1 General Introduction

We use the following notation: the working space is $\mathbb{R}^n$, where the scalar product will be denoted indifferently by (x, y) or $\langle x, y\rangle$ or $x^\top y$ (actually, it will be the usual dot-product: $(x, y) = \sum_{i=1}^n x^i y^i$); $|\cdot|$ or $\|\cdot\|$ will denote the associated norm. The *gradient* (vector of partial derivatives) of a function $f : \mathbb{R}^n \to \mathbb{R}$ will be denoted by ∇f or f'; the *Hessian* (matrix of second derivatives) by $\nabla^2 f$ or f''. We will also use continually the notation $g(x) = f'(x)$.

1.1 Generalities on Optimization

1.1.1 The Problem

Given a set X and a function $f : X \to \mathbb{R}$ (the *objective function*), we want to find $x^* \in X$ such that, for all $x \in X$, there holds $f(x) \geqslant f(x^*)$. The variable x is usually called *decision* or *control* variable.

We will consider only the case where X is a subset of $\mathbb{R}^n$, defined by *constraints*, i.e., given a number $m_I + m_E$ of functions $c_j : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, \ldots, m_I + m_E$, the problem is

$$\begin{cases} \min f(x) & x \in \mathbb{R}^n \\ c_j(x) \leqslant 0 & j \in I \\ c_j(x) = 0 & j \in E\,. \end{cases} \tag{P}$$

Here, I and E are two disjoint sets of integers, of cardinalities m_I and m_E respectively. We thus have m_I *inequality* constraints, indexed in I, and m_E *equality* constraints, indexed in E.

Remark 1.1. We do not consider problems of *combinatorial optimization*, where the set X is discrete, or even finite. They could be covered by our formalism via constraints of the type $x^i(1 - x^i) = 0$ (to express $x^i \in \{0, 1\}$) but this is very artificial – and not at all efficient in general. Actually, combinatorial optimization problems call for methods totally different from those presented in this book. Their intersection is not totally empty, though: §8.2 will mention the use of continuous optimization to bound the optimal value in combinatorial problems. Section 1.2.4 will give an illustrative example.

In another class of problems, the vector-variable $x \in \mathbb{R}^n$ becomes a function of time $x(t), t \in [0,T]$: these are *optimal control* problems. They are close to our formalism, possibly after discretizing $[0,T]$; in fact, examples are given in §1.2.2 and 1.2.3.

Perhaps rather paradoxically, the methods in this book extend easily to optimal control problems, while they fit very badly to combinatorial optimization. □

1.1.2 Classification

Among the various possible classifications, the following is made according to the difficulty of the problem to solve.

1. Unconstrained problems ($m_I = m_E = 0, I = E = \emptyset$)
   1.1 Quadratic problems: $f(x) = \frac{1}{2}(x, Mx) - (b, x)$ (M symmetric $n \times n$)
   1.2 Nonlinear problems: f neither linear nor quadratic.
2. Linearly constrained problems (the functions c_j are affine)
   2.1. Problems with equality constraints only ($m_I = 0$, $I = \emptyset$)
      2.1.1 Linear-quadratic problems: f quadratic
      2.1.2 Nonlinear problems: f neither linear nor quadratic
   2.2 Problems with inequality constraints
      2.2.1 Linear programming: f linear (needs $m_I \geqslant n - m_E$)
      2.2.2 Linear-quadratic problems: f quadratic
      2.2.3 Linearly constrained nonlinear problems.
3. Nonlinear programming
   3.1 With equality constraints only
   3.2 General nonlinear programming.

Observe that

– in optimization, the word "linear" is frequently (mis)used, instead of affine (see 2; recall that an affine function is the sum of a linear function and a constant term);
– 2.1 is the minimization in a hyperplane, isomorphic to a subspace of dimension $n - m_E$, so that 2.1 is equivalent to 1, at least theoretically;
– 1.1 reduces to solving a linear system ($Ax = b$ – at least if A is positive definite); 2.1.1 as well, in view of the preceding remark;
– 2.2 minimizes f in a convex polyhedron, the simplest being a parallelotope, defined by simple bounds: $a^i \leqslant x^i \leqslant b^i$, for $i = 1, \ldots, n$;
– 2.2 is considerably more complicated than 2.1, simply because one does not know in advance which inequalities will play a role at the optimal point. Said otherwise, there are 2^{m_I} ways of putting a problem 2.2 into the form 2.1; the question is: which is the correct one? An inequality constraint is said to be *active* at x (not necessarily optimal) when $c_j(x) = 0$. To put 2.2 into the form 2.1, one needs to know which constraints will be active at the (unknown!) optimum point.

1.2 Motivation and Examples

In this section, we show with some examples the variety of domains where one finds optimization problems considered in the present book. Since problems of the linear type (categories 2.2.1 and 2.2.2 in §1.1.2, described in the fourth part) have existed for a long time, and are well known, it is not necessary to motivate this branch. This is why the four examples below are of the "general" nonlinear type.

1.2.1 Molecular Biology

An important problem in biochemistry, for example in pharmacology, is to determine the geometry of a molecule. Various techniques are possible (X-ray crystallography, nuclear magnetic resonance,...) one of these is convenient when

– the chemical formula of the molecule is known,
– the molecule is not available, making it impossible to conduct any experiment,
– one has some knowledge of its shape and one wants to *refine* it.

The idea is then to compute the positions of the atoms in the space that minimize the associated potential energy. Let N be the number of atoms and call $x_i \in \mathbb{R}^3$ the spatial position of the i^{th} atom. To the vector $X = (x_1, \ldots, x_N) \in \mathbb{R}^{3N}$ is associated a potential energy $f(X)$ (the "conformational energy"), which is the sum of several terms. For example:

– Bond length: between two atoms i and j at distance $|x_i - x_j|$, there is first an energy of the type

$$L_{ij}(x_i, x_j) = \lambda_{ij}(|x_i - x_j| - d_{ij})^2 \,.$$

– There is also a Van der Waals energy, say

$$V_{ij}(x_i, x_j) = v_{ij}\Big(\frac{\delta_{ij}}{|x_i - x_j|}\Big)^6 - w_{ij}\Big(\frac{\delta_{ij}}{|x_i - x_j|}\Big)^{12} \,.$$

Here, the $\lambda_{ij}, v_{ij}, w_{ij}, d_{ij}, \delta_{ij}$'s are known constants, depending on the pair of atoms involved (carbon-carbon, carbon-nitrogen, etc.)

– Valence angle: between three atoms i, j, k forming an angle θ_{ijk} (writing down the value of θ_{ijk}, as a function of x_i, x_j, x_k, is left as an exercise!), there is an energy

$$A_{ijk}(x_i, x_j, x_k) = \alpha_{ijk}(\theta_{ijk} - \bar{\theta}_{ijk})^2 \,,$$

where, here again, α_{ijk} and $\bar{\theta}_{ijk}$ are known constants.

Other types of energies may also be considered: electrostatic, torsion angles, etc. The total energy is then the sum of all these terms, over all pairs/-triples/quadruples of atoms. The important thing to understand here, is that this energy can be computed (as well as its derivatives) for any numerical values taken by the variables x_i. And this is true even if these values do not correspond to any reasonable configuration; simply, the resulting energy will then be unreasonably large (if the model is reasonable!); the optimization process, precisely, will aim at eliminating these values.

This is obviously a problem from category 1.2 in §1.1.2. Note that the objective function is disagreeable:

– With its many terms, it is long to compute.
– With its strong nonlinearities, it does not enjoy the properties useful for optimization: it is definitely not quadratic, and not even convex. Actually, in most examples there are many equilibrium points X^* (local minima); this is why the only hope is to *refine* a specific one: by assumption, some estimate X_0 is available, close to the sought "optimal" X^*. Otherwise the optimization algorithm could only find some uncontrolled equilibrium, "by chance".

Such a problem will call for methods from the first part of this book, more precisely §4.4. Actually, since nowaday's "interesting" molecules have 10^3 atoms and more, this problem is also large-scale; as a result, it will rather be necessary to use methods from Sections 5.6, 6.3, or also 6.4.

1.2.2 Meteorology

To forecast the weather is to know the state of the atmosphere in the future. This is quite possible, at least theoretically (and within limits due to the chaotic character of phenomena involved). Let $p(z,t)$ be the state of the atmosphere at point $z \in \mathbb{R}^3$ and time $t \in [0,7]$ (assuming a forecast over one week, say); p is actually a vector made up of pressure, wind speed, humidity... The evolution of p along time can be modeled: avoiding technicalities, fluid mechanics tells us that

$$\frac{\partial p}{\partial t}(z,t) = \Phi(p(z,t))\,, \tag{1.1}$$

where Φ is a certain differential operator. For example, (1.1) could be the Navier-Stokes equation, but approximations are generally introduced.

To forecast the weather once our model Φ is chosen, it "suffices" to integrate (1.1). For this, initial conditions are needed (the question of boundary conditions is neglected here; for example, we shall say that they are periodicity conditions, (1.1) being integrated on the whole earth). Here comes optimization, in charge of estimating $p(\cdot,0)$ via an *identification* process, which we roughly explain.

In fact, the available information also contains all the meteorological observations collected in the past, say during the preceding day. Let us denote

by $\Omega = \{\omega_i\}_{i\in I}$ these observations. To fix ideas, we could say that each ω_i represents the value of p at a certain point (z_i, t_i) (but actually, only some coordinates of the vector $p(z_i, t_i)$ are observed). To take these – noisy – data into account, a natural and well-known idea is to consider the problem

$$\min_p \|p - \Omega\|\,, \tag{1.2}$$

(1.1) being considered as a constraint (called in this context the *state equation*).

– Observe here that our optimization problem is not posed with respect to some $x \in \mathbb{R}^n$ but to p, varying in a functional, infinite-dimensional, space. See Remark 1.1; we are dealing with an optimal control problem. Notwithstanding, any numerical implementation implies first a discretization, which reduces the problem to the framework of this book.
– Note also that (1.1) is a priori valid on the whole interval $[-1, +7]$, but (1.2) concerns $[-1, 0]$ only. Actually, optimization just deals with this latter interval; it is only for the forecast itself, after optimization is finished, that the interval $[0, 7]$ will come into play.
– Since p and Ω do not live in the same space (the number $|I|$ of observations, possibly very large, is certainly finite), Ω must first be embedded in the same function space as p. Besides, the norm $\|\cdot\|$ in (1.2) must be carefully chosen. These aspects, which concern modeling only, have a big influence on the behaviour of solution algorithms.

At this point, it is a good idea not to view (1.1), (1.2) as a nonlinearly constrained optimization problem (category 3.2 in §1.1.2), but rather as an unconstrained one (category 1.2). In fact, call $u(z) = p(z, -1)$ the state of the atmosphere at z, at initial time $t = -1$. A fundamental remark is then: assuming u to be known, (1.1) gives unambiguously $p(z, t) = p_u(z, t)$ for all z and all $t \geqslant -1$: the unknown p_u depends on the variable u *only*. Hence, the objective value in (1.2) also depends on u *only*. Our problem can therefore be formulated as $\min_u \|p_u - \Omega\|$, which means:
– to minimize with respect to u (unconstrained variable)
– the function defined by (1.2),
– where $p = p_u$ is obtained from (1.1)
– via the initial condition $p(\cdot, -1) = u$.

The actual decision variable in this formulation is u indeed: p plays only the role of a parameter, called *state variable*, while the terminology *control variable* is here reserved to u. The objective function will be denoted by $J(u)$, rather than $f(x)$. Thus, the number of variables is reduced (drastically: passing from about 10^9 for p, to about 10^7 for u alone) and, more importantly, any form of constraint is eliminated.

Remark 1.2. The "normal", *direct*, problem is to compute $p(z, t)$ from $p(z, 0)$ via (1.1). Here we solve the *inverse* problem: to compute $p(z, 0)$ from (a partial knowledge of) $p(z, t)$.

The above description is of course very sketchy and does not reveal all the difficulty of the problem. For instance: the number of observations is about 10^5, which is by far insufficient to identify the 10^7 unknowns. To orient the search toward reasonable p_u's, any a priori information on the stationary solutions to (1.1) is an important element, which is taken into account in actual implementations. □

Here again, the methods from the first part of this book will be used. The problem is more than ever large-scale: after discretization, $u \in \mathbb{R}^{10^7}$; calling for §6.3 therefore becomes a must.

1.2.3 Trajectory of a Deepwater Vehicle

Most optimal control problems consist in optimizing a trajectory; an example is towing a submarine vehicle. Consider a deepwater observation device (the "fish"), moving close to the sea bottom, and pulled from the surface by a tug. The problem is to control the tug so that the fish makes a given maneuver, while avoiding obstacles. For example, one may ask to make a U-turn in minimal time.

Let L be the length of the pulling cable. One may assume that L is a known constant, or that the cable is inextensible; anyway L is for this problem several kilometers long, and one cannot assume that the cable behaves like a rigid rod. As a result, the fish's trajectory is a rather complicated function of the tug's. A possible model is as follows.

– Let $y(s,t) \in \mathbb{R}^3$ be the position in the sea of a point at time t and (curvilinear) coordinate $s \in [0, L]$ along the cable.
– Then $y(0,t)$ is the tug's position, it is the control variable; $y(L,t)$ is the fish's, it is the variable to be controlled.
– These two variables are not independent: from inextensibility, we have

$$\left\| \frac{\partial y}{\partial s} \right\| = 1 \tag{1.3}$$

and y obeys the state equation

$$\frac{\partial^2 y}{\partial t^2} - \frac{\partial}{\partial t}\Big(T(s,t)\frac{\partial y}{\partial s}\Big) + \tau\Big(\frac{\partial y}{\partial t}\Big) = w\,. \tag{1.4}$$

Here T is the cable's tension (unknown), w its linear weight rate and τ models the drag.
– In addition to this system of equations, there are appropriate initial and boundary conditions, among which $y(0,t) = u(t)$, which simply expresses that $y(0,\cdot)$ plays a special role (the control!).

Just as in §1.2.2, we are again faced with an optimal control problem: the objective function (for example the time needed to make a U-turn) depends

on the control u implicitly, via a state (y_u, T_u), solution to a state equation. However, the situation is no longer as "simple"(!) as in §1.2.2: we still have to express that the fish must evolve above the sea bottom, which yields constraints on the state: if $\varphi(z^1, z^2)$ is the height of free water at $z \in \mathbb{R}^2$, one must impose

$$y^3(L,t) \geqslant \varphi(y^1(L,t), y^2(L,t)), \quad \text{for all } t. \tag{1.5}$$

These constraints in turn depend implicitly on u, and they are actually infinitely many (i.e. many, after discretization). As a result, it is hardly possible to "reduce" the problem with respect to u only. We now have to call for the third part of this book (constrained nonlinear optimization): the distinction between control and state variables is no longer relevant. In the sense of §1.1.1, the decision variables are now the couple (y, T), with respect to which one must

– minimize a certain function $f(y)$ (for example the time of the U-turn)
– under equality constraints $c_j(y, T) = 0$, $j \in E$, which symbolize the state equations (1.3), (1.4) (here E is big)
– and inequality constraints $c_j(y) \leqslant 0$, $j \in I$, which symbolize constraints on the state (1.5) (and I is just as big).

This example illustrates, among other things, the ambiguity which can exist concerning the decision variables: in the sense of optimal control, the control variable is u; however, the optimization algorithm "sees" as decision variable the whole of (y, T). Of course, the algorithm designer is allowed – and even strongly advised – to remember the origin of the problem, and to let $y(0, \cdot)$ play a particular role in the complete set of variables $\{(y, T)(s, t)\}_{s,t}$.

1.2.4 Optimization of Power Management

We complete this list of examples with a problem having nothing to do with the preceding : to optimize the production of electrical power plants. The following constitutes a simplest instance among realistic models. Consider a set I of power plants (hydro-electrical, thermal, nuclear or not). One wishes to optimize their production over a horizon $\{1, \dots, T\}$, for example $T = 48$ half-hours; the demand is supposed to be known, call it $d_1, \dots, d_T$. If p_t^i denotes the energy produced by the production unit $i \in I$ during the period t, one must first satisfy the demand constraints

$$\textstyle\sum_{i \in I} p_t^i \geqslant d_t, \quad \text{for } t = 1, \dots, T. \tag{1.6}$$

Use the notation $p^i = \{p_1^i, \dots, p_T^i\}$ for the production-vector of unit i. To each unit is associated a production cost $c^i : \mathbb{R}^T \to \mathbb{R}$: one wishes to solve

$$\min \textstyle\sum_{i \in I} c^i(p^i). \tag{1.7}$$

Besides, each unit has its own *technological constraints* describing the set D^i of possible production vectors:

$$p^i \in D^i\,, \quad \text{for } i \in I\,. \tag{1.8}$$

Describing the c^i's and D^i's may not be a simple task, which goes beyond our framework. We just note here their *disparity*: nuclear and hydro plants have nothing to do with each other, neither in their operation costs, nor in their constraints. For one thing, a hydro plant has basically linear characteristics (category 2.2.1 in §1.1.2), although it becomes nonlinear (category 3.2) in accurate models. By contrast, thermal plants have an important combinatorial aspect, owing to a $0-1$ behaviour: it is not possible to change their production level continuously, neither at any time.

The crude problem is to minimize (1.7) under constraints (1.6), (1.8). This problem is large-scale: as an example, the French power mix has about 200 plants working every day, which gives birth to $200 \times 48 = 10^4$ variables p_t^i (and even many more, due to combinatorics; actually, each unit i is an optimal control system, with its own additional state variables). Yet, the real difficulty of the problem is not its size but its heterogeneity: nonlinear methods of this book will fail, just as combinatorial methods.

This is why it is suitable to transform this problem. The key is to observe that, if constraints (1.6) were not present, each plant could be treated separately: one would have to solve, for each $i \in I$

$$\min c^i(q)\,, \quad q \in D^i\,. \tag{1.9}$$

Here, the dummy variable q represents the production-vector p^i. Each of the latter problems becomes solvable, by a method tailored to each case, depending on i. Starting from this remark, a particular heuristic technique is rather well-suited for (1.6)–(1.8). More precisely, Lagrangian relaxation (§8.2) approximates a solution by minimizing a convex nonsmooth function, to be seen in Chap. 10.

1.3 General Principles of Resolution

The problems of interest here – such as those of §1.2 – are solved via an algorithm which constructs iteratively $x_1, x_2, \ldots, x_k, \ldots$ To obtain the next iterate, the algorithm needs to know some information concerning the original problem (P) of §1.1.1: essentially, the numerical value of f and c for each value of x; often, their derivatives as well.

– If there are only linear or quadratic functions, this information is globally and explicitly available in the data: a linear [resp. quadratic] function (b, x) [resp. (x, Ax)] is completely characterized by the vector b [resp. the matrix A]. As a result, categories 1.1, 2.1.1, 2.2.1, 2.2.2 of § 1.1.2 make up a very particular class, and call for very particular methods, studied in the fourth part of this volume.

– By contrast, as soon as really general functions are involved, this information is computed in a *black box* (subprogram) characterizing (P), and independent of the selected algorithm. This subprogram can be called *simulator*, since it simulates the behaviour of the problem under the action of the decision variables (optimal or not).

Hence (and it is important to convince oneself with this truth), a computer program solving an optimization problem is made up of *two distinct parts*:

– One is in charge of managing x and is the algorithm proper; call it (A), as Algorithm; it is generally written by a mathematician, specialized in optimization.
– The other, the simulator, depending on (P), performs the required calculations for each x decided by (A); it is generally written by a practitioner (engineer, physicist, economist, etc.), the one who wishes to solve the specific optimization problem.

The distinction between (A) and (P) is not always straightforward, actually it depends on the modeling. Consider the examples of the preceding section:

§1.2.1. There is no ambiguity in the biochemistry problem: (A) places the atoms in the space, (P) computes the resulting energy, and perhaps its derivatives as well: they are very useful for (A).

§1.2.2. The case of meteorology is also relatively clear: (A) decides the initial conditions (denoted by u or $p(\cdot,-1)$ rather than x); (P) integrates the state equation over $[-1,0]$, which allows the computation of the objective function (1.2); call $J(u)$ this objective. Note that differentiating J is now far from trivial; yet, it is certainly possible (at least after discretization, in case of theoretical difficulties for the continuous version). More is given on this topic in §1.6 below.

§1.2.3. In the cable problem the situation is no longer so clear-cut. In a control-like formulation as in §1.2.2, (A) would decide the tug's trajectory, and (P) would integrate (1.3), (1.4) to obtain the fish's trajectory; the objective value and the constraint value (1.5) would ensue.

In the suggested "general-constrained" formulation, (A) fixes the trajectory and tension of every point on the cable. The job of (P) is now much more elementary: it knows the values of $(y,T)(s,t)$ for each (s,t) – they have been fixed by (A) – and it just have to compute the values (and derivatives) of the objective, of the equality constraints (1.3), (1.4), and of the inequality constraints (1.5).

§1.2.4. A complication appears in production optimization because the problem is not really (1.6)–(1.8), but rather an auxiliary abstract problem, which will be seen in §8.3.2. The objective is actually a perturbation of (1.7), namely a *Lagrange function* incorporating the term $\sum_t \lambda_t\big(\sum_i p_t^i - d_t\big)$; the decision variables are no longer the p_t^i's

but the λ_t's, i.e. the multipliers associated with (1.6). Thus, (A) fixes the λ_t's, while (P) solves for each i a perturbation of (1.9), namely

$$\min_{q \in D^i} c^i(q) + \sum_t \lambda_t q_t \, .$$

Remark 1.3. In addition to the (A)–(P) distinction, another fundamental thing to understand here is the following: for any problem considered, the only information available for (P) is the result of a numerical calculation, generally complicated; for example, the resolution of a partial differential equation, or the optimization of a number of nuclear plants, etc. Hence, (A) has to proceed by "trial and error": it assigns trial values to the decision variables x, and it corrects these values upon observation of the answer from (P); and this will repeatedly make up the iterations of the optimization process. □

Now the current iteration of an optimization algorithm is made up of two phases: to compute a direction, and to perform a line-search.

– Computing a direction: (P) is replaced by a model (P_k), which is simpler; then (P_k) is solved to yield a new approximation $x_k + d$.
– Line-search: a stepsize $t > 0$ is computed so that $x_k + td$ is "better" than x_k in terms of (P).
– The new iterate is then $x_{k+1} = x_k + td$.

Remark 1.4. The direction is computed by solving (usually accurately) an *approximation* (P_k) of (P). By contrast, the stepsize is computed by observing the *true* (P) on the restriction of $x \in \mathbb{R}^n$ to the half-line $\{x_k + td\}_{t \in \mathbb{R}_+}$ (x_k and d fixed).

Replacing the given problem (P) by a simpler (P_k) is a common technique in numerical analysis. By contrast, the second phase which corrects $x_k + d$, is a technique specific to optimization. Its motivation is *stabilization*. All this will be seen in detail in the next chapters. □

The next two subsections are devoted to some convergence theory tailored to optimization algorithms.

1.4 Convergence: Global Aspects

Let an optimization algorithm generate some sequence $\{x_k\}$. This algorithm is said to converge *globally* when

$\{x_k\}$ converges to "what is wished" for any initial iterate x_1.

Caution: this terminology is ambiguous because "what is wished" does not mean a solution to the initial problem (P), often called *global optimum*. Here, one rather stresses the fact that the initial iterate can be arbitrarily far from

"what is wished", without impairing convergence; actually, "what is wished" generally means an x satisfying what is called the necessary optimality conditions (see below and the sections involved: §§2.2 and 13.3).

In connection with Remark 1.4, one generally has a *merit function* $\Theta : \mathbb{R}^n \to \mathbb{R}$, which is minimal at "what is whished": (P) is thus equivalent to minimizing Θ over the *whole* of $\mathbb{R}^n$. The simplest example is unconstrained optimization: one must minimize f over $\mathbb{R}^n$, so one naturally takes $\Theta = f$. The word "better" introduced in §1.3 can then be given the meaning

$$\Theta(x_{k+1}) < \Theta(x_k)\,. \tag{1.10}$$

Then let us review the various convergence properties that an optimization algorithm may enjoy. First, a direct consequence of (1.10) is that

$$\{\Theta(x_k)\} \text{ has a limit, possibly } -\infty$$

– of course, $\Theta(x_k) \to -\infty$ reveals an ill-posed problem (P).

Minimal requirement To make things simple, let us assume that Θ is a continuously differentiable function and consider its first-order development around a given x:

$$\Theta(x+h) \simeq \Theta(x) + (\nabla\Theta(x), h)\,.$$

Assuming $\nabla\Theta(x) \neq 0$ and taking $h = -t\nabla\Theta(x)$ with a small $t > 0$, we obtain $\Theta(x+h) - \Theta(x) \simeq -t|\nabla\Theta(x)|^2 < 0$; as a result, x cannot minimize Θ. We say that $\nabla\Theta(x) = 0$ is an *optimality condition* for x to minimize Θ. The least property that should be satisfied by a sequence $\{x_k\}$ constructed as in §1.3 is then[1]

$$\liminf |\nabla\Theta(x_k)| = 0\,; \tag{1.11}$$

this means that the gradient $\nabla\Theta(x_k)$ will certainly have a norm smaller than ε for some finite k, no matter how $\varepsilon > 0$ is chosen. Thus, in this context, a globally convergent algorithm has to satisfy (1.11) for any starting point x_1.

It should be noted that (1.11), or even the property $\lim |\nabla\Theta(x_k)| = 0$, is fairly weak indeed: it does not tell much unless $\{x_k\}$ itself has some limit point. For example, it does not imply that $\{x_k\}$ is a *minimizing sequence*, i.e. that $\Theta(x_k) \to \inf\Theta$.

Boundedness If the original minimization problem (P) is reasonably well-posed, a reasonable merit function satisfies

$$\Theta(x) \to +\infty \quad \text{when} \quad |x| \to +\infty$$

(for example, minimizing e^x over $x \in \mathbb{R}$ is an ill-posed optimization problem: it has no solution). Together with (1.10), this property automatically guarantees that $\{x_k\}$ is a bounded sequence. As a result, $\{x_k\}$ has a cluster point; and every subsequence $\{x_k\}_{k\in K}$ is also bounded.

[1] The lim inf [resp. lim sup] of a numerical sequence is its smallest [resp. largest] cluster point.

Convergent sequences Assume boundedness of $\{x_k\}$. Then (1.11) guarantees the existence of a subsequence $\{x_k\}_{k\in K}$ satisfying

$$x_k \xrightarrow{k\in K} x^* \quad \text{and} \quad \nabla\Theta(x_k) \xrightarrow{k\in K} 0\,,$$

from wich continuity of $\nabla\Theta$ implies $\nabla\Theta(x^*) = 0$.

On the other hand, the monotonicity property (1.10) implies that the whole sequence $\{\Theta(x_k)\}$ tends to $\Theta(x^*)$: all cluster points of $\{x_k\}$ have the same Θ-value. Whether this value is the minimum value of Θ is more delicate.

When Θ is a convex function, the optimality condition $\nabla\Theta(x^*) = 0$ is (necessary and) sufficient for x^* to minimize Θ (use for example the well-known property $\Theta(y) \geqslant \Theta(x^*) + (\nabla\Theta(x^*), y - x^*)$ for all y). In this situation, we conclude that *all* the cluster points of $\{x_k\}$ minimize Θ; and finally, the whole of $\{x_k\}$ converges to the same limit x^* if Θ has a single minimum point x^* (for example if Θ is strictly convex).

Let us summarize our considerations: admitting that (P) can be formulated as minimizing a differentiable function Θ, the key property to be satisfied by an algorithm is (1.11). If Θ enjoys appropriate additional properties, then the limit points of $\{x_k\}$ will minimize Θ, and hence solve (P).

1.5 Convergence: Local Aspects

Now $\{x_k\}$ is assumed to have a limit x^* – which may or may not be "what is wished" – and one wants to know at what speed $x_k - x^*$ tends to 0; in particular, one tries to compare this error to an exponential function. This study is limited to large values of k (hence x_k is already close to x^*): it is only a *local* study. First recall some notation: $s = o(t)$ means that s is "infinitely smaller" than t; more precisely $\frac{s}{t} \to 0$. Here t and s are two variables (depending on a parameter x, on an iteration number k, etc.); t is scalar-valued and positive; strictly speaking, s as well; when s is vector-valued, the correct and complete notation should be $|s| = o(t)$. In practice, it is implicitly understood that $t \downarrow 0$ (say when $x \to x^*$, or $k \to +\infty$) and $s = o(t)$ means that s tends to 0 infinitely faster than t. The notation $s = O(t)$ means that s is not infinitely bigger than t: there exists a constant C such that $s \leqslant Ct$.

Consider now a sequence $\{x_k\}$ converging to x^*; two types of convergence are relevant:

Q-convergence : this is a study of the quotient $q_k := |x_{k+1} - x^*|/|x_k - x^*|$.

- Q-linear convergence is said to hold when $\limsup q_k < 1$.
- Q-superlinear convergence when $\lim q_k = 0$.
- Particular case: Q-quadratic convergence when $q_k = O(|x_k - x^*|)$; or equivalently: $|x_{k+1} - x^*| = O(|x_k - x^*|^2)$; roughly, the number of exact digits doubles at each iteration.

Often, "Q" is omitted: superlinear convergence implicitly means Q-superlinear convergence.

R-convergence : even though Theorems 1.7 and 1.8 below give a more natural definition, R-convergence is originally a study of the rate $r_k := |x_k - x^*|^{1/k}$.
– $\limsup r_k < 1$: R-linear convergence,
– $\lim r_k = 0$: R-superlinear convergence.

Remark 1.5. A sequence converging sublinearly to its limit (q_k or r_k tends to 1) is in practice considered as not converging at all, because convergence is so slow; an algorithm with sublinear convergence must simply be forgotten. □

R-linear convergence means geometric or exponential convergence: setting $r := \limsup r_k$, we have $r_k \leqslant r + \varepsilon$ for all $\varepsilon > 0$ and k large enough; this is equivalent to $|x_k - x^*| \leqslant (r + \varepsilon)^k$ (and note: $r + \varepsilon$ can be made < 1).

Q-convergence is more powerful, in that the error at iteration $k + 1$ can be bounded in terms of the error at iteration k: if $q = \limsup q_k$,

$$|x_{k+1} - x^*| \leqslant (q + \varepsilon)|x_k - x^*|\,, \quad \text{for all } \varepsilon > 0 \text{ and } k \text{ large enough.}$$

In a way, Q-convergence is a Markovian concept: it only involves what happens at the present iteration. In the above writing, "iteration k [resp. $k+1$]" can be replaced by "current iterate x [resp. next iterate x_+]" and "k large enough" by "x close enough to x^*". In plain words, Q-superlinear convergence is expressed by: if the current iterate is close to the limit, then the next iterate is infinitely closer. This is not true for R-convergence, since k plays its role in the definition of r_k, which has to be a kth root. The next result confirms that Q-linear convergence implies geometric convergence:

Theorem 1.6. *If x_k tends Q-linearly to x^*, then: for all $q > \limsup q_k$, there exists k_0 and $C > 0$ such that*

$$|x_k - x^*| \leqslant Cq^k \text{ for all } k \geqslant k_0.$$

Proof. Fix q as announced, k_0 such that

$$|x_{i+1} - x^*| \leqslant q|x_i - x^*| \text{ for } i \geqslant k_0,$$

which gives (multiplying out for $i = k_0, \ldots, k-1$)

$$|x_k - x^*| \leqslant |x_{k_0} - x^*|q^{k-k_0} = \frac{|x_{k_0} - x^*|}{q^{k_0}} q^k$$

and the result is obtained with $C := |x_{k_0} - x^*|/q^{k_0}$. □

Once again, this theorem does not contain all the power of Q-convergence, since it does not say that the error decreases at the rate $q < 1$ at *each* iteration.

Quite often, convergence speed is established via a study of an upper bound of the error. Q-convergence of an upper bound of $|x_k - x^*|$ becomes R-convergence for $\{x_k\}$. For example:

Theorem 1.7. *If $|x_k - x^*| \leqslant s_k$ where s_k converges Q-superlinearly to 0, then $\{x_k\}$ converges R-superlinearly to x^*.*

Proof. Fix $\varepsilon > 0$. From Theorem 1.6, there is C such that $s_k \leqslant C\varepsilon^k$ for k large enough. Hence, by assumption,

$$|x_k - x^*|^{1/k} \leqslant s_k^{1/k} \leqslant C^{1/k}\varepsilon\,.$$

Pass to the limit on k: $C^{1/k} \to 1$ and $\limsup |x_k - x^*|^{1/k} \leqslant \varepsilon$. □

Actually, the converse is also true. To show it, we give a last result, stated in terms of linear convergence, to make a change:

Theorem 1.8. *Let x_k tend to x^* R-linearly. Then $|x_k - x^*|$ is bounded from above by a sequence s_k tending to 0 Q-linearly.*

Proof. Call $r < 1$ the limsup of $|x_k - x^*|^{1/k}$ and take $\varepsilon \in]0, 1-r[$. For k large enough, $|x_k - x^*| \leqslant (r+\varepsilon)^k$. The sequence $s_k := \max\{|x_k - x^*|, (r+\varepsilon)^k\}$ is indeed an upper bound of $\{|x_k - x^*|\}$ and, for k large enough, $s_k = (r+\varepsilon)^k$; hence s_k answers the question. □

These two theorems establish the equivalence between R-convergence of a nonnegative sequence tending to 0, and Q-convergence of an upper bound. This gives another definition of R-convergence, perhaps more natural than the original one; namely: $x_k \to x^*$ R-superlinearly when $|x_k - x^*| \leqslant s_k$, for some $\{s_k\}$ tending to 0 Q-superlinearly.

1.6 Computing the Gradient

As seen in §1.3, the main duty of the user of an optimization algorithm is to write a simulator computing information needed by the algorithm. It has also been said (and it will be confirmed all along this book) that the simulator should compute not only function- but also derivatives-values. This is not always a trivial task, especially in optimal control problems. Take for example the case of meteorology in §1.2.2: it is easy to understand how the objective function of (1.2) (call it f) can be computed via (1.1), for given values of the control variable $u(\cdot) = p(\cdot, -1)$; but how about the *total derivative* of f with respect to u? Since f is given *implicitly* by (1.1), one must somehow invoke the implicit function theorem, which may be tricky. Indeed, computing the

Jacobian of the operator "control variable $\mapsto$ state variable" is often out of question, and useless anyway. Here we demonstrate a technique commonly used, which involves the *adjoint equation*. For reasons to be explained in Remark 1.9 below, we do this computation in a finite-dimensional setting, even though optimal control problems are usually set in some function space.

So we consider the following situation. The control variables are $\{u_t\}_{t=1}^T$ where $u_t \in \mathbb{R}^n$ for each t. The state variables are likewise $\{y_t\}_t$ with $y_t \in \mathbb{R}^m$, given by the state equation

$$\begin{cases} y_t = F_t(y_{t-1}, u_t)\,, \quad \text{for } t = 1, \dots, T, \\ y_0 \text{ given.} \end{cases} \tag{1.12}$$

Here, for each t, F_t is a function (possibly nonlinear) from $\mathbb{R}^m \times \mathbb{R}^n$ to $\mathbb{R}^m$. Besides, a function is given, say

$$f = \sum_{t=1}^{T} f_t(y_t, u_t)\,,$$

where, for each t, f_t sends $\mathbb{R}^m \times \mathbb{R}^n$ to $\mathbb{R}$. It is purposedly that we do not specify formally which variables f depends on. Incidentally, note that f can be the objective function of our optimal control problem; but it can equally be a constraint, involving the state variables; for example a *final-time constraint* $c(y_T)$ (imposed to be 0, or nonnegative, etc.)

Call $v = \mathrm{d}u \in \mathbb{R}^{nT}$ a differential of u; it induces from (1.12) a differential $z = \mathrm{d}y \in \mathbb{R}^{mT}$, and finally a differential $\mathrm{d}f$. To be specific, we assume the usual dot product in each of the spaces involved and we use the notation $(\cdot,\cdot)_n$ [resp. $(\cdot,\cdot)_m$] for the dot-product in $\mathbb{R}^n$ [resp. $\mathbb{R}^m$]. In the control space, the scalar product is therefore

$$(g, v) = \sum_{j=1}^{T} (g_t, v_t)_n\,.$$

Our problem is then as follows: find $\{g_t\}_{t=1}^T$ such that the differential of f is given by $\mathrm{d}f = (g, v)$. This will yield $\{g_t\}_t \in \mathbb{R}^{nT}$ as the gradient of f, considered as a function of the control variable u alone.

To solve this problem, we have from (1.12) (assuming appropriate smoothness of the data)

$$\begin{cases} z_t = (F_t)'_y(y_{t-1}, u_t) z_{t-1} + (F_t)'_u(y_{t-1}, u_t) v_t \quad \text{for } t = 1, \dots, T, \\ z_0 = 0 \end{cases} \tag{1.13}$$

($z_0 = 0$ because y_0 is fixed!). In this writing, the Jacobian $(F_t)'_y(y_{t-1}, u_t)$ is an $m \times m$ matrix and $(F_t)'_u(y_{t-1}, u_t)$ is $m \times n$. We have also

$$\mathrm{d}f = \sum_{t=1}^{T} (\nabla_y f_t(y_t, u_t), z_t)_m + \sum_{t=1}^{T} (\nabla_u f_t(y_t, u_t), v_t)_n\,;$$

here $\nabla_y f_t(y_t, u_t) \in \mathbb{R}^m$ and $\nabla_u f_t(y_t, u_t) \in \mathbb{R}^n$. We need to eliminate z between these various relations; this is done by a series of tricks:

Trick 1. Multiply the t^{th} linearized state equation in (1.13) by a vector $p_t \in \mathbb{R}^m$ (unspecified for the moment) and sum up. Setting $G_t := (F_t)'_y(y_{t-1}, u_t)$ and $H_t := (F_t)'_u(y_{t-1}, u_t)$, we obtain

$$\sum_{t=1}^{T}(p_t, z_t)_m = \sum_{t=1}^{T}(p_t, G_t z_{t-1})_m + \sum_{t=1}^{T}(p_t, H_t v_t)_m \,.$$

Single out $(p_T, z_T)_m$ in the lefthand side, transpose G_t and H_t, and re-index the sum in z; remembering that $z_0 = 0$, this gives

$$0 = -(p_T, z_T)_m - \sum_{t=1}^{T-1}(p_t, z_t)_m + \sum_{t=1}^{T-1}(G_{t+1}^{\top} p_{t+1}, z_t)_m + \sum_{t=1}^{T}(H_t^{\top} p_t, v_t)_n \,.$$

Trick 2. Add to the expression of $\mathrm{d}f$ and identify with respect to the z_t's. Setting $\gamma_t := \nabla_y f_t(y_t, u_t)$ and $h_t := \nabla_u f_t(y_t, u_t)$:

$$\mathrm{d}f = (-p_T + \gamma_T, z_T)_m + \sum_{t=1}^{T-1}(-p_t + G_{t+1}^{\top} p_{t+1} + \gamma_t, z_t)_m + \sum_{t=1}^{T}(H_t^{\top} p_t + h_t, v_t)_n \,.$$

Trick 3. Now it suffices to choose p so as to cancel out the coefficient of each z_t: requiring

$$\begin{cases} p_T = \gamma_T \,, \\ p_t = G_{t+1}^{\top} p_{t+1} + \gamma_t \quad \text{for } t = T-1, \dots, 1 \,, \end{cases} \tag{1.14}$$

we obtain the gradient in the desired form:

$$g_t = H_t^{\top} p_t + h_t \quad \text{for } t = 1, \dots, T.$$

The (backward) recurrence relations (1.14) form the so-called *adjoint equation*, whose solution p is the *adjoint state*.

Remark 1.9. In optimal control problems, the state variable is often given by a differential equation, say

$$\begin{cases} \dot{y}(t) = F(y(t), u(t), t) \,, \quad \text{for } t \in]0, T[\,, \\ y(0) \text{ given}, \end{cases}$$

instead of the recurrence relations (1.12). Then the "adjoint trick" can nevertheless be reproduced: multiply the above equation by a function $p(t)$ (the continuous adjoint state), integrate from 0 to T, and integrate the lefthand side by parts. The resulting adjoint equation is another differential equation, instead of (1.14).

However, the actual minimization algorithm, implemented on the computer, certainly does not solve this original problem; it can but solve some *discretized* form of it (a computer can hardly work in infinite dimension). Using a subscript δ to connote such a discretization, we are eventually faced with

minimizing a certain function $f_\delta(u_\delta)$, with respect to some finite-dimensional variable u_δ. For numerical efficiency of the minimization algorithm, *it is important that the simulator computes the exact gradient of* f_δ, and not some discretized form of the continuous gradient ∇f. One way of achieving this is to carefully select the discretization scheme of the adjoint equation. But the safest approach is to *discretize first* the problem (and in particular the state equation), and then only to construct the adjoint equation *of the discretized problem*.

This is why we bothered to demonstrate the mechanism for the tedious discrete case; after this, reproducing the calculations in the continuous case is an easy exercise (only formal, though: differentiability properties of the infinite-dimensional problem must still be carefully analyzed; otherwise, difficulties may occur for $\delta \to 0$). □

Remark 1.10. The adjoint technique opens the way to the so-called *automatic* or *computational differentiation*. Indeed, consider a computer code which, taking an input u, computes an output f. Such a code can be viewed as a "control process" of the type (1.12):

– The t^{th} line of this code is the t^{th} equation in (1.12).
– The intermediate results of this code (the lefthand sides of the assignment statements) form altogether a "state" y, which is a function of the "control" u.
– Forming the righthand side of the adjoint equations then amounts to differentiating one by one each line of the code.
– Afterwards, solving the adjoint equations – to obtain finally the gradient ∇f – amounts to writing these "linearized lines" bottom up.

These operations are all purely mechanical and lend themselves to automatization. Thus, one can conceive the existence of a software which

– takes as input a computer code able to calculate $f(u)$ (for given u),
– and produces as output another computer code able to calculate $\nabla f(u)$ (again for given u).

It is worth mentioning that such software do not need to know anything about the problem. They do not even need mathematical formulae representing the computation of f. What they need is just the first half of a simulator; and then they write down its second half. □

Bibliographical Comments

Among other monographs devoted to optimization algorithms, [107, 27, 277, 86] can be suggested. See also [128, 160] for a style very close to users' concerns, while [239] insists more on theorems.

A function Θ for which a stationary sequence ($\nabla\Theta(x_k) \to 0$) is not necessarily minimizing ($\Theta(x_k) \not\to \inf\Theta$) is given in [350]. The various types of local convergence are defined and studied in [278].

As for available optimization software, the situation is rapidly evolving. First, there is the monograph [267], which reviews most individual codes and organized libraries existing in the beginning of the 90's. Generally speaking, the Harwell library has well-considered optimization codes. In fact, this library goes far beyond optimization, as it covers the whole of numerical analysis, from linear algebra to differential equations:

`http://www.cse.clrc.ac.uk/Activity/HSL`.

On the other hand, the Galahad software is exclusively devoted to optimization and can normally be used for free:

`http://galahad.rl.ac.uk/galahad-www`.

The Scilab environment and the Modulopt library include implementations of some of the algorithms presented in this book:

`http://www-rocq.inria.fr/scilab/scilab.html`
`http://www-rocq.inria.fr/estime/modulopt`.

The internet address

`http://www-neos.mcs.anl.gov/neos`

collects and updates, under the name NEOS, the vast majority of software existing throughout the world, even allowing a "push-button" use of some of them.

For computational differentiation, see for example [181], [88], [151] (but the idea is much older, going back to [339, 208] and others). We mention Adolc, Adifor, Tapenade as available software; the addresses are as follows:

`http://www.math.tu-dresden.de/wir/project/adolc`
`http://www-unix.mcs.anl.gov/autodiff/ADIFOR`
`http://www-sop.inria.fr/tropics/tapenade/tutorial`
`http://www-unix.mcs.anl.gov/autodiff/AD_Tools`