Optimization & Learning Lecture Notes

Antonio Frangioni

Version 0.0.1
September 24, 2024

Department of Computer Science
University of Pisa
https://www.di.unipi.it/~frangio
mailto:frangio@di.unipi.it

Preface

These lecture notes are intended as a support for some courses at the University of Pisa dealing with the intersection between Data Science / Artificial Intelligence / Machine Learning and Optimization, such as (the optimization part of) Computational Mathematics for Learning and Data Analysis of the Master Program in Computar Science, Optimization for Data Science in the Master Program in Data Science and Business Informatics, and Optimization Methods and Game Theory in the Master Program in Artificial Intelligence and Data Engineering.

The aim of these courses is to provide mathematical optimization concepts that are useful in the design and analysis of methods for learning out of (large sets of) data. It has to be intended that the fundamental DS/AI/ML concepts are *not* covered in these lectures; some of the main models (regression, Neural Networks, Support Vector Machines, Clustering) are described, but only as relevant examples of applications of the optimization techniques. Issues like the learning capabilities of these models and how to improve them are completely outside of the focus of these lectures, since they are dealt with in specific courses. What is covered here is purely some of the mathematical optimization methods that are relevant among several others, for DS/AI/ML applications.

Several students of the courses have contributed to early versions of these lectures, among which Gemma Martini, Ivan Grujic, Federica Di Pasquale, and Erica Neri. The responsibility of any error, inconsistency or incompleteness in the lecture notes fully remains with the extensors.

Contents

Part I

A Gentle Introduction

Chapter 1

Simple Optimization Problems

1.1 Optimization Problems

1.1.1 First definitions

An *optimization problem* corresponds, roughly speaking, to the following question: "among all my choices, which one is best?" In order to poise the question in mathematical terms, there are two things that need be defined mathematically: what are the choices, and how to define "best".

For the first part we start assuming that we are given a *set* X that contains all the choices. We don't formally discuss the intuitive notion of set [11, §1.3], and we will postpone to much later the discussion of how the set is actually represented/implemented. Anyway, most sets considered in these lecture notes are "easy". The set of choices is called *feasible region*, and any element $x \in X$ is called *feasible solution* (the adjective "feasible" would seem to be redundant, but we'll see later on that the concept of unfeasible solution also make sense in some contexts). The set may be finite (but, often, "very large") or infinite, provided that it can be appropriately handled algorithmically.

Choosing among the elements in X requires a way to define "best"; this is done via a *function*, i.e., a mapping $f : X \to Y$ between the x *input space* X and the *output space* Y. Again we don't insist on mathematical details of the concept (cf. e.g. [10, §2.2.1]); anyway all our functions will be (relatively) "easy", i.e., either algebraic or with a "reasonably cheap" algorithm available to evaluate them pointwise (that is, produce $y = f(\,x\,)$ given x). While we will be quite free (within limits) with our input space X, using some f as the *objective* (function) in an optimization problem severely limits the choices for Y. In fact, the objective is there to answer the following question: "given two different feasible solutions $x' \in X$ and $x'' \in X$, which one is best?" The idea is that f maps this concept into

$$x' \text{ is better than } x'' \iff f(\,x'\,) = y' < y'' = f(\,x''\,)\;.$$

Intuitively, $y = f(x)$ is the "cost" of choice x, and we want to select the least costly choice. However, for this to make sense the notion "$y' < y''$" must be well defined; mathematically, Y must be a (total) *order* [10, Def. 2.2.8], [11, §1.1]. For our purpose this will mean that $Y = \mathbb{R}$, i.e., our objective is a *real* function (produces real numbers). This is, in some sense, a strong simplifying assumption: there are many situations in life where decisions need to taken considering multiple aspects of the choices. Indeed, *multi-objective optimization* exists; yet, many concepts then become much less convenient. Furthermore, as briefly discussed in §1.1.7, there are a few "dirty tricks" that are regularly used to turn a multi-objective optimization problem into an ordinary (single-objective) one. These are more than enough (and, in fact, frequently used) in the applications of interest here, and therefore in these notes we will only work with real functions as objectives.

This gives us the basic form of *optimization* (minimization) *problem*

$$(P) \quad f_* = \min\{\, f(\,x\,) \,:\, x \in X \,\}\;. \tag{1.1}$$

Problems are often given "names", in this case "P", to be able to speak about them (especially if one is speaking about multiple ones at the same time, as it often happens). A distinction need be made between "optimization problem" and "instance of an optimization problem", though. The latter is one very specific question (= instance) regarding a completely defined f and X, and therefore (possibly, as discussed below) has a very specific answer. The former is rather a class, or (often, infinite) set, of questions, each one characterised by the values of some *parameters* that can be set in (typically, infinitely) different ways. This is relevant because *algorithms are devised for optimization problems*: one does not want to write an algorithm just to solve one specific questions, but (infinitely) many of them. The following example illustrates the concept.

Example 1.1. The equipartition problem
The equipartition problem corresponds to the following question: given a set of n natural numbers, $N = \{ a_1 , a_2 , \dots , a_n \}$, what is the subset S of N such that the difference in magnitude between the sum of the numbers in S and that of the numbers in $N \setminus S$ is the smallest possible ? A mathematical formulation of the problem is

$$(EQ) \quad \min \left\{ f(x) = \left| \sum_{a \in x} a - \sum_{a \in N \setminus x} a \right| \, : \, S \subseteq N \right\} \ . \tag{1.2}$$

In this case, X is the set of all subsets of N; in fact, the only condition that an answer must satisfy is to be a subset of the given n numbers. For this problem, the *parameters* are the number "n" and the n numbers "a_1, a_2, ..., a_n"; choosing for example $n = 4$ and $N = \{ 7 , 3 , 4 , 6 \}$ we obtain a particular instance of the problem, in which all the parameters are specified. The specific instance can be given a definite answer (in this case it is easy to see that $x = \{ 3 , 7 \}$ with $f(x) = 0$ is one); yet, when designing algorithms one would want them to work for any possible instance of the problem, i.e., irrespectively on how the parameters are fixed.

Because algorithms generally work on sets of problems, a fundamental issue that immediately rises is that of *efficiency*, i.e., how fast the solution time grows as the size of the problem does; this will be a central concern in these lecture notes. The problem in Example 1.1 is *combinatorial*: the number of possible different decisions to the problem, i.e., the size of X, grows as 2^n. In fact, despite its apparent simplicity the problem is "difficult" ($\mathcal{NP}$-hard): there are no known algorithms capable of solving it in general in polynomial time in n, which can be thought at first to be due to the fact that "there are too many possible decisions to look at". Most problems considered in these lectures are "even worse", in that they are *continuous*: X is composed of (uncountably) *infinitely many* solutions. Finding the optimal choice in such a set may well be impossible, and relaxed notions of "solving the problem" will have to be devised, as we shall see. With appropriate assumptions there are, of course, continuous problems that admit "efficient" solution algorithms; in fact. these will be the main focus of these lecture notes.

The value f_* as defined in (1.1) is called the *optimal value of problem* P, and it is denoted as $\nu(P)$ when it is useful to make it explicit its dependency on P. Since the minimum of a set of real numbers is unique (when it is well defined, as discussed below) one can then easily use it to define the *optimal solutions* of P, i.e., any $x^* \in X$ such that $f(x^*) = f_*$; assuming that any exists. In fact, what one is primarily interested is x^* rather than f_*, hence the problem should actually be regarded as

$$(P) \quad x^* \in \operatorname{argmin} \{ f(x) \, : \, x \in X \} \tag{1.3}$$

with "argmin" denoting the set of $x \in X$ that attain the minimal value of f; in other words, the set X^* of all optimal solutions to P. While f_* is unique (when it is well-defined), x^* is not, i.e., it often happens that X^* is not a singleton. Indeed, X^* may easily contain (uncountably) infinitely many solutions, which might make the task impossible if one were required to produce them *all*. However, as (1.3) underlines, the idea is that one is tasked with finding *any one* of the optimal solutions, regardless to which one. This makes sense, in that "in the eyes of the objective all optimal solutions are equal", and "there are no other eyes except the objective's": there is no way to distinguish two different solutions $x' \in X^*$ and $x'' \in X^*$, and therefore to choose among them. If, from the viewpoint of the practical application driving the definition of (P), there were any reason to actually distinguish x' from x'', and favour one over the other, then f would not be a "complete" objective function, in that it would not properly represent all the preferences of the decision-maker on behalf of whom the problem is being solved. Although devising a "complete" objective may entail difficulties (cf. again §1.1.7), in these lecture notes we will always assume that this has in fact been done: as a consequence, solving an optimization problem requires finding just any one of its optimal solutions.

As it often happens, these intuitive notions require specification and clarification once they need to be expressed into exact mathematical notations. Indeed, when P is defined there are no less than *four* different situations that may arise:

1. The "normal" one in which $f_* \in \mathbb{R}$ is well-defined and at least one optimal solution x^* exists: this is what will happen in almost all the cases of interest in these lecture notes. As we will see, this by no means implies that finding f_* / x^* is efficiently doable, or in fact at all possible. One specific issue (in some sense, the crucial one) is whether or not, given some x, it is possible to efficiently detect whether or not $x \in X^*$. This might seem simple in the sense that computing $y = f(x)$ is efficiently doable (by our basic assumption on f), so it would seem that one is only required to check that $y = f_*$: this is true, except that f_* *is unknown* as well. Indeed, in some sense *finding* f_* is "the really difficult part" of solving (P), at least in a number of cases. We will return to this concept later on.

2. The case in which f_* is not a real value: the obvious example is the optimization problem
$$\min\{ x \, : \, x \in \mathbb{R} \} \ .$$
There clearly is no optimal solution: however fixed $M \in \mathbb{R}$ there exist some $x_M \in X$ (in our case, just $x = M$) such that $f(x_M) \leq M$. Formally speaking, the *image of X through f*

$$\mathrm{im}_X(\,f\,) = \{\, y \,:\, \exists\, x \in X \text{ s.t. } y = f(\,x\,) \,\} \subset \mathbb{R}$$

is *unbounded below*: there is no finite value $\underline{f} \in \mathbb{R}$ such that $y \geq \underline{f}$ for all $y \in \mathrm{im}_X(\,f\,)$. In plain words, one can obtain solutions that as "as good as one wants them". In this case one customarily says that $f_* = -\infty$. That is, f_* is still well-defined provided one is happy to let the value live in the set of *extended reals* $\bar{\mathbb{R}} = \mathbb{R} \cup \{\, -\infty\,,\, +\infty \,\}$ [11, p. 7]. The two special symbols "$-\infty$" and "$+\infty$" have their intuitive meaning and can, most of the times, be treated like ordinary real numbers with a bit of tweaking of the standard algebraic rules [11, p. 7]. In fact it will be sometimes be expedient to allow f to be an *extended real* function, i.e., $f(\,x\,) = \pm\infty$ to happen for some x. Even if this is not the case, however, $f_* = -\infty$ (or $+\infty$ for maximization problems, as we will see) still "naturally happens". This is not, in principle, an issue: in fact, in practice one would likely be very happy (were it not that such an occurrence almost invariably means that the model that one is solving does not correctly represent the reality). Yet, formally speaking this means that there cannot be any optimal solution: $X^* = \emptyset$. Also, *proving* that $f_* = -\infty$ is possibly even trickier than proving it has a finite value. Indeed, one should basically provide a(n easy to compute) function that, given any $M \in \mathbb{R}$, provides the $x_M \in X$ such that $f(\,x_M\,) \leq M$. This is indeed possible in a few cases, but not in many others. Fortunately, in the applications of interest here f is usually bounded below (on X) and the issue is hardly ever significant. Yet, in general this is a case that optimization algorithms have to properly take into account.

3. The case in which there cannot be an optimal solution because there is no solution at all: $X = \emptyset$. This may seem weird on a first look, but deciding about the emptiness of a set may not be trivial (in fact, it may be extremely difficult) depending on how the set is specified. This will not be an issue for most of the problems and algorithms studied in these notes because X will be "very simple" and often clearly nonempty; however, in Chapter **??** some algorithms will be discussed where emptiness of the feasible set is a possibility that have to be dealt with. Proving that a set is empty may be nontrivial, but once it is achieved it is a valid and complete answer to (P): there are no solutions to choose from, and therefore no optimal solution. For good mathematical reasons, when $X = \emptyset$ is it customary to set $f_* = +\infty$ ($-\infty$ for a maximization problem), which confirms that f_* "naturally lives in $\bar{\mathbb{R}}$" even if f is real-valued.

The three cases above are succinctly described by the fact that $f_* \in \mathbb{R}$, $f_* = -\infty$ or $f_* = +\infty$. However, a fourth case exists (in theory) where f_* is ill-defined. This is simply the case when $\mathrm{im}_X(\,f\,)$ is bounded below (therefore, f_* is not $-\infty$) but it does not have a minimum: the obvious example is the optimization problem

$$\min\{\, 1\,/\,x \,:\, x \in \mathbb{R}_+ \,\} \tag{1.4}$$

(with $\mathbb{R}_+ = \{\, x \in \mathbb{R} \,:\, x \geq 0 \,\}$, and $\mathbb{R}_-$ defined accordingly). Clearly, $\mathrm{im}_X(\,f\,)$ contains all strictly positive numbers, but not 0: that is, any value $v > 0$ cannot be f_*, but on the other hand there is no $x \in X$ such that $f(\,x\,) = 0$. This is because any bounded set of real numbers have an *infimum* (cf. §A.1, [11, Theorem 1.1.8]), but not necessarily a *minimum*: thus, f_* would be well-defined, and $= 0$ in the example, if one would write "inf" instead of "min" in the definition. Yet, the problem would not have any optimal solution.

There are reasonably simple (even if not always practical) conditions under which the fourth case can be shown not to happen (e.g., [10, Theorem 16.2.2]), but this is not really an issue in practice. The point is that finding the *exact* solution of an optimization problem is, in general, impossible. This will be argued in general terms in Chapter **??**, but a case will be done immediately on the grounds that, in practice, writing "$x \in \mathbb{R}$" is an abstraction that does not really represent what happens in actual solution methods on actual computers. There, "real numbers" is actually "fixed-size floating-point numbers", typically 64-bits `double`. Although these numbers have a quite good accuracy, corresponding to about 16 digital digits, they do have a finite one: in other words, two "close enough" real numbers are indistinguishable once represented as `double`. In our example, anything that is closer to 0 than the machine precision (roughly 10^{-16}, or `1e-16` in the computer notation that we will favour) is basically indistinguishable from 0, and this even before considering the inevitable numerical errors that have typically been incurred while computing it and that further decrease the accuracy. While there are even more accurate floating-point numbers (e.g., 128-bit quadruple-precision floating-point), and even algebraic systems capable of doing computations in theoretically infinite precision, these are usually way too slow for the kind of applications of interest here. In fact, floating-point numbers with even *less* accuracy are common, such as 32-bit single-precision (`float`), 16-bit half-precision (`_Float16`). Recently, the trend in Machine Learning applications has been towards *even less* accuracy, such as 8-bit or even (almost incredibly) 4-bit mini-floats [65], which have been found useful to reduce the tremendous memory requirements of very large models. Without going in any detail about the issue of the errors that this entails and their impact on the algorithms, we use this discussion just to motivate the fact that "solutions with an objective value close enough to f_* are good enough answers".

1.1.2 Approximate optimality

The previous discussion leads to the following definition: the (absolute) *gap* of some $x \in X$, intended as a solution to (P), is

$$A(x) = f(x) - f_* \;(\geq 0) \; . \tag{1.5}$$

Any solution $x \in X$ with $A(x) \leq \varepsilon$ is an *ε-optimal solution*, and "solving (P)" will in general always be intended as: "*given* ε, find a ε-optimal solution." Of course ε should not be chosen too small, i.e., smaller than the machine precision, for the task to be at all possible. Even more importantly, *most often the solution cost of the algorithms will depend on* ε, and in particular grow (possibly rather fast) as ε is reduced. This is very intuitive: more accurate solutions are more costly to obtain than less accurate ones. In fact, in many cases choosing ε even remotely close to the machine precision will not be feasible because of the very high cost that this would have. This justifies the fact that we will not be too concerned about the impact of the accuracy of the floating-point numbers used to "approximate" $\mathbb{R}$: the accuracy attainable in practice by an optimization process is usually way lower (the error way larger) than the inherent accuracy of the floating point numbers because of the computational cost.

To make this discussion more practical we introduce a slightly different concept: the *relative gap*

$$R(x) = (f(x) - f_*) / |f_*| = A(x) / |f_*| \;(\geq 0) \tag{1.6}$$

of some solution $x \in X$. There are good reasons for wanting to define *ε-optimal solution in a relative sense* any $x \in X$ with $R(x) \leq \varepsilon$. Some will be discussed shortly, but one is already implicit in the working of floating-point numbers. As the name implies, the accuracy of these numbers is *relative to their size.* That is, saying that `double` have (roughly) `1e-16` accuracy means that they have (about) 16 significant digits in their decimal scientific representation; any digit (way) beyond the 16th is not really conveying useful information. This is illustrated by the following snippet of `Matlab` code:

```
>> fprintf( '%1.16e\n' , pi )
3.1415926535897931e+00
>> pi3 = pi^(pi^pi);
>> fprintf( '%1.16e\n' , pi3 )
1.3401641830063398e+18
>> fprintf( '%1.16e\n' , pi3 - ( pi3 + pi ) )
0.0000000000000000e+00
```

That is, the number π, which is order of `1e+0`, is represented with an error that is $\approx$ `1e-16`: all the figures that are printed are correct. Its thrice exponentiation is of the order of `1e+18`: it means that its digit corresponding to `1e+0` is not significant. In fact, adding π to it gives a number that is identical to the original one. That is, the inherent error in a floating-point number is related to its magnitude.

This implies that knowing that, say, $A(x) \approx$ 1e-3 gives no meaningful information unless one also knows f_*: for $f_* \approx$ `1e+10` this would mean that x is a very accurate solution, with its function value $f(x)$ agreeing with f_* at least on the first 13 decimal digits, while for $f_* \approx$ `1e+0` it would rather be a rather poor one (only the first 3 decimal digits right). Relative errors immediately solve the issue: $R(x) \approx$ `1e-D` means that $f(x)$ agrees with f_* at least on the first `D` decimal digits. Hence, $R(x) \approx$ `1e-16` is arguably the best possible result while using `double`, and often too much to ask: $R(x) \approx$ `1e-12` is generally considered a very good accuracy already. Any $R(x)$ between `1e-8` and `1e-6` is usually regarded as "decent but not exceptional", `1e-3` to `1e-4` is "low accuracy but adequate in some cases", and anything larger than that is usually regarded as "poor". There can be exceptions, in that for very difficult problems even accuracies of the order of `1e-2` (a few percentage points) can be acceptable. However, these lecture notes are about "easy" problems (possibly "cheating about the difficulty", as we shall see), and hence relative accuracies at least of the order of `1e-4`, or better, are usually expected.

In practice, the relative error is considered a much better measure of the accuracy of a solution. Besides for the reason previously outlined, this is useful to make the condition *scale invariant*, as discussed in §1.1.3. Yet, when analysing the behaviour of algorithms in theory, using the relative gap may complicate the arguments; among other things, for the reason outlined in Exercise 1.1 below. Hence, it is more common to run the analysis for the absolute error, knowing that in principle the two are only different by fixed factor $1 / |f_*|$. In these notes "gap" and "ε-optimal solution" will therefore have to be intended, unless otherwise specified, as referring to the absolute notion. However, in practical implementations the relative versions should be used.

Exercise 1.1. $R(x)$ ill-defined if $f_* = 0$: propose solutions and justify why your proposal makes sense. **[solution]**

A final important remark is the following: while the (absolute or relative) gap is a very natural measure of (sub)optimality, *actually computing* (1.5) or (1.6) requires *knowing* f_*, which is typically *unkown*. In fact, one could argue that computing (an estimate of) f_* "is the issue" in optimization: in a number of cases, any efficient approach for doing this would translate in an efficient approach for solving the problem (or at least make existing approaches much more efficient). This is in fact more relevant for other classes of problems and algorithms than the ones that these lecture notes are concerned with, hence we will not focus too much on this particular aspect. It is however useful to point out immediately that knowledge of f_*, and therefore the ability to gauge how much any given x is a "good" solution, is absolutely *not* given. This is relevant in the current discussion because, while it is true that $A(x)$ and $R(x)$ only differ by the fixed factor $1 / | f_* |$, it is also true that this factor is unknown during the actual execution of the algorithms, which is one of the reasons why the analysis of $A(x)$ is usually easier.

1.1.3 (Simple) reformulations

Exercise 1.1 has given a first hint of an important concept in optimization, that of *reformulation*. The idea is that of rewriting a given optimization problem in a different but equivalent form that is more convenient for some reason. This is a deep concept, and very sophisticated (and intricate) reformulation schemes have been devised that can have a substantial impact on the ability to efficiently solve some optimization problem. A much less complex, but still useful, use of the concept is that of establishing *normal forms* of optimization problems, so that one can always assume (P) to have a specific form while in fact allowing flexibility in actual implementations.

The first obvious example is that of *maximization* problems

$$(P_{\max}) \quad f_* = \max \, / \, x^* \in \operatorname{argmax} \{ \, f(x) \, : \, x \in X \, \}$$

where one is seeking for the choices with larger profit rather than the one with smaller cost. This clearly is not significantly different from a minimization problem, in the sense that

$$(P^-_{\min}) \quad \min\{ \, -f(x) \, : \, x \in X \, \}$$

has the property that $\nu(P_{\max}) = -\nu(P^-_{\min})$: in plain words, maximizing f is equivalent to minimizing $-f$, and then changing the sign to the optimal value. It is immediate to see that every optimal solution to $(P_{\max})$ (if any) is an optimal solution to $(P^-_{\min})$ and vice-versa, which is a very reasonable definition (although not the most general one) of "$(P_{\max})$ and $(P^-_{\min})$ are equivalent problems"; in fact, the equivalence immediately extends to the unbounded and unfeasible cases $(-(-\infty) = +\infty)$. It is important to remark that maximizing and minimizing the *same* function f (without changing its sign) is instead *not* at all equivalent; in fact they can be "very different problems", as we shall see. However, by just changing a sign of the computed value $y = f(x)$ (an operation that can only be trivial), one can assume without loss of generality that any optimization problem is minimization one.

A similar property holds between (P) and

$$\begin{aligned} (P_c) &\quad f_* = \max \, / \, x^* \in \operatorname{argmax} \{ \, f(x) + c \, : \, x \in X \, \} \\ (P_\alpha) &\quad f_* = \max \, / \, x^* \in \operatorname{argmax} \{ \, \alpha f(x) \, : \, x \in X \, \} \end{aligned}$$

provided that $\alpha > 0$, i.e., optimal solutions are conserved while the optimal value is modified in the obvious way (namely, $\nu(P_c) = \nu(P) + c$ and $\nu(P_\alpha) = \alpha\nu(P)$). In plain words, optimization is not affected by any *constant* or *scale factor* applied to the objective. This will, for instance, allow to disregard constant terms in the functions that are analysed in §1.2 and §1.3. Note that this provides another rationale for the use of the relative gap, in that any solution x has the same value of $R(x)$ in (P) and in (P_α), irrespectively to any scale factor α applied to the objective that would, instead, impact on $A(x)$.

1.1.4 Representing the feasible set

So far, membership in X has been implicitly considered "easy", in the sense that checking whether "$x \in X$" is true or not has never been discussed. In reality checking membership in a set can be hard and may have some of the same issues as checking optimality, starting with the accuracy ones. It all, of course, hinges on how the set X is specified. In our setting, this will always be done by specifying an "easy" *ground set* $F \supseteq X$, and then (possibly) some *constraints* that impose the conditions that the elements $x \in X$ must satisfy. This means that elements $x \in X$ (if any) are designated *feasible solutions* to (P), while those $x \in F \setminus X$ (if any) are designated *unfeasible solutions*. In these lecture notes, the choice of the ground set will be fixed: F will always be the *finite-dimensional vector space* $\mathbb{R}^n$, typically for any value of n. That is, (feasible or unfeasible) solutions will be n-vectors of reals $x \in \mathbb{R}^n$, i.e., $x = [\, x_1 \, , \, x_2 \, , \, \ldots \, , \, x_n \,] = [\, x_i \,]_{i=1}^n$; hence, the objective will be a function $f : \mathbb{R}^n \to \mathbb{R}$ of n real (in practice, `double` or suchlike) variables

$$f(x) = f(x_1, x_2, \ldots, x_n) .$$

We refer to §A.2 for the reminders about finite-dimensional vector spaces and their fundamental operations.

One standard way to represent a subset $X \subseteq \mathbb{R}^n$ is via (more than) one function. In particular, some *constraint function* $g : \mathbb{R}^n \to \mathbb{R}$ and some *right-hand side* $\delta \in \mathbb{R}$ can be used to define

- an *equality constraint* $g(x) = \delta$;
- an *inequality constraint* $g(x) \leq \delta$.

Note that this correspond to the definition of *level set* of the function g at value δ

$$L(f, v) = \{ x \in \mathbb{R}^n : f(x) = v \} \subset \mathbb{R}^n$$

and *sublevel set* of the function g at value δ

$$S(f, v) = \{ x \in \mathbb{R}^n : f(x) \leq v \} \subset \mathbb{R}^n$$

In these lecture notes, the feasible region X will always be specified by a *finite number* of equality and inequality constraints (semi-infinite optimization where the constraints are infinitely many is also possible under some conditions, but not a explored here). Note that the assumption is always that constraints operate in logical conjunction: "$g_1(x) \leq 0$, $g_2(x) \leq 0$" must always be taken as "the set of x that satisfy *both* the first condition *and* the second condition". In other words, one is always looking at the *intersection* of (sub)level sets of a finite number of functions.

This potentially paves the way for many different forms of problems, but, taking a leaf from §1.1.3, we can consider simple *reformulation rules* that allow to define standard forms for optimization problems:

1. The right-hand side δ of a constraint can be "hidden in the constraint function g"; that is, one can define $\bar{g}(x) = g(x) - \delta$, and the constraint as $\bar{g}(x) = 0$ (or ≤ 0). It is often expedient to do that, thus unless otherwise specified the right-hand side of the constraint(s) is always assumed to be 0.
2. It may be natural to rather define constraints via a $\geq$ relation. For instance, in (1.4) $X = \mathbb{R}_+$ is naturally defined as $X = \{ x \in \mathbb{R} : x \geq 0 \}$. Yet, it is always possible to assume all inequality constraints to have the $\leq$ form since $g(x) \geq 0 \implies -g(x) \leq 0$, i.e., again, by re-defining the constraint function $\bar{g}(x) = -g(x)$.
3. It would be possible to assume that there are *no equality constraint*, since $g(x) = 0$ is equivalent to
$$g(x) \leq 0 \quad , \quad g(x) \geq 0 \;\; [\equiv -g(x) \leq 0] ;$$
in plain words, one equality constraint is equivalent to two inequality ones. This will sometimes be expedient, but in some sense (as we shall see) "equality constraints work very differently from inequality ones" and in a number of cases it will be appropriate to keep the distinction.
4. Somehow in the opposite direction, it would be possible to assume that *all constraints are equality ones save for very special inequalities, i.e., sign constraints.* This is due to the following equivalence
$$g(x) \leq 0 \quad \equiv \quad g(x) - s = 0 \;\; , \;\; s \geq 0$$
where s is a "new variable" (called *slack variable*) that "appears nowhere else in the optimization problem save in these two constraints"; in other words, none of the constraint functions (nor the objective) save these two must depend on s. This of course comes at the cost of introducing new variables (as the previous trick came at the cost of introducing new constraints), which may impact the efficiency of the solution algorithms; but this will be discussed in due time. On the other hand, sign constraints are "very easy" inequality constraints, that in a number of cases can be treated in special ways, making the solution approaches (and/or their theoretical analysis) simpler and/or more efficient.
5. Finally, in a problem with only inequality constraints (which can be assumed w.l.o.g., cf. 3.) one could as well assume that *there is only one inequality constraint*, owing to the fact that
$$g_i(x) \leq 0 \;\; \forall i = 1, \ldots, m \quad \equiv \quad \bar{g}(x) = \max\{ g_i(x) : i = 1, \ldots, m \} \leq 0 .$$
While having only one constraint (as opposed to, say, one million) may appear convenient, in practice this last trick is seldom used because the max operator it involves is not "nice" (as we shall see, $\bar{g}$ is not differentiable even if all the g_i are, which is a significant downside). However, it is worth reporting it since it underlines the power of reformulation techniques to rewrite optimization problems in different forms that can be more or less well-suited for specific algorithmic approaches.

These lecture notes are mainly concerned with optimization problems where X is "simple". Indeed, most of the work is on problems where $X = \mathbb{R}^n$ outright, i.e., there are no constraints at all: not surprisingly these are called *unconstrained optimization problems*. In a number of other cases constraints will be present, but they will always be "easy", more or less in the same vein as the sign constraints. In particular, common and useful classes of (inequality) constraints are *upper / lower bound constraints*, i.e., $x \geq \underline{x}$ / $x \leq \overline{x}$, and their intersection $\underline{x} \leq x \leq \overline{x}$ known as *box constraints*. In the above formulae, $\underline{x}$ and $\overline{x}$ have to be intended as constant vectors. However, it is customary to allow that

$$\underline{x} \in [\, \mathbb{R} \cup \{ -\infty \} \,]^n \quad \text{and} \quad \overline{x} \in [\, \mathbb{R} \cup \{ +\infty \} \,]^n \ .$$

That is, each variable has a lower and/or upper bound, $x_i \geq \underline{x}_i$ and/or $x_i \leq \overline{x}_i$; however, it is allowed that $\underline{x}_i = -\infty$, i.e., that there actually is no lower bound, and/or $\overline{x}_i = +\infty$, i.e., that there actually is no upper bound. Note that obviously it has to be $\underline{x}_i < \overline{x}_i$ (which is why $\underline{x}_i = +\infty$ or $\overline{x}_i = -\infty$ cannot be allowed): in fact, $\underline{x}_i > \overline{x}_i$ immediately makes X empty (and it is very easy to detect), while $\underline{x}_i = \overline{x}_i$ means that the variable x_i is fixed to their common value and is basically eliminated from the problem (fixing a variable almost invariably makes the problem easier). Despite their simplicity box constraints somehow "capture a good part of the complexity of (inequality) constraints" (cf. point 4. above) and therefore are a good compromise between generality and simplicity, which will be repeatedly exploited.

1.1.5 Approximate feasibility

When X is specified by a finite number of constraints, checking membership becomes "easy". This is of course true if pointwise evaluation of all constraint functions g_i is efficient and the number of constraints is "not too large". Both conditions are of course satisfied in the case of box constraints, that are at most twice the number of variables (which can be "many", but this means that the problem is inherently large, and not so because of the presence of constraints). However, this is still subject to the same accuracy issues as the computation of the objective (cf. §1.1.2). That is, once the constraint function $y = g(\, x \,)$ has been computed, one is still left with the issue of deciding whether or not $y = 0$, $y > 0$, or $y < 0$ (as the case may be). Numerically speaking this still requires finding, say, an appropriate threshold $\varepsilon > 0$ and implementing the checks as $|\, y \,| \leq \varepsilon$, $y \geq \varepsilon$, and $y \leq -\varepsilon$, respectively. ε should clearly not be smaller than the precision of the floating numbers, ideally considering the magnitude of all intermediate results that concurred in the computation of y. This underlies the fact that even membership in a "simple" X characterised by a "small" number of "easy" functions is in principle not necessarily nontrivial.

On the other hand, it also reveals that some situations that would be issues in a mathematical sense are not such in practice. This is related to the fact that (inequality) constraints have been defined by means of the $\leq$ relationship rather than the $<$ one. Mathematically speaking this is necessary, in that an optimization problem with strict inequality constraints may have analogous undesirable behaviour as (1.4), as illustrated, e.g., by

$$\min\{\, x \,:\, x > 0 \} \ . \tag{1.7}$$

Such a problem has ill-defined optimal value (save accepting the inf version) and no optimal solution. However, it does have ε-optimal solutions for any $\varepsilon > 0$. Numerically speaking, there is something like the smaller ε that is distinguishable from 0 (considering the magnitude of numbers . . .), and "$x > 0$" means "$x \geq \varepsilon$".

Things are of course potentially easier for "simple" forms of the constraints like the box ones. In principle, membership there could just be checked by the natural form $\underline{x} \leq x \leq \overline{x}$, assuming that any numerical error made in translating the "true" value of $\underline{x}$ and $\overline{x}$ into their floating-point form is acceptable (and it's better be, since it's unavoidable). However, usually the preferred implementation is rather akin to one among

$$\begin{array}{c} \underline{x}(1 - sign(\, \underline{x} \,)\varepsilon) \leq x \leq \overline{x}(1 + sign(\, \overline{x} \,)\varepsilon) \\ \underline{x}(1 + sign(\, \underline{x} \,)\varepsilon) \leq x \leq \overline{x}(1 - sign(\, \overline{x} \,)\varepsilon) \end{array} \ ,$$

with $sign(\, x \,) = -1$ if $x < 0$, $sign(\, 0 \,) = 0$, $sign(\, x \,) = 1$ if $x > 0$, and ε "close to machine precision but not too much", say `1e-12` for `double`. The first form slightly enlarges the interval ensuring that x can get the value of the extremes, the second form slightly restrict it ensuring that x is not beyond than the value of the extremes. This does not really work in case of $\underline{x}$ or $\overline{x}$ being 0 (which is not uncommon, these being then sign constraints), but this can be managed the same way as in Exercise 1.1. This becomes more important for general (possibly, two-sided) constraints

$$\underline{\delta} \leq g(\, x \,) \leq \overline{\delta}$$

where the error in computing the constraint function may be significant.

These aspects are important for the practical implementation of the algorithms, especially if one aims at developing production-quality code that can be used in demanding environments where correctness of the produced solution may have significant (possibly, even life-and-death) consequences. It is less so in prototypical codes like the ones that students may be expected to develop in a course using these lecture notes, especially because most of the effort will be directed to problems that are either unconstrained or have very simple constraints like box ones. Furthermore, the mechanisms for dealing with these issues are not terribly complex and quite well-established, although they would significantly complicate the notation if one would insist in detailing them all the time. For this reason in the rest of these lecture notes these issues will be purposely ignored and the "natural" form of the constraints (if any) will always be used.

1.1.6 Domains and extended-valued functions

A relevant detail has been (purposely) been overlooked in all the previous discussion: a function $f : \mathbb{R}^n \to \mathbb{R}$ may *not* be well-defined for *all* possible input values $x \in \mathbb{R}^n$. This is well-known already for univariate functions, such as:

- $f(x) = 1/x$ is not well-defined for $x = 0$;
- $f(x) = \log(x)$ is not well-defined for $x \leq 0$;
- $f(x) = \sqrt{x}$ is not well-defined for $x < 0$;

and many others. Since multivariate functions are often obtained by the algebraic combination of univariate ones, they may not be well-defined in a "complicated" subset of $\mathbb{R}^n$.

Indeed, for a proper mathematical treatment one should identify the *domain* $D \subseteq \mathbb{R}^n$ of f in which $f(x)$ is well-defined, and always take care that mathematical statements about $f(x)$ are only made for $x \in D$. When multiple functions (e.g., constraint functions) are involved, the *intersection* of their domains should be used to ensure that all function values are available. This usually makes the notation, and the arguments, significantly more complex without, most of the time, adding really significant information for the practical applications of interest in these lecture notes. Here we will therefore take the stance that, unless explicitly stated, all involved functions are well-defined over all $\mathbb{R}^n$: every domain (and hence their intersection) is $D = \mathbb{R}^n$. This is a potentially strong assumption, but can be justified in two (somewhat opposite) ways.

The first is to assume that the functions can be (typically, one-sided) *extended-valued* ones, i.e., $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$. If f is the objective function of a minimisation problem, $f(x) = +\infty$ clearly means "for no reason whatsoever x can be considered as a possible optimal solution", and therefore is in fact equivalent to "$x \notin X$". For a constraint function $g(x) \leq 0$, $g(x) = +\infty$ clearly implies that x is unfeasible. Of course $+\infty$ should be exchanged with $-\infty$ for a maximisation problem or $\geq$ constraint, but §1.1.3 and §1.1.4 have already illustrated how to deal with this. In fact, by this token one could even *completely disregard constraints and assume that an optimization problem is only characterised by its objective.* That is, given some $X \subset \mathbb{R}^n$ one can define its *characteristic function*

$$\imath_X(x) = \begin{cases} 0 & \text{if } x \in X \\ +\infty & \text{if } x \notin X \end{cases}$$

and define the *essential objective* of (P) as $f_X(x) = f(x) + \imath_X(x)$. Then, clearly (P) is mathematically equivalent to the unconstrained problem of minimising f_X over all $\mathbb{R}^n$. Note that if $X = \emptyset$ then $f_X(x) = +\infty$ for all $x \in \mathbb{R}^n$ and "the optimal value is $+\infty$", as expected. Such an approach is not, in practice, as outlandish as it may seem since floating-point numbers like `double` do have an explicit representation for infinite values [54] that obey the rules of extended-value real algebra [11, p. 8] and that are readily accessible by programming languages (e.g., `Inf` in `Matlab` and `std::numeric_limits< double >::infinity()` in `C++`). One could then envision actual implementation of the *oracle* that computes the function values $f_X(x)$ producing $\pm\infty$ values, and this can even work well in some cases.

However, this is not a convenient approach in general. The main reasons, that will be discussed in details later on, are:

1. the function value $f(x)$ is *not* the only information one needs computed by the oracle when devising algorithms for solving optimisation problems: other information (derivatives) is as important (or even more), and it would be "seriously messed up" in functions like $\imath_X$ (that are "ferociously nondifferentiable");
2. efficiently solving optimization problems requires to exploit as much as possible their *structure*, i.e., all available knowledge upon their objective and constraints: "hiding the latter into the former" would conceal a great deal of the structure, possibly destroying many opportunities for building more efficient algorithmic approaches.

For these reasons, in these lecture notes we will keep pretending that all functions have the whole $\mathbb{R}^n$ as domain with the following implicit justification: if f is not defined on some x, then some constraint has been imposed (in the problem-building phase, which is not discussed) so that $f(x)$ will never be evaluated. This is in some sense the opposite approach, in that one is possibly concealing some information about the objective in some properly constructed constraints. Such an approach is only partly satisfactory, as not all algorithmic approaches guarantee that $f(x)$ (and even less the constraint functions) will only be evaluated at feasible points. However, in all the applications of interest here, either all functions are actually well-defined in all $\mathbb{R}^n$, or there are effective ways to "keep out of the danger zone" where they are not. Thus, even if not completely satisfactory mathematically speaking, the simplifying assumption that all functions are defined everywhere is appropriate for these lecture notes, and it will make for a leaner and cleaner treatment. Anyway, this choice is far from being uncommon: many books/articles about optimization problems/algorithms have at the very beginning the apparently inconspicuous statement that "$f : \mathbb{R}^n \to \mathbb{R}$", which in fact means "$f$ is not an extended-value function, and therefore defined everywhere", i.e., "we do not want to bother with domains". These lecture notes just follow this well-established tradition.

1.1.7 An aside: (why not) multivariate optimization

Before moving on with the solution of actual optimization problem, it is appropriate to briefly comment–once and for all–on the underlying assumption that the objective is a real-valued function. In practice, this means that all the "value" of any choice $x \in X$ can be expressed by using a single (real) number. However, it is common experience that, when evaluating a set of choices (e.g., buying a new car/house . . .) there are *multiple criteria* that must be used to evaluate different aspects of the decision. In real life some of these criteria are even hard to be given numerical values; but assuming this is done somehow, the result is that to each $x \in X$ one has naturally associated a *vector* of (real) values $y = [\, y_1 \, , \, y_2 \, , \, \ldots \, , \, y_k \,] \in \mathbb{R}^k$ that represent the evaluation of the k different criteria on x. Thus, $f : \mathbb{R}^n \to \mathbb{R}^k$ is a *vector-valued function*, which is basically nothing else than "a vector of real-valued functions":

$$f(x) = [\, f_1(x) \, , \, f_2(x) \, , \, \ldots \, , \, f_k(x) \,] \; .$$

One is therefore faced with the *multicriteria*, or *vector-valued*, optimization problem

$$(M) \qquad \text{“min”} \{\, [\, f_1(x) \, , \, f_2(x) \, , \, \ldots \, , \, f_k(x) \,] \, : \, x \in X \,\} \; ,$$

with the scare quotes underlying the fact that the meaning of the minimisation is now not obvious. To simplify the discussion we will assume $k = 2$ (bi-criteria optimization problem) since all arguments readily generalise to any $k > 2$.

The issue with vector-valued optimization is clearly that $\mathbb{R}^k$ is not totally ordered: given $y' \neq y''$, it is not always the case that either $y' < y''$ or $y' > y''$ must occur. Two vectors can be *incomparable*, leading to the situation where the objective is not capable to decide between two choices; and if the objective cannot, who can? We use the following textbook example to illustrate the issue.

Example 1.2. The portfolio selection problem

In the financial sector, portfolio managers are given some amount of money and have to decide how to invest it, over some investment period (anything between a lifetime and a few milliseconds), choosing among a large array of financial instruments: government bonds (from different states/regions, with different maturities, ratings, . . .), corporate bonds (from many different emitters in different geographical regions and different industrial sectors, . . .), equities (in many different publicly traded companies in different markets and different industrial sectors, . . .), commodities, ETFs, derivatives, cryptocurrencies, and many others. It is therefore possible to construct (infinitely) many different *portfolios* in which different fractions of the total wealth are invested in different instruments. Let X be the set of all these portfolios, and let us assume for illustration purposes that it is finite. Each $x \in X$ must be evaluated according to *two* different criteria:

- $f_1(x)$ is the *(expected) return* of the portfolio, i.e., how much the instruments will (presumably) be worth at the end of the investment period;
- $f_2(x)$ is the *(estimated) risk* of the portfolio, i.e., a measure of how likely it is that the return will not be the expected one.

Crucially, the two criteria may have different (and incomparable) units of measure: while the first is "money" (say, €) the second may be a probability. This depends upon which of the *risk measures* (variance, VaR, CVaR, Sharpe ratio, . . .) are used, an issue that is clearly way beyond our interest here. As an added twist, f_1 should be maximised while f_2 should be minimised, so one change in sign is also in order. A possible example is depicted in Figure 1.1(left), where each $\star$ marks the position $[\, f_1(x) \, , \, f_2(x) \,]$ corresponding to one $x \in X$ in the return / risk plane.

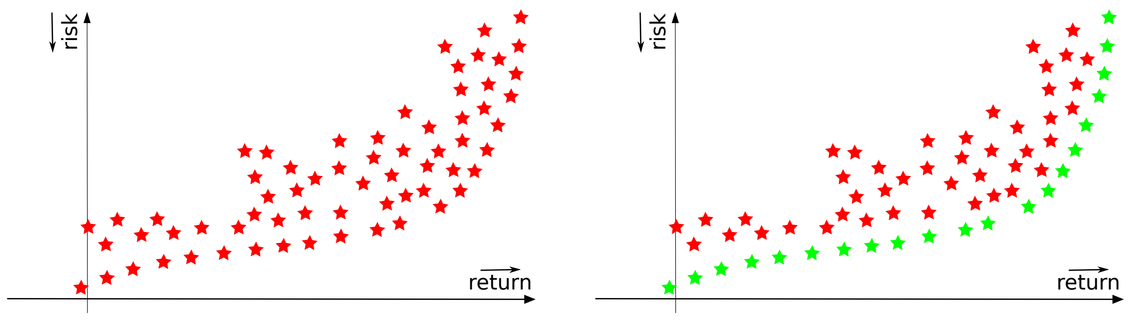


Figure 1.1: A portfolio example (left) and its Pareto frontier (right)

As it can be expected, there is no single portfolio that is better than all others on both criteria: typically, as the return of a portfolio increases (good), so also does its risk (bad). Such *contrasting* objectives—when one improves, the other deteriorates—are very common. What can be identified are portfolios x' that are *dominated* by others, i.e., for which there exist some x'' such that $f_1(x') > f_1(x'')$ and $f_2(x') < f_2(x'')$ (it has better return and lower risk). These can reasonably be excluded from the choices one should make. However, some portfolios x' are *non dominated*: if there exists some x'' such that $f_1(x') > f_1(x'')$ (it has better return), then $f_2(x') > f_2(x'')$ (it has worse risk). These are represented as the green $\star$ in Figure 1.1(right). Basically, these are the portfolios that offer the minimum risk for their given level of return, or, conversely, the maximum level of return for their given level of risk. Without entering in the mathematical details (as usual, the devil can be found in there), these identify the *Pareto-optimal frontier* of the problem. Each non-dominated solution (portfolio) is a rational choice, but which one should be chosen among these depends on the *risk propensity* of the investor: how much (s)he is willing to tolerate risk in view of possible higher returns. This is *not* encoded in any way in the problem, and therefore there is no way in which an algorithm can select one specific solution and declare it "optimal". The idea in this case is that algorithms will produce *all* the solutions in the Pareto-optimal frontier (which can be infinitely many), and somehow present them to another decision-making level (typically, a human being) where the final choice will be made according to some criteria that is unknown, and possibly difficult to formalise.

While the above "interactive" approach can be sensible in some settings, it is typically not in the ones these lecture notes are focused on. Thus, multicriteria (a.k.a. *multiobjective*) optimization proper has to, and will, be avoided. Yet, the contrasting needs of on one hand considering multiple criteria, and on the other hand automatically producing one final answer, can be satisfied by means of some "dirty tricks". The two classical forms are *scalarization* and *budgeting*

$$\begin{array}{ll} (M_\alpha) & \min\{\, f_1(x) + \alpha f_2(x) \,:\, x \in X \,\} \\ (M_2^\beta) & \min\{\, f_1(x) \,:\, f_2(x) \leq \beta \;,\; x \in X \,\} \end{array} .$$

In plain words, in the first one concocts a "magic" constant α that "translates the units of measure of f_1 into those of f_2", allowing to sum them and treat them as a single objective. In portfolio parlance this may be termed a *risk-adjusted return*, i.e., providing a monetary value to the incurred level of risk, as depicted in Figure 1.2(left). In the second approach one is rather setting a bound on the highest possible value of one criteria that can be tolerated (e.g., the maximum risk) and is optimising on the other criteria subject to that constraint (e.g., find the portfolio with maximum return among those with acceptable risk), as depicted in Figure 1.2(right). Note that in the budgeting approach one can freely choose which one of the criteria has to be turned into constraint and which one will be left as the objective; that is, there would as well be the choice of setting a minimum level of return and seeking the least-risk portfolio attaining it.

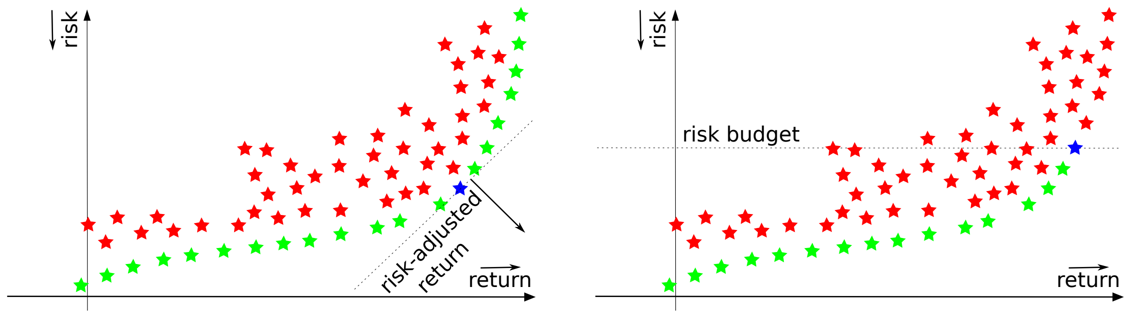


Figure 1.2: Scalarization (left) and budgeting (right)

Of course, the issue in the two approaches is to identify the constants α or β; this typically requires knowledge of the actual decision process, which in the financial sector in particular is heavily regulated (you should not sell high-risk derivatives to an unknowledgeable pensioner ...). Whenever this is possible, multicriteria optimization problems can be recast as ordinary (single-criteria) ones and solved with the tools discussed in these lecture notes.

The two approaches outlined in the Example, and in particular scalarization, are actually used in the applications of interest for these lecture notes. In particular, in (supervised) Machine Learning applications *models* are constructed that should reconstruct an unknown function $h : I \to O$ between some input and some output space (the latter not restricted to be $\mathbb{R}$) out of a set of examples. The chosen models are parametrised over a (large) set of variables (weights) w, and an optimization problem is constructed and solved to find the "best" weights. For this, a *loss function* $L(w)$ is devised that, roughly speaking, measures how well the model, with the given weighs w, reconstructs the function h on the known examples. However, doing so "too well" (that is, optimising $L(w)$ alone) is usually detrimental to the learning outcomes. The issue, very broadly speaking, is that a "complex" model is needed to fully reconstruct h on the given input. However, such a model may "overfit": in striving to perfectly representing h on the known examples it does poorly in its real task, which is to predict it for unknown data. This is called the *bias / variance dilemma*. It turns out that "simpler" models predicting a little (but not too much) less accurately h on the known examples may predict it much more accurately the unknown ones. For this reason a *regularisation term* $R(w)$ is devised that "measures the complexity of the model", and one is in principle confronted with a bi-criteria problem of optimising $[\, L(w)\, ,\, R(w)\,]$. The standard way in which this is approached is precisely scalarization: an arbitrary real (hyper)parameter λ is chosen, and $L(w) + \lambda R(w)$ is optimised. The "right" value of λ is typically unknown, but standard approaches (k-fold cross-validation) are available to "measure how good a given value of λ is" for the "true objective" of having weights w that provide a good approximation of h on all (known and unknown) data. Basically, this is repeatedly done with λ chosen in a finite (small-ish) set of values (grid search), and the best value is retained as the final choice. None of the crucial details of this process are of interest for these lecture notes, since they will be dealt with in Machine Learning courses.

What is relevant here is that the objective function $f(\, x\,)$ that is optimised, or some of the constraints, may well be the result of a scalarization or budgeting approach applied to a multi-criteria problem. Whether or not this is the case, and how the crucial (hyper)parameters α and β have been choses, is completely outside the scope of these lecture notes and supposedly decided during the *modelling phase* in which the optimization problem has been constructed. While the proper construction of (optimization) models is an interesting subject in itself, it will not be pursued in details here. The rationale is that for the applications of interest here a relatively small set of models have been identified over the years that "work well", and the issue is rather how to solve them efficiently. This is mostly what these lecture notes are about.

1.2 (Outrageously) Simple (Univariate) Optimization

We will now proceed with the exercise of picking up two extremely simple classes of *univariate* functions (i.e., x is a single real number: $n = 1$), linear and quadratic ones, and discuss solving optimization problems with them. Although the arguments here are so simple as to be almost offensive, the ability of solving "simple" problems is typically crucial when solving "complex" ones. In particular, we will see that solving problems with general functions typically entails the use of *models*, i.e., simpler functions that approximate the given one close to some given points. The useful models are linear and quadratic functions, so it makes sense to just quickly review the properties of these, if only to set the nomenclature right, and be done with it. Skipping this section on first reading, and coming back to it only if needed, is a sensible option. However, when dealing with quadratic function the section introduces the first example of a simple but interesting mechanism that will be repeatedly useful in the following.

1.2.1 Linear univariate functions

Linear univariate functions are perhaps the simplest possible imaginable ones: they have the form $f(\, x\,) = bx$ for some *fixed* value $b \in \mathbb{R}$. There are, therefore, as many different linear functions as real numbers. These are in fact *homogeneous* linear functions; a different definition that is often also used is that of $f(\, x\,) = bx + c$, i.e., allowing some constant term. These are called *affine* functions. Due to the discussion in §1.1.3, constant terms are irrelevant when f is the objective of an optimization problem and we therefore dispense with c. b is called the *linear coefficient* or *slope* of the function. Linear functions are very easy to visualise by plotting their *graph*, i.e., the (two-dimensional) curve

$$\mathrm{gr}(\, f\,) = (\, (\, f(\, x\,)\, ,\, x\,)\; :\; x \in \mathbb{R}\,) \subset \mathbb{R}^{n+1}\ .$$

This is done in Figure 1.3 for some different values of b.

The picture makes it plain how the function behaves, starting of course from justifying its name. In terms of optimization, a fundamental property one is interested in is whether the function is *(strictly) increasing / non decreasing / constant / non increasing / (strictly) decreasing* on some (not necessarily bounded) interval $[\, \underline{x}\, ,\, \overline{x}\,] \subseteq \mathbb{R}$, which trivially means that

$$f(\, x\,) > / \geq / = / \leq / < f(\, z\,) \qquad \forall\, \underline{x} \leq x < z \leq \overline{x}\ .$$

This is obviously related with b: f is increasing / constant / decreasing if b is > 0 / $= 0$ / < 0. Albeit trivial, one may want to step aside for a second to prove the fact, is only to start getting used to the fact that every mathematical statement made in these lecture notes can eventually be proven.

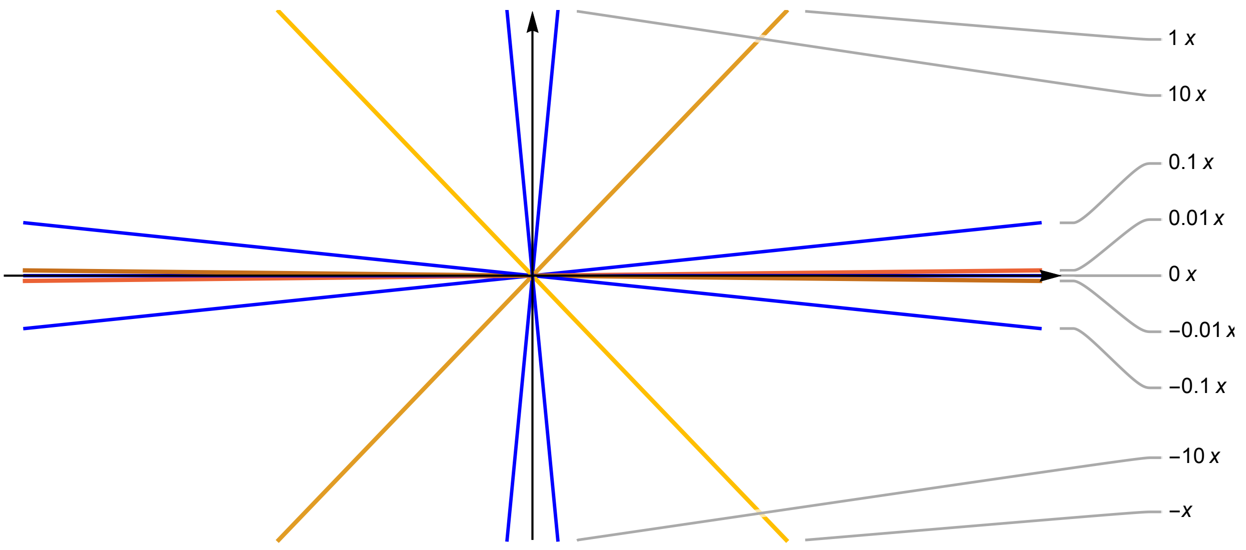


Figure 1.3: The graph of linear functions for some values of b

Exercise 1.2. Formally prove the stated properties. **[solution]**

Optimising a linear function without any constraint is obviously pointless: the minimization problem is unbounded below ($\nu(P) = -\infty$) and the maximization one unbounded above ($\nu(P) = +\infty$) unless of course $b = 0$, in which case $\nu(P) = 0$ and every $x \in \mathbb{R}$ is an optimal solution. This can be made ever so slightly more amusing by considering the box-constrained optimization

$$(L) \quad \min\{\, bx \,:\, \underline{x} \leq x \leq \overline{x} \,\}$$

on a not necessarily bounded interval, i.e., allowing $\underline{x} = -\infty$ and/or $\overline{x} = +\infty$ to happen. The result is fairly obvious: thanks to the rules of extended-real arithmetic, if $b > 0$ then $\nu(L) = f(\,\underline{x}\,)$, if $b < 0$ then $\nu(L) = f(\,\overline{x}\,)$, and if $b = 0$ then $\nu(L) = 0 = f(\,\underline{x}\,) = f(\,\overline{x}\,)$. Intuitively, if the function is increasing the optimal solution is on the left endpoint of the interval (the function "grows when going from left to right"), if the function is decreasing the optimal solution is on the right endpoint of the interval (the function "decreases when going from left to right"). One could even say that $b > 0$ then $x^* = \underline{x}$, if $b < 0$ then $x^* = \overline{x}$, and if $b = 0$ then x^* is any point in the interval, accepting the slight abuse of notation that $\pm\infty$ is the optimal solution in case of an unbounded problem.

Exercise 1.3. Formally prove the above result. **[solution]**

Ultimately, optimizing a linear function over a possibly unbounded interval can only be defined as trivial: an $O(\,1\,)$ closed formula achieves it. We avoid commenting again on the issue that "$b > 0$", "$b = 0$" and "$b < 0$" need be treated with some care over floating-point numbers. Thus, as expected optimization of (exceedingly) simple functions is (exceedingly) easy. We will see this happening for a while, and breaking surprisingly soon.

1.2.2 Homogeneous quadratic univariate functions

The next simplest functions besides linear ones are the *homogeneous quadratic* ones: $f(\,x\,) = ax^2$ for some *fixed* value $a \in \mathbb{R}$. Again, there are as many different homogeneous quadratic functions as real numbers, ignoring the possibility of adding constant terms. a is called the *quadratic coefficient* or *curvature* of the function. The graphs plotted in Figure 1.4 basically tell all the story.

There are two different cases, excluding the special one in which $a = 0$ and the function is constant (and therefore linear): $a > 0$, in which the graph is a $\cup$-shaped parabola, and $a < 0$, in which the graph is a $\cap$-shaped parabola. The first is called *convex* and the other is called *concave*, as special cases of extremely important definitions with which it is useful to get familiar as early as possible. Similarly to the linear case, the *sign* of the curvature is crucial to determine the overall shape of the function (and, therefore, drive its minimization or maximization), where the *magnitude* influences "how quickly the function increases / decreases", i.e., "how steep the parabola is". The fundamental property for our purposes is: if $a > 0$ then f is decreasing for $x \leq 0$ and increasing for $x \geq 0$, while if $a < 0$ then (vice-versa) f is increasing for $x \leq 0$ and decreasing for $x \geq 0$.

Exercise 1.4. Formally prove the stated properties. **[solution]**

The optimization of a homogeneous quadratic function

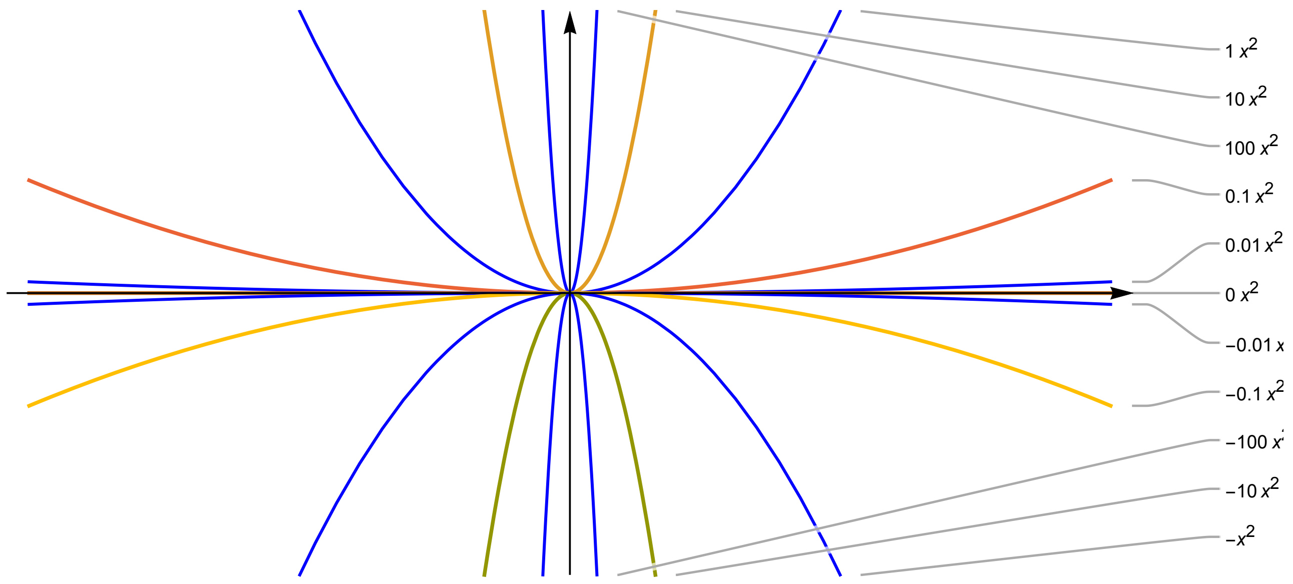


Figure 1.4: The graph of homogeneous quadratic functions for some values of a

$$(Q) \quad \min\{\, ax^2 \,:\, \underline{x} \leq x \leq \overline{x} \,\}$$

is therefore only slightly less trivial than that of a linear one in the unconstrained case, i.e., where $\underline{x} = -\infty$ and $\overline{x} = +\infty$:

- if $a > 0$ then $\nu(Q) = 0$ and the unique optimal solution is $x^* = 0$;
- if $a < 0$ then $\nu(Q) = -\infty$.

Obviously, the rule reverses for maximization: $x^* = 0$ is the unique optimal solution if $a < 0$, and (Q) is unbounded above if $a > 0$. The "true" box-constrained case, where $\underline{x}$ and $\overline{x}$ can have finite values, is ever so slightly more nuanced. For minimization, and $a > 0$, there are already three different cases, depending on "where the extremes stand w.r.t. 0":

- if $\overline{x} < 0$ ($\Longrightarrow \underline{x} < 0$) then $\nu(Q) = f(\,\overline{x}\,)$;
- if $\underline{x} > 0$ ($\Longrightarrow \overline{x} > 0$) then $\nu(Q) = f(\,\underline{x}\,)$;
- if $\underline{x} \leq 0 \leq \overline{x}$ then $\nu(Q) = 0$.

In plain words, the optimal solution is 0 if it is feasible, and otherwise the one of the two extremes that is closer to it. Not surprisingly, the situation is *not* entirely symmetric if (Q) is a maximization problem

- if $\overline{x} < 0$ ($\Longrightarrow \underline{x} < 0$) then $\nu(Q) = f(\,\underline{x}\,)$;
- if $\underline{x} > 0$ ($\Longrightarrow \overline{x} > 0$) then $\nu(Q) = f(\,\overline{x}\,)$;
- if $\underline{x} \leq 0 \leq \overline{x}$ then $\nu(Q) = \max\{\, f(\,\underline{x}\,)\,,\, f(\,\overline{x}\,)\,\}$.

In plain words, if 0 is not feasible then the optimal solution is the extreme that is *farthest* from 0, otherwise one have to look to the value of both extremes and pick the largest (this basically again boils down to find the one that is farthest). As for the linear case, the formulæ work even if any of the extremes is $\pm\infty$, and one can turn them in formulæ for x^* if accepting $\pm\infty$ answers in the unbounded cases.

Exercise 1.5. Formally prove the above result, state and prove the case $a < 0$ **[solution]**

Again, optimizing a homogeneous quadratic function over a possibly unbounded interval requires trivial $O(\,1\,)$ closed formulæ. They are, however, not symmetric for minimization and maximization, with the maximization of a concave function ever so slightly more complex than its minimisation. As we shall see, this is only the presage of much more significant differences.

1.2.3 Quadratic non-homogeneous univariate functions

The natural next step is the *non-homogeneous* quadratic function $f(\,x\,) = ax^2 + bx$. Needless to say, this generalise the previous cases where only one among the slope and the curvature were nonzero. Basically, a non-homogeneous quadratic function is just the sum of a homogeneous quadratic and a linear one: however, for the avoidance of doubt let us state upfront that minimizing it is *not* equivalent to minimizing the two components separately and then "putting the results together" somehow. In general—and save a few relevant cases that we will discuss—an optimization problem is more complex than the sum of its individual parts.

Optimizing a non-homogeneous quadratic function (with both $a \neq 0$ and $b \neq 0$) cannot obviously be overly too complicated than an homogeneous one, but it is still somewhat different. While there are many ways to address this, we exploit the chance to introduce in a very simple setting a general concept that will repeatedly prove useful: *if $f(x)$ is "too complicated", make it "simpler"*. That is, *reformulate* (cf. §1.1.3 and §1.1.4) the problem in a form that makes it simpler to solve. A powerful way of doing this, in general, is that of *changing the space of variables*. In this simple case the idea is just to change (translate) the input space so that f becomes homogeneous.

The idea is to pick some fixed $\bar{x}$ and to translate the variable x so that $\bar{x}$ is the origin. That is, we define a new variable z by $z = x - \bar{x}$, which just means $x = z + \bar{x}$. With simple algebra we can then rewrite f as a function of this new variable:

$$f(x) = a(z + \bar{x})^2 + b(z + \bar{x}) = az^2 + 2az\bar{x} + a\bar{x}^2 + bz + b\bar{x} = az^2 + (2a\bar{x} + b)z + f(\bar{x}) = g(z) \ .$$

It is then plain to see that by choosing $\bar{x} = -b / 2a$ (which is possible since $a \neq 0$) one obtains an homogeneous quadratic function in the z-space (save the constant term $f(\bar{x})$ that we know we can ignore). This immediately provides the complete solution to the optimization problem by just applying the formulæ of §1.2.2 to g in the z-space and then translating back the solution in x-space. Graphically, a non-homogeneous quadratic function is still a parabola (be it convex or concave), except its *apex* is not in the origin, but rather in $\bar{x} = -b / 2a$. The point $x = 0$ is still a *root* of f (i.e., such that $f(x) = 0$), but while in the homogeneous case the apex is the only root, in the non-homogeneous one there are two separate ones. The z-space is chosen so that its origin coincides with the apex, and the constant $f(\bar{x})$ ensures it is the only root. All in all, by a "clever trick" the optimization of a non-homogeneous quadratic function can still be performed in $O(1)$.

Exercise 1.6. Discuss the position of $\bar{x}$ and of the roots of f depending on a and b [**solution**]

1.3 (Not always) Simple Multivariate Optimization

We now move to "simple" multivariate optimization: $f : \mathbb{R}^n \to \mathbb{R}$ with f a "simple" (linear or quadratic) function. Even when the problem remains a "simple" one—and, as we shall see, this is often but not always the case—the complexity of solving it cannot be dismissed as irrelevant right away because it cannot but depend on n. While examples will be given in small (2, 3, 100) dimension, practical applications may require largish sizes (10^4), large sizes (10^6) or *heinously large* ones (10^9, 10^{11}). Thus, by dint of their sheer size alone, multivariate optimization problems can be costly even for "simple" functions. In some cases, however, the cost grows linearly with n, which is the best one can hope for.

1.3.1 Linear multivariate functions

A linear function $f : \mathbb{R}^n \to n$ is defined as $f(x) = \langle b , x \rangle = \sum_{i=1}^{n} b_i x_i$, i.e., as the scalar product between the vector of variables $x \in \mathbb{R}^n$ and a fixed vector $b \in \mathbb{R}^n$. In other words, $f(x)$ is the sum of n univariate linear functions, one in each of the variables. It is expedient to define the set of indices of variables $I = [\, 1 , 2 , \ldots , n \,]$. A linear function has the following properties:

1) $f(\gamma x) = \gamma f(x)$ however chosen $\gamma \in \mathbb{R}$ and $x \in \mathbb{R}^n$;
2) $f(x + z) = f(x) + f(z)$ however chosen $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$.

This is easy to prove by the fundamental properties of the scalar product (cf. §A.2); in fact, the two properties are an alternative definition of linear function.

Exercise 1.7. Prove that if $f(x)$ has the properties i. and ii. then it necessarily has the form $f(x) = \langle b , x \rangle$ for some fixed $b \in \mathbb{R}^n$. Discuss if this extends to *affine* functions, i.e., of the form $f(x) = \langle b , x \rangle + c$. [**solution**]

Linear functions are "very simple" ones, as one can easily verify by picturing two-dimensional cases. The *graph* $\mathrm{gr}(f)$ for a bi-variate linear function ($f : \mathbb{R}^2 \to \mathbb{R}$) is a *plane* in $\mathbb{R}^3$, as shown in Figure 1.5(left). It becomes an *hyperplane* in $\mathbb{R}^{n+1}$ in the general case $f : \mathbb{R}^n \to \mathbb{R}$. The *level sets* $L(f , v)$ in $\mathbb{R}^2$ are parallel lines orthogonal to the vector b defining f, as shown in Figure 1.5(right); these become parallel hyperplanes in $\mathbb{R}^n$ in the general case $f : \mathbb{R}^n \to \mathbb{R}$.

Picturing functions with more than two variables is challenging; level sets of function of three variables can be represented in 3D pictures, but that's about the maximum (moving pictures allow to reach four variables, but

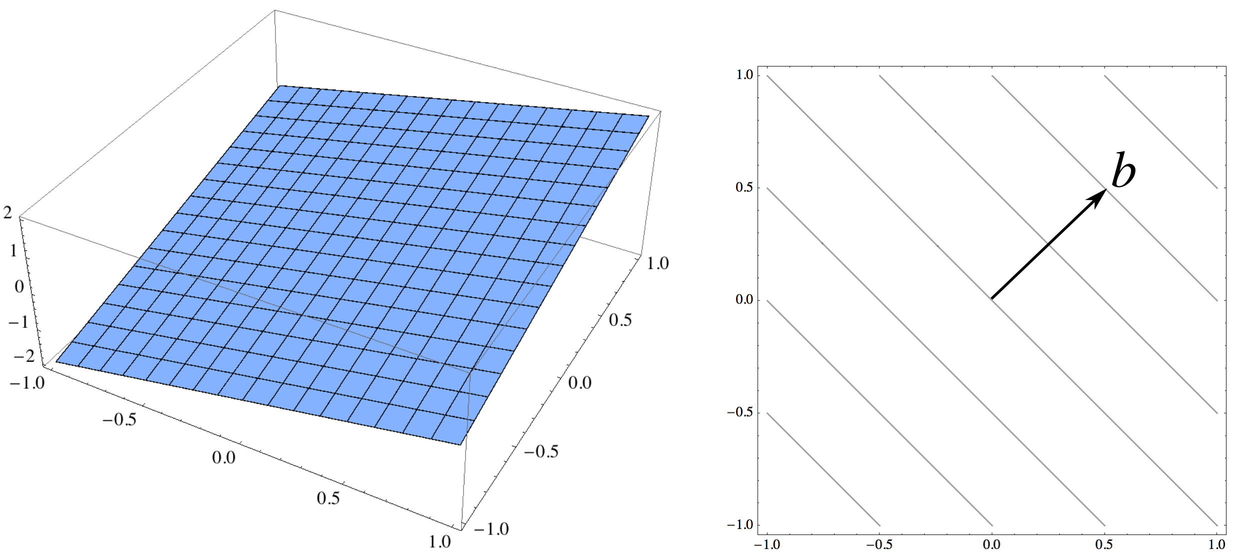


Figure 1.5: Graph and level sets of the linear function $f(x) = x_1 + x_2$

they are not easy to manage). A simple concept that is useful to visualise—and, actually, optimise—functions with any number of variables is the *tomography* of the function. This is obtained by choosing some fixed $x \in \mathbb{R}^n$ (the origin) and $d \in \mathbb{R}^n$ (the direction), and building the univariate function

$$\varphi_{x,d}(\alpha) = f(x + \alpha d) : \mathbb{R} \to \mathbb{R} \ .$$

Here, the value α is the *step* that identifies a point obtained by starting from x and moving "α units" along d. Often, but not always, the tomography is only considered "in the direction of d", i.e., for $\alpha \geq 0$. Indeed, clearly, d has to be interpreted as a direction, which means that $\| d \|$ does not really change it except for a "scale factor": formally speaking, $\varphi_{x,\beta d}(\alpha) = \varphi_{x,d}(\beta\alpha)$. As a consequence, often (but not always) it is convenient to use a normalised direction ($\| d \| = 1$). It is also often useful to consider the (vertically) *translated* tomography defined by $\varphi_{x,d}(\alpha) = f(x + \alpha d) - f(x)$, so that $\varphi_{x,d}(0) = 0$; which one of the two definitions is used will be made clear from the context. Similarly, in many cases x and / or d will be clear from the context and simplified notations like $\varphi(\alpha)$ will be used. One advantage of the tomography is that $\mathrm{gr}(\varphi_{x,d})$ can always be pictured, allowing to get some insight on the features of the function. Of course, the issue is that this depends on the choice of x and d. Often, however, there will be "natural" choices for the origin and the direction that will simplify the analysis.

For instance, in the case of linear functions the translated tomography is

$$\varphi_{x,d}(\alpha) = f(x + \alpha d) - f(x) = \langle b \, , \, x + \alpha d \rangle - \langle b \, , \, x \rangle = \alpha \langle b \, , \, d \rangle \ .$$

Hence, the choice of the centre x does not change the tomography, confirming that linear functions "look similar everywhere". Then, the tomography of a multivariate linear function is a univariate linear function whose slope is $\langle b \, , \, d \rangle$. This immediately allows, for instance, to formally verify the proprtiey suggested by Figure 1.5(right) that the level sets of f are lines (in $\mathbb{R}^2$, planes in $\mathbb{R}^3$, hyperplanes in general) orthogonal to b. In fact, consider some x and the corresponding level set $L(f \, , \, f(x))$. For any direction d orthogonal to b, $\varphi_{x,d}(\alpha) = \alpha \langle b \, , \, d \rangle = 0$ (since $\langle b \, , \, d \rangle = 0$, cf. §A.2), i.e., e constant function. Thus, all points $x + \alpha d$ have the same function value and therefore belong to the level set. Conversely, for any z belonging to the level set one has $f(x) = f(z)$, i.e., $\langle b \, , \, x \rangle = \langle b \, , \, z \rangle$, i.e., b is orthogonal to $z - x$.

Regarding the minmisation of a linear function, the unconstrained case is trivial (the problem can only be unbounded) so let us immediately consider the box-constrained case. That is bounded if the box is *compact* (all bounds are finite), but also trivial due to an important property that we encounter for the first—but by means not last—time and that is worth commenting: the problem is *decomposable*. Formally,

$$\min\{ \langle b \, , \, x \rangle \ : \ \underline{x} \leq x \leq \overline{x} \} = \textstyle\sum_{i \in I} \min\{ b x_i \ : \ \underline{x}_i \leq x_i \leq \overline{x}_i \} \ .$$

In plain words, the variables "do not talk to each other": the optimal choice for the variable x_i and that for any other variable x_j with $j \neq i$ are completely independent on one another. This means that one is not facing a "monolithic" problem in n variables, but rather n independent problems, each with just one variable. This is clearly a very desirable property, since of course the solution of many smaller problems cannot be more costly than that of the large one—if only because the former can be performed in parallel, yielding a prime example of "embarrassingly parallel" computation. Furthermore, in this particular case the solution of the individual problems is trivial by means of the formulæ devised in §1.2.1. This is problem is therefore as simple as one can hope an n-variables optimization problem to be; indeed its solution would hardly merit being discussed, were it not for the opportunity to introduce some concepts that will be useful later on.

1.3.2 Very simple quadratic functions: separable (non-homogeneous)

We immediately put the last concept into action by briefly considering a "very simple" class of multivariate quadratic functions: the *separable* (non-homogeneous) quadratic ones

$$f(x) = \sum_{i \in I} [\, f_i(x_i) = a_i x_i^2 + b_i x_i \,] \ ,$$

i.e., the sum of n (non-homogeneous) univariate quadratic functions, one over each of the variables. This is then easy to optimise it, possibly over a (bounded or not) box $\underline{x} \leq x \leq \overline{x}$, by applying the formulæ of §1.2.3 to each x_i individually. However, the fact that there are multiple variables opens the way to a somewhat variegated panorama:

- the minimisation problem is guaranteed to have an optimal solution only if both bounds are finite ($-\infty < \underline{x}_i \leq \overline{x}_i < +\infty$) whenever $a_i < 0$, for otherwise the minimisation of x_i alone is unbounded below; the symmetric condition holds for the maximisation problem, which means that there are cases where neither is bounded;
- hence, in the unconstrained cases the problem is bounded only if minimising with $a_i > 0$ for all $i \in I$, or maximising with $a_i < 0$ for all $i \in I$; in fact the problem can be bounded even if $a_i = 0$ for some i, but only if also $b_i = 0$ for those same i (the objective value does not really depend on those variables, that can therefore take any value).

It is somewhat curious, and instructive, that such a simple problem can already give rise to a number of different cases. Although the problem is "too simple to be interesting", its study helps understanding the significantly more complex "true quadratic problems" discussed in the subsequent sections. For this reason alone we keep illustrating some of its features.

The first observation is that whenever $a_i \neq 0$ for all $i \in I$, then it is clearly possible to restrict the analysis to the homogeneous case. Indeed, by applying the trick of §1.2.3 one can rewrite the function as $f(z) = \sum_{i=1}^{n} a_i z_i^2$ (ignoring constant terms) where $z = x - \bar{x}$ with $\bar{x} = [\, -b_1 \,/\, 2a_1 \,,\, -b_2 \,/\, 2a_2 \,,\, \ldots \,,\, -b_n \,/\, 2a_n \,]$. Again, basically one is translating the space so that the origin is moved to $\bar{x}$, and the function is homogeneous in that space. The case where some $a_i = 0$ deserves some discussion, but one could argue that it is "irrelevant", as soon as optimization is concerned. Indeed, either $b_i \neq 0$, and therefore the problem (irrespectively if minimising or maximising) is unbounded, or $b_i = 0$ and therefore the variable x_i is irrelevant and can be fixed to any value (say, 0). This allows us to concentrate our discussion on the homogeneous case.

Clearly the sign pattern of the a_i is crucial for the "shape" of the function. This is illustrated in Figure 1.6 for bi-variate examples:

- On the top left the graph of $f(x) = 3x_1^2 + x_2^2$, i.e., both $a_1 = 3 > 0$ and $a_2 = 1 > 0$, is shown. The graph is a three-dimensional (convex) "paraboloid". To better understand its shape one can consider the special form of tomography corresponding to the restriction along i-th coordinate: that is, $\varphi_{0,[\,1\,,\,0\,]}(\alpha) = f(\alpha\,,\,0) = 3\alpha^2$ and $\varphi_{0,[\,0\,,\,1\,]}(\alpha) = f(0\,,\,\alpha) = \alpha^2$. This immediately extends to any other center than 0, save for a constant term. Thus, for the direction along the one axis x_i the tomography is just the parabola with the corresponding a_i, as the figure clearly shows. The case with $a_1 < 0$ and $a_2 < 0$ would be analogous but "upside-down": the parabolas would be concave and 0 would be the maximum rather than the minimum.
- On the top right the graph of $f(x) = 3x_1^2 - x_2^2$, i.e., with $a_1 = 3 > 0$ but $a_2 = -1 < 0$, is shown. The graph is an "hyperboloid", or "saddle-shaped", and 0 is neither a maximum nor a minimum (it is called a *saddle point*, and the picture clearly justifies the name). This is because on one variable the tomography (restriction along i-th coordinate) is a convex parabola, but on the other is a concave one.
- On the bottom left the graph of $f(x) = 3x_1^2$, i.e., with $a_1 = 3 > 0$ but $a_2 = 0$, is shown. This is a "degenerate paraboloid": on one variable the tomography is a convex parabola, but on the other it is a constant function. In this case $[\,0\,,\,0\,]$ is still a minimum, but so are all points $[\,0\,,\,x_2\,]$ however chosen x_2. This is because $b_2 = 0$, which can *not* be assumed without loss of generality since $a_2 = 0$.
- Indeed, on the bottom left the graph of $f(x) = 3x_1^2 + x_2$, i.e., with the same a_i pattern but non homogeneous on x_2, is shown. This is still a "degenerate paraboloid", but "not flat in the x_2 variable". In fact, $\varphi_{0,[\,0\,,\,1\,]}(\alpha) = f(0\,,\,\alpha) = b_2\alpha$, i.e., the tomography (restriction along 2nd coordinate) is linear (and *not* constant), and the function value is bounded neither below nor above.

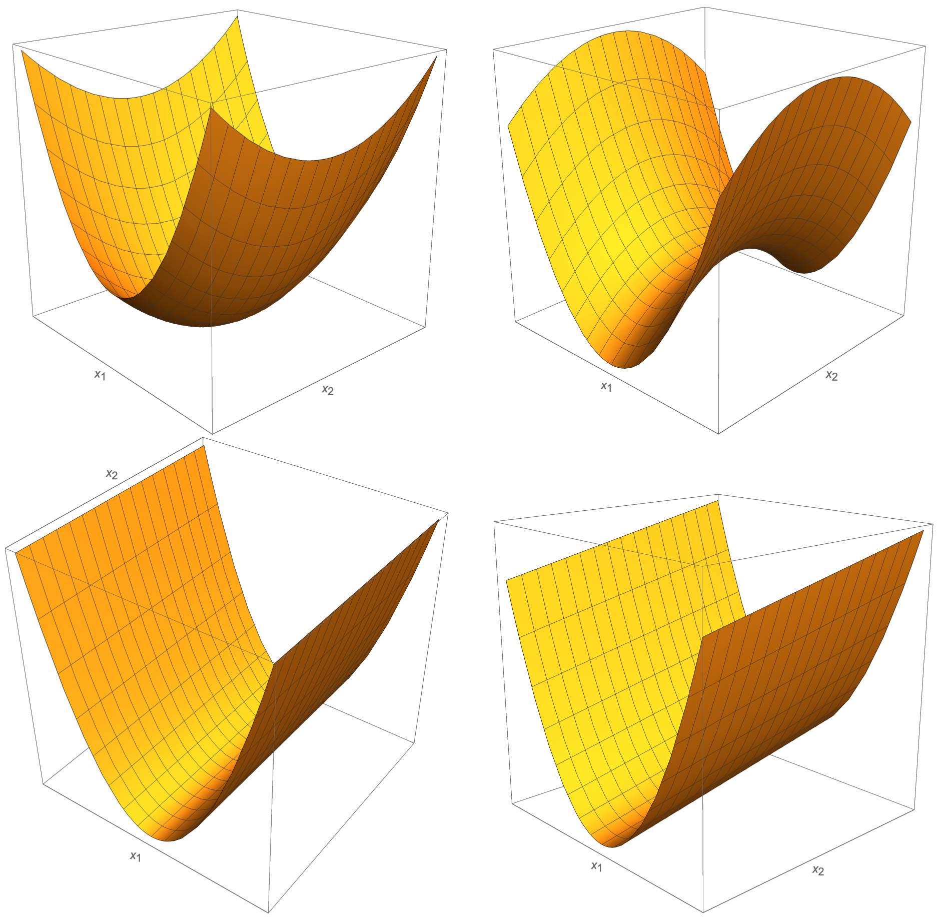


Figure 1.6: Graphs of four examples of separable quadratic functions

A final aspect that is useful to discuss is the impact of the a_i on the "shape" of the function. We focus on the case where both a_i are non-negative: then, the level sets of the function are ellipses (ellipsoids for general n) whose axes are aligned with the Cartesian ones. This is shown in Figure 1.7 and it is kept as an observation, without proof, since it will come as an immediate consequence if the more general analysis in the next section. The Figure shows $L(\, f \,,\, 1\,)$, i.e., the level sets at $v = 1$, of the parametric function $f(\, x\,) = a_1 x_1^2 + x_2^2$ for some different values of a_1 (while keeping $a_2 = 1$). The picture shows a clear trend: the larger a_1 the shorter the x_1 axis of the ellipse, and vice-versa. In fact, for $a_1 = 0$ the ellipse is "degenerate along x_1", with an "infinitely long axis" and therefore turns into two parallel lines. The rationale for this is made clear by considering, e.g., the point where the x_1 axis meets the level set: this is for $f(\, x_1 \,,\, 0\,) = a_1 x_1^2 = 1$, i.e., $x_1 = \pm\sqrt{1 \,/\, a_1}$. Thus, the smaller a_1 the farther from 0 this point is, and vice-versa, up until for $a_1 = 0$ the two only meet "infinitely far away". This chimes well with Figure 1.6(bottom left), showing that if $a_i = 0$ then the graph is "infinitely elongated" along the axis of the corresponding variable. While this analysis may seem rather pointless for a function that is so easy to optimize, it is a good introduction to the "real" quadratic functions to be discussed next.

1.3.3 The multivariate homogeneous quadratic function

The "real" quadratic function whose optimization is worth studying is the *nonseparable* one, whereby the impact of a variable x_i on the objective value *does* depend on the other variables x_j for $j \neq i$. This in practice means that there will be *bilinear* terms depending on the product $x_i x_j$. These functions are arguably "significantly more complex" than all those we have seen before since they "depend on a lot more data": that is, they are characterised by a $n \times n$ matrix Q, i.e., n^2 real numbers as opposed at most $2n$ in any previous case. Formally

$$f(\, x\,) = x^T Q x = \left[\, \textstyle\sum_{i \in I} Q_{ii} x_i^2 + \sum_{i \in I} \sum_{j=1,\, j \neq i}^{n} Q_{ij} x_i x_j \,\right]$$

(we assume minimal familiarity with matrix/vector operations, cf. §A.3). However, w.l.o.g. one can assume Q *symmetric*, i.e., $Q_{ij} = Q_{ji}$ for all different pairs $(\, i \,,\, j\,)$: in fact,

$$x^T Q x = [\,(x^T Q x) + (x^T Q x)^T\,] \,/\, 2 = x^T [\,(Q + Q^T)\,/\,2\,] x$$

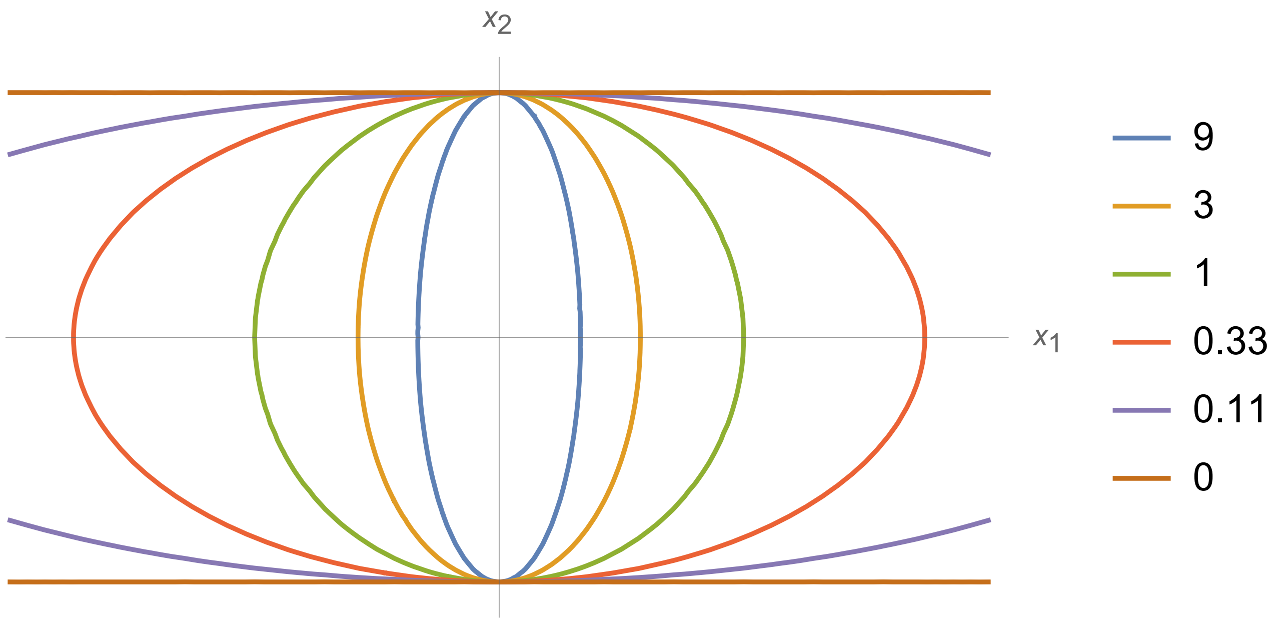


Figure 1.7: $L(\, f \,,\, 1\,)$ of $f(\,x\,) = a_1 x_1^2 + x_2^2$ for some values of a_1

and $(Q + Q^T)\,/\,2$ is symmetric; in plain words, if Q is not symmetric to start with one can just replace Q_{ij} with $(Q_{ij} + Q_{ji})\,/\,2$ and forget about Q_{ji}. This is advantageous if only because it requires roughly half of the data of a non-symmetric matrix to characterise the function. Not surprisingly, such a function is *not* linear: $f(\,x+z\,) = (\,x+z\,)^T Q(\,x+z\,) = f(\,x\,) + f(\,z\,) + 2z^T Qx \neq f(\,x\,) + f(\,z\,)$. It would be natural to consider the non-homogeneous version with an extra linear term, but this extension will be examined later: not doing it ensures that the function itself is symmetric, i.e., $f(\,x\,) = f(\,-x\,)$ for all $x \in \mathbb{R}^n$. This immediately implies that the level sets of the function are "centred in $x = 0$", and therefore suggest to consider the tomography with 0 centre

$$\varphi_d(\,\alpha\,) = f(\,\alpha d\,) = \alpha^2(\,d^T Q d\,)\ .$$

which is then a homogeneous quadratic univariate function whose curvature is $d^T Q d$. Thus, in order to understand the "shape" of this function it is necessary to characterise how the number $d^T Q d$ varies when d changes. This is well known to be related to the *spectral decomposition* of Q

$$Q = \lambda_1 H_1 H_1^T + \ldots + \lambda_n H_n H_n^T \tag{1.8}$$

where $H_i \in \mathbb{R}^n$ are its *eigenvectors* and λ_i are the corresponding *eigenvalues*, i.e., $QH_i = \lambda_i H_i$, for all $i \in I$. *Since Q is symmetric* $\lambda_i \in R$ (otherwise they could be complex numbers, a case we will always carefully avoid). It is obvious that the norm of the eigenvectors is irrelevant, since $QH_i = \lambda_i H_i$ implies $Q(\alpha H_i) = \lambda_i(\alpha H_i)$; thus eigenvectors can always be considered normalised ($\|\, H_i \,\| = 1$). They can also always be taken as orthogonal to each other, i.e., $\langle\, H_i \,,\, H_j \,\rangle = 0$ for all $i \neq j$; in other words, $H^T H = I$ (H is *orthonormal*). Since they are also necessarily linearly independent, they therefore form an orthonormal basis of $\mathbb{R}^n$ (cf. §A.2).

Exercise 1.8. Prove the linear independence of the eigenvalues [**solution**]

Thus, with $\Lambda = \mathrm{diag}(\,\lambda_1\,, \ldots,\, \lambda_n\,)$, (1.8) can be rewritten as $Q = H\Lambda H^T$.

Exercise 1.9. Prove the above result. [**solution**]

It is customary to consider (w.l.o.g.) eigenvalues ordered in non increasing sense, i.e., $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$: λ_1 is the maximum eigenvalue, λ_n the minimum. Again, this crucial property would break if Q were non-symmetric and its eigenvalues were *complex* numbers, which—by dint of being *pairs* of real values (cf. §1.1.7)—are not totally ordered. Anyway, the spectral decomposition immediately allows to characterise products $d^T Q d$ (for $d \neq 0$). It is useful to consider the direction $v = d\,/\,\|\,d\,\|$ corresponding to d: hence, $d = \|\,d\,\| v$. Since the eigenvectors are a basis of $\mathbb{R}^n$, one has

$$v = \alpha_1 H_1 + \ldots + \alpha_n H_n$$

for a vector $\alpha = [\,\alpha_i\,]_{i=1}^n \in \mathbb{R}^n$ of multipliers, where

$$1 = \|\,v\,\| = \|\,\alpha_1 H_1 + \ldots + \alpha_n H_n\,\| = \alpha_1^2 + \ldots + \alpha_n^2 = \|\,\alpha\,\|^2$$

because $\langle\, H_i \,,\, H_j \,\rangle = 0$ for all $i \neq j$ and $\|\, H_i \,\| = 1$ for all i. Then, by the same token

$$d^TQd = \| d \|^2[\, v^TQv \,] = \| d \|^2[\, \alpha_1^2\lambda_1 + \ldots + \alpha_n^2\lambda_n \,] \tag{1.9}$$

which immediately gives

$$\lambda_1 \geq d^TQd \,/\, d^Td \geq \lambda_n \tag{1.10}$$

a relationship that will be repeatedly useful. Indeed this gives the *variational characterization of eigenvalues*

$$\begin{aligned} \lambda_1 &= \max\{\, d^TQd \,/\, d^Td \,:\, d \neq 0 \,\} = \max\{\, d^TQd \,:\, \| d \| = 1 \,\} \\ \lambda_n &= \min\{\, d^TQd \,/\, d^Td \,:\, d \neq 0 \,\} = \min\{\, d^TQd \,:\, \| d \| = 1 \,\} \end{aligned} \tag{1.11}$$

by showing that $d = H_1$ and $d = H_n$ attain respectively the maximum and minimum value in (1.10). Thus, the sign of d^TQd is characterised by that of the eigenvalues of Q, giving rise to five different cases:

- $Q \succ 0$, i.e., is *positive definite*, if all eigenvalues are positive ($\lambda_n > 0$), which implies that $d^TQd > 0$ however chosen d ($\neq 0$);
- $Q \succeq 0$, i.e., is *positive semidefinite*, if all eigenvalues are non-negative ($\lambda_n \geq 0$), which implies that $d^TQd \geq 0$ however chosen d;
- $Q \succ\!\prec 0$, i.e., is *indefinite*, if some eigenvalues are positive and some are negative ($\lambda_1 > 0$ and $\lambda_n < 0$), which implies that d^TQd can be both positive and negative according to how d is chosen;
- $Q \preceq 0$, i.e., is *negative semidefinite*, if all eigenvalues are non-positive ($\lambda_1 \leq 0$), which implies that $d^TQd \leq 0$ however chosen d;
- $Q \prec 0$, i.e., is *negative definite*, if all eigenvalues are negative ($\lambda_1 < 0$), which implies that $d^TQd < 0$ however chosen d ($\neq 0$).

This provides a clear way to characterize the solution of the unconstrained optimization (minimization) problem

$$(QU) \quad \min\{\, f(\, x \,) = x^TQx \,:\, x \in \mathbb{R}^n \,\} \,.$$

It is instructive to consider the tomography along the eigenvectors, i.e.,

$$\varphi_{H_i}(\, \alpha \,) = (\alpha H_i)^TQ(\alpha H_i) = \alpha^2[\, H_i^T(\, \lambda_i H_i \,) \,] = \lambda_i\alpha^2$$

(note the use of $\| H_i \| = 1$), which is a homogeneous quadratic function whose curvature is the corresponding eigenvalue. This allows to "recycle" the analysis of the simple case of §1.3.2 in the following sense: the eigenvalues H_i take the place of the axes u^i, and the eigenvalues λ_i take the place of the coefficients a_i. This is indeed correct: the separable case corresponds to the diagonal matrix $Q = \mathrm{diag}(\, a_1 \,, \ldots, \, a_n \,)$, and therefore $\lambda_i = a_i$ and $H_i = u^i$. Thus, defining the *span* of any finite set of vectors $V = \{\, V_1 \,, V_2 \,, \ldots, V_k \,\} \subset \mathbb{R}^n$ as

$$\mathit{span}(\, V \,) = \{\, x = \textstyle\sum_{h=1}^k \beta_h V_h \,:\, \beta \in \mathbb{R}^k \,\} \,,$$

and $I^0 = \{\, i \in I \,;\, \lambda_i = 0 \,\}$ the—possibly, empty—set of (indices of) null eigenvalues, one has for (QU)

- if $Q \succ 0$ then $x = 0$ is the unique optimal solution;
- if $Q \succeq 0$ then $x \in \mathit{span}(\, \{\, H_i \,:\, i \in I^0 \,\} \,)$ is the set of all optimal solutions;
- if either $Q \succ\!\prec 0$, $Q \preceq 0$ or $Q \prec 0$ (i.e., there exists at least a negative eigvenalue), then the problem is unbounded.

The result for the maximization version of (QU) is obtained by exchanging $\succ/\succeq$ with $\prec/\preceq$. This is illustrated in Figure 1.8, which shows the level sets of quadratic functions with Q all having the same eigenvectors

$$H = [\, H_1 \,, H_2 \,] = \frac{\sqrt{2}}{2}\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$$

but different eigenvalues. In particular, on the top left $\lambda = [\, \lambda_1 \,, \lambda_2 \,] = [\, 8 \,, 4 \,]$, on the top right $\lambda = [\, 8 \,, 2 \,]$, on the bottom left $\lambda = [\, 8 \,, 0 \,]$, and on the bottom right $\lambda = [\, 8 \,, -2 \,]$. As expected, the axis along $H_2 = [\, 1 \,, 2 \,]$ is longer than that along $H_2 = [\, -1 \,, 1 \,]$ since $\lambda_2 < \lambda_1$. In particular, its length grows as λ_2 shrinks (top, left to right), up until it becomes "infinitely long" (bottom left). However, when $\lambda_2 < 0$ (bottom right) the graph is no longer a paraboloid but an hyperboloid: the tomography along H_1 is convex but that along H_2 is concave (cf. Figure 1.6 bottom right).

Exercise 1.10. Compute the Q matrices corresponding to the four examples. **[solution]**

These results precisely characterise when (QU) has an optimal solution, and what it is. This does not mean that the solution process is necessarily straightforward, since computing the eigenvalues is nontrivial. Furthermore, the "really interesting" form of the quadratic (unconstrained) optimization problem is that with non-homogeneous functions, where the results—discussed in the next section—are somewhat more complex

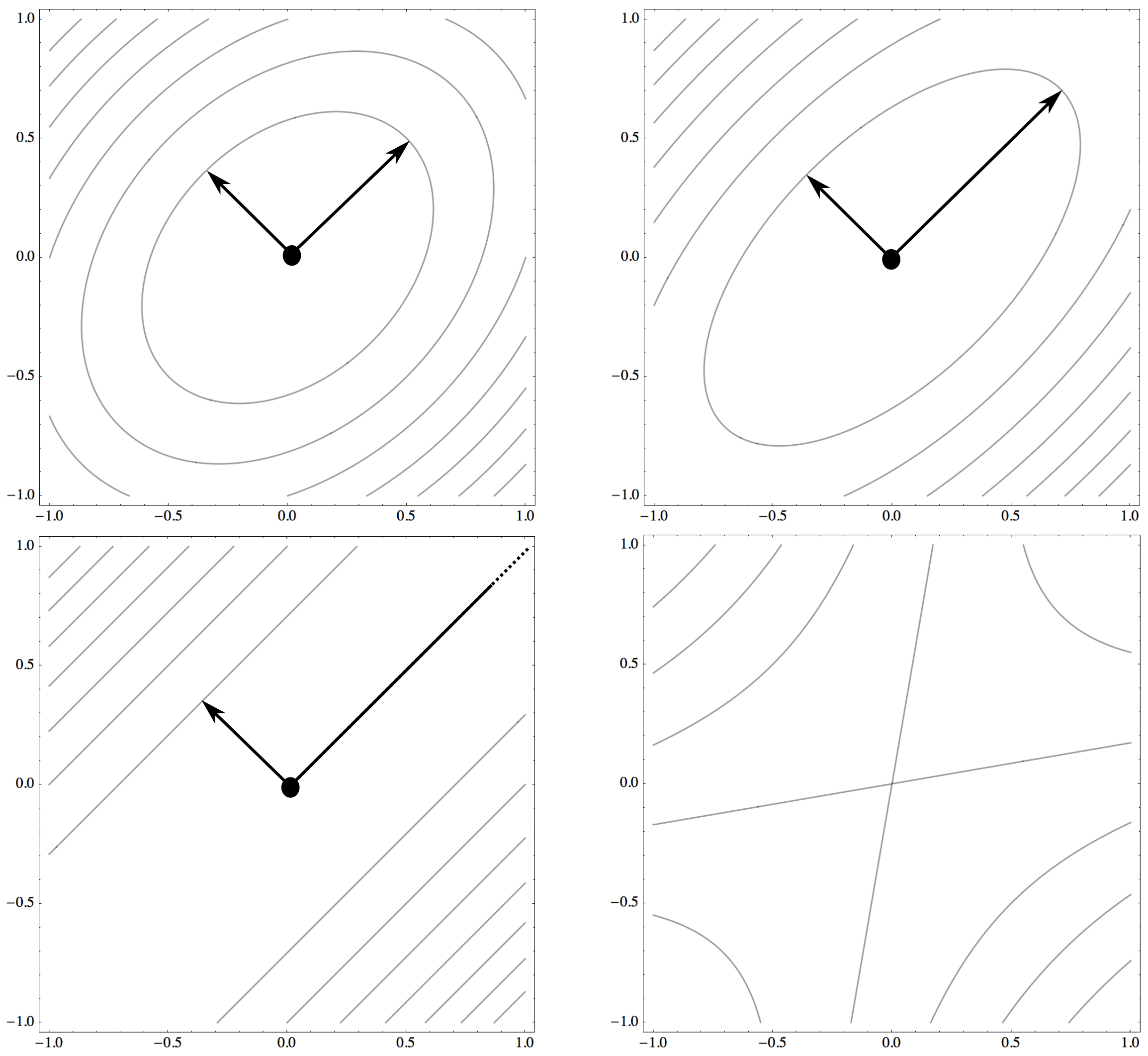


Figure 1.8: Level sets of some homogeneous quadratic functions

(although following the same general lines). In fact, in the following some nontrivial solution procedures will have be developed. However, before doing so it is worth pointing out that the problems are *significantly more challenging in the constrained case*, even for "simple" constraints like the box ones. This may seem surprising, since

$$(QB) \quad \min\{\, f(\,x\,) = x^T Qx \;:\; \underline{x} \leq x \leq \overline{x}\,\}$$

is at least guaranteed to have an optimal solution whenever all bounds are finite. However:

- if $Q \succeq 0$, the problem can be solved "efficiently" but with significantly more complex algorithms, as discussed in Chapter **??**;
- if either $Q \succ\!\!\prec 0$, $Q \preceq 0$ or $Q \prec 0$ (i.e., there exists at least a negative eigvenalue) the problem is *extremely difficult* ($\mathcal{NP}$-hard) [14]: there are no known algorithms to solve it with worst-case complexity that grows less than exponentially fast in n, and it is generally conjectured that there cannot be any.

Again, the resuts hold the maximization version is obtained by exchanging $\succ / \succeq$ with $\prec / \preceq$. This is a striking example showing that minimizing one function f or maximizing it (as opposed to maximizing $-f$) are by no means equivalent problems. Since these lecture notes are about "easy" problems, a detailed discussion about the very significant difference of complexity between the two problems is not appropriate here; anyway, there is still no formal proof that the "difficult problems" are really such ($\mathcal{P} \neq \mathcal{NP}$ is still a conjecture, although generally believed to be true). Yet, a very brief illustration about the nature of the complexity may be useful.

Example 1.3. Concave minimization

Useful assumptions to simplify somewhat the discussion are that $Q \prec 0$, i.e., all eigenvalues are negative, and that $\underline{x}_i = -1$ and $\overline{x}_i = 1$ for all $i \in I$, i.e., X is the *unitary hypercube*. It is then easy to show that, in the optimal solution x^* to (QB), either $x^*_i = -1$ or $x^*_i = 1$ for all $i \in I$ (the solution is on a *vertex* of the unitary hypercube). Indeed, assume that $-1 < x^*_i < 1$

for some i, and consider the tomography $\varphi_{x^*,u^i}(\alpha)$: in plain words, all variables x_j save the i-th are fixed to the value x^*_j, while x_i is free to change provided it is feasible, that is, $-1 \leq x^*_i + \alpha \leq 1$ must hold, i.e., $-1 - x^*_i = \underline{\alpha} \leq \alpha \leq \overline{\alpha} = 1 - x^*_i$, where by the hypotheses $\underline{\alpha} < 0$ and $\overline{\alpha} > 0$. Since

$$\varphi_{x^*,u^i}(\alpha) = (x^*)^T Q x^* + 2(x^*)^T Q u^i \alpha + (u^i)^T Q u^i \alpha^2$$

where $(u^i)^T Q u^i < 0$, φ is a concave quadratic function that is minimised on the interval $[\,\underline{\alpha}\,,\,\overline{\alpha}\,]$, and by the analysis of §1.2.3 its minimum can only lie in one of extremes of the interval, which clearly correspond to the values of α such that either $x_i = -1$ or $x_i = 1$. No point inside the interval—such as $\alpha = 0$, corresponding to $x = x^*_i$–can be optimal, contradicting the fact that x^* was optimal.

The continuous problem (QB) is then equivalent to the combinatorial one

$$(QC) \quad \min\{\, f(x) = x^T Q x \,:\, \{-1\,,\,1\}^n \,\}$$

whose feasible region contains a finite, but *exponential* (2^n) number of solutions. It is therefore possible to solve it by enumerating all these solutions and keeping the best one, but such a process would have at least $O(2^n)$ cost. Unfortunately, all the known algorithms for solving such a problem have an analogous exponential worst-case complexity: in plain words, no algorithm has ever been found that is "substantially more clever" than blind enumeration in the worst case. In practice clever and sophisticated algorithms exist that may be able to solve some instances of such problems efficiently, but none of them with provable non-exponential accuracy in all cases. Indeed, even the best algorithms may actually require extremely long time, compatible with a real exponential complexity, to solve many instances of the problem. This is true for a huge class of problems that all have "the same complexity" [5], leading to the belief that no polynomial-time algorithm exists for any of them. Unless a very surprising result proves otherwise, the only safe assumption is therefore that (QB) cannot be solved efficiently.

The example shows how "brittle" optimization problems are, at least with respect to complexity: an apparently minor change—in this case, just one negative eigenvalue in Q—brings an enormous difference in the ability to solve large-scale instances. This is why these lecture notes will tread very carefully to remain on the "easy" side of optimization, possibly "cheating" whenever appropriate and necessary.

1.3.4 Optimizing non-homogeneous quadratic functions

The "truly interesting" case is that of the non-homogeneous quadratic function

$$f(x) = \tfrac{1}{2} x^T Q x + \langle\, q\,,\, x\,\rangle$$

with $q \neq 0$, i.e., a homogeneous quadratic function plus a linear one. The constant factor of $1\,/\,2$ on the quadratic part may look weird at first, but it is useful to streamline the results somewhat. Taking a leaf from §1.2.2, the result is actually at hand under one assumption: Q *nonsingular*, i.e., $\lambda_i \neq 0$ for all $i = 1, \dots, n$ (regardless of the sign). Indeed, in this case one can again translate the variables space as $z = x - \bar{x}$ for $\bar{x} = -Q^{-1} q$ to obtain

$$f(x) = g(z) = \tfrac{1}{2} z^T Q z + f(\bar{x}) \;,$$

i.e., an homogeneous quadratic function in the z-space (disregarding the constant).

Exercise 1.11. Prove the result, but it should look familiar **[solution]**

This immediately allows to apply the results of the previous section:

- if $Q \succ 0$ then $x = \bar{x}$ is the unique optimal solution;
- if $Q \succeq 0$ then $x \in \bar{x} + \mathit{span}(\,\{\, H_i \,:\, i \in I^0 \,\}$ is the set of all optimal solutions;
- if either $Q \succ\!\!\prec 0$, $Q \preceq 0$ or $Q \prec 0$ (i.e., there exists at least a negative eigvenalue), then the problem is unbounded.

Again, exchanging $\succ/\succeq$ with $\prec/\preceq$ give the result for the maximization version, and the box-constrained cases remains hard, or at least nontrivial.

However, this does not conclude the analysis like in the univariate case. Indeed, there if $a = 0$ then the function was linear. In this case, "different pieces of Q can be 0 without all Q being", which means that a more complicated analysis is needed; however, this is so only in the minimization case where $Q \succeq 0$ and in the maximization one where $Q \preceq 0$, since in the other ones the problem is surely unbounded (below/above). We will then focus on minimization with $Q \succeq 0$, since the other case is symmetric. The set I^0 of (indices of) null eigenvalues is nonempty, i.e., $k = |\, I^0 \,| > 0$; for the set of (indices of) *positive* eigenvalues $I^+ = I \setminus I^0$, with $h = |\, I^+ \,| = n - k$, we can also assume $h > 0$ (since the only matrix with all 0 eigenvalues is the null one, and in this case f is "truly linear"). Clearly, it is crucial do distinguish the directions corresponding to the null eigenvalues (where the tomography is linear) from those corresponding to the non-null ones (where the tomography is convex quadratic). The first is $\ker(\,Q\,) = \mathit{span}(\,\{\, H_i \,:\, i \in I^0 \,\}\,)$: it is easy to see that $Qv = 0$ for all $v \in \ker(\,Q\,)$.

Exercise 1.12. Prove the result. **[solution]**

Thus, this is the set of all vectors "along which the quadratic part il linear": note that obviously $0 \in \ker(Q)$, but the assumption is that the subspace contains other vectors than 0. The other interesting subspace, of course, is $\mathrm{im}(Q) = span(\{ H_i \,:\, i \in I^+ \})$. Its fundamental property is that $\forall\, w \in \mathrm{im}(Q)$ exists some $x \in \mathbb{R}^n$ s.t. $Qx = w$; in plain words, $\mathrm{im}(Q)$ is the set of all right-hand-sides of solvable linear systems with Q.

Exercise 1.13. Prove the result (recall $Q = \lambda_1 H_1 H_1^T + \ldots + \lambda_n H_n H_n^T$, use [68]). **[solution]**

Because H is a (orthonormal) base of $\mathbb{R}^n$ and each H_i either belongs to I^0 or I^+, $\mathbb{R}^n = \mathrm{im}(Q) + ker(Q)$ and $\mathrm{im}(Q) \perp ker(Q)$. In other words, every vector $v \in \mathbb{R}^n$ can be written as $v = v^0 + v^+$, with $v^0 \in ker(Q)$, $v^+ \in \mathrm{im}(Q)$, and $\langle\, v^0 \,,\, v^+ \,\rangle = 0$. In particular, hence, $q = q^0 + q^+$ with $q^0 \in ker(Q)$ and $q^+ \in \mathrm{im}(Q)$. This means that the system

$$Q\bar{x} = -q^+$$

has at least one solution (obviously, $v \in \mathrm{im}(Q) \iff -v \in \mathrm{im}(Q)$). Picking *any* such $\bar{x}$ and defining as usual $z = x - \bar{x}$, it is then easy to see that

$$f(x) = g(z) = \tfrac{1}{2} z^T Q z + q^0 z + f(\bar{x}) \;.$$

Exercise 1.14. Prove the result, but it should look very very familiar. **[solution]**

In plain words, f is "truly quadratic" on $\mathrm{im}(Q)$ but actually linear on $\ker(Q)$, with the linear coefficient being q^0. This was also true in the homogeneous case, but since there $q = 0 \implies q^0 = 0$, the linear function was constant, and any point in $\ker(Q)$ was a minimum. In this case, this may or may not happen. That is:

- if $q^0 \neq 0$, then the problem is unbounded below: indeed, the tomography—in the z-space—along with centre 0 and $q^0 \in \ker(Q)$ is
$$\varphi_{0,d}(\alpha) = \alpha \langle\, q^0 \,,\, q^0 \,\rangle + f(\bar{x}) \;,$$
i.e., a linear (affine) function with slope $\|\, q^0 \,\| > 0$;
- if $q^0 = 0$, then every $x \in \bar{x} + \ker(Q)$ is optimal for the problem.

Exercise 1.15. Prove the result. **[solution]**

Considering that $q^0 = 0 \iff q^+ = q \iff$ there exists some $\bar{x} \in \mathbb{R}^n$ such that $Q\bar{x} = -q$, all the above discussion leads to the following result:

Theorem 1.1. If $Q \succeq 0$, the optimal solutions of the (unconstrained) problem

$$(QU) \quad \min\{\, f(x) = \frac{1}{2} x^T Q x + \langle\, q \,,\, x \,\rangle \;:\; x \in \mathbb{R}^n \,\}$$

are all and only the solutions to the linear system

$$Qx = -q$$

if any (that is, the problem is unbounded below if the system has no solution).

Thus, if $Q \succ 0$ then $\bar{x} = -Q^{-1} q$ is the unique optimal solution to (QU). If, rather, Q is only positive semidefinite, the problem may or may not have optimal solutions. In plain words this has to do with the fact that "along an eigenvector corresponding to a zero eigenvalue the quadratic part of the function is constant", which means that "the linear part now decides the outcome": the problem is bounded below (if and) only if any direction along which the quadratic part of f is constant is also orthogonal to q, which means that also the linear part of f is constant.

Despite being somehow simple, the above result merits some comments:

- it is the first example of (nontrivial) *optimality condition* in these lecture notes, i.e., a condition that allows to precisely identify the optimal solutions to a problem; optimality conditions are crucial for the development of solution algorithms, in that they provide the *stopping criterion*, i.e., a way to be able to terminate the search when the desired solution is found, but even more—as we shall see—because they typically guide the design of the algorithms themselves;
- it underlines the fundamental role of (non)linear algebra in the development of optimization algorithms: the task of solving an optimization problem, or proving it has no solution, is traced back to the task of proving that a certain set of linear (algebraic) conditions are satisfied.

Of course this is just the simplest possible result: more complex problems will require more complex optimality conditions. However, the previous analysis has already revealed—only using the simplest possible mathematical tools—many of the fundamental ideas of all the subsequent developments.

1.4 Multivariate Quadratic optimization: Gradient Method

Having established the optimality conditions of the optimization problem (Q) paves the way to developing approaches to *actually construct* a solution satisfying them (or prove that none exists), which is the central issue in these lecture notes. For the "very simple" current case, in which it is sufficient to solve the linear system $Qx = -q$ (with Q real, symmetric and ideally positive semidefinite), *direct* methods are available that can construct the solution $\bar{x}$. One of these will even be briefly recalled later on. However, this is a very special case that only occurs for quadratic functions and does not readily generalise. For most problems, computing the exact solution with a finite approach is not possible. As previously discussed, this is not really an issue, though. Even directly solving $Qx = -q$ will necessarily incur in numerical errors, which may be significant, so the exact solution is not available anyway (although some exceptions may exist, and numerical errors will typically be "small" w.r.t. those inherent in other approaches). By the same token, any approach that eventually constructs a solution x that "satisfies the system $Qx = -q$ well enough" will be appropriate. This leads to the concept of *iterative* methods, that is central in these lecture notes.

1.4.1 Some generalities on iterative methods

By and large, all iterative methods work in the same way: they start from an *initial guess* x^0 of the solution, upon which little assumptions are usually made (that is, x^0 is not required to be a "good" solution in any sense of the term, $x^0 = 0$ being one usual choice even if 0 may be arbitrarily far from the desired final answer), and then employ *some process* that take the *current iterate* x^i and transforms it in the *next iterate* x^{i+1}. This process is iterated, in principle for an arbitrary number of times, giving rise to *a(n infinite) sequence* $\{ x^i \}$ *of iterates*. The idea is that the sequence should "eventually approach the desired solution x^*", the optimal solution to the optimization problem at hand, in a sense to be formally specified in the following. In practice, the process is finitely interrupted when some iterate x^i is "close enough to x^*", again in a sense to be better specified.

The fundamental aspects of iterative methods that will be primarily investigates in these lecture notes are:

- *correctness*, or *convergence*: proving the that the sequence will eventually reach "close enough to x^*", with "enough" dictated by one (or more) parameter(s) that can be freely set by the "user" of the approach (possibly subject to limits due to the numerical accuracy);
- *efficiency*, or *complexity*: characterising how rapidly the algorithm will achieve its intended result, depending on the chosen accuracy parameter(s), typically measured in terms of *number of iterations* required to reach the *stopping criterion* proving that the current iterate x^i is "close enough" to x^*, and possibly taking into account the cost (complexity) of each single iteration so as to be able to estimate the total computational cost of the whole process.

The above concepts are purposely stated in a vague way, since, as we will see, there will need to be different choices for the "close enough to x^*" definition according to the details of the problem and algorithm at hand. The most obvious version is given by the fact that the sequence of iterates $\{ x^i \}$ naturally defines the *sequence of (objective) function values* ("f-values" for short) $\{ f^i = f(x^i) \}$; one then wants that "$\{ f^i \}$ go towards $f_* = f(x^*)$". The natural mathematical interpretation is that

$$\lim_{i \to \infty} f^i = f_* \quad \text{usually written as} \quad \{ f^i \} \to f_* \; .$$

A sequence of iterates $\{ x^i \}$ such that the corresponding sequence $\{ f^i \}$ of f-values have the above property is called a *minimizing sequence* (for a minimization problem). Since not all sequences have limits, this may not always be possible; cf. §A.5 for alternative notions of convergence that can be used. However, a very natural assumption on the sequence of f-values produced by an algorithm is that it is *decreasing*: $f^{i+1} = f(x^{i+1}) < f^i = f(x^i)$, corresponding to the fact that "x^{i+1} is strictly better than x^i". This is called a *descent*, or *monotone*, algorithm. Although not all algorithms have this feature many do, among which all those of interest in this section; hence for a start one can focus on descent algorithms. Since monotone sequences always have a limit, the asymptotic value $f^\infty = \lim_{i \to \infty} f^i$ is always well-defined. It is immediate to realise that $f^\infty \geq f_*$, which means that $f^\infty = -\infty$ "proves" that the problem is unbounded; but this is largely a theoretical concept, as in practice no algorithms run forever and "one cannot really wait for the sequence to go all the way down to $-\infty$". Indeed, *proving unboundedness* is usually nontrivial (the current case being one of the few positive exceptions to this rule); fortunately, most applications of interest for these lecture notes are naturally bounded, which means that necessarily $f^\infty > -\infty$. The first step when designing a (descent) algorithm is therefore to ensure that $f^\infty = f_*$, i.e., *convergence*. The second step is to try to estimate "how fast" f^i get "close" to f_*, i.e., *efficiency*. A first, elementary algorithm will now be introduced for (Q) providing a first example of convergence and efficiency analysis. Although the algorithm is simple, its analysis is not trivial. Furthermore, the results obtained in this case will be shown to be mirrored in many more complex contexts, providing a useful baseline for the analysis of (unconstrained) optimization algorithms.

Before moving along with an actual algorithm, it is appropriate to remark that $f^\infty = f_*$ is not necessarily a completely satisfactory notion of "convergence". Indeed, the reason for solving problems such as (Q) is typically that of finding an approximately optimal *solution.* While the intuition is that a minimizing sequence will correspond to a sequence of iterates x^i that "get close" to some optimal solution, this may only be partly true. Indeed, consider the minimization of

$$f(\, x_1 \,,\, x_2 \,) = x_1^2 \;,$$

with $f_* = 0$ corresponding to the set of optimal solutions $X^* = \{\, (\, 0 \,,\, x_2 \,) \,:\, x_2 \in \mathbb{R} \,\}$. The sequence of iterates given by

$$x^i = [\, 1 \,/\, i \,,\, (-1)^i i \,]$$

clearly gives $\{\, f^i = 1 \,/\, i^2 \,\} \to 0 = f_*$; that is, $\{\, x^i \,\}$ is a minimizing sequence. However, the iterates themselves can hardly be seen to "converge" somewhere: $\{\, \|\, x^i \,\| \,\} \to +\infty$, and every two subsequent iterates are "very far apart". In some sense, of course, the sequence is correctly identifying the optimal set: $\{\, x_i^1 \,\} \to 0$, while the x_i^2 values "span all of $\mathbb{R}$". However, at the very least an algorithm producing such a minimising sequence should be considered as "unstable", in the sense that it could easily produce very different (approximate) solutions. This is not to say that practical algorithms would naturally produce such iterates; rather, the intent here is to mention how *convergence of f-values* and *convergence of iterates* (to be defined in details later) are in principle different. Of course, the issue here is that X^* is not a singleton, which may in principle lead an algorithm to be "attracted to different optimal solutions": if $X^* = \{\, x^* \,\}$ is a singleton, convergence of f-values is equivalent to convergence of iterates (again, to be discussed in details later). In the quadratic case, this—and the analysis of §1.3.4—suggests that having $Q \succ 0$ may be "more convenient" than having $Q \succeq 0$. This will indeed be shown to be true, justifying why initially $Q \succ 0$—implying that (Q) always has the unique solution $x^* = -Q^{-1}q$—will be assumed.

1.4.2 The Gradient Method for Quadratic Functions

The basic idea of the approach is quite simple: given one iterate x^i, it is reasonably necessary to compute $g^i = Qx^i + q$, called (for reasons that is preferable not to discuss right now) the *gradient of f at x^i*. Clearly, if $g^i = 0$ then one optimal solution x^* has been obtained and the algorithm can stop. As previously discussed, a check such as "$g^i = 0$" is not doable in floating point arithmetic, and it is typically substituted with "g^i is close enough to be the all-0 vector", i.e., "$\|\, g^i \,\| \leq \varepsilon$ for some appropriate $\varepsilon > 0$". The choice of ε is itself not trivial and it will be discussed in some detail later on; for now it is just a user-defined value upon which the algorithm has no control (it is one, and in fact the only, *algorithmic parameter* of the approach). The idea is that whenever $\|\, g^i \,\| > \varepsilon$ ("bad news", x^i is not "close to being optimal") then it is possible to *use g^i to produce a x^{i+1} "better" than x^i*, in the sense already discussed below. Indeed, consider the (scaled) tomography

$$\varphi_{x^i,-g^i}(\, \alpha \,) = f(\, x^i - \alpha g^i \,) - f(\, x^i \,)$$

of center x^i and direction $-g^i$ (called the *antigradient*). Simple algebra shows that

$$\varphi_{x^i,-g^i}(\, \alpha \,) = \tfrac{1}{2}\alpha^2 (g^i)^T Q g^i - \alpha[\, (g^i)^T Q x^i + q g^i \,] = \tfrac{1}{2}\alpha^2 (g^i)^T Q g^i - \alpha \|\, g^i \,\|^2 \;.$$

That is, the tomography is (not surprisingly) a non-homogeneous quadratic function with curvature $a = (g^i)^T Q g^i > 0$ and slope $b = -\|\, g^i \,\|^2 \;[< -\varepsilon^2\,] < 0$. The analysis of §1.2.3 now immediately proves that some $\bar{\alpha} > 0$ exists such that $\varphi_{x^i,-g^i}(\, \bar{\alpha} \,) < 0$, i.e., $f(\, x^i - \bar{\alpha} g^i \,) < f(\, x^i \,)$: this immediately provides a clue about how to construct a next iterate x^{i+1} of a descent algorithm.

Exercise 1.16. Check all the above. **[solution]**

In plain words, the same information (g^i) indicating that the algorithm *cannot* stop at the same time provide the means to construct a "better" next iterate x^{i+1}. To fully turn this intuition into an algorithm it is only necessary to specify how exactly the appropriate step $\bar{\alpha}$ is selected. Although there may be different choices, the most natural one is to pick *the best possible iterate x^{i+1} that can be obtained*, i.e., the value

$$\alpha^i = \operatorname{argmin}\{\, \varphi_{x^i,-g^i}(\, \alpha \,) \,:\, \alpha \in \mathbb{R} \,\} \;.$$

Again, the results in §1.2.3 ensure that such a value exists, is unique, and

$$\alpha^i = \|\, g^i \,\|^2 \,/\, (g^i)^T Q g^i \;. \tag{1.12}$$

Using (1.11) it is easy to provide finite bounds on the *stepsize*: $1/\lambda_1 \leq \alpha^i \leq 1 \,/\, \lambda_n$.

Exercise 1.17. Prove the last statement. **[solution]**

All this finally gives the *Gradient Method* for the unconstrained optimization of quadratic functions:

Algorithm 1.1. Gradient Method for Quadratic Function

```
procedure $x = SDQ\ (Q, q, x, \varepsilon)$
  for( ; ; )
    $g \leftarrow Qx + q$;
    if( $\| g \| \leq \varepsilon$ ) then break;
    $\alpha \leftarrow \| g \|^2 / g^T Qg$; $x \leftarrow x - \alpha g$;
```

Although having been devised for the case where $Q \succ 0$, the algorithm can be easily adapted to the case where $Q \succeq 0$, or even $Q \succ\!\!\prec 0$ (basically, no hypotheses on Q). Starting from the former case, the issue clearly is that $g^T Qg = 0$ can happen; numerically speaking, this means that it may be a "very small" number creating issues (e.g., overflows) while computing α. This is, however, easy to solve: $g^T Qg = 0$ means that $g \in \ker(Q)$, i.e., is a direction along which $\varphi_{x,-g}$ is (almost) linear. However, $b = -\| g^i \|^2$ is not 0, and "sizeably so": thus, $\varphi_{x,-g}$ is *not* constant, and therefore unbounded below. Thus, $f_* = -\infty$ for (Q). Adapting the algorithm then just requires adding the instruction

if($g^T Qg \leq \delta$) **then break**;

for a "very small" value δ. In this case, however, it is also necessary to signal the different kind of outcome: that is, the algorithm should return information allowing the user to distinguish whether the returned x is indeed an (approximately) optimal solution, or the problem has been found to be unbounded below. This is usually done by returning a "status code" (often, a simple integer value) encoding this information. It would also be appropriate to return $x = -g$, as this is the *certificate of unboundedness* of the problem: a direction along which the tomography (hence, f) is unbounded below; depending on the status code the user would know in which of the two ways the returned vector should be interpreted. Interestingly, this is all that is needed to extend the algorithm to the case of $Q \succ\!\!\prec 0$. Indeed, in that case the condition above can be verified when $g^T Qg < 0$ ($g^T Qg \leq -\delta < 0$, i.e., "negative and not very close to zero"), which still means that $\varphi_{x,-g}$ is unbounded below; except, in this case it's not a linear function, but a quadratic one with negative curvature. With possibly a different returned status code, the algorithm should stop all the same. It should be remarked, however, that in none of the cases the algorithm is guaranteed to be able to determine the certificate; rather, a sequence of iterates with decreasing f-values may be produced where f^i may "numerically approach $-\infty$" (possibly eventually yielding overflow errors). Again, *proving* unboundedness is nontrivial. However, all this means that adapting the code for maximization is trivial.

Exercise 1.18. Discuss how to change the code to solve $\max\{ f(x) \}$ instead. [**solution**]

In terms of complexity of the each iteration, all operations are linear in n, i.e., $O(n)$, except the matrix-vector products in $g \leftarrow Qx + q$ and the computation of $a = g^T Qg$. Thus, the complexity of each iteration is $O(n^2)$. Although the algorithm does not, strictly speaking, need to compute the f-value $f^i = f(x^i)$, this is also $O(n^2)$ due to the matrix-vector product $x^T Qx$. Thus, the costly part of the algorithm is the computation of two (or three) matrix-vector products involving Q. With some creativity these can be reduced to *only one*; this is called a *streamlining* of the algorithm.

Exercise 1.19. Discuss how to streamline the algorithm. [**solution**]

1.4.3 Convergence of the gradient method for $Q \succ 0$

In order to prove convergence one needs to show that $f^\infty = \lim_{i \to \infty} f^i = f_*$. This is usually rewritten in terms of the absolute gap $a^i = f^i - f_* \geq 0$ as $\{ a^i \} \to 0$. Since the algorithm is of descent, a^i is monotone decreasing and therefore $\{ a^i \} \to a^\infty \geq 0$; it is however not trivial to prove that $a^\infty = 0$. Usually, an issue is that f_* is not known, and hard to characterise a-priori, which makes it difficult to estimate a^i. In this simple case, however, the *homogeneous form of the error*

$$A(x) = f(x) - f_* = \tfrac{1}{2}(x - x^*)^T Q(x - x^*) \tag{1.13}$$

is valid for all $x \in \mathbb{R}^n$ and allows to considerably simplify the arguments.

Exercise 1.20. Prove the formula above. [**solution**]

The above relationship, the definition of $x^{i+1} = x^+ - \alpha^i g^i$ with the quantities defined in the algorithm, and $Q \succ 0$ allow to precisely characterise a^{i+1} in terms of a^i:

$$A(x^{i+1}) = \left(1 - \frac{\| g^i \|^4}{((g^i)^T Qg^i)((g^i)^T Q^{-1} g^i)} \right) A(x^i) \ . \tag{1.14}$$

Proving (1.14) requires some simple, albeit somewhat tedious, algebra [7, Lm. 8.6.1]. One of the steps is worth pointing out for later reference: the gradients at two subsequent iterations are orthogonal, i.e., $\langle\, g^{i+1} \,,\, g^i \,\rangle = 0$.

Exercise 1.21. Prove the last statement. **[solution]**

Exercise 1.22. Prove (1.14). **[solution]**

Although exact, (1.14) is not completely informative because it depends on g^i, which in turn depends on x^i. It would therefore be useful to bound from below the crucial term of the form

$$s(\, x \,) = \|\, x \,\|^4 \,/\, [\, (x^T Q x)(x^T Q^{-1} x) \,]$$

for all possible $x \in \mathbb{R}^n$ (except 0, since we know $g^i \neq 0$). Since $Q \succ 0$, this is indeed possible using the *condition number of* Q $\kappa = \lambda_1 \,/\, \lambda_n \geq 1$, i.e., the ratio between its largest and smallest eigenvalue. Simple arguments using the variational characterisation (1.11) and the fact thateigenvalues of Q^{-1} are the inverse of those of Q (cf. §A.4) yield

$$s(\, x \,) \geq s = \lambda_n \,/\, \lambda_1 = 1 \,/\, \kappa \quad \forall\, x \in \mathbb{R}^n \;, \tag{1.15}$$

which is independent of the iterations and only depends on inherent properties of Q.

Exercise 1.23. Prove (1.15). **[solution]**

Plugging (1.15) in (1.14) gives

$$A(\, x^{i+1} \,) \leq r A(\, x^i \,) \quad \text{with} \quad r = (1 - 1 \,/\, \kappa) = (\kappa - 1) \,/\, \kappa = (\lambda_1 - \lambda_n), /\, \lambda_1 < 1 \;. \tag{1.16}$$

This immediately confirms that the error strictly decreases, since $a^{i+1} = r a^i < a^i$. In fact, the error decreases "exponentially fast": given the initial error $a^0 = A(\, x^0 \,)$ depending on the initial iterate x^0—upon which there is little control, but that it is at least finite—one has

$$A(\, x^i \,) \leq r^i A(\, x^0 \,) \;, \tag{1.17}$$

where $r^i \to 0$ as $i \to \infty$.

Exercise 1.24. Prove (1.17). **[solution]**

As a consequence, the algorithm converges: $a^i \to 0$ as $i \to \infty$, i.e., $\lim_{i\to\infty} f^i = f_*$. Due to the specific formula above, it is also possible to estimate its complexity, i.e., the (maximum) number of iterations that are required to achieve a *prescribed accuracy* δ. That is, one wants to compute a function $c(\,\cdot\,)$ such that the algorithm is guaranteed (if ran in exact arithmetic) to produce within $c(\, \delta \,)$ iterations an iterate x^i such that $A(\, x^i \,) \leq \delta$. One just has to plug the requirement in (1.17) to get

$$i \geq [\, 1 /\, \log(\, 1 \,/\, r \,) \,] \log(\, a^0 \,/\, \delta \,) \;. \tag{1.18}$$

Exercise 1.25. Prove (1.18). **[solution]**

Thus, the complexity of the algorithm is $O(\, \log(\, 1 \,/\, \delta \,) \,)$, which is on the outset a quite good result. In plain words, assume that k iterations are needed to get an accuracy of, say, $\delta =$ `1e-4`; this means that

$$k \geq R \log(\, \texttt{1e} + 4a^0 \,) = R[\, \log(\, \texttt{1e} + 4 \,) + \log(\, a^0 \,) \,] = R[4 + \log(\, a^0 \,)]$$

for some fixed $R > 0$. Assume now one wants a solution with $\delta =$ `1e-8`; if $f_* \approx 1$, this is basically asking 8 decimal digits of accuracy rather than 4, i.e., a significantly better accuracy. By the same token as above one now have

$$k \geq R[\, 8 + \log(\, a^0 \,) \,] \;;$$

even if the term $\log(\, a^0 \,)$ is insignificant, this means that "doubling the number of accurate decimal digits requires at most doubling the number of iterations". Since one can have at most 16 decimal digits (and usually way less), this means that the algorithm should terminate "quickly" even if highly accurate solutions are required. It is particularly relevant that k in the above formulæ *does not (explicitly) depend on n*. This *dimension independent* complexity estimate can be taken to mean that highly accurate solutions may be obtained efficiently even for very-large-scale problems, since the number of iterations may not grow as n does.

The above analysis is, however, disregarding the role of the two constants a^0 and $R = 1 /\, \log(\, 1 \,/\, r \,)$. For the former, little can be said since in general there is hardly any control on how accurate the initial estimate x^0 is. The general gist is that *starting with a more accurate initial estimate will lead to requiring less iterations*, which, albeit a nice property, is hardly surprising. After all, if x^0 were approximately optimal already, in the sense that $\|\, g^0 = Qx^0 + q \,\| \leq \varepsilon$, then the algorithm would stop immediately (it would be right to wonder how this is related to the accuracy δ, but discussing this is left for later). The impact of R is, however, much more significant. The issue here is that, as the problem becomes *ill-conditioned*, i.e, $\lambda_1 \gg \lambda_n$ *LRA* $r = (1 - 1 \,/\, \kappa) \approx 1$, R grows large. In fact, it is easy to see that

$$r \approx 1 \implies k \approx [\, r \,/\, (\, 1 - r \,) \,] \log(\, a^0 \,/\, \delta \,) \approx \kappa \log(\, a^0 \,/\, \delta \,) \;.$$

Exercise 1.26. Verify the formula above. **[solution]**

In plain words, all other things—δ and a^0—being equal, the number of iterations can be expected to grow proportionally with the condition number of Q. This is in some sense the "bad news" version of a dimension independent efficiency result: the number of iterations required to solve the problem can grow rather large even when n remains the same. In fact, things may be even worse in case there is a *hidden dependency*: λ_1 and λ_n may depend on n, which means that κ may grow as n does.

Unfortunately, this analysis is far from only pointing to a remote possibility: the actual computational behaviour of the approach does trail the result quite closely in practice. This is shown in Figure 1.9, which illustrates the behaviour of the gradient method on small-scale ($n = 2$) instances. In particular, the top row corresponds to the matrix

$$Q = \begin{bmatrix} 11 & 9 \\ 9 & 11 \end{bmatrix} \quad \text{with} \quad \lambda_1 = 20, \lambda_2 = 2 \quad \text{and therefore} \quad \kappa = 10,$$

while the bottom row corresponds to the matrix

$$Q = \begin{bmatrix} 101 & 99 \\ 99 & 101 \end{bmatrix} \quad \text{with} \quad \lambda_1 = 200, \lambda_2 = 2 \quad \text{and therefore} \quad \kappa = 100.$$

On the left the iterates performed by the algorithm are plotted upon some level sets of the objective, while on the right the *convergence plot* shows the evolution of the relative error $a^i \,/\, |\, f_* \,|$ along the iterations on a log-linear scale.

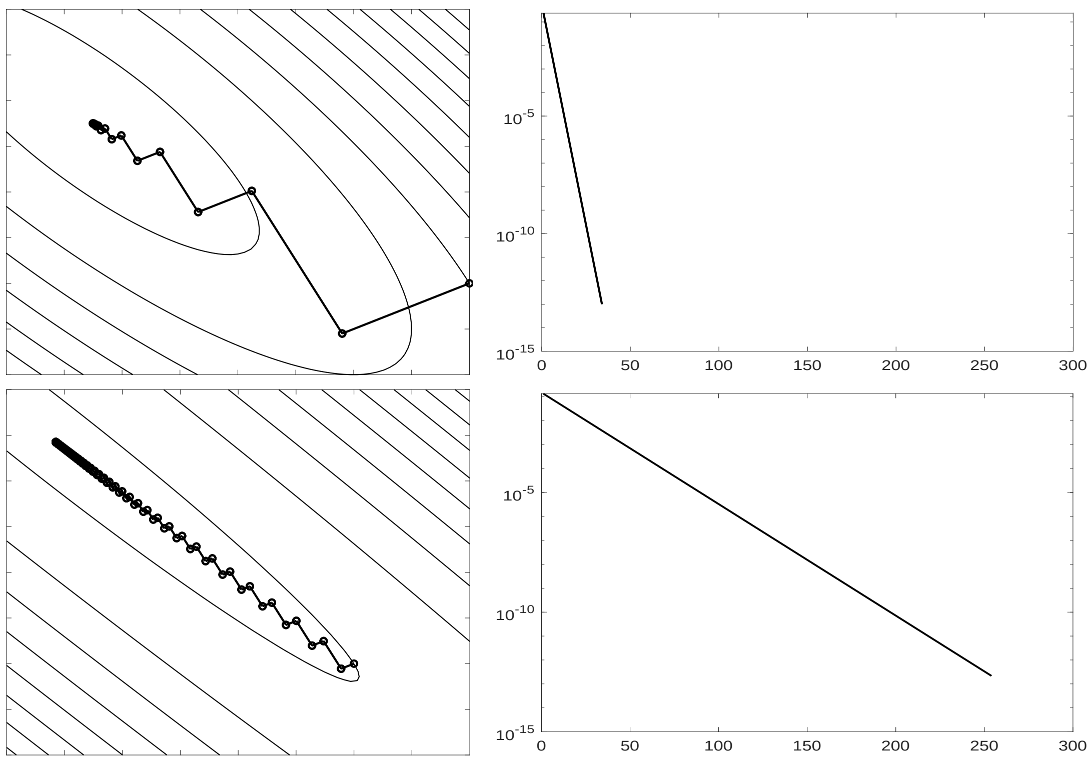


Figure 1.9: Convergence of the gradient method in two examples

In the first instance the algorithm performs 34 iterations to reach $\|\, g^i \,\| \leq$ `1e-6`, which corresponds to $\delta \approx$ `1e-13`, starting from $a^0 \approx$ `2.4e-1`. In the second, starting from $a^0 \approx$ `1.4e-1`, it rather requires 254 iterations to get a solution of comparable quality. Although the number of iterations do not scale exactly as the condition number, it does significantly increase as κ grows. Geometrically, a larger condition number corresponds to the fact that the level sets are "more elongated" (Figure 1.9(left), top to bottom) and the iterates of the algorithm "zig-zag a lot", leading to slow convergence. This is related to the fact that, as previously mentioned, "the trajectory always turns right angles": the gradients at two subsequent iterations are orthogonal, and therefore so are the lines along which the tomography is made. This is confirmed by Figure 1.9(right), which clearly illustrates why (1.17) is known as *linear convergence*: while in both cases the gap decreases linearly in a log-linear plot, which means exponentially, the slope of the line is much steeper (convergence is much faster) when κ is smaller, i.e., r is farther away from 1.

The fact that the scaling in the previous example is not exact should not come as a surprise: (1.15) is a worst-case estimate, and the decrease may actually be faster. In fact, the estimate is not even the tightest possible one: a more refined analysis using the Kantorovich inequality [7, 8.6.(34)] gives better estimate

$$\frac{\| x \||^4}{(x^T Q x)(x^T Q^{-1} x)} \geq \frac{4\lambda_1\lambda_n}{(\lambda_1 + \lambda_n)^2} \implies r \leq \left(\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}\right)^2 = \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 \tag{1.19}$$

This is close but significantly different from the previous one. For instance, $\kappa = 10$ gives $r = 0.9$ with the rougher estimate and $r = (9/11)^2 \approx 0.67$ with (1.19), while $\kappa = 100$ gives $r = 0.99$ against $r = (99/101)^2 \approx 0.96$. Although the difference may seem minor, exponentials are very sensitive to changes of the basis. For instance, $0.99^{100} \approx 0.37$, whereas $0.96^{100} \approx 0.017$. In plain words, according to the rougher estimate 100 iterations of the method would only reduce the (absolute) gap to 37% of its initial value. However, the better estimate reveals that in fact the remaining gap will only be at most 1.7% of its initial value.

There are of course other factors at play in the actual behaviour of the algorithm, mainly the choice of the starting point x^0 and its relationship with q (which has not been mentioned before, but it clearly plays a role). For instance, the results in Figure 1.9 have been obtained with $x^0 = 0$ and $q = [\,2\,,\,2\,]$. The apparently minor change $q = [\,1\,,\,1\,]$ leads to the somewhat dramatic impact whereby the algorithm terminates at *just the second iteration* with $\| g^1 \| \approx$ `3e-16` and δ numerically 0. This is due to the fact that, by sheer chance, the optimal solution $x^* = [\,-0.005\,,\,-0.005\,]$ happen to be exactly on the line $x^0 - \alpha g^0 = -\alpha q$: thus, the minimization along this line immediately achieves $x^1 = x^*$ and the algorithm promptly terminates, even if κ is "large". Thus, in this lucky case the algorithm "does not zig-zag" and it converges (very) efficiently.

In fact, for future reference it is worth mentioning a very special case, suggested by both (1.16) and (1.19), in which the algorithm will always terminate in one iteration: that of $\kappa = 1$, i.e., $\lambda_1 = \lambda_n$, i.e., all eigenvalues being equal. This is typically the case where Q is a scalar multiple of the identity, so let us examine $Q = I$ for simplicity. Clearly, $x^* = -Q^{-1}q = -q$. Also, $g = Qx + q = x + q$. Hence, whatever the initial guess x^0, one has that for $i = 1$ the algorithm is minimising the tomography over the line

$$x^0 - \alpha g^0 = x^0 - \alpha(x^0 + q)\ .$$

Thus, $\alpha^0 = 1$ immediately provides $x^1 = x^*$ and the algorithm terminates, irrespectively of what x^0 was. This is of course not a surprising feat: due to the special properties of Q, x^* could be computed directly with $O(\,n\,)$ effort. In fact, the corresponding quadratic function is separable, and the solution could have been obtained working on each x_i individually as done in §1.3.2. Yet, it is positive that the algorithm "automatically recognises an easy case and is efficient there". Furthermore, this simple observation will turn out to be surprisingly useful later on.

1.4.4 Some notes on the stopping criterion

Before moving on, it can be useful to remark about the stopping criterion of the algorithm. The examples above have already illustrated one significant issue, which is that the "natural" stopping criterion would be $A(\,x^i\,) \leq \varepsilon$, or—even better—$R(\,x^i\,) \leq \varepsilon$. However, such a condition cannot typically be implemented in practice for the simple reason that f_* is typically unknown.

Indeed, as already mentioned, it could be argued that "the real difficulty in optimization comes from estimating the optimal value". In this case this would boil down to computing *lower bounds* $\underline{f}^i \leq f_*$, so as to be able to estimate the (absolute) gap by $a^i = f^i - f_* \leq f^i - \underline{f}^i$. For this to be useful, the lower bounds should be "tight" *at least towards termination*; that is, one would want that $\{\,\underline{f}^i\,\} \to f_*$ (or some weaker convergence notion, cf. §A.5) as well. In plain words, one would have *two sequences*, of upper and lower bounds $\underline{f}^i \leq f_* \leq f^i$, which should be made to simultaneously converge to the optimal value. This is often "hard" but in fact possible in some cases, some of which will be discussed in these lecture notes, but *not* (easily) in the current one.

For the current algorithm (and, as we shall see, unconstrained optimization in general), the stopping criterion is rather that $\| g^i \|$ is "small". This is a "proxy" of the required one, in the sense that $\| g^i \| \leq \varepsilon$ for some "small ε" hopefully implies that $A(\,x^i\,) \leq \delta$ for some "small δ": however, the exact relationship between ε and δ is nontrivial. This is significant in that it is ε that has to be chosen by the user of the algorithm, and the cost of the algorithm grows as ε gets smaller: hence, it would be reasonable to use the largest ε that attains the desired gap $A(\,x^i\,)$.

This is nontrivial to do even in the current particularly simple case, where (1.13) gives

$$a^i = \tfrac{1}{2}(x^i - x^*)^T Q (x^i - x^*) = \tfrac{1}{2}\langle\, x^i - x^*\,,\, g^i\,\rangle \leq \tfrac{1}{2}\| g^i \|\| x^i - x^* \| \tag{1.20}$$

(using $Q(x^i - x^*) = Qx^i + q = g^i$). Thus, if one knew a bound $\Delta \geq \| x^i - x^* \|$, then $\| g^i \| \leq 2\delta \,/\, \Delta$ would imply $a^i \leq \delta$; in other words, taking $\varepsilon = 2\delta \,/\, \Delta$ in the stopping criterion would give the desired final gap. However, in general such a bound is not available. Alternatively, one can use the fact that if λ is an eigenvalue of Q then λ^2 is an eigenvalue of $Q^2 = QQ$ and (1.11) to write

$$\| g^i \|^2 = (x^i - x^*)^T Q^T Q(x^i - x^*) \geq \lambda_n^2 \| x^i - x^* \|^2 \implies \| x^i - x^* \| \leq \| g^i \| / \lambda_n \ ;$$

plugging this in (1.20) gives $a^i = \frac{1}{2} \| g^i \|^2 / \lambda_n$, and therefore using $\varepsilon = \sqrt{2\lambda_n \delta}$ in the stopping criterion would imply ensuring that $a^i \leq \delta$ at termination. However, λ_n is typically unknown, and in fact $\lambda_n = 0$ may happen, rendering the rule numerically useless. All this shows that achieving an exact control over final gap by means of choosing the ε in the stopping criterion is not obvious.

A similar, albeit different, note concerns the relationship among the gap at the current iterate and the distance from the optimum. That is, one expects that $a^i \leq \delta$ (assuming it is possible to guarantee it) implies that $\| x^i - x^* \| \leq \gamma$ for some $\gamma > 0$, but the relationships between the two thresholds is again nontrivial. This may be important in view that the overall solution process is ultimately aimed at constructing a "good approximation" of x^*, and this may have to be defined in terms of $\| x^i - x^* \|$ rather than of the gap. Exploiting the same relationships already used above readily shows that $a^i \leq \delta$ implies $\| x^i - x^* \| \leq \sqrt{\delta / \lambda_n}$, again a difficult relationship to use in practice since λ_n is usually unknown and can be 0. This again confirms that controlling the (different aspects of the) accuracy of the obtained final iterate x^i by means of properly choosing the stopping tolerance ε is nontrivial.

Exercise 1.27. Characterise the other direction, i.e., upper bound a^i in terms of $\| x^i - x^* \|$. **[solution]**

1.4.5 Convergence of the gradient method for $Q \succeq 0$

The previous discussions have repeatedly stressed the importance of the assumption $\lambda_n > 0$, i.e., $Q \succ 0$. This does not, however, mean that the algorithm does not converge in the positive semidefinite case: it does, but the previous analysis no longer applies. In fact, a significantly different analysis is required, which will not be detailed here since it actually is a special case of more general results [3, Theorem 3.3] that will be illustrated in some details later on. It is however worth anticipating the gist of the results, that is the following: for the version of the method with *fixed stepsize* $\alpha^i = 1 / \lambda_1$ for all i, it holds

$$A(x^i) \leq 2\lambda_1 \| x^0 - x^* \|^2 / (i + 2) \ . \tag{1.21}$$

As previously remarked, $1 / \lambda_1$ is the *lower bound* on the *optimal stepsize* (1.12). Hence, using the optimal stepsize leads to better (not worse) improvements at each iteration, indicating that the standard gradient method should converge faster than the fixed-stepsize version; this in fact typically happens in practice. Thus, the method is still convergent: however, its *efficiency* is (at least in theory) dramatically reduced.

Indeed, the number of iterations required to achieve accuracy δ is now (approximately)

$$i \geq \lambda_1 \| x^0 - x^* \|^2 / \delta \ ,$$

i.e., grows *linearly* in $1 / \delta$ instead as *logarithmically* as in the previous case. In plain words, if k iterations are required to achieve, say, δ = `1e-4`, it should be expected that $10000k$ iterations will be required to achieve δ = `1e-8`: each additional digit of accuracy requires *one order of magnitude* more iterations. Methods with this kind of efficiency can hardly be considered to obtain accurate solutions with, say, better than `1e-3` / `1e-4` (relative) accuracy, although as always the actual performances are highly dependent on the specific problem.

Although worst-case performance analysis should always be taken with an abundant pinch of salt, in that it does not necessarily represent real-world performances, indications that "some problems may be harder to solve than others" should never be overlooked. In this case, the presence of null eigenvalues can indeed have a detrimental effect on the performances of the gradient method. This is illustrated in Figure 1.10 with reference to two specifically constructed instances with $n = 1000$.

The matrix Q of the first instance is randomly generated so as to be positive semidefinite, with around 10% of null eigenvalues. Then, q is constructed in such a way as to ensure that the problem is bounded below (by randomly generating a $\bar{x} \in \mathbb{R}^n$ and selecting $q = -Q\bar{x}$). The second instance is obtained by the first by adding that Q a small multiple of the identity, i.e., $Q' = Q + 10I$ (note that the average of the original diagonal elements Q_{ii} is ≈ 300, hence 10 is reasonably small in comparison). Figure 1.10 shows the two convergence plots superimposed, with an iteration limit of `1e+5` (that is, each tick on the axis represents `1e+4` iterations). Within this time limit, on the original instance (with $Q \succeq 0$) the gradient method reaches a relative gap of $\approx$ `1e-7`. The convergence curve shows a markedly *sublinear* convergence: the slope of the curve is not constant, but it gradually diminishes. In other words, the convergence speed slows down as the algorithm proceeds. This is what (1.21) suggests: at each iteration the gap is reduced, but the *ratio* $\rho^i = a^{i+1} / a^i$ is *not* constant, as in the linear case: indeed,

$$a^{i+1} / a^i \approx (i + 2) / (i + 3) \ .$$

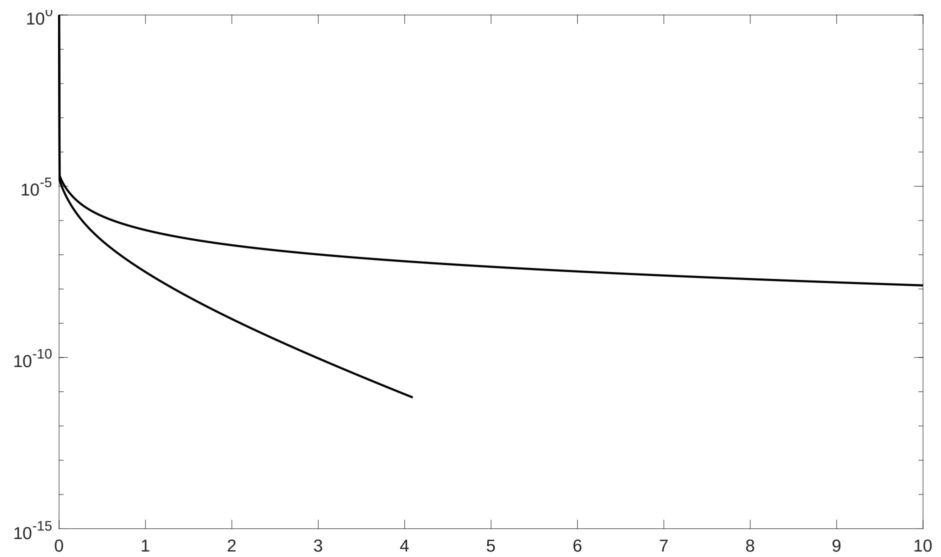


Figure 1.10: Convergence of the gradient method in two larger examples

While $\rho^i < 1$, indicating convergence, $\lim_{i \rightarrow \infty} \rho^i = 1$: the fraction of the gap that the algorithm manages to reduce diminishes at each iteration. The situation is markedly different for the instance having $Q' \succ 0$: in around `4e+4` iteration the algorithm achieves a relative gap of $\approx$ `1e-12`, which would have required an astronomical number of iterations to be achieved with the original semidefinite Q. The corresponding convergence plot does show an initial reduction of the slope, but after $\approx$ `2e+4` iterations the convergence speed stabilises and a clear linear convergence rate ensues. This rather dramatic difference in behaviour is due to an apparently minor change in the data of the problem.

We will not delve further in these issues at this point, since this discussion is only meant to foreshadow future arguments that will be made more precise later on. However, it allows to highlight some important facts that will be found true over and over again in these lecture notes:

- achieving accurate solutions of optimization problems may be prohibitively costly;
- the difficulty of solving a problem depends on the characteristics of its data, with apparently minor changes possibly have dramatic effects.

An important remark is appropriate at this point: the above results have been obtained with one specific algorithm, and a not-really-sophisticated one at that. That a given algorithm may struggle at efficiently solving a problem does not necessarily indicate that the problem is inherently "difficult"; rather, the algorithm may not be "smart enough". The next section will illustrate this point by presenting a "cleverer" version of the gradient method where apparently minor changes (in the algorithm, this time) translate into very substantial efficiency gains. This is *not* meant to imply that the difficulties can always be circumvented: complexity results can be proven that bound the efficiency with which certain classes of problems can be solved in general. However, the next section does illustrate the point that one should not be discouraged too soon, as there is usually ample scope to improve the performances of approaches for specific problems.

1.5 The Conjugate Gradient Method

As the previous section has shown, ideas are needed to improve the gradient method. The intuition under the proposal of this section is that the gradient method is inefficient because it only "optimise over thin slices of the space each time". At iteration $i = 0$, the next iterate x^1 is chosen as the best point along the line $\{ x = x^0 - \alpha g^0 \, : \, \alpha \in \mathbb{R} \}$, i.e., $x^0 + span(\{ g^0 \})$, which is clearly a "very low dimensional (affine) subspace of $\mathbb{R}^n$" (even when $n = 2$, just a line in the plane). Then, at the subsequent iterate x^2 is chosen as the best point in $x^1 + span(\{ g^1 \})$. As it was already mentioned, g^0 and g^1 are orthogonal, hence linearly independent: thus, $span(\{ g^0 , g^1 \})$ would be a 2-dimensional subspace of $\mathbb{R}^n$, as opposed to $span(\{ g^0 \}) \cup span(\{ g^1 \})$, which remains overall a 1-dimensional space. Thus, if one were able to obtain x^2 which optimises f over all $span(\{ g^0 , g^1 \})$, then arguably a "much better progress" would have been achieved; in particular, with $n = 2$ this would mean that the optimal solution would have been achieved. Of course this is not true for the gradient method, as the previous numerical examples show. Hence, if one wants to obtain such a property, the *search direction* d^i at each iteration, so that the next point is selected inside $x^i + span(\{ d^i \})$, will have to be different from $(-)g^i$. One could of course question whether such directions actually exist at all, but the next section will show that they indeed do.

1.5.1 Q-conjugate directions

In order to introduce the desired directions we take an apparent detour. This is not strictly speaking required, but it does introduce some powerful ideas that will be repeatedly useful later on. The observation is that directions "with the right property" do surely exist in a very special case: that of a diagonal $Q \succeq 0$, i.e., $Q = diag(d)$ for some $d \in \mathbb{R}^n_+$. This is a *separable* quadratic function, and the solution procedure sketched in §1.3.2 can be re-interpreted as the following algorithm:

Algorithm 1.2. Reinterpreted Separable Quadratic Optimization

```
procedure $x = SDQ\ (d\,,\,q\,,\,x)$
  for( $i = 1$ ; $i < n$ ; $++i$ )
    $x \leftarrow \text{argmin}\ \{\, f(\,x + \alpha u^i\,) \ :\ \alpha \in \mathbb{R}\,\}$;
```

That is, at each iteration i the algorithm focuses on the i-th variable, which means it uses the i-th vector u^i (with all 0s except an 1 in the i-th entry) of the canonical base, and finds the optimal value for x_i. After that iteration, x_i will never be changed again. It is plain to see that the final solution is optimal. Furthermore, the algorithm does have the desired property: after k iterations the variables $x_1, \ldots, x_k$ have the optimal value (while $x_{k+1}, \ldots, x_n$ have the original one), i.e., *the current iterate x^k minimises the objective over $x^0 + span(\{\, u^1\,, u^2\,, \ldots, u^k\,\})$.*

The example may look rather artificial and uninteresting, since one wants to work with non-diagonal Q. However, it serves to introduce the really important idea, which in fact had already been used (albeit in a simple way) in §1.2.3 and §1.3.4: *if the problem at hand is "difficult" in its natural space, find a different space in which it is "easy"*. In this particular instance, this could be declined as: *find a space where Q is diagonal.*

This is in fact possible, at least if Q is nonsingular. The relevant object is the *square root of* Q, i.e., the matrix $R = Q^{1/2}$ such that $Q = RR$. Such a matrix surely exists if $Q \succeq 0$, and in fact there can be many different ones. A simple way to choose a *symmetric* one is via the spectral decomposition $Q = H\Lambda H^T$, where as usual H is orthonormal: indeed, it is easy to check that

$$R = H\sqrt{\Lambda}H^T$$

has the desired property.

Exercise 1.28. Verify the last statement. **[solution]**

If Q is nonsingular, i.e., $\lambda_n > 0$, then also $\sqrt{\lambda_n} > 0$ and hence R is nonsingular as well: this allows to define the linear transformation

$$z = Rx \qquad \equiv \qquad x = R^{-1}z$$

which is clearly a bijection from $\mathbb{R}^n$ to $\mathbb{R}^n$. The useful property is that

$$h(\,z\,) = f(\,R^{-1}z\,) = \tfrac{1}{2}z^T I z + qR^{-1}z\ ,$$

i.e., in the z-space the objective becomes separable. Thus, one could apply Algorithm 1.2 in the z-space to find the optimal solution z^*, and then map it back as $x^* = R^{-1}z^*$ to solve the original problem. Of course, the line $z + \alpha u^i$ in z-space corresponds to the line $R^{-1}(z + \alpha u^i) = (x + \alpha R^{-1}u^i)$ in x-space: hence, the algorithm can be entirely re-interpreted in x-space as moving along the directions $d^i = R^{-1}u^i$. These directions have the sough-after property, i.e., after k iterations the optimal solution on $x^0 + span(\{\, d^1\,, d^2\,, \ldots, d^k\,\})$ has been found. The feat is, then, possible.

The outlined approach is not practical, in that computing $Q^{1/2}$ is no less costly than solving $Qx = -q$ in the first place. However, it does suggest what the fundamental property is: the employed directions d^i should be *all orthogonal in z-space.* That is, with v^i the direction in z-space one has $d^i = R^{-1}v^i$, i.e., $Rd^i = v^i$, and wants $\langle\, v^i\,, v^j\,\rangle = 0$ for all $i \neq j$. Intuitively, this leads to an efficient optimization since at every iteration k, the new direction v^k is orthogonal to $span(\{\, v^1\,, \ldots, v^{k-1}\,\})$, and therefore "is exploring a part of the space that has never been searched through before". Remarkably, this property is not true for the gradient method. In fact, while it is true that $\langle\, g^i\,, g^{i+1}\,\rangle = 0$ for all, i, the property does not extend to subsequent iteration: that is $\langle\, g^i\,, g^{i+2}\,\rangle \neq 0$ can happen. Indeed, this surely happens when $n = 2$: since g^i and g^{i+2} are both orthogonal to g^{i+1}, they are necessarily parallel to each other (cf. Figure 1.9), which leads to the detrimental "zig-zagging" phenomenon. Ensuring orthogonality in z-space means $0 = \langle\, v^i\,, v^j\,\rangle = \langle\, Rd^i\,, Rd^j\,\rangle = d^iQd^j$: directions with this property are called *Q-conjugate.*

It is then possible to prove that using any n Q-conjugate directions in x-space leads to an algorithm that terminates in n iterations. This is still glossing over on how these directions are obtained, which will be discussed later on. Assuming that $p^i \neq 0$, $i = 1, 2, \ldots n$, are Q-conjugate directions, it is easy to see that they are necessarily *linearly independent* from each other, and therefore they form a basis of $\mathbb{R}^n$.

Exercise 1.29. Verify the linear independence. [**solution**]

One then considers the following abstract algorithm: with arbitrary x^0, compute $\alpha^i = \text{argmin}\{\, \varphi_{x^i,p^i}(\, \alpha\,) \,:\, \alpha \in \mathbb{R}\,\}$ and set $x^{i+1} = x^i + \alpha^i p^i$. It will now be proven that $x^n = x^*$, i.e., the algorithm terminates in n steps.

Since the p^i are a base of $\mathbb{R}^n$, "the optimal solution is reachable from the initial estimate by moving along the p^i"; formally speaking, there exist some $\sigma^i \in \mathbb{R}$, $i = 1, \ldots, n$, such that $x^* = x^0 + \sum_{i=1}^n \sigma^i p^i$. Let $g^i = Qx^i + q$ be the gradient at iterate x^i of the process: we want to prove that

$$\sigma^i = \alpha^i = \text{argmin}\{\, \varphi_{x^i,p^i}(\, \alpha\,)\,\} = -\langle\, g^i\,,\, p^i\,\rangle \,/\, (\,(p^i)^T Q p^i\,)$$

i.e., that the *optimal stepsizes* produced by minimizing the tomography precisely provide the multipliers σ^i. This immediately proves that the algorithm finds x^* in n steps.

Exercise 1.30. Prove the optimal stepsize formula. [**solution**]

To prove the claim, fix any $k = 1, \ldots, n$ and multiply $x^* - x^0$ by $(p^k)^T Q$: then,

$$(p^k)^T Q(x^* - x^0) = (p^k)^T Q(\textstyle\sum_{i=1}^n \sigma^i p^i) = \sigma^k (p^k)^T Q p^k$$

(using Q-conjugacy, i.e., $(p^k)^T Q p^i = 0$ if $i \neq k$). Similarly, multiply $x^k = x^0 + \sum_{i=1}^{k-1} \sigma^i p^i$ by $(p^k)^T Q$ to obtain

$$(p^k)^T Q(x^k - x^0) = (p^k)^T Q(\textstyle\sum_{i=1}^{k-1} \sigma^i p^i) = 0$$

(again, using Q-conjugacy). Putting all together,

$$\begin{aligned} \sigma^k (p^k)^T Q p^k &= (p^k)^T Q(x^* - x^0) = (p^k)^T(-b) - (p^k)^T Q x^0 \\ &= (p^k)^T(-b) - (p^k)^T Q x^k = -(p^k)^T(Qx^k + b) = -\langle\, g^k\,,\, p^k\,\rangle \end{aligned}$$

which proves the claim. Thus, the algorithm works, provided one is able to generate n Q-conjugate directions. Unsurprisingly, the eigenvectors of Q would do the job, but they are not generally available.

Exercise 1.31. Prove that the eigenvectors are Q-conjugate. [**solution**]

Fortunately, a very simple formula is available to construct the directions p^i iteratively, just at the moment when they are needed.

1.5.2 The conjugate gradient method for quadratic functions

The *conjugate gradient method* for quadratic functions is based on the following result: the i-th Q-conjugate direction can be obtained by properly *deflecting* the gradient at the i-th iterate g^i and the *previous* direction p^{i-1} (taken as 0 when $i = 0$). More formally, $p^0 = g^0$ and

$$p^i = -g^i + \beta^i p^{i-1} \quad \text{where} \quad \beta^i = \langle\, g^i\,,\, p^{i-1}\,\rangle \,/\, (\,(p^{i-1})^T Q p^{i-1}\,)$$

for $i > 0$. This is known as the *Fletcher-Reeves formula.* The fundamental property is the following: if the stepsize α^i is chosen as $\text{argmin}\{\, \varphi_{x^i,p^ki}(\alpha) \,:\, \alpha \in R\,\}$, i.e., $\alpha^i = -\langle\, g^i\,,\, p^i\,\rangle \,/\, (\,(p^i)^T Q p^i\,)$, then the directions p^i generated in this way are Q-conjugate. The proof, by induction, is somewhat long and tedious and is not reported here; it can be found, e.g., in [8, Theorem 5.2, 5.3]. All in all, this defines the following algorithm:

Algorithm 1.3. Conjugate Gradient for Quadratic Function

```
procedure $x = CGQ\ (\, Q\,,\, q\,,\, x\,,\, \varepsilon\,)$
  $d^- \leftarrow 0$;
  do forever
    $g \leftarrow Qx + q$;
    if( $\|\, g\,\| \leq \varepsilon$ ) then break;
    if( $d^- = 0$ )  then $d \leftarrow -g$;
                     else { $\beta = (\, g^T Q\, d^-\,) \,/\, (\,(d^-)^T Q\, d^-\,)$; $d \leftarrow -g + \beta d^-$; }
    $\alpha \leftarrow -\langle\, g\,,\, d\,\rangle \,/\, (\, d^T Q d\,)$; $x \leftarrow x + \alpha d$; $d^- \leftarrow d$;
```

The naïve statement of the algorithm contains several matrix-vector products with Q, each of which costs $O(\, n^2\,)$. However, the following alternative formulæ

$$\beta^i = \|\, g^i\,\|^2 \,/\, \|\, g^{i-1}\,\|^2 \quad , \quad \alpha^i = \|\, g^i\,\|^2 \,/\, (\,(d^i)^T Q\, d^i\,)$$

[8, p. 11] and further appropriate streamlining along the lines of Exercise 1.19 makes it possible to implement the algorithm with approximately the same iteration cost as the gradient method: only one $O(n^2)$ matrix-vector product plus a smattering of $O(n)$ vector operations. The algorithm is therefore going to be competitive if it takes (many) fewer iterations than the gradient method.

Differently from the gradient method, the conjugate gradient algorithm enjoys *finite convergence* in n steps, since it is precisely constructed to achieve this result. However, this is only true in infinite-precision arithmetic, and therefore not in practical implementations. The numerical errors inherent in the use of floating-point arithmetic make it so that the algorithm may require more than n iterations to achieve an approximately optimal solution; on the other hand, depending on the ε of the stopping criterion it may also terminate in less than n iterations. This is fortunate, since for large-scale problems n is not a "small" number of iterations: indeed, such a convergence result is clearly *not dimension independent*, and therefore paradoxically the gradient method may turn out to be preferable (at least for well-conditioned Q) as n grows large. This is generally not the case, as several powerful efficiency results can be proven for the conjugate gradient that show that the method should in general be expected to be significantly more efficient than the gradient method. We will briefly outline the main ones of them, without any proof: these can be found, e.g., in [8, p. 112+] and the references therein, but they are too complex to be reproduced here.

- The convergence rate of the algorithm is at least linear [8, (5.36)], i.e.,

  $$a^i \leq [\,(\sqrt{\kappa} - 1\,)\,/\,(\sqrt{\kappa} + 1\,)\,]^i (2a^0)\ .$$

  Although the convergence rate still deteriorates as the conditioning increases, it is already significantly better than that of the gradient method. Disregarding the extra factor, the base $r = (\sqrt{\kappa} - 1\,)\,/\,(\sqrt{\kappa} + 1\,)$ of the exponential is significantly smaller than that of (1.19) due to the fact that $\sqrt{\kappa} < \kappa$. For $\kappa = 1000$, for instance, (1.19) yields $r \approx 0.996$, while the current formula yields $r \approx 0.88$, and the difference is very significant. Indeed, while $0.96^{100} \approx 0.017$, $0.00^{100} \approx$ `3e-6`: for the same Q, 100 iterations would give around two significant digits with the gradient method and around six with the conjugate gradient one.

- In fact, the convergence depends on "more than the largest and the smallest eigenvalue", but on the *whole distribution of the eigenvalues.* The fundamental result [8, Theorem 5.5] is

  $$a^i \leq [\,(\lambda_{i+1} - \lambda_n\,)\,/\,(\lambda_{i+1} + \lambda_n\,)\,]a^0\ ,$$

  which basically states that "a^i converges to 0 as fast as the (finite, decreasing) sequence of eigenvalues $\{\lambda_i\}$ converges to its smallest element λ_n". This implies, for instance, that if Q has *only k distinct eigenvalues*, then the algorithm terminates in k iterations [8, Theorem 5.4]. Eigenvalues need not to be strictly identical for the result to apply: for instance, if Q has k eigenvalues *clustered around* 1, then the algorithm can be expected to terminate—or at least provide a highly accurate solution—in $n-k$ iterations [8, p. 116].

Thus, the conjugate gradient algorithm can be expected to be more efficient than the gradient method, especially so if *Q has some structure*, meaning for instance that it has several identical, or at least closely clustered, eigenvalues.

An illustration of the superior practical efficiency of the conjugate gradient approach is given in Figure 1.11 on the same two instances with $n = 1000$ of Figure 1.10. The difference is striking. Even in the case where $Q \succeq 0$ (around 10% of null eigenvalues), the conjugate gradient reaches $\approx$ `1e-15` in less than 200 iterations, where the gradient method was still at `1e-8` after 100000 iterations. In the positive definite case, a similarly highly accurate solution is reached in about 40 iterations, i.e., *three orders of magnitude less* than those the gradient needed to reach `1e-12`. In both cases the convergence plot shows signs of *superlinear* convergence, whereby the slope of the curve becomes *steeper* as the iterations progress, or at least of very good linear one (with $r \ll 1$), as the theory suggests.

1.5.3 Preconditioning the Conjugate Gradient method

Although the conjugate gradient method is already quite (more) efficient (than the gradient method), there can be ways to make it even more efficient. We will rapidly sketch one of these ways, primarily to stress that 1) some powerful ideas will keep showing their worth over and over again, and 2) that there is often ample scope to improve the performances of optimization algorithms exploiting specific forms of *structure* on the instances at hand—or, perhaps more to the point, "forcefully imposing" the structure on the instance.

In this case, the previous analysis has shown that the conjugate gradient method would behave much better if all eigenvalues were clustered around 1. In fact, we have already seen in §1.5.1 that this can be done with an appropriate change in the variables space. We now generalise the approach by selecting any *nonsingular* matrix

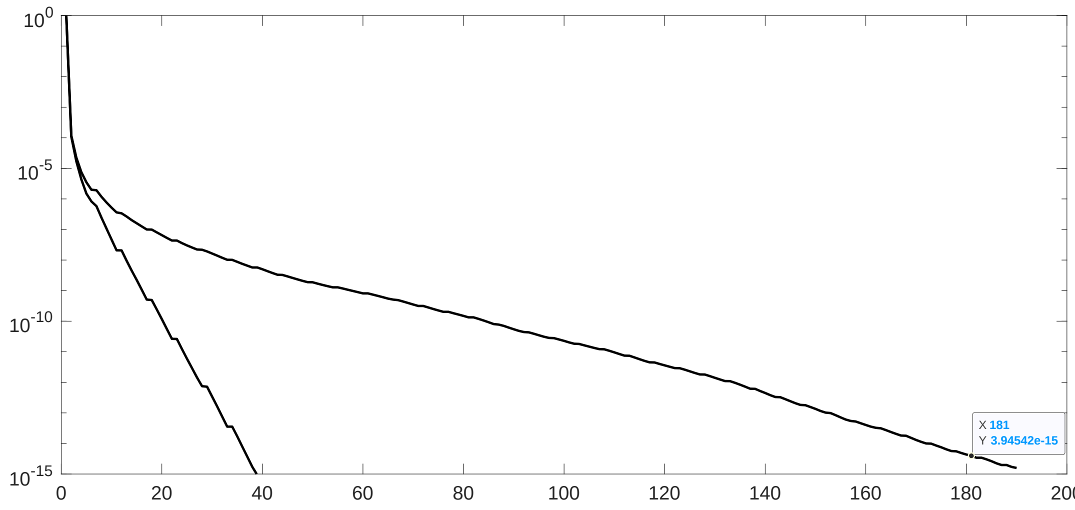


Figure 1.11: Convergence of the conjugate gradient method in two larger examples

P and defining the linear bijection in the usual way, i.e., $z = Px \equiv x = P^{-1}z$. The objective function is transformed into

$$h(\,z\,) = f(\,P^{-1}z\,) = \tfrac{1}{2}z^T[\,P^{-T}QP^{-1}\,]z + qP^{-1}z\ .$$

The choice $P = R = \sqrt{Q}$ would yield $P^{-T}QP^{-1} = I$, i.e., all eigenvalues identical to 1: the best possible scenario for the conjugate gradient, that would solve the problem—in z-space—in one iteration. In fact, even the humble gradient method would: the first iteration of the two coincide. Yet, as already mentioned, computing $P = R$ and inverting it is in general as costly as solving $Qx + q = 0$ outright, hence again the naïve approach would not be convenient.

Yet, this suggests a clever idea: choose "$P \approx Q$" such that computing P^{-1} is "cheap". It is outside of the scope of these lecture notes to go in details of what "$P \approx Q$" should exactly mean, but the previous section suggests that what one wants is that $P^{-T}QP^{-1}$ has more identical/clustered eigenvalues. Such a P is called *preconditioner*, on the grounds that, by hopefully "compressing the range of eigenvalues", it could reduce the conditioning number κ. Applying the conjugate gradient method to the *preconditioned problem*—modified using P—is called the *preconditioned conjugate gradient method*. This can be done without really changing Q and q, but rather streamlining the algorithm so that only extra products between vectors and P^{-1} are required [8, Algorithm 5.3]. It is thus highly convenient if P^{-1}, besides being "cheap to compute", is also *sparse*.

There is a huge literature about constructing preconditioners for problems with very specific structure that is entirely out of scope for these lecture notes. A very common and simple "general-purpose" preconditioner is obtained by the *Cholesky factorization* procedure, briefly described in §1.6, that could be used to solve the system outright—basically, computing a "perfect preconditioner". The *incomplete Cholesky* preconditioner [8, p. 120] is obtained by forcibly dropping (many of the) elements of the factorisation so that it is (much) sparser and cheaper to use. Rather than going into the (somewhat intricate) details of the method, we briefly describe its "extreme version": the *diagonal preconditioner* $P = \sqrt{diag(\,Q\,)}$. Basically, one pretends that Q is diagonal and constructs the preconditioner only looking at the diagonal elements. P^{-1} is therefore extremely cheap to compute and extremely sparse. An issue is that P may be singular ($Q_{ii} = 0$ for some i, which implies $Q \not\succ 0$), but easy heuristics can be used to skirt around this case.

Exercise 1.32. Propose a heuristic to modify the diagonal preconditioner if $Q \not\succ 0$. **[solution]**

The effectiveness of such a "poorman's preconditoner" cannot be expected to be extremely high; it mostly dependent on how much *diagonally dominant* Q is, a notion that will not be discussed in details but that, roughly speaking, measures "how much the diagonal weighs in the context of the whole matrix". Obviously, a diagonal matrix is completely diagonal dominant, and a matrix where the diagonal elements are much larger than the non-diagonal ones will tend to be highly diagonally dominant. The effect of the diagonal preconditioner can be expected to be significant on highly diagonally dominant matrices and to decrease, up to basically vanishing, on less and less diagonally dominant Q. An illustration of the effect of preconditioning is given in Figure 1.12 for a problem with $n = 1000$ where Q is "mildly diagonally dominant" (diagonal elements are, on average, between two and three times the non-diagonal ones). Both the grandient method and the conjugate gradient method are tested, with and without preconditioning. All instances were solved to a gap of $\approx$ `1e-15`.

The example shows that preconditioning can be effective in the gradient method, too. In fact, while the original condition number of Q is $\kappa =$ `2e+3`, the condition number of $P^{-1}QP^{-1}$ is $\kappa =$ `1.4e+3`. This difference is enough

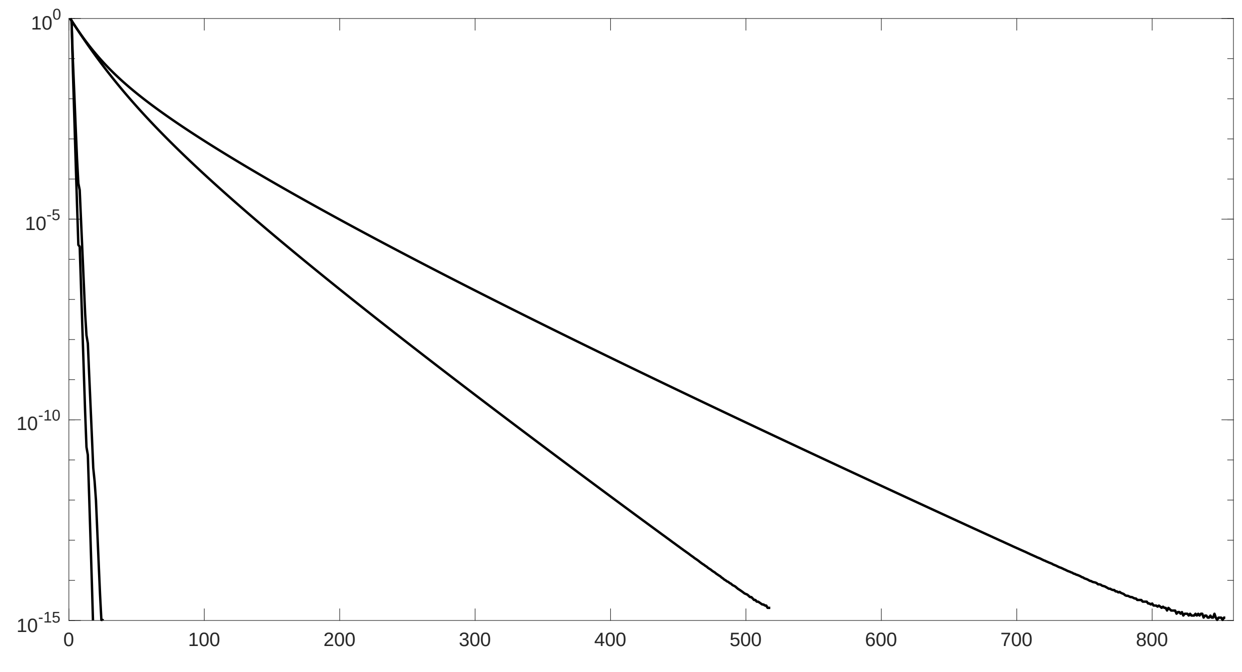


Figure 1.12: Convergence of preconditioned/or not conjugate/or not gradient methods

to decrease the required number of iterations from 854 to 518: decreasing the condition number by 30% reduces the number of iterations by 40%, roughly in accordance with the theory. The non-preconditioned conjugate gradient is way faster, solving the problem in only 25 iterations. Yet, the preconditioned version is even faster, requiring only 17 ones; yet another reduction of 30%. This unique example does not pretend to be illustrative of the trade-offs (cost for computing/using the preconditioner vs. decrease in iterations) of the many proposed preconditioned techniques, but just to hint at the fact that appropriate exploitation of the structure of the problem—in this case, of Q—may lead to improved performances. Much more sophisticated ways of exploiting the structure exist, even for problems as simple as the one at hand; a few will be hinted at later on in these lecture notes.

1.6 Multivariate Quadratic optimization: a Direct Method

Since the very simple optimization problem at hand reduced to the solution of the linear system $Qx = -q$, it is worth exploring *direct methods* that aim ad producing the solution in one blow. These will no longer be an option on more complex problems, but the techniques discussed here will find several uses down the line. There is a huge amount of sophisticated theory about direct methods for linear systems, e.g., [8, Algorithm A.1, A.2], and many efficient readily available implementations (`lapack`, ...). In these lecture notes only one technique will be discussed is some detail.

Assuming $Q \succ 0$, the solution to the system can just be written as $x = -Q^{-1}q$. However, in practice computing inverses is not the preferred way for solving systems; rather, *factorizations* are considered the be the better options. In the case of symmetric positive semidefinite matrices, the *Cholesky—a.k.a. LLT—factorization* is usually the go-to technique, although not the only possible one. It is based on rewriting

$$Q = LL^T$$

(whence the name) where L is a *lower triangular* matrix. This is clearly related to the preconditioning idea of the previous section: the "perfect preconditioner" would be such that $P^{-T}QP^{-1} = I$, i.e., $Q = P^TP$. However, the L sought after here is clearly not the "abstract" $P = H\sqrt{\Lambda}H^T$; not only because it is nontrivial to compute, but especially because rationale of the Cholesky factorization is that solving linear systems with triangular matrices, be them lower or upper, is "very easy", as recalled below. This implies that, once the factorization is computed (which is the costly part), the system can be efficiently solved with the two-step process

$$\text{solve} \quad Lv = -q \quad \text{and then solve} \quad L^Tx = v \; .$$

It is worth mentioning that, once L is known, one can basically assume to have L^{-1} available as well, in the sense that "having available" usually means being able to (efficiently) compute matrix-vector products with it. Here, $z = L^{-1}v$ and $z = L^{-T}v$ are clearly available by solving the systems $Lz = v$ and $L^Tz = w$, which, as shown next, is "cheap".

1.6.1 Backsolve

The backsolve procedure for solving a linear system with a (lower or upper) triangular matrix is really trivial, and it is better illustrated by a simple example.

Example 1.4. Backsolve
We have

$$Q = \begin{bmatrix} 4 & 8 & 4 \\ 8 & 25 & 2 \\ 4 & 2 & 12 \end{bmatrix} = LL^T \qquad \text{where} \qquad L = \begin{bmatrix} 2 & 0 & 0 \\ 4 & 3 & 0 \\ 2 & -2 & 2 \end{bmatrix}$$

(how to compute L out of Q will be seen later on). We want to solve

$$Lx = \begin{bmatrix} 2 & 0 & 0 \\ 4 & 3 & 0 \\ 2 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \\ 4 \end{bmatrix} = b \qquad \textit{i.e.} \qquad \left\{ \begin{array}{llll} 2x_1 & & & = 6 \\ 4x_1 & +3x_2 & & = 9 \\ 2x_1 & -2x_2 & +2x_3 & = 4 \end{array} \right. .$$

Obviously, the first equation immediately gives $x_1 = 6/2 = 3$. Then, the second reduces to

$$4(3) + 3x_2 = 9 \quad \equiv \quad 3x_2 = -3 \quad \equiv \quad x_2 = -1 \; .$$

Similarly, the third gives

$$2(3) - 2(-1) + 2x_3 = 4 \quad \equiv \quad 2x_3 = -4 \quad \equiv \quad x_3 = -2 \; .$$

At each step, all variables save the rightmost one (on the diagonal), if any, have already been fixed; then, it's a trivial matter to compute the remaining one.

Only slightly more formally, at the first step $L_{11}x_1 = b_1$ gives $x_1 = b_1 \,/\, L_{11}$. Then, at the k-th step, the variables $x_1, \ldots, x_{k-1}$ are known; hence

$$\textstyle\sum_{i=1}^{k} L_{ki}x_i = b_k \quad \equiv \quad L_{kk} + x_k + \sum_{i=1}^{k-1} L_{ki}x_i = b_k \quad \equiv \quad x_k = (\, b_k - \sum_{i=1}^{k-1} L_{ki}x_i \,) \,/\, L_{kk} \; .$$

This is just repeated for $k = 2, \ldots n$; each iteration has a $O(\, n \,)$ cost, which means that the whole backsolve is $O(\, n^2 \,)$. It is completely obvious how the approach is applied to L^T, i.e., in the upper triangular case: just proceed backwards for $k = n, n-1, \ldots, 1$.

It is apparent how the whole procedure hinges on the fact that $L_{ii} \neq 0$ (in fact, $L_{ii} > 0$) for all $i = 1, \ldots, n$; numerically, then, there could be issues if $L_{ii} \approx 0$ for some i, which means (as it will be clear in the following) that Q is "almost singular", i.e., some of its eigenvalues are "close to 0". Apart from this, the procedure is as efficient as it can get. In fact, if L has $m \leq n(n+1) \,/\, 2$ nonzero entries, then the total cost is $O(\, m \,)$ (sums and products, plus $O(\, n \,)$ divisions); thus, the procedure readily exploits *sparsity* ($m \ll n(n+1) \,/\, 2$).

1.6.2 Cholesky factorization

Once backsolve is established, the Cholesky factorization process becomes easy. Again, an example is the efficient way to present it.

Example 1.5. Backsolve
We want to determine

$$L = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \qquad \text{such that} \qquad Q = \begin{bmatrix} 4 & & \\ 8 & 25 & \\ 4 & 2 & 12 \end{bmatrix} = LL^T$$

which means

$$\begin{bmatrix} 4 & & \\ 8 & 25 & \\ 4 & 2 & 12 \end{bmatrix} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11} & L_{21} & L_{31} \\ L_{22} & L_{32} & 0 \\ L_{33} & 0 & 0 \end{bmatrix} = \begin{bmatrix} L_{11}^2 & & \\ L_{11}L_{21} & L_{21}^2 + L_{22}^2 & \\ L_{11}L_{31} & L_{21}L_{31} + L_{22}L_{32} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{bmatrix}$$

(for symmetry only the lower triangular parts need be considered). This is a system of $n(n+1) \,/\, 2 = 6$ *nonlinear* (containing products of variables, and therefore "quadratic") equations in the L_{ij} variables. As such it is nontrivial to solve; however, the observation is that *if the equations are approached in just the right order they can be solved as linear ones*:

1. This of course starts with with the top-left equation: $L_{11}^2 = 4$ immediately gives $L_{11} = \sqrt{4} = 2$.
2. Since now L_{11} is known, the system corresponding to Q_{21} is $L_{11}L_{21} = 2L_{21} = 8$, yielding $L_{21} = 4$.
3. This paves the way for solving the system corresponding to Q_{22}, i.e., $L_{21}^2 + L_{22}^2 = 16 + L_{22}^2 = 25$, yielding $L_{22} = \sqrt{25 - 16} = 3$.

So far things have been simple, with only one variable at the time involved; but in general the process is more complex and it involves linear systems. In fact, we have now to consider together *both* positions $(\,3\,,\,1\,)$ and $(\,3\,,\,2\,)$ because they both depend on the variable L_{31}. That is, one has to consider the system

$$\begin{cases} L_{11}L_{31} &= 4 \\ L_{21}L_{31} + L_{22}L_{32} &= 2 \end{cases} \equiv \begin{cases} 2L_{31} &= 4 \\ 4L_{31} + 3L_{32} &= 2 \end{cases} \equiv \begin{bmatrix} 2 & 0 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} L_{31} \\ L_{32} \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} .$$

This is, however, a lower triangular system and therefore it can be efficiently (back)solved, yielding $[\, L_{31} \, , \, L_{32} \,] = [\, 2 \, , \, -2 \,]$. This finally put us in the position of considering the $(\, 3 \, , \, 3 \,)$ position:

$$12 = L_{31}^2 + L_{32}^2 + L_{33}^2 = 4 + 4 + L_{33}^2 \quad \equiv \quad L_{33} = \sqrt{12 - 8} = 2 \, .$$

Thus, in general the process requires backsolves to compute the off-diagonal elements of L, one row at a time, and then a square root extraction to compute the diagonal ones.

In a more formal way, after having done the trivial "base case" of $L_{11} = \sqrt{Q_{11}}$, the general process looks as follows. At the generic k-th iteration ($k > 1$), one has

$$\begin{bmatrix} L_{-k} & 0 & \dots \\ L_k & L_{kk} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} L_{-k}^T & L_k^T & \dots \\ 0 & L_{kk} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} L_{-k}L_{-k}^T & L_{-k}L_k^T & \dots \\ L_k L_{-k}^T & L_{kk} + L_k^T L_k & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} Q_{-k} & Q_k^T & \dots \\ Q_k & Q_{kk} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} .$$

Here, L_{-k} is the $(k-1) \times (k-1)$ principal submatrix of L, that has been already correctly computed in the previous $k-1$ iterations: that is, $L_{-k}L_{-k}^T = Q_{-k}$, where Q_{-k} is the $(k-1) \times (k-1)$ principal submatrix of L. One then wants to compute the k-th row of L, distinguishing the $k-1$ off-diagonal elements, L_k, from the diagonal one L_{kk}.

For the former, it is immediate to see that the nonlinear system requires $L_{-k}L_k^T = Q_k^T$, where Q_k are the $k-1$ off-diagonal elements of the k-th row of Q: since L_{-k} is known, this can be (back)solved in $O(\, k^2 \,)$. Once this is done, for the diagonal element one has

$$L_k^T L_k + L_{kk}^2 = Q_{kk} \quad \equiv \quad L_{kk} = \sqrt{Q_{kk} - \|\, L_k \,\|^2}$$

and the process has been iterated. Since it takes n iterations, each of which $O(\, n^2 \,)$, the total cost is $O(\, n^3 \,)$.

A variant of the procedure produces the so-called LDLT factorization, where $Q = LDL^T$, L is a lower triangular matrix with all 1s on the diagonal, and D is a diagonal matrix (with all non-negative diagonal entries). This basically requires the same memory (the diagonal elements of L, being all 1s, do not need to be stored and therefore the corresponding n memory locations can be used for the diagonal elements of D) and avoids taking square roots. This is reminiscent of the spectral decomposition $Q = H\Lambda H^T$, but obviously different; yet, sometimes the diagonal elements of D are used as approximation of the eigenvalues. In terms of solving systems, the LDLT factorization is of course also as efficient as the LLT one; both are numerically stable, i.e., they provide numerically accurate solutions (insomuch as this is possible) even if Q is ill-conditioned. Many other interesting results on the Cholesky factorization can be found in the literature, starting from [50].

It is easy to see that the numerically delicate part of the procedure is the square root extraction when computing the diagonal element. This first of all requires $Q_{kk} \geq \|\, L_k \,\|^2$; this property is guaranteed to hold for $Q \succeq 0$, but it breaks if a negative eigenvalue exists. In fact, if $Q_{kk} < \|\, L_k \,\|^2$ happens at some k the procedure is stopped declaring that the decomposition does not exist. Of course, the square root of a negative number does exist if complex numbers are allowed, but this is of no interest for these lecture notes. Rather, the case where some eigenvalue is 0 merits some disscusion. This is revealed precisely by the occurrence that $Q_{kk} = \|\, L_k \,\|^2$, yielding a null diagonal element $L_{kk} = 0$ in the factor. Of course numerical safeguards have to be put in place in this context, as there is no such thing as a check "$Q_{kk} = \|\, L_k \,\|^2$" in floating-point arithmetic, but we assume this done without further details. The issue is rather that such an occurrence actually "breaks the process", in that subsequent iterations rely on backsolves with L_{-k}, but these are ill-defined if some diagonal elements of the factor are 0.

There clearly is, however, a case where this is not an issue: that of $k = n$. In this case, the result is "just" that the last diagonal element of L is null. For instance, it is easy to see that

$$Q = \begin{bmatrix} 4 & 8 & 4 \\ 8 & 25 & 2 \\ 4 & 2 & 8 \end{bmatrix} = LL^T \qquad \text{where} \qquad L = \begin{bmatrix} 2 & 0 & 0 \\ 4 & 3 & 0 \\ 2 & -2 & 0 \end{bmatrix} .$$

The issue of course is that there is no guarantee that this will only happen at the last iteration. However, a simple and elegant trick can be used to solve the conundrum: *if* $Q_{kk} = \|\, L_k \,\|^2$ *happens, swap the k-th row* (and column) *of Q with the n-th.* Indeed, for the purpose of solving the linear system $Qx = -q$, the order of the rows of Q is irrelevant and can be permuted arbitrarily (provided that, of course, the entries of q are permuted in the same way). Since Q is symmetric, swapping two rows requires swapping the two corresponding columns as well, but this only entails at permuting the corresponding entries of the solution vector x, so that the original solution can still be efficiently recovered (provided that the final permutation is stored, which can be done efficiently). Thus, if $Q_{kk} = \|\, L_k \,\|^2$ happens *just once*, even if for $k < n$, the situation can be handled.

This suggests a way in which the Cholesky factorisation could be extended to the singular case. The idea will only be roughly sketched since this is *not* the approach that is usually advised (e.g., [8, p. 610]), although the two share the fact that they perform row exchanges "when things go badly". The idea is that the previous result, working for just one row, can be extended to finding a *lower trapezoidal factorization* of Q: after a permutation of the rows and columns of Q, the matrix can be rewritten in such a way that

$$Q = \begin{bmatrix} Q_{++} & Q_{0+}^T \\ Q_{0+} & Q_{00} \end{bmatrix} = LL^T \qquad \text{where} \qquad L = \begin{bmatrix} L_{++} & 0 \\ L_{0+} & 0 \end{bmatrix} ,$$

where $Q_{0+} \in \mathbb{R}^{h \times (n-h)}$ with h being the number of null eigenvalues, and $Q_{++} \in \mathbb{R}^{(n-h)\times(n-h)}$ is nonsingular. Obviously, $n - h$ is the number of strictly positive eigenvalues. This is also called a *rank revealing Cholesky factorisation* in that it explicitly reveals the rank of Q, i.e., the number of nonzero eigenvalues.

Exercise 1.33. Prove the existence of the rank revealing Cholesky factorisation. **[solution]**

Computing such lower trapezoidal factorization is simply a matter of performing appropriate row and column exchanges. That is, at each step of the process ons has found a $k \times k$ nonsingular triangular factor L_{-k} of the first k rows of Q (possibly, after permutations performed in previous iterations), and then h subsequent rows that so far give rise to the rank-deficent part, i.e.,

$$P = \begin{bmatrix} L_{-k} & 0 \\ L_h & 0 \end{bmatrix} \qquad \text{with} \qquad PP^T = \begin{bmatrix} L_{-k}L_{-k}^T & L_{-k}L_h^T \\ L_hL_{-k}^T & L_hL_h^T \end{bmatrix} = \begin{bmatrix} Q_{-k} & Q_h^T \\ Q_h & Q_{hh} \end{bmatrix} .$$

At the beginning $k = h = 0$ and everything is empty. One then proceeds at examining row $k+1+h$, computing $L_{k+1+h} = Q_{k+1+h}L_{-k}^{-T}$ (by backsolve), with Q_{k+1+h} indicating the first k entries of the $(k+1+h)$-th row of Q (empty if $k = 0$), and then $L_{k+1+h,k+1+h} = Q_{k+1+h,k+1+h} - \| L_{k+1+h} \|^2$. If $L_{k+1+h,k+1+h} = 0$ (note that this can happen at the first iteration if $Q_{11} = 0$) then h is increased by 1 and the process repeated. Otherwise, row (and column) $k+1+h$ of Q is swapped with row (and column) $k+1$, so that $[\, L_{k+1+h} \,,\, L_{k+1+h,k+1+h} \,]$ becomes the next row $[\, L_{k+1} \,,\, L_{k+1,k+1} \,]$ of the "nonsingular part" of the factorization, and k is increased by 1. Notice than in this case all the elements $L_{i,k+1}$ for $i = k+1, \ldots, k+h-1$ need be computed to keep the invariant that $L_h \in \mathbb{R}^{h\times k}$ when k is increased. It is however obvious how to do that efficiently. The process terminates in n steps when $k + h = n$ having produced the required rank-revealing factorization with an overall cost of $O(\, n^3 \,)$. All in all, the Cholesky factorization "just works" for Q having an arbitrary number of null eigenvalues, at the cost of accepting an all-zero bottom-left block in the factor.

1.6.3 Using the Cholesky factorization

If $Q \succ 0$ the Cholesky factorisation can be computed (albeit at a rather high cost when n is large) and used to solve the $Qx = -q$ system, and hence the optimization problem, as previously discussed. It is interesting to detail, however, whet happens when $Q \succeq 0$. As seen in §1.3.4, this may lead to two different outcomes:

1. $Qx = -q$ has infinitely many solutions, each of which is optimal for the optimization problem;
2. $Qx = -q$ has no solution, which means that the optimization problem can be proven unbounded below.

We will now discuss how the Cholesky factorisation, in its lower-trapezoidal (rank-revealing) form

$$Q = \begin{bmatrix} L_{++} & 0 \\ L_{0+} & 0 \end{bmatrix} \begin{bmatrix} L_{++}^T & L_{+0}^T \\ 0 & 0 \end{bmatrix} ,$$

allows to precisely characterise which of the two cases happen, as well as "all the ways in which the problem can be optimal / unbounded below". As in the nonsingular version, the process starts with solving the sustem

$$Lv = -q \qquad \equiv \qquad \begin{bmatrix} L_{++} & 0 \\ L_{0+} & 0 \end{bmatrix} \begin{bmatrix} v_+ \\ v_0 \end{bmatrix} = - \begin{bmatrix} q_+ \\ q_0 \end{bmatrix} .$$

In such a system, the entries of v_0 are always multiplied by 0 and therefore play no role: in fact only those of v_+ matter. The system is therefore *overdetermined*: it has n constraints in $n - k$ variables. Yet, it may still have a solution: to check if this is the case it is sufficient to compute

$$\bar{v}_+ = -L_{++}^{-1} q_+$$

that "solves the first part of the system", and check whether

$$L_{0+}\bar{v}_+ = -q_0 \tag{1.22}$$

holds or not. This distinguishes the two cases.

In fact, if (1.22) holds, the original system has infinitely many solutions, and therefore the optimization problem has infinitely many optimal solutions. These can be fully characterised as the points satisfying

$$\begin{bmatrix} L_{++}^T & L_{+0}^T \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_+ \\ x_0 \end{bmatrix} = \begin{bmatrix} \bar{v}_+ \\ 0 \end{bmatrix}$$

(since the entries of v_0 are irrelevant they can as well taken to be 0). Such a system is *underdetermined*: it has $n - k$ constraints (the last k ones being always verified) in n variables, and therefore it has infinitely many solutions, i.e., all vectors of the form

$$\begin{bmatrix} x_+ \\ x_0 \end{bmatrix} = \begin{bmatrix} -L_{++}^{-T}(q_+ + L_{+0}^T x_0) \\ x_0 \end{bmatrix} \quad \text{however chosen} \quad x_0 \in \mathbb{R}^k \ ,$$

as these are all and only the vectors that satisfy $Qx = -q$.

If, conversely, (1.22) does not hold, then the system has no solution: the problem is unbounded below, i.e., there exist directions $\xi \in \mathbb{R}^n$ such that $Q\xi = 0$ but $\langle q \, , \, \xi \rangle \neq 0$, so that the tomography $\varphi_{0,\xi}(\, \alpha \,)$ is a linear function with nonzero slope. Again, these directions can be fully characterised. In fact, any ξ such that

$$\begin{bmatrix} L_{++}^T & L_{+0}^T \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \xi_+ \\ \xi_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

clearly has the property that $Q\xi = 0$; again, that system is underdetermined and it has the infinitely many solutions

$$\begin{bmatrix} \xi_+ \\ \xi_0 \end{bmatrix} = \begin{bmatrix} -L_{++}^{-T} L_{+0}^T \xi_0 \\ \xi_0 \end{bmatrix} \quad \text{however chosen} \quad \xi_0 \in \mathbb{R}^k \ .$$

For each of these

$$\langle \, q \, , \, \xi \, \rangle = \langle \, q_+ \, , \, \xi_+ \, \rangle + \langle \, q_0 \, , \, \xi_0 \, \rangle = -q_+^T L_{++}^{-T} L_{+0}^T \xi_0 + \langle \, q_0 \, , \, \xi_0 \, \rangle = \langle \, q_0 - L_{0+} L_{++}^{-1} q_+ \, , \, \xi_0 \, \rangle$$

where $\gamma_0 = q_0 - L_{0+} L_{++}^{-1} q_+ = q_0 - L_{0+} \bar{v}_+ \neq 0$ by hypothesis. Hence, taking any ξ_0 such that $\langle \, \gamma_0 \, , \, \xi_0 \, \rangle \neq 0$ (surely some exist, say $\xi_0 = \gamma_0$) provides a direction where the tomography is linear and not constant, providing a "certificate of unboundedness" of the optimization problem. All this provides an (efficient, insomuch as possible) algorithmic way to "substantiate" the results of §1.3.4.

1.6.4 Efficiency of the Cholesky factorization

There is not too much to discuss about the efficiency of the Cholesky factorization: for better or for worse, it quite closely track the $O(\, n^3 \,)$ complexity predicted by the theory (save for the usual effect of memory hierarchies). This is, however, mostly "for worse" in one sense: *L may be rather dense even if Q were fairly sparse to start with.* Albeit there are heuristic approaches that permute the rows and columns of Q to try to diminish the effect, the *fill-in* can be very significant. Indeed, it is memory consumption, before computational cost, that usually makes Cholesky factorization unfeasible as n grows. To get a sense of its behaviour, a few experiments about the relative performances of `Matlab`'s highly optimised Cholesky factorization routines and a home-made, by-the-book `Matlab` implementation of the Conjugate Gradient algorithm (even without preconditioning) is reported below for two classes of completely random matrices: dense (all elements nonzero) and mildly sparse (10% of the elements nonzero).

	dense		sparse	
n	Chol	CG	Chol	CG
1000	0.005	0.002	0.047	0.007
2000	0.026	0.016	0.176	0.002
4000	0.131	0.083	0.773	0.010
8000	0.895	0.421	5.240	0.042
16000	5.477	1.807	25.65	0.166

Table 1.1: Running times of Cholesky factorization and Conjugate Gradient on dense and sparse metrices

The results show that the growth of the running time is approximately cubic. For it being exactly so, a doubling of size would lead to the running time increasing by a factor of 8. However, for Cholesky the time includes that of backsolves, that are rather $O(\, n^2 \,)$, so the total factor should be somewhat less. For dense instances Cholesky shows a factor between 5 and 6, while CG between 4 and 5 (i.e., slightly more than quadratic). The running time favours CG (despite it being a by-the-book implementation vs. `Matlab`'s highly optimised native Cholesky) by a factor between 2 and 3; a significant but not dramatic advantage. For sparse matrices, however, CG grows by a factor very close to 4, i.e., almost perfectly quadratic. Cholesky does grow somewhat slower than in the dense case, but by a factor always visibly larger than 4; coupled with the fact that CG is already much faster even for $n = 1000$, the advantage grows up to over two orders of magnitude for larger values of n. Interestingly, and somewhat counter-intuitively, Cholesky is significantly *slower* in the sparse case than in the dense one.

The main culprit is likely the fact that `Matlab` tries to work with sparse factors due to Q being sparse, but ultimately ending up with dense ones due to catastrophic fill-in: indeed, the obtained Cholesky factors are completely dense. Using sparse data structures for dense matrices is way less efficient, possibly explaining some of the above results. By contrast, sparsity is efficiently exploited in the matrix-vector products that constitutes the computational bottleneck of the Conjugate Gradient method: indeed, for large n the method is close to 10 times faster on sparse than on dense instances (for the same size), closely matching the expected performance advantage due to the fact that the scalar product need work only with 1/10 of the entries.

These few experiments do not pretend to portray the full gamut of the possible computational trade-offs; for instance, CG is significantly influenced by the distribution of eigenvalues while Cholesky is not, hence the relative performances may be rather different on different classes of matrices. However, they at least give a first taste as to how different solution methods applied to the same problem can provide very different performance profiles. Although direct methods like Cholesky factorization are usually not preferred, especially for large-scale problems, they still have their uses in the context of complex optimization algorithms. For instance, a use case that may significantly favour Cholesky factorization is multiple systems, with the same Q but different q^1, q^2, $\dots$, q^k, need be solved. Clearly, in this case the $O(n^3)$ cost is paid only once, and then each solution costs $O(n^2)$; if k is large the ammortised cost of the solution of each individual system may be close to $O(n^2)$. This is indeed occurring in some of the more complex optimization algorithms to be explored in the these lecture notes (although typically not with a very large k), justifying the interest in having direct methods in the toolbox of available numerical techniques.

1.7 Ex-post motivation: Polynomial Interpolation

We now provide a motivation for wanting to solve quadratic optimization problems that is related to *learning*. The setting is that of simple *interpolation* problem: we are given a real-world process that can be described by a univariate function $h : \mathbb{R} \to \mathbb{R}$. The physical foundations of the process may be unclear, or too complex to be described by an *analytical model*, i.e., under the form of algebraic or differential equations that can be explicitly or numerically solved. What is available is a finite set (but possibly large) of observations, or *samples*, $S = \{ (x_i , y_i) : i \in I \}$, with $y_i = h(x_i)$. The task is to construct a *syntetic, or data-driven, model*, i.e., some function $m : \mathbb{R} \to \mathbb{R}$ that approximates h. This of course implies that $m(x_i) \approx y_i = h(x_i)$ for $i \in I$; more importantly, the model should be able to *predict* the right value of $h(x)$ even for x that does not coincide with any of the x_i. For a number of good reasons, some of which will be commented upon later on, it is also desirable that m is "simple".

The standard way in which the problem is addressed is to *choose a-priori a parametric family of simple functions*, and then seek the value of the parameters that identify the function in the family that best *fits* the available data. In our case we choose perhaps a family of simple algebraic functions: *polynomials* of an arbitrarily fixed degree k, i.e.,

$$m_c(x) = c_0 + \textstyle\sum_{i=1}^{k} c_i x^i \ .$$

A polynomial of degree k is identified by $k{+}1$ real parameters, i.e., a vector $c = [\, c_0 \, , \, c_+ = [\, c_1 \, , \, \dots \, , \, c_k \,] \,] \in \mathbb{R}^{k+1}$; we use the notation m_c to stress this dependency. The *fitting problem* could then be stated as: find $c \in \mathbb{R}^{k+1}$ such that $y_i = m_c(x_i)$ for all $i \in I$. It is, however, obvious that this may be impossible to achieve, in particular if the degree k is "small" with respect with the cardinality n of I (the number of samples). It is therefore bettnecessaryer to seek for the $c \in \mathbb{R}^{k+1}$ that "fits as well as possible". This is typically implemented by defining a *loss function* $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ that measures "how unequal its two arguments are"; that is, $\mathcal{L}(z , y) = 0$ if and only if $z = y$, with the value becoming larger and larger as z and y become less and less similar. This therefore measures the "loss in accuracy if one accepts z as a substitute value for y". Perhaps the most natural loss is $\mathcal{L}(z , y) = (z - y)^2$. Then, the problem becomes to minimize the *total loss* $\mathcal{L}(c) = \sum_{i \in I} \mathcal{L}(y_i , m_c(x_i))$, also called the *empirical risk*: how much *the model fails the predict the true function* on the *available observations*.

We can now formally state the polynomial fitting problem (for degree k). It is convenient to define the vector of outputs and the matrix of inputs

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad , \quad X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^k \end{bmatrix} \ .$$

The crucial point is that, given S, y and X are *fixed*, i.e., they can be easily computed (in fact, if k is "large" the rows of X may contain numbers of wildly varying magnitude and this may lead to numerical problems, but we avoid to discuss these fine details). Clearly, given the row $X_i = [\, 1 \, , \, x_i \, , \, x_i^2 \, , \, \dots \, , \, x_i^k \,]$, $m_c(x_i) = \langle c \, , \, X_i \rangle$; i.e., the function is *linear* in the coefficients c and in the fixed vector X_i. Hence, the fitting problem is

$$\min \left\{ \mathcal{L}(c) = \sum_{i \in I} \mathcal{L}(y_i, \langle c, X_i \rangle) = \left|\left| y - Xc \right|\right|^2 : c \in \mathbb{R}^{k+1} \right\} .$$

This is called a *Linear Least Squares* problem, and it is clearly the unconstrained minimization of a quadratic function. In fact,

$$f(c) = \left|\left| y - Xc \right|\right|^2 = (y - Xc)^T (y - Xc) = \| y \|^2 + 2(y^T X)c + c^T (X^T X)c .$$

It is immediate to realize that $Q = X^T X \succeq 0$. Furthermore, the problem cannot clearly be unbounded below since the norm is always non-negative: hence, the problem surely has an optimal solution (albeit it may not be unique) that can be found with the techniques seen above. In fact, solving this kind of problems is so common that in `Matlab` this is just obtained with the simple command `c = y / X`.

Example 1.6. Polynomial interpolation

We now illustrate the process, and point to some of its issues, with a small example. We consider the function $h(x) = x \sin(20x)$, and we collect samples in the interval $[0, 1]$ regularly spaced at distance 0.01 from each other. We then perform polynomial interpolation with $k = 8$ and $k = 12$, respectively. In Figure 1.13(left) these are plotted as, respectively, the blue and the red continuous line, while h is plotted as the thicker black dashed line. It is easy to see that the "more complex" model with $k = 12$ tracks h much better than the "less complex" one. Of course, the model is somehow more costly to obtain, but not tremendously so, and the much higher accuracy is arguably worth the increase in computational cost.

However, the issue with "complex" models is that they tend to *overfit*: that is, learn very accurately the phenomenon (function) at the sampled points, but at the cost of accuracy in points different from the sampled ones. This is illustrated (cheating a bit, but this is not a significant issue for these lecture notes) in Figure 1.13(right), where the predicted polynomials are plotted "out of sample": not only in the interval $[0, 1]$, but also "slightly more on the right" in $[1, 1.05]$. It is easy to see that the quality of the prediction dramatically deteriorates; while this may be expected, the more complex model, that does better "in sample", does way worse "out of sample". Again, this is only for illustration purposes and this particular example should not be taken too seriously, but it does point to a real conundrum, perhaps one of the most important ones, in learning known as the *bias/variance dilemma*. In layman terms, models should be "of the right complexity": if they are not enough complex they are not even capable of faithfully reproducing the phenomenon (function) in the sampled points, but if they are too complex they do not *generalize* well on points that are not very close to the sampled one.

This issue, which would be central in a learning course, is *not* relevant for these lecture notes: here we don't care if the *fitting problem* we solve leads to good learning outcomes or not, just that we solve it as efficiently as possible. This is reasonable since synthetic models have *hyperparameters* that can be tuned to attain the best learning outcomes; in the case of polynomial interpolation, for instance, this is the degree k. Since it is usually unknown a-priori which (degree) will be best, *hyperparameter tuning* is required that implies solving the fitting problem many times over. Thus, efficient optimization algorithms, that are what these lecture notes are all about, are important for effective learning. This is not to say that learning completely boils down to optimization; on the contrary, a number of issues must be faced that these lecture notes purposely ignore. It remains true that efficient optimization techniques for solving fitting problems, usually much more demanding ones than the very simple one illustrated here, are an important component of effective learning approaches.

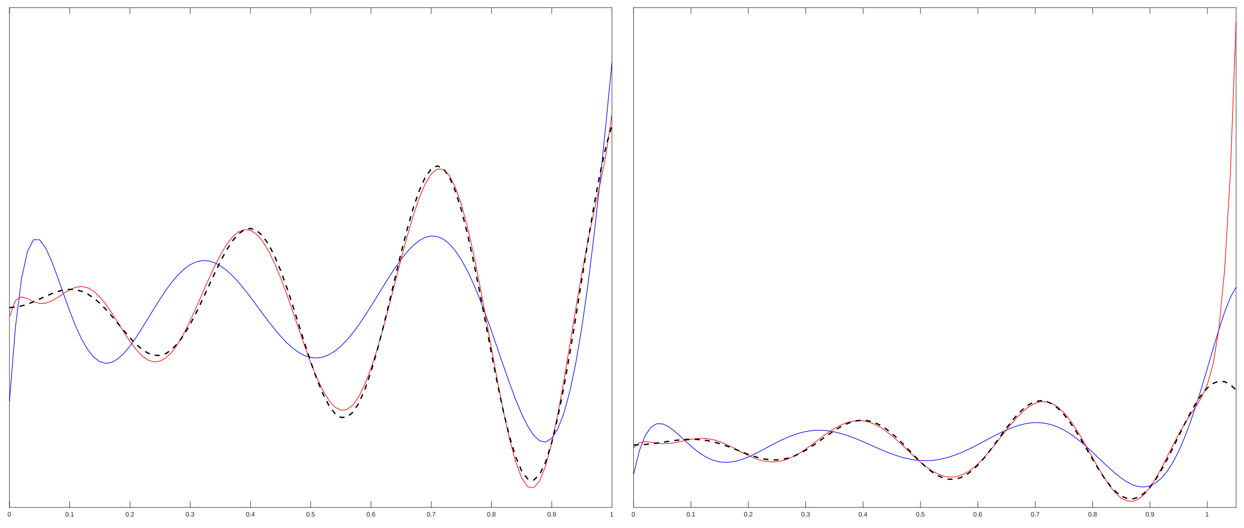


Figure 1.13: "In-sample" (left) and "out-of-sample" (right) polynomial interpolation

1.8 Wrap up

This section has hopefully provided a useful "warm up" for the (much) more complex problems that will be tackled in the next sections of these lecture notes. Although the optimization of quadratic (multivariate) functions is generally "easy", it is already a complex enough task to clearly delineate the typical workflow for solving some optimization problem:

1. study the properties of the problem and devise *optimality conditions* that characterise the sought-after

optimal solutions (comprised understanding when none exists);

2. starting from the optimality conditions devise solution approaches that allow to construct solutions (approximately) achieving them;
3. analyse the efficiency of the devised solution approaches to identify the cases in which the approach may not be performant enough;
4. on the base of this analysis hopefully devise alternative solution methods that (possibly, in some cases) have better performances.

This typically ends up with multiple solution approaches, each of which more appropriate for some cases and less on others. Almost never the "perfect" approach turns up that is always unquestionably better than every other, since there can be many different *structures* in the problem that some approaches may be more capable of exploiting than others.

Many of the tools that have been introduced in this section for "simple" problems will be useful, possibly in more sophisticated versions, for more complex ones. Indeed, one of the most useful ideas is that of *using simple problems as stepping stones for solving complex ones.* This may happen in different forms:

- simple problems are typically easier to analyse and therefore may provide intuition for the solution of more complex, and harder to analyse, ones;

- solving a complex problem may entail solving a sequence of simpler ones;

- a powerful strategy for solving a complex problem is to devise a simpler one that "approximates" it somehow (a *model*) and solve it instead, typically iterating the process (cf. the previous point).

The "simple" linear and quadratic functions are in fact crucial for devising *models* of the more complex ones to be tackled in the following. Indeed, it will soon become apparent that while linear functions are "really too simple", quadratic ones are "already sophisticated enough to provide excellent models", justifying the in-depth analysis of this section. As a final remark, the complexity of optimization problems must never be underestimated: even optimising "simple" quadratic functions over "trivial" boxes is an exceedingly difficult ($\mathcal{NP}$-hard) problem. Keeping on the right side of the complexity landscape will be a constant struggle across all these lecture notes.

1.9 Solutions

- **Solution to Exercise 1.1:** The practical solution is to use $\max\{\,|\,f_*\,|\,,\,1\,\}$ in the denominator instead of $|\,f_*\,|$. This can be seen as "use the relative error for large f_* and the absolute one for small f_*". That is, one argues that 0 is of the order of `1e+0`, and therefore the right "scale" for the error is 1. One may contend that if f_* is very close to 0 but not 0, say `1e-12`, then a relative error of, say `1e-6` would correspond to `1e-18`. However, the issue here becomes that of the numerical accuracy. Roughly speaking, the error one makes for computing some quantity v is related to the machine precision and *the largest* numbers that went in the computation. Since it is very unlikely that none of the numbers involved in the computation of $f(\,x\,)$ is at least of the order of `1e+0`, one should expect the numerical error to be at least of the order of the machine precision. This is a rough argument and debatable, but works most of the time in practice. Another way of seeing it is to assume one is minimizing $g(\,x\,) = f(\,x\,) + 1$, so that $f_* = 1$; this problem is equivalent to the original one, as discussed next.

- **Solution to Exercise 1.2:** If $b > 0$ and $x - z > 0$, then $b(x - z) > 0$, i.e., $bx > bz$, i.e., $f(\,x\,) > f(\,z\,)$; the other cases are analogous (or simpler).

- **Solution to Exercise 1.3:** Let $b > 0$. If $\underline{x} = -\infty$, then obviously $x^* = -\infty = \underline{x}$. If $\underline{x} > -\infty$, since $f(\,x\,)$ is increasing, $f(\,x\,) > f(\,\underline{x}\,)$ $\forall x > \underline{x}$, i.e., $\underline{x}$ is the minimum. The treatment of $\overline{x}$ when maximising is analogous. If $b < 0$, the role of $\underline{x}$ and $\overline{x}$ reverses: $\overline{x}$ is the argmin, and $\underline{x}$ is the argmax. Obviously, if $b = 0$ then every point in X is an optimal solution.

- **Solution to Exercise 1.4:** Pick $x > z$, $a > 0$ and $x > 0$: then $f(\,z\,) = ax^2 > axz > az^2 = f(\,z\,)$. Since $f(\,x\,)$ is symmetric ($ax^2 = a(-x)^2$), the fact that f is increasing for $x > 0$ immediately implies that it is deceasing for $x < 0$. When $a < 0$ the sign of the inequalities in inverted: the function is reflected upon the x axis. The case $a = 0$ is trivial.

- **Solution to Exercise 1.5:** $f(\,x\,)$ has a minimum in 0, is decreasing for $x < 0$ and increasing for $x > 0$. If $\underline{x} > 0$ then $f(\,x\,)$ is increasing along all $X = [\,\underline{x}\,,\,\overline{x}\,]$, hence $\underline{x}$ is the minimum and $\overline{x}$ the maximum. The argument is symmetric if $\overline{x} < 0$. Obviously, if $0 \in X$ then it is the minimum. For the maximization, since the function is increasing when x moves away from 0 in both directions, the maximum has to be one of the two extremes but we don't know which until we test. The rest is trivial.

- **Solution to Exercise 1.6:** $f(\,x\,) = (\,ax + b\,)x$, hence the roots are $x = 0$ and $x = x_p = -b\,/\,a$. Clearly, $\bar{x} = -b\,/\,2a$ is always in the middle of the interval defined by the roots. If a and b have the same sign then $x_p < \bar{x} < 0$, otherwise $x_p > \bar{x} > 0$

- **Solution to Exercise 1.7:** We assume that i. and ii. hold for f and we want to show that $f(\,x\,) = \langle\, b\,,\,x\,\rangle$ for some $b \in \mathbb{R}^n$. Let u_i, $i \in I$, the i-th vector of the canonical base of $\mathbb{R}^n$ (having 1 in the i-th position and 0 otherwise), and $b_i = f(\,u_i\,)$. For any $x \in \mathbb{R}^n$, $x = \sum_{i \in I} x_i u_i$, hence $f(\,x\,) = f(\,\sum_{i \in I} x_i u_i\,) = \sum_{i \in I} f(\,x_i u_i\,)$ (using ii. recursively n times) $= \sum_{i \in I} x_i f(\,u_i\,)$ (using i. on each individual term) $= \sum_{i \in I} b_i x_i$ (using the definition of b_i) $= \langle\, b\,,\,x\,\rangle$ (using the definition of scalar product). The result clearly breaks in the affine case ($c \neq 0$): for $f(\,x\,) = x + 1$, $f(\,2x\,) = 2x + 1 \neq 2(\,x + 1\,) = 2f(\,x\,)$.

- **Solution to Exercise 1.8:** By contradiction, assume there exists $\gamma \in \mathbb{R}^n \setminus \{\,0\,\}$ s.t. $H\gamma = 0$; this implies $0 = \|\,H\gamma\,\|^2 = \gamma^T[\,H^T H\,]\gamma = \|\,\gamma\,\|^2 > 0$ since $H^T H = I$, a contradiction.

- **Solution to Exercise 1.9:** This is based on a general result: for $[\,A^1\,,\,A^2\,,\,\dots\,,\,A^n\,] = A \in \mathbb{R}^{m \times n}$ (not necessarily square) written by columns, $AA^T = M \in \mathbb{R}^{m \times m}$ (symmetric, prove it using $[\,AB\,]^T = B^T A^T$) can be written as the sum of the n rank-one matrices corresponding to the columns, i.e., $M = \sum_{i \in I}[\,D^i = A^i(A^i)^T\,]$. In fact, the h-th row of A is $A_h = [\,A^1_h\,,\,A^2_h\,,\,\dots\,,\,A^n_h\,]$ and the k-th column of A^T is the k-th row of A, thus $M_{hk} = \langle\,A_h\,,\,A_k\,\rangle = \sum_{i \in I} A^i_h A^i_k$. But $D^i_{hk} = A^i_h A^i_k$, hence $M_{hk} = \sum_{i=1}^n D^i_{hk}$ for all h and k. To complete the result, for $\Lambda = \mathrm{diag}(\,[\,\lambda_1\,,\,\lambda_2\,,\,\dots\,,\,\lambda_n\,]\,) \in \mathbb{R}^{n \times n}$, $L = A\Lambda = = [\,\lambda_1 A^1\,,\,\lambda_2 A^2\,,\,\dots\,,\,\lambda_n A^n\,]$. In fact, the h-th row $A_h = [\,A^1_h\,,\,A^2_h\,,\,\dots\,,\,A^n_h\,]$ and the k-th column of Λ, i.e., $\lambda_k u_k$ (u_k being the k-th vector of the canonical base) give $L_{hk} = \langle\,A^h\,,\,\lambda_k u_k\,\rangle = \lambda_k A^k_h$.

- **Solution to Exercise 1.10:** For the first case we have
$$Q = \left(\frac{\sqrt{2}}{2}\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}\right)\begin{bmatrix} 8 & 0 \\ 0 & 4 \end{bmatrix}\left(\frac{\sqrt{2}}{2}\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} 12 & -4 \\ -4 & 12 \end{bmatrix} = \begin{bmatrix} 6 & -2 \\ -2 & 6 \end{bmatrix}.$$
Analogously, in the other three cases we have respectively
$$Q = \begin{bmatrix} 5 & -3 \\ -3 & 5 \end{bmatrix} \quad , \quad Q = \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix} \quad , \quad Q = \begin{bmatrix} 3 & -5 \\ -5 & 3 \end{bmatrix} .$$

- **Solution to Exercise 1.11:** $x = z + \bar{x} \implies \frac{1}{2}x^T Q x + qx = \frac{1}{2}(\,z + \bar{x}\,)^T Q(\,z + \bar{x}\,) + q(\,z + \bar{x}\,) = \frac{1}{2}z^T Q z + z^T(\,Q\bar{x} + q\,) + [\,\frac{1}{2}\bar{x}^T Q\bar{x} + q\bar{x}\,] = \frac{1}{2}z^T Q z + f(\,\bar{x}\,)$ as $Q\bar{x} + q = Q(-Q^{-1}q) + q = -q + q = 0$.

- **Solution to Exercise 1.12:** $Qv = Q[\,\sum_{i \in Z} \beta_i H_i\,] = \sum_{i \in Z} \beta_i Q H_i = \sum_{i \in Z} \beta_i \lambda_i H_i = 0$.

- **Solution to Exercise 1.13:** $Q = H\Lambda H^T = \sum_{i=1}^n \lambda_i H_i H_i^T = \sum_{i \in Z} \lambda_i H_i H_i^T[\,= 0\,] + \sum_{i \in N} \lambda_i H_i H_i^T$. We want to prove $\exists\, x$ s.t. $(\,\sum_{i \in N} \lambda_i H_i H_i^T\,)x = \sum_{i \in N} \mu_i H_i = w$. This is true if $\lambda_i H_i^T x = \mu_i \quad i \in N \quad \equiv H_i^T x = \gamma_i = \mu_i\,/\,\lambda_i \quad i \in N$, a linear system of $k \leq n$ equations in n variables (likely underdetermined). Since all H_i are linearly independent, $H_N = [\,H_i\,]_{i \in N} \in \mathbb{R}^{n \times k} \implies rank(\,H_N\,) = k$, which implies that $[\,H_N^T\,,\,\gamma\,] \in \mathbb{R}^{k \times n+1}$ has rank k (rank $\leq$ number of rows), i.e., by [68] the system has a solution x (∞-ly many if $k < n$).

- **Solution to Exercise 1.14:** $\frac{1}{2}x^TQx + qx = \frac{1}{2}(z+\bar{x})^TQ(z+\bar{x}) + q(z+\bar{x}) = \frac{1}{2}z^TQz + z^T(Q\bar{x} + q^+ + q^0) + f(\bar{x}) = \frac{1}{2}z^TQz + q^0z + f(\bar{x})$ as $Q\bar{x} + q^+ = 0$.

- **Solution to Exercise 1.15:** We know that $f(z) = z^TQz + f(\bar{x})$, with $z = x - \bar{x}$. For $x \in \bar{x} + v$, with $v \in \ker(Q)$, $z = x - \bar{x} = \bar{x} + v - \bar{x} = v$. Hence $f(z) = f(\bar{x})$. On the other hand, $f(z) \geq f(\bar{x})$ for all z since $Q \succeq 0$, thus any such point is a minimum. Any point $x \in \bar{x} + v$ with $v \notin \ker(Q)$ has $f(x) = v^TQv + f(\bar{x}) > f(\bar{x})$ since $v^TQv > 0$

- **Solution to Exercise 1.16:** $\varphi(\alpha) = a\alpha^2 + b\alpha$ is quadratic non-homogeneous with $a = (g^i)^TQg^i \geq 0$ and $b = -\|g^i\|^2 < 0$. If $a > 0$, then $\varphi(\bar{\alpha}) < \varphi(0) = f(x^i)\ \forall\bar{\alpha} \in (0, -b/a)$; in particular, $\bar{\alpha} = \|g^i\|^2/(2(g^i)^TQg^i)$ is the minimum of φ. If $a = 0$ then φ is decreasing and $\varphi(\bar{\alpha}) < \varphi(0) = f(x^i)\ \forall\bar{\alpha} > 0$.

- **Solution to Exercise 1.17:** The variational characterization of the eigenvalues implies that $\lambda_1 \geq d^TQd/\|d\|^2 \geq \lambda_n$ for all $d \neq 0$; this immediately gives $1/\lambda_1 \leq \|d\|^2/d^TQd \leq 1/\lambda_n$ for all d, and therefore in particular $d = g^i$ (knowing that $g^i \neq 0$ otherwise the algorithm would have stopped).

- **Solution to Exercise 1.18:** Because $a < 0$, the step α will be negative, which basically means one is going in direction g rather than $-g$. The algorithm remains the same, except that the extra check has to become $g^TQg \geq -\delta$.

- **Solution to Exercise 1.19:** Assuming the gradient is computed in the "natural way" as $g = Qx + q$ before the algorithm starts (i.e., with x the initial guess x^0), both quantities depending from matrix-vector products can be recovered by computing the vector $v = Qg$. In fact, $a = g^TQg = \langle g, v\rangle$. Then, with $x' = x - \alpha g$ one has $g' = Qx' + q = Q(x - \alpha g) + q = (Qx + q) - \alpha Qg = g - \alpha v$. Hence, the gradient at the next iteration can be computed in $O(n)$ out of that of the previous iteration and the vector v. As for the objective function, $x^TQx/2 + \langle q, x\rangle = (x^TQx + 2\langle q, x\rangle)/2 = x^T(Qx + q + q)/2 = \langle q + g, x\rangle/2$, i.e., it can be computed in $O(n)$ once g is known.

- **Solution to Exercise 1.20:** All arguments boil down to the crucial $Qx^* + q = 0$. This first of all gives
$$\begin{aligned} f(x^*) &= \tfrac{1}{2}(x^*)^TQx^* + \langle x^*, q\rangle = (x^*)^TQx^* + \langle x^*, q\rangle - \tfrac{1}{2}(x^*)^TQx^* \\ &= (x^*)^T(Qx^* + q) - \tfrac{1}{2}(x^*)^TQx^* = -\tfrac{1}{2}(x^*)^TQx^* \end{aligned}$$
Then, $\frac{1}{2}(x - x^*)^TQ(x - x^*) = \frac{1}{2}x^TQx + \frac{1}{2}(x^*)^TQx^* - x^T(Qx^*) = \frac{1}{2}x^TQx - \langle x, q\rangle + \frac{1}{2}(x^*)^TQx^* = f(x) - f(x^*)$ (in the penultimate step we have used $Qx^* = -q$).

- **Solution to Exercise 1.21:** $g^i = Q(x^i - x^*) = Qx^i + q$ and $\alpha^i = \|g^i\|^2/[(g^i)^TQg^i]$. This gives $g^{i+1} = Qx^{i+1} + q = Q(x^i - \alpha^ig^i) + q = (I - \alpha^iQ)g^i$, and therefore $\langle g^{i+1}, g^i\rangle = \|g^i\|^2 - \alpha^i[(g^i)^TQg^i] = 0$.

- **Solution to Exercise 1.22:** Since Q is nonsingular $x^i - x^* = Q^{-1}g^i$, whence
$$a^i = \tfrac{1}{2}(x^i - x^*)^TQ(x^i - x^*) = \tfrac{1}{2}(g^i)^TQ^{-1}g^i\ .$$
This gives
$$\begin{aligned} a^{i+1} &= \tfrac{1}{2}(x^{i+1} - x^*)^TQ(x^{i+1} - x^*) = \tfrac{1}{2}(x^i - \alpha^ig^i - x^*)^Tg^{i+1} = \tfrac{1}{2}(x^i - x^*)^Tg^{i+1} \text{ [using } \langle g^{i+1}, g^i\rangle = 0] \\ &= \tfrac{1}{2}(x^i - x^*)^TQ(x^i - \alpha^ig^i - x^*) = \tfrac{1}{2}(x^i - x^*)^TQ(x^i - x^*) - \tfrac{1}{2}\alpha^i(x^i - x^*)^TQg^i \\ &= a^i - \tfrac{1}{2}\alpha^i\|g^i\|^2 \text{ [using } Q(x^i - x^*) = g^i] = a^i - \tfrac{1}{2}\|g^i\|^4/(g^i)^TQg^i \\ &= a^i - \frac{\|g^i\|^4}{((g^i)^TQg^i)((g^i)^TQ^{-1}g^i)}\frac{\frac{1}{2}(g^i)^TQ^{-1}g^i}{} = a^i\left(1 - \frac{\|g^i\|^4}{((g^i)^TQg^i)((g^i)^TQ^{-1}g^i)}\right) \end{aligned}$$

- **Solution to Exercise 1.23:** Recall that the eigenvalues of Q^{-1} are $1/\lambda_n \geq \ldots \geq 1/\lambda_1 > 0$; from the usual $\lambda_n\|x\|^2 \leq x^TQx \leq \lambda_1\|x\|^2$ (applied to Q^{-1} as well) one has $\|g\|^2/g^TQg \geq 1/\lambda_1$ and $\|g\|^2/g^TQ^{-1}g \geq 1/[1/\lambda_n]$.

- **Solution to Exercise 1.24:** Just induction: obvious for $i = 0$, if it holds for $i - 1$ then $A(x^i) \leq rA(x^{i-1}) \leq r(r^{i-1}A(x^0))$.

- **Solution to Exercise 1.25:** $r^i a^0 \leq \delta \equiv r^i \leq \delta / a^0 \equiv \log(r^i) \leq \log(\delta / a^0)$ (log monotone) $\equiv i\log(r) \leq \log(\delta / a^0)$ (property of log); since $r < 1$, $\log(r) < 0$, giving $i \geq \log(\delta / a^0) / \log(r) = [-\log(\delta / a^0)] / [-\log(r)] = \log(a^0 / \delta) / \log(1/r) = \log(a^0 / \delta)[1 / \log(1 / r)]$.

- **Solution to Exercise 1.26:** This requires a bit of elementary calculus. The derivative of $\ln(x)$ is $1 / x$. The first-order Taylor approximation is $f(x + h) \approx f(x) + f'(x)h$ for $h \approx 0$. Applied to $\ln(\cdot)$ with $x = 1$ gives $\ln(1 + h) \approx h$, whence $1 / \ln(1 / r) = = 1 / \ln(1 + (1 - r) / r) = r / (1 - r)$. But $\log_a(x) = \log_b(x) / \log_b(a)$, hence $\ln(x) = \log_e(x) = \log_{10}(x) / \log_{10}(e) \approx \log(x) / 0.43 \approx 2.3\log(x)$, i.e., $\ln(x) \in O(\log(x))$. Then, since $r = (1 - 1 / \kappa)$, one has $\kappa = 1 / (1 - r) \approx r / (1 - r)$ since $r \approx 1$.

- **Solution to Exercise 1.27:** $\lambda_1 \| x^i - x^* \|^2 \geq (x_i - x^*)^T Q(x_i - x^*) = 2a^i \equiv \| x^i - x^* \| \geq \sqrt{2a^i / \lambda_1}$, hence $\| x^i - x^* \| \leq \gamma \implies a^i \leq \lambda_1 \gamma^2 / 2$.

- **Solution to Exercise 1.28:** $[H\sqrt{\Lambda}H^T][H\sqrt{\Lambda}H^T] = H\sqrt{\Lambda}[H^T H]\sqrt{\Lambda}H^T = H\Lambda H^T = Q$

- **Solution to Exercise 1.29:** By contradiciton, $\exists\, \gamma \in \mathbb{R}^n \setminus \{0\}$ s.t. $\sum_{i=1}^n \gamma^i p^i = 0 \implies$
  $0 = [\sum_{i=1}^n \gamma^i p^i]^T Q[\sum_{i=1}^n \gamma^i p^i] = \sum_{i=1}^n (\gamma^i)^2 (p^i)^T Q p^i$ [using $(p^i)^T Q p^j = 0$ if $i \neq j$]. But $(p^i)^T Q p^i > 0 \;\forall\, i$ since $Q \succ 0$, hence $\gamma \neq 0$ implies $\sum_{i=1}^n (\gamma^i)^2 (p^i)^T Q p^i > 0$: a contraddiction.

- **Solution to Exercise 1.30:** $\varphi_{x^i,p^i}(\alpha) = \frac{1}{2}(x^i + \alpha p^i)^T Q(x^i + \alpha p^i) + q(x^i + \alpha p^i) = f(x^i) + \frac{1}{2}\alpha^2 (p^i)^T Q p^i + \alpha\langle Qx^i + q\, ,\, p^i \rangle = f(x^i) + \frac{1}{2}\alpha^2 (p^i)^T Q p^i + \alpha\langle g^i\, ,\, p^i \rangle$ then use the results of §1.2.3.

- **Solution to Exercise 1.31:** $H^i Q H^j = H^i(\lambda_j H_j) = \lambda_j \langle H^i\, ,\, H^j \rangle = 0$ if $i \neq j$.

- **Solution to Exercise 1.32:** $Q \not\succ 0$ may mean some $Q_{ii} \leq 0$; take $\sqrt{|\, diag(Q)\, |}$ and replace every 0 (small) element with 1

- **Solution to Exercise 1.33:** It all starts with the standard spectral decomposition $Q = H\Lambda H^T$, with H the orthonormal unitary matrix of eigenvectors and Λ the diagonal matrix of eigenvalues. The vector of eigenvalues is partitioned as $\lambda = [\lambda_+\, ,\, \lambda_0] = [\lambda_+\, ,\, 0]$, i.e., distinguishing the h null ones from the $n - h$ strictly positive ones; similarly, $H = [H_+\, ,\, H_0]$, where H_+ are the eigenvectors corresponding to positive eigenvalues and H_0 these corresponding to null ones. Then, obviously $Q = H_+ \Lambda_+ H_+^T$. This is rewritten in a more explicit form as
$$Q = \begin{bmatrix} Q_{++} & Q_{0+}^T \\ Q_{0+} & Q_{00} \end{bmatrix} = \begin{bmatrix} H_{++} \\ H_{0+} \end{bmatrix} \Lambda_+ [H_{++}^T\, ,\, H_{0+}^T] = \begin{bmatrix} H_{++}\Lambda_+ H_{++}^T & H_{++}\Lambda_+ H_{0+}^T \\ H_{0+}\Lambda_+ H_{++}^T & H_{0+}\Lambda_+ H_{0+}^T \end{bmatrix} .$$
Since $Q_{++} \succ 0$ and $\Lambda_+ \succ 0$, $H_{++} \succ 0$. Thus, $H_{0+}\Lambda_+ H_{++}^T = Q_{0+}$ implies $H_{0+}\Lambda_+ = Q_{0+} H_{++}^{-T}$. Hence,
$$Q_{00} = H_{0+}\Lambda_+ H_{0+}^T = [Q_{0+} H_{++}^{-T}][\Lambda_+ - 1 H_{++}^{-1} Q_{0+}^T] = Q_{0+}[H_{++}^{-T}][\Lambda_+ - 1 H_{++}^{-1}] Q_{0+}^T = Q_{0+} Q_{++}^{-} 1 Q_{0+}^T .$$
Let now $Q_{++} = L_{++} L_{++}^T$ the standard Cholesky factorization of $Q_{++} \succ 0$, and define $L_{0+} = Q_{0+} L_{++}^{-T}$. Then
$$LL^T = \begin{bmatrix} L_{++} & 0 \\ L_{0+} & 0 \end{bmatrix} \begin{bmatrix} L_{++}^T & L_{0+}^T \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} L_{++} L_{++}^T & L_{++} L_{0+}^T \\ L_{0+} L_{++}^T & L_{0+} L_{0+}^T \end{bmatrix} = \begin{bmatrix} Q_{++} & Q_{0+}^T \\ Q_{0+} & L_{0+} L_{0+}^T \end{bmatrix} .$$
But since $L_{0+} = Q_{0+} L_{++}^{-T}$,
$$L_{0+} L_{0+}^T = Q_{0+} L_{++}^{-T} L_{++}^{-1} Q_{0+}^T = Q_{0+} Q_{++}^{-1} Q_{0+}^T = Q_{00} ,$$
proving that the proposed one is the desired rank-revealing Cholesky factorization.

Part II

Unconstrained Optimization

Chapter 2

Univariate Optimization

2.1 General Univariate Optimization Problems

Proceeding very carefully in our trip among optimization problems, we make a step forward and (apparently) three back: while we now allow the objective f to be any function (for which a pointwise evaluation oracle is available), we restrict at the same time $n = 1$, i.e., $f : \mathbb{R} \rightarrow \mathbb{R}$ to be *univariate*. As for the feasible region $X \subset \mathbb{R}$, we want it to be "easy"; thus, we only consider the (possibly, unbounded on one or both sides) interval $X = [\, \underline{x} \,, \overline{x} \,]$ with $-\infty \leq \underline{x} < \overline{x} \leq +\infty$. In §1.2, such problems were seen to be (outrageously) simple for linear and quadratic functions; however, this is far from being the case for general f. If fact, the problem is basically *impossible to solve in finite time to any fixed accuracy* $\varepsilon > 0$.

This harsh result can be proven formally [4, p. 408], but it basically boils down to the counterexample in Figure 2.1(left). Without any assumption on the oracle, save that it is "cheap", one could be confronted with an *adversarial* one that basically answers always the same value, say $f(\, x \,) = 1$. Any algorithm can only probe the value in a sequence of iterates $(\, x^i \,)$, that can (in theory) be infinite, but *countable*: "after that the sequence if finished", the oracle could "declare" that the minimum x^* of the function, say with $f_* = f(\, x^* \,) = 0$, may live in one of the points of X that have *not* been samples, since these are uncountably many, even if X is a bounded interval $(-\infty < \underline{x} < \overline{x} < +\infty)$. Thus, no algorithm can be guaranteed to find a ε-optimal solution with any given $\varepsilon > 0$ against such an adversarial oracle, albeit (with a bounded interval) one may in fact have produced iterates arbitrarily close to x^*. However, this is hardly useful since, even if this happened, the algorithm (and its user) is not able to understand it, and therefore "save" one of the 'promising" iterates (which, however, all formally correspond to arbitrarily bad solutions).

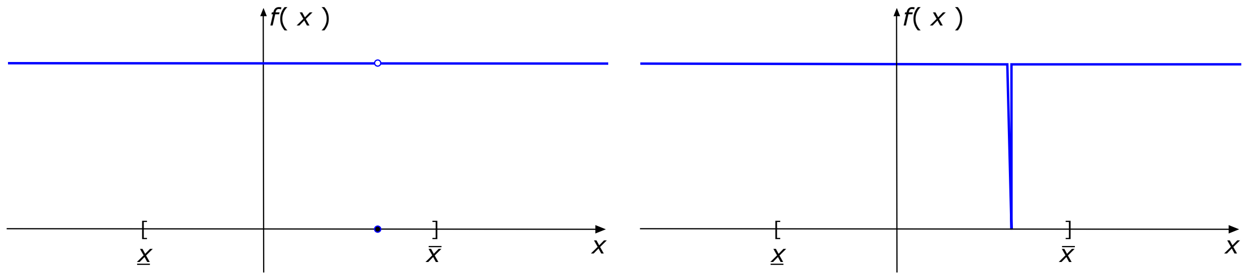


Figure 2.1: Adversarial oracles for general univariate optimization

Of course there are many objections that could be raised against this counterexample, starting from the fact that it clearly assumes x to be a "real real", while we have repeatedly insisted that in practice x will be, say, a `double` and therefore it only has a finite number of possible values. This may be little solace, as these would be at least of the order of $2^{53} \approx 10^{16}$ (a `double` has 53 bits of mantissa); even with a 10Ghz processor capable of evaluating 10^{10} points per second, checking them all would require $\approx 10^5$ seconds, i.e., more than a day. If this still looks doable, moving to the $2^{113} \approx 10^{34}$ possible values of a 128-bits `quad` would skyrocket to $\approx 10^{24}$ seconds, i.e., $\approx 10^{16}$ years, way more than the current age of the universe. Other nontrivial questions would be how the oracle could store information about all the iterates in order to place x^* somewhere else. Perhaps the most fitting observation is that no-one is really interested in optimizing an adversarial oracle; indeed, the functions involved in the applications of interest for these lecture notes are "decidedly not adversarial". However, rather than delving in philosophical issues we underline the real gist of the counterexample: *there is no hope of being able to efficiently optimize a function if evaluating it in some x does not somehow convey information about its* *behaviour in points different from x.* If the function can change arbitrarily from x to a points x' "extremely close to x", every evaluation only conveys "too limited information" to be able to efficiently drive the optimization.

One therefore may surmise that the fundamental issue with the adversarial oracle of the counterexample is that the function "jumps": it is not *continuous* (cf. A.6). However, Figure 2.1(right) readily dispels the notion. After any finite number of iterates x^i, the oracle can select two ones, x^h and x^k, so that no other one lives in the interval $(\, x^h \,,\, x^k \,)$, and then "fit a very narrow downward spike inside the interval". Such a function is continuous, and still impossible to optimize upon. Of course, the two cases are hardly different when x is actually a floating-point number, as spikes that are narrower than the machine precision actually are basically single points. However, rather than putting foo fine a point on the discussion, the gist of the result is the following: *it is impossible to optimise upon a function that is allowed to change arbitrarily rapidly.* If, again, the value $f(\, x\,)$ and $f(\, x'\,)$ can be "arbitrarily far apart" even if x and x' are "near", then evaluating the function at any point x hardly gives any useful information about its behaviour in any other points. Since *optimizing is learning about the function on the grounds of information extracted by function evaluations*, this means that such ground must be "somehow firm enough" for the process to succeed. Just any function does not provide it, hence we will do for the first time something that will be needed over and over again: *make proper assumptions on the problem* (in this case, its objective function) so that the task at hand becomes possible / efficient enough.

2.2 Lipschitz (Global) Optimization

The analysis in the previous section should have clearly revealed what kind of assumption is needed: something ensuring that "downward spikes can't be arbitrarily narrow", i.e., basically, *imposing speed limits on the rate of change of the function.* The useful concept is that of *Lipschitz continuity* (L-c) on X (cf. §A.6), which in this case reads just

$$|\, f(\, x\,) - f(\, z\,)\,| \leq L|\, x - z\,| \qquad \forall x \in X \;,\; z \in X \;.$$

for some fixed *Lipschitz constant* $L > 0$. This assumption immediately makes it possible to design an algorithm that is capable of finding ε-optimal solutions to the problem for any chosen $\varepsilon > 0$. However, for the algorithm to work it is also necessary that $X = [\, \underline{x} \,,\, \overline{x}\,]$ is *bounded*, i.e., that its *diameter* $D = \overline{x} - \underline{x}$ is *finite* (trivially, $\underline{x} > -\infty$ and $\overline{x} < \infty$). This allows to proceed as follows [4, p. 411]:

Algorithm 2.1. Global Optimization of a Lipschitz Function

```
procedure $x = SDQ\,(\, f \,,\, \underline{x} \,,\, \overline{x} \,,\, L \,,\, \varepsilon\,)$
  $D = \overline{x} - \underline{x}$; $k = \lceil\, LD \,/\, (2\varepsilon)\,\rceil$; $\Delta = D \,/\, k$; $f^{best} = +\infty$;
  for( $i \leftarrow 0$ , $z = \underline{x}$ ; $i \leq k$ ; $i \leftarrow i + 1$ , $z \leftarrow z + \Delta$ )
    if( $f(\, z\,) < f^{best}$ ) then { $f^{best} \leftarrow f(\, z\,)$; $x \leftarrow z$; }
```

Note that $\lceil\cdot\rceil$ indicates rounding up to the closest larger integer; similarly, $\lfloor\cdot\rfloor$ will indicate rounding down to the closest smaller integer, and $\lfloor\cdot\rceil$ just rounding to the closest integer. In plain words, the algorithm samples f uniformly in the interval at the smallest possible number of points such that the distance between two consecutive ones is $\Delta \leq 2\varepsilon \,/\, L$, and returns the best among all the function values it finds. This can be easily proven to be a ε-optimal solution. The intuition is that, being z^i and z^{i+1} the end-points of any one of the intervals, then

$$f(\, x\,) \geq \min\{\, f(\, z^i\,) \,,\, f(\, z^{i+1}\,)\,\} - \varepsilon \quad \text{for all} \quad x \in [\, z^i \,,\, z^{i+1}\,] \;.$$

In fact, at any point x inside the interval L-c gives

$$f(\, x\,) \geq f(\, z^i\,) - L(x - z^i) \quad \text{and} \quad f(\, x\,) \geq f(\, z^{i+1}\,) - L(z^{i+1} - x) \;;$$

with $\underline{f}^i = \min\{\, f(\, z^i\,) \,,\, f(\, z^{i+1}\,)\,\}$ this gives

$$f(\, x\,) \geq \max\{\, \underline{f}^i - L(x - z^i) \,,\, \underline{f}^i - L(z^{i+1} - x)\,\} = \underline{f}^i - L \min\{\, x - z^i \,,\, z^{i+1} - x\,\} \;.$$

Of course, the minimum on the rightmost term is maximised when x is exactly in the middle of the interval, i.e., $x - z^i = z^{i+1} - x = \Delta \,/\, \leq \varepsilon \,/\, L$, which readily yields the desired results. Since this holds for all intervals, wherever the optimal solution x^* lies its optimal value can only have been "missed" by at most ε when iterates z^i and z^{i+1} "close" to it have been evaluated, and this must have happened. This discussion is only meant to highlight the fundamental property: *Lipschitz continuity allows to bound the value of f in points "close" to those where the function is evaluated*, which is what allows to algorithm to work.

Exercise 2.1. Formally prove ε-optimality of the returned x. [**solution**]

Although this looks finally a positive results, it has a number of significant drawbacks. First of all, it requires

$O(\,LD\,/\,\varepsilon\,)$ function evaluations, i.e., at least $O(\,LD\,/\,\varepsilon\,)$ complexity even if the oracle is "very cheap". As previously discussed, this kind of complexity that is inversely proportional to the accuracy means that finding solutions with "small" ε is often practically impossible: gaining one further digit of accuracy requires one order of magnitude more iterations. The constant of the $1\,/\,\varepsilon$ term also worsens as the interval gets larger (D increases) and the function "gets faster", i.e., L increases, i.e., steeper spikes are possible. A significant issue of the result is also that *the algorithm actually requires and uses the value of* L, which is in general unknown and not easy to estimate.

These are, however, not all the bad news:

- The algorithm above is actually "optimal", in the sense that it can be proven [15, Theorem 4.4] that for every algorithm capable of finding guaranteed ε-optimal solutions there exists at least one function for which it will require no less than $O(\,LD\,/\,\varepsilon\,)$ function evaluations. The result again builds an *adversarial function*, precisely constructed to be "nasty" for that very specific algorithm; therefore, it is not necessarily significant for those of interest in these lecture notes. It still is an indication that (global) optimization is "inherently difficult" even for a univariate function; this will actually be confirmed "to the n-th degree" when we will move to multivariate case, i.e., $X \subset \mathbb{R}^n$ with $n > 1$.
- Some other interesting results can be proven, that cannot be discussed any extensively here, that confirm that "optimization is difficult". Among them, variants of the *no free lunch theorem* [16] fundamentally prove that "all algorithms are equally bad on average", i.e., "if an algorithm is very good in some cases it has to be very bad in others". This does not rule out that algorithm exist that can be extremely efficient for specific cases of interest (indeed, this is mostly what these lecture notes are about), but it does reinforce the message that "you have to carefully choose the problem if you want to have a chance at being able to efficiently solve it".

All in all, these results indicate that efficiently finding (global) ε-optimal solutions for "small" ε may be impossible in general, even in the univariate case. We will therefore be interested in understanding in which specific cases this is indeed possible. We will approach the question in a seemingly roundabout way, that however has important consequences, by poising a different question: is there a *different notion of optimality* that can be considered and that allows efficient solution approaches? Of course the answer is in the positive, as the next sections will detail.

2.3 Local optimization

Finding the (ε-)optimal solution of an (univariate) optimization problem is in general "too difficult", even when it's at all possible. However, there is a *weaker* notion of optimality that is conducive to more efficient algorithms: that of *local optimum*. The notion is very intuitive, and just corresponds to the fact that the solution is "better than all those very close by". Formally speaking, for

$$(P) \quad \min\{\, f(\,x\,) \,:\, x \in X \,\}\ ,$$

some $\bar{x} \in X$ is a local optimum if there exists some *neighbourhood* X' *of* $\bar{x}$—a set such that $\bar{x} \in I$, in the univariate case typically an interval $[\,\underline{x}'\,,\,\overline{x}'\,]$ with $\underline{x}' < \bar{x} < \overline{x}'$—such that

$$\bar{x} \in \operatorname{argmin}\{\, f(\,x\,) \,:\, x \in X \cap X' \,\}\ . \tag{2.1}$$

In plain words, $\bar{x}$ is an optimal solution of a *restriction* of (P) where one can only consider "the subset of the feasible region close to $\bar{x}$". It is immediate to see that any optimal solution x^* to (P) is a local minimum; just take $X' = \mathbb{R}$. So, "there are more local optima than bona-fide optimal solutions". To underline this fact, x^* is typically denoted—if the doubt arises—as the *global optimum* of (P). It is clearly useful if there are "not too many" local optima: indeed, for the concept of local optimum to make sense it is important to require that "X' is not too small": obviously, for any $x \in X$, taking $X' = \{\,x\,\}$ would make x a local minima. This is why one asks $\underline{x}' < \bar{x} < \overline{x}'$: it should be possible to "move a little bit both ways" (technically, $\bar{x}$ should be in the interior of X', but we'll try to avoid the finer details), and still $\bar{x}$ must remain an optimal solution. While the concept applies to the constrained case, it is perhaps more simply described when $X = \mathbb{R}$. There, the optimal solution of (P) is called the *minimum*, or *global minimum*, of f, while any local optimal solution is a *local minimum*. Of course, one can symmetrically define *global maximum* and *local maximum* of f. It is completely intuitive, by just looking at Figure 2.2(left), that a function may have multiple local minima that are not a global minimum. Of course, it may as well happen that "there is only one local minima, which necessarily has to be the global one"; this important case will be discussed later on. For now we stick to the general case, as depicted in Figure 2.2(left), where local minima may not be the global one.

The interest in the notion of local optimum/minimum is that *they can be found efficiently*, basically due to the

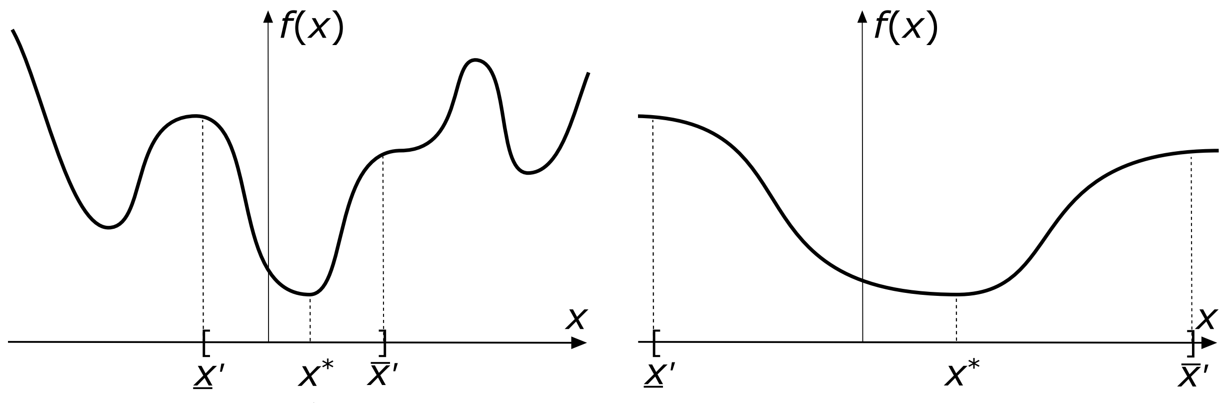


Figure 2.2: A function with local-non-global minima and one of its attraction basins

fact that "near $\bar{x}$, f has a predictable shape". The precise definition is (almost, cf. [1, Ex. 8.10] for a missing detail): f is *(strictly) unimodal* on $X = [\,\underline{x}'\,,\,\overline{x}'\,]$ if

- it has a minimum $\bar{x} \in X$;
- is *(strictly) decreasing* in $[\,\underline{x}\,,\,\bar{x}\,]$ and *(strictly) increasing* in $[\,\bar{x}\,,\,\overline{x}\,]$.

Simply put: slightly on the left of the (local) minimum the function need be decreasing, because the minimum still has not reached, but once it is reached it need increase, otherwise it would not be a minimum. The "strict" version of the notion ties in with the notion of *strict* local minimum, which just says that $\bar{x}$ is the unique optimal solution: all other points in $X' \setminus \{\,\bar{x}\,\}$ have strictly larger objective value. Of course this is hardly significant for the practical case where all values are actually `double`. Anyway, the useful observation is then: for any (strict) local minimum $\bar{x}$ there exist an interval $X = [\,\underline{x}'\,,\,\overline{x}'\,]$ containing it such that f is (strictly) unimodal on X'. The largest such X' can be called the *attraction basin* of $\bar{x}$. Figure 2.2(right) depicts the fact that while f of Figure 2.2(left) is not unimodal on the whole $\mathbb{R}$, it is so when restricted to the attraction basin of one of its local minima (in this case x^*, the global minima at least in the interval that is shown).

2.4 First local optimization algorithms

A fundamental—if obvious—property of an attraction basin is that it contains a local minima. Now, clearly if we knew an attraction basin $[\,\underline{x}\,,\,\overline{x}\,]$ "of very small diameter $D = \overline{x} - \underline{x} \leq \delta$", for a "small δ", then any point x in the basin, e.g., its extremes, could be considered a "good approximation of $\bar{x}$", in that $|\,x - \bar{x}\,| \leq \delta$. We will discuss later the nontrivial aspect of how this translates to the standard notion of approximate (local) optimality in terms of function value; cf. §1.4.4. For now, in view of the fact that (approximately) solving an optimization problem in fact amounts at (approximately) finding its optimal solution, we will be more than content with the concept of finding an approximate solution of (2.1), i.e., approximately find $\bar{x}$ to within a given tolerance δ. We assume to know beforehand an initial attraction basin $[\,\underline{x}\,,\,\overline{x}\,]$—this assumption will be discussed later on—and we want to iteratively shrink it, "without excluding $\bar{x}$", up until the point where its diameter is smaller than a user-defined threshold δ. The crucial point is that this can be done efficiently by *evaluating f at two arbitrary intermediate points*. That is [1, Th. 8.11 + Ex. 3.60]: if is f (strictly) unimodal in $[\,\underline{x}\,,\,\overline{x}\,]$, with a local minimum $\bar{x}$ lying somewhere inside the interval, then however chosen $\underline{x} < \underline{x}' < \overline{x}' < \overline{x}$:

- if $f(\,\underline{x}'\,) \geq f(\,\overline{x}'\,)$ then $\bar{x} \in [\,\underline{x}'\,,\,\overline{x}\,]$;
- if $f(\,\underline{x}'\,) \leq f(\,\overline{x}'\,)$ then $\bar{x} \in [\,\underline{x}\,,\,\overline{x}'\,]$.

The principle is simple and it is illustrated in Figure 2.3. If $f(\,\underline{x}'\,) \geq f(\,\overline{x}'\,)$ (top), then f is "decreasing in the sub-interval": $\bar{x}$ may be "further on the right" (top left) or "inside the sub-interval already", but surely it is *not* in $[\,\underline{x}\,,\,\underline{x}'\,]$. Symmetrically if $f(\,\underline{x}'\,) \leq f(\,\overline{x}'\,)$, i.e., f is "increasing in the sub-interval". Thus, at the cost of *two* evaluation of f we can restrict the attraction basin: clearly, iterating the process we can make it shrink down to any required size.

Once the general concept is established, the next relevant question obviously is: how should one choose $\underline{x}'$ and $\overline{x}'$ so that the interval shrinks as quickly as possible. The issue clearly is that each iteration of the process excludes *either* $[\,\underline{x}\,,\,\underline{x}'\,]$ *or* $[\,\overline{x}'\,,\,\overline{x}\,]$, but one does not know beforehand which of the two. Hence, by having them "of very different size" the behaviour of the algorithm would be "strongly subject to (bad) luck". That is, if one were to take $\underline{x}'$ "close" to $\underline{x}$ but $\overline{x}'$ "far" from $\overline{x}$, then in the first occurrence the attraction basin would shrink

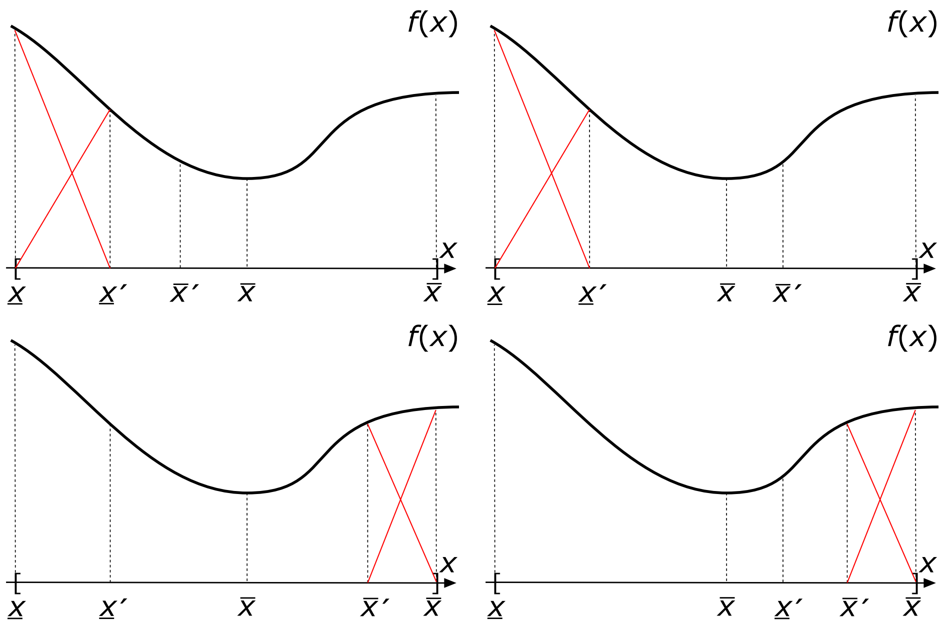


Figure 2.3: Restricting attraction basins by evaluating points inside them

"very little", while in the second it would shrink "a lot"; and of course symmetrically. Thus, the algorithm may end up being either very efficient or very inefficient (in fact, basically not working at all) depending on how "lucky" one is in choosing the interval. Since this is undesirable, the typical approach is that *optimizing the worst-case behaviour*, i.e., choose the sub-interval in such a way that the attraction basin shrinks as much as possible in the worst case. It is plain to see that this implies to take $[\,\underline{x}\,,\,\underline{x}'\,]$ and $[\,\overline{x}'\,,\,\overline{x}\,]$ of equal size. That is, one selects $r \in (\,1\,/\,2\,,\,1\,)$ and choses

$$\underline{x}' = \underline{x} + (1-r)D \quad , \quad \overline{x}' = \underline{x} + rD = \overline{x} - (1-r)D\;.$$

Thus, both intervals $[\,\underline{x}\,,\,\underline{x}'\,]$ and $[\,\overline{x}'\,,\,\overline{x}\,]$ have size $(1-r)D$, and whichever of the two get excluded the size of the remaining attraction basin is $rD < D$. This immediately implies that the smaller r is, the faster the interval shrinks and therefore the sooner the algorithm ends. On the other hand, r must remain strictly larger than $1\,/\,2$. Thus, one would select some small $\sigma > 0$, and have $r = 1\,/\,2 + \sigma$. This of course corresponds to

$$\underline{x}' = \underline{x} + D\,/\,2 - \sigma \quad , \quad \overline{x}' = \underline{x} + D\,/\,2 + \sigma = \overline{x} - D\,/\,2 + \sigma\;;$$

in plain words, "go to the middle of the interval and then step a little bit on the left and a little bit on the right". For reasons that we have previously discussed, when implemented with floating-point numbers the formula should rather be written with a "relative" σ, i.e., as

$$\underline{x}' = \underline{x} + (1-\sigma)D\,/\,2 \quad , \quad \overline{x}' = \underline{x} + (1+\sigma)D\,/\,2$$

and σ be chosen appropriately w.r.t. the stopping tolerance δ, but we avoid these fine details since, as we will see, this is not the algorithm that one really wants to run. Rather, to simplify the analysis we will ignore σ altogether and pretend that $r = 1\,/\,2$: at each time the interval is cut precisely in half. Denoting with $\overline{x}^i$ and $\underline{x}^i$ the values of the extremes at the i-th iteration, and $D^i = \overline{x}^i - \underline{x}^i$ the corresponding diameter, one then has that

$$D^i = \overline{x}^i - \underline{x}^i = D^0 r^i\;.$$

Thus, with the stopping criterion $D^i \leq \delta$, the approach stops when

$$D^0 r^i \leq \delta \quad \equiv \quad i \geq \log(\,D\,/\,\delta\,)\,/\,\log(\,1\,/\,r\,)\;.$$

Exercise 2.2. Prove the last bound. **[solution]**

Thus, the complexity of finding the local minimum only depends on $\log(\,1\,/\,\delta\,)$, as opposed on $1\,/\,\delta$ for finding the global one: the process is "exponentially faster", ad therefore can be ran with "small" δ. While the two complexities are not directly comparable, since one bounds the error ε in the output space and the other the error δ in the input one, it is easy to see that, if f is L-c with known constant $L > 0$, one can easily ensure to achieve a ε-optimal solution in $O(\,\log(\,LD\,/\,\varepsilon\,)\,)$, i.e., "exactly exponentially faster" than the global optimization bound.

Exercise 2.3. Prove the last statement. **[solution]**

Unlike in the case of the gradient method for quadratic functions, r cannot become "too close to 1". In fact, one can then precisely estimate the constant in the $O(\ )$ notation: for $r = 0.5$, $\log(1 / r) \approx 0.3$ and $1 / \log(1 / r) \approx 3.32$. It is important to remark, however, that this is the bound on the number of *iterations*. One should expect that the most costly operation in any such approach is the invocation of the *oracle* computing the f-values, which indeed could have an arbitrary cost. Since all other operations in the algorithm are $O(1)$, the reasonable metric for the cost of the algorithm should therefore be that of the number of *function evaluations*. Since clearly both points $\underline{x}^{i\prime}$ and $\overline{x}^{i\prime}$ probed at the i-the iteration are different from any of the ones probed at previous iterations, there are *two* function evaluations at each step; thus, the algorithm requires $\approx 6.64 \log(D / \delta)$ function evaluations to obtain $D^i \leq \delta$. One may therefore wonder whether it is possible to improve such a result, and the answer is positive.

The strategy for improving the efficiency is actually quite simple: accept a slight increase in the number of iterations, but compute the function in only one point at each iteration. This is clearly possible on one condition: one of the two between $\underline{x}^{i\prime}$ and $\overline{x}^{i\prime}$ (the one that does *not* become an extreme of the interval) has to "survive" and become one among $\underline{x}^{i+1\prime}$ and $\overline{x}^{i+1\prime}$, so that its function value need not be computed again. This just requires some easy algebra. By symmetry, whatever choice is made the interval restricts to being of size rD. Let us consider the case where the interval $[\,\underline{x}\,,\,\underline{x}'\,]$ is eliminated (the other case is symmetric). As shown in Figure 2.4, $\underline{x}'$ is at distance rD from $\overline{x}$, while $\overline{x}'$ is at distance $(1-r)D$. The question is whether the "surviving" $\overline{x}'$ is on the left of the middle of the new interval, and therefore becomes $\underline{x}'$ at the next iteration, or on the right and therefore remains $\overline{x}'$. In order for it to be on the left it must hold

$$1 - r > r / 2 \quad \equiv \quad r < 2 / 3 \; ;$$

since we prefer r to be as small as possible (whilst $r > 1 / 2$), to obtain the fastest possible convergence, we assume that this will be the case. Hence, $\overline{x}'$ will become $\underline{x}'$ at the next iteration. This means that (ignoring the constant factor D on all terms) $1 - r$ will have to correspond to the fraction r of the new interval $[\,\underline{x}'\,,\,\overline{x}\,]$ of length r. In formulæ

$$r \,:\, 1 = (1 - r) \,:\, r \quad \equiv \quad r \cdot r = 1 - r \; ;$$

that is, r has to be a solution of the quadratic equation $r^2 + r - 1 = 0$. These are well-known to be $(-1 \pm \sqrt{5}) / 2$, one of which is negative: then, the only choice is $r = (\sqrt{5} - 1) / 2 \approx 0.618$ ($< 2 / 3$). This is a somewhat "magic" number: $r = 1 / g$, where $g = (\sqrt{5} + 1) / 2 \approx 1.618$ is known as the *golden ratio* [56], and it has a number of nice geometric and algebraic properties, such as $g = 1 + r = 1 + 1 / g$. However, in our case this just gives us the crucial number that allows us to define the *golden ratio search* algorithm.

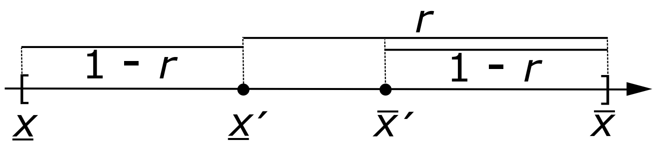


Figure 2.4: Derivation of the Golden Ratio rule

Algorithm 2.2. Golden Ratio Search

```
procedure $[\,\underline{x}\,,\,\overline{x}\,] = GRS(\,f\,,\,\underline{x}\,,\,\overline{x}\,,\,\delta\,)$
  $\underline{x}' \leftarrow \underline{x} + (1-r)(\overline{x} - \underline{x})$; $\overline{x}' = \underline{x} + r(\overline{x} - \underline{x})$; compute $f(\,\underline{x}'\,)$, $f(\,\overline{x}'\,)$;
  while( $\overline{x} - \underline{x} > \delta$ ) do
    if( $f(\underline{x}') > f(\overline{x}')$ )
      then { $\underline{x} \leftarrow \underline{x}'$; $\underline{x}' \leftarrow \overline{x}'$; $\overline{x}' \leftarrow \underline{x} + r(\overline{x} - \underline{x})$; compute $f(\,\overline{x}'\,)$; }
      else { $\overline{x} \leftarrow \overline{x}'$; $\overline{x}' \leftarrow \underline{x}'$; $\underline{x}' \leftarrow \underline{x} + (1-r)(\overline{x} - \underline{x})$; compute $f(\,\underline{x}'\,)$;}
```

By the previous analysis, $\overline{x}^i - \underline{x}^i = D^0 r^i$ with $r \approx 0.618$ gives that the algorithm stops in at most (approximately) $4.78 \log(D / \delta)$ iterations: in fact, $\log(1 / 0.618) \approx \log(1.618) \approx 0.21$, and $1 / 0.21 \approx 4.78$. Thus, the algorithm requires about 44% more iterations than the previous version to stop ($4.78 / 3.32 \approx 1.44$). However, since now each iteration requires only one function evaluation, as opposed to two, the new algorithm requires only the 72% of the function evaluations of the previous one ($4.78 / 6.64 \approx 0.72$). The difference is not staggering, but significant enough so that the golden ratio search is the preferred algorithm *if no other information is available on the function besides the function value.* Indeed, in this case the algorithm can be proven to be asymptotically optimal [1, p. 355]. It can also be proven that, if the total number n of steps to be performed is fixed in advance (and it is, once D and δ are given), then a slightly better performance can be obtained by using a *variable* r given by the ration of two successive *Fibonacci numbers*: $r^i = F_{n-i} / F_{n-i+1}$, where F_i is the i-the element of the Fibonacci sequence [61]. This is asymptotically the same, as the ratio of two consecutive Fibonacci numbers tends to the golden ratio as i increases. We leave the details of these entertaining, but ultimately not particularly useful, versions to the interested reader; the point here is to underline the fact that better algorithm can be devised by more carefully studying the problem at hand. In fact, the significance of these results is somewhat limited in view of the fact that *obtaining more information about the function is possible*, which makes it possible in turn to devise even more efficient algorithms.

Before doing so in the sext section, let us stress again the main point of this one: *finding local minima is exponentially faster than finding global ones.* This statement should be tempered by the observation that the algorithm is supposed to work on a unimodal function, and many functions are not such. However, *the algorithm can be applied as well to a non-unimodal function.* Indeed, it is clear that nothing in the algorithm "really breaks" if f is not unimodal. What happens if there are multiple local minima, and therefore multiple attraction basins, in the original interval $[\, \underline{x} \,,\, \overline{x} \,]$ is fairly obvious: at some iteration, the algorithm will *unknowingly eliminate some local minima* and the corresponding attraction basin. Sooner or later, the current interval $[\, \underline{x} \,,\, \overline{x} \,]$ will be "small enough" so that f will actually be unimodal in there, and the algorithm will eventually (approximately) find the local minimum in there. Thus, in some sense the algorithm operates in *two phases*:

1. in the *global phase*, where $[\, \underline{x} \,,\, \overline{x} \,]$ contains multiple local minima, the algorithm "randomly" choses one among the (possibly, many) attraction basins to focus on;
2. in the *local phase*, where $[\, \underline{x} \,,\, \overline{x} \,]$ finally contains only one attraction basin, the algorithm works to find the corresponding local minimum.

Of course, "randomly" in the previous description has not to be taken literally: the algorithm is rigidly deterministic, and its path is established once the initial interval is given. However, the word underlines the fact that there is in general absolutely no control on which of the (many) local minima the algorithm will ultimately be attracted to: very small changes in the initial interval may dramatically alter the outcome, and it is in general impossible to gauge beforehand if a given choice will yield a "good" local minimum (say, the actual global minimum x^*) or a "bad one". In other words, there always is the chance that the *local* algorithm will find a global solution, but there is no control on how likely this is to happen. Intuitively, "if there are few local-not-global minima and their attraction basins are small", there could be a "good chance" that the algorithm indeed finds the global one; but this is only a qualitative statement (unless it can be made precise in the very strong sense of §2.8). To further underline the total lack of control on the quality of the obtained result, let us mention that, at each given iteration, the algorithm has no way of knowing which phase it actually is in.

Hence, local algorithms have rather dramatic trade-offs w.r.t. global ones:

1. the complexity of local optimization is (dramatically) lower;
2. the returned local minimum can be (dramatically) worse than the global one.

While this has been described in details for just this particular case, the principle is general and will hold true throughout all these lecture notes. In plain words: all local minima "look the same", *comprised the global one.* This makes it finding *some* local optimum "a lot easier", but on the other hand *once a local minimum has been found, there is no way of telling whether or not it is the global one.* Finding the global minimum, and *proving* it is such, is a completely different problem that these lecture notes will not tackle in any detail, save for a fleeting glimpse—on entirely intuitive terms—in §2.8. The issue can, again, be linked to the *difficulty of estimating the value of f_**; indeed, if one knew that, it would be easy to ascertain whether or not the found local minima $\bar{x}$ is a global one. While one could then immediately stop in case this were true, it would not be entirely obvious what to do if this turned out not to be the case; again, this cannot be discussed in any detail in these lecture notes, save for the brief hints in §2.8. What matters in our discussion is that, since f_* is generally unknown, local and global optimization are two very different processes.

Due to the lack of any guarantee on the quality of the returned solution, one may therefore deem local algorithms to be scarcely useful. This is not the case, for a number of reasons:

- in some "lucky" cases the two processes actually are the same (cf. again §2.8), and in several applications it is in fact possible to build the optimization problem in order ensure that this actually happens;
- even if the problem cannot be guaranteed to be solved exactly, it may be that in practice the quality of the solutions returned by a local algorithm is most often satisfactory: this is in fact typically the case for the applications of interest in these lecture notes, where the functions to be minimised are *not adversarial*:
- when the size of the problem grows—as it invariably does in the applications of interest in these lecture notes—the complexity of global optimization becomes so dramatically high (cf. §**??**) that attempting it is completely out of question: running a local optimization approach, and having a chance (even if possibly slim) at getting a solution of reasonably quality is simply the only option.

For all these reasons, these lecture notes will almost entirely focus on *local algorithms*. The issue of the quality of the obtained solution will simply be "showed under the rug and kept there": in many applications it will anyway be satisfactory (even optimal), and in case it is not, nothing can be done about it. Of course, many things *can* in fact be done, but these will not be discussed here: the curious readers can refer to the many other available courses in optimization [18].

2.5 Towards faster local optimization algorithms

The discussion of the previous sections can perhaps be summarised by the following (somewhat obvious) tagline: *you can get much more efficiently where you want to if you know which way to go*. In our case, "where we want to go" is a local minimum, and—assuming unimodality—our "directions" are: "go right if the function is decreasing, go left if the function is increasing". However, to gauge where the function is increasing or decreasing we need its value in no less than four points. It would clearly be interesting to be able to obtain this information more efficiently, and this is of course possible with a very well-known mechanism.

The idea is quite simple: knowing $f(x)$ for some x, one would take some z "very close" to x and compute $f(z) - f(x)$; if this is positive the function "looks increasing" in $[x, z]$, otherwise it "looks decreasing". However, we know that functions "can change very rapidly", hence if the interval is "large" this information may not be very significant because the function may "change mode" even many times inside the interval; hence, we want the interval to be "very small". We could therefore consider $\lim_{h\to 0} f(x+h) - f(x)$, but this limit should always be 0 if f is continuous, which is a baseline assumption. The solution is to rather consider:

Definition 2.1. The (first) *derivative* of f at $x \in \mathbb{R}$ is the limit of the *incremental ratio* of f between x and $x + h$ when $h \to 0$, i.e.,

$$f'(x) = \lim_{h\to 0}(f(x+h) - f(x)) / h , \tag{2.2}$$

provided it *exists and it is finite.*

It is easy to see that the incremental ratio is the slope of the linear function interpolating f at x and $x + h$, i.e., passing by $(x, f(x))$ and $(x+h, f(x+h))$. Thus, $f'(x)$—if it exists—can be interpreted as the *slope of the line tangent to the graph of f at the point $(x, f(x))$*: this is the *first-order model of f at x*

$$L_x(z) = f'(x)(z - x) + f(x) , \tag{2.3}$$

i.e., the "best possible linear approximation of f close to x". As such, "the derivative is our GPS": if it is negative then f is decreasing in x, if it is positive then f is increasing in x.

This can be easily verified for our simple functions of §1.2. Indeed, for $f(x) = bx$,

$$(b(x+h) - bx) / h = b ,$$

i.e., the derivative is the constant function $f'(x) = b$: the function has always the same slope (always increasing if $b > 0$, always decreasing if $b < 0$), hence it has no minimum. For the (homogeneous) quadratic function $f(x) = ax^2$,

$$f'(x) = \lim_{h\to 0}[(a(x+h)^2 - ax^2) / h = a(h^2 + 2hx) / h = a(h + 2x)] = 2ax .$$

Hence, if $a > 0$ the derivative is negative when $x < 0$ and positive if $x > 0$, meaning that the minimum is in $x = 0$ (where $f'(x) = 0$). Thus, at least in this case—not surprising, since the function is clearly unimodal on all $\mathbb{R}$—the derivative is a reliable indication about "which way the minimum is". Indeed, in this example it is also a reliable indication about "where exactly the minimum is": $f'(x) = 0$ in the minimum. However, if $a < 0$ the sign swaps and $x = 0$ is the maximum, as we know well; thus, the derivative being zero is a not necessarily a reliable indicator per se, and it needs to be combined with othe information.

Of course, none of this would be any helpful if derivatives were "hard to compute". However, if f is a "reasonably simple function", then its derivative also is. That is, if f is an algebraic function, obtained combining the standard "basic" functions (polynomials, exponentials, trigonometric, . . .) by the standard algebraic operations (sum, product, composition, . . .), then the very same expression tree used to represent the computation of f can be tackled to efficiently compute f'. This is well, known, and briefly recalled in §A.7 and the references therein. In fact, there are efficient ways to compute derivatives even for "rather complicated functions", as we shall see. In particular, if *actual computer code* is available that implements the computation of $f(x)$, then well-developed, available and efficient *automatic differentiation tools* exist [17] that can automatically—under some assumptions on the original one—*produce code that computes $f'(x)$*. For this reason, in all these lecture notes we will assume that *derivatives are generally available for about the same cost as computing the function, and that doing so is none of our business*. This is a simplification: there are indeed cases, among which some regarding the applications of interest for these lecture notes, where the computation of derivatives is highly nontrivial and the subject of ongoing research. We will not have the possibility of going in any of these details, though, so the general stance remains that "derivatives are there".

Of course, computing $f'(x)$ can only be considered if it *exists at all*, which means that the limit in (2.2) must *exists and be finite*. In this case we say that f is *differentiable at x*. If the function $f' : \mathbb{R} \rightarrow \mathbb{R}$ is well-defined and continuous over all points of a set $X \subset \mathbb{R}$ we say that f is *continuously differentiable over X*. Given our reluctance to specify domains and suchlike, we will particularly cherish functions that are continuously differentiable over the whole of $\mathbb{R}$, which we will denote by "$f \in C^1$". The notation is appropriate, since *if f is differentiable at x then f is continuous at x*; that is, $f \in C^1 \Longrightarrow f \in C^0$.

Exercise 2.4. Prove this result. [**solution**]

Thus, if f is not continuous at x is cannot be differentiable there. However, there are functions that are continuous, but still not differentiable. A convenient concept is that of the *left and right* derivatives

$$f'_-(x) = \lim_{h \rightarrow 0_-}(f(x+h) - f(x))/h \quad , \quad f'_+(x) = \lim_{h \rightarrow 0_+}(f(x+h) - f(x))/h \; ;$$

f is differentiable at x if and only if both the left and right derivative at x exist, are finite, and $f'_-(x) = f'_+(x)$. Our preferred example of nondifferentiable function is $f(x) = |x| = \max\{x, -x\}$, which is continuous everywhere: yet, it is clearly not differentiable at 0 (while it is everywhere else), since it is immediate to see that $f'_-(0) = -1 \neq 1 = f'_0(0)$. In plain words, the function "looks linear both left and right of 0, but with two different slopes". In fact, it is common to say that "the derivative of the absolute value is the *sign*() function", which is not continuous; we will see later one some more precise statement that partly justify (but largely confutes) this definition. This is by no means the "worst case", as functions can be found whose left and right derivative at some x are "as much different as possible".

Exercise 2.5. Exhibit such a function. [**solution**]

The example $f(x) = |x|$ clearly illustrates what lack of differentiability means: basically, "there is a kink in the graph of f". That is, the graph is not a "smooth line", with a well-defined tangent at each point; rather, in some point the graph "abruptly changes direction". This is why *nondifferentiable* functions are also dubbed *nonsmooth*.

Before moving decidedly towards describing how derivatives can be used for optimization we need to introduce the concept of *higher-order derivatives*; in particular, $f''(x)$, i.e., the *second derivative* of f. Fortunately, this is "trivial": $f'' = (f')'$, i.e., just the derivative of the derivative. In this sense, f' is also called the *first derivative*. Since one is just applying the same concept again, there is not much that needs be said: f'' is typically "easy to compute" by applying to f' the same process used to produce f' out of f, and the sign of $f''(x)$ tells whether f' is increasing or decreasing in x. This information will turn out to be useful for our purposes, first in terms of *optimality conditions*—identifying which x are local minima—and then, as we shall see, to construct actual algorithms. Of course, there is nothing preventing to play the same game an arbitrary number of times: for instance, the *third derivative* $f'''(x)$ is just $(f'')'$, i.e., the first derivative of the second derivativo, or alternatively $(f')''$, i.e., the second derivative of the first derivative. In practice, at least for the approaches described in these lecture notes, no derivatives higher than second-order ones will ever be considered. However, occasionally some mention of third derivatives (but not any higher) will be done in some theoretical results. Let us just finish by commenting that, by the result of Exercise 2.4, continuity of higher-order derivatives implies continuity of all lower-order ones and of the function. That is, for C^2 the class of functions with continuous second-order derivatives, one clearly has $f \in C^2 \Longrightarrow f \in C^1 \Longrightarrow f \in C^0$. Of course, for C^3 the class of functions with continuous third-order derivatives, $f \in C^3 \Longrightarrow f \in C^2 \Longrightarrow f \in C^1 \Longrightarrow f \in C^0$. The class C^3 is the "strictest" possible one that will ever be mentioned in these lecture notes, and even there only in a few theoretical results. In general one is more than happy with C^2 functions, and C^1 ones (not necessarily C^2) are also common in some applications. Indeed, these lecture notes will even consider non-C^1, i.e., nonsmooth, functions, since these are required for some important applications.

The last important remark that needs be done concerns the relationship between Lipschitz continuity and the derivative. The result is quite intuitive: L-c implies that the slope of the function cannot be too steep (either positive or negative), and therefore *boundedness of the derivative*. Formally: $f \in C^1$ is *globally L-c* on some (open) set X, with a given constant $L > 0$, implies that $|f'(x)| \leq L$ for all $x \in X$, and the converse also holds.

Exercise 2.6. Prove the last statement. [**solution**]

It is therefore "reasonably easy" to construct functions that are not globally L-c on the whole of $\mathbb{R}$ by ensuring that the derivative is not bounded: the simple $f(x) = ax^2$, with $f'(x) = 2ax$, immediately provides a clear example. Yet, L-c is easy to achieve on *bounded sets*, in that it just corresponds to continuity of the derivatives. This is due to the *extreme value theorem* [11, Th. 2.2.9], stating that every function that is continuous on a bounded set achieves maximum and minimum there; to keep notation simple, for $f \in C^0$ and $X = [\underline{x}, \overline{x}]$ with

$-\infty < \underline{x} \leq \overline{x} < +\infty$

$$\max\{\, f(\,x\,) \;:\; x \in X \,\} < \infty \quad \text{and} \quad \min\{\, f(\,x\,) \;:\; x \in X \,\} > -\infty \;.$$

As an immediate consequence, if $f \in C^1$ on such a finite X then it is f globally L-c on X, since f' is bounded there. By the same token (and this will be useful), $f \in C^2$ (on a finite X) means that f' is L-c on X, and so on. However, given all the previous description it is not difficult to construct examples of C^0 functions that are *not* L-c on some finite $X = [\,\underline{x}\,,\,\overline{x}\,]$.

Exercise 2.7. Exhibit such a function. [**solution**]

We are now in the position of discussing *optimality conditions* for univariate problems based on the derivatives. We introduce the discussion with an example.

Example 2.1. Local minima, maxima and saddle points of a polynomial

We consider the 10-th degree polynomial

$$f(\,x\,) = \tfrac{91}{30}x^2 - \tfrac{19}{6}x^3 - \tfrac{54}{25}x^4 + \tfrac{93}{23}x^5 - \tfrac{23}{36}x^6 - \tfrac{121}{93}x^7 + \tfrac{72}{91}x^8 - \tfrac{13}{74}x^9 + \tfrac{9}{640}x^{10} \;.$$

Using standard results, it is easy to show that

$$\begin{aligned} f'(\,x\,) &= \tfrac{91}{15}x - \tfrac{19}{2}x^2 - \tfrac{216}{25}x^3 + \tfrac{465}{23}x^4 - \tfrac{23}{6}x^5 - \tfrac{847}{93}x^6 + \tfrac{576}{91}x^7 - \tfrac{117}{74}x^8 + \tfrac{9}{64}x^9 \\ f''(\,x\,) &= \tfrac{91}{15} - 19x - \tfrac{648}{25}x^2 + \tfrac{1860}{23}x^3 - \tfrac{115}{6}x^4 - \tfrac{1694}{31}x^5 + \tfrac{576}{13}x^6 - \tfrac{468}{37}x^7 - \tfrac{81}{64}x^8 \end{aligned}$$

The function and its first and second derivative are shown in Figure 2.5. In particular, the six roots of the derivative, x^i, $i = 1, \ldots, 6$, are shown (ordered in increasing sense). Since f' is a 9-th degree polynomial, the roots are 9, but the others are complex (in fact I'm slightly cheating, even x^4 is actually a pair of complex roots with real part ≈ 0.8881, but it is difficult to construct good examples and this one is good enough, so please pretend). The function shows the following:

- x^1 corresponds to a (shallow) local minimum of f. This corresponds to the fact that $f''(\,x^1\,) > 0$. That is, f' is increasing in x^1: it is negative before x^1 and positive after x^1, meaning that f is decreasing before x^1 and increasing after it.
- Symmetrically, x^2 corresponds to a local maximum of f; albeit "shallow", it is in fact the local maximum with highest value (but not a global maximum since the function is unbounded above as $|\,x\,| \to \infty$). This corresponds to the fact that $f''(\,x^1\,) < 0$. That is, f' is decreasing in x^1: it is positive before x^1 and negative after x^1, meaning that f is increasing before x^1 and decreasing after it.
- x^3 is a local, and global, minimum of f: again this corresponds to the fact that $f''(\,x^3\,) > 0$.
- x^4 (disregarding the minor cheating) is a "strange" point, being neither a local minimum nor a local maximum. This corresponds to the fact that $f''(\,x^4\,) = 0$. In fact, x^4 is a local minimum of f': f' is positive *both before and after it.* That is, "f is increasing, it fractionally stops increasing in x^4 but then immediately resumes increasing". This is called a *saddle point* of the function.
- x^5 is a local maximum of f; albeit "not too shallow", it is not the best local maximum. This corresponds to the fact that $f''(\,x^5\,) < 0$.
- x^6 is a local not global minimum of f; this corresponds to the fact that $f''(\,x^6\,) > 0$.

Thus, the example shows that the roots of f', and the corresponding values of f'', identify local minima and maxima. However, x^4 also shows that this is only "authoritative" if $f''(\,x\,) \neq 0$: for null values of the second derivative the point may be a local minimum, a local maximum, or none of the two (a saddle point). It would be possible to further refine the analysis by looking up the values of even higher-order derivatives which ultimately would give a definite answer, but we will carefully avoid doing it. Anyway, as we will discuss, "even looking at the second derivative is too much in practice".

A further, rather informal but maybe useful, description is the following: *f' gives the slope of the graph of f and f'' gives its curvature.* That is, $f' > 0$ says that the graph is "trending upwards" and $f' < 0$ that is "trending downwards". f' characterise the "rate of change" in the function value. Indeed, the roots of f' are called *stationary point* to indicate that (locally) "the function is not changing"; the graph (if zoomed very close to the point) "looks flat". Yet, the rate of change in itself would describe a straight line (think a linear function with its constant derivative, or the first-order model). Since f'' characterise the rate of change of f', it describes how the rate of change of f is (locally) itself changing, i.e., "how the slope of the line is changing", i.e., "how curved the function is". A positive f'' implies that the slope is increasing, i.e., a "convex" curvature, while a negative f'' implies a "concave" one. In fact, the roots of f''—typically, maxima or minima of f'—identify points where "the curvature briefly stops changing" and the graph is "locally linear"; this often happens when "the function changes curvature from convex to concave or vice-versa" (for instance, the root of f'' between x^2 and x^3), although in fact the slope may just briefly stop changing but then immediately resume its previous mode (cf. x^4). Albeit perhaps too pedestrian, this description may turn out to be informative when multivariate functions will be considered.

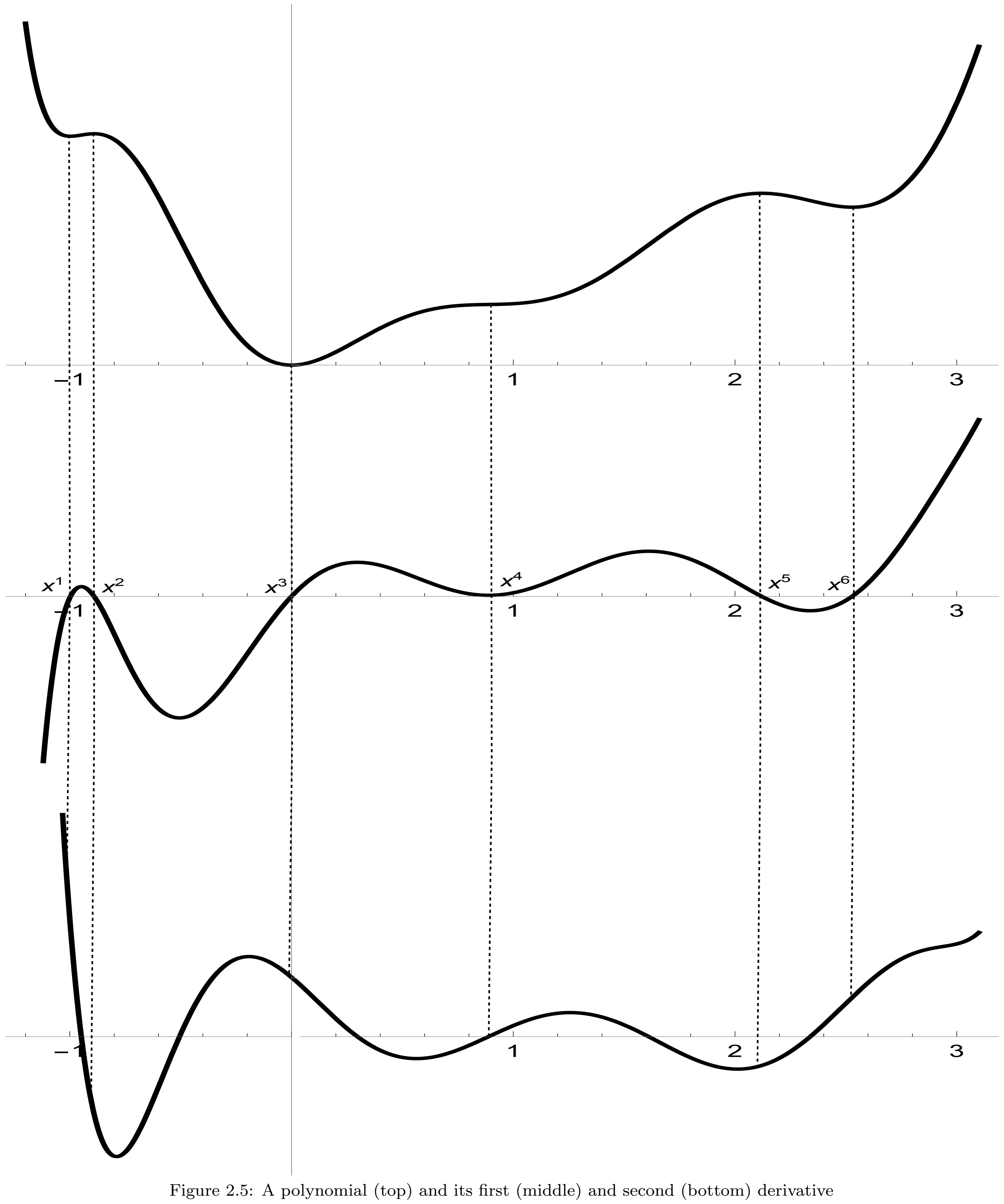


Figure 2.5: A polynomial (top) and its first (middle) and second (bottom) derivative

The example illustrates, among others, the following crucial property (that we won't formally prove as this will be done later on in a more general context): *if $f'(x) \neq 0$, then x cannot be a local minimum* (or maximum). In other words, *if we want to find a local minimum we need to find a root of the derivative.* However, *not all roots of the derivative are local minima*: some are local maxima, some other saddle points.

A final important observation is the following: *even if we find a bona-fide local minimum ($f'(x) = 0$, $f''(x) > 0$), there is no way of saying whether or not it is the global minimum* (even assuming a global minimum is known to exist). In plain words, "all local minima looks equal". It is possible to efficiently find (approximate) roots of f', that may (or may not) le local minima, but ascertain whether or not they are *global* minima is *an entirely different process.* This will almost completely ignored in these lecture notes, save for some very brief hints at the end of this Chapter (§2.8). Of course, this is an issue *if there are local-not-global minima* at all. This may not happen, with a (globally) unimodal function providing convenient example. However, for reasons to be discussed later on, unimodality is not the most convenient concept to use. Rather, the development in this section allows us to define (in a stricter way than necessary, but this will also be discussed later on) "the class of functions we really love to minimise".

Definition 2.2. $f \in C^2$ is *convex* on $X \subseteq \mathbb{R}$ if $f''(x) \geq 0$ for all $x \in X$.

As we will see this is only a very special case of a much more general definition, but it will do for our purposes here. Needless to say, $f(x) = ax^2$ with $a \geq 0$ is convex. It is immediate to see (again, no formal proof as these will be deferred to the more general treatment later on) that if a convex function has a local minimum—this is not given: a linear function is convex—then it is unimodal on all $\mathbb{R}$, which means that *any local minimum is the global minimum.* Besides, *x is a local minimum if and only if $f'(x) = 0$*, i.e., *global minima are the same as stationary points.* Thus, convex functions are "perfect to minimise": not only "you always know the way towards the global minimum", but also "as soon as you are close to a local minimum you know it because $f' \approx 0$". This is drastically different from the general situation. Fortunately, as we shell see, "there are many convex functions": a fair number of basic functions are convex, and a number of operations between functions preserve convexity. Thus, *convex minimization* can happen in a reasonable number of cases, which justifies why there is plenty of theory [2], and software [19], for it. Indeed, if one has some choice about the function to be minimised—which happens in some of the applications of interest for these lecture notes, whereby one is constructing *models*—then all other things being equal one should most definitely choose convexity. Even if the convex model is somehow less than ideal for the application, the optimization problem being dramatically easier could very well tilt the balance towards a convex function being chosen anyway.

Motivated by convexity, we now take a fundamental stance that we will keep throughout all these lecture notes (save very briefly in §2.8): *our aim is to find any stationary point, i.e., root of f'.* This is mostly because efficient algorithms exist for doing so. Then, if f is convex this is a guaranteed global minimum. Otherwise, it may hopefully be a local minimum, and therefore it still has a chance of being a global one; this indeed happens frequently is some prominent applications of interest for these lecture notes. If we are "unlucky", we will just put up with it: at least the "unsatisfactory" solution has been obtained quickly, and if there is still time to spare some other mechanism (which will not be discussed) can be put in place to try to find better ones. That is, *efficient local optimization is the cornerstone of global optimization*, although for us the two "coincide" (in the sense that we only consider global optimization when it is equivalent to local).

2.6 Dichotomic Search

We have then reduced our optimization task—finding the minimum of f—to an essentially algebraic one: *find one of the roots of f'* (hoping there will only be one, and it will correspond to the global minimum). This first and foremost requires $f \in C^1$: the perennial counterexample $f(x) = |x|$ clearly shows that roots may not exist if f' is discontinuous (exactly where the minimum lies). It is obvious that in very simple cases this can actually be done by a closed formula. For instance, for a linear function $f(x) = bx\ [+c]$, $f'(x) = b = 0$ is only possible if $b = 0$, reproducing §1.2.1. For the quadratic non-homogeneous $f(x) = ax^2 + bx\ [+c]$, $f'(x) = 2ax + b = 0$ yields $x = -b / 2a$, clearly the unique minimum if $a > 0$ and maximum if $a < 0$, reproducing §1.2.3. This generalises to cubic and quartic polynomials, whose derivatives have respectively degree 2 and 3 and for which therefore root have a (somewhat involved in the cubic case [51]) closed formula; but almost nowhere else. Algorithms for polynomial factorization exist that can work efficiently up to quite high degrees [55], although they are far from trivial. However, there is little hope for most algebraic functions containing trascendental and/or trigonometric functions. One therefore needs an *algorithm* for solving the *nonlinear* equation $f'(x) = 0$.

Fortunately there are efficient algorithms for finding the root of a continuous function over an interval *in which a root is guaranteed to exist.* This is down to the well-known *intermediate value theorem* [11, Th. 2.2.10] (basically, the intuitive definition of continuous function) that guarantees the following: if $f'(\underline{x}) < 0$ and $f'(\overline{x}) > 0$ then there exists $x \in [\underline{x}, \overline{x}]$ such that $f'(x) = 0$. Assuming one knows beforehand an interval with the right property (the assumption will be challenged soon), the *theorem breeds an algorithm*, since it is then trivial to restrict the interval by just computing $f'(x)$ in any point inside the interval. This is the basic *dichotomic search* approach:

Algorithm 2.3. Dichotomic Search

```
procedure $x = DS\ (f, \underline{x}, \overline{x}, \varepsilon)$
  do forever            // invariant: $f'(\underline{x}) < -\varepsilon$, $f'(\overline{x}) > \varepsilon$
      $x \leftarrow$ in_middle_of( $\underline{x}, \overline{x}$ ); compute $f'(x)$;
      if( $|f'(x)| \leq \varepsilon$ ) then break;
      if( $f'(x) < 0$ )   then $\underline{x} \leftarrow x$; else $\overline{x} \leftarrow x$;
```

The trivial (and sensible) choice of choosing the middle point, i.e.,

$$\text{in\_middle\_of}(\, \underline{x} \,,\, \overline{x} \,)\{\, \textbf{return}(\, (\, \overline{x} + \underline{x} \,) \,/\, 2 \,) \,\} \;,$$

clearly gives the standard $\overline{x}^i - \underline{x}^i = Dr^i$ with $r = 0.5 < 0.618$; that is, assuming that the cost of computing $f'(\, x \,)$ is close to that of computing $f(\, x \,)$ (a nontrivial, but not too wild, assumption) the algorithm should be faster than the Golden Ratio Search. The analysis then follows a well-beaten path: $D^0 r^i \leq \delta$ surely happens when $i \geq \log(\, D \,/\, \delta \,) \,/\, \log(\, 1 \,/\, r \,)$, i.e., $i \approx 3.32 \log(\, D \,/\, \delta \,)$. Now, we know that $f'(\, x \,) = 0$ for some $x \in [\, \underline{x}^i \,,\, \overline{x}^i \,]$; *assuming f' is L-c with constant L*, then

$$|\, 0 - f'(\, \underline{x}^i \,) \,| \leq L |\, x - \underline{x}^i \,| \quad \text{and} \quad |\, f'(\, \overline{x}^i \,) - 0 \,| \leq L |\, \overline{x}^i - x \,| \;.$$

This means that

$$\min\{\, f'(\, \underline{x}^i \,) \,,\, f'(\, \overline{x}^i \,) \,\} \leq L \min\{\, |\, x - \underline{x}^i \,| \,,\, |\, x - \underline{x}^i \,| \,\} \leq L\delta \,/\, 2$$

(in the worst case, x is equidistant from the extremes); thus choosing $\delta = 2\varepsilon \,/\, L$ ensures that the stopping criterion have to be satisfied. All in all, the algorithm will surely terminate after at most $3.32 \log(\, LD \,/\, 2\varepsilon \,)$ iterations. Whether this is better or worse than the Golden Ratio Search also depends on how the Lipschitz constant of f compares with that of f'. In fact, requiring f' to be L-c (i.e., f to be L-smooth) in the first place is nontrivial, but as we shall see is very common when results about the speed of convergence of derivative-based algorithms need be proven. This is intuitive: the "crucial function" is no longer really f, but rather f'. Hence, it is the derivative that need "not to change too fast" (which *can* happen, cf. Exercise 2.7) for being able to exercise some control.

The whole approach hinges on having the invariant ($f'(\, \underline{x} \,) < 0$ and $f'(\, \overline{x} \,) > 0$) satisfied at the beginning. Of course, even choosing a random x one of the two conditions will surely be satisfied, so the issue is ensuring the other. We will see later on that in many cases $f'(\, \underline{x} \,) < 0$ will hold, so we deal with the problem of finding $\overline{x} > \underline{x}$ such that $f'(\, \overline{x} \,) > 0$; however, the opposite case would be symmetric. In practice this is hardly a significant problem, since the following very rough solution usually works:

Algorithm 2.4. Right extreme search

```
$\Delta x \leftarrow 1$;                                  // or whatever value $> 0$
while( $f'(\overline{x}) \leq -\varepsilon$ ) do
    $\overline{x} \leftarrow \overline{x} + \Delta x$; $\Delta x \leftarrow 2\Delta x$;           // or whatever factor $> 1$
```

The approach just explores values on the right of $\underline{x}$ "at rapidly increasing steps". The expectation is that if a minimum is sought for on the right of $\underline{x}$, where f is decreasing, then "sooner or later the function will have to be increasing", i.e., its derivative will turn positive. A formal way to ensure the property is to require that f is *coercive*, i.e., $\lim_{|\, x \,| \to \infty} f(\, x \,) = \infty$; in plain words, "when going away from the origin, sooner or later the function grows unboundedly". Note that, for instance, both $|\, x \,|$ and x^2 are coercive. This is relevant in that in many applications of interest for these lecture notes these functions (or multivariate generalizations thereof) are explicitly added to the "original" objective, which (among other things) may have the nice side-effect to making it coercive. Thus, in the following we will skip all finer analysis and just pretend that Algorithm 2.4, if at all needed, will work. Of course, counterexamples can be carefully constructed so that this fails.

Exercise 2.8. Construct an example where $\overline{x} > \underline{x}$ such that $f'(\, \overline{x} \,) > 0$ exists but is not found by Algorithm 2.4 [**solution**]

Before abandoning this issue for good, it is worth remarking on the obvious case where Algorithm 2.4 cannot but fail: that where f has no local minimum on the right of $\underline{x}$ since it is always decreasing there, i.e., the derivative is always negative. A particularly clear case is that where the function is unbounded below on the right of $\underline{x}$, and $\{\, f(\overline{x}^i \,) \,\} \to -\infty$. This case is hardly relevant in the applications of interest for these lecture notes (cf. the previous comment about purposely adding a coercive term to the original objective), but it does underline the difficulty of dealing with unbounded problems: how does one *prove unboundedness*? Basically, what one would need is a "finite $-\infty$": a large negative enough value M such that if one finds x with $f(\, x \,) \leq M$, then the problem can be safely declared unbounded below. This is actually possible in some cases, but these cannot be discussed here.

2.6.1 Improving the dichotomic search by interpolation

Although the dichotomic is already "fast", it is possible to make it significantly more efficient in practice. The observation is that choosing the next iterate x "right in the middle" of the current interval $[\, \underline{x} \,,\, \overline{x} \,]$ is the reasonable choice to optimize the *worst-case* behaviour of the algorithm, but not the *best-case* one. Indeed, if one were able to choose x so that $f'(\, x \,) = 0$ (or close enough), then the algorithm would stop in exactly one iteration. This suggests that trying to select x is "close to x_*" may lead to better performances; of course, the issue is then to try to predict where x_* actually lies.

A powerful general way to approach the issue is that of exploit available information on f to build a *model of the function*, i.e., a *simple* function g that approximates f and "for which the required information is easy to find"; in our case this means "whose minimum can readily be found". Then, one can hope that the minimum of g will be an approximation of that of f. This approach is particularly sensible in view of the fact that the algorithm has anyway to compute information about f by evaluating it: at least the values of the derivative $f'(\underline{x})$ and $f'(\overline{x})$ are known, and computing the values of the function $f(\underline{x})$ and $f(\overline{x})$ could obviously also be an option if the cost is limited. It is clear that, in its basic form, the algorithm "only partly uses" knowledge of $f'(\underline{x})$ and $f'(\overline{x})$ in that it only looks at their sign, not at their magnitude. Yet, clearly the magnitude may convey useful information. In fact, the function and derivative values can be used to construct a *model* of f, i.e., a "*simple*" function g that "somehow resembles f". This is a very important concept that we will see many other time during the course. Once the model is constructed it can be used to derive information, the typical one being *where the minimum of f would be if it were true that g was identical to f*, i.e., where the minimum of g lies. The obvious use of this information is to drive the selection of the next iterate, although, as we shall see, slightly more involved choices may be better.

Constructing a model of f given function (and derivative) values is *interpolation*, which requires choosing a form of the interpolating function. The simplest choice is that of a *polynomial*, which still requires to select a degree. A degree-k polynomial has $k+1$ coefficients, and each information we have yields a linear equation in the coefficients to impose that g agrees with f (g' with f') there; thus, the degree cannot be too high. On the other hand, a linear g ($k=1$) had no minimum. So, a reasonable value is $k=2$, i.e., $g(x) = ax^2 + bx + c$, for which the interpolation requires

$$\begin{cases} a\overline{x}^2 + b\overline{x} + c = f(\overline{x}) & , \quad 2a\overline{x} + b = f'(\overline{x}) \\ a\underline{x}^2 + b\underline{x} + c = f(\underline{x}) & , \quad 2a\underline{x} + b = f'(\underline{x}) \end{cases} .$$

This is an overdetermined system, hence some of the conditions need be dropped. A natural choice is to only keep the two ones on the right, corresponding to imposing that $f'(\overline{x}) = g'(\overline{x})$ and $f'(\underline{x}) = g'(\underline{x})$. Notably, c is not involved in those two equations, but obviously c does not anyway influence where the minimum of g is, hence this does not seem a significant loss. A bit of simple algebra then yields

$$a = \frac{f'(\overline{x}) - f'(\underline{x})}{2(\overline{x} - \underline{x})} \qquad , \qquad b = \frac{\overline{x}f'(\underline{x}) - \underline{x}f'(\overline{x})}{\overline{x} - \underline{x}} \; ;$$

these are well-defined values since $\overline{x} > \underline{x}$, and furthermore $a > 0$ since $f'(\overline{x}) > 0$ while $f'(\underline{x}) < 0$. Thus, $g'(x)$ is a (strictly) convex quadratic function, whose (unique) minimum solves $2ax + b = 0$, i.e.,

$$x = \frac{\underline{x}f'(\overline{x}) - \overline{x}f'(\underline{x})}{f'(\overline{x}) - f'(\underline{x})} \; . \tag{2.4}$$

This is called the *secant formula*, and–as one should geometrically expect–it is easy to prove that $\underline{x} < x < \overline{x}$. Thus it is possible to use x from (2.4) as the next iterate in the dichotomic search, which is known as the "method of false position".

Exercise 2.9. Prove that x is always in the middle between $\overline{x}$ and $\underline{x}$. **[solution]**

The process looks therefore pretty reasonable, and in fact it can be proven to (in a certain sense) improve upon the standard dichotomic search. In particular, [9, Theorem 2.4.1] proves the following:

Theorem 2.1. Let $f \in C^3$ be such that $f'(x_*) = 0$ and $f''(x_*) > 0$; then, there $\exists\, \delta > 0$ such that, however chosen $\underline{x} < \overline{x}$ in the interval $[\, x_* - \delta \,,\, x_* + \delta \,]$, the dichotomic search with quadratic interpolation started with $\underline{x}$ and $\overline{x}$ produces a sequence of iterates $\{ x^i \} \to x_*$, and the convergence is *superlinear* with $p = (1 + \sqrt{5}) / 2$ ($1 < p \approx 1.618 < 2$).

In plain words: *if the algorithm is started close enough to a local minimum*, then it eventually converges there superlinearly fast. We don't delve in the convergence proof since we will shortly see a similar but more significant one. The observation is that the radius δ of the interval around x_* in which the algorithm "works well" can be proven to be nonzero, but it is difficult to characterise in practice; basically, one has to run the algorithm and check how it is actually converging. Hence, in general there is no way of knowing whether or not one is starting inside the interval and will converge fast; but anyway, one does not really even know how many local minima there are in the initial interval and to which one the process will eventually land. Since the interval is (hopefully) shrinking, even if the initial interval is not "appropriate" it will (hopefully) eventually become such. Thus, the algorithm *eventually, in the "tail" of the process*, will be superlinear. Anyway, the definition of convergence always is concerned with the "tail" of the process, hence it is correct to say that the dichotomic search with quadratic interpolation is superlinear.

However, the issue of "when the tail actually starts" if obviously important. In practice, it is not very comforting that the algorithm will eventually be fast when it is close enough to some local minimum, if it will take an inordinate number of iterations to get there in the first place. One may surmise that this is not going to happen since the dichotomic search is at least linearly convergent to start with, but unfortunately this is no longer obviously true with quadratic interpolation. Indeed, linear convergence is predicated on choosing x^{i+1} "in the middle" of the interval; obviously, it keeps holding (with a worse convergence rate) provided that one ensures that x^{i+1} "always remains close enough to the middle". That is, for $D = \overline{x} - \underline{x}$, if one ensures that $x^{i+1} \in [\, \underline{x} + \sigma D \,,\, \overline{x} - \sigma D \,]$ for some $\sigma \leq 1\,/\,2$ then we know that the interval length will be reduced by at least a fraction of σ, which yields linear convergence with ratio $r = 1 - \sigma$. However, "pure" dichotomic search offers no such guarantee.

The issue is relevant in that it is conceptual: the point is that the efficiency of the approach is predicated on g being a "good model" of f, but there is no guarantee that this will in fact happen. In plain words: *never fully trust a model that you are not certain is "good"*, for otherwise you may be led to making "very bad choices". In this particular case, the model g can end up being "very skewed": it is plain to see from (2.4) that if $f'(\,\overline{x}\,) \gg -f'(\,\underline{x}\,)$ then x will end up being "very close" to $\underline{x}$, and, symmetrically, if $f'(\,\overline{x}\,) \ll -f'(\,\underline{x}\,)$ then x will end up being "very close" to $\overline{x}$. This *may* be a bad choice since one of the two intervals is "very small"; if it regularly ends up being the selected one, the algorithm may converge rather slowly. This is not only a theoretical issue, as the behaviour can show up in practice, leading to an algorithm that has a rather inefficient *cruise phase*—the sequence of iterations from the start to when the right interval of the local minimum is finally reached—and that only becomes efficient in the *local phase* at the tail of the convergence process.

This is a common phenomenon in *model-based algorithm*, that can be faced in a general way: *never completely trust the model.* The details of how this is done vary, and we'll see a number of different occurrences; in this case, however, the solution is pretty easy. One just has to pick a *minimum guaranteed decrease* $\sigma \leq 1\,/\,2$ as a *safeguard*, and then choose the next iterate as

$$x \leftarrow \max\{\, \underline{x} + \sigma(\overline{x} - \underline{x}) \,,\, \min\{\, \overline{x} - \sigma(\overline{x} - \underline{x}) \,,\, x' \,\}\,\}$$

with x' the value suggested by (2.4). This guarantees that the next iterate will never be "too close" to the extremes, and therefore linear convergence with ratio $r = 1 - \sigma$ during the cruise phase; eventually, superlinear convergence will hopefully kick-in at the tail.

While simple, the solution has an obvious issue: *how should one choose* σ? There is no clear answer to this question, as the choice involves navigating an *algorithmic trade-off.* The point is clearly illustrated by considering the extreme choice $\sigma = 1\,/\,2$. In this case, quadratic interpolation is basically deactivated: the algorithm will converge linearly with rate $r = 1\,/\,2$, even when "very close" to x_*. As σ decreases, the quadratic interpolation has a larger interval in which to choose the iterate, and therefore a better chance to actually drive the algorithm to faster convergence; but, of course, the other extreme $\sigma = 0$ incurs the risk of a rather slow cruise phase. This is a simple but poignant example of an *algorithmic parameter* that need be *tuned* to achieve the desired trade-off; in this case, between the algorithm being "robust" during the cruise phase and "fast" during the local one. We will see that algorithms can have several parameters, whose tuning can be highly nontrivial, especially in that it depends on the need of the specific application that drives the optimization process. For our case: is it more important to guarantee the worst-case behaviour (even if a slow cruise phase may materialise only very infrequently) or to get faster local convergence (even if serious slowdowns may frequently show up in the cruise phase)? Only considering the details of the actual application this can be answered. Fortunately, not all algorithmic parameters are particularly tricky to tune; in some cases, "default" values that tend to work well in almost all the cases can be found experimentally (and in the literature). However, finding the right value of other parameters may be nontrivial, and failing to do so can have a very dramatic impact on the algorithm performances. We will return on the issue as appropriate.

There clearly is another approach to the issue, though: *if the model is bad, make a better model* (vaguely reminiscent of the classical "if life gives you lemons, make a lemonade" of Portal 2 memory). In fact, this could be possible since in (2.4) we only used a part of the available information; arguably, by using more one could obtain a better model, that therefore may lead to a better behaviour (although it must never be forgotten that a model is, by definition, always going to be somehow inaccurate). There could be several ways for doing this even in our simple case. For instance, one could stick with $k = 2$ and choose any three of the four conditions (except as already done) to get different quadratic models; it is not really clear, though, why these should be significantly more accurate than the one we presented. A different approach to quadratic fitting can be found, e.g., in [1, p. 361]. Perhaps more reasonable is to move to $k = 3$, i.e., *cubic* fitting, since this entails finding four coefficients and therefore perfectly fits with the four conditions we can impose. Crucially, the derivative of a cubic polynomial is a quadratic one, hence its zeroes can be easily found; thus, it is possible to characterise where the minimum of the model in the interval lies. While not conceptually difficult, the approach is rather tedious to write down, analyse and implement; thus we will give it a wide berth, since the details can anyway be found, e.g., in [9, § 2.4.2] and [8, p. 57]. Cubic interpolation is generally considered a good practical approach which can lead to even faster (quadratic) convergence; but rather than insisting on it we move to a more conceptually interesting idea that we will actually prove having (local) quadratic convergence.

2.7 Newton's method

In retrospective, all we have seen so far can be read as different ways to construct models of f that drive the algorithmic choices. Basically, the point is that we need to have information about where a local minima lies, and this requires forming some "quadratic-like model", i.e., one where there is some information about "where the function decreases and where it starts increasing again". This is "complex" information, and in fact we have seen it extracted from:

- function values at *four* different points;
- function *and derivative* values at *two* different points.

This could be summarised as "more derivatives, less points", suggesting that one could try a further step: considering *second* derivatives. This is in fact possible, and a very powerful mechanism.

The crucial idea is the generalization of the first-order model (2.3) to the *second-order model*

$$Q_x(z) = f(x) + f'(x)(z-x) + \tfrac{1}{2}f''(x)(z-x)^2 = Q_x(z) + \tfrac{1}{2}f''(x)(z-x)^2 \ , \tag{2.5}$$

i.e., the first-order one plus a second-order term using the second derivative. Taylor's theorem [10, Lecture 4] [11, §2.5] basically state that a higher-order model is "more accurate"—*close to* x—than a lower-order one, in a specific sense that will be discussed in more details when it will be needed. That is, Q_x can be expected to be a better model than L_x, at least close to x. Furthermore, Q_x inherently possesses the *curvature* that had to be somewhat haphazardly extrapolated from the data in the previous cases. This means that choosing the next iterate can be done with the simple formula

$$x^{i+1} = x^i - f'(x^i) \,/\, f''(x^i) \ , \tag{2.6}$$

where x^{i+1} is just the minimum of Q_{x^i} computed with the simple tools of §1.2.3 and elementary algebra.

Exercise 2.10. Prove *Newton's step* (2.6). **[solution]**

This immediately gives rise to the deceivingly *Newton's algorithm* as illustrated below; note that, unlike all previous ones, the algorithm does not work with any interval, and therefore it only needs one point to start.

Algorithm 2.5. Newton's method

```
**procedure** $x = NM\ (f, x, \varepsilon)$
  **while**( $|f'(x)| > \varepsilon$ ) **do**
    $x \leftarrow x - f'(x) / f''(x)$;
```

The apparent simplicity of the algorithm hides a number of non-trivial points to discuss. First of all, *it is not at all a minimization algorithm*: (2.6) identifies the point x^{i+1} such that $L'_{x^i}(x^{i+1}) = 0$, but there is no guarantee that this is a minimum. Started close to a local maximum, the algorithm can happily converge there. The issue is that Newton's method is in fact a method for solving *non-linear equations*. The crucial observation is that *the derivative of the second-order model is the first-order model of the derivative*: for $f'(x)$, the first-order model reads

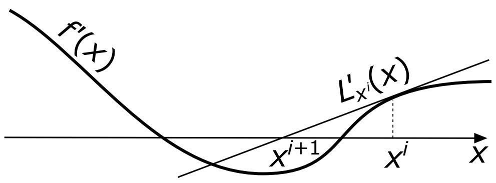


Figure 2.6: Illustration of the tangent method

$$L'_x(z) = f'(x^i) + (f')'(x)(x-z) = (Q_x)' \ .$$

Thus, Newton's method can be interpreted as the *tangent method* applied to the solution of the nonlinear equation $f'(x) = 0$, where one just replaces the nonlinear function f' with its first-order model L'_x and then choses the next iterate as the (unique) root of the model; again, $L'_{x^i}(x^{i+1}) = 0$. The principle is illustrated in Figure 2.6. While the approach is sound and, under some conditions, can be extremely efficient, it is clear that it is *not able to distinguish a minimum from a maximum*, as the first derivative itself cannot. The other delicate aspect is that L'_{x^i} may not have any root at all: this happens if $f''(x^i) = 0$. In fact, in this case (2.6) is plainly ill-defined and the method simply breaks down. More in general, Newton's method may significantly suffer if iterates happen that generate "very small" values of $f''(x^i)$, unless the corresponding values of $f'(x)$

are not "suitably small" as well.

Yet, the method can be proven to be, under conditions, extremely efficient. While the proof can be found in the literature (e.g., [1, Theorem 8.2.3]) we reproduce it here to give a taste of how these kinds of results can be proven.

Theorem 2.2. Let $f \in C^3$ be such that $f'(x_*) = 0$ and $f''(x_*) \neq 0$; then, there exists $\delta > 0$ such that, however chosen $x^0 \in [x_* - \delta, x_* + \delta]$, Newton's method started at x^0 produces a sequence of iterates $\{x^i\} \to x_*$, and the convergence is *quadratic*.

The proof hinges on the fundamental results that describes "how accurate" first- and second-(and higher-)order models are, i.e., Taylor's theorem. The particular form required in this case is *second-order Taylor's formula* [11, Theorem 2.5.4] stating that however chosen z there exists $w \in [x, z]$ such that

$$f(z) = L_x(z) + f''(w)(z - x)^2 / 2 .$$

This says that "the second-order model would be *exact* at z if the second derivative—but not the first—were computed at some point w between z and x instead of at x". Note that the "magical" point w changes with z. However, the more interpretable version is

$$f(z) - L_x(z) = f''(w)(z - x)^2 / 2 , \tag{2.7}$$

which says "the error between the true value of f and that of the first-order model is bounded by a term that depends quadratically in $z - x$ and on the value of the second derivative at some point between z and x". This is a lesser-known version of the more usual result concerning how the error of the model in z depends on the distance $z - x$ (quadratically, in the case of the second-order one). We will see the use of the more common form later on; the current proof requires (2.7) since it makes it explicit that the coefficient of the $(z - x)^2$ term depends on the value of the second derivative "close" to x.

Proof. Starting from (2.6), we subtract x_* from both sides and use $f'(x_*) = 0$ to get

$$x^{i+1} - x_* = x^i - x_* + (f'(x_*) - f'(x^i)) / f''(x^i) .$$

Then, by simple algebra

$$x^{i+1} - x_* = [f'(x_*) - f'(x^i) - f''(x^i)(x_* - x^i)] / f''(x^i) .$$

It is plain to see in the rightmost term, at the numerator, the error between *the value of the derivative* in x_* (which is known to be zero) and the value of the *first-order model of the derivative* in x_*. We can therefore use (2.7) *applied to* f': there exists $w \in [x^i, x^*]$ such that

$$f'(x_*) - [f'(x^i) + f''(x^i)(x_* - x^i)] = f'''(w)(x_* - x^i)^2 / 2 .$$

Plugging this in the relation above yields

$$x^{i+1} - x_* = [f'''(w) / 2f''(x^i)](x^i - x_*)^2 ,$$

where one clearly sees that $x^{i+1} - x_*$ can be bounded in terms of $(x^i - x_*)^2$ as required by quadratic convergence. However, one also needs to bound the term in the middle, that depends on the second and third derivatives of f (for the latter, in some unknown point w between x^i and x_*).

In order to upper bound the ratio, we have to prove that the numerator is bounded above, and the denominator is bounded away from zero. That is, we want to prove that there exists three constants $k_1 < \infty$, $k_2 > 0$, and $\delta > 0$ such that

$$f'''(w) | \leq k_1 \text{ and } f''(x) | \geq k_2 \quad \text{for all} \quad x , w \in [x_* - \delta, x_* + \delta] .$$

For k_2, since $f''(x_*) \neq 0$ one has that $|f''(x_*)| > 0$. Hence, by continuity of f'' ($f \in C^3 \implies f \in C^2$) there exists an interval around x_* where f'' is always strictly positive, and in fact bounded away from 0. Formally speaking, take, e.g., $k_2 = |f''(x_*)| / 2$ [> 0]: there exists $\delta > 0$ such that

$$|2k_2 - |f''(x)|| \leq k_2 \implies |f''(x)| \geq k_2 \; \forall x \in [x_* - \delta, x_* + \delta] .$$

We can now find k_1: since f''' is continuous, also $|f'''|$ is. Hence, by the *extreme value theorem* [11, Theorem 2.2.9], the maximum of a continuous function on a compact set is finite:

$$k_1 = \max\{ |f'''(w)| : w \in [x_* - \delta, x_* + \delta] \} < \infty .$$

We now rewrite the fundamental relationship as

$$|x^{i+1} - x_*| \leq \frac{k_1}{2k_2}(x^i - x_*)^2 = \Big[\frac{k_1}{2k_2}|x^i - x_*|\Big]|x^i - x_*| \tag{2.8}$$

(we have used the obvious facts that $a \leq |a|$ and $a^2 = |a|^2$ however chosen $a \in \mathbb{R}$). The crucial point is that *for x^i close enough to x_**,

$$0 < \frac{k_1}{2k_2} |\, x^i - x_* \,| < 1 \; .$$

Hence, there is some $\delta > 0$ (possibly smaller than the one in the previous formulæ) such that, if $x^i \in [\, x_* - \delta \, , \, x_* + \delta \,]$, then $|\, x^{i+1} - x_* \,| < |\, x^i - x_* \,|$ and therefore $x^{i+1} \in [\, x_* - \delta \, , \, x_* + \delta \,]$ as well: once the sequence enters the interval it never leaves it, and $\{\, x^i \,\} \to x_*$. But then, (2.8) guarantees that the convergence is quadratic. □

Thus, Newton's method has a very efficient local phase, *provided it is started close enough to some local minima or maxima* x_*. Looking at the proof it is obvious that actually computing the $\delta > 0$ that guarantees convergence, and therefore quadratic convergence, is nontrivial. Furthermore, there is in principle no guarantee that the local phase is ever started, even less how fast this eventually happens. There are ways to *globalize* Newton's method to provide it with a cruise phase that eventually converges to some local minima or maxima, hopefully fast towards the end. We do not delve in these details since we'll see globalization of Newton's method later on in a more general and interesting setting. We just finish the section with a few numerical illustrations.

Example 2.2. Execution of Newton's method

Let us start with the fantastic case of $f(\,x\,) = \frac{1}{2}x^2$. Then, $f'(\,x\,) = x$ and $f''(\,x\,) = 1$. The Newton's step is

$$x - f'(\,x\,) \,/\, f''(\,x\,) = x - x \,/\, 1 = 0$$

however chosen x, and 0 is the exact global minimum. In this case, $Q_x = f$: the model is "perfect", hence minimizing the model immediately provides the true solution of the problem.

Of course, things can go worse. For $f(\,x\,) = \frac{1}{6}x^3$, the convergence theory does not work because $f'(\,x\,) = \frac{1}{2}x^2$ and $f''(\,x\,) = x$; hence, the only place where $f'(\,x\,) = 0$ is $x = 0$, where also $f''(\,x\,) = 0$. In fact, one has

$$x - f'(\,x\,) \,/\, f''(\,x\,) = x - \tfrac{1}{2}x^2 \,/\, x = \tfrac{1}{2}x \; .$$

It is therefore still true that $\{\, x^i \,\} \to x_* = 0$, but the convergence is linear (with rate $r = \frac{1}{2}$) rather than quadratic.

For more practical examples we turn to the polynomial $f(\,x\,)$ of Example 2.2. The following is the log of a by-the-book implementation of Newton's method on that f started from $x = -0.1$:

```
feval   rel gap         x              f(x)        f'(x)       f''(x)
  1    3.3243e-02  -1.00000000e-01  3.3243e-02  -6.9098e-01  7.6253e+00
  2    2.6968e-04  -9.38338883e-03  2.6968e-04  -5.7755e-02  6.2426e+00
  3    5.2563e-08  -1.31629096e-04  5.2563e-08  -7.9871e-04  6.0692e+00
  4    2.2300e-15  -2.71140416e-08  2.2300e-15  -1.6449e-07  6.0667e+00
```

Clearly, $x = -0.1$ is inside the interval predicted by the theorem: the negative exponent of the relative gap *doubles* at each iteration; this basically means that the number of right significant digits in the value of x doubles as well. It is clear that only a fixed number of iterations are required before the accuracy reaches the machine one, making quadratic convergence an exceptionally efficient process.

However, Newton's method is not a minimizaition process. In fact, started with $x = -0.8$ it runs as

```
feval   rel gap          x              f(x)        f'(x)        f''(x)
  1   -2.4841e-02  -8.00000000e-01  1.6162e+00  -9.7155e-01  -1.2192e+01
  2   -3.0160e-04  -8.79687906e-01  1.6568e+00  -9.2538e-02  -8.8770e+00
  3   -8.4012e-07  -8.90112321e-01  1.6573e+00  -4.7004e-03  -7.9529e+00
  4   -1.0608e-11  -8.90703358e-01  1.6573e+00  -1.6662e-05  -7.8964e+00
  5   -1.3398e-16  -8.90705468e-01  1.6573e+00  -2.1350e-10  -7.8962e+00
```

Here, the gap is computed w.r.t. the f-value at the *local maximum* $x_* \approx -8.907$ where the algorithm is happily converging: note $f'(x_*) \approx 0$, $f''(x_*) < 0$, and the consistently negative relative gap as the function uniformly increase. The algorithm steadfastly maintain its quadratic rate of convergence with which it indeed very efficiently finds a root of f', except this is not a minimum.

Besides, the unfettered division by $f''(x)$ is clearly risky. Ran with $x = 1$, the algorithm obtains

```
feval   rel gap          x              f(x)        f'(x)        f''(x)
  1    4.3978e-01   1.00000000e+00  4.3978e-01   9.2412e-02   1.1291e+00
  2    4.3511e-01   9.18152291e-01  4.3511e-01   3.2736e-02   3.1418e-01
  3    4.3144e-01   8.13956159e-01  4.3144e-01   5.6550e-02  -7.6251e-01
  4    4.3422e-01   8.88118273e-01  4.3422e-01   2.8010e-02   1.1920e-04
  5    7.3305e+21  -2.34102775e+02  7.3305e+21  -3.1148e+20   1.1912e+19
```

The issue reveals itself at iteration 4, where $x \approx 8.881$ is close to a *saddle point*: it has a reasonably small value of f', but a much smaller value of f''. Thus, in one iteration the algorithm "shots away" in a completely unrelated region of space, worsening the gap by 22 orders of magnitude, and it is basically incapable of getting back to a "decent" solution since.

2.8 A Fleeting Glimpse to Global Optimization

We have seen that local optimization can be performed very efficiently, albeit with a fundamental caveat: there is no guarantee on the quality of the local minima one finds (and, possibly, not even it it is a local minima at all). This turns out to be sufficient in the applications of interests for these lecture notes, and therefore our stance will be to largely ignore the problem. However, it is worth to provide at least a very high-level description of how one could design *global optimization* methods capable of (trying to) find a guaranteed global minimum. We know these methods cannot possibly be "very efficient", but still there are a number of clever ideas that can be put to use to obtain approaches that are in fact efficient enough in several practical applications.

2.8.1 A very quick glimpse to convexity

The first, obvious (but still poignant) observation is that global optimization is difficult . . . until it isn't. The point is that some functions, as we have seen, do not have local minima that are not also global ones. Thus, if one were sure to be minimizing one of these, (s)he could safely employ a local optimization approach with the certainty that a global minimum would be reached. The issue is, therefore, how to be sure that the function has this very desirable property.

The obvious request would be that f be *unimodal*, but in general it is not easy to verify if this is true. A stronger condition turns out to be more convenient, which can be intuitively understood as follows. Looking, e.g., to Figure 2.5, one can make the following intuitive consideration: if f has a local minima that is *not* the global one, then it it also has local *maxima* (the function has to grow away from the local minima, then stop growing and start decreasing again to get to the global minima). Thus, *if one were able to avoid local maxima*, the function would have the desired property. Another way of saying this is the following: any stationary point should be a local minima, i.e., whenever $f'(x) = 0$, one also has $f''(x) \geq 0$. There is then an obvious sufficient condition for this to happen: $f''(x) \geq 0$ *for all* $x \in \mathbb{R}$. In other words, f' *is monotone increasing (non decreasing)*. It is clear that in this case, *all the points such that* $f'(x) = 0$ *must form a contiguous interval*, before which the derivative is always negative, and after which it is always positive. These are, therefore, all the saddle points of f. Clearly they are all local minima, and clearly they as well are also global minima; let us call such a function *convex*. We avoid proving these statements because we'll see these definitions again in a more formal way later on; for the time being we only stress that we have given a definition that requires $f \in C^1$, which as we will see is not necessary.

The reason why convexity is preferred to unimodality is that convex functions are easier to characterize, and even more to *construct*. In fact, it can be proven (as we will see later on) that a reasonably large number of "basic" functions are convex, and furthermore that *carefully chosen functional operations preserve convexity*. This means that there is a "grammar of convex functions" [2]: by choosing the "atoms" properly, and only applying carefully selected operations, one can be sure that the result is a convex function. Thus, "the convex world is reasonably large": many different convex functions can be constructed, and there is even software for *disciplined convex programming* that enforces convexity [19]. This means that whenever one is constructing a model of a real-world problem, there is the choice to construct it convex. Of course, this is not always possible, as the model may therefore not properly represent reality. However, in many cases "reality" is not clearly definite; this is, for instance, the case in some *data-driven models* where one is trying to approximate some unknown function by picking the one that fits best in a(n infinite) family of functions with given forms. By properly choosing the target family of functions one can ensure that the resulting optimization problem is convex. We have already seen an example in §1.7, and we will see other cases later on (as well as cases where this does not happen). All in all, the case where a local optimization approach can be guaranteed to actually deliver global minima is by far not uncommon, especially if there is leverage in choosing aspects of the problem to be solved. In optimization, the catchphrase is "if you have the choice, choose convex" (when minimizing, we'll see the other direction); and it does happen that one has the choice. Actually, convexity arguments typically turn out to be crucial even if the problem is not convex.

2.8.2 Exploiting convexity

Thus, *global optimization is easy provided that* f *is convex*, since it then boils down to local optimization. This can be exploited algorithmically in several ways to help optimize *nonconvex* functions. Here we will provide an *extremely informal* description of the most important class of *exact algorithms* for global optimization, called *Spatial Branch-and-Bound* (SBB) approaches. The description will be provided by means of graphical examples only, leaving out all the (many) intricate algorithmic details that can be found in the literature (e.g., [6]) and that would require a whole course in themselves.

We consider global optimization on a *finite interval* $X = [\,\underline{x}\,, \overline{x}\,]$, i.e., with $-\infty < \underline{x} < \overline{x} < \infty$; finiteness is crucial for the approach. The crucial step on which the SBB is based—and whose absolutely nontrivial implementation we will *not* discuss—is that of finding a *convex valid global underestimator on* X, i.e., a *convex* function $\underline{f}$ such that $\underline{f}(x) \leq f(x)$ for all $x \in X$. This is represented by the blue line in Figure 2.7(left), which always lies below the black line representing (the graph of) f *inside* X; outside the interval this is not necessary, as the

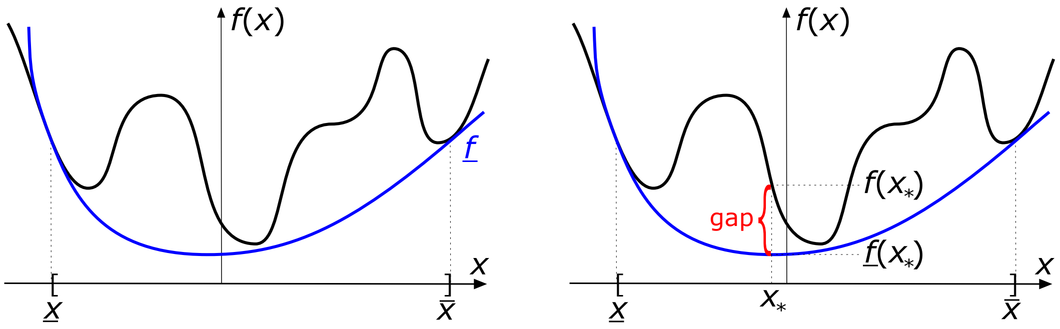


Figure 2.7: Spatial Branch-and-Bound: the root node

figure illustrates. Having obtained $\underline{f}$ (however this is done), one can efficiently compute its *global minimum* x_* by local approaches, especially if $\underline{f}$ is also continuously differentiable (at least once). Then, $\underline{f}_* = \underline{f}(\, x_* \,)$ is a *valid global lower bound* on the optimal value f_* of the problem: $\underline{f}_* \leq f_*$. We purposely ignore the fact that computing the exact x_*, and therefore its exact value $\underline{f}_*$, is in general impossible. This is significant, since only the bona-fide optimal value provides a valid lower bound on f_*: the value of any $\underline{f}(\, x^i \,)$ during the optimization process provides an *upper bound* on $\underline{f}_*$ that is therefore not guaranteed to be a lower bound on f_*. This is not a huge issue in practice since it may lead to an error in the computation of f_* of the same order of magnitude of the error in the computation of $\underline{f}_*$, and in general one has to be content with much worse accuracy in f_* than can be expected in $\underline{f}_*$; but we will eschew all these details and pretend to be able to compute the true value (or a valid lower approximation) of $\underline{f}_*$.

It is then natural (although, strictly speaking, not the only option) to compute the value of f at x_*, since $f(\, x_* \,)$ is then a *valid upper bound on* f_*. Hence, from the fact that $f(\, x_* \,) \geq f_* \geq \underline{f}_*$ (assuming $\underline{f}_* > 0$ to simplify the notation) we can estimate the (relative) *gap* (cf. Figure 2.7(right)) of x_*:

$$(\, f(\, x_* \,) - \underline{f}_* \,) \,/\, \underline{f}_* > (\, f(\, x_* \,) - f_* \,) \,/\, f_* \;.$$

Hence, if the gap is smaller than the desired tolerance ε, the algorithm can immediately terminate declaring x_* a ε-optimal solution.

This may, of course, not happen: $\underline{f}_*$ may be "much smaller" than f_*, and/or x_* may be a rather poor approximation of the optimal solution, i.e., $f(\, x_* \,)$ may be "much larger" than f_*. Here comes the second crucial element of the algorithm: *divide et impera*, also called *branching*. The idea is simply to divide the original interval $X = [\, \underline{x} \,,\, \overline{x} \,]$ into two (disjoint) sub-intervals; this is typically done by using x_* as the separation point, although there may be other choices. Then, one has *two* optimization problems defined on $X_0 = [\, \underline{x} \,,\, x_* \,]$ and $X_1 = [\, x_* \,,\, \overline{x} \,]$; the algorithm proceeds at solving each of them separately (we incidentally note, but not comment any further, that this can be done in parallel) and then pick the best solution between the two as the optimal solution of the original problem.

Of course, each of the two subproblems is in principle as difficult as the original one, hence "solving them" is easier said than done. However, *each of the two sub-intervals is smaller than the original one.* Since the crucial property of $\underline{f}$ is to be a valid global underestimator *on the relevant interval*, this means that $\underline{f}$ *can be recomputed anew for each of the subproblems.* Without going in any details, this means that *since the interval is smaller, the global underestimator can be expected to be a better approximation of* f; this actually happens in practice, for reasons that, again, we cannot even start discussing. The phenomenon is (artistically) depicted in Figure 2.8(left): the underestimator $\underline{f}^1$ now need only to be such over $X_1 = [\, x_* \,,\, \overline{x} \,]$, which allows a different and "tighter" (the word can be given a formal meaning, but we will keep the informal treatment) one to be generated. Once its minimum x^1_* is computed, one may discover that $f(\, x^1_* \,) - \underline{f}^1(\, x^1_* \,)$ is "small enough" so that one can consider the problem *restricted to* X^1 to be solved; let us assume that this actually happens. This, of course, does not terminate the process, since there still is X^0 to examine. However, in the picture, something else has happened that is useful: $f(\, x^1_* \,) < f(\, x_* \,)$. That is, the new point x^1_* is actually a better solution to the original problem. Clearly, this is the solution that one would report if the algorithm were stopped at this point. In general, an *incumbent solution* x_{best} is kept that is simply the one with lowest f-value found so far, with $f_{best} = f(\, x_{best} \,)$ being the *incumbent value*, i.e., the best (lowest) upper bound on f_* available to the algorithm.

The SBB then proceeds to examine X^0, which once again requires finding an underestimator $\underline{f}^0$—again, hopefully "tighter" than $\underline{f}$—over $X_0 = [\, \underline{x} \,,\, x_* \,]$ and finding its minimum x^0_*. Here, Figure 2.8(right) shows the other crucial mechanism, called *bounding*: *since* $\underline{f}^0(\, x^0_* \,) > f_{best}$*, one is sure that there is no better solution than* x_{best} *in* X^0. In fact, $f(\, x \,) \geq \underline{f}^0(\, x^0_* \,)$ for all $x \in X^0$, which immediately imply that $f(\, x \,) \geq f_{best}$ for all $x \in X^0$. The interval X^0 is "pruned": there is no reason to further consider *all* the points in there, as surely there is no way

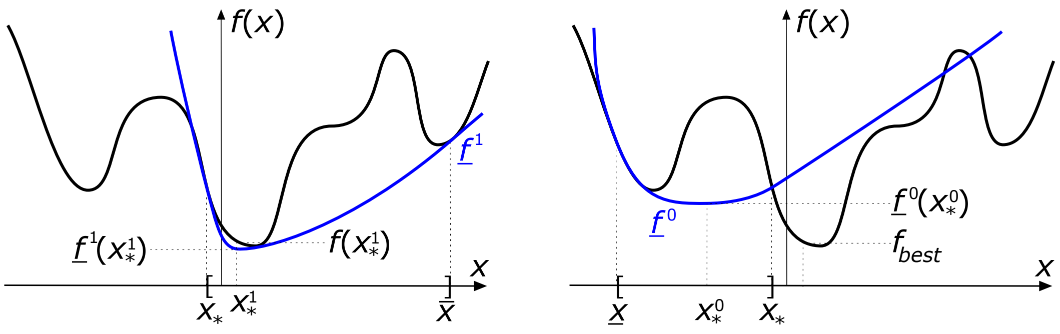


Figure 2.8: Spatial Branch-and-Bound: after "branching"

in which any of those can improve f_{best}. In the example, *the whole problem is solved*: x_{best} is an (approximately) optimal solution. It is clear how the solution of the convex approximations of the original problem (known as *relaxations*), play a crucial role in the approach by allowing to derive *valid lower bounds* on the optimal value (in each sub-interval) and using these to terminate the search without having to "explore all nooks and crannies of the feasible region" as in the approaches of §2.2.

Of course, the example is not complete in that the conditions for terminating the exploration of an interval may not be achieved. However, one can then repeat the approach by *branching again*, i.e., further subdividing the intervals. Under appropriate conditions, the algorithm will keep "slicing and dicing X" until the sub-intervals are "small enough" so that the problem is solved, one way or another, in each of them and the algorithm finally terminate; it is possible to show that the final incumbent x_{best} at the end of the algorithm is ε-optimal for the whole problem with the same ε that is used to declare optimality in the subproblems. The issue is clearly that the number of intervals to be explored grows exponentially; hence, these algorithms can in principle (and, unfortunately, easily in practice) be very expensive, up to the bounds seen in §2.2. Yet, depending on the f at hand, SBB can even be quite effective. Since we are not really discussing any of the crucial details it is impossible to give a precise account on when this does or does not happen, but intuitively one can think this to depend on: i) "how much nonconvex" f really is, and especially ii) how "tight" the computed $\underline{f}$ are as a lower approximation of f. There are several available well-engineered implementations of the approach, even open-source ones, that tackle the different complex tasks in possibly different ways; they are typically way less inefficient in practice than "blind global search", but still way more in efficient in practice than local optimization. The next example illustrates how this works on a toy problem.

Example 2.3. Global minimization of o polynominal

We want to miimize to global optimality the 10th degree polynomial of Example 2.2 over the interval $[-2, 4]$. To do this, first of all we write the problem in an *algebraic modelling language*, in this case `AMPL` [26], as follows:

```
var x >= -2, <= 4;

minimize objective:
  91 * x^2 / 30 - 19 * x^3 / 6 - 54 * x^4 / 25
+ 93 * x^5 / 23 - 23 * x^6 / 36 - 121 * x^7 / 93
+ 72 * x^8 / 91 - 13 * x^9 / 74 + 9 * x^10 / 640;
```

This is saved to a simple text file; then (skipping some technical steps for the sake of conciseness), the problem is passed to the open-source global optimization solver `Couenne` [43], that produces the following output (slightly edited for clarity):

```
Couenne 0.5.8 -- an Open-Source solver for Mixed Integer Nonlinear Optimization
ANALYSIS TEST: Couenne: new cutoff value 4.3977638536e-01 (0.003618 seconds)
NLP0012I
Constraints:            0
Variables:              1 (0 integer)
Auxiliaries:           10 (0 integer)
NLP0014I             1         OPT 2.0488868e-19        6 0.001125
Couenne: new cutoff value 2.0488868292e-19 (0.004999 seconds)
Coin0506I Presolve 52 (-8) rows, 10 (-1) columns and 112 (-16) elements
Clp0006I 0  Obj -70871.048 Primal inf 145848.9 (11)
Clp0006I 13  Obj -57241.771
Clp0032I Optimal objective -57241.77089 - 13 iterations time 0.002, Presolve 0.00
Cbc0012I Integer solution of 2.0488868e-19 found by Couenne Rounding NLP after 0 iterations and 0 nodes
NLP0014I             2         OPT 1.1312381        7 0.001798
Optimality Based BT: 4 improved bounds
Probing: 2 improved bounds
Cbc0031I 1 added rows had average density of 2
```

```
Cbc0013I At root node, 7 cuts changed objective from -57241.771 to 3074.8854 in 1 passes
Cbc0014I Cut generator 0 (Couenne convexifier cuts) - 7 row cuts average 2.0 elements, 2 column cuts (2 active)
Cbc0001I Search completed - best objective 2.048886829156005e-19, took 3 iterations and 0 nodes (0.01 seconds)

  "Finished"

Linearization cuts added in total:            60  (separation time: 0s)
Total solve time:                          0.007846s (0.007845s in branch-and-bound)
Lower bound:                             2.04889e-19
Upper bound:                             2.04889e-19  (gap: 0.00\%)
Branch-and-bound nodes:                        0
```

We cannot discuss in details what the output shows, but a few comments are appropriate. First of all, in order to solve the complex problem `Couenne` relies on (open-source) solvers for "simpler" problems, in particular `CLP` for linear programs [42] and `CBC` for mixed-integer linear programs [41]. This hinges on the fact that the problem is *reformulated* into a different form that allows to use specific algorithms, in particular by adding *auxiliary variables*

```
Variables:              1 (0 integer)
Auxiliaries:           10 (0 integer)
```

The optimal value is 0, and the corresponding feasible solution (giving an upper bound) is found very early on by local optimization:

```
NLP0014I             1         OPT 2.0488868e-19        6 0.001125
```

However, in order to terminate the algorithm needs to *certify* that this is the optimal solution by computing lower bounds on the optimal value as well. In fact, by not knowing that the obtained solution is already optimal, the solver tries to find a better one

```
NLP0014I             2         OPT 1.1312381        7 0.001798
```

and clearly fails to. At the beginning, computing the lower bounds also fails quite spectacularly:

```
Clp0000I Optimal - objective value -57241.771
```

Intuitively, the convex underestimator is "extremely bad". If the approach had started branching, as in our simplistic rendition, it would likely have required doing that many times. However, the solver has a fundamental trick up its sleeve: ways to improve the convex underestimator. These are called "cuts", and they work quite spectacularly:

```
Cbc0013I At root node, 7 cuts changed objective from -57241.771 to 3074.8854 in 1 passes
Cbc0014I Cut generator 0 (Couenne convexifier cuts) - 7 row cuts average 2.0 elements, 2 column cuts (2 active)
Cbc0001I Search completed - best objective 2.048886829156005e-19
```

Improving the convex underestimator requires some nontrivial work, that we cannot even start discussing here, but it gradually lifts the lower bound from `-5.6e+5` to `2.0e-19`, reducing the gap by some 24 orders of magnitude and solving the problem "at the root node": no branching at all is needed. This was of course a simple example, but the idea of heavily investing in improving the convex underestimator is one of the fundamental steps that actually dramatically contributes to these approaches being effective—when they are–in practice.

There would be many things to discuss in this line of approach, but these arguments are not appropriate for these lecture notes since exact approaches are typically not feasible for the kind of applications of interest here. Readers interested in global optimization algorithms are referred to the abundant literature on the subject and to the corresponding optimization courses [18].

2.9 Wrap up

Univariate (unconstrained, or box-constrained) optimization is a "good warm-up" to general (unconstrained) optimization: despite being considerably simpler in many respects, as we shall see, it already shows the fundamental traits found in the more complex cases:

- global optimization is in general "difficult", local optimization is "much easier";
- the only practical case in which global optimization is "easy" is when it is in fact equivalent to local optimization, i.e., f is *convex* (or, much less frequently, unimodal);
- local, hence global, optimization of convex functions is the computational workhorse of global optimization algorithms (if one is interested in such a thing, which we are not in these lecture notes);
- (local) optimization algorithms can be made faster by cleverly using more information about the function / problem, but this depends on the fact that this information is actually available;
- the fundamental form of information is that provided by *derivatives*: the more *continuous* derivatives one has access to, the faster algorithms one can construct;

- the role of derivatives is to construct *models* (first- and second-order ones) of the objective f, i.e., “simple” functions that “look like f” (at some points) and that can be used to inform algorithmic decisions;
- but “the map is not the world”: blindly trusting a model may lead to undesirable behaviour, hence one must always be careful not to be overly reliant on the model unless it can be proven / verified that it is indeed accurate enough.

These concepts will show up again and again in the *multivariate* optimization algorithms that we will start examining next.

2.10 Solutions

- **Solution to Exercise 2.1:** Let x_* be any optimal solution in X; by definition it belongs to (at least) one interval $[\, z^i \,,\, z^{i+1} \,]$, with $z^{i+1} - z^i = \Delta \leq 2\varepsilon \,/\, L$. Assume that $x_* - z^i \leq z^{i+1} - x_*$ (the other case is analogous); then $x_* - z^i \leq \varepsilon \,/\, L$. Hence, L-c gives $f(\, z^i \,) - f(\, x^* \,) \leq L|\, z^i - x^* \,| \leq \varepsilon$.

- **Solution to Exercise 2.2:** Basically done this already: $Dr^k < \varepsilon \;\equiv\; r^k < \varepsilon \,/\, D \;\equiv\; \log(\, r^k \,) < \log(\, \varepsilon \,/\, D \,) \;\equiv\; k \log(\, r \,) < \log(\, \varepsilon \,/\, D \,) \;\equiv\; k > \log(\, \varepsilon \,/\, D \,) \,/\, \log(\, r \,)$ as $r < 1 \implies \log(\, r \,) < 0$. Hence, $k \geq \log(\, D \,/\, \varepsilon \,) \,/\, \log(\, 1 \,/\, r \,)$.

- **Solution to Exercise 2.3:** Let us assume that $f(\, \overline{x}^i \,) \leq f(\, \underline{x}^i \,)$, so that the algorithm returns $f^i = f(\, \overline{x}^i \,)$ as its best estimate of the optimal value; the other case is analogous. Since f is L-c, $\overline{x}^i - \bar{x} \leq D^i = \overline{x}^i - \underline{x}^i \leq \delta$ implies $r^i = f^i - f(\, \bar{x} \,) \leq Ld^i \leq L\delta$. Hence, in order to to get $r^i \leq \varepsilon$ it is sufficient to ensure that $d^i \leq \varepsilon \,/\, L$, whence the bound.

- **Solution to Exercise 2.4:** Assume that $\lim_{h \to 0}(\, f(\, x + h \,) - f(\, x \,) \,) \,/\, h = L$, with L finite; then, multiplying both sides by h and using the fact that the limit of a product is the product of the limits, one gets $0 = L \lim_{h \to 0} h = \lim_{h \to 0} h(\, (\, f(\, x + h \,) - f(\, x \,) \,) \,/\, h \,) = \lim_{h \to 0} f(\, x + h \,) - f(\, x \,)$, i.e, f is continuous at x.

- **Solution to Exercise 2.5:** Consider $f(\, x \,) = \sqrt[3]{x^2}$, whose derivative is $f'(\, x \,) = 2 \,/\, 3x^3$ (possibly written in the more complex but algebraic-proof form $(2x) \,/\, (3(x^2)^{2/3})$). Hence, $\lim_{x \to 0_-} f'(\, x \,) = -\infty$ and $\lim_{x \to 0_+} f'(\, x \,) = \infty$. In plain words, this is because the cubic root is the inverse function of x^3, which is “flat in 0”; inverse functions “exchange the axes”, which means that if the graph of the function is “horizontal” as some point, the graph of its inverse becomes “vertical” at the same point. A “vertical” graph has infinite derivative.

- **Solution to Exercise 2.6:** The fact that f is L-c implies that $|\, f(x + h\,) - f(x) \,| \leq L|\, h \,|$, i.e., $|\, (\, f(\, x + h \,) - f(\, x \,) \,) \,/\, h \,| \leq L$; now just take the $\lim_{h \to 0}$. For the other direction one needs use the Mean Value Theorem [11, Theorem 2.3.9]: if $f \in C^1$ in the interval $[\, x \,,\, z \,]$, then $f(\, z \,) - f(\, x \,) = f'(\, w \,)(\, z - x \,)$ for some $x \leq w \leq z$; now take the $|\, \cdot \,|$ and use the fact that $|\, f'(\, w \,) \,| \leq L$.

- **Solution to Exercise 2.7:** As already seen, $f(\, x \,) = \sqrt[3]{x^2}$ has $\lim_{x \to 0_-} f'(\, x \,) = -\infty$ and $\lim_{x \to 0_+} f'(\, x \,) = \infty$. Thus $f'(\cdot)$ is not bounded in any interval around 0, and therefore $f(\cdot)$ is not L-c there. Of course, $f'(\, x \,)$ is not continuous in 0.

- **Solution to Exercise 2.8:** For $\overline{x} = \Delta x = 1$, the algorithm tries the iterates $1, 2, 4, 8, \ldots$, i.e., 2^i. With $f(\, x \,) = \sin(\, \pi x + 3\pi \,/\, 4 \,) \implies f'(\, x \,) = \pi \cos(\, \pi x + 3\pi \,/\, 4 \,)$ we have $f'(\, 2^i \,) = \pi \cos(\, \pi 2^i + 3\pi \,/\, 4 \,) = \pi \cos(\, 3\pi \,/\, 4 \,) = -\pi\sqrt{2} \,/\, 2 \approx -2.22$ (2^i is always even and $\cos(\cdot)$ has period 2π); that is, the algorithm always finds a “very negative” derivative and never stops, although $f(\cdot)$ has plenty of local minima. Clearly, by only very slightly changing the constants the counterexample would break down

- **Solution to Exercise 2.9:** Rewrite $\underline{x}f'(\, \overline{x} \,) - \overline{x}f'(\, \underline{x} \,) = \underline{x}f'(\, \overline{x} \,) - \overline{x}f'(\, \underline{x} \,) + \underline{x}f'(\, \underline{x} \,) - \underline{x}f'(\, \underline{x} \,)$ and regroup to get $\underline{x}(\, f'(\, \overline{x} \,) - f'(\, \underline{x} \,) \,) - f'(\, \underline{x} \,)(\, \overline{x} - \underline{x} \,)$. Now divide by $f'(\, \overline{x} \,) - f'(\, \underline{x} \,)$ to get $x = \overline{x} + \alpha(\, \overline{x} - \underline{x} \,)$ with $0 \leq \alpha = -f'(\, \underline{x} \,) \,/\, (\, f'(\, \overline{x} \,) - f'(\, \underline{x} \,) \,) \leq 1$; it is then plain to see that $\underline{x} \leq x \leq \overline{x}$.

- **Solution to Exercise 2.10:** $[\, Q_i \,]'(\, x \,) = L_i'(\, x \,) = f'(\, x^i \,) + f''(\, x^i \,)(\, x - x^i \,) = 0 \;\equiv$
  $x - x^i = -f'(\, x^i \,) \,/\, f''(\, x^i \,)$

Chapter 3

Unconstrained Multivariate Optimality and Convexity

3.1 Unconstrained Multivariate Optimization

We now move to general *unconstrained multivariate problems* where the objective is $f : \mathbb{R}^n \rightarrow \mathbb{R}$. That is,

$$f(\,x\,) = f(\,x_1\,,\,\ldots\,,\,x_n\,) \qquad \text{with} \qquad x = [\,x_i\,]_{i=1}^n = [\,x_1\,,\,\ldots\,,\,x_n\,] \in \mathbb{R}^n\ ;$$

in layman terms, f has more than one variable. As soon as $n > 1$ a number of issues immediately become more complex, so we will not in general comment on how large n really is. In practice, it can be either quite small (2, 3, ...), small-ish (10s to 100s), quite large (1000s to tens/hundreds of 1000s), and even heniously large (millions, many billions, closing in to trillions). This clearly does have a very significant impact in many senses and we will extensively commenta about it in the following.

3.1.1 Multivariate (box-constrained) global optimization

We start with a general (informal) remark: $\mathbb{R}^n$ is the n-times Cartesian product $\mathbb{R} \times \mathbb{R} \times \ldots \mathbb{R}$, and therefore *the size of* $\mathbb{R}^n$ *grows very rapidly with* n. Of course, $\mathbb{R}$ is already uncountably infinite, and in theory $\mathbb{R}^n$ is not larger than $\mathbb{R}$. But in practice, things are quite different. Indeed, consider the case where one has box constraints, so rather than $\mathbb{R}$ the function has to be minimized over, say, $[\,-1\,,\,1\,]$. Then, of course, the multivariate case is $[\,-1\,,\,1\,]^n$. And then, recall Example 1.3: if f is (quadratic) concave, then the optimal solution is on one of the vertices of the unitary hypercube $\{\,-1\,,\,1\,\}^n$. This would seem to be good news, since these are a finite number, and therefore the minimization problem can surely be finitely (exactly) solved. This is indeed good news if n is small, but the number of vertices is 2^n: as $2^{10} \approx 10^3$, 2^{100} is already the astronomical 10^{30}, and no task requiring 2^{1000} operations can ever be considered. Thus, *problems that can be tackled for small n become entirely unfeasible when n grows larger.* Informally speaking, $\mathbb{R}^n$ is "exponentially larger than $\mathbb{R}$": hence, in n dimensions "there is a lot more solutions to look at", and optimizing is much more difficult.

Of course, this statement holds if it is true that the "size" of the space is directly correlated with the difficulty of finding something in it. In other words, one is assuming that there is no "guiding light" that can be used to efficiently drive an algorithm towards the desired point, and therefore that "looking (almost) everywhere is needed". Unfortunately, this happens to be the case, unless of course some strong (but not necessarily always unfeasible) assumptions are made. Of course one first requires L-c, for otherwise the impossibility results if §2.1 immediately apply; Appendix A.6 has already provided the definition so that it works for arbitrary n. With L-c, the already mentioned [15, Theorem 4.4] is in fact directly stated for the general multivariate case, and proves that any ϵ-approximate algorithm has an "adversarial nemesis" that will require it to expand—to keep the notation as close as possible to the previous one—$O(\,(\,LD\,/\,\epsilon\,)^n\,)$ iteration to get there (with D the size the underlying hyper-rectangle). This confirms that, at least in the worst-case sense, (approximate) global optimality cannot be achieved "without looking in every nook and cranny of the space", and therefore that the exponential increase of the "places to look at" can translate in a proportional increase of the computational effort. Similar dependency on $(\,1\,/\,\epsilon\,)^n$ can be found in results about the expected number of iterations for randomised algorithms [15, Lemma 4.5], proving that randomisation is, in this case, no "silver bullet" for the underlying issue: global optimization is "difficult". Although practical implementations of global optimization algorithms, e.g., along the lines of §2.8.2, typically do not show all of the dramatic impact that the worst-case result would suggest, it is true that the computational cost can increase very rapidly with n.

This has spurned the development of many *heuristics*, of which efficient implementations are available [20], which can efficiently provide good (but not provably optimal) solutions in many practical cases. These are based on a number of clever ideas, often using randomization, that cannot be discussed in these lecture notes. One of these, however, is that once some iterate x^i is produced by whatever main algorithmic process is employed, it may make sense to *refine it by trying to identify a close-by local minimum*, insomuch as this can be done efficiently. Indeed, the multivariate extension of (2.1) is trivial with the natural concept of *neighbourhood* provided by §A.8. Hence, efficient *local optimization* is typically the workhorse of global optimization, in particular for *bound computation* purposes as hinted at in §2.8.2. It therefore makes sense to start with—and, in these lectures, to limit ourselves to—local optimization algorithms.

3.1.2 Multivariate (unconstrained) local optimization

We will therefore be mainly interested in *local optimization* approaches aiming at finding a local minima (or maxima). Of course, the fundamental issue here is *how to identify a local minimum* is one is already close by, and *how to reach closer* if one is not. As we have seen, these issues are best solved by the use of *derivatives*. However, derivatives for a multivariate function are "more complex objects" (especially as the order grows) than for univariate functions, and this will require the introduction of an appropriate set of concepts in the next section.

Doing so pays off, in the sense that one can obtain *dimension-independent* efficiency results that are often surprisingly analogous to these seen in Chapter 1 for the (multivariate) quadratic case: the converge speed depends on some properties of the function, but most often *does not explicitly depend on n*. When it does, the dependence is not exponential. A closer analysis will reveal that the analogy between convergence results for "simple" and "not simple" functions is not so surprising after all, since simple functions are used to construct the (simple) *models* that then drive the behaviour of the algorithms. One of the fundamental ideas is that once the algorithm starts converging to some point, successive iterates get closer and closer to each other: as a consequence, the model starts getting a better and better approximation of the function. In other words, *the function almost behaves as the model*, which justifies why convergence results for the model will be relevant, and convergence can be fast. Such dimension-independent results are therefore enticing for very- to extremely-large-scale problems in that they foreshadow algorithms that can converge in "few" iterations even when n is huge.

Of course, this does not mean that these algorithms will necessarily be "fast"; for once, convergence speed may be rather low ("badly linear" or worse). Furthermore, the cost of the algorithm cannot be entirely independent of the dimension of the problem: each iteration will require computing f and its derivatives, whose cost—and, as we will see, memory footprint—necessarily increases with n, sometimes significantly so. Indeed, for very-large-scale problems, even a cost (memory footprint) of the order of $O(\, n^2 \,)$ may render an algorithm unfeasible, negating any advantage due to superior convergence speed. Besides, the properties of the function on which the convergence depends may not themselves be dimension-independent: in practice they may vary with n, which may introduce "hidden" dependencies from the number of variables. Nonetheless, these algorithms are clearly the ones that make sense to study first and foremost. It therefore makes sense to acquaintance oneself with the necessary concepts about derivatives, which these algorithms depend onto. Indeed, as we shall see, *the quality of the available derivative information is the fundamental driver of the efficiency of the algorithms*: "more" and "better" derivatives translate into "faster" algorithms (with caveats), and vice-versa. This is not to say that it doesn't make sense in practice to consider algorithms that do not make any use of derivative information: this is clearly possible, as illustrated, e.g., in §2.4, and in several cases plainly necessary since, quite simply, derivative information is not there to be had. It is also highly sub-optimal: by dint of the same principle described above, no derivative information typically translates into "slower" algorithms. Since derivatives are naturally—which does not necessarily means easily—available for the applications of interest in these lectures, we will not describe the several available methods that do away with them; the interested reader is referred, e.g., to [8, Chapter 9], [1, §8.4].

3.2 Gradients, Jacobians, and Hessians

3.3 Optimality conditions

3.4 A Quick Look to Convex Functions

3.5 Ex-post Motivation: (Artificial, Deep) Neural Networks

3.6 Solutions

Chapter 4

Smooth Unconstrained Optimization

Chapter 5

Nonsmooth Unconstrained Optimization

Part III

Constrained Optimization

Chapter 6

Constrained Optimality and Duality

Chapter 7

Constrained Optimization

Part IV

Combinatorial Optimization

Chapter 8

A Fleeting Glimpse to Combinatorial Optimization

Part V

Supplementary Material

Bibliography

[1] M.S. Bazaraa, H.D. Sherali, C.M. Shetty *Nonlinear Programming: Theory and Algorithms*, John Wiley & Sons, 2006.

[2] S. Boyd, L. Vandenberghe *Convex Optimization*, Cambridge University Press, 2008 [https://web.stanford.edu/~boyd/cvxbook]

[3] S. Bubeck *Convex Optimization: Algorithms and Complexity*, arXiv:1405.4980v2 [https://arxiv.org/abs/1405.4980, 2015]

[4] P. Hansen, B. Jaumard "Lipschitz Optimization" in *Handbook of Global Optimization – Nonconvex optimization and its applications*, R. Horst and P.M. Pardalos (Eds.), Chapter 8, 407–494, Springer, 1995

[5] M.R. Garey, D.S. Johnson *Computers and Intractability: A Guide to the Theory of NP-Completeness* Freeman, 1979

[6] D. Li, X. Sun *Nonlinear Integer Programming* Springer International Series in Operations Research & Management Science, 2006

[7] D.G. Luenberger, Y. Ye *Linear and Nonlinear Programming*, Springer International Series in Operations Research & Management Science, 2008

[8] J. Nocedal, S.J. Wright, *Numerical Optimization – second edition*, Springer Series in Operations Research and Financial Engineering, 2006

[9] W. Sun, Y.-X. Yuan, *Optimization Theory and Methods – Nonlinear Programming*, Springer Optimization and Its Applications, 2006.

[10] H.C. Pinkham *Analysis, Convexity, and Optimization* Draft of September 4, 2014 [https://www.math.columbia.edu/department/pinkham/Optimizationbook.pdf]

[11] W.F. Trench, *Introduction to Real Analysis* Free Hyperlinked Edition 2.04, December 2013 [http://ramanujan.math.trinity.edu/wtrench/texts/TRENCH_REAL_ANALYSIS.PDF]

[12] M. Cacciola, A. Frangioni, X. Li, A. Lodi "Deep Neural Networks pruning via the Structured Perspective Regularization" *SIAM Journal on Mathematics of Data Science* 5(4), 1051—1077, 2023

[13] M. Collins *Computational Graphs, and Backpropagation*
Course notes for NLP, Columbia University https://www.cs.columbia.edu/~mcollins/ff2.pdf

[14] E. de Klerk "The complexity of optimizing over a simplex, hypercube or sphere: a short survey" *Central European Journal of Operations Research* 16: 111–125, 2008 [https://link.springer.com/content/pdf/10.1007/s10100-007-0052-9.pdf]

[15] M. Gaviano, D. Lera "Complexity of general continuous minimization problems: a survey" *Optimization Methods and Software* 20(4-5), 1–20, 2005

[16] L. Serafino *Optimizing Without Derivatives: What Does the No Free Lunch Theorem Actually Say?* Notices of the AMS 61(7):750–755, 2014 https://www.ams.org/notices/201407/rnoti-p750.pdf

[17] AutoDiff Org: https://www.autodiff.org

[18] CommaLab: https://commalab.di.unipi.it/courses

[19] CVX: http://cvxr.com

[20] The Derivative-Free Library (DFL) at DIAG: https://github.com/DerivativeFreeLibrary

[21] The Fido Project: https://fidoproject.github.io

[22] OSI (Optimization Solver Interface) https://github.com/coin-or/Osi

[23] PyTorch: https://pytorch.org

[24] scikit-learn: https://scikit-learn.org

[25] TensorFlow: https://www.tensorflow.org

[26] AMPL https://ampl.com

[27] GAMS https://www.gams.com

[28] AIMMS https://www.aimms.com/platform/aimms-development

[29] OPL https://www.ibm.com/docs/en/icos/12.8.0.0?topic=opl-optimization-programming-language

[30] Coliop http://www.coliop.org

[31] ZIMPL https://zimpl.zib.de

[32] FLOPCpp https://github.com/coin-or/FlopCpp

[33] COIN Rehearse https://github.com/coin-or/Rehearse

[34] Gravity https://github.com/coin-or/Gravity

[35] PuLP https://github.com/coin-or/pulp

[36] Pyomo http://www.pyomo.org

[37] JuMP https://github.com/jump-dev/JuMP.jl

[38] YALMIP https://yalmip.github.io

[39] The SMS++ Structured Modelling System https://gitlab.com/smspp/smspp-project

[40] The SCIP (Solver Constraint Integer Programs) solver https://www.scipopt.org

[41] The CBC (COIN Branch & Cut) Solver https://github.com/coin-or/Cbc

[42] CLP: COIN-OR Linear Program Solver https://www.coin-or.org/Clp

[43] Couenne (Convex Over and Under ENvelopes for Nonlinear Estimation) https://www.coin-or.org/Couenne

[44] The HiGHS (high performance software for linear optimization) solver https://highs.dev

[45] The IBM/ILOG CPLEX solver https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

[46] The Gurobi solver https://www.gurobi.com

[47] The Mosek solver https://www.mosek.com

[48] The MCFClass project https://github.com/frangio68/Min-Cost-Flow-Class

[49] Wikipedia – Bolzano-Weierstrass theorem:
https://en.wikipedia.org/wiki/Bolzano%E2%80%93Weierstrass_theorem

[50] Wikipedia – Cholesky decomposition: https://en.wikipedia.org/wiki/Cholesky_decomposition

[51] Wikipedia – Cubic equation: https://en.wikipedia.org/wiki/Cubic_equation

[52] Wikipedia – Determinant: https://en.wikipedia.org/wiki/Determinant

[53] Wikipedia – Derivative: https://en.wikipedia.org/wiki/Derivative

[54] Wikipedia – Double-precision floating-point format:
https://en.wikipedia.org/wiki/Double-precision_floating-point_format

[55] Wikipedia – Factorization of polynomials:
https://en.wikipedia.org/wiki/Factorization_of_polynomials

[56] Wikipedia – Golden ratio https://en.wikipedia.org/wiki/Golden_ratio

[57] Wikipedia – Laplace expansion: https://en.wikipedia.org/wiki/Laplace_expansion

[58] Wikipedia – Eigenvalue Algorithm: https://en.wikipedia.org/wiki/Eigenvalue_algorithm

[59] Wikipedia – Eigenvalues and Eigenvectors:
https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors

[60] Wikipedia – Extreme Value Theorem: https://en.wikipedia.org/wiki/Extreme_value_theorem

[61] Wikipedia – Fibonacci sequence https://en.wikipedia.org/wiki/Fibonacci_sequence

[62] Wikipedia – Intermediate Value Theorem:
https://en.wikipedia.org/wiki/Intermediate_value_theorem

[63] Wikipedia – Islands of Space: https://en.wikipedia.org/wiki/Islands_of_Space

[64] Wikipedia – Matrix Norm: https://en.wikipedia.org/wiki/Matrix_norm

[65] Wikipedia – Minifloat: https://en.wikipedia.org/wiki/Minifloat

[66] Wikipedia – Norm: https://en.wikipedia.org/wiki/Norm_(mathematics)

[67] Wikipedia – The Hitchhiker's Guide to the Galaxy:
https://en.wikipedia.org/wiki/The_Hitchhiker's_Guide_to_the_Galaxy

[68] Wikipedia – Underdetermined System: https://en.wikipedia.org/wiki/Underdetermined_system

Appendix A

Miscellaneous Mathematical Background

A.1 Infima, suprema and $\overline{\mathbb{R}}$

Some fundamental properties of the set $\mathbb{R}$ of real numbers are briefly recalled. For more detailed/formal treatment consult, e.g., [11, Chapter 1], [10, Lecture 2], [2, A.2.2].

Une of the fundamental, although apparently trivial, properties of $\mathbb{R}$ is that of being *totally ordered*: however chosen x and $z \in \mathbb{R}$, at least one among $x \leq z$ and $z \leq x$ holds true. This is crucial for optimization, as discussed in the introduction; formally speaking this means that any set $S \subseteq \mathbb{R}$ has a well identified "best element" when minimizing or maximizing. Formally, the number $\underline{s} \in R$ is the *infimum* of S, written $\underline{s} = \inf S$, if $\underline{s} \leq s$ for all $s \in S$, but however chosen $t > \underline{s}$ there exists some $s \in S$ such that $s \leq t$. In plain words, $\underline{s}$ is "below" all elements of S, but for any t "above" $\underline{s}$ this is not true, i.e., there is an element of S "below" t. Another way of expressing it is that the infimum is *the largest among all the minorants of* S, i.e., the numbers $t \in R$ such that $t \leq s$ for all $s \in S$. This however requires defining "the largest", which is of course the completely symmetric definition of the *supremum* of S: $\overline{s} = \sup S$ if $\overline{s} \geq s$ for all $s \in S$, but however chosen $t < \overline{s}$ there exists $s \in S$ such that $s \geq t$. Again, the supremum is *the smallest of the majorants*.

The issue is that not all set of real numbers have *finite* infimum or supremum: $S = \mathbb{R}$ is the obvious example. It is therefore convenient to introduce the set of *extended reals*: $\overline{\mathbb{R}} = \{ -\infty \} \cup \mathbb{R} \cup \{ +\infty \}$, where the special elements $-\infty$ and $+\infty$ are, respectively, the infimum of sets with no infimum and the supremum of sets with no supremum. Formally,

$$\inf S = -\infty \quad \Longleftrightarrow \quad \forall\, t \in \mathbb{R}\; \exists\, s \in S \text{ such that } s \leq t\;,$$

i.e., if the set S is *unbounded below* (there are elements "as small (large negative) as one wants them"). Symmetrically,

$$\sup S = +\infty \quad \Longleftrightarrow \quad \forall\, t \in \mathbb{R}\; \exists\, s \in S \text{ such that } s \geq t\;,$$

i.e., if the set S is *unbounded above* (there are elements "as large (positive) as one wants them"). The two special symbols "$-\infty$" and "$+\infty$" can, most of the times, be treated like ordinary real numbers with a bit of tweaking of the standard algebraic rules [11, p. 7], therefore usually we will avoid distinguishing between $\mathbb{R}$ and $\overline{\mathbb{R}}$ unless strictly necessary.

By contrast, *lower/upper bounded* subsets of $\mathbb{R}$ always possess finite inf/sup. Formally, one should distinguish the case where $\underline{s} \in S$ (which implies $\underline{s} \in \mathbb{R}$, i.e., S lower bounded) from the case where $\underline{s} \notin S$ (but still $\underline{s} \in \mathbb{R}$). In the first case $\underline{s}$ is also the *minimum* of S ($\underline{s} = \min S$), whereas in the second case S has no minimum. In our setting where "$s \in R$" actually means "s is a fixed-size floating-point number" the distinction is hardly relevant, thus "min" will largely be considered equivalent to "inf". Symmetrically, $\overline{s} = \max S$ will largely be considered equivalent to "sup", even if formally $\sup S \notin S$.

As a final note, under the spirit that the infimum is the largest among all the minorants of S one readily obtains the convenient definition that $\inf \emptyset = +\infty$ (any $t \in R$ is below all elements of $S = \emptyset$, and $+\infty$ is the largest of them all), and symmetrically $\sup \emptyset = -\infty$ since the supremum is the smallest of the majorants.

A.2 Vector space, scalar product

A vector space is any set V that is closed under two operations: addition (sum), and scalar multiplication (by a real value). That is, if $v \in V$, $w \in V$ and $\alpha \in \mathbb{R}$, then both $v + w \in V$ and $\alpha v \in V$.

For each integer $n \geq 1$, the set

$$\mathbb{R}^n = \{ \, x = [\, x_1 \, , \, x_2 \, , \, \dots , \, x_n \,] = [\, x_i \,]_{i=1}^n \, : \, x_1 \in \mathbb{R} \, , \, \dots \, , \, x_n \in \mathbb{R} \, \}$$

of all n-dimensional vectors of reals is a vector space once the two operations are defined in the obvious way:

$$\begin{array}{l} x + z = [\, x_1 + z_1 \, , \, x_2 + z_2 \, , \, \dots , \, x_n + z_n \,] = [\, x_i + z_i \,]_{i=1}^n \\ \alpha x = [\, \alpha x_1 \, , \, \alpha x_2 \, , \, \dots , \, \alpha x_n \,] = [\, \alpha x_i \,]_{i=1}^n \end{array} .$$

$\mathbb{R}^n$ is in particular a *finite-dimensional* vector space in that it has (at least) a finite *basis*: there exists a finite set of vectors, in particular exactly n of them, $V = `u^1 u^2 \dots v^n = \{ \, v^i \, \}_{i=1}^n$ such that "all other vectors can be obtained by sum and scalar multiplication out of them". Formally

$$\forall \, x \in \mathbb{R}^n \; \exists \, \alpha_1, \dots, \alpha_n \text{ such that } x = \alpha_1 u^1 + \dots + \alpha_n u^n \; .$$

There are of course infinitely many different bases of $\mathbb{R}^n$, but the most obvious is the *canonical base* whereby for each $i = 1, \dots, n$, $u_i^i = 1$ and $u_h^i = 0$ for $h \neq i$; in this case, $\alpha_i = x_i$. Although these lecture notes only deal with $\mathbb{R}^n$ and therefore give finite dimensionality for granted, it is worth remarking that not all vector spaces are finite-dimensional.

After the two defining operations, the most important one is the *(Euclidean) scalar product* of $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$

$$\langle \, x \, , \, z \, \rangle = \textstyle\sum_{i=1}^n x_i z_i = x_1 z_1 + \dots + x_n z_n \; .$$

The scalar product has the following properties, that are easy to verify:

1. $\langle \, x \, , \, z \, \rangle = \langle \, z \, , \, x \, \rangle \; \forall x \, , \, z \in \mathbb{R}^n$ (symmetry);
2. $\langle \, x \, , \, x \, \rangle \geq 0 \; \forall x \in \mathbb{R}^n$, and $\langle \, x \, , \, x \, \rangle = 0 \iff x = 0$;
3. $\langle \, \alpha x \, , \, z \, \rangle = \alpha \langle \, x \, , \, z \, \rangle \; \forall x \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$;
4. $\langle \, x + w \, , \, z \, \rangle = \langle \, x \, , \, z \, \rangle + \langle \, w \, , \, z \, \rangle \; \forall x \, , \, w \, , \, z \in \mathbb{R}^n$.

These could be taken as the definition of the concept of scalar product, of which different versions exist in other vector spaces (matrices, integrable functions, random variables, . . .) which are mostly of no concern here, although some "nonstandard" notion of scalar product will turn out to be useful in one particular application. Rather, point 2. immediately introduces the concept of *(Euclidean) norm induced by the (Euclidean) scalar product*:

$$\| \, x \, \| = \sqrt{\textstyle\sum_{i=1}^n x_i^2} = \sqrt{\langle \, x \, , \, x \, \rangle} \; .$$

The norm provides a measure of the "size" of a vector: from the properties of the scalar product immediately follows that

1. $\| \, x \, \| \geq 0 \; \forall x \in \mathbb{R}^n$ and $\| \, x \, \| = 0 \iff x = 0$
2. $\| \, \alpha x \, \| = | \, \alpha \, | \| \, x \, \| \; \forall x \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$;
3. $\| \, x + z \, \| \leq \| \, x \, \| + \| \, z \, \| \; \forall x \, , \, z \in \mathbb{R}^n$ (triangle inequality).

Again, these could be taken as the definition of the concept of norm [66][2, A.1.2][8, p. 600]; in fact, even in these lecture notes different norms than the Euclidean one will be considered, but all satisfying the properties above. This will especially—but not only—be true for *matrices*. However, any norm immediately provides a mean of characterising "how far" (or "close") two elements of a vector space are by their *(Euclidean) distance*

$$d(\, x \, , \, z \,) = \sqrt{\textstyle\sum_{i=1}^n (x_i - z_i)^2} = \| \, x - z \, \| \; ,$$

i.e., "the norm of x when z is the origin". Of course, the distance inherit the properties of the norm:

1. $d(\, x \, , \, z \,) \geq 0 \; \forall x \, , \, z \in \mathbb{R}^n$ and $d(\, x \, , \, z \,) = 0 \iff x = z$;
2. $d(\, \alpha x \, , \, 0 \,) = |\alpha| d(\, x \, , \, 0 \,) \; \forall x \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$;
3. $d(\, x \, , \, w \,) \leq d(\, x \, , \, z \,) + d(\, z \, , \, w \,) \; \forall x \, , \, w \, , \, z \in \mathbb{R}^n$ (triangle inequality).

In turn, the distance immediately provides the notion of "all the points close enough to x", i.e., the *ball with center $x \in \mathbb{R}^n$ and radius $r > 0$*

$$\mathcal{B}(\, x \, , \, r \,) = \{ \, z \in \mathbb{R}^n \, : \, \| \, z - x \, \| \leq r \, \} \; .$$

This is a central concept in the *topology* of vector spaces, as briefly discussed later on.

Here we rather focus on the important *geometric characterization of the scalar product*:

$$\langle\, x \,,\, z \,\rangle = \|\, x \,\| \cdot \|\, z \,\| \cdot \cos(\, \theta \,) \;,$$

where θ is the angle formed between x and z (when both are considered applied in the origin). This immediately provides a characterization of "when two vectors point roughly in the same direction":

- $\langle\, x \,,\, z \,\rangle > 0$ means that θ is less than a square angle, i.e., "x and z roughly point in the same direction";
- $\langle\, x \,,\, z \,\rangle = 0$ means that the angle θ is square, i.e., x and z are orthogonal ($x \perp z$);
- $\langle\, x \,,\, z \,\rangle < 0$ means that θ is greater than a square angle, i.e., "x and z roughly point in opposite directions".

This interpretation will be repeatedly useful and it is illustrated in Figure A.1.

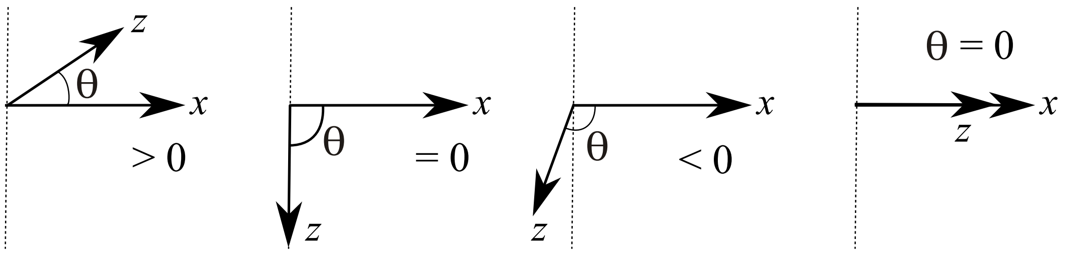


Figure A.1: Geometric characterization of the scalar product

An immediate consequence of the geometric characterization is the *Cauchy-Schwarz inequality*

$$|\, \langle\, x \,,\, z \,\rangle \,| \leq \|\, x \,\| \|\, z \,\| \quad \forall x \,,\, z$$

stating that (absolute value of) the scalar product is always less than or equal to the product of the norms. In particular, the scalar product is *equal* to the product of the norms only when the vectors are *collinear* (cf. Figure A.1(right)), i.e., they lie exactly along same line save possibly for the sign (when $\cos(\,\theta\,) = -1$, i.e., x and z point in exactly opposite directions); this will repeatedly come useful. Two other useful properties are

$$\|\, x + z \,\|^2 = \|\, x \,\|^2 + \|\, z \,\|^2 + 2\langle\, x \,,\, z \,\rangle$$

(this only holds true for the Euclidean norm), and the parallelogram Law

$$2\|\, x \,\|^2 + 2\|\, z \,\|^2 = \|\, x + z \,\|^2 + \|\, x - z \,\|^2 \;.$$

A more thorough presentation of these concepts can be found in almost every book, e.g., [2, A.1.1], [10, Lecture 5], [11, §5.1].

A.3 Matrices, transpose, symmetry, products

Matrices [2, A.5] generalise vectors in that, rather than assuming the numbers to we in an "unstructured list", the elements have a precise geometrical arrangement. That is, a (real) matrix $A \in R^{m \times n}$ is a rectangular arrangement of $m \times n$ numbers with m rows and m columns. In a formula

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix} = \left[\, A^1 \,,\, A^2 \,,\, \dots \,,\, A^n \,\right] \;.$$

That is, each element of A is characterised by two indices: A_{ij} is the element in row i and column j. It is often expedient to see A as formed by m "horizontal" rows $A_i \in \mathbb{R}^n$, $i = 1, \dots, m$, stacked "vertically" on top of one another, or alternatively as formed by n "vertical" columns $A^j \in \mathbb{R}^m$, $j = 1, \dots, n$, stacked "horizontally" next to one another. This may ingenerate confusion as to whether a vector $x \in \mathbb{R}^n$ is "horizontal" or "vertical"; albeit it's mostly a matter of notation, the usual choice is that $x \in \mathbb{R}^n$ is the same as $x \in \mathbb{R}^{n \times 1}$, i.e., a "column vector".

A useful operation is exchanging rows with columns, i.e., the transpose. This may start with the humble vector: if $x \in \mathbb{R}^{n \times 1}$ is a column vector, then $x^T \in \mathbb{R}^{1 \times m}$ is a row vector: from "vertical" it becomes "horizontal". In general, if $A \in R^{m \times n}$, then $A^T \in R^{n \times m}$ with

$$A^T = \left[\begin{array}{c} (A^1)^T \\ (A^2)^T \\ \vdots \\ (A^n)^T \end{array} \right] = \left[\begin{array}{cccc} A_{11} & A_{21} & \dots & A_{m1} \\ A_{12} & A_{22} & \dots & A_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \dots & A_{mn} \end{array} \right] = \left[\, A_1^T \, , \, A_2^T \, , \, \dots \, , \, A_m^T \, \right] .$$

Of course, transposing the transpose gives back the original matrix: $(A^T)^T = A$. In general, a matrix and its transpose "look very different" although they are formed of the same elements. There is one crucial case, however, when this does not happen: this is the case where $A = A^T$. i.e., A is *symmetric*. This can of course only happen if $n = m$, i.e., A is a *square matrix* and just means that $A_{ij} = A_{ji}$ for all $1 \leq i \leq j \leq n$. Symmetric matrices have the very useful property that they can be represented only with $n(n-1)/2$ numbers instead of n^2, i.e., they require roughly half the memory of a non-symmetric matrix of the same size. Note that $a \in \mathbb{R}$ can be considered $a \in \mathbb{R}^{1 \times 1}$, and obviously $a = a^T$: a (real) number is a symmetric square (real) matrix.

The fundamental operation with matrices in these lecture notes is the *matrix-vector product*. Since matrices are in general non-symmetric and $m \neq m$ happens, there are two versions of matrix-vector product: *on the left*, which is only possible if the vector is $x \in \mathbb{R}^n$, i.e., it has the same size as rows

$$Ax = \left[\begin{array}{c} \langle \, A_1 \, , \, x \, \rangle \\ \langle \, A_2 \, , \, x \, \rangle \\ \vdots \\ \langle \, A_m \, , \, x \, \rangle \end{array} \right] = \left[\begin{array}{c} A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n \\ A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n \\ \vdots \\ A_{m1}x_1 + A_{m2}x_2 + \dots + A_{mn}x_n \end{array} \right] = \left[\begin{array}{c} v_1 \\ v_2 \\ \vdots \\ v_m \end{array} \right] \in \mathbb{R}^{m \times 1}$$

and produces a (column) vector $v \in \mathbb{R}^m$, i.e., with the same size as columns, and, vice-versa, *on the right*, which is only possible if the vector is $x \in \mathbb{R}^m$, i.e., it has the same size as columns

$$x^T A = \left[\, \langle \, A^1 \, , \, x \, \rangle \, , \, \langle \, A^2 \, , \, x \, \rangle \, , \, \dots \, , \, \langle \, A^n \, , \, x \, \rangle \, \right] = \left[\, w_1 \, , \, w_2 \, , \, \dots \, , \, w_n \, \right] \in \mathbb{R}^{1 \times n}$$

and produces a (row) vector $w \in \mathbb{R}^n$, i.e., with the same size as rows. The computational cost of the operations is therefore $O(\, mn \,)$ in both cases, i.e., as the number of elements in A. This is obvious since each element of A is used only once. In fact, if A is sparse, i.e., its number of nonzero elements is $k \ll nm$, the operation can be implemented with cost $O(\, k \,)$ by just ignoring the zero elements, provided of course that the pattern of zero elements is known in advance.

Note that multiplying on the left we insist that x is a column vector, while multiplying on the right we rather transpose it and therefore assume it is a row vector. This is because the matrix-vector product is a special case of of the *matrix-matrix product*. This is possible between two matrices $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$, i.e., only of the rows of the leftmost matrix are of the same size as the columns of the rightmost one, and it basically consists in computing the scalar product between each row-column pair:

$$AB = \left[\begin{array}{cccc} \langle \, A_1 \, , \, B^1 \, \rangle & \langle \, A_1 \, , \, B^2 \, \rangle & \dots & \langle \, A_1 \, , \, B^n \, \rangle \\ \langle \, A_2 \, , \, B^1 \, \rangle & \langle \, A_2 \, , \, B^2 \, \rangle & \dots & \langle \, A_2 \, , \, B^n \, \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \, A_m \, , \, B^1 \, \rangle & \langle \, A_m \, , \, B^2 \, \rangle & \dots & \langle \, A_m \, , \, B^n \, \rangle \end{array} \right] \in \mathbb{R}^{m \times n} .$$

Of course, then, the matrix-vector product obeys the same rule; note that $Ax = z \in \mathbb{R}^{m \times 1}$, i.e., is a column vector like x, while $x^T A = z \in \mathbb{R}^{1 \times n}$, i.e., it is a row vector like x^T. The cost of computing the matrix-matrix product is $O(\, nmk \,)$ in general, unless A and/or B are sparse and their nonzero pattern is known in advance, hence it can be rather steep; indeed, matrix-matrix products will be carefully avoided in these lecture notes (at least in the practical implementation of algorithms). This is of course unless k is "small", as in the special case

$$\langle \, x \, , \, z \, \rangle = x^T z = z^T x$$

of the scalar product where $k = 1$, which is of course $O(\, n \,)$ (unless at least one among z and x is sparse). By dint of the fact that one usually tries to avoid to distinguish between row vectors and column vectors unless strictly necessary, in these lecture notes the scalar product will also sometimes be written as just "zx" or "xz" whenever this does not create (too much) confusion.

A.4 Eigenvalues and the determinant, in practice

Given a symmetric $Q \in \mathbb{R}^{n \times n}$, its eigenvalues $\lambda \in \mathbb{R}$—such that there exist the eigenvector $v \in \mathbb{R}^n$ satisfying $Qv = \lambda v$—are crucial to determine if Q is singular or not. In particular, the *rank* of Q is the number of eigenvectors having non-zero eigenvalues, and Q is nonsingular if and only it it is full-rank, i.e., the rank is n, i.e., it has no zero eigenvalues. As usual we denote by $\lambda_1, \dots, \lambda_n$ the eigenvalues of Q. Also, recall that if Q is nonsingular and $Qv = \lambda v$ then $(1 \, / \, \lambda)v = Q^{-1}v$, i.e., the eigenvalues of Q^{-1} are the inverse of those of Q.

One way to study non-singularity of Q is therefore that of considering its *determinant*

$$det(\,Q\,) = \lambda_1 \lambda_2 \dots \lambda_n = \Pi_{i=1}^{n} \lambda_i \ ,$$

i.e., the product of its eigenvalues. Then, Q is nonsingular if and only if $\lambda_i \neq 0$ for all $i = 1, \dots, n$, i.e., $det(\,Q\,) \neq 0$. Indeed, whenever Q is nonsingular

$$det(\,Q^{-1}\,) = \Pi_{i=1}^{n} (1 \,/\, \lambda^i) = 1 \,/\, det(\,Q\,) \ .$$

This clearly generalises the fact that, if $Q \in \mathbb{R}^{1 \times 1}$ is a single real number, the system $Qx = q$ is guaranteed to always have the solution $x = q \,/\, Q$ if and only if $Q \neq 0$. In fact, in this case $Q1 = Q$ means that Q is its own single eigenvalue corresponding to the "eigenvector" $v = 1$, and therefore $Q = det(\,Q\,)$; then, $Q^{-1} = 1 \,/\, Q$.

One may therefore want to be able to compute the determinant. This is indeed easy for very small values of n: for $n = 2$ (and Q symmetric as always)

$$det\left(\begin{bmatrix} Q_{11} & Q_{21} \\ Q_{21} & Q_{22} \end{bmatrix} \right) = Q_{11} Q_{22} - Q_{21}^2 \ .$$

This for instance allows to precisely characterise 2×2 positive (semi) definite matrices by

$$Q_{11} \geq 0 \quad , \quad Q_{22} \geq 0 \quad , \quad Q_{11} Q_{22} \geq Q_{21}^2 \ .$$

More general formulæ can be defined for any n, but they quickly spiral out of control due to the fact that they require to recursively compute the determinant of (many) smaller matrices. That is, one has to define Q^{ij} as the $(n-1) \times (n-1)$ matrix obtained from Q by removing row i and column j: then

$$\begin{aligned} det(\,Q\,) &= Q_{11} det(\,Q^{11}\,) - Q_{12} det(\,Q^{12}\,) + Q_{13} det(\,Q^{13}\,) \\ &= Q_{11}(\,Q_{22}Q_{33} - Q_{23}Q_{32}\,) - Q_{12}(\,Q_{21}Q_{33} - Q_{31}Q_{23}\,) + Q_{13}(\,Q_{21}Q_{32} - Q_{31}Q_{22}\,) \ . \end{aligned}$$

This *Laplace expansion* formula [57] can be generalised to any n and recursively applying the definition of the determinant, but this ends up with an exponential number of terms and it is therefore not suitable unless for extremely small values of n.

This is somewhat unfortunate due to the fact that the eigenvalues can be proven to be the roots of *characteristic polynomial*, i.e., the solutions to

$$det(\,Q - \lambda I\,) = 0$$

in the λ variable. This for instance allows to write the explicit formula of the eigenvalues of a 2×2 matrix, since

$$det\left(\begin{bmatrix} Q_{11} & Q_{21} \\ Q_{21} & Q_{22} \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right) = det\left(\begin{bmatrix} Q_{11} - \lambda & Q_{21} \\ Q_{21} & Q_{22} - \lambda \end{bmatrix} \right) = (Q_{11} - \lambda)(Q_{22} - \lambda) - Q_{21}^2 \ .$$

Solving the quadratic equation $det(\,Q - \lambda I\,) = 0$ then yields

$$\lambda = \tfrac{1}{2}\big(\,Q_{11} + Q_{22} \pm \sqrt{4Q_{21}^2 + (Q_{11} - Q_{22})^2}\,\big) \ .$$

Unfortunately, this kind of approach does not scale at all; it can be applied to 3×3 matrices, yielding rather ugly formulæ, due to the fact that root of cubic polynomial have a (complicated) closed formula [51], but not further unless Q has some very special structure. The only result that is worth mentioning here is that if Q if (upper or lower) triangular, which of course includes diagonal, then

$$det(\,Q\,) = \Pi_{i=1}^{n} Q_{ii} \ ,$$

i.e., the determinant is the product of the diagonal elements. For further details check, e.g., [2, A.5.2][8, p. 603][59][52].

Exercise A.1. Prove the last statement. **[solution]**

Exercise A.2. Find the eigenvalues of $Q = \begin{bmatrix} 6 & -2 \\ -2 & 6 \end{bmatrix}$ and $P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 6 & -2 \\ 0 & -2 & 6 \end{bmatrix}$. **[solution]**

A.5 Limits and optimization

The basic object here is a(n infinite) sequence $\{\,v_i\,\} = \{\,v_1\,,\,v_2\,,\,\dots\} = [\,v_i\,]_{i=1}^{\infty} \subset \mathbb{R}$ of real numbers. The sequence has a real limit $v \in \mathbb{R}$, written

$$\{\,v_i\,\} \to v \quad \text{or} \quad \lim_{i \to \infty} v_i = v$$

if "eventually v_i gets arbitrarily close to v and never leaves"; that is

$$\forall \varepsilon > 0 \, \exists \, h \text{ s.t. } | \, v_i - v \, | \leq \varepsilon \; \forall i \geq h \; .$$

Sequences may not have a limit, the obvious example is $v_i = (-1)^i$ that infinitely alternates between 1 and -1. Alternatively, a sequence may not have a real bur rather an extended real limit $\pm\infty$:

$$\lim_{i \to \infty} v_i = +\infty \quad \Longleftrightarrow \quad \forall M > 0 \, \exists \, h \text{ s.t. } v_i \geq M \; \forall i \geq h$$

$$\lim_{i \to \infty} v_i = -\infty \quad \Longleftrightarrow \quad \forall M > 0 \, \exists \, h \text{ s.t. } v_i \leq -M \; \forall i \geq h \; .$$

If a sequence is monotone, i.e., either $v_{i+1} \geq v_i$ for all i (increasing) or $v_{i+1} \leq v_i$ for all i (decreasing), then it surely has a limit, although possibly $\pm\infty$ [11, Theorem 4.1.6]. In these lectures, $\{ \, v_i \, \}$ is often the sequence of function values $\{ \, f^i \, \}$ of some optimization algorithm, and it is natural to call $\{ \, f^i \, \} \to f_*$ a *minimizing sequence* (in fact "minimising sequence" should be referred to se sequence of iterates x^i such that $f^i = (\, x^i \,)$, but in this context is directly used for the sequence of f-values).

In some cases it is not possible, or desired, to ensure that $\{ \, f^i \, \}$ is monotone. However, a non-monotone sequence can still exhibit useful behaviour when related to some optimization algorithm. Consider for instance $v_i = e^{(-1)^i i}$. If this were the f-values sequence of some algorithm for which $f_* = 0$, one could still consider it a minimising sequence in the sense that eventually one gets f-values as close to 0 as desired; only, when some iterate h is reached that $v_i \leq \varepsilon$, it is not guaranteed that *all* subsequent iterates have the same property. Similarly, if one were maximising a function unbounded above, it would be fair to say that one gets f-values "as close as possible to $+\infty$", if disregarding those iterates where f^i is instead "small" is not an issue. In other words, there can be weaker notion of convergence where one does *not* insist that *eventually all iterates become arbitrarily close to an optimal solution*, but rather than *some* do.

A useful mathematical notion that represents this choice is that of *limit inferior* and *limit superior*. This starts by "extract monotone sequences from $\{ \, v_i \, \}$ the hard way"; that is, define

$$\underline{v}_i = \inf\{ \, v_h \; : \; h \geq i \, \} \qquad , \qquad \overline{v}_i = \sup\{ \, v_h \; : \; h \geq i \, \} \; .$$

Since the set of v_h for $h \geq i$ reduces as i increases, it is obvious that $\{ \, \underline{v}_i \, \}$ is *nondecreasing* and $\{ \, \overline{v}_i \, \}$ is *nonincreasing*: as such, they both have a limit

$$\liminf_{i \to \infty} v_i = \lim_{i \to \infty} \underline{v}_i = \underline{v}_\infty \quad , \quad \limsup_{i \to \infty} v_i = \lim_{i \to \infty} \overline{v}_i = \overline{v}_\infty \; .$$

Since obviously $\underline{v}_i \leq \overline{v}_i$ for all i, it necessarily holds $\underline{v}_\infty \leq \overline{v}_\infty$; in fact, it can be proven that $\lim_{i \to \infty} v_i = v$ if and only if $\underline{v}_\infty = v = \overline{v}_\infty$, i.e., a sequence as limit if an only if it limit inferior and limit superior coincide. The operative definition for the limit inferior (the superior one is symmetric) is that *there exists a sub-sequence of* $\{ \, v_i \, \}$ *that converges to* $\underline{v}_\infty$. A sub-sequence is just obtained by taking an infinite increasing sequence of natural numbers $\{ \, n_i \, \}$ and considering $\{ \, v_i \, \}$ "restricted" to these indices; that is, the sequence $\{ \, r_i \, \} = \{ \, v_{n_i} \, \}$. Thus, "not all the numbers in the original sequence $\{ \, v_i \, \}$ converge to $\underline{v}_\infty$, but an appropriate (infinite) subset of them does". Thus, a sequence of f-values $\{ \, f^i \, \}$ such that $\liminf_{i \to \infty} f^i = \underline{f}_\infty = f_*$ can still be called a minimizing sequence with the intent defined above. These lecture notes will try to steer as clear as possible from this more complicated notion of convergence by focussing on monotone algorithms as much as possible, but the concept may turn up useful in some cases.

A final useful remark is that the definition of limit inferior is somehow "counterintuitive" in that it works "backwards" w.r.t. the one that would be natural in an optimization algorithm. If the sequence $\{ \, f^i \, \}$ is *not* monotone, it is trivial to make it so by considering that of *record values* $f^{i,rec} = \min\{ \, f^h \; : \; h \leq i \, \}$ (for a minimization problem). In plain words, one is just keeping track of the best f-value f^h found so far (at some iteration $i \leq h$, and typically of the corresponding iterate x^h that generated it). Obviously $\{ \, f^{i,rec} \, \}$ is monotone *nonincreasing*, while note that $\{ \, \underline{f}^i \, \}$ giving $\underline{f}_\infty$ is *nondecreasing*: in fact, in the former case the minimization is over the (known) *past* values (a set that increases with i), while in the latter case the minimization is over the (unknown) *future* values (a set that decreases with i). One could reasonably call $\{ \, f^i \, \}$ a minimizing sequence if the corresponding $\{ \, f^{i,rec} \, \} \to f_*$, on the ground that some iteration reaches as close as desired to the optimal value. However, this corresponds to a weaker notion of "convergence", as the following stylised example shows: consider $v_i = f^i = i$, with $f_* = 1$. Clearly, $f^{i,rec} = 1$ for all i, and such a $\{ \, f^i \, \}$ could then be called a minimising sequence. However, this corresponds to the notion "f-values close to f_* are generated at some point, but eventually they can become arbitrarily far away and never come back". By contrast, $\liminf_{i \to \infty} f^i = f_*$ corresponds to "f-values close to f_* are generated at some point; much worse ones can be generated later, but eventually the sequence always produce again close ones", and $\lim_{i \to \infty} f^i = f_*$ is the strongest one at "f-values close to f_* are generated at some point, and never become worse since".

However, all these notions rely on the fact that the sequence is infinite, and hence they are mainly theoretical. In practice the sequence of iterates, and their f-values, must necessarily be finite, and the hope is that algorithms will quickly identify a "good enough" solution and promptly stop. Yet, the issue is precisely that identifying when an iterate x^i is "good enough" is not always possible, and some algorithms need just to be let ran for a while in the (possibly, theoretically founded) hope that they will eventually provide "good enough" solutions. In this case, the difference between different forms of convergence—i.e., whether all iterated will eventually "cluster around an optimal solution" or only some of them do—becomes relevant.

For further details on sequences and limits check, e.g., [8, p. 623], [11, p. 33], [10, §3.1].

A.6 Continuity

We discuss continuity mainly with the viewpoint of univariate functions $f : \mathbb{R} \to \mathbb{R}$, but using a general notation so that the concepts can easily be generalised to the multivariate case. This starts by recall that the ball of center x and radius r in the univariate case is just $\mathcal{B}(\, x \,,\, r \,) = (\, z \in \mathbb{R} \,:\, |\, z - x \,| \leq r \,) = (\, z \in \mathbb{R} \,:\, x - r \leq z \leq x + r \,)$. Similarly, $\|\, x \,\|$ can basically only mean $|\, x \,|$ for $x \in \mathbb{R}$. Then, f *is continuous at* x [10, Definition 3.1.5] [11, §2.2] if, *however chosen* a sequence $\{\, x^i \,\} \to x$ one has $\{\, f^i = f(\, x^i \,) \,\} \to f(\, x \,)$. In other words,

$$\forall \varepsilon > 0 \; \exists \, \delta > 0 \quad \text{such that} \quad z \in \mathcal{B}(\, x \,,\, \delta \,) \implies \mathcal{B}(\, f(\, x \,) \,,\, \varepsilon \,) \;,$$

or more simply

$$\lim\nolimits_{i \to \infty} x^i = x \quad \implies \quad \lim\nolimits_{i \to \infty} f(\, x^i \,) = f(\, x \,) \;.$$

The intuitive concept is that f cannot "jump away from the expected behaviour" at x: if, for some sequence $\{\, x^i \,\}$ converging to x the sequence of f-values is converging to some value f^∞, then the function has no choice but to have $f(\, x \,) = f^\infty$. As a consequence, for a continuous function *the limits of all the sequences* $\{\, f(\, x^i \,) \,\}$ *for all the possible sequences* $\{\, x^i \,\} \to x$ *must be equal.* This makes it easy to see how a function can be not continuous: a simple but important one is

$$sign(\, x \,) = \begin{cases} -1 & \text{if } x < 0 \\ \;\;0 & \text{if } x = 0 \\ +1 & \text{if } x > 0 \end{cases} \;.$$

Clearly, $sign()$ is continuous for all x save $x = 0$. In fact, the example suggests the introduction of the useful concept of *left limit* and *right limit* (only for univariate functions)

$$\lim\nolimits_{x^i \to x_-} f(\, x^i \,) = f_- \quad \text{and} \quad \lim\nolimits_{x^i \to x_+} f(\, x^i \,) = f_+$$

where in the first case the sequence can only be chosen so that $x^i < x$, while in the second $x^i > x$. Obviously, for a continuous function $f_- = f_+$. Here, instead,

$$\lim\nolimits_{x^i \to 0_-} sign(\, x^i \,) = -1 < sign(\, 0 \,) = 0 < \lim\nolimits_{x^i \to 0_+} sign(\, x^i \,) = 1 \;.$$

That is, "from whatever direction 0 is approached", the function suddenly "jumps" to a value that "cannot be predicted by the behaviour in close points".

If f is continuous at all points of some set X then it is continuous on X. In order to avoid as much as possible to have to deal with "the set of points where the function is continuous", like for the domain, we will strongly favour functions *continuous everywhere* (on all $\mathbb{R}$, or in general $\mathbb{R}^n$); the set of all such functions will be defined (for reasons to become clear later on), C^0. Many "simple" functions belong to C^0, and continuity is easily preserved under many operations between functions: if f and g are C^0 (an useful shortcut for "belong to C^0"), then all of $f + g$, $f \cdot g$, $\max\{\, f \,,\, g \,\}$, $\min\{\, f \,,\, g \,\}$, and—most importantly—$f(g(\cdot))$ are C^0. Thus, continuity is not a hard property to come by. Of course there are plenty of simple non-continuous functions: the most obvious example is $1 \,/\, x$ at $x = 0$ (the left and right limits "cannot be more different", coming in at $-\infty$ and $+\infty$, respectively), but more contrived examples can be built such as $\sin(\, 1 \,/\, x \,)$ (the function keeps oscillating between -1 and 1 ever faster as $x \to 0$). Thus, continuity cannot always be given for granted.

Although it will not be used much at all in these lecture notes, we still remark that there are weaker forms of continuity that are sometimes useful. In particular, one says that f is *lower semi-continuous* (l.s.c.) at x if

$$\{\, x^i \,\} \to x \quad \implies \quad f(\, x \,) \leq \liminf\nolimits_{i \to \infty} f(\, x_i \,)$$

or, equivalently,

$$\liminf\nolimits_{z \to x} f(\, z \,) \geq f(\, x \,) \;.$$

Symmetrically, an *upper semi-continuous* (u.s.c.) function satisfies

$$\{\, x^i \,\} \to x \quad \implies \quad f(\, x \,) \geq \limsup\nolimits_{i \to \infty} f(\, x_i \,) \;,$$

i.e.,

$$\limsup\nolimits_{z \to x} f(\, z \,) \leq f(\, x \,) \;.$$

In plain words, a l.s.c. function "can jump down wildly, but can never jump up", and the converse holds for a u.s.c. one. Being l.s.c. is "convenient for minimization"; if $\{ x^i \}$ is a minimising sequence, i.e., $\{ f(x^i) \} \to f_*$, and $\{ x^i \} \to x$, then necessarily $f(x) = f_*$, i.e., x is optimal: in principle the function may jump down, but in a minimizing sequence it has "nowhere further down to go". The argument readily extends to an ε-minimising sequence ($(f^i) \to f^\infty \leq f_* + \varepsilon$). The symmetric of course holds for an u.s.c. one, which is convenient for maximization. It is easy to see that f is continuous at x if and only if it is *both* l.s.c. and u.s.c. at x, i.e., plain continuity is "good both for minimization and maximization"; we will therefore stick with continuity. The *sign*() example shows a function that is neither l.s.c. nor u.s.c. (at 0).

More interesting for our developments is the *stronger* notion of continuity that "limits how fast f can change close to x": f is said to be (locally) *Lipschitz continuous (L-c) at x with Lipschitz constant* $L > 0$ [8, p. 624] if there exists some $\varepsilon > 0$ such that

$$\| f(x) - f(z) \| \leq L \| x - z \| \qquad \forall z \in \mathcal{B}(x , \varepsilon) .$$

The two $\| \cdot \|$ in the definition allow to use it for the general case $f : \mathbb{R}^n \to \mathbb{R}^k$ even if $k > 1$: although such functions are not suitable to be objectives, they still have important uses throughout these lectures. In plain words, not only $f(z)$ must converge to $f(x)$ as z converges to x, as ordinary continuity requires; this has to happen "uniformly", i.e., if "how much z is close to x dictates how much $f(z)$ is close to $f(x)$". This basically means that f *cannot change too fast*.

Exercise A.3. Prove that f locally L-c at x imply f continuous at x. [**solution**]

Note that, in the definition, the constant L depends on x (it should be indicated as L_x to make it apparent, but we avoid it). A function can clearly be L-c with two different constants at x and at $z \neq x$. Of course, if f is L-c with constant $L > 0$, then it is also L-c with any constant $L' \geq L$. Hence, if it is L-c with constant (at most) L on all points of some set X then it is L-c on the whole set. If $X = \mathbb{R}$ ($\mathbb{R}^n$ in general), then f is said to be *globally L-c*. This is a much stronger notion than locally L-c, and a fortiori of continuity. However, this is a crucial notion to make optimization possible, or efficient.

Exercise A.4. Come up with f locally L-c everywhere but not globally L-c. [**solution**]

Exercise A.5. Come up with f continuous but not L-c on some bounded interval $X = [\underline{x} , \overline{x}]$. [**solution**]

A.7 (Univariate) Derivatives

Derivatives of univariate algebraic functions are easy to compute, since there is a powerful and complete algebra for this [10, Lecture 3] [11, §2.3]. For all the "basic" functions, the formula for the derivative is computed "from prime principles", i.e., applying those for limits [11, §2.1] to the very definition of derivative. This yields the derivatives of many simple functions, i.e.,

- $[x^k]' = k x^{k-1}$;
- $[e^x]' = e^x \quad , \quad [\ln(x)]' = 1 / x$;
- $[\sin(x)]' = \cos(x) \quad , \quad [\cos(x)]' = - \sin(x)$

Notably, derivatives of basic continuous functions are typically continuous themselves. Then, for many general functional operations formulæ provide the derivative in terms of these of the arguments, such as

- $[\alpha f(x) + \beta g(x)]' = \alpha f'(x) + \beta g'(x)$;
- $[f(x) \cdot g(x)]' = f'(x) \cdot g(x) + f(x) \cdot g'(x)$;
- $[f(x) / g(x)]' = [f'(x) \cdot g(x) - f(x) \cdot g'(x)] / g(x)^2$;
- $[f(g(x))]' = f'(g(x)) \cdot g'(x) \qquad$ (*chain rule*).

Again, most functional operations preserve continuous differentiability, but not all of them. Besides quotient, some important ones that don't are

$$\max\{ f(x) , g(x) \} \;\; , \;\; \min\{ f(x) , g(x) \} .$$

For more details see, e.g., [11, §2.3, Ex. 2.3.1, Th. 2.3.4 / 2.3.5], [10, §3.1]. The computation of derivatives for "very complex" algebraic functions, such as Deep Neural Networks, may nonetheless pose some interesting problems in terms of being able to *efficiently* compute them. There is, however, the whole field of study of *automatic differentiation* [17] devoted to solving these problems, in terms of both developing the requisite theory and implementing it in efficient, well-engineered. available software packages for many different programming languages / systems that can be effectively integrated with the applications that require derivatives computation.

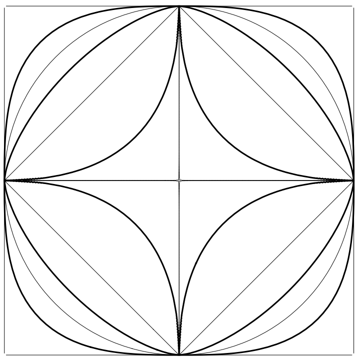
This is true in particular for Deep Neural Networks, whose layered structure allows layer-wise algorithms [13]. Thus, in these lecture notes we will just assume that "derivatives are efficiently computed somehow" and avoid to delve much further in the process.

A.8 Topology and limit in $\mathbb{R}^n$

When generalising the concept of limit—which itself is crucial for that of derivative—to $\mathbb{R}^n$, some aspects naturally become more involved. The point is that "there are many more ways for two points in $\mathbb{R}^n$ to be close to each other (or not)". This requires to formally state some *topological concepts* that serve to generalize the one seen in univariate optimization. This starts with the very easy one of *ball* of center $x \in \mathbb{R}^n$ and radius $r > 0$ in some norm

$$\mathcal{B}(\, x \,,\, r \,) := \{\, z \in \mathbb{R}^n \;:\; \|\, z - x \,\| \leq r \,\} \;,$$

i.e., the points "close" to x (in the chosen norm), that we have extensively used already in §A.6. There $n = 1$ and there is basically only one possible norm, but things are different when $n > 1$. Indeed, while it is very natural to imagine the Euclidean norm to be used in the definition above, any other norm satisfying the conditions set out in §A.2 will do, and there are very many different one.



In fact, the Euclidean norm just one member of the large family of *p-norms*, or L_p-norms, defined as

$$\|\, x \,\|_p := (\, \textstyle\sum_{i=1}^n |\, x_i \,|^p \,)^{1/p}$$

however chosen the scalar $p > 0$; obviously, $p = 2$ for the Euclidean one, which in fact is also known as the L_2 norm. The balls of radius 1 are depicted in the figure here on the right for $p = 0$, $1/24$,1,$3/24$, 2, 3, ∞. It is easy to see that the ball for smaller p is strictly included in that for larger p. Indeed, the L_∞ norm (obtained when $p \to \infty$)

$$\|\, x \,\|_\infty = \max\{\, |\, x_i \,| \;:\; i = 1, \dots, n \,\}$$

is the outermost square, as it is plain to see, while the L_0 norm

$$\|\, x \,\|_0 = \#\{\, i \;:\; |\, x_i \,| > 0 \,\}$$

that counts the number of nonzero entries in the vector x is the "very thin" cross in the centre. It should be noted that the L_0 norm does not satisfy all the requirements and it is therefore not a norm, but it is still called such due to being one of the two extremes element of the L_p family; it is also relevant in that it is the natural measure of the *sparsity* (or lack thereof) of the vector x. Although the first time this Appendix is referenced we have not yet formally defined the concept for multivariate functions, we also mention that the L_p norms are convex functions (and, therefore, the corresponding balls are convex sets, which we will define later) for $p \geq 1$, and nonconvex ones for $p < 1$. Hence, the L_1—also known as *Lasso*—norm

$$\|\, x \,\|_1 = \textstyle\sum_{i=1}^n |\, x_i \,|$$

it is relevant in that it is the *closest convex approximation of the L_0 norm*, and it is often used as such (i.e., as a convenient proxy for "the sparsity of x"). While there are other relevant norms than the L_p ones, the family already has many interesting properties, starting from the fact that p-norms come in pairs: $\|\cdot\|_p$ and $\|\cdot\|_q$ are called *dual* of each other if $1\,/\,p + 1\,/\,q = 1$ [2, A.1.6], and dual norms satisfy *Hölder's inequality*

$$|\, \langle\, x \,,\, z \,\rangle \,| \leq \|\, x \,\|_p \|\, z \,\|_q \;.$$

It is easy to see that for dual norms, p is "large" if and only if q is "small"; indeed, L_1 and L_∞ are dual. These are both convex and their balls are "particularly simple sets" (*polyhedra*, as we shall see), hence they are used in many applications. The only *self-dual* norm, i.e., for which $p = q$, is clearly the Euclidean one ($p = 2$), confirming its special role among norms. We will not make too much use of these concepts, though, so we don't delve any further.

In fact, for the matter at hand here, the exact choice of the norm is immaterial. Indeed, all concepts in topology are related to properties like "there exists a ball" or "for all small balls", as we shall see, but *all norms are topologically equivalent* [66]: however chosen two norms $\|\cdot\|$ and $|||\cdot|||$, there exist two constants $0 < \alpha < \beta$ such that $\alpha\|\, x \,\| \leq |||\, z \,||| \leq \beta\|\, x \,\|$ however chosen x and z in $\mathbb{R}^n$. Thus, for the subsequent treatment we can avoid specifying which norm—and distance—is actually intended.

Then, the definition of $\{\, x_i \,\} \to x$ in $\mathbb{R}^n$ can be stated as

$$\lim_{i \to \infty} x_i = x \quad \equiv \quad \forall \varepsilon > 0 \;\exists\, h \text{ s.t. } x_i \in \mathcal{B}(\, x \,,\, \varepsilon \,) \;\forall\, i \geq h \;.$$

In layman terms, the points of $\{\, x_i \,\}$ *eventually all* come arbitrarily close to x. In fact, the definition can be given in terms of distance as

$$\forall \varepsilon > 0 \; \exists \, h \text{ s.t. } d(\, x_i \,,\, x \,) \leq \varepsilon \; \forall \, i \geq h \quad \equiv \quad \lim_{i \to \infty} d(\, x_i \,,\, x \,) = 0 \; .$$

It should be noted, however, that $\mathbb{R}^n$ is "exponentially larger" than $\mathbb{R}$; that is, "there are many more ways for $\{\, x_i \,\} \to x$ in $\mathbb{R}^n$ than in $\mathbb{R}$". This requires care when some arguments are made, as we shall see.

A.9 Topology and feasibility

We now build upon the concept of §A.8 to define some properties of sets that are useful in the analysis of *constrained* optimization algorithms. In these lecture notes we keep a "light" approach to details and we will try to avoid to be too specific about these issues, but it makes sense nonetheless to at least mention these here. This starts with the definition of *interior* and *boundary* of an arbitrary set $S \subseteq \mathbb{R}^n$:

- $x \in int(\, S \,)$, i.e., is a point of the *interior of* S, if *there exists* a small enough ball centred in x and completely contained in S, i.e., any $r > 0$ such that $\mathcal{B}(\, x \,,\, r \,) \subseteq S$;
- $x \in \partial(\, S \,)$, i.e., is a point of the *boundary of* S, if *any* ball centred in x contains both points of S and points outside of S, that is, for each $r > 0$ there exist z and w, both belonging to $\mathcal{B}(\, x \,,\, r \,)$, such that $z \in S$ while $w \notin S$.

Note that by definition $x \in int(\, S \,)$ implies $x \in S$, but this is not true for points in the boundary. In fact, S is *open* if $S = int(\, S \,)$, i.e., "no points of S are on the boundary". One can then define the *closure of* S as $cl(\, S \,) = S \cup \partial S$, i.e., the set containing S and all the points on its boundary; then, S is *closed* is $S = cl(\, S \,)$, i.e., the closure operation does nothing since "all the points on the boundary of S already belong to it". It can be proven that S is closed if and only if its complement, $\bar{S} = \mathbb{R}^n \setminus S$, is open. More relevant for our application is the following characterisation: S is closed if and only if *all limit points of sequences in* S *also belong to* S. Formally, however chosen $\{\, x_i \,\} \subset S$ such that $\{\, x_i \,\} \to x$, one is guaranteed that $x \in S$. It can be seen why this is (theoretically) important when $S = X$: provided that all iterates remain feasible, one is sure that also any limit point is such.

It could be mentioned that many sets in $\mathbb{R}^n$ are "too thin" to have an interior; just picture for instance a segment, or a line, in $\mathbb{R}^2$. Having a point in the interior requires a "full dimensional ball" to be included in S, which means that S itself must be *full dimensional* (while a line in $\mathbb{R}^2$ has dimension 1, i.e., smaller than that of the space). We will not delve too much on these aspects, but let us mention that sometimes the concept of *relative interior* can be useful; picture all the points in the segment except the two extremes.

Thus, being closed is a very useful (in theory, necessary) property to ask to our feasible regions X; it is therefore reasonable to briefly discuss when this can be obtained, i.e., the *algebra of closed (and open) sets*. As usual, some sets can be easily seen to be closed (or open); this will be discussed in due time. The issue is therefore that of set operations that preserve closeness (or openness). The recap is simple enough:

- if $\{\, S_i \,\}$ is a family of (even infinitely many) open sets, then $\bigcup_i S_i$ is open;
- if S_1 and S_2 are open, then $S_1 \cap S_2$ is open;
- if $\{\, S_i \,\}$ is a family of (even infinitely many) closed sets, then $\bigcap_i S_i$ is closed;
- if S_1 and S_2 are closed, then $S_1 \cup S_2$ is closed.

We leave the proofs as exercises, together with simple facts such as: i) $\mathbb{R}^n$ and $\emptyset$ are *both* closed and open (hint: where's the boundary?); ii) the union of infinitely many closed sets need not be closed, and iii) the intersection of infinitely many open sets need not be open.

Closed sets are therefore preferred in optimization, especially if also *bounded*: $\exists \, r > 0$ such that $S \subseteq \mathcal{B}(\, 0 \,,\, r \,)$, i.e., "the set does not go all the way down to infinity". A closed and bounded set is called *compact*, and compactness equates to "piece of mind" for optimization (of continuous functions), at least theoretically, due to *Bolzano-Weierstrass Theorem* [49]. Let us recall that for a sequence $\{\, x_i \,\} \subset \mathbb{R}^n$ and a(n increasing) *index sequence* $\{\, n_i \,\}$, with $n_i \in \mathbb{N}$, the corresponding *subsequence* of $\{\, x_i \,\}$ is just $\{\, x_{n_i} \,\}$; in layman terms, only some of the elements of the original sequence. Then, x is an *accumulation point* of $\{\, x_i \,\}$ if *there exists a subsequence converging to it*: $\{\, x_{n_i} \,\} \to x$, i.e., $\liminf_{i \to \infty} d(\, x_i \,,\, x \,) = 0$. Clearly, this is a weaker requirement than the whole of $\{\, x_i \,\}$ converging to x: some iterates x_i may "stray away" from x occasionally, but "sooner or later a subsequent iterate will be again close to x". Now, assume that $\{\, x_i \,\}$ is a minimizing sequence, which in particular means that all iterates are feasible. Bolzano-Weierstrass Theorem states that if X is compact, then there exists a subsequence $\{\, x_{n_i} \,\}$ that converges to some x_*; obviously $x_* \in X$ by closeness, and therefore clearly x_* is a candidate at being an optimal solution (it surely is if, e.g., $\{\, f(\, x_i \,) \,\}$ is monotone). In other words, "optimization is easy with a compact feasible region": *all you need to do is to generate a minimizing sequence*, and—taking a subsequence if necessary—that will "automatically converge to some optimal solution". Of course, this is only a very abstract statement: constructing a minimizing sequence, in particular for a nonconvex problem, remains an extremely challenging task.

All this discussion is largely theoretical, though, due to the oft-repeated fact that "there is no sharp boundary when using floating-point numbers". So, for instance, one may have $X = (\,0\,,\,1\,)$, a clearly open set, and a sequence of feasible iterates $\{\,x_i\,\} \rightarrow 1$ that "points towards" an unfeasible extreme. While this would be a serious issue mathematically, it has been already discussed in §1.1.5 why this is not so in practice. Thus, we close this section by making our stance explicit: *as far as we are concerned, all sets are closed.* And this is it.

Solutions

- **Solution to Exercise A.1:** The only nonzero in the first row is Q_{11}, hence all the other terms vanish; repeat the argument in Q^{11}.

- **Solution to Exercise A.2:** We want to solve $det(\,Q - \lambda I\,) = (6-\lambda)(6-\lambda) - 4 = \lambda^2 - 12\lambda + 32 = 0$, a quadratic equation whose discriminant is $\Delta = 12^2 - 4\cdot 1\cdot 32 = 16$. Since $\sqrt{\Delta} = 4$, $\lambda_{1,2} = (\,-(-12)\pm 4\,)\,/\,2 = \{\,8\,,\,4\,\}$; this is confirmed by the provided closed formula. Now, since P is block-diagonal, applying the Laplace expansion one has $det(\,P\,) = P_{11} det(\,P^{11}\,) = 1 \cdot det(\,Q\,)$, hence $det(\,P - \lambda I\,) = (\,1 - \lambda\,) det(\,Q - \lambda I\,)$. Thus, the eigenvalues of P are the same as these of Q with the addition of $\lambda_3 = 1$.

- **Solution to Exercise A.3:** Take $\delta = \varepsilon\,/\,L$; then, for all $z \in \mathcal{B}(\,x\,,\,\delta\,)$ one has $|\,f(\,z\,) - f(\,x\,)\,| \leq L\|\,x - z\,\| \leq L\delta \leq L(\,\varepsilon\,/\,L\,) = \varepsilon$.

- **Solution to Exercise A.4:** Note: we'll have a much simpler proof later, after we present the relationships between L-c and derivatives. $f(\,x\,) = x^2$ is locally but not globally L-c; we prove this for $x \geq 0$, but the same arguments work for $x \leq 0$ (the function is symmetric). If $\delta > 0$ then $0 \leq f(\,x + \delta\,) - f(\,x\,) = (\,2x + \delta\,)\delta \leq 3x\delta$ if $\delta \leq x$; hence, f is L-c at x with Lipschitz constant $3x$ in some (right) interval around x. However, $\delta > 0$ also gives $f(\,x + \delta\,) - f(\,x\,) = (\,2x + \delta\,)\delta \geq 2x\delta$; hence, f cannot be L-c at x with Lipschitz constant less than $2x$, and that value is not bounded as $x \rightarrow \infty$. Symmetric arguments works for left intervals $(f(\,x\,) - f(\,x - \delta\,))$.

- **Solution to Exercise A.5:** The standard example is $f(\,x\,) = \sqrt{|\,x\,|}$, which is easily verified to be continuous and to become "infinitely steep" as $x \rightarrow 0$, because it is the inverse function of $y = x^2$ (for $x \geq 0$), and x^2 becomes "infinitely flat" as $x \rightarrow 0$. Again, we'll have a much simpler proof later, after we present the relationships between L-c and derivatives; in fact, the proof is so much simpler that it is not worth proceeding now, we just wait until we have the right tools.