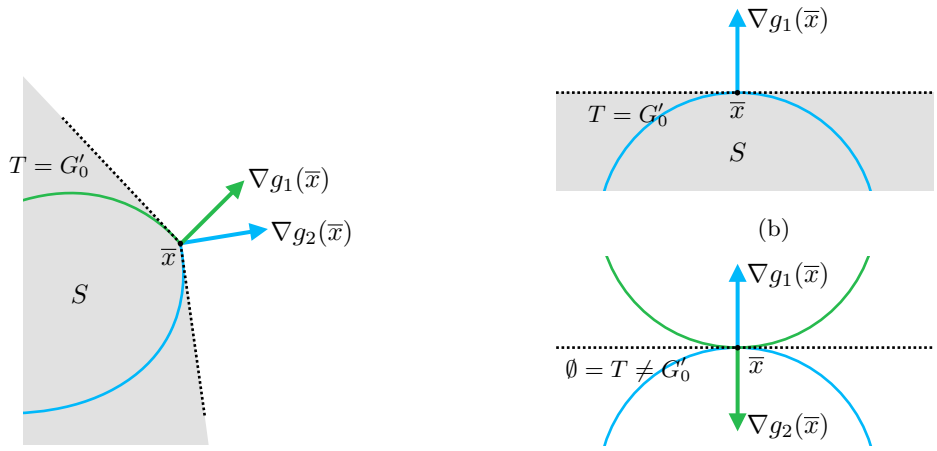


Figure 18.4: CQ holds for 18.4a and 18.4b, since the tangent cone T and the cone of feasible directions G_0' (denoted by the dashed black lines and grey area) match; for 18.4c, they do not match, as $T = \emptyset$

qualification. In the presence of equality constraints, the condition becomes $T = G_0' \cap H_0$, with

$$H_0 = \left\{d : \nabla h_i(\overline{x})^\top d = 0, i = 1, \ldots, l\right\}.$$

Figure 18.4 illustrates the tangent cone T and the cone of feasible directions (G_0') for cases when constraint qualification holds (Figures 18.4a and 18.4b) for which case $T = G_0'$, and a case for when it does not (Figure 18.4c, where $T = \emptyset$ and G_0' is given by the dashed black line).

The importance of Abadie constraint qualification is that it allows for generalising the KKT conditions by replacing the condition the linear independence of the gradients $\nabla g_i(\overline{x})$ for $i \in I$. This allows us to state the KKT conditions as presented in Theorem 18.7.

Theorem 18.7 (Karush-Kuhn-Tucker necessary conditions II)**.** *Consider the problem*

$$(P): \quad \textit{min.} \ \left\{f(x) : g_i(x) \leq 0, i = 1, \ldots, m, x \in X\right\}.$$

Let $X \subseteq \mathbb{R}^n$ be a nonempty open set, and let $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}$ be differentiable for all $i = 1, \ldots, m$. Additionally, for a feasible $\overline{x}$, let $I = \{i : g_i(\overline{x}) = 0\}$ and suppose that Abadie CQ holds *at $\overline{x}$. If $\overline{x}$ solves P locally, there exist scalars u_i for $i \in I$ such that*

$$\begin{aligned}
&\nabla f(\overline{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\overline{x}) = 0 \\
&u_i g_i(\overline{x}) = 0, \ \ i = 1, \ldots, m \\
&u_i \geq 0, \ \ i = 1, \ldots, m.
\end{aligned}$$

Despite being a more general result, Theorem 18.7 is of little use, as Abadie's constraint qualification cannot be straightforwardly verified in practice. Alternatively, we can rely on verifiable constraint qualification conditions that imply Abadie's constraint qualification. Examples include

1. **Linear independence (LI)CQ:** holds at $\overline{x}$ if $\nabla g_i(\overline{x})$, for $i \in I$, as well as $\nabla h_i(\overline{x})$, $i = 1, \ldots, l$ are *linearly independent*.

2. **Affine CQ:** holds for all $x \in S$ if g_i, for all $i = 1, \ldots, m$, and h_i, for all $i = 1, \ldots, l$, are *affine*.

3. **Slater's CQ:** holds for all $x \in S$ if g_i is a *convex* function for all $i = 1, \ldots, m$, h_i is an *affine* function for all $i = 1, \ldots, l$, and there exists $x \in S$ such that $g_i(x) < 0$ for all $i = 1, \ldots, m$.

Slater's constraint qualification is the most frequently used, in particular in the context of convex optimisation problems. One important point to notice is the requirement of not having an empty relative interior, which can be a source of error.

Consider, for example: $P = \left\{\text{min. } x_1 : x_1^2 + x_2 \leq 0, x_2 \geq 0\right\}$. Notice that P is convex and therefore the KKT system for P is

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = 0; u_1, u_2 \geq 0,$$

which has no solution. Thus, the KKT conditions are not necessary for the global optimality of $(0, 0)$. This is due to the lack of CQ, since the feasible region is the single point $(0, 0)$ and the fact that KKT conditions are only sufficient (not necessary), in the presence of convexity.

Corollary 18.8 summarises the setting in which one should expect the KKT conditions to be necessary an sufficient conditions for global optimality, i.e., convex optimisation.

Corollary 18.8 (Necessary and sufficient KKT conditions)**.** *Suppose that Slater's CQ holds. Then, if f is convex, the conditions of Theorem 18.7 are* necessary and sufficient *for $\overline{x}$ to be a global optimal solution.*

Chapter 19

Lagrangian duality

19.1 The concept of relaxation

The idea of using relaxations is central in several constrained optimisation methods. In a general sense, it consists of techniques that remove constraints from the problem to allow for a version, i.e., a *relaxation*, that is simpler to solve and/or can provide information to be used for solving the original problem.

A classical example of the employment of relaxations for solving constrained problems is the branch-and-bound method that uses linear (continuous) relaxations of integer problems to guide the search for optimal solutions that are also integers. However, there are several other examples of settings in which relaxations are purposely derived to lead to problems with a convenient structure that can be exploited.

Let $f : \mathbb{R}^n \to \mathbb{R}$ and $S \subseteq \mathbb{R}^n$. Consider the following problem:

$$(P): \quad \text{min. } \{f(x) : x \in S\}$$

Definition 19.1 provides the conditions for P_R to be a *relaxation* of P, where

$$(P_R): \quad \text{min. } \{f_R(x) : x \in S_R\}$$

with $f_R : \mathbb{R}^n \to \mathbb{R}$, $S_R \subseteq \mathbb{R}^n$.

Definition 19.1 (Relaxation)**.** *P_R is a relaxation of P if and only if:*

1. *$f_R(x) \leq f(x)$, for all $x \in S$;*
2. *$S \subseteq S_R$.*

In specific, P_R is said to be a relaxation for P if $f_R(x)$ bounds $f(x)$ from below (in a minimisation setting) for all $x \in S$ and the enlarged feasible region S_R contains S.

The motivation for using relaxations arises from the possibility of finding a solution to the the original problem P by solving P_R. Clearly, such a strategy would only make sense if P_R possess some attractive property or feature that we can use in our favour to, e.g., improve solution times or create separability that can be further exploited using parallelised computation (which we will discuss in more details in the upcoming lectures). Theorem 19.2 presents the technical result that allows for using relaxations for solving P.

Theorem 19.2 (Relaxation theorem)**.** *Let us define*

$$(P): \quad \textit{min. } \{f(x) : x \in S\} \quad \textit{and} \quad (P_R): \textit{min. } \{f_R(x) : x \in S_R\}$$

If P_R is a relaxation of P, then the following hold:

1. *if P_R is infeasible, so is P;*
2. *if $\overline{x}_R$ is an optimal solution to P_R such that $\overline{x}_R \in S$ and $f_R(\overline{x}_R) = f(\overline{x}_R)$, then $\overline{x}_R$ is optimal to P as well.*

Proof. Result (1) follows since $S \subseteq S_R$. To show (2), notice that $f(\overline{x}_R) = f_R(\overline{x}_R) \leq f_R(x) \leq f(x)$ for all $x \in S$. □

The first part of the proof is a consequence of $S \subset S_R$, meaning that if $x \notin S$, then $x \notin S_R$. The second part combines the optimality of $\overline{x}_R$ (first inequality) and the definition of a relaxation (second inequality) to derive the optimality condition of $\overline{x}_R$ for P, which is $f(\overline{x}_R) \leq f(\overline{x})$ for all $x \in S$.

19.2 Lagrangian dual problems

Lagrangian duality is the body of theory supporting the use of *Lagrangian relaxations* to solve constrained optimisation problems. In what follows, we refer to the relaxation obtained using Lagrangian duality as the *(Lagrangian) dual* problem. Consequently, we refer to original problem as the *primal* problem.

Let $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^l$, and assume that $X \subseteq \mathbb{R}^n$ is an open set. Then, consider P defined as

$$\begin{aligned} (P): \quad \text{min.} \quad & f(x) \\ \text{s.t.:} \quad & g(x) \leq 0 \\ & h(x) = 0 \\ & x \in X. \end{aligned}$$

For a given set of *dual variables* $(u, v) \in \mathbb{R}^{m+l}$ with $u \geq 0$, the *Lagrangian relaxation* (or *Lagrangian dual function*) of P is

$$(D): \ \theta(u, v) = \inf_{x \in X} \ \phi(x, u, v)$$

where

$$\phi(x, u, v) := f(x) + u^\top g(x) + v^\top h(x)$$

is the *Lagrangian function*.

Notice that the Lagrangian dual function $\theta(u, v)$ has a built-in optimisation problem in x, meaning that evaluating $\theta(u, v)$ still requires solving an optimisation problem, which amounts to finding the minimiser $\overline{x}$ for $\phi(x, u, v)$, given (u, v).

19.2.1 Weak and strong duality

Weak and strong duality are, to some extent, consequences of Theorem 19.2 and the fact that the Lagrangian relaxation is indeed a relaxation of P. We start with the equivalent to Definition 19.1, which is referred to as *weak duality*.

Theorem 19.3 (Weak Lagrangian duality)**.** *Let x be a feasible solution to P, and let (u, v) be such that $u \geq 0$, i.e., feasible for D. Then $\theta(u, v) \leq f(x)$.*

Proof. From feasibility, $u \geq 0$, $g(x) \leq 0$ and $h(x) = 0$. Thus, we have that

$$\theta(u, v) = \inf_{x \in X} \left\{ f(x) + u^\top g(x) + v^\top h(x) \right\} \leq f(x) + u^\top g(x) + v^\top h(x) \leq f(x).$$

which completes the proof. □

The proof uses the fact that the infimal of the Lagrangian function $\phi(x, u, v)$, and in fact any value for $\phi(x, u, v)$ for all primal feasible x and dual feasible $u \geq 0$ (a condition for the Lagrangian relaxation to be indeed a relaxation) are bounds to $f(x)$. This arises from observing that $g(x) \leq 0$ for a feasible x. The *Lagrangian dual problem* is the problem used to obtain the best possible relaxation bound $\theta(u, v)$ for $f(x)$, in light of Theorem 19.3. This can be achieved by optimising $\theta(u, v)$ in the space of the dual variables (u, v), that is

$$(D): \ \theta(u, v) = \inf_{x \in X} \ \phi(x, u, v).$$

The use of Lagrangian dual problems is an alternative for dealing with constrained optimisation problems, as they allow to convert the constrained primal into a (typically) unconstrained dual that is potentially easier to handle, or present exploitable properties that can benefit specialised algorithms, such as separability.

Employing Lagrangian relaxations to solve optimisation problems is possible due to the following important results, which are posed as corollaries of Theorem 19.3.

Corollary 19.4 (Weak Lagrangian duality II)**.**

$$\sup_{u,v} \{\theta(u, v) : u \geq 0\} \leq \inf_{x} \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}.$$

Proof. We have $\theta(u, v) \leq f(x)$ for any feasible x and (u, v), thus implying $\sup_{u,v} \{\theta(u, v) : u \geq 0\} \leq \inf_x \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}$ □

Corollary 19.5 (Strong Lagrangian duality)**.** *If $f(\overline{x}) = \theta(\overline{u}, \overline{v})$, $\overline{u} \geq 0$, and $\overline{x} \in \{x \in X : g(x) \leq 0, h(x) = 0\}$, then $\overline{x}$ and $(\overline{u}, \overline{v})$ are optimal solutions to P and D, respectively.*

Proof. Use part (2) of Theorem 19.2 with D being a Lagrangian relaxation. □

Notice that Corollary 19.5 implies that if the optimal solution value of the primal and the dual problems match, then the respective primal and dual solutions are optimal. However, to use Lagrangian relaxations to solve constrained optimisation problems, we need the opposite clause to also hold, which is called *strong duality* and, unfortunately, does not always hold.

Geometric interpretation of Lagrangian duality

To investigate the cases in which strong duality can hold, let us focus on a graphical interpretation of Lagrangian dual problems. For that, let us first define some auxiliary elements.

For the sake of simplicity, consider (P) : min. $\{f(x) : g(x) \leq 0, x \in X\}$ with $f : \mathbb{R}^n \mapsto \mathbb{R}$, a single constraint $g : \mathbb{R}^n \mapsto \mathbb{R}$ and $X \subseteq \mathbb{R}^n$ an open set.

Let us define the mapping $G = \{(y, z) : y = g(x), z = f(x), x \in X\}$, which consists of a mapping of points $x \in X$ to the (y, z)-space obtained using $(f(x), g(x))$. In this setting, solving P means finding a point with minimum ordinate z for which $y \leq 0$. Figure 19.1 illustrate this setting.

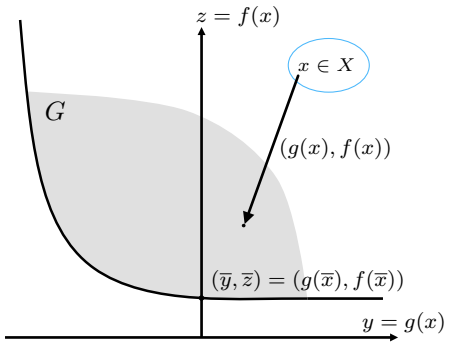


Figure 19.1: Illustration of the mapping G, in which one can see that solving P amounts to finding the lowermost point on the vertical axis (the ordinate) that is still contained within G.

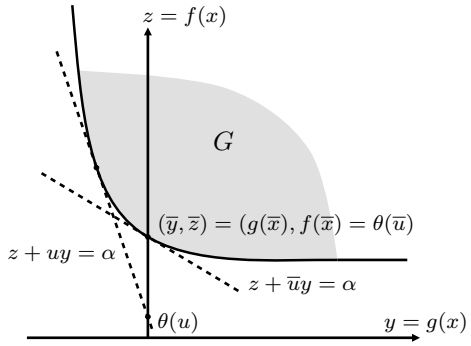


Figure 19.2: Solving the Lagrangian dual problem is the same as finding the coefficient u such that $z = \alpha - uy$ is a supporting hyperplane of G with the uppermost intercept α. Notice that, for $\overline{u}$, the hyperplane supports G at the same point that solves P.

Now, assume that $u \geq 0$ is given. The Lagrangian function is given by

$$\theta(u) = \min_{x} \{f(x) + ug(x) : x \in X\},$$

which can be represented by a hyperplane of the form $z = \alpha - uy$. Therefore, optimising the Lagrangian dual problem $(D) : \sup_u \{\theta(u)\}$ consists of finding the slope $-u$ that would achieve the maximum intercept on the ordinate z while being a supporting hyperplane for G. Figure 19.2 illustrates this effect. Notice that, in this case, the optimal values of the primal and dual problems coincide. The *perturbation function* $v(y) = \min. \{f(x) : g(x) \leq y, x \in X\}$ is an analytical tool that plays an important role in understanding when strong duality holds, which, in essence, is the underlying reason why the optimal values of the primal and dual problems coincide.

Specifically, notice that $v(y)$ is the greatest monotone nonincreasing lower envelope of G. Moreover, the reason why $f(\overline{x}) = \theta(\overline{u})$ is related to the convexity of $v(y)$, which implies that

$$v(y) \geq v(0) - \overline{u}y \text{ for all } y \in \mathbb{R}.$$

Notice that this is a consequence of Theorem 12 from Lecture 2 (that states that convex sets have supporting hyperplanes for all points on their boundary) and Theorem 5 in Lecture 3 (that convex functions have convex epigraphs)

A *duality gap* exists when the perturbation function $v(y)$ does not have supporting hyperplanes within all its domain, which is otherwise the case when $v(y)$ is convex. Figure 19.3 illustrates a case in which $v(y)$ is not convex and therefore $\theta(\overline{u}) < f(\overline{x})$.

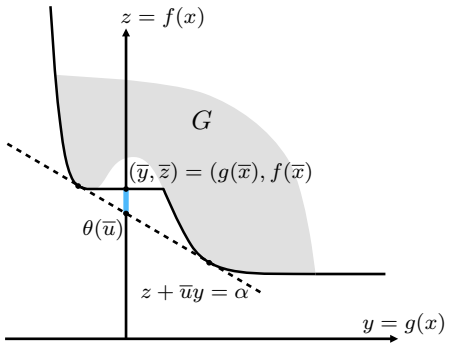


Figure 19.3: An example in which the perturbation function $v(y)$ is not convex. Notice the consequent mismatch between the intercept of the supporting hyperplane and the lowermost point on the ordinate still contained in G.

Let us illustrate the above with two numerical examples. First, consider the following problem

$$\begin{aligned}(P): \quad \text{min.} \quad & x_1^2 + x_2^2 \\ & x_1 + x_2 \geq 4 \\ & x_1, x_2 \geq 0.\end{aligned}$$

The Lagrangian dual function is given by

$$\begin{aligned}(D): \ \theta(u) &= \inf\left\{x_1^2 + x_2^2 + u(-x_1 - x_2 + 4) : x_1, x_2 \geq 0\right\} \\ &= \inf\left\{x_1^2 - ux_1 : x_1 \geq 0\right\} + \inf\left\{x_2^2 - ux_2 : x_2 \geq 0\right\} + 4u \\ &= \begin{cases} -1/2u^2 + 4u, \ \text{if } u \geq 0 \\ -4u, \ \ 1 + 4u^2 \ \text{if } u < 0. \end{cases}\end{aligned}$$

Figures 19.4a and 19.4b provide a graphical representation of the primal problem P and dual problem D. As can be seen, both problems have as optimal value $f(\overline{x}_1, \overline{x}_2) = \theta(\overline{u}) = 8$, with the optimal solution $\overline{x} = (2, 2)$ for P and $\overline{u} = 4$ for D.

To draw the (g, f) map of X, we proceed as follows. First, notice that

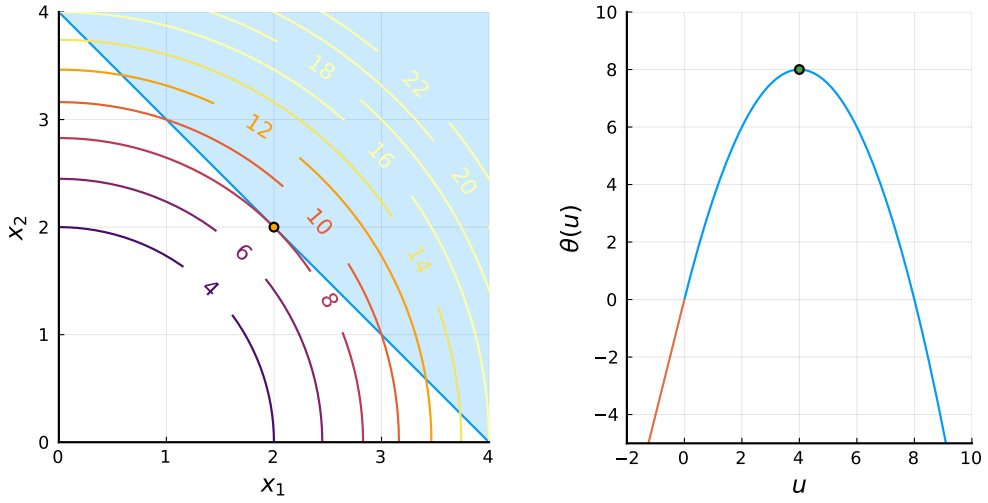
$$v(y) = \text{min.} \ \left\{x_1^2 + x_2^2 : -x_1 - x_2 + 4 \geq y\right\}$$

which shows that $(x_1, x_2) = (0, 0)$ if $y > 4$. For $y \leq 4$, $v(y)$ can be equivalently rewritten as

$$v(y) = \text{min.} \ \left\{x_1^2 + x_2^2 : -x_1 - x_2 + 4 = y\right\}.$$

Let $h(x) = -x_1 - x_2 + 4$ and $f(x) = x_1^2 + x_2^2$. Now, the optimality conditions for $\overline{x}$ to be an optimum for v are such that

$$\nabla f(\overline{x}) + u\nabla h(\overline{x}) = 0 \Rightarrow \begin{cases} 2x_1 - u = 0 \\ 2x_2 - u = 0 \end{cases} \Rightarrow \overline{x}_1 = \overline{x}_2 = u/2.$$



(a) P (b) D

Figure 19.4: The primal problem P as a constrained optimisation problem, and the dual problem D, as an unconstrained optimisation problem. Notice how the Lagrangian dual function is discontinuous, due to the implicit minimisation in x of $\theta(u) = \inf_{x \in X} \phi(x, u)$.

From the definition of $h(x)$, we see that $u = 4 - y$, and thus $\overline{x} = (\frac{4-y}{2}, \frac{4-y}{2})$, which, substituting in $f(x)$ gives $v(y) = (4-y)^2/2$. Note that $v(y) \geq v(0) - \overline{u}y$ holds for all $y \in \mathbb{R}$, that is, $v(y)$ is convex. Also, notice that the supporting hyperplane is exactly $z = 8 - 4y$.

Now, let us consider a second example, in which the feasible set is not convex and, therefore, the mapping G will not be convex either. For that, consider the problem

$$\begin{aligned} (P): \quad \text{min.} \quad & -2x_1 + x_2 \\ \text{s.t.:} \quad & x_1 + x_2 = 3 \\ & x_1, x_2 \in X. \end{aligned}$$

where $X = \{(0,0), (0,4), (4,4), (4,0), (1,2), (2,1)\}$. The optimal point $\overline{x} = (2,1)$. The Lagrangian dual function is given by

$$\begin{aligned} \theta(v) &= \min \{(-2x_1 + x_2) + v(x_1 + x_2 - 3) : (x_1, x_2) \in X\} \\ &= \begin{cases} -4 + 5v, & \text{if } v \leq -1 \\ -8 + v, 5 & \text{if } -1 \leq v \leq 2 \\ -3v, +p_1 & \text{if } v \geq 2. \end{cases} \end{aligned}$$

Figure 19.6a provides a graphical representation of the problem. Notice that to obtain the Lagrangian dual function one must simply take the lowermost segments of the hyperplanes obtained when considering each $x \in X$, which leads to a piecewise concave function, as represented in Figure 19.6b.

Similarly to the previous example, we can plot the G mapping, which in this case consists of the points $x \in X$ mapped as $(h(x), f(x))$, with $h(x) = x_1 + x_2 - 3$ and $f(x) = -2x_1 + x_2$. Notice that $v(y)$ in this case is discontinuous, represented by the three lowermost points. Clearly, $v(y)$

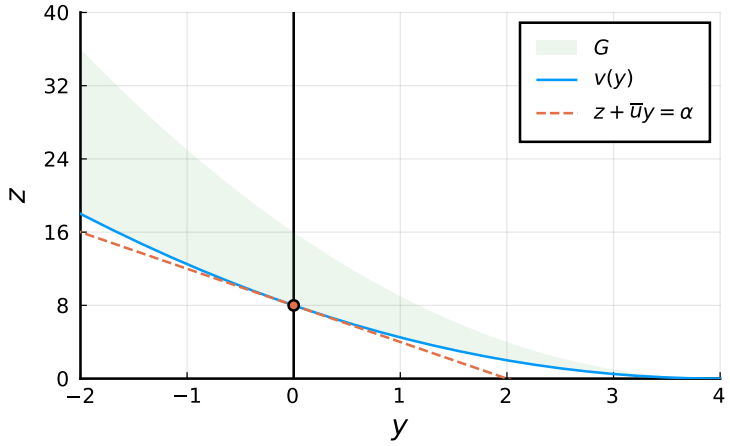


Figure 19.5: The G mapping for the first example.

does not have a supporting hyperplane at the minimum of P, which illustrates the existence of a duality gap, as stated by the fact that $-3 = f(\overline{x}) > \theta(\overline{v}) = -6$.

Strong duality

From the previous graphical interpretation and related examples, it becomes clear that there is a strong tie between strong duality and the convexity of P. This is formally described in Theorem 19.6.

Theorem 19.6. *Let $X \subseteq \mathbb{R}^n$ be a nonempty convex set. Moreover, let $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ be convex functions, and let $h : \mathbb{R}^n \to \mathbb{R}^l$ be an affine function: $h(x) = Ax - b$. Suppose that Slater's constraint qualification holds true. Then*

$$\inf_x \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\} = \sup_{u,v} \{\theta(u,v) : u \geq 0\},$$

where $\theta(u,v) = \inf_{x \in X} \left\{f(x) + u^\top g(x) + v^\top h(x)\right\}$ is the Lagrangian function. Furthermore, if $\inf_x \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}$ is finite and achieved at $\overline{x}$, then $\sup_{u,v} \{\theta(u,v) : u \geq 0\}$ is achieved at $(\overline{u}, \overline{v})$ with $\overline{u} \geq 0$ and $\overline{u}^\top g(\overline{x}) = 0$.

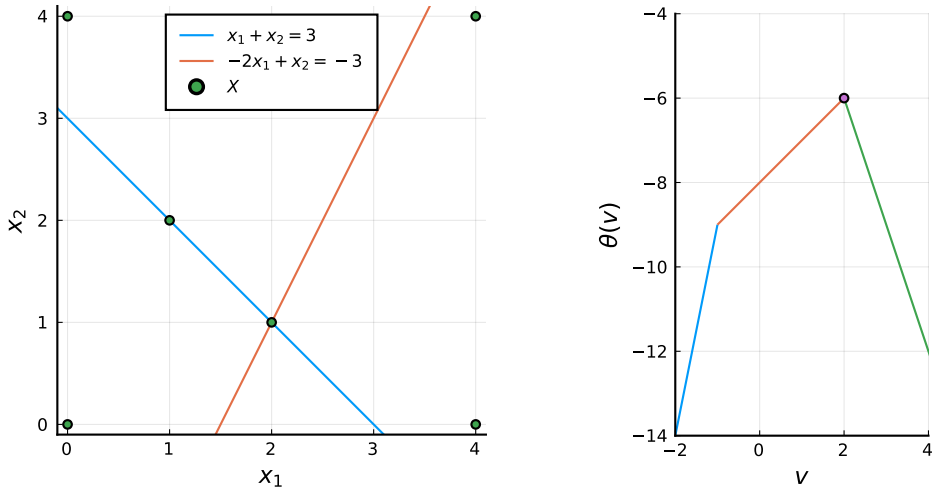
The proof for the strong duality theorem follows the following outline:

1. Let $\gamma = \inf_x \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}$. Suppose that $-\infty < \gamma < \infty$, hence finite (for unbounded problems, $f(x) = -\infty$ implies $\theta(u,v) = -\infty$ since $\theta(u,v) \leq f(x)$ from Theorem 19.3; the right-hand side holds by assumption of the existence of a feasible point from Slater's constraint qualification).

2. Formulate the inconsistent system:

$$f(x) - \gamma < 0, \quad g(x) \leq 0, \quad h(x) = 0, \quad x \in X.$$

3. Use the separation theorem (or a variant form of Farkas theorem) to show that $(\overline{u}_0, \overline{u}, \overline{v})$ with $\overline{u}_0 > 0$ and $\overline{u} \geq 0$ exists such that, after scaling using $\overline{u}_0$ one obtains $\theta(\overline{u}, \overline{v}) := f(x) + \overline{u}^\top g(x) + \overline{v}^\top h(x) \geq \gamma$, $x \in X$, which requires the assumption of Slater's constraint qualification.



(a) P (b) (D) : max. $\theta(v)$

Figure 19.6: The primal problem P as a constrained optimisation problem and the dual problem D. Notice how the Lagrangian dual function is concave and piecewise linear, despite the nonconvex nature of P.

4. From weak duality (Theorem 19.3), we have that $\theta(\overline{u}, \overline{v}) \leq \gamma$, which combined with the above, yields $\theta(\overline{u}, \overline{v}) = \gamma$.

5. Finally, an optimal $\overline{x}$ solving the primal problem implies that $g(\overline{x}) \leq 0$, $h(\overline{x}) = 0$, $\overline{x} \in X$, and $f(x) = \gamma$. From 3, we have $\overline{u}^\top g(\overline{x}) \geq 0$. As $g(\overline{x}) \leq 0$ and $\overline{u} \geq 0$, $\overline{u}^\top g(\overline{x}) \geq 0 = 0$.

The proof uses a variant of the Farkas theorem that states the existence of a solution for the system $u_0(f(x) - \gamma) + u^\top g(x) \geq 0, x \in X$ with $(u_0, u, v) \neq 0$, what can be shown to be the case if Slater's constraint qualification holds. This, combined with weak duality stated in Theorem 19.3 yields strong duality.

19.2.2 Employing Lagrangian duality for solving optimisation problems

Weak duality can be used to derive a stopping criterion for solution methods that can generate both primal and dual feasible solutions, also known as primal-dual pairs. Such methods are typically referred to as primal-dual methods, being the primal-dual interior point method (which we will discuss in details in an upcoming lecture) perhaps the most widely known.

For feasible x and (u, v), one can bound how suboptimal $f(x)$ is, by noticing that

$$f(x) - f(\overline{x}) \leq f(x) - \theta(u, v),$$

which is a consequence of $f(\overline{x}) \geq \theta(u, v)$ (i.e., weak duality). We say that x is ϵ-optimal, with $\epsilon = f(x) - \theta(u, v)$.

In essence, (u, v) is a certificate of (sub-)optimality of x, as (u, v) proves that x is ϵ-optimal. Moreover, in case strong duality holds, under the conditions of Theorem 6, one can expect ϵ converge to zero.

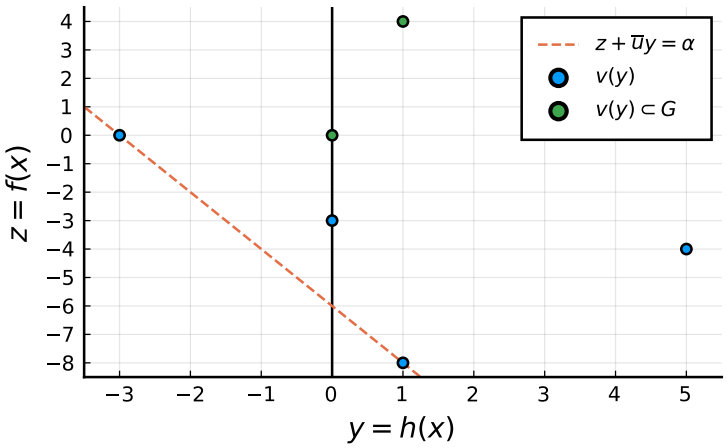


Figure 19.7: The G mapping for the second example. The blue dots represent the perturbtion function $v(y)$, which is not convex and thus cannot be supported everywhere. Notice the duality gap represented by the difference between the intercept of $z = -6 - 2y$ and the optimal value of P at (0,-3).

To see how this the case, observe the following. First, as can be seen in Theorem 19.6, a consequence of strong duality is that complementarity conditions $\overline{u}^\top g(\overline{x}) \geq 0 = 0$ hold for an optimal primal-dual pair $(\overline{x}, (\overline{u}, \overline{v}))$. Secondly, notice that, by definition, $\overline{x}$ and $(\overline{u}, \overline{v})$ are primal and dual feasible, respectively.

The last component missing is to notice that, if $\overline{x}$ is a minimiser for $\phi(x, \overline{u}, \overline{v}) = f(x) + \overline{u}^\top g(x) + \overline{v}^\top h(x)$, then we must have

$$\nabla f(\overline{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\overline{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\overline{x}) = 0.$$

Combining the above, one can see that we have listed all of the KKT optimality conditions, which under the assumptions of Theorem 19.6 are known to be necessary and sufficient for global optimality. That is, in this case, any primal dual pair for which the objective function values match will automatically be a point satisfying the KKT conditions and therefore globally optimal. This provides an alternative avenue to search for optimal solutions, relying on Lagrangian dual problems.

19.2.3 Saddle point optimality and KKT conditions*

An alternative perspective for establishing necessary and sufficient conditions for strong duality to hold involves identifying the existence of saddle points for the Lagrangian dual problem.

Let us first define saddle points in the context of Lagrangian duality. Let

$$(P): \quad \min. \ \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}.$$

Let us define the Lagrangian function $\phi(x, u, v) = f(x) + u^\top g(x) + v^\top h(x)$. A solution $(\overline{x}, \overline{u}, \overline{v})$ is called a *saddle point* if $\overline{x} \in X$, $\overline{u} \geq 0$, and

$$\phi(\overline{x}, u, v) \leq \phi(\overline{x}, \overline{u}, \overline{v}) \leq \phi(x, \overline{u}, \overline{v})$$

for all $x \in X$ and (u, v) such that $u \geq 0$.

Notice that this definition implies that:

- $\overline{x}$ minimises $\phi(x, u, v)$ when (u, v) is fixed at $(\overline{u}, \overline{v})$;

- $(\overline{u}, \overline{v})$ maximises $\phi(x, u, v)$ when x is fixed at $\overline{x}$.

This insight allows for the development of methods that can alternatively solve the Lagrangian dual problem in the space of primal variables x and dual variables (u, v) in a block-coordinate descent fashion.

Theorem 19.7 establishes the relationship between the existence of saddle points for Lagrangian dual problems and zero duality gaps.

Theorem 19.7 (Saddle point optimality and zero duality gap)**.** *A solution $(\overline{x}, \overline{u}, \overline{v})$ with $\overline{x} \in X$ and $\overline{u} \geq 0$ is a saddle point for the Lagrangian function $\phi(x, u, v) = f(x) + u^\top g(x) + v^\top h(x)$ if and only if:*

1. $\phi(\overline{x}, \overline{u}, \overline{v}) = min.\ \{\phi(x, \overline{u}, \overline{v}) : x \in X\}$

2. $g(\overline{x}) \leq 0$, $h(\overline{x}) = 0$, *and*

3. $\overline{u}^\top g(\overline{x}) = 0$

Moreover, $(\overline{x}, \overline{u}, \overline{v})$ is a saddle point if and only if $\overline{x}$ and $(\overline{u}, \overline{v})$ are optimal solutions for the primal (P) and dual (D) problems, respectively, with $f(\overline{x}) = \theta(\overline{u}, \overline{v})$.

From Theorem 19.7 it becomes clear that there is a strong connection between the existence of saddle points and the KKT conditions for optimality. Figure 19.8 illustrates the existence of a saddle point and the related zero optimality gap.

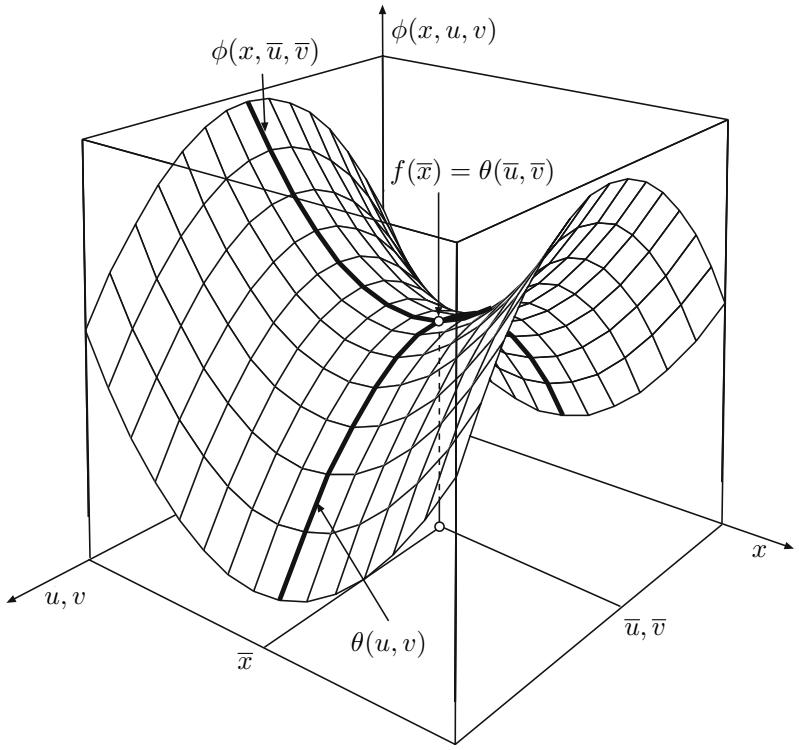


Figure 19.8: Illustration of a saddle point for the Lagrangian dual problem

19.3 Properties of Lagrangian functions

Lagrangian duals are a useful framework for devising solution methods for constrained optimisation problems if solving the dual problem can be done efficiently or exposes some exploitable structure.

One important property that Lagrangian dual functions present is that they are *concave piecewise linear* in the dual multipliers. Moreover, they are continuous and thus have subgradients everywhere. Notice however that they are typically not differentiable, requiring the employment of a *nonsmooth optimisation* method to be appropriately solved. Theorem 19.8 establishes the concavity of the Lagrangian dual function.

Theorem 19.8 (Concavity of Lagrangian dual functions)**.** *Let $X \subseteq \mathbb{R}^n$ be a nonempty compact set, and let $f : \mathbb{R}^n \to \mathbb{R}$ and $\beta : \mathbb{R}^n \to \mathbb{R}^{m+l}$, with $w^\top \beta(x) = \binom{u}{v}^\top \binom{g(x)}{h(x)}$ be continuous. Then $\theta(w) = \inf_x \{f(x) + w^\top \beta(x) : x \in X\}$ is concave in $\mathbb{R}^{m+l}$*

Proof. Since f and β are continuous and X is compact, θ is finite on R^{m+l}. Let w_1, $w_2 \in \mathbb{R}^{m+l}$,

and let $\lambda \in (0,1)$. We have

$$\begin{aligned}
&\theta[\lambda w_1 + (1-\lambda)w_2] = \inf_x \left\{ f(x) + [\lambda w_1 + (1-\lambda)w_2]^\top \beta(x) : x \in X \right\} \\
&= \inf_x \left\{ \lambda[f(x) + w_1^\top \beta(X)] + (1-\lambda)[f(x) + w_2^\top \beta(x)] : x \in X \right\} \\
&\geq \lambda \inf_x \left\{ f(x) + w_1^\top \beta(x) : x \in X \right\} + (1-\lambda) \inf_x \left\{ f(x) + w_2^\top \beta(x) : x \in X \right\} \\
&= \lambda\theta(w_1) + (1-\lambda)\theta(w_2). \qquad \square
\end{aligned}$$

The proof uses the fact that the Lagrangian function $\theta(w)$ is the infimum of affine functions in w, and therefore concave. An alternative approach to show the concavity of the Lagrangian function is to show that it has subgradients everywhere. This is established in Theorem 19.9.

Theorem 19.9. *Let $X \subset \mathbb{R}^n$ be a nonempty compact set, and let $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $\beta : \mathbb{R}^n \mapsto \mathbb{R}^{m+l}$, with $w^\top \beta(x) = \binom{u}{v}^\top \binom{g(x)}{h(x)}$ be continuous. If $\overline{x} \in X(\overline{w}) = \left\{ x \in X : x = \arg\min \left\{ f(x) + w^\top \beta(x) \right\} \right\}$, then $\beta(\overline{x})$ is a subgradient of $\theta(\overline{w})$.*

Proof. Since f and β are continuous and X is compact, $X(\overline{w}) \neq \emptyset$ for any $w \in \mathbb{R}^{m+l}$. Now, let $\overline{w} \in \mathbb{R}^{m+l}$ and $\overline{x} \in X(\overline{w})$. Then

$$\begin{aligned}
\theta(w) &= \inf \left\{ f(x) + w^\top \beta(x) : x \in X \right\} \\
&\leq f(\overline{x}) + w^\top \beta(\overline{x}) \\
&= f(\overline{x}) + (w - \overline{w})^\top \beta(\overline{x}) + \overline{w}^\top \beta(\overline{x}) \\
&= \theta(\overline{w}) + (w - \overline{w})^\top \beta(\overline{x}). \qquad \square
\end{aligned}$$

Theorem 19.9 can be used to derive a simple optimisation method for Lagrangian functions using subgradient information that is easily available from the term $\beta(w)$.

19.3.1 The subgradient method

One challenging aspect concerning the solution of Lagrangian dual functions is that very often they are not differentiable. This requires an adaptation of the gradient method to consider subgradient information instead.

The challenge with using subgradients (instead of gradients) is that subgradients are not guaranteed to be descent directions (as opposed to gradients being the steepest descent direction under adequate norm). Nevertheless, for suitable step size choices, convergence can be observed. Figure 19.9 illustrates the fact that subgradients are not necessarily descent directions.

Algorithm 22 summarises the subgradient method. Notice that the stopping criterion emulates the optimality condition $0 \in \partial\theta(w_k)$, but in practice, one also enforces more heuristically driven criteria such as maximum number of iterations or a given number of iterations without observable improvement on the value of $\theta(w)$.

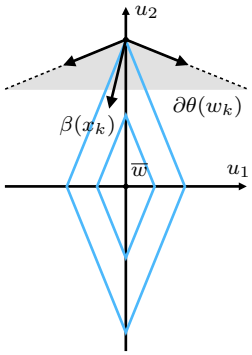


Figure 19.9: One possible subgradient $\beta(x_k)$ that is a descent direction for suitable step size. Notice that within the subdifferential $\partial\theta(w_k)$ other subgradients that are not descent direction are available.

Algorithm 17 Subgradient method

1: **initialise.** tolerance $\epsilon > 0$, initial point w_0, iteration count $k = 0$.
2: **while** $||\beta(x_k)||_2 > \epsilon$ **do**
3: $x_k \leftarrow \arg\min_x \left\{\theta(w_k) = \inf_x \left\{f(x) + w_k^\top \beta(x)\right\}\right\}$
4: $LB_k = \max\{LB_k, \theta(w_k)\}$
5: update λ_k
6: $w_{k+1} = w_k + \lambda_k \beta(x_k)$.
7: $k \leftarrow k + 1$.
8: **end while**
9: **return** $LB_k = \theta(w_k)$.

One critical aspect associated with the subgradient method is the step size update described in Step 5 of Algorithm 22. Theoretical convergence is guaranteed if Step 5 generates a sequence $\{\lambda_k\}$ such that $\sum_{k=0}^{\infty} \lambda_k \to \infty$ and $\lim_{k\to\infty} \lambda_k = 0$. However, discrepant performance can be observed for distinct parametrisation of the method.

The classical step update rule employed for the subgradient method is known as the Polyak rule, which is given by

$$\lambda_{k+1} = \frac{\alpha_k(LB_k - \theta(w_k))}{||\beta(x_k)||^2}$$

with $\alpha_k \in (0, 2)$ and LB_k being the best-available lower-estimate of $\theta(\overline{w})$. This rule is inspired by the following result.

Proposition 19.10 (Improving step size)**.** *If w_k is not optimal, then, for all optimal dual solutions $\overline{w}$, we have*

$$||w_{k+1} - \overline{w}|| < ||w_k - \overline{w}||$$

for all step sizes λ_k such that

$$0 < \lambda_k < \frac{2(\theta(\overline{w}) - \theta(w_k))}{||\beta(x_k)||^2}.$$

Proof. We have that $||w_{k+1} - \overline{w}||^2 = ||w_k + \lambda_k\beta(x_k) - \overline{w}||^2 =$

$$||w_k - \overline{w}||^2 - 2\lambda_k(\overline{w} - w_k)^\top \beta(x_k) + (\lambda_k)^2||\beta(x_k)||^2.$$

By the subgradient inequality: $\theta(\overline{w}) - \theta(w_k) \leq (\overline{w} - w_k)^\top \beta_k$. Thus

$$||w_{k+1} - \overline{w}||^2 \leq ||w_k - \overline{w}||^2 - 2\lambda_k(\theta(\overline{w}) - \theta(w_k))^\top \beta(x_k) + (\lambda_k)^2 ||\beta(x_k)||^2.$$

Parametrising the last two terms by $\gamma_k = \frac{\lambda_k ||\beta(x_k)||^2}{\theta(\overline{w}) - \theta(w_k)}$ leads to

$$||w_{k+1} - \overline{w}||^2 \leq ||w_k - \overline{w}||^2 - \frac{\gamma_k(2 - \gamma_k)(\theta(\overline{w}) - \theta(w_k))^2}{||\beta(x_k)||^2}.$$

Notice that if $0 < \lambda_k < \frac{2(\theta(\overline{w}) - \theta(w_k))}{||\beta(x_k)||^2}$ then $0 < \gamma_k < 2$ and, thus, $||w_{k+1} - \overline{w}|| < ||w_k - \overline{w}||$. □

In practice, since $\theta(\overline{w})$ is not known, it must be replaced by a proxy LB_k, which is chosen to be a lower bound on $\theta(\overline{w})$ to still satisfy the subgradient inequality. The α_k is then reduced from the nominal value 2 to correct for the estimation error of term $\theta(\overline{w}) - \theta(w_k)$.

Chapter 20

Penalty methods

20.1 Penalty functions

The employment of penalty functions is a paradigm for solving constrained optimisation problems. The central idea of this paradigm is to convert the constrained optimisation problem into an unconstrained optimisation problem that is augmented with a *penalty function*, which penalises violations of the original constraints. The role of the penalty function is to allow steering the search towards feasible solutions in the search for optimal solutions.

Consider the problem (P) : min. $\{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}$. A *penalised version* of P is given by

$$(P_\mu) : \text{min. } \{f(x) + \mu\alpha(x) : x \in X\},$$

where $\mu > 0$ is a *penalty term* and $\alpha(x) : \mathbb{R}^n \mapsto \mathbb{R}$ is a *penalty function* of the form

$$\alpha(x) = \sum_{i=1}^{m} \phi(g_i(x)) + \sum_{i=1}^{l} \psi(h_i(x)). \tag{20.1}$$

For $\alpha(x)$ to be a suitable penalty function, one must observe that $\phi : \mathbb{R}^n \mapsto \mathbb{R}$ and $\psi : \mathbb{R}^n \mapsto \mathbb{R}$ are continuous and satisfy

$$\begin{aligned} \phi(y) &= 0 \text{ if } y \leq 0 \text{ and } \phi(y) > 0 \text{ if } y > 0 \\ \psi(y) &= 0 \text{ if } y = 0 \text{ and } \psi(y) > 0 \text{ if } y \neq 0. \end{aligned}$$

Typical options are $\phi(y) = ([y]^+)^p$ with $p \in \mathbb{Z}_+$ and $\psi(y) = |y|^p$ with $p = 1$ or $p = 2$.

Figure 20.1 illustrates the solution of (P) : min. $\left\{x_1^2 + x_2^2 : x_1 + x_2 = 1, x \in \mathbb{R}^2\right\}$ using a penalty-based approach. Using $\alpha(x_1, x_2) = (x_1 + x_2 - 1)^2$, the penalised auxiliary problem P_μ becomes (P_μ) : min. $\left\{x_1^2 + x_2^2 + \mu(x_1 + x_2 - 1)^2 : x \in \mathbb{R}^2\right\}$. Since f_μ is convex and differentiable, necessary and sufficient optimality conditions $\nabla f_\mu(x) = 0$ imply:

$$\begin{aligned} x_1 + \mu(x_1 + x_2 - 1) &= 0 \\ x_2 + \mu(x_1 + x_2 - 1) &= 0, \end{aligned}$$

which gives $x_1 = x_2 = \frac{\mu}{2\mu+1}$.

One can notice that, as μ increases, the solution of the unconstrained penalised problem, represented by the level curves, becomes closer to the optimal of the original constrained problem P, represented by the dot on the hyperplane defined by $x_1 + x_2 = 1$.

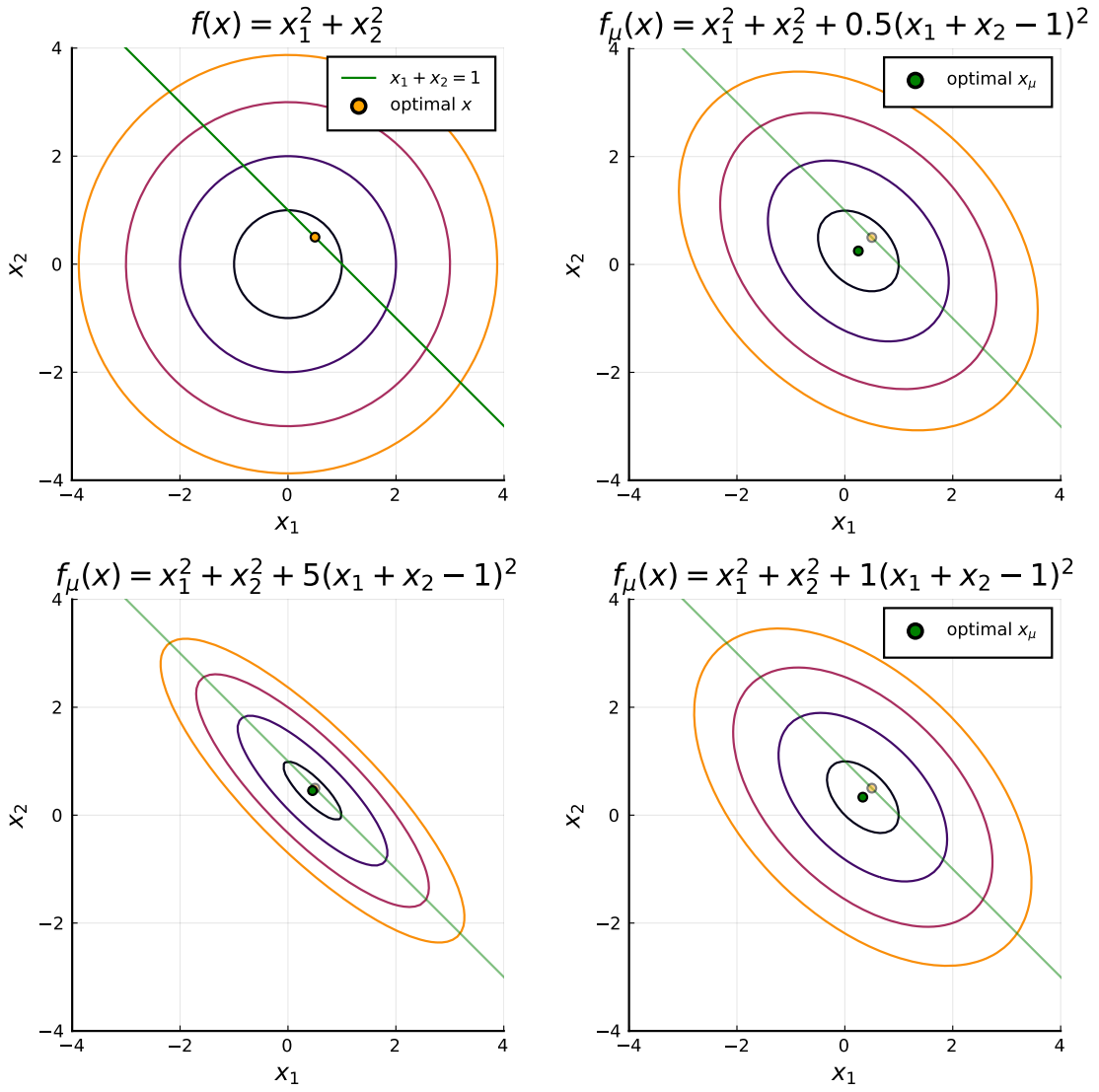


Figure 20.1: Solving the constrained problem P (top left) by gradually increasing the penalty term μ (0.5, 1, and 5, in clockwise order)

20.1.1 Geometric interpretation

A similar geometrical analysis to that performed with the Lagrangian duals can be employed for understanding how penalised problems can obtain optimal solutions. For that, let us the problem from the previous example (P) : min. $\left\{x_1^2 + x_2^2 : x_1 + x_2 = 1, x \in \mathbb{R}^2\right\}$. Let $G : \mathbb{R}^2 \to \mathbb{R}^2$ be a mapping $\left\{[h(x), f(x)] : x \in \mathbb{R}^2\right\}$, and let $v(\epsilon) = \text{min.} \left\{x_1^2 + x_2^2 : x_1 + x_2 - 1 = \epsilon,\ x \in \mathbb{R}^2\right\}$. The optimal solution is $x_1 = x_2 = \frac{1+\epsilon}{2}$ with $v(\epsilon) = \frac{(1+\epsilon)^2}{2}$.

Minimising $f(x) + \mu(h(x)^2)$ consists of moving the curve downwards until a single contact point ϵ_μ remains. One can notice that, as $\mu \to \infty$, $f + \mu h$ becomes sharper ($\mu_2 > \mu_1$), and ϵ_μ *converges*

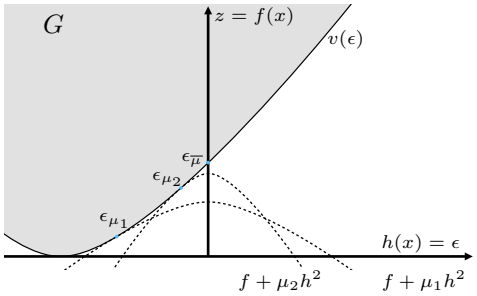


Figure 20.2: Geometric representation of penalised problems in the mapping $G = [h(x), f(x)]$

to the optimum $\epsilon_{\overline{\mu}}$. Figure 20.2 illustrates this behaviour.

The shape of the penalised problem curve is due to the following. First, notice that

$$\begin{aligned}
&\min_x \left\{ f(x) + \mu \sum_{i=1}^{l} (h_i(x))^2 \right\} \\
=&\min_{x,\epsilon} \left\{ f(x) + \mu ||\epsilon||^2 : h_i(x) = \epsilon, i = 1, \ldots, l \right\} \\
=&\min_{\epsilon} \left\{ \mu ||\epsilon||^2 + \min_x \left\{ f(x) : h_i(x) = \epsilon, i = 1, \ldots, l \right\} \right\} \\
=&\min_{\epsilon} \left\{ \mu ||\epsilon||^2 + v(\epsilon) \right\}.
\end{aligned}$$

Consider $l = 1$, and let $x_\mu = \arg\min_x \left\{ f(x) + \mu \sum_{i=1}^{l} (h_i(x))^2 \right\}$ with $h(x_\mu) = \epsilon_\mu$, implying that $\epsilon_\mu = \arg\min_\epsilon \left\{ \mu ||\epsilon||^2 + v(\epsilon) \right\}$. Then, the following holds

1. $f(x_\mu) + \mu (h(x_\mu))^2 = \mu \epsilon_\mu^2 + v(\epsilon_\mu) \Rightarrow f(x_\mu) = v(\epsilon_\mu)$, since $h(x_\mu) = \epsilon_\mu$;
2. and $v'(\epsilon_\mu) = \frac{\partial}{\partial \epsilon} (f(x_\mu) + \mu (h(x_\mu))^2 - \mu \epsilon_\mu^2) = -2\mu\epsilon_\mu$.

Therefore, $(h(x_\mu), f(x_\mu)) = (\epsilon_\mu, v(\epsilon_\mu))$. Denoting $f(x_\mu) + \mu h(x_\mu)^2 = k_\mu$, we see the parabolic function $f = k_\mu - \mu\epsilon^2$ matching $v(\epsilon_\mu)$ for $\epsilon = \epsilon_\mu$ and has the slope $-2\mu\epsilon$, matching that of $v(\epsilon)$ at that point.

20.1.2 Penalty function methods

The convergent behaviour of the penalised problem as the penalty term μ increases inspires the development of a simple yet powerful method for optimising constrained optimisation problems.

That is, consider the problem P defined as

$$\begin{aligned}
(P) : \min.\ & f(x) \\
& g_i(x) \leq 0, i = 1, \ldots, m, \\
& h_i(x) = 0, i = 1, \ldots, l, \\
& x \in X.
\end{aligned}$$

We seek to solve P by solving $\sup_\mu \{\theta(\mu)\}$ for $\mu > 0$, where

$$\theta(\mu) = \min.\ \{ f(x) + \mu\alpha(x) : x \in X \}$$

and $\alpha(x)$ is a penalty function as defined in (20.1). For that to be possible, we need first to state a convergence result guaranteeing that

$$\text{min. } \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\} = \sup_{\mu \geq 0} \theta(\mu) = \lim_{\mu \to \infty} \theta(\mu).$$

In practice, that would mean that μ_k can be increased at each iteration k until a suitable tolerance is achieved. Theorem 20.1 states the convergence of penalty based methods.

Theorem 20.1 (Convergence of penalty-based methods)**.** *Consider the problem P, where f, g_i for $i = 1, \ldots, m$, and h_i for $i = 1, \ldots, l$ are continuous, and $X \subset \mathbb{R}^n$ a compact set. Suppose that, for each μ, there exists $x_\mu = \arg\min \{f(x) + \alpha(x) : x \in X\}$, where α is a suitable penalty function and $\{x_\mu\}$ is contained within X. Then*

$$\min_x \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\} = \sup_{\mu \geq 0} \{\theta(\mu)\} = \lim_{\mu \to \infty} \theta(\mu),$$

where $\theta(\mu) = \min_x \{f(x) + \mu\alpha(x) : x \in X\} = f(x_\mu) + \mu\alpha(x_\mu)$. Also, the limit of any convergent subsequence of $\{x_\mu\}$ is optimal to the original problem and $\mu\alpha(x_\mu) \to 0$ as $\mu \to \infty$.

Proof. We first show that $\theta(\mu)$ are nondecreasing function of μ. Let $0 < \lambda < \mu$. From the definition of $\theta(\mu)$, we have that

$$f(x_\mu) + \lambda\alpha(x_\mu) \geq f(x_\lambda) + \lambda\alpha(x_\lambda) \tag{20.2}$$

Adding and subtracting $\mu\alpha(x_\mu)$ in the left side of (20.2), we conclude that $\theta(\mu) \geq \theta(\lambda)$. Now, for $x \in X$ with $g(x) \leq 0$ and $h(x) = 0$, notice that $\alpha(x) = 0$. This implies that

$$f(x) = f(x) + \mu\alpha(x) \geq \inf_x \{f(x) + \mu\alpha(x) : x \in X\} = \theta(\mu) \tag{20.3}$$

and, therefore, $\theta(\mu)$ is bounded above, and thus $\sup_{\mu \geq 0} \theta(\mu) = \lim_{\mu \to \infty} \theta(\mu)$. For that to be the case, we must have that $\mu\alpha(x_\mu) \to 0$ as $\mu \to \infty$. Moreover, we notice from (20.3) that

$$\min_x \{f(x) : g(x) \leq 0, h(x) = 0, x \in X\} \geq \lim_{\mu \to \infty} \theta(\mu). \tag{20.4}$$

On the other hand, take any convergent subsequence $\{x_{\mu_k}\}$ of $\{x_\mu\}_{\mu \to \infty}$ with limit $\overline{x}$. Then

$$\sup_{\mu \geq 0} \theta(\mu) \geq \theta(\mu_k) = f(x_{\mu_k}) + \mu\alpha(x_{\mu_k}) \geq f(x_{\mu_k}).$$

Since $x_{\mu_k} \to \overline{x}$ as $\mu \to \infty$ and f is continuous, this implies that $\sup_{\mu \geq 0} \theta(\mu) \geq f(\overline{x})$. Combined with (20.4), we have that $f(\overline{x}) = \sup_{\mu \geq 0} \{\theta(\mu)\}$ and thus the result follows. □

The proof starts by demonstrating the nonincreasing behaviour of penalty functions and nondecreasing behaviour of $\theta(\mu)$ to allow for convergence. By noticing that

$$f(x_\mu) + \lambda\alpha(x_\mu) + \mu\alpha(x_\mu) - \mu\alpha(x_\mu) = \theta(\mu) + (\lambda - \mu)\alpha(x_\mu) \geq f(x_\lambda) + \lambda\alpha(x_\lambda) = \theta(\lambda)$$

and that $\lambda - \mu < 0$, we can infer that $\theta(\mu) \geq \theta(\lambda)$. It is also interesting to notice how the objective function $f(x)$ and infeasibility $\alpha(x)$ behave as we increase the penalty coefficient μ. For that, notice that using the same trick in the proof for two distinct values $0 < \lambda < \mu$, we have

1. $f(x_\mu) + \lambda\alpha(x_\mu) \geq f(x_\lambda) + \lambda\alpha(x_\lambda)$

2. $f(x_\lambda) + \mu\alpha(x_\lambda) \geq f(x_\mu) + \mu\alpha(x_\mu)$.

Notice that in 1, we use the fact that $x_\lambda = \arg\min_x \theta(\lambda) = \arg\min_x \{f(x) + \lambda\alpha(x)\}$ and therefore, must be less or equal then $f(x_\mu) + \lambda\alpha(x_\mu)$ for an arbitrary $x_\mu \in X$. The same logic is employed in 2, but reversed in λ and μ. Adding 1 and 2, we obtain $(\mu - \lambda)(\alpha(x_\lambda) - \alpha(x_\mu)) \geq 0$ and conclude that $\alpha(x_\mu) \leq \alpha(x_\lambda)$ for $\mu > \lambda$, i.e., that $\alpha(x)$ is nonincreasing in μ.

Moreover, from the first inequality, we have that $f(x_\mu) \geq f(x_\lambda)$. Notice how this goes in line with what one would expect from the method: as we increase the penalty coefficient μ, the optimal infeasibility, measured by $\alpha(x_\mu)$ decreases, while the objective function value $f(x_\mu)$ worsens at it is slowly "forced" to be closer to the original feasible region.

Note that the assumption of compactness plays a central role in this proof, such that $\theta(\mu)$ can be evaluated for any μ as $\mu \to \infty$. Though this is a strong assumption, it tends to not be so restrictive in practical cases, since variables typically lie within finite lower and upper bounds. Finally, notice that $\alpha(\overline{x}) = 0$ implies that $\overline{x}$ is feasible for g_i for $i = 1, \ldots, m$, and h_i for $i = 1, \ldots, l$, and thus optimal for P. This is stated in the following corollary.

Corollary 20.2. *If $\alpha(x_\mu) = 0$ for some μ, then x_μ is optimal for P.*

Proof. If $\alpha(x_\mu) = 0$, then x_μ is feasible. Moreover, x_μ is optimal, since

$$\begin{aligned}\theta(\mu) &= f(x_\mu) + \mu\alpha(x_\mu) \\ &= f(x_\mu) \leq \inf\{f(x) : g(x) \leq 0, h(x) = 0, x \in X\}.\end{aligned}$$

□

A technical detail of the proof of Theorem 20.1 is that the convergence of such approach is asymptotically, i.e., by making μ arbitrarily large, x_μ can be made arbitrarily close to the true optimal $\overline{x}$ and $\theta(\mu)$ can be made arbitrarily close to the optimal value $f(\overline{x})$. In practice, this strategy tends to be prone to computational instability.

The computational instability arises from the influence that the penalty term exerts in some of the eigenvalues of the Hessian of the penalised problem. Let $H_\mu(x_\mu)$ be the Hessian of the penalised function at x_μ. Recall that conditioning is measured by $\kappa = \frac{\max_{i=1,\ldots,n} \lambda_i}{\min_{i=1,\ldots,n} \lambda_i}$, where $\{\lambda_i\}_{i=1,\ldots,n}$ are the eigenvalues of $H_\mu(x_\mu)$. Since the influence is only on some of the eigenvalues, this affects the conditioning of the problem and might lead to numerical instabilities. An indication of that can be seen in Figure 20.1, where one can notice the elongated profile of the function as the penalty term μ increases.

Consider the following example. Let the penalised function $f_\mu(x) = x_1^2 + x_2^2 + \mu(x_1 + x_2 - 1)^2$.

The Hessian of $f_\mu(x)$ is

$$\nabla^2 f_\mu(x) = \begin{bmatrix} 2(1+\mu) & 2\mu \\ 2\mu & 2(1+\mu) \end{bmatrix}.$$

Solving $\mathbf{det}(\nabla^2 f_\mu(x) - \lambda I) = 0$, we obtain $\lambda_1 = 2$, $\lambda_2 = 2(1 + 2\mu)$, with eigenvectors $(1, -1)$ and $(1, 1)$, which gives $\kappa = (1 + 2\mu)$. This illustrates that the eigenvalues, and consequently the conditioning number, is proportional to the penalty term.

20.2 Augmented Lagrangian method of multipliers

For simplicity, consider the (primal) problem P as $(P) : \min.\ \{f(x) : h_i(x) = 0,\ i = 1, \ldots, l\}$. The augmented Lagrangian method of multipliers arises from the idea of seeking for a penalty term that would allow for exact convergence for a finite penalty.

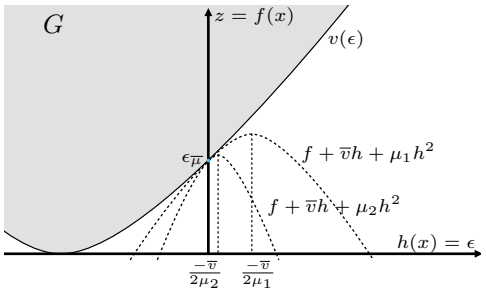


Figure 20.3: Geometric representation of augmented Lagrangians in the mapping $G = [h(x), f(x)]$

Considering the geometrical interpretation in Figure 20.2, one might notice that a horizontal shift in the penalty curve would allow for the extreme point of the curve to match the optimum on the z ordinate.

Therefore, we consider a modified penalised problem of the form

$$f_\mu(x) = f(x) + \mu \sum_{i=1}^{l} (h_i(x) - \theta_i)^2$$

where θ_i is the shift term. One can notice that

$$\begin{aligned} f_\mu(x) &= f(x) + \mu \sum_{i=1}^{l} (h_i(x) - \theta_i)^2 \\ &= f(x) + \mu \sum_{i=1}^{l} h_i(x)^2 - \sum_{i=1}^{l} 2\mu\theta_i h_i(x) + \mu \sum_{i=1}^{l} \theta_i^2 \\ &= f(x) + \sum_{i=1}^{l} v_i h_i(x) + \mu \sum_{i=1}^{l} h_i(x)^2, \end{aligned}$$

with $v_i = -2\mu\theta_i$. The last term is a constant and can be dropped.

The term *augmented Lagrangian* refers to the fact that $f_\mu(x)$ is equivalent to the Lagrangian function of problem P, augmented by the penalty term.

This allows for noticing important properties associated with the augmented Lagrangian $f_\mu(x)$. For example, assume that $(\overline{x}, \overline{v})$ is a KKT solution to P. Then the optimality condition

$$\nabla_x f_\mu(x) = \nabla f(x) + \sum_{i=1}^{l} \overline{v}_i \nabla h_i(x) + 2\mu \sum_{i=1}^{l} h_i(x) \nabla h_i(x) = 0,$$

implies that the optimal solution $\overline{x}$ can be recovered using a finite penalty term μ, unlike with the previous penalty-based method. The existence of finite penalty terms $\mu > 0$ that can recover optimality has an interesting geometrical interpretation, in light of what was previously discussed. Consider the same setting from Figure 20.2, but now we consider curves of the form $f + \overline{v}h + \mu h^2 = k$. This is illustrated in Figure 20.3.

Optimising the augmented Lagrangian function amounts to finding the curve $f + \overline{v}h + \mu h^2 = k$ in which $v(\epsilon) = k$. The expression for k can be conveniently rewritten as $f = -\mu \left[h + (\overline{v}/2\mu)\right]^2 + \left[k + (\overline{v}^2/4\mu)\right]$, exposing that f is a parabola shifted by $h = -\overline{v}/2\mu$.

20.2.1 Augmented Lagrangian method of multipliers

We can employ an unconstrained optimisation method to solve the augmented Lagrangian function

$$L_\mu(x, v) = f(x) + \sum_{i=1}^{l} v_i h_i(x) + \mu \sum_{i=1}^{l} h_i(x)^2,$$

which amount to rely on strong duality and search for KKT points (or primal-dual pairs) $(\overline{x}, \overline{v})$ by iteratively operating in the primal (x) and dual (v) spaces. In particular, the strategy consists of

1. *Primal space:* optimise $L_\mu(x, v^k)$ using an unconstrained optimisation method
2. *Dual space:* perform a dual variable update step retaining the optimality condition $\nabla_x L(x^{k+1}, v^k) = \nabla_x L(x^{k+1}, v^{k+1}) = 0$

This strategy is akin to applying the subgradient method to solving the augmented Lagrangian dual. The update step for the dual variable is given by

$$\overline{v}^{k+1} = \overline{v}^k + 2\mu h(\overline{x}^{k+1}).$$

The motivation for the dual step update stems from the following observation:

1. $h(\overline{x}^k)$ is a subgradient of $L_\mu(x, v)$ at $\overline{x}^k$ for any v.
2. The step size is devised such that the optimality condition of the Lagrangian function is retained, i.e., $\nabla_x L(\overline{x}^k, \overline{v}^{k+1}) = 0$.

Part 2 refers to the following:

$$\begin{aligned}
\nabla_x L(\overline{x}^k, \overline{v}^{k+1}) &= \nabla f(\overline{x}^k) + \sum_{i=1}^{l} \overline{v}_i^{k+1} \nabla h_i(\overline{x}^k) = 0 \\
&= \nabla f(\overline{x}^k) + \sum_{i=1}^{l} (\overline{v}_i^k + 2\mu h_i(\overline{x}^k)) \nabla h_i(\overline{x}^k) = 0 \\
&= \nabla f(\overline{x}^k) + \sum_{i=1}^{l} \overline{v}_i^k \nabla h_i(\overline{x}^k) + \sum_{i=1}^{l} 2\mu h_i(\overline{x}^k) \nabla h_i(\overline{x}^k) = \nabla_x L_\mu(\overline{x}^k, \overline{v}^k) = 0.
\end{aligned}$$

That is, by employing $\overline{v}^{k+1} = \overline{v}^k + 2\mu h(\overline{x}^{k+1})$ one can retain optimality in the dual variable space for the Lagrangian function from the optimality conditions of the penalised functions, which is a condition for $\overline{x}$ to be a KKT point.

Algorithm 22 summarises the augmented Lagrangian method of multipliers (ALMM).

Algorithm 18 Augmented Lagrangian method of multipliers (ALMM)

1: **initialise.** tolerance $\epsilon > 0$, initial dual solution v^0, iteration count $k = 0$
2: **while** $|h(\overline{x}^k)| > \epsilon$ **do**
3: $\quad \overline{x}^{k+1} = \arg\min L_\mu(x, \overline{v}^k)$
4: $\quad \overline{v}^{k+1} = \overline{v}^k + 2\mu h(\overline{x}^{k+1})$
5: $\quad k = k + 1$
6: **end while**
7: **return** x^k.

The method can be specialised such that μ is individualised for each constraint and updated proportionally to the observed infeasibility $h_i(x)$. Such a procedure is still guaranteed to converge, as the requirement in Theorem 20.1 that $\mu \to \infty$ is still trivially satisfied.

One important point about the augmented Lagrangian method of multipliers is that linear convergence is to be expected, due to the gradient-like step taken to find optimal dual variables. This is often the case with traditional Lagrangian duality based approaches.

20.2.2 Alternating direction method of multipliers - ADMM

ADMM is a distributed version of the augmented Lagrangian method of multipliers, and is more suited to large problems with a decomposable structure.

Consider the problem P to be of the following form:

$$\begin{aligned}(P): \min.\ & f(x) + g(y) \\ \text{s.t.: } & Ax + By = c.\end{aligned}$$

We would like to be able to solve the problem separately for x and y, which could, in principle be achieved using ALMM. However, the consideration of the penalty term prevents the problem from being completely separable. To see that, let

$$\phi(x, y, v) = f(x) + g(y) + v^\top(c - Ax - By) + \mu(c - Ax - By)^2$$

be the augmented Lagrangian function. One can notice that the penalty term $\mu(c - Ax - By)^2$ prevents the separation of the problem in terms of the x and y variables. However, separability can be recovered is one employs a coordinate descent approach in which three blocks are considered: x, y, and v. The ADMM is summarised in Algorithm 23.

Algorithm 19 ADMM

1: **initialise.** tolerance $\epsilon > 0$, initial dual and primal solutions v^0 and y^0, $k = 0$
2: **while** $|c - A\overline{x}^k - B\overline{y}^k|$ and $||y^{k+1} - y^k|| > \epsilon$ **do**
3: $\quad \overline{x}^{k+1} = \arg\min \phi_\mu(x, \overline{y}^k, \overline{v}^k)$
4: $\quad \overline{y}^{k+1} = \arg\min \phi_\mu(\overline{x}^{k+1}, y, \overline{v}^k)$
5: $\quad \overline{v}^{k+1} = \overline{v}^k + 2\mu(c - A\overline{x}^{k+1} - B\overline{y}^{k+1})$
6: $\quad k = k + 1$
7: **end while**
8: **return** (x^k, y^k).

One important feature regarding ADMM is that the coordinate descent steps are taken in a cyclic order, not requiring more than one (x, y) update step. Variants consider more than one of these steps, but no clear benefit in practice has been observed. Moreover, μ can be updated according to the amount of infeasibility observed at iteration k, but no generally good update rule is known.

ADMM is particularly relevant as a method for (un)constrained problems in which it might expose a structure that can be exploited, such as having in some of the optimisation problems (in Lines 3 and 4 in Algorithm 23) that might have solutions in closed forms.

Chapter 21

Barrier methods

21.1 Barrier functions

In essence, barrier methods also use the notion of using proxies for the constraints in the objective function, so that an unconstrained optimisation problem can be solved instead. However, the concept of barrier functions is different than penalty functions in that they are defined to *prevent* the solution search method from leaving the feasible region, which is why some of these methods are also called *interior point methods.*

Consider the primal problem P being defined as

$$\begin{aligned}(P): \quad \text{min.} \quad & f(x) \\ \text{s.t.:} \quad & g(x) \leq 0 \\ & x \in X.\end{aligned}$$

We define the *barrier problem* BP as

$$\begin{aligned}(BP): \quad \inf_{\mu} \quad & \theta(\mu) \\ \text{s.t.:} \quad & \mu > 0\end{aligned}$$

where $\theta(\mu) = \inf_x \{f(x) + \mu B(x) : g(x) < 0, x \in X\}$ and $B(x)$ is a *barrier function.* The barrier function is such that its value approaches $+\infty$ as the boundary of the region $\{x : g(x) \leq 0\}$ is approached from its interior. Notice that, in practice, it means that the constraint $g(x) < 0$ can be dropped, as they are automatically enforced by the barrier function.

The barrier function $B : \mathbb{R}^m \to \mathbb{R}$ is such that

$$B(x) = \sum_{i=1}^{m} \phi(g_i(x)), \text{ where } \begin{cases} \phi(y) \geq 0, & \text{if } y < 0; \\ \phi(y) = +\infty, & \text{when } y \to 0^-. \end{cases} \tag{21.1}$$

Examples of barrier functions that impose this behaviour are

- $B(x) = -\sum_{i=1}^{m} \frac{1}{g_i(x)}$
- $B(x) = -\sum_{i=1}^{m} \ln(\min\{1, -g_i(x)\})$

Perhaps the most important barrier function is the *Frisch's log barrier function*, used in the highly successful primal-dual interior point methods. We will describe its use later. The log barrier is defined as

$$B(x) = -\sum_{i=1}^{m} \ln(-g_i(x)).$$

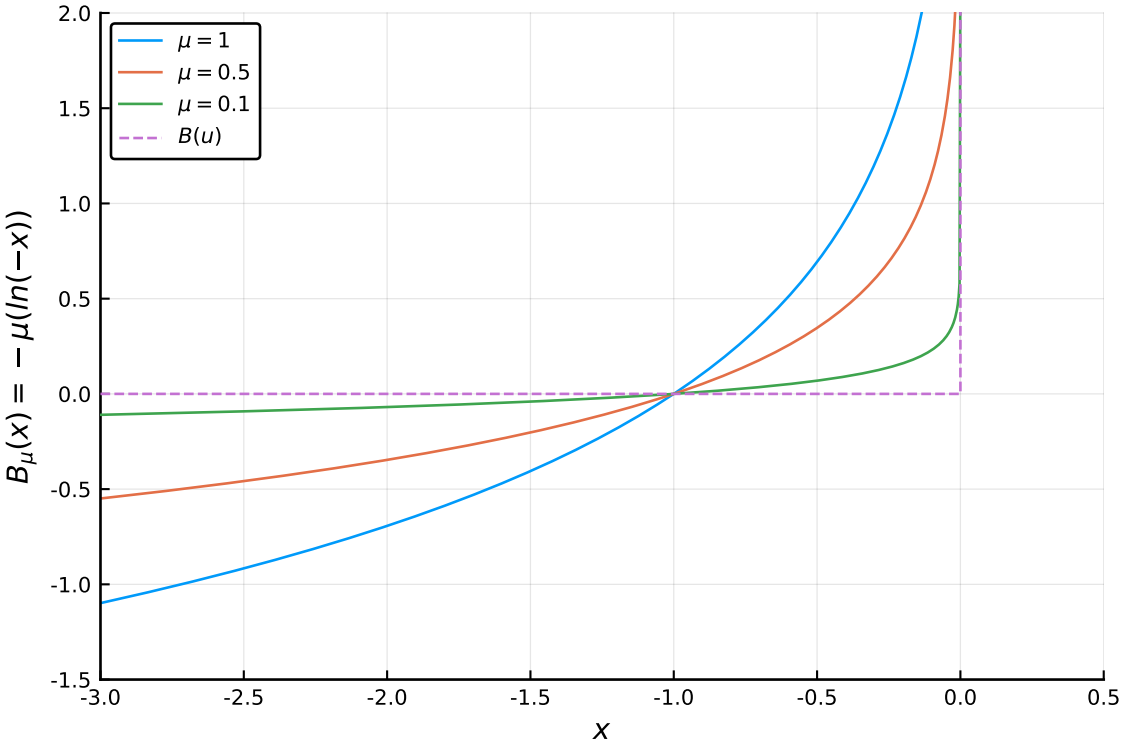


Figure 21.1: The barrier function for different values of μ

Figure 21.1 illustrates the behaviour of the barrier function. Ideally, the barrier function $B(x)$ has the role of an *indicator function*, which is zero for all feasible solutions $x \in \{x : g(x) < 0\}$ but assume infinite value if a solution is at the boundary $g(x) = 0$ or outside the feasible region. This is illustrated in the dashed line in Figure 21.1. The barrier functions for different values of barrier term μ illustrate how the log barrier mimics this behaviour, becoming more and more pronounced as μ decreases.

21.2 The barrier method

In a similar nature to what was developed for penalty methods, we can devise a solution method that successively solves the barrier problem $\theta(\mu)$ for decreasing values of the barrier term μ.

We start by stating the result that guarantees convergence of the barrier method.

Theorem 21.1 (Convergence of barrier methods)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$ be continuous functions and $X \in \mathbb{R}^n$ a nonempty closed set in problem P. Suppose $\{x \in \mathbb{R}^n : g(x) < 0, x \in X\}$ is not empty. Let $\overline{x}$ be the optimal solution of P such that, for any neighbourhood $N_\epsilon(\overline{x}) = \{x : ||x - \overline{x}|| \leq \epsilon\}$, there exists $x \in X \cap N_\epsilon$ for which $g(x) < 0$. Then*

$$\min\{f(x) : g(x) \leq 0, x \in X\} = \lim_{\mu \to 0^+} \theta(\mu) = \inf_{\mu > 0} \theta(\mu).$$

Letting $\theta(\mu) = f(x_\mu) + \mu B(x_\mu)$, where $B(x)$ is a barrier function as described in (21.1), $x_\mu \in X$ and $g(x_\mu) < 0$, the limit of $\{x_\mu\}$ is optimal to P and $\mu B(x_\mu) \to 0$ as $\mu \to 0^+$.

Proof. First, we show that $\theta(\mu)$ is a nondecreasing function in μ. For $\mu > \lambda > 0$ and x such that $g(x) < 0$ and $x \in X$, we have

$$\begin{aligned} f(x) + \mu B(x) &\geq f(x) + \lambda B(x) \\ \inf_x \{f(x) + \mu B(x)\} &\geq \inf_x \{f(x) + \lambda B(x)\} \\ \theta(\mu) &\geq \theta(\lambda). \end{aligned}$$

From these, we conclude that $\lim_{\mu \to 0^+} \theta(\mu) = \inf \{\theta(\mu) : \mu > 0\}$. Now, let $\epsilon > 0$. As $\overline{x}$ is optimal, by assumption there exists some $\hat{x} \in X$ with $g(\hat{x}) < 0$ such that $f(\overline{x}) + \epsilon > f(\hat{x})$. Then, for $\mu > 0$ we have

$$f(\overline{x}) + \epsilon + \mu B(\hat{x}) > f(\hat{x}) + \mu B(\hat{x}) \geq \theta(\mu).$$

Letting $\mu \to 0^+$, it follows that $f(\overline{x}) + \epsilon \geq \lim_{\mu \to 0^+} \theta(\mu)$, which implies $f(\overline{x}) \geq \lim_{\mu \to 0^+} \theta(\mu) = \inf \{\theta(\mu) : \mu > 0\}$. Conversely, since $B(x) \geq 0$ and $g(x) < 0$ for some $\mu > 0$, we have

$$\begin{aligned} \theta(\mu) &= \inf \{f(x) + \mu B(x) : g(x) < 0, x \in X\} \\ &\geq \inf \{f(x) : g(x) < 0, x \in X\} \\ &\geq \inf \{f(x) : g(x) \leq 0, x \in X\} = f(\overline{x}). \end{aligned}$$

Thus $f(\overline{x}) \leq \lim_{\mu \to 0^+} \theta(\mu) = \inf \{\theta(\mu) : \mu > 0\}$. Therefore, $f(\overline{x}) = \lim_{\mu \to 0^+} \theta(\mu) = \inf \{\theta(\mu) : \mu > 0\}$. □

The proof has three main steps. First, we show that $\theta(\mu)$ is a nondecreasing function in μ, which implies that $\lim_{\mu \to 0^+} \theta(\mu) = \inf \{\theta(\mu) : \mu > 0\}$. This can be trivially shown as only feasible solutions x are required to be considered.

Next, we show the convergence of the barrier method by showing that $\inf_{\mu > 0} \theta(\mu) = f(\overline{x})$, where $\overline{x} = \arg\min \{f(x) : g(x) \leq 0, x \in X\} = \lim_{\mu \to 0^+} \theta(\mu) = \inf_{\mu > 0} \theta(\mu)$. The optimality of $\overline{x}$ implies that $f(\hat{x}) - f(\overline{x}) < \epsilon$ for feasible $\hat{x}$ and $\epsilon > 0$. Moreover, $B(\hat{x}) \geq 0$ by definition. In the last part, we use the argument that including the boundary can only improve the objective function value, leading to the last inequality. It is worth highlighting that, to simplify the proof, we have assumed that the barrier function has the form described in (21.1). However, a proof in the veins of Theorem 21.1 can be still be developed for the Frisch log barrier (for which $B(x)$ is not necessarily nonnegative) since, essentially, (21.1) only needs to be observed in a neighbourhood of $g(x) = 0$.

The result in Theorem 21.1 allows to design a optimisation methods that, starting from a strictly feasible (interior) solution, is based on successively reducing the barrier term until a solution with arbitrarily small barrier term is obtained. Algorithm 22 present a pseudo code for such method.

Algorithm 20 Barrier method

1: **initialise.** $\epsilon > 0, x^0 \in X$ with $g(x^k) < 0, \mu^k, \beta \in (0,1), k = 0$.
2: **while** $\mu^k B(x^k) > \epsilon$ **do**
3: $\quad \overline{x}^{k+1} = \arg\min \left\{ f(x) + \mu^k B(x) : x \in X \right\}$
4: $\quad \mu^{k+1} = \beta \mu^k$, $k = k + 1$
5: **end while**
6: **return** x^k.

One important aspect to notice is that starting the algorithm requires a strictly feasible point, which in some applications, might be challenging to be obtained. This characteristic is what renders the name *interior point methods* for this class of algorithms.

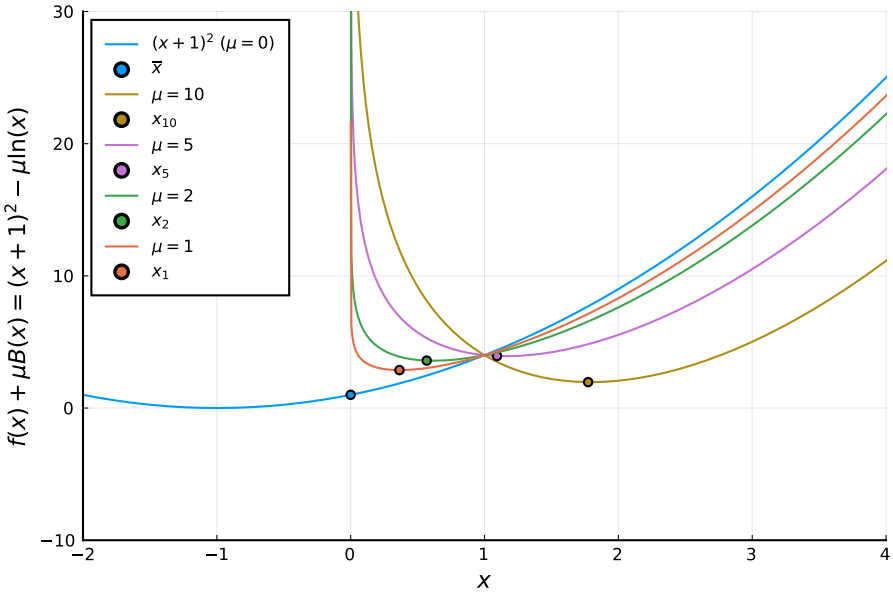


Figure 21.2: Example 1: solving a one-dimensional problem with the barrier method

Consider the following example. Let $P = \{(x+1)^2 : x \geq 0\}$. Let us assume that we use the barrier function $B(x) = -\ln(x)$. Then, we unconstrained barrier problem becomes

$$(BP) : \min_x \ (x+1)^2 - \mu \ln(x). \tag{21.2}$$

Since this is a convex function, the first order condition $f'(x) = 0$ is necessary and sufficient for optimality. Thus, solving $2(x+1) - \frac{\mu}{x} = 0$ we obtain the positive root and unique solution $x_\mu = \frac{-1}{2} + \frac{\sqrt{4+8\mu}}{4}$. Figure X shows the behaviour of the problem as μ converges to zero. As can be seen, as $\mu \to 0$, the optimal solution x_μ converges to the constrained optimum $\overline{x} = 0$.

We now consider a more involved example. Let us consider the problem

$$P = \min. \ \{(x_1 - 2)^4 + (x_1 - 2x_2)^2 : x_1^2 - x_2 \leq 0\}$$

with $B(x) = -\frac{1}{x_1^2 - x_2}$. We implemented Algorithm 22 and solved it with two distinct values for the penalty term μ and reduction term β. Figure 21.2 illustrate the trajectory of the algorithm with each parametrisation, exemplifying how these can affect the convergence of the method.

21.3 Interior point method for LP/QP problems

Perhaps ironically, the most successful applications of barrier methods in terms of efficient implementations are devoted to solving linear and quadratic programming (LP/QP) problems. The primal-dual interior point method has become in the last decade the algorithm of choice for many applications involving large-scale LP/QP problems.

To see how barrier methods can be applied to LP problems, consider the following primal/dual

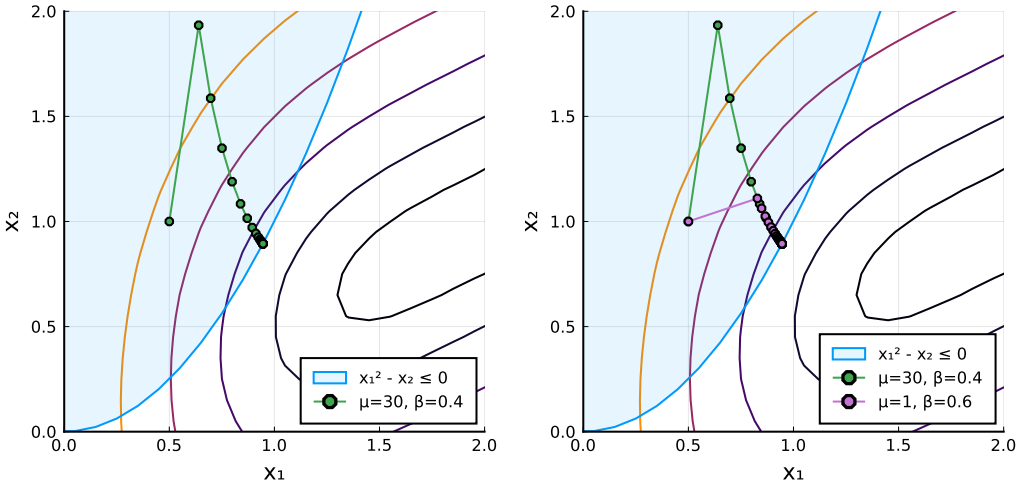


Figure 21.3: The trajectory of the barrier method for problem P. Notice how the parameters influence the trajectory and number of iterations. The parameters on the left require 27 iterations while those on the right require 40 iterations for convergence.

pair formed by a LP primal P

$$\begin{aligned}(P) \ : \ \text{min.} \ & c^\top x \\ \text{s.t.:} \ & Ax = b \ : v \\ & x \geq 0 \ : u\end{aligned}$$

and its respective dual formulation D

$$\begin{aligned}(D) \ : \ \text{max.} \ & b^\top v \\ \text{s.t.:} \ & A^\top v + u = c \\ & u \geq 0, v \in \mathbb{R}^m.\end{aligned}$$

The optimal solution $(\overline{x}, \overline{v}, \overline{u}) = \overline{w}$ is such that it satisfies KKT conditions of P, given by

$$\begin{aligned}& Ax = b, \ x \geq 0 \\ & A^\top v + u = c, \ u \geq 0, \ v \in \mathbb{R}^m \\ & u^\top x = 0.\end{aligned}$$

These are going to be useful as a reference for the next developments. We start by considering the *barrier problem* for P by using the logarithmic barrier function to represent the condition $x \geq 0$. Thus, the barrier problem BP can be defined as

$$\begin{aligned}(BP) \ : \ \text{min.} \ & c^\top x - \mu \sum_{i=1}^{n} \ln(x_j) \\ \text{s.t.:} \ & Ax = b.\end{aligned}$$

Notice that this problem is a strictly-convex equality constrained problem that is suitable to be solved using the constrained variant of Newton's method (which simply consists of employing Newton's method to solve the KKT conditions of equality constrained problems). Moreover, observe

that the KKT conditions of BP are

$$\begin{aligned} &Ax = b,\ x > 0 \\ &A^\top v = c - \mu \left(\frac{1}{x_1}, \ldots, \frac{1}{x_n} \right) \end{aligned}$$

Notice that, since $\mu > 0$ and $x > 0$, $u = \mu \left(\frac{1}{x_1}, \ldots, \frac{1}{x_n} \right)$ serves as an estimate for the Lagrangian dual variables. To further understand the relationship between the optimality conditions of BP and P, let us define $X \in \mathbb{R}^{n \times n}$ and $U \in \mathbb{R}^{n \times n}$ as

$$X = \mathbf{diag}(x) = \begin{bmatrix} \ddots & & \\ & x_i & \\ & & \ddots \end{bmatrix} \text{ and } U = \mathbf{diag}(u) = \begin{bmatrix} \ddots & & \\ & u_i & \\ & & \ddots \end{bmatrix}$$

and let $e = [1, \ldots, 1]^\top$ be a vector of ones of suitable dimension. We can rewrite the KKT conditions of BP as

$$Ax = b,\ x > 0 \tag{21.3}$$

$$A^\top v + u = c \tag{21.4}$$

$$u = \mu X^{-1} e \ \Rightarrow\ XUe = \mu e. \tag{21.5}$$

Notice how the condition (21.5) resembles the complementary slackness from P, but *relaxed* to be μ instead of zero. This is why this system is often referred to as the *perturbed KKT system*.

Theorem 21.1[1] guarantees that $w_\mu = (x_\mu, v_\mu, u_\mu)$ approaches the optimal primal-dual solution of P as $\mu \to 0^+$. The trajectory formed by successive solutions $\{w_\mu\}$ is called the *central path*, which is due to the interiority enforced by the barrier function. When the barrier term μ is large enough, the solution of the barrier problem is close to the analytic centre of the feasibility set. The analytic centre of a polyhedral set $S = \{x \in \mathbb{R}^n : Ax \leq b\}$ is given by

$$\begin{aligned} \max_x.\ & \prod_{i=1}^{m} (b_i - a_i^\top x) \\ \text{s.t.: } & x \in X, \end{aligned}$$

which corresponds to finding the point of maximum distance to each of the hyperplanes forming the polyhedral set. This is equivalent to the convex problem

$$\begin{aligned} \min_x.\ & \sum_{i=1}^{m} -\ln(b_i - a_i^\top x) \\ \text{s.t.: } & x \in X, \end{aligned}$$

and thus justifying the nomenclature.

One aspects should be observed for defining stopping criterion. Notice that the term $u^\top x$ is such

[1] In fact, we require a slight variant fo Theorem 1 that allow for $B(x) \geq 0$ only being required in a neighbourhood of $g(x) = 0$.

that it measures the duality gap at a given solution. That is, notice that

$$\begin{aligned} c^\top x &= (A^\top v + u)^\top x \\ &= (A^\top v)^\top x + u^\top x \\ &= v^\top (Ax) + u^\top x \\ c^\top x - b^\top v = u^\top x &= \sum_{i=1}^{n} u_i x_i = \sum_{i=1}^{n} \left(\frac{\mu}{x_i}\right) x_i = n\mu. \end{aligned}$$

which gives the *total slack violation* that can be used to determine the convergence of the algorithm.

21.3.1 Primal/dual path-following interior point method

The primal/dual path following interior point method (IPM) is the specialised version of the setting described earlier for solving LP/QP problems.

It consists of building upon employing logarithmic barriers to LP/QP problems and solving the system (21.3) - (21.5) using Newton's method. However, instead of solving the problem to optimality for each μ, only a *single* Newton step is taken before the barrier term μ is reduced.

Suppose we start with a $\overline{\mu} > 0$ and a $w^k = (x^k, v^k, u^k)$ sufficiently close to $w_{\overline{\mu}}$. Then, for a sufficiently small $\beta \in (0, 1)$, $\beta\overline{\mu}$ will lead to a w^{k+1} sufficiently close to $w_{\beta\overline{\mu}}$. Figure 21.4 illustrates this effect, showing how a suboptimal solution x^k do not necessarily need to be in the central path (denoted by the dashed line) to guarantee convergence, as long as they are guaranteed to remain within the some neighbourhood $N_\mu(\theta)$ of the central path.

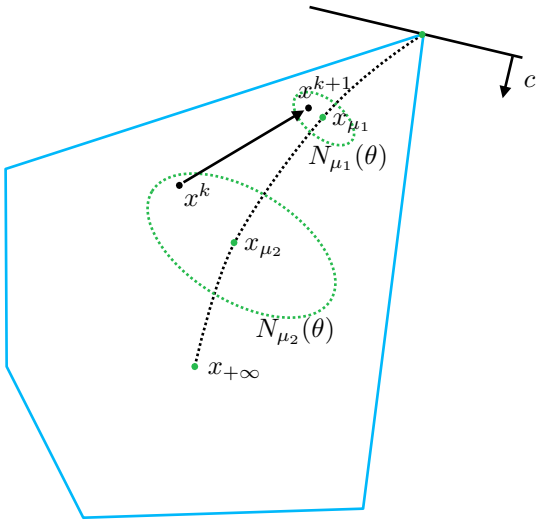


Figure 21.4: an illustrative representation of the central path and how the IPM follows it approximately.

For example, let $N_\mu(\theta) = ||X_\mu U_\mu e - \mu e|| \leq \theta\mu$. Then, by selecting $\beta = 1 - \frac{\sigma}{\sqrt{n}}$, $\sigma = \theta = 0.1$, and $\mu^0 = (x^\top u)/n$, successive Newton steps are guaranteed to remain within $N_\mu(\theta)$.

To see how the setting works, let the perturbed KKT system (21.3) – (21.5) for each $\hat{\mu}$ be denoted as $H(w) = 0$. Let $J(\overline{w})$ be the Jacobian of $H(w)$ at $\overline{w}$.

Applying Newton's method to solve $H(w) = 0$ for $\overline{w}$, we obtain

$$J(\overline{w})d_w = -H(\overline{w}) \tag{21.6}$$

where $d_w = (w - \overline{w})$. By rewriting $d_w = (d_x, d_v, d_u)$, (22.1) can be equivalently stated as

$$\begin{bmatrix} A & 0^\top & 0 \\ 0 & A^\top & I \\ \overline{U} & 0^\top & \overline{X} \end{bmatrix} \begin{bmatrix} d_x \\ d_v \\ d_u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \hat{\mu}e - \overline{X}\,\overline{U}e \end{bmatrix}. \tag{21.7}$$

The system (21.7) is often referred to as the Newton's system.

In practice, the updates incorporate primal and dual infeasibility, which precludes the need of additional mechanisms to guarantee primal and dual feasibility throughout the algorithm. This can be achieved with a simple modification in the Newton system, rendering the direction update step

$$\begin{bmatrix} A & 0^\top & 0 \\ 0 & A^\top & I \\ U^k & 0^\top & X^k \end{bmatrix} \begin{bmatrix} d_x^{k+1} \\ d_v^{k+1} \\ d_u^{k+1} \end{bmatrix} = - \begin{bmatrix} Ax^k - b \\ A^\top v^k + u^k - c \\ X^k U^k e - \mu^{k+1} e \end{bmatrix}, \tag{21.8}$$

To see how this still leads to primal and dual feasible solutions, consider the primal residuals (i.e., the amount of infeasibility) as $r_p(x, u, v) = Ax - b$ and the dual residuals $r_d(x, u, v) = A^\top v + u - c$. Now, let $r(w) = r(x, u, v) = (r_p(x, u, v), r_d(x, u, v))$, recalling that $w^k = (x, v, u)$. The optimality conditions can be expressed as requiring that the residuals must vanish, that is $r(\overline{w}) = 0$.

Now, consider the first-order Taylor approximation for r at w for a step d_w

$$r(w + d_w) \approx r(w) + Dr(w)d_w,$$

where $Dr(w)$ is the derivative of r evaluated at w, which is given by the two first rows of the Newton system (21.7). The step d_w for which the residue vanishes is

$$Dr(w)d_w = -r(w), \tag{21.9}$$

which is the same as (22.1) without the bottom equation. Now, if we consider the directional derivative of the square of the norm of r in the direction d_w, we obtain

$$\frac{d}{dt}||r(w + td_w)||_2^2 \bigg|_{t \to 0^+} = 2r(w)^\top Dr(w)d_w = -2r(w)^\top r(w), \tag{21.10}$$

which is strictly decreasing. That is, the step d_w is such that it will make the residual decrease and eventually become zero. From that point onwards, the Newton system will take the form of (21.7).

The algorithm proceeds by iteratively solving the system (21.8) with $\mu^{k+1} = \beta\mu^k$ with $\beta \in (0, 1)$ until $n\mu^k$ is less than a specified tolerance. Algorithm 23 summarises a simplified form of the IPM.

Algorithm 21 Interior point method (IPM) for LP

1: **initialise.** primal-dual feasible w^k, $\epsilon > 0$, μ^k, $\beta \in (0, 1)$, $k = 0$.
2: **while** $n\mu = c^\top x^k - b^\top v^k > \epsilon$ **do**
3: compute $d_{w^{k+1}} = (d_{x^{k+1}}, d_{v^{k+1}}, d_{u^{k+1}})$ using (21.8) and w^k.
4: $w^{k+1} = w^k + d_{w^{k+1}}$
5: $\mu^{k+1} = \beta\mu^k$, $k = k + 1$
6: **end while**
7: **return** w^k.

Figure 21.5 illustrates the behaviour of the IPM when employed to solve the linear problem

$$\begin{aligned} \text{min. } & x_1 + x_2 \\ \text{s.t.: } & 2x_1 + x_2 \geq 8 \\ & x_1 + 2x_2 \geq 10, \\ & x_1, x_2 \geq 0 \end{aligned}$$

considering two distinct initial penalties μ. Notice how higher penalty values enforce a more central convergence of the method. Some points are worth noticing concerning Algorithm 23.

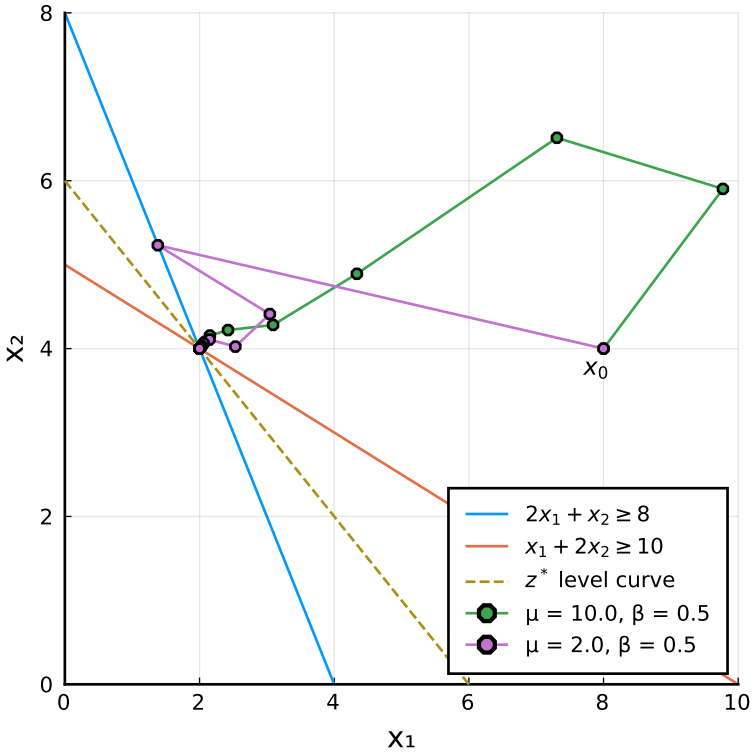


Figure 21.5: IPM applied to a LP problem with two different barrier terms

First, notice that in Line 4, a fixed step size is considered. A line search can be incorporated to prevent infeasibility and improve numerical stability. Typically, it is used $\lambda_i^k = \min\left\{\alpha, -\frac{x_i^k}{d_i^k}\right\}$ with $\alpha < 1$ but close to 1.

Also, even though the algorithm is initialised with a feasible solution w^k, this might in practice not be necessary. Implementations of the infeasible IPM method can efficiently handle primal and dual infeasibility.

Under specific conditions, the IPM can be shown to have complexity of $O(\sqrt{n}\ln(1/\epsilon))$, which is polynomial and of much better worst-case performance than the simplex method, which makes it the algorithm of choice for solving large-scale LPs. Another important advantage is that IPM can be modified with little effort to solve a wider class of problems under the class of *conic optimisation problems*.

Predictor-corrector methods are variants of IPM that incorporate a two-phase direction calculation using a *predicted* direction d_w^{pred}, calculated by setting $\mu = 0$ and a *correcting* direction, which is computed considering the impact that d_w^{cor} would have in the term $\overline{X}\,\overline{U}e$.

Let $\Delta X = \mathbf{diag}(d_x^{\text{pred}})$ and $\Delta U = \mathbf{diag}(d_u^{\text{pred}})$. Then

$$\begin{aligned}(X + \Delta X)(U + \Delta U)e &= XUe + (U\Delta X + X\Delta U)e + \Delta X \Delta U e \\ &= XUe + (0 - XUe) + \Delta X \Delta U e \\ &= \Delta X \Delta U e \end{aligned} \tag{21.11}$$

Using the last equation (21.11), the corrector Newton step becomes $\overline{U}d_x + \overline{X}d_u = \hat{\mu}e - \Delta X \Delta U e$. Finally, d_w^k is set to be a combination of d_w^{pred} and d_w^{cor}.

Chapter 22

Primal methods

22.1 The concept of feasible directions

Feasible direction methods are a class of methods that incorporate both improvement and feasibility requirements when devising search directions. As feasibility is also observed throughout the solution process, they are also referred to as *primal methods*. However, depending. on the geometry of the feasible region, it might be so that the method allow for some infeasibility in the course of the algorithm, as we will see later.

An *improving feasible direction* can be defined as follows.

Definition 22.1. *Consider the problem min.* $\{f(x) : x \in S\}$ *with* $f : \mathbb{R}^n \to \mathbb{R}$ *and* $\emptyset \neq S \subseteq \mathbb{R}^n$. *A vector* d *is a feasible direction at* $x \in S$ *if exists* $\delta > 0$ *such that* $x + \lambda d \in S$ *for all* $\lambda \in (0, \delta)$. *Moreover,* d *is an improving feasible direction at* $x \in S$ *if there exists a* $\delta > 0$ *such that* $f(x+\lambda d) < f(x)$ *and* $x + \lambda d \in S$ *for* $\lambda \in (0, \delta)$.

The key feature of feasible direction methods is the process of deriving such directions and associated step sizes that retain feasibility, even if approximately. Similarly to the other methods we have discussed in the past lectures, these methods progress following two basic steps:

1. Obtain an *improving feasible direction* d^k and a step size λ^k;
2. Make $x^{k+1} = x^k + \lambda^k d^k$.

As a general rule, for a feasible direction method to perform satisfactorily, it must be that the calculation of the directions d^k and step sizes λ^k are simple enough. Often, these steps can be reduced to closed forms or, more frequently, solving linear or quadratic programming problems, or even from posing modified Newton systems.

22.2 Conditional gradient - the Frank-Wolfe method

The conditional gradient method is named as such due to the direction definition step, in which the direction d is selected such that the angle between the gradient $\nabla f(x)$ and d is as close to 180° degrees as the feasible region S allows.

Recall that, if $\nabla f(x^k)$ is a *descent direction*, then

$$\nabla f(x^k)^\top (x - x^k) < 0 \text{ for } x \in S.$$

A straightforward way to obtain improving feasible directions $d = (x - x^k)$ is by solving the *direction search problem DS* of the form

$$(DS) : \min. \quad \left\{\nabla f(x^k)^\top (x - x^k) : x \in S\right\}.$$

Problem DS consists of finding the furtherest feasible point in the direction of the gradient, that is we move in the direction of the gradient, under condition that we stop if the line search mandates so, or that the search reach at the boundary of the feasible region. This is precisely what gives the name *conditional gradient*.

By letting $\overline{x}^k = \arg\min_{x \in S} \left\{\nabla f(x^k)^\top (x - x^k)\right\}$ and obtaining $\lambda^k \in (0, 1]$ employing a line search, the method iterates making

$$x^{k+1} = x^k + \lambda^k (\overline{x}^k - x^k).$$

One important condition to observe is that λ^k has to be constrained such that $\lambda \in (0, 1]$ to guarantee feasibility, as $\overline{x}^k$ is feasible by definition. Also, notice that the condition $\nabla f(x^k) = 0$ might never be achieved, since it might be so that the unconstrained optimum is outside the feasible region S. In that case, after two successive iterations we will observe that $x^k = x^{k-1}$ and thus that $d^k = 0$. This eventual stall of the algorithm will happen at a point x^k satisfying first-order (constrained) optimality conditions. Therefore, the term $\nabla f(x)^\top d^k$ will become zero regardless whether the minimum of them function belongs to S, and is hence used as the stopping condition of the algorithm. Algorithm 22 summarises the Frank-Wolfe method.

Algorithm 22 Franke-Wolfe method

1: **initialise.** $\epsilon > 0, x^0 \in S, k = 0$.
2: **while** $\nabla |f(x)^\top d^k| > \epsilon$ **do**
3: $\quad \overline{x}^k = \arg\min \left\{\nabla f(x^k)^\top d : x \in S\right\}$
4: $\quad d^k = \overline{x}^k - x^k$
5: $\quad \lambda^k = \arg\min_\lambda \left\{f(x^k + \lambda d^k) : 0 \leq \lambda \leq \overline{\lambda}\right\}$
6: $\quad x^{k+1} = x^k + \lambda^k d^k$
7: $\quad k = k + 1$
8: **end while**
9: **return** x^k

Notice that, for a polyhedral feasibility set, the subproblems are linear programming problems, meaning that the Frank-Wolfe method can be restarted fairly efficiently using dual simplex at each iteration.

Figure 22.1 shows the employment of the FW method for optimising a nonlinear function within a polyhedral feasibility set. We consider the problem

$$\begin{aligned} \text{min. } & e^{-(x_1-3)/2} + e^{(4x_2+x_1-20)/10} + e^{(-4x_2+x_1)/10} \\ \text{s.t.: } & 2x_1 + 3x_2 \leq 8 \\ & x_1 + 4x_2 \leq 6, \qquad\qquad x_1, x_2 \geq 0 \end{aligned}$$

starting from $(0, 0)$ and using an exact line search to set step sizes $\lambda \in [0, 1]$. Notice that the method can be utilised with inexact line searches as well.

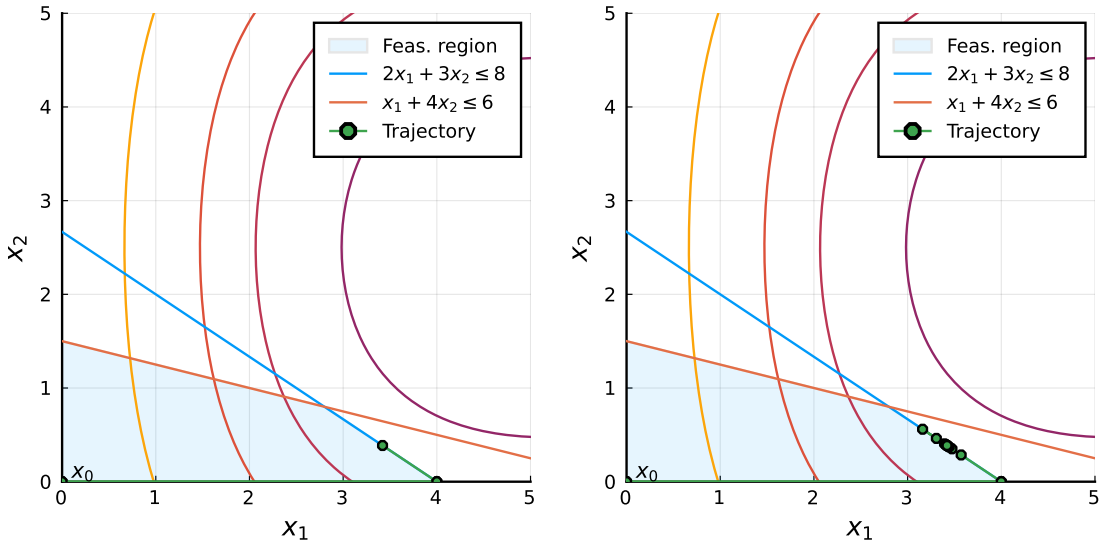


Figure 22.1: The Frank-Wolfe method applied to a problem with linear constraints. The algorithm takes 2 steps using an exact line search (left) and 15 with Armijo line search (right).

22.3 Sequential quadratic programming

Sequential quadratic programming (SQP) is a method inspired by the idea that the KKT system of a nonlinear problem can be solved using Newton's method. It consists perhaps of the most general method in terms of considering both nonlinear constraints and nonlinear objective function.

To see how that works, let us first consider an equality constraint problem P as

$$P = \min.\ \{f(x) : h(x) = 0, i = 1, \ldots, l\}.$$

The KKT conditions for P are given by the system $W(x, v)$ where

$$W(x, v) = \begin{cases} \nabla f(x) + \sum_{i=1}^{l} v_i \nabla h_i(x) = 0 \\ h_i(x) = 0,\ i = 1, \ldots, l \end{cases}$$

Using the Newton(-Raphson) method, we can solve $W(x, v)$. Starting from (x^k, v^k), we can solve $W(x, v)$ by successively employing Newton steps of the form

$$W(x^k, v^k) + \nabla W(x^k, v^k) \begin{bmatrix} x - x^k \\ v - v^k \end{bmatrix} = 0. \tag{22.1}$$

Upon closer inspection, one can notice that the term $\nabla W(x, v)$ is given by

$$\nabla W(x^k, v^k) = \begin{bmatrix} \nabla^2 L(x^k, v^k) & \nabla h(x^k)^\top \\ \nabla h(x^k) & 0 \end{bmatrix},$$

where

$$\nabla^2 L(x^k, v^k) = \nabla^2 f(x^k) + \sum_{i=1}^{l} v_i^k \nabla^2 h_i(x^k)$$

is the *Hessian of the Lagrangian* function

$$L(x, v) = f(x) + v^\top h(x)$$

at x^k. Now, setting $d = (x - x^k)$, we can rewrite (22.1) as

$$\nabla^2 L(x^k, v^k)d + \nabla h(x^k)^\top v = -\nabla f(x^k) \tag{22.2}$$

$$\nabla h(x^k)d = -h(x^k), \tag{22.3}$$

which can be repeatedly solved until

$$||(x^k, v^k)^\top - (x^{k-1}, v^{k-1})^\top|| = 0,$$

i.e., convergence, is observed. Then, (x^k, v^k) is a KKT point by definition.

This is fundamentally the underlying idea of SQP, however the approach is taken under a more specialised setting. Instead of relying on Newton steps, we resort on successively solving quadratic subproblems of the form

$$QP(x^k, v^k) : \text{min.} \quad f(x^k) + \nabla f(x^k)^\top d + \frac{1}{2} d^\top \nabla^2 L(x^k, v^k)d \tag{22.4}$$

$$\text{s.t.: } h_i(x^k) + \nabla h_i(x^k)^\top d = 0, i = 1, \ldots, l. \tag{22.5}$$

Notice that QP is a linearly constrained quadratic programming problem, for which we have seen several solution approaches. Moreover, notice that the optimality conditions of QP are given by (22.2) and (22.3), where v is the dual variable associated with the constraints in (22.5), which, in turn, represent first-order approximations of the original constraints.

The objective function in QP can be interpreted as being a second-order approximation of $f(x)$ enhanced with the term $(1/2)\sum_{i=1}^{l} v_i^k d^\top \nabla^2 h_i(x^k)d$ that captures constraint curvature information.

An alternative interpretation for the objective function of QP is to notice that it consists of the second order approximation of the Lagrangian function $L(x, v) = f(x) + \sum_{i=1}^{l} v_i h_i(x)$ at (x^k, v^k), which is given by

$$\begin{aligned} L(x, v) &\approx L(x^k, v^k) + \nabla_x L(x^k, v^k)^\top d + \frac{1}{2} d^\top \nabla^2 L(x^k, v^k)d \\ &= f(x_k) + {v^k}^\top h(x^k) + (\nabla f(x^k) + {v^k}^\top \nabla h(x^k))^\top d + \frac{1}{2} d^\top (\nabla^2 f(x^k) + \sum_{i=1}^{l} v_i^k \nabla^2 h_i(x^k))d \end{aligned}$$

To see this, notice that terms $f(x^k)$, ${v^k}^\top h(x^k)$ are constants and that $\nabla h(x^k)^\top (x - x^k) = 0$ (from (22.5), as $h(x^k) = 0$).

The general subproblem in the SQP method can be stated as

$$\begin{aligned} QP(x^k, u^k, v^k) : \text{min.} \quad & \nabla f(x^k)^\top d + \frac{1}{2} d^\top \nabla^2 L(x^k, u^k, v^k)d \\ \text{s.t.: } & g_i(x^k) + \nabla g_i(x^k)^\top d \leq 0, i = 1, \ldots, m \\ & h_i(x^k) + \nabla h_i(x^k)^\top d = 0, i = 1, \ldots, l, \end{aligned}$$

which includes inequality constraints $g_i(x) \leq 0$ for $i = 1, \ldots, m$ in a linearised from and their respective associated Lagrangian multipliers u_i, for $i = 1, \ldots, m$. This is possible since we are using an optimisation setting rather than a Newton system that only allows for equality constraints,

even though the latter can be obtained by simply introducing slack variables. Clearly, there are several option that could be considered to handle this quadratic problem, including employing a primal/dual interior point method.

A pseudocode for the standard SQP method is presented in Algorithm 23.

Algorithm 23 SQP method

1: **initialise.** $\epsilon > 0, x^0 \in S, u^0 \geq 0, v^0, k = 0$.
2: **while** $||d^k|| > \epsilon$ **do**
3: $\quad d^k = \arg\min QP(x^k, u^k, v^k)$
4: $\quad$ obtain u^{k+1}, v^{k+1} from $QP(x^k, u^k, v^k)$
5: $\quad x^{k+1} = x^k + d^k, k = k + 1$.
6: **end while**
7: **return** x^k.

Notice that in Line 4, dual variable values are retrieved from the constraints in $QP(x^k, u^k, v^k)$. therefore, $QP(x^k, u^k, v^k)$ needs to be solved by an algorithm that can return these dual variables, such as the (dual) simplex method.

Figure 22.2 illustrates the behaviour of the SQP method on the problem. Notice how the trajectory might eventually become infeasible due to the consideration of linear approximations of the nonlinear constraint.

$$\text{min. } \left\{2x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2 : x_1^2 - x_2 \leq 0, x_1 + 5x_2 \leq 5, x_1 \geq 0, x_2 \geq 0\right\}$$

One important feature for the SQP method is that it closely mimics the convergence properties of Newton's method and therefore, under appropriate conditions, superlinear convergence can be observed. Moreover, the BFGS method can be used to approximate $\nabla^2 L(x^k, v^k)$, which can turn the method dependent only on first order derivatives.

Notice that, because the constraints are considered implicitly in the subproblem $QP(x^k, u^k, v^k)$, one cannot devise a line search for the method, which in turn, being based on successive quadratic approximations, presents a risk for divergence.

The l_1-SQP is a modern variant of SQP that addresses divergence issues arising in the SQP method when considering solutions that are far away from the optimum, while presenting superior computational performance.

In essence, l_1-SQP relies on a similar principle than penalty methods, encoding penalisation for infeasibility in the objective function of the quadratic subproblem. In the context of SQP algorithms, these functions are called "merit" functions. This not only allow for considering line searches (since feasibility becomes encoded in the objective function with feasibility guaranteed at a minimum. cf. penalty methods) or, alternatively, relying on a trust region approach, ultimately preventing divergence issues.

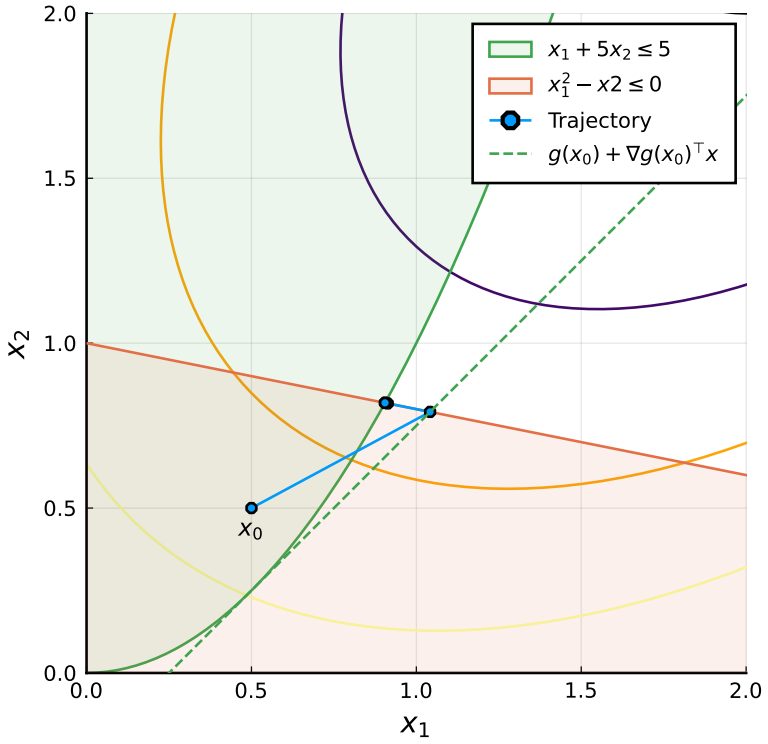


Figure 22.2: The SQP method converges in 6 iterations with $\epsilon = 10^{-6}$

Let us consider the trust-region based l_1-penalty QP subproblem, which can be formulated as

$$
\begin{aligned}
& l_1 - QP(x^k, v^k) : \\
& \quad \text{min.} \;\; \nabla f(x^k)^\top d + \frac{1}{2} d^\top \nabla^2 L(x^k, v^k) d \\
& \qquad\qquad + \mu \left[\sum_{i=1}^{m} [g_i(x^k) + \nabla g_i(x^k)^\top d]^+ + \sum_{i=1}^{l} |h_i(x^k) + \nabla h_i(x^k)^\top d| \right] \\
& \quad \text{s.t.:} \;\; - \Delta^k \leq d \leq \Delta^k,
\end{aligned}
$$

where μ is a penalty term, $[\,\cdot\,] = \max\{0, \cdot\}$, and Δ^k is a trust region term. This variant is of particular interest, because the subproblem $l_1 - QP(x^k, v^k)$ can be recast as a QP problem with linear constraints:

$$
\begin{aligned}
& l_1 - QP(x^k, v^k) : \\
& \quad \text{min.} \;\; \nabla f(x^k)^\top d + \frac{1}{2} d^\top \nabla^2 L(x^k, v^k) d + \mu \left[\sum_{i=1}^{m} y_i + \sum_{i=1}^{l} (z_i^+ - z_i^-) \right] \\
& \quad \text{s.t.:} \;\; - \Delta^k \leq d \leq \Delta^k \\
& \qquad\quad y_i \geq g_i(x^k) + \nabla g_i(x^k)^\top d, i = 1 \ldots, m \\
& \qquad\quad z_i^+ - z_i^- = h_i(x^k) + \nabla h_i(x^k)^\top d, i = 1, \ldots, l \\
& \qquad\quad y, z^+, z^- \geq 0.
\end{aligned}
$$

The subproblem $l_1 - QP(x^k, v^k)$ enjoys the same benefits the original form, meaning that it can be solved with efficient simplex-method based solvers.

The trust-region variant of l_1-SQP is globally convergent (does not diverge) and enjoys superlinear convergence rate, as the original SQP. The l_1-penalty term is what is often called a *merit function* in the literature. Alternatively, one can disregard the trust region and employ a line search (either exact or inexact) which would also enjoy globally convergent properties.

22.4 Generalised reduced gradient*

The generalised reduced gradient method resembles in many aspects the simplex method for linear optimisation problems. It derives from the Wolfe's reduced gradient. The term "reduced" refers to the consideration of a reduced variable space, formed by a subset of the decision variables.

22.4.1 Wolfe's reduced gradient

Let us consider the linearly constrained problem:

$$
\begin{aligned}
(P) \;\; : \;\; & \text{min.} \;\; f(x) \\
& \text{s.t.:} \;\; Ax = b \\
& \qquad\;\; Ax \geq 0,
\end{aligned}
$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, A is $m \times n$ and $b \in \mathbb{R}^m$.

To ease the illustration, we assume linear programming *nondegeneracy*, i.e., that any m columns of A are linearly independent and every extreme point of feasible region has at least m positive components and at most $n - m$ zero components.

Being so, let us define a partition of A as $A = (B, N)$, $x^\top = (x_B^\top, x_N^\top)$, where B is an invertible $m \times m$ matrix with $x_B > 0$ as a basic vector and $x_N \geq 0$ as a nonbasic vector. This implies that $\nabla f(x)^\top$ can also be partitioned as $\nabla f(x)^\top = (\nabla_B f(x)^\top, \nabla_N f(x)^\top)$.

In this context, for d to be an improving feasible direction, we must observe that

1. $\nabla f(x)^\top d < 0$
2. $Ad = 0$, with $d_j \geq 0$ if $x_j = 0$ to retain feasibility.

We will show how to obtain a direction d that satisfies conditions 1 and 2. For that, let $d^\top = (d_B^\top, d_N^\top)$. We have that $0 = Ad = Bd_B + Nd_n$ for any d_N, implying that $d_B = -B^{-1}Nd_N$. Moreover,

$$\begin{aligned} \nabla f(x)^\top d &= \nabla_B f(x)^\top d_B + \nabla_N f(x)^\top d_N \\ &= (\nabla_N f(x)^\top - \nabla_B f(x)^\top B^{-1}N)d_N \end{aligned} \tag{22.6}$$

The term $r_N^\top = (\nabla_N f(x)^\top - \nabla_B f(x)^\top B^{-1}N)$ is referred to as the reduced gradient as it express the gradient of the function in terms of the nonbasic directions only. Notice that the reduced gradient r holds a resemblance to the reduced costs from the simplex method. In fact

$$\begin{aligned} r^\top = (r_B^\top, r_N^\top) &= \nabla f(x) - \nabla_B f(x)^\top B^{-1}A \\ &= (\nabla_B f(x) - \nabla_B f(x)^\top B^{-1}B, \nabla_N f(x) - \nabla_B f(x)^\top B^{-1}N) \\ &= (0, \nabla_N f(x) - \nabla_B f(x)^\top B^{-1}N), \end{aligned}$$

and thus $\nabla f(x) = r^\top d$.

Therefore, d_N must be chosen such that $r_N^\top d_N < 0$ and $d_j \geq 0$ if $x_j = 0$. One way of achieving so is employing the classic *Wolfe's rule*, which states that

$$d_j = \begin{cases} -r_j, \text{ if } r_j \leq 0, \\ -x_j r_j, \text{ if } r_j > 0. \end{cases}$$

Notice that the rule is related with the direction of the optimisation. For $r_j \leq 0$, one wants to increase the value of x_j in that coordinate direction, making d_j non-negative. On the other hand, if the reduced gradient is positive ($r_j > 0$), one wants to reduce the value of x_j, unless it is already zero, a safeguard created by multiplying x_j in the definition of the direction d.

The following result guarantee the convergence of Wolfe's reduced gradient to a KKT point.

Theorem 22.2. *Let $\overline{x}$ be a feasible solution to P such that $\overline{x} = (\overline{x}_B^\top, \overline{x}_N^\top)$ and $x_B > 0$. Consider that A is decomposed accordingly into (B, N). Let $r^\top = \nabla f(\overline{x})^\top - \nabla_B f(\overline{x})^\top B^{-1}A$ and let d be formed using Wolfe's rule. Then*

1. *if $d \neq 0$, then d is an improving direction;*
2. *if $d = 0$, then $\overline{x}$ is a KKT point.*

Proof. d is a feasible direction by construction. Now, notice that

$$\begin{aligned}\nabla f(\overline{x})^\top d &= \nabla_B f(\overline{x})^\top d_B + \nabla_N f(\overline{x})^\top d_N \\ &= [\nabla_N f(\overline{x})^\top - \nabla_B f(\overline{x})^\top B^{-1} N] d_N = \sum_{j \notin I_B} r_j d_j\end{aligned}$$

where I_B is the index set of basic variables. By construction (using Wolfe's rule), either $d = 0$ or $\nabla f(\overline{x})^\top d < 0$, being thus an *improvement direction.*

$\overline{x}$ is a KKT point if and only if there exists $(u_B^\top, u_N^\top) \geq (0, 0)$ and v such that

$$(\nabla_B f(x)^\top, \nabla_N f(x)^\top) + v^\top (B, N) - (u_B^\top, u_N^\top) = (0, 0) \tag{22.7}$$

$$u_B^\top x_B = 0, u_N^\top x_N = 0. \tag{22.8}$$

Since $x_B > 0$ and $u_B \geq 0$, $u_B^\top x_B = 0$ if and only if $u_B = 0$. From top row in (22.7), $v^\top = -\nabla_B f(x) B^{-1}$. Substituting in the bottom row of (22.7), we obtain $u_N^\top = \nabla_N f(x)^\top - \nabla_B f(x)^\top B^{-1} N = r_N$.

Thus, the KKT conditions reduce to $r_N \geq 0$ and $r_N^\top x_N = 0$, only observed when $d = 0$ by definition. □

Algorithm 24 presents a pseudocode for the Wolfe's reduced gradient. A few implementation details stand out. First, notice that the basis is selected choosing the largest components in value, which differs from the simplex method by allowing for nonbasic variables to assume nonzero values. Moreover, notice that a line search is employed conditioned on bounds on the step size λ to guarantee that feasibility $x \geq 0$ is retained.

Algorithm 24 Wolfe's reduced gradient method

initialise. $\epsilon > 0, x^0$ with $Ax^k = b, k = 0$, columns a_j, $j = 1, \ldots, n$ of A
while $||d^k|| > \epsilon$ **do**
  I^k = index set for m largest components of x^k
  Let $A = (B, N)$, where $B = \left\{a_j : j \in I^k\right\}$, and $N = \left\{a_j : j \notin I^k\right\}$
  $r^{k\top} = \nabla f(x^k)^\top - \nabla_B f(x^k)^\top B^{-1} A$
  $d_j^k = \begin{cases} -r_j^k, \text{ if } j \notin I^k \text{ and } r_j \leq 0; \\ -r_j x_j, \text{ if } j \notin I^k and r_j > 0. \end{cases}$
  $d_B = -B^{-1} N d_N$
  if $d = 0$ **then**
    return x^k
  end if
  $\overline{\lambda} = \begin{cases} +\infty, \text{ if } d^k \geq 0; \\ \min\left\{x_j^k / d_j^k : d_j^k < 0\right\}, \text{ if } d^k < 0. \end{cases}$
  $\lambda^k = \arg\min\left\{f(x^k + \lambda d^k) : 0 \leq \lambda \leq \overline{\lambda}\right\}$.
  $x^{k+1} = x^k + \lambda^k d^k; k = k + 1$.
end while
return x^k.

22.4.2 Generalised reduced gradient method

The *generalised reduced gradient* extends Wolfe's method by:

1. Nonlinear constraints are considered via first-order approximation at x^k

$$h(x^k) + \nabla h(x^k)^\top (x - x^k) = 0 \Rightarrow h(x^k)^\top x = h(x^k)^\top x^k.$$

   with an additional *restoration phase* that has the purpose of recovering feasibility via projection or using Newton's method.

2. Consideration of *superbasic variables*. In that, x_N is further partitioned into $x_N^\top = (x_S^\top, x_{N'}^\top)$.

The superbasic variables x_S (with index set J_S, $0 \leq |J_S| \leq n-m$), are allowed change value, while $x_{N'}$ are kept at their current value. Hence, $d^\top = (d_B^\top, d_S^\top, d_{N'}^\top)$, with $d_{N'} = 0$. From $Ad = 0$, we obtain $d_B = -B^{-1}Sd_S$. Thus d becomes

$$d = \begin{bmatrix} d_B \\ d_S \\ d_{N'} \end{bmatrix} = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} d_S.$$

Bibliography

[1] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[2] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[3] Changhyun Kwon. *Julia Programming for Operations Research*. Changhyun Kwon, 2019.

[4] H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.