Chapter 1

Introduction

1.1 Why derivative-free optimization

It is well known that extensive useful information is contained in the derivatives of any function one wishes to optimize. After all, the "standard" mathematical characterization of a local minimum, given by the first-order necessary conditions, requires, for continuously differentiable functions, that the first-order derivatives are zero. However, for a variety of reasons there have always been many instances where (at least some) derivatives are unavailable or unreliable. Nevertheless, under such circumstances it may still be desirable to carry out optimization. Consequently, a class of nonlinear optimization techniques called derivative-free optimization methods has always been needed. In fact, we consider optimization without derivatives one of the most important, open, and challenging areas in computational science and engineering, and one with enormous practical potential. The reason that it is challenging is that, from the point of view of optimization, one gives up so much information by not having derivatives. The source of its current, practical importance is the ever growing need to solve optimization problems defined by functions for which derivatives are unavailable or available at a prohibitive cost. Increasing complexity in mathematical modeling, higher sophistication of scientific computing, and an abundance of legacy codes are some of the reasons why derivative-free optimization is currently an area of great demand.

In earlier days of nonlinear optimization perhaps one of the most common reasons for using derivative-free methods was the lack of sophistication or perseverance of the user. The users knew they wanted to improve on their current "solution," but they wanted to use something simple that they could understand, and so they used (and, unfortunately, sometimes continue to use) nonderivative methods, like the method by Nelder and Mead [177], even when more appropriate algorithms were available. In defense of the practitioner, we should remember that until relatively recently computing derivatives was the single most common source of user error in applying optimization software (see, for example, [104, Chapter 8, page 297]). As the scale and difficulty of the applications increased, more sophisticated derivative-based optimization methods became more essential. With the growth and development of derivative-based nonlinear optimization methods it became evident that large-scale problems can be solved efficiently, but only if there is accurate derivative infor-

mation at hand. Users started either to provide such derivatives (by hand-coding them or applying automatic differentiation tools to their software) or to estimate the derivatives by finite differences. Some optimization software packages perform the finite-difference gradient evaluation internally, but it is usually better to leave this to the user since the ideal size of the finite-difference step may depend on the application.

However, there are situations where none of these approaches for obtaining the derivatives works. For example, in the case of legacy or proprietary codes, i.e., codes that have been written in the past and which have not been maintained by the original authors, rewriting such a code now, or adding to it what would be required to provide first-order derivatives, can be an extremely time-consuming task. The problem might also depend on a code owned by a company for which there is access only to the binary files. Automatic differentiation techniques (see, for example, [111, 112]) also cannot be applied in all cases. In particular, if the objective function is computed using a black-box simulation package, automatic differentiation is typically impossible, and even in the case where the computation is more accessible, legacy or proprietary issues may make such an approach unacceptable.

There are also two situations where applying finite-difference derivative approximation is inappropriate: when the function evaluations are costly and when they are noisy. In the first case, it may be prohibitive to perform the necessary number of function evaluations (normally no less than the number of variables plus one) to provide a single gradient estimation. In the second case, the gradient estimation may be completely useless. Ironically, with the growing sophistication of computer hardware and mathematical algorithms and software, situations like these are becoming more, rather than less, frequent. The reason is simply that while, before, simulation of complex systems was a difficult and costly task and did not provide a sufficiently good setting for optimization, now such simulations are becoming more routine and also more accurate; hence optimization of complex systems is becoming a reasonable possibility. The growing demand for sophisticated derivative-free optimization methods has triggered the development of a relatively wide range of approaches. In recent years, the field has undergone major changes with improvements in theory and practice and increased visibility in the nonlinear optimization community. By writing this book we hope to provide a unifying theoretical view of the derivative-free methods existing today. We discuss briefly the practical performance, and what can be expected, but the main purpose of the book is to give the general picture of why and how the algorithms work.

The methods we will consider do not rely on derivative information of the objective function or constraints, nor are the methods designed explicitly to approximate these derivatives. Rather, they build models of the functions based on sample function values or they directly exploit a sample set of function values without building an explicit model. Not surprisingly, as we already suggested, there are considerable disadvantages in not having derivative information, so one cannot expect the performance of derivative-free methods to be comparable to those of derivative-based methods. In particular, the scale of the problems that can currently be efficiently solved by derivative-free methods is still relatively small and does not exceed a few hundred variables even in easy cases. Stopping criteria are also a challenge in the absence of derivatives, when the function evaluations are noisy and/or expensive. Therefore, a near-optimal solution obtained by a derivative-free method is often less accurate than that obtained by a derivative-based method, assuming derivative information is available. That said, for many of the applications that exist today these limitations are acceptable.

1.2 Examples of problems where derivatives are unavailable

There is really an enormous number of problems where derivatives are unavailable but one has imperative practical reasons for wanting to do some optimization. It is almost a natural perversity that practical problems today are often complex, nonlinear, and not sufficiently explicitly defined to give reliable derivatives. Indeed, such problems were always numerous, but, 30 years ago, when nonlinear optimization techniques were relatively more naive than they are today, even the most optimistic practitioners would not try to optimize such complex problems. Not so in 2008! The diversity of applications includes problems in engineering, mathematics, physics, chemistry, economics, finance, medicine, transportation, computer science, business, and operations research.

As examples we include a subset of known applications and references. We start by some illustrations (like algorithmic parameter tuning and automatic error analysis) which may be atypical of the applications of derivative-free optimization but are easy to understand.

Tuning of algorithmic parameters and automatic error analysis

An interesting (and potentially useful) application of derivative-free optimization has been explored in [22] to tune parameters of nonlinear optimization methods, with promising results. Most numerical codes (for simulation, optimization, estimation, or whatever) depend on a number of parameters. Everybody implementing numerical algorithms knows how critical the choices of these parameters are and how much they influence the performance of solvers. Typically, these parameters are set to values that either have some mathematical justification or have been found by the code developers to perform well. One way to automate the choice of the parameters (in order to find possibly optimal values) is to consider an optimization problem whose variables are the parameters and whose objective function measures the performance of the solver for a given choice of parameters (measured by CPU time or by some other indicator such as the number of iterations taken by the solver). Such problems might have constraints like upper and lower bounds on the values of the solver parameters, and look like

$$\min_{p \in \mathbb{R}^{n_p}} f(p) = CPU(solver; p) \quad \text{s.t.} \quad p \in P,$$

where n_p is the number of parameters to be tuned and P is of the form $\{p \in \mathbb{R}^{n_p} : \ell \leq p \leq u\}$. Not only is it hard to calculate derivatives for such a function f, but numerical noise and some form of nondifferentiability are likely to take place.

Derivative-free optimization has also been used for automatic error analysis [126, 127], a process in which the computer is used to analyze the accuracy or stability of a numerical computation. One example of automatic error analysis is to analyze how large the growth factor for Gaussian elimination can be for a specific pivoting strategy. The relevance of such a study results from the influence of the growth factor in the stability of Gaussian elimination. Given a pivoting strategy and a fixed matrix dimension n, the optimization problem posed is to determine a matrix that maximizes the growth factor for

Gaussian elimination:

$$\max_{A\in\mathbb{R}^{n\times n}} f(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|},$$

where the $a_{ij}^{(k)}$ are the intermediate elements generated during the elimination. A starting point could be the identity matrix of order n. When no pivoting is chosen, f is defined and continuous at all points where elimination does not break down. There could be a lack of differentiability when ties occur at the maxima that define the growth factor expression. For partial pivoting, the function f is defined everywhere (because the elimination cannot break down), but it can be discontinuous when a tie occurs at the choice of the pivot element. Other examples of automatic error analysis where derivative-free optimization has been used are the estimation of the matrix condition number and the analysis of numerical stability for fast matrix inversion and polynomial root finding [126, 127].

Engineering design

A case study in derivative-free optimization is the helicopter rotor blade design problem [38, 39, 206]. The goal is to find the structural design of the rotor blades to minimize the vibration transmitted to the hub. The variables are the mass, center of gravity, and stiffness of each segment of the rotor blade. The simulation code is multidisciplinary, including dynamic structures, aerodynamics, and wake modeling and control. The problem includes upper and lower bounds on the variables, and some linear constraints have been considered such as an upper bound on the sum of masses. Each function evaluation requires simulation and can take from minutes to days of CPU time. A surrogate management framework based on direct-search methods (see Section 12.2) led to encouraging results. Other multidisciplinary or complex design problems have been reported to be successfully solved by derivative-free optimization methods and/or surrogate management frameworks, like wing platform design [16], aeroacoustic shape design [164, 165], and hydrodynamic design [87].

Circuit design

Derivative-free methods have also been used for tuning parameters of relatively small circuits. In particular, they have been used in the tuning of clock distribution networks. Operations on a chip are done in cycles (e.g., eight cycles per operation), and every part of the chip has to be synchronized at the beginning of each cycle. The clock distribution network serves to synchronize the chip by sending a signal to every part, and the signal has to arrive everywhere with minimum distortion and delay in spite of the fact that there is always some of both. The resulting optimization problem includes as possible parameters wire length, wire width, wire shields, and buffer size. The possible constraints and objectives considered include delay, power, slew over/undershoot, the duty cycle, and the slew. The function values are computed by the `PowerSpice` package [2], a well-known accurate simulation package for circuits that, however, provides no derivative computation. Derivative-free optimization is applied to assist circuit designers in the optimization of their (small) circuits. In the particular case of clock distribution networks, it makes it possible for the circuit designers to try different objective functions and constraint combinations, relatively quickly, without manual tuning. This leads to a much better understanding of the problem and thereby enables discovery of alternative designs.

Molecular geometry

Another area where it is not unusual to use derivative-free methods of the type we are interested in is in the optimization of molecular geometries and related problems. An example of this would be considering the geometry of a cluster of N atoms (which amounts to $3N-6$ variables). The aim is then the unconstrained minimization of the cluster's total energy computed by numerical simulation of the underlying dynamics. We point out that for these classes of problems there is the presence of several local minimizers (although the more realistic the computation of the total energy, the lower the number of local minima), and so there must be some caution when using the type of derivative-free methods covered in this book (see Section 13.3 for extensions to global optimization). The gradient might be available, but it could be either expensive or affected by noise or even undesirable to use given the presence of nonconvexity. Derivative-free optimization methods, in particular direct-search methods (see Chapter 7), have been appropriately adapted to handle classes of these problems (see [10, 170]).

Other applications

Many other recent applications of derivative-free optimization could be cited in diverse areas such as groundwater community problems [99, 174, 233], medical image registration [179, 180], and dynamic pricing [150].

1.3 Limitations of derivative-free optimization

Perhaps foremost among the limitations of derivative-free methods is that, on a serial machine, it is usually not reasonable to try and optimize problems with more than a few tens of variables, although some of the most recent techniques can handle unconstrained problems in hundreds of variables (see, for example, [192]). Also, even on relatively simple and well-conditioned problems it is usually not realistic to expect accurate solutions. Convergence is typically rather slow. For example, in order to eventually achieve something like a quadratic rate of local convergence, one needs either implicit or explicit local models that are reasonable approximations for a second-order Taylor series model in the current neighborhood. With 100 variables, if one was using interpolation, a local quadratic function would require 5151 function evaluations (see Table 1.1). Just as one can achieve good convergence using approximations to the Hessian matrix (such as quasi-Newton methods [76, 178]) in the derivative-based case, it is reasonable to assume that one can achieve similar fast convergence in the derivative-free case by using incomplete quadratic models or quadratic models based on only first-order approximations. These ideas are successfully used in [52, 59, 61, 191, 192]. For instance, the algorithm in `NEWUOA` [192] typically uses $2n+1$ function evaluations to build its models. However, unless the function looks very similar to a quadratic in the neighborhood of the optimal solution, in order to progress, the models have to be recomputed frequently as the step size and the radius of sampling converge to zero. Even in the case of linear models this can be prohibitive when function evaluations are expensive. Thus typically one can expect a local convergence rate that is closer to linear than quadratic, and one may prefer early termination.

As for being able to tackle only problems in around 20 or moderately more variables (currently the largest unconstrained problems that can be tackled on a serial machine appear

Table 1.1. *Number of points needed to build a "fully quadratic" polynomial interpolant model.*

n	10	20	50	100	200
$(n+1)(n+2)/2$	66	231	1326	5151	20301

to be in several hundred variables), the usual remedy is to use statistical methods like analysis of variance (see, for example, [203]) to determine, say, the most critical 20 variables, and optimizing only over them. It may also be reasonable to take advantage of the relative simplicity of some of the derivative-free algorithms, like directional direct-search methods (explained in Chapter 7), and compute in a parallel, or even massively parallel, environment.

Another limitation of derivative-free optimization may occur when minimizing nonconvex functions. However, and although nothing has been proved to support this statement, it is generally accepted that derivative-free optimization methods have the ability to find "good" local optima in the following sense. If one has a function with an apparently large number of local optimizers, perhaps because of noise, then derivative-free approaches, given their relative crudeness, have a tendency to initially go to generally low regions in the early iterations (because of their myopia, or one might even say near-blindness) and in later iterations they still tend to smooth the function, whereas a more sophisticated method may well find the closest local optima to the starting point. The tendency to "smooth" functions is also why they are effective for moderately noisy functions. There are many situations where derivative-free methods are the only suitable approach, capable of doing better than heuristic or other "last-resort" algorithms and providing a supporting convergence theory.

There are, however, classes of problems for which the use of derivative-free methods that we address here are not suitable. Typically, rigorous methods for such problems would require an inordinate amount of work that grows exponentially with the size of the problem. This category of problems includes medium- and large-scale general global optimization problems with or without derivatives, problems which not only do not have available derivatives but which are not remotely like smooth problems, general large nonlinear problems with discrete variables including many combinatorial optimization ones (so-called NP hard problems), and stochastic optimization problems. Although relatively specialized algorithms can find good solutions to many combinatorial optimization problems perfectly adequately, this is not the case for general large nonlinear problems with integer variables. We do not address algorithmic approaches designed specifically for combinatorial or stochastic optimization problems.

For some of the extreme cases mentioned above, heuristics are frequently used, such as simulated annealing [143], genetic and other evolutionary algorithms [108, 129], artificial neural networks [122], tabu-search methods [107], and particle swarm or population-based methods [142], including (often very sophisticated variations of) enumeration techniques. The authors think of these as methods of a last resort (that is, applicable to problems where the search space is necessarily large, complex, or poorly understood and more sophisticated mathematical analysis is not applicable) and would use them only if nothing better is available. We do not address such methods in this book. However, sometimes these approaches are combined with methods that we do address; see, for example, [13, 135, 222] and Section 13.3 of this book.

It is sometimes perceived that derivative-free optimization methods should be simple and easy to implement. However, such methods are typically inferior in theory and in practice. The Nelder–Mead algorithm, however, can work very well and it is expected to survive a very long time. Nevertheless it is seriously defective: it is almost never the best method and indeed it has no general convergence results, because it can easily not converge (although modifications of it are provably convergent, as we will explain in our book). Since the authors' combined research experience in optimization is over 60 years, we believe that ultimately more sophisticated and successful methods will earn their rightful place in practical implementations once a comprehensive description of such methods is widely available. This, in part, is motivation for this book.

Finally, we want to make a strong statement that often councils against the use of derivative-free methods: if you can obtain clean derivatives (even if it requires considerable effort) and the functions defining your problem are smooth and free of noise you should not use derivative-free methods.

1.4 How derivative-free algorithms should work

This book is mostly devoted to the study of derivative-free algorithms for unconstrained optimization problems, which we will write in the form

$$\min_{x \in \mathbb{R}^n} f(x). \tag{1.1}$$

We are interested in algorithms that are globally convergent to stationary points (of first- or second-order type), in other words, algorithms that regardless of the starting point are able to generate sequences of iterates asymptotically approaching stationary points.

Some of the main ingredients

Perhaps this is oversimplifying, but one could say that there are three features present in all globally convergent derivative-free algorithms:

1. They incorporate some mechanism to impose descent away from stationarity. The same is done by derivative-based algorithms to enforce global convergence, so this imposition is not really new. It is the way in which this is imposed that makes the difference. Direct-search methods of directional type, for instance, achieve this goal by using *positive bases or spanning sets* (see Chapter 7) and moving in the direction of the points of the pattern with the best function value. Simplex-based methods (Chapter 8) ensure descent from *simplex operations* like *reflections*, by moving in the direction *away* from the point with the worst function value. Methods like the implicit-filtering method (Chapter 9) aim to get descent along negative *simplex gradients*, which are intimately related to polynomial models. Trust-region methods, in turn, minimize trust-region subproblems defined by *fully linear* or *fully quadratic models*, typically built from polynomial interpolation or regression—see Chapters 10 and 11.

   In every case, descent is guaranteed away from stationarity by combining such mechanisms with a possible reduction of the corresponding step size parameter. Such a parameter could be a mesh size parameter (directional direct search), a simplex diameter (simplicial direct search), a line-search parameter, or a trust-region radius.

2. They must guarantee some form of control of the geometry of the sample sets where the function is evaluated. Essentially, such operations ensure that any indication of stationarity (like model stationarity) is indeed a true one. Not enforcing good geometry explains the lack of convergence of the original Nelder–Mead method.

   Examples of measures of geometry are (i) the cosine measure for positive spanning sets; (ii) the normalized volume of simplices (both to be kept away from zero); (iii) the Λ-poisedness constant, to be maintained moderately small and bounded from above when building interpolation models and simplex derivatives.

3. They must drive the step size parameter to zero. We know that most optimization codes stop execution when the step size parameter passes below a given small threshold. In derivative-based optimization such terminations may be premature and an indication of failure, perhaps because the derivatives are either not accurate enough or wrongly coded. The best stopping criteria when derivatives are available are based on some form of stationarity indicated by the first-order necessary conditions.

   In derivative-free optimization the step size serves a double purpose: besides bounding the size of the minimization step it also controls the size of the local area where the function is sampled around the current iterate. For example, in direct-search methods the step size and the mesh size (defining the pattern) are the same or constant multiples of each other. In a model-based derivative-free method, the size of the trust region or line-search step is typically intimately connected with the radius of the sample set. Clearly, the radius of the sample set or mesh size has to converge to zero in order to ensure the accuracy of the objective function representation. It is possible to decouple the step size from the size of the sample set; however, so far most derivative-free methods (with the exception of the original Nelder–Mead method) connect the two quantities. In fact, the convergence theory of derivative-free methods that we will see in this book shows that the sequence (or a subsequence) of the step size parameters do converge to zero (see, e.g., Theorem 7.1 or Lemma 10.9). It is an implicit consequence of the mechanisms of effective algorithms and should not (or does not have to) be enforced explicitly. Thus, a stopping criterion based on the size of the step is a natural one.

The details of the ingredients listed above—it is hoped—will be made much clearer to the reader in the chapters to follow.

With the current state-of-the-art derivative-free optimization methods one can expect to successfully address problems (i) which do not have more than, say, a hundred variables; (ii) which are reasonably smooth; (iii) in which the evaluation of the function is expensive and/or computed with noise (and for which accurate finite-difference derivative estimation is prohibitive); (iv) in which rapid asymptotic convergence is not of primary importance.

An indication of typical behavior

We chose two simple problems to illustrate some of the validity of the above comments and some of the advantages and disadvantages of the different classes of methods for derivative-free optimization. We do not want to report a comprehensive comparison on how the different methods perform since such a numerical study is not an easy task to perform well, or even adequately, in derivative-free optimization given the great diversity of the application problems. A comparison among different methods is easier, however, if we focus on

a particular application. Another possibility is to select a test set of problems with known derivatives and treat them as if the derivatives were unavailable. Moré and Wild [173] suggested the use of data profiles to compare a group of derivative-free algorithms on a test set. In any case we leave such comparisons to others.

The first problem consists of the minimization of the Rosenbrock function

$$\min_{(x_1,x_2)\in\mathbb{R}^2} 100(x_1^2 - x_2)^2 + (1 - x_1)^2,$$

which has a unique minimizer at $(1,1)$. The level curves of this function describe a strongly curved valley with steep sides (see Figure 1.1). Depending on the starting point picked, methods which do not explore the curvature of the function might be extremely slow. For instance, if one starts around $(-1,1)$, one has to follow a curved valley with relatively steep sides in order to attain the minimum.

The second problem involves the minimization of a deceptively simple, perturbed quadratic function. The perturbation involves cosine functions with periods of $2\pi/70$ and $2\pi/100$:

$$\begin{aligned}\min_{(x_1,x_2)\in\mathbb{R}^2} \quad & 10(x_1^2)(1 + 0.75\cos(70x_1)/12) + \cos(100x_1)^2/24 \\ & + 2(x_2^2)(1 + 0.75\cos(70x_2)/12) + \cos(100x_2)^2/24 + 4x_1x_2.\end{aligned}$$

The unique minimizer is at $(0,0)$ (see Figure 1.1). As opposed to the Rosenbrock function, the underlying smooth function here has a mild curvature. However, the perturbed function has been contaminated with noise which will then pose different difficulties to algorithms.

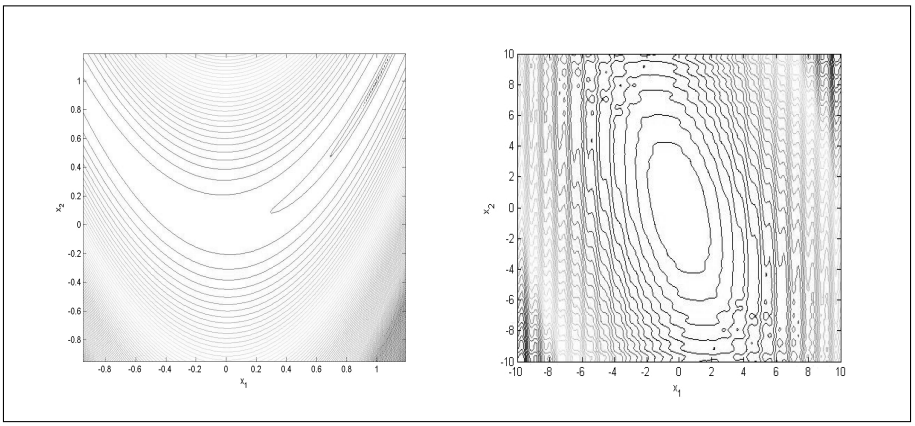

Figure 1.1. *Level curves of the Rosenbrock function (left) and a perturbed quadratic (right).*

We selected four methods to run on these two problems as representatives of the four main classes of methods addressed in this book (see Chapters 7–10). The first method is coordinate search, perhaps the simplest of all direct-search methods of directional type (see Chapter 7). In its simplest form it evaluates the function at $2n$ points around a current iterate defined by displacements along the coordinate directions and their negatives

and a step size parameter (a process called *polling*). This set of directions forms a *positive basis*. The method is slow but robust and capable of handling noise in the objective function. The implementation used is reported in [70]. The second choice is the Nelder–Mead method [177], a simplicial direct-search method (see Chapter 8). Based on simplex operations such as reflections, expansions, and contractions (inside or outside), the Nelder–Mead method attempts to replace the simplex vertex that has the worst function value. The simplices generated by Nelder–Mead may adapt well to the curvature of the function. However, the original, unsafeguarded version of Nelder–Mead (as coded in the MATLAB® [1] routine we used in our testing) is not robust or reliable since the simplices' shapes might deteriorate arbitrarily. Coordinate search, on the contrary, is guaranteed to converge globally.

The other two methods follow a different approach. One of them is the implicit-filtering algorithm (see Chapter 9). This is a line-search algorithm that imposes *sufficient decrease* along a quasi-Newton direction. The main difference from derivative-based methods is that the true gradient is replaced by the simplex gradient. So, to some extent, the implicit-filtering method resembles a quasi-Newton approach based on finite-difference approximations to the gradient of the objective function. However, the method and its implementation [141] are particularly well equipped to handle noisy functions. The last method is a trust–region-based algorithm in which the quadratic models are built from polynomial interpolation or regression. We used the implementation from the `DFO` code of Scheinberg (see Chapter 11 and the appendix). Both methods can adapt well to the curvature of the function, the implicit-filtering one being less efficient than the trust-region one but more capable of filtering the noise (perhaps due to the use of the simplex gradient, which corresponds to the gradient of a linear interpolation or regression model, and to an inaccurate line search).

The results are reported in Figure 1.2 for the two functions of Figure 1.1 and follow the tendency known for these classes of methods. The type of plots of Figure 1.2 is widely used in derivative-free optimization. The horizontal axis marks the number of function evaluations as the optimization process evolves. The vertical axis corresponds to the value of the function, which typically decreases. We set the stopping tolerance for all four methods to 10^{-3} (related to the different parameters used for controlling the step sizes).

On the Rosenbrock function, the trust-region method performs the best because of its ability to incorporate curvature information into the models (from the start to the end of the iteration process). We chose the initial point $(1.2, 0)$ for all methods. As expected, coordinate search is reasonably fast at the beginning but is rather slow on convergence. Attempts to make it faster (like improving the poll process) are not very successful in this problem where curvature is the dominant factor (neither would other fancier directional direct-search methods of poll type). If we start coordinate search form $(1, -1)$ (curvature more favorable), coordinate search will do much better. But if we start it from $(-1, 1)$, it will perform much worse. The other two methods perform relatively well. It is actually remarkable how well a *direct-search* method like Nelder–Mead performed and seems to be able to exploit the curvature in this problem.

The results for the perturbed quadratic are less clear, as happens many times in derivative-free optimization. The Nelder–Mead method failed for the initial point chosen $(0.1, 0.1)$, although it would do better if we started not so close to the solution (but not always...). The method performs many inside contractions which are responsible for its lack of convergence (see Chapter 8 for more details). The implicit-filtering method does a

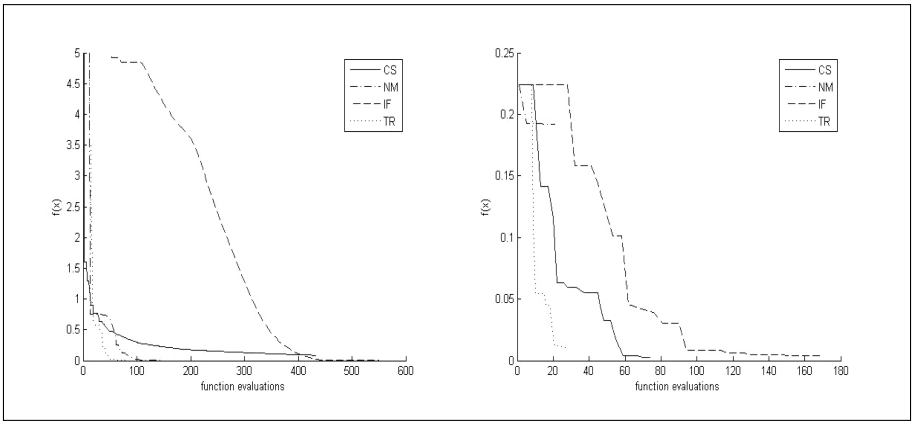

Figure 1.2. *Results of coordinate-search, Nelder–Mead, implicit-filtering, and trust-region interpolation-based methods for the Rosenbrock function (left) and a perturbed quadratic (right).*

good job in filtering the noise for this problem and progressing towards the solution (and its performance is not affected by the starting point). The interpolation-based trust-region approach is very efficient at the beginning but soon stagnates—and reducing the tolerance does not help in this case. Coordinate search performs relatively well (remember that the curvature is mild) and does not seem affected by the noise.

In summary, we would say that model-based methods (like interpolation with trust regions) are more efficient. Direct search loses in comparison when the function is smooth and the curvature is adverse, but it offers a valid alternative for noisy or nonsmooth problems and can be successfully combined with model-based techniques. The ease of parallelization is also one of its strongest points. There is certainly room for other methods like modified, safeguarded Nelder–Mead methods and for approaches particularly tailored for noisy problems and easy to parallelize like implicit filtering.

1.5 A short summary of the book

Thus, having convinced you, perhaps, that there is a need for good derivative-free methods we hope there is also some intellectual encouragement for pursuing research in the area, or at least continuing to read this book. The basic ideas, being reasonably simple, lead to new interesting results. Hence, besides being useful, it is fun.

After the introduction in this chapter, we begin the first part of the book dedicated to *Sampling and Modeling*. Thus Chapter 2 introduces the reader to positive spanning sets and bases, linear interpolation and regression models, simplex gradients, and the importance of geometry. It also includes error bounds in the linear case. Chapters 3, 4, and 5 then consider nonlinear polynomial interpolation models in a determined, regression and underdetermined form, respectively. They give due consideration to the concept of well poisedness, Lagrange polynomials, the conditioning of appropriate matrices, and Taylor-

type error bounds. Chapter 6 is devoted to constructive ways to ensure that well poisedness holds and to prepare the material on derivative-free models for use in model-based algorithms such as the trust-region ones.

The second part of the book on *Frameworks and Algorithms* begins in Chapter 7, which addresses direct-search methods where sampling is guided by desirable sets of directions. Included is global convergence with integer lattices and sufficient decrease. The next chapter covers direct-search methods based on simplices and operations over simplices, of which a classical example is the Nelder–Mead method mentioned earlier, for which we include a globally convergent variant. Chapter 9 is devoted to line-search methods based on simplex derivatives, establishing a connection with the implicit-filtering method. Chapter 10 presents trust–region-based methods, including the relationship with derivative-based methods, the abstraction to fully linear and fully quadratic models, and a comprehensive global convergence analysis. The more practical aspects of derivative-free trust-region methods with particular examples of modified versions of existing methods are covered in Chapter 11, in connection with the material of Chapter 6.

Finally, Chapters 12 and 13 (the third part of the book) are concerned with some relevant topics not covered in full detail. In Chapter 12 we review surrogate models built by techniques different from polynomial interpolation or regression, and we describe rigorous optimization frameworks to handle surrogates. A survey of constrained derivative-free optimization is presented in Chapter 13, where we also discuss extension to other classes of problems, in particular global optimization. The book ends with an appendix reviewing the existent software for derivative-free optimization.

The reader is also referred to a number of survey papers on derivative-free optimization, namely the more recent ones by Conn, Scheinberg, and Toint [60], Kelley [141], Kolda, Lewis, and Torczon [145], Lewis, Torczon, and Trosset [156], Nocedal and Wright [178, Chapter 9], Powell [187], and Wright [231], as well as older ones by Brent [47], Fletcher [94], and Shawn [207].

Notation

In this book we have tried to use intuitive notation to simplify reading. This inevitably implies some abuse, but we hope the meaning will nevertheless be clear to the reader. For instance, vectors can be considered row vectors or column vectors according to context without always changing the notation or using transposes.

The uppercase letters typically denote sets or matrices. There are several constants denoted by κ with a subscript acronym; for instance, κ_{bhm} stands for the constant bound on the Hessian of the model. Each acronym subscript is intended to help the reader to remember the meaning of the constant.

The big-$\mathcal{O}$ notation is used in an intuitive way, and the meaning should be clear from the text. All balls in the book are considered closed. All norms are ℓ_2-norms unless stated otherwise.