Evaluating bound-constrained minimization software[*]

Ernesto G. Birgin[†] Jan M. Gentil[†]

August 23th, 2011[‡]

Abstract

Bound-constrained minimization is a subject of active research. To assess the performance of existent solvers, numerical evaluations and comparisons are carried on. Arbitrary decisions that may have a crucial effect on the conclusions of numerical experiments are highlighted in the present work. As a result, a detailed evaluation based on performance profiles is applied to the comparison of bound-constrained minimization solvers. Extensive numerical results are presented and analyzed.

Key words: Bound-constrained minimization, benchmarking, numerical evaluation, performance profiles.

AMS Subject Classification: 90C30, 49K99, 65K05.

1 Introduction

The development of nonlinear optimization software is a very active field of research. On one hand, many solvers have achieved a maturity stage and are being used for tackling a wide range of applied problems in areas such as Chemistry, Economy, Engineering, Medicine, and Physics, just to name a few. On the other hand, new methods are frequently proposed and their effectiveness and efficiency need to be assessed. In any case, testing and comparing solvers and proposals is required, and coming to conclusions is as tricky as developing fair testing environments. Performance profiles [17] and data profiles [23] have become standards for presenting numerical comparisons in the last years. However, their usage requires several arbitrary decisions to be made, which may influence the obtained conclusions.

In the present work, we focus on bound-constrained minimization software. A numerical evaluation of the most well-known open-source solvers plus fmincon [10, 11] is conducted. fmincon was included in the comparison due to its popularity among the huge number of Matlab users. Open-source solvers comprised Algencan [1, 2], ASA [21], Ipopt [24], Lancelot B [14, 20], L-BFGS-B [9, 25, 22], and SPG [6, 7]. ASA and L-BFGS-B are solvers developed for bound-constrained minimization. fmincon calls different methods (SQP, active-set strategies, interior-point methods and trust-region reflective) depending on the problem at hand. SPG targets convex-constrained minimization. Algencan, Ipopt and Lancelot B are nonlinear programming (NLP) solvers. In this study we are interested in the application of the former methods to bound-constrained minimization assuming that, in some way, their performances on this simpler case may have some relation to their performances in the harder NLP case. This is true for Augmented Lagrangian methods like Algencan and Lancelot B in the following way. Algencan makes use a bound-constrained

[*]This work was supported by PRONEX-CNPq/FAPERJ (E-26/111.449/2010-APQ1), FAPESP (2006/53768-0, 2006/03496-3, 2007/56645-9 and 2009/10241-0) and CNPq (Grant 304484/2007-5).

[†]Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, São Paulo SP, Brazil. e-mail: {egbirgin | jgmarcel}@ime.usp.br

[‡]Typos corrected on September 12th, 2011, and January 10th, 2012.

minimization solver named Gencan [3, 4, 5] in order to solve the Augmented Lagrangian subproblems, while Lancelot B employs SBMIN [12, 13] for the same purpose. Should a numerical evaluation reveal, for example, that ASA is "better" than Gencan for solving bound-constrained minimization problems (see [21]), it could imply that Algencan using ASA for solving the Augmented Lagrangian subproblems would perform better than the current version of Algencan for solving NLP problems. To check this possibility was, in fact, the motivation of the present work.

Summing up, the issue of whether to use performance profiles or data profiles for comparing solvers is debated in the present work. Some decisions that have to be taken for building the performance profiles are tackled. A detailed evaluation based on performance profiles is applied to the comparison of the aforementioned bound-constrained minimization solvers. The remainder of the paper is organized as follows. In Section 2, the evaluation framework is described. Section 3 is devoted to the numerical experiments. Alternative approaches are discussed on Section 4. Conclusions are drawn in Section 5.

2 Comparison framework

2.1 Performance profiles or data profiles?

Consider m methods $M_1, \dots, M_m$ and p problems $P_1, \dots, P_p$ and let t_{ij} be a metric of the effort that method M_i made in problem P_j in order to arrive to a point with functional value f_{ij}. It is assumed that the metric t_{ij} is such that the smaller its value, the higher the performance of method M_i on problem P_j. Moreover, let $t_j^{\min}$ denote the smallest among all the performance measurements required by each method that "found a solution" for problem P_j. In performance profiles, each method M_i is related to a curve

$$\Gamma_i(\tau) = \frac{\#\{j \in \{1, \dots, p\} \mid M_i \text{ found a solution for } P_j \text{ with } t_{ij} \le \tau\, t_j^{\min}\}}{p}, \tag{1}$$

where $\#\mathcal{S}$ denotes the cardinality of set $\mathcal{S}$. Performance profiles are useful to graphically represent a comparison between several methods on a large set of test problems. They show the fraction $\Gamma_i(\tau)$ of problems a method M_i solved within a prescribed limit on its performance measurement (like, for example, CPU time). For each problem, the imposed limit is a proportion $\tau \ge 1$ of the performance measurement of the most efficient method for this particular problem. It means that, for a method M_i, $\Gamma_i(\tau \equiv 1)$ represents the fraction of problems for which the method was the most efficient over all the methods. On the other hand, $\Gamma_i(\tau \equiv \infty)$ represents the fraction of problems solved by method M_i, irrespective of the required effort. Therefore, the fraction $\Gamma_i(\tau \equiv 1)$ is usually associated with the *efficiency* of method M_i, while $\Gamma_i(\tau \equiv \infty)$ is associated with its *robustness*.

Performance profiles were designed to give "easy" and "hard" problems the same importance within the test set, where by "easy" it should be understood that the problem can be rapidly or effortlessly solved. This decision may, as a matter of fact, strongly affect the conclusions of a comparison. If, on the other hand, it is assumed that hard-to-solve, or, equivalently, time-consuming problems should be more relevant in the comparison, performance profiles (1) might be replaced by data profiles

$$\widehat{\Gamma}_i(\tau) = \frac{\#\{j \in \{1, \dots, p\} \mid M_i \text{ found a solution for } P_j \text{ with } t_{ij} \le \tau\}}{p}. \tag{2}$$

Data profile $\widehat{\Gamma}_i(\tau)$ represents the fraction of problems method M_i is able to solve within a prescribed limit on its performance measurement (like CPU time or number of functional evaluations). The difference with the performance profile is that the limit is independent of the behavior of the other methods being tested.

Are time-consuming problems more important than rapidly-solved problems when analyzing the behavior of two or more methods? There is no clear answer to that question and, hence, any assumption that shall be made must be clearly stated in every numerical evaluation.

2.2 Meaning of “to solve a problem”

When comparing methods theoretically capable of finding stationary points, how should we deal with problems in which the methods found different local minimizers? For sure, comparing the effort made by the methods to converge to different points should not be an option. Therefore, the question is whether those problems should be removed from the performance comparison or not. Although doing this is, in fact, a common practice, removing those problems with no further consideration leads to a loss of information. The elimination of the problems in which the methods arrived at qualitatively different solutions is commonly justified by the claim that a method theoretically capable of finding stationary points cannot be penalized for having found poor quality local minimizers. If on one hand it seems to be a fair reasoning, on the other hand it ignores the fact that well-designed methods accomplish more than what is guaranteed by their convergence theory.

Most box-constrained optimization methods are guaranteed to find stationary points. In practice, good methods do more than that, such as “magical steps” in the sense of [15] (pp. 387–391), which happen to be effective to increase the probability of convergence to global minimizers. The line-search procedures of Gencan [5], for example, include extrapolation steps that are not necessary from the point of view of KKT convergence. This extra job may deteriorate the efficiency of the method as it increases the number of objective function evaluations, but it enhances the probability of convergence to global minimizers. This is an example of a design choice of the developers of Gencan that aims robustness in detriment of efficiency.

If, in a hypothetical situation, two well designed methods found solutions of different quality in 20% of the test set problems and each method found solutions of better quality in half of those cases, including those problems will not affect the relative comparison among the methods — it will only reduce in 10% the robustness of each method, preserving the fact that both methods are equally robust. However, if, due to design reasons, one of the methods has a tendency to find better quality minimizers than the other and, let’s say, it found better solutions in 14% of the cases (while the opposite situation occurred in 6% of the cases), including those problems will reflect this fact by attributing a greater robustness to the first method, as it deserves. This choice will not affect the comparison if there are no main robustness differences in the methods, while it will highlight the advantages of a method over the others should some substantial differences exist in the quality of the solutions found.

The statement above brings us to the point where we need to determine whether two solutions are of equivalent quality or not. In the present work we are dealing with bound-constrained minimization problems, for which it is a trivial task to preserve feasibility of the solutions. The precise satisfaction of the bound constraints leads us to the simple case where solutions can be evaluated by comparing the objective functional value only. Since we are dealing with floating-point arithmetic, we cannot simply ask whether two functional values are equal, and a comparison considering relative errors is in order.

Let $f_1, \dots, f_m$ be the objective function values found by methods $M_1, \dots, M_m$ when applied to a given problem. Let $f^{\min} = \min\{f_1, \dots, f_m\}$ and consider

$$\varepsilon_i = \frac{f_i - f^{\min}}{\max\{1, |f^{\min}|\}}, i = 1, \dots, m. \tag{3}$$

For a given tolerance $\varepsilon^f > 0$, we say that *method M_i found a solution* if

$$\varepsilon_i \leq \varepsilon^f, \tag{4}$$

i.e. we are considering “small” absolute errors whenever $|f^{\min}| \leq 1$ and “small” relative errors otherwise. In addition, we also say that method M_i found a solution if $f_i \leq -f_\infty$, where f_∞ is a very large positive number. In this case, we assume the objective function is unbounded from below within the feasible region and any value of $f_i \leq -f_\infty$ is considered a solution. Needless to say, arbitrary choices of the threshold parameter ε^f may exert a great influence in the comparison process. Postponing the discussions related to the determination of ε^f and the choice and measurement of an appropriate performance metric, we finish this subsection with an illustrative example.

Let us assume that we have methods M_1 and M_2 and a test set with 293 problems. Assume that we used certain performance metric and some value of the threshold parameter ε^f to decide, according to (3–4), whether two solutions are equivalent or not. Figure 1a shows the performance profiles ignoring those problems in which the methods found solutions of different quality, while Figure 1b shows the performance profiles using the whole set of problems. Figure 1a says that method M_1 is more efficient than method M_2, but it gives no clue about the robustness of the methods (or, even worse, it may suggest to an unadvised reader that both methods are equally robust). On the other hand, Figure 1b, which provides very similar knowledge with respect to the efficiency of the methods, presents an extra information: M_1 is approximately 8% more robust than M_2.

The missing information in Figure 1a is the following: according to (3–4), both methods found equivalent solutions in 234 problems (the ones used to build Figure 1a) and different quality solutions in the remaining 59 problems. Among these 59 problems, method M_1 found better quality solutions in 41, while the opposite situation occurred in the other 18 problems. Summing up, using the convention introduced in (3–4), method M_1 found a solution in 275 problems, while method M_2 found a solution in 252 problems. Figure 1b shows this information on the right hand side of the graphic, saying that method M_1 has a robustness of $0.94 \approx 275/293$, while method M_2 has a robustness of $0.86 \approx 252/293$. Whether Figure 1a or Figure 1b is preferable is a matter of taste, but if Figure 1a is chosen, the information related to the disregarded problems, accompanied by every applicable warning, should be provided to the reader.

2.3 Performance metric

Whenever different versions of a certain method are subject to evaluation, the comparison of the number of iterations executed by each of them is usually enough to provide a reasonably accurate figure of their relative performance. In the derivative-free optimization case, it is well accepted that the most time-consuming task for any method is the evaluation of the objective function and, under this hypothesis, the number of functional evaluations is used as a metric to assess the performance of the methods. In any other scenario, such as that of the evaluation of distinct methods, the only metric whose comparison carries any significance is the CPU time required for task completion. Thankfully, most of the state-of-the-art software in the nonlinear programming field is open source and can be freely downloaded, compiled and run in a unified computational environment for measuring CPU time.

Even in such an ideal scenario, some special care is called for. That is because outward factors such as programming languages, compilers and compilation directives do exert some influence on the machine code generated and, thus, on the measured CPU time. Moreover, other environment aspects at runtime can also interfere with the solver's algorithmic choices — Gencan may dynamically opt to use CG instead of trust-regions or Newton in order to compute some "inner-to-the-face steps" if there is not enough memory for a direct linear-systems solver such as MA27 or MA57 to complete a matrix factorization. Another example of a factor that may have an influence in numerical evaluations including Lancelot B is related to the use of the widely accepted Cuter collection [8, 19] of test problems. In [8], p. 136, when describing the interfaces between Cuter and several solvers, it is written *"Of course, LANCELOT also solves problems in SIF, but it does not require an interface using the CUTE tools. Note that LANCELOT exploits much more structure than that provided by the interface tools."*. The efficiency gains obtained by Lancelot B by exploiting the structure of a problem coded in SIF (like the ones in the Cuter collection) over a competitor method that uses an interface remains to be elucidated. Nevertheless, even though the exact effects of the aforementioned conditions on a method's performance are not fully understood, programming languages, compiler options, software versions, operating system and platform description details should be meticulously reported when presenting numerical experiments in order to at least increase the chance of reproducibility.

Some considerations about measurement of the CPU time spent by a method during a problem's resolution are in order. On one hand, timing small intervals became a matter of growing importance with the introduction of performance profiles, which were designed to give "easy" and "hard" problems the same importance. On the other hand, it is known that tools traditionally used for measuring CPU time have a resolution somewhere between a hundredth and millionth of a second [26]. The precise resolution is platform-dependent, being a hundredth of a second one of the most common situations, as is the case of the platform where the numerical experiments of the present work were run[1]. With the aim of improving the accuracy of the measured time interval, the timing of a single execution of a method may be obtained by the timing of a sufficiently large number η of runs without interruption. Denoting by $T(\eta)$ the CPU time actually measured, the CPU time $t(\eta)$ that corresponds to the execution of a single instance of the problem can be easily computed as

$$t(\eta) = \frac{T(\eta)}{\eta}.$$

One might ask how long the interval $T(\eta)$ must be in order to guarantee that the underlying $t(\eta)$ will carry enough accuracy. Aiming at providing an answer to this question, the following experiment was conducted. For a random sample of six problems, the value of $t(\eta)$ was computed for increasingly longer time intervals $T(\eta)$ and, for each one of them, a relative error was obtained as follows:

$$\varepsilon(t(\eta)) = \frac{t(\eta) - t^*}{t^*},$$

where t^* corresponds to the CPU time attained for the largest number of uninterrupted runs performed. From this data, it was possible to establish that, for values of $T(\eta)$ longer than 10 seconds, the associated relative error $\varepsilon(t(\eta))$ would be no greater than 2%. Figure 2 illustrates the results of the experiment performed with problem BQP1VAR from the Cuter collection. Performance profiles in Figures 1a and 1b were built using CPU time as the performance metric, which was measured as described in previous paragraphs.

Unfortunately, it is not always possible to modify a given method to measure the CPU time as described above. Consider $\hat{t}_{ij}$ the CPU time of a single measurement (thereby, prone to error) of method M_i applied to problem P_j, and let $\hat{t}_j^{\min}$ be the shortest among all the times required by each method that found a solution for problem P_j according to (3–4). Error measurements in $\hat{t}_{ij}$ may lead to two different inconveniences: (a) comparing small and error-prone metric measurements may lead to wrong conclusions, and (b) if $\hat{t}_j^{\min} = 0$ for some problem P_j then, by (1), problem P_j will never be considered in the curve of a method M_i that found a solution for problem P_j with $\hat{t}_{ij} > 0$.

At least three trivial and arbitrary decisions to overcome (a) and (b) may be considered:

(i) disregard problems P_j such that $\hat{t}_j^{\min} \leq 0.01$ seconds and use $t_{ij} \equiv \hat{t}_{ij}$ as a performance metric for the remaining problems;

(ii) disregard problems P_j such that all methods being considered found equivalent solutions in $\hat{t}_{ij} \leq 0.01$ seconds and use $t_{ij} \equiv \max\{0.01 \text{ seconds}, \hat{t}_{ij}\}\ \forall i$ as a performance metric for the remaining problems; or

(iii) do not disregard any problem and use $t_{ij} \equiv \max\{0.01 \text{ seconds}, \hat{t}_{ij}\}$ as a performance metric.

We dismissed option (i) since it may eliminate a problem P_j with $\hat{t}_j^{\min} \leq 0.01$ seconds even when a method M_i may have found a solution taking $\hat{t}_{ij} \gg \hat{t}_j^{\min}$, canceling the advantage of the fastest method (the one with $\hat{t}_{ij} = \hat{t}_j^{\min}$) in the comparison procedure.

Figures 3a and 3b show the single-measurement counterparts of Figure 1b corresponding to alternatives (ii) and (iii), respectively. Figure 3a shows a nice property of the test set of problems: the efficiency

[1]See [16], pp. 183–186, for a guideline to check the timing resolution of your own Linux platform.

	Metric				
	Minimum	Q_1	Median	Q_3	Maximum
# of variables	1	4	100	5000	37,311
# of bound constraints	0	0	0	12	20,000

Table 1: Minimum, maximum, median and first and third quartiles of the number of variables and bound-constraints of the 293 unconstrained and bound-constrained problems from the Cuter test set.

rates of the methods are mostly preserved even when, following alternative (ii), the 79 problems in which both methods found a solution in no more than one hundredth of a second are disregarded. It is worth noting that, for the remaining $214 = 293 - 79$ problems, 29 intervals shorter than one hundredth of a second corresponding to method M_1 were overestimated when substituted by 0.01 seconds, while the same situation occurred for method M_2 in only 7 problems. Since in these 36 problems one method took more than one hundredth of a second (otherwise the problem would have been disregarded), such overestimation does not affect the computed efficiency rate of the methods at all, its influence being restricted to the intermediate parts of the performance profile graphics. On the other hand, Figure 3a reduces the apparent robustness rates of the methods by eliminating those 79 problems. Note that the robustness rates of methods M_1 and M_2 decreased in different proportions. This difference can be easily explained. From Figure 3a, we have that M_1 solves $276 \approx 0.92 \times (293 - 79) + 79$ problems, while M_2 solves $252 \approx 0.81 \times (293 - 79) + 79$ problems. It means that the true robustness rates of methods M_1 and M_2 are $0.94 \approx 276/293$ and $0.86 \approx 252/293$, respectively, as shown in Figure 1b. If Figure 3a is to be used to illustrate a comparison, the true robustness rates of the methods should be informed.

Following (iii), Figure 3b preserves the robustness of the methods and artificially increases their efficiency by adding 79 ties. (Those 79 ties come from the overestimation of 108 and 85 measured times shorter than one hundredth of a second for methods M_1 and M_2, respectively.) The efficiency of method M_1 goes from 0.67 (in Figure 3a) to $0.76 \approx (0.67 \times (293 - 79) + 1.0 \times 79)/293$, while the efficiency of method M_2 goes from 0.33 to $0.51 \approx (0.33 \times (293 - 79) + 1.0 \times 79)/293$. If Figure 3b is to be used to illustrate a comparison, the efficiency rates of the methods computed ignoring the artificially introduced ties (0.67 and 0.33 for methods M_1 and M_2, respectively) should be reported. None of the graphics seem to be completely satisfactory, but any of them may be used, accompanied by the due clarifications, to present the results of a numerical comparison.

3 Numerical experiments

In the numerical experiments, we considered 293 problems from the Cuter collection [19] (version.date: 'CUTEr: Mon Jan 8 15:36:20 EST 2007'). It corresponds to all the unconstrained and bound-constrained problems from the Cuter collection with the exception of problem WALL100, for which we were not able to run the interface subroutines. Test problems are of the form

$$\text{Min } f(x) \text{ subject to } x \in \Omega,$$

where $\Omega = \{x \in \mathbb{R}^n \mid \ell \leq x \leq u\}$, $\ell, u \in \mathbb{R}^n$, $\ell \leq u$, and $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. A few figures related to the test set are shown in Table 1. The stopping criterion associated with successful convergence, common to all the considered methods, was

$$\|P_\Omega(x - \nabla f(x)) - x\|_\infty \leq \varepsilon^g, \tag{5}$$

where $P_\Omega(\cdot)$ represents the Euclidean projection onto Ω.

In the numerical experiments we considered Algencan 2.3.7, ASA 2.2, fmincon included in the Matlab Optimization Toolbox Software Version 4.1 (R2008b), Ipopt 3.9.3, Lancelot B included in GALAHAD version 2.40003, L-BFGS-B (modified on April 25, 2011) and SPG (downloaded from the TANGO Project web page on 08/10/2011).[2] In Algencan, Ipopt and Lancelot B, we provided every HSL subroutine required to improve their performance, but not Metis. In all cases we preserved software's default parameters. It is worth noting that Algencan, fmincon, Ipopt, and Lancelot B use second-order information, while ASA, L-BFGS-B, and SPG use only first-order information. Software was compiled with GNU Fortran (gfortran) and GCC version 4.3.3. The compiler optimization option -O4 was adopted. All the experiments were executed on a 2.4GHz Intel Core 2 Quad Q6600 with 4.0GB of RAM memory running the GNU/Linux Operating System. CPU time measuring was made by calling to the Cuter tool `creprt` designed to obtain statistics concerning function evaluations and CPU time.

We opted for using performance profiles to present the results of the numerical experiments. Dealing with a large set of mostly academic test problems, we believe that easy-to-solve instances have the same importance as the harder ones. This is because similar easy-to-solve bound-constrained problems may have to be processed a huge number of times when nonlinear programming tools are employed as sub-algorithms in the context of Global Optimization or Mixed-Integer Nonlinear Programming.

In the performance profiles, we relied on CPU time as the performance measurement. The CPU time of each pair method/problem was limited to thirty minutes. This limitation should be taken into account in the global performance analysis. Note that, according to the concepts of efficiency and robustness applied in the present work, a short time limit may depict an inefficiency case as a failure, affecting the robustness of a method.

Because we understand that providing a robustness analysis of the methods being tested is one of the main features of the present comparison, we decided to build the performance profiles using the whole set of test problems, i.e., including those for which the methods found different quality solutions as well. With regard to considering or not problems for which all methods found equivalent solutions in at most one hundredth of a second, the numerical experiments will show that it makes no difference in the present analysis — fmincon never finished running in one hundredth of a second or less. It is worth noting that the present performance study refers to a comparative analysis between the seven methods being considered, and that the exclusion or inclusion of a method may modify the relative comparison among the methods.

In order to run the tested methods, a value for the threshold parameter ε^g in the stopping criterion (5) has to be determined. This is a dimensional parameter and there is no clear rule to establish its value. Moreover, the difficulty in achieving small values of the sup-norm of the projected gradient might vary according to whether a method makes use of second-order information or not. To cope with this situation, we performed five different runs of each pair method/problem for $\varepsilon^g \in \{10^{-4}, 10^{-5}, \dots, 10^{-8}\}$. With all this data at hand, we needed to determine a value for the threshold parameter ε^f in (4), as well as for $-f_\infty$. Once again, since there is no easy way to select an arbitrary value for ε^f, for each possible choice of ε^g we present the results for every $\varepsilon^f \in \{10^{-4}, 10^{-5}, \dots, 10^{-8}\}$ such that $\varepsilon^f \geq \varepsilon^g$. We also arbitrarily set $f_\infty = 10^{20}$.

Considering all possible combinations of ε^g and ε^f, we arrived at fifteen different performance profiles. It is a common practice in most of the published papers that make use of performance profiles to only present one of those combinations. Table 2 shows the efficiency and the robustness rates of each of the seven methods in all fifteen possible performance profiles. Roughly speaking, Table 2 makes it clear that Lancelot B appears as the most robust method for almost any combination of ε^g and ε^f. The second place in the robustness ranking is disputed between Ipopt for loose values of the tolerance ε^f and Algencan for tight values of the tolerance ε^f. The third place is occupied by Algencan in the former case and by ASA in the later one. The first three places in the ranking of efficiency rates are shared between Algencan, ASA, Ipopt and Lancelot B, being Algencan in first place in fourteen out of the fifteen combination of ε^g

[2]As of this writing, these are the newest versions of all the considered open-source solvers.

and ε^f. Figures 4a and 4b show the performance profiles for the extreme cases $\varepsilon^g = \varepsilon^f = 10^{-8}$ and $\varepsilon^g = \varepsilon^f = 10^{-4}$, respectively. Note that the top of the ranking is shared by Lancelot B and Algencan in the former case, and by Ipopt, Algencan, and Lancelot B in the later one. The most notable difference refers to Ipopt, which ascended from the fourth position in the former to the first one in later case. This variation in the efficiency and robustness rates as a function of the threshold parameter ε^f is illustrated in Figure 5. As expected, the efficiency remains almost constant, while the robustness rates decrease when ε^f becomes smaller. All methods show a similar behavior with the exception of Ipopt — many final iterates of Ipopt are not considered solutions according to (3–4) when ε^f is small, causing a larger deterioration in its robustness.

For bound-constrained minimization, fmincon consists in the Trust Region Reflective Algorithm [10, 11] due to historical reasons. Development policies that can be loosely described as "do not modify the current behavior of the method in a given problem" prevent its developers from updating their code. At its current state, fmincon features a large number of alternative stopping criteria related to lack of progress, which cause the method to stop prematurely with respect to the final objective functional value. Seeking efficiency, several solvers use to have some stopping criteria associated with "lack of progress" that halt the execution of the method when it seems that no further progress can be made. In cases in which the function value found is considered a solution by the comparison procedure, this premature stop favors the solver, presenting it as an efficient method. Note that other solvers may struggle through hundreds of iterations with slow progress, trying to satisfy the required stopping criterion associated with convergence, but without making any significant improvement in the objective function. On the other hand, if a solver stops its execution (due to a supposed lack of progress) and the objective function value found is not considered a solution by the comparison procedure, the premature stop makes the solver appear to be less robust than its competitors. This whole situation is neither bad nor good, it is just the reflection of design choices made by developers. Of course, in general, solvers allow the user to inhibit those alternative stopping criteria related to lack of progress and any experienced user is able to bypass them if desired. However, in the present work we opted to run all the solvers with their default parameters and to attempt to explain the results.

An explanation connecting the results being reported in the present work and the ones presented in [21] is in order. We consider the result of a method M_1 more satisfactory than the result of a method M_2 in two situations: when M_1 finds a feasible point and M_2 does not, and when both find feasible points and the objective function value attained by M_1 is sufficiently smaller than the one obtained by M_2. In any of these two situations, we consider M_1 more robust than M_2. If both methods arrive at feasible points with similar functional values, we consider the method that consumed less CPU time the most efficient one. In the context of bound-constrained minimization, returning a feasible point is a trivial task. Therefore, we claim that a comparison should be centered around the objective function values obtained by the methods (the picture is far more complicated in the presence of nonlinear constraints). The whole comparison procedure elaborated on the present work is based on that foundation. A different perspective was taken in [21]. From the originally considered set of test problems, those for which different local minimizers were found by the methods were discarded, as were those in which all methods stopped in no more than one hundredth of a second. The stopping criterion adopted was (5) with $\varepsilon^g = 10^{-6}$. The specific test used to determine whether local minimizers were equivalent or not, as well as a possible preference of a certain solver for better quality local minimizers, were not reported. Note that a comparison based only on problems for which all the methods found equivalent solutions can only arrive at conclusions related to the efficiency of the methods, but not to their robustness.

Irrespective of the different methodologies used for comparison, there are two other possible sources of discrepancies: the test sets of problems are different and the methods being compared themselves were updated since the publication of [21]. One of the main changes in the active-sets-based method Gencan is that it was originally developed [5] as a first-order method that estimated Hessian-vector products by means of incremental quotients to compute truncated-Newton directions using CG within the faces.

	Efficiency					Robustness				
$\varepsilon^f =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`Algencan`										
$\varepsilon^g = 10^{-4}$	0.5324	—	—	—	—	0.8601	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.5392	0.5324	—	—	—	0.8635	0.8430	—	—	—
$\varepsilon^g = 10^{-6}$	0.5290	0.5290	0.5358	—	—	0.8703	0.8567	0.8294	—	—
$\varepsilon^g = 10^{-7}$	0.5529	0.5495	0.5563	0.5529	—	0.8771	0.8635	0.8430	0.8294	—
$\varepsilon^g = 10^{-8}$	0.5631	0.5563	0.5700	0.5700	0.5597	0.8771	0.8669	0.8464	0.8328	0.8157
`ASA`										
$\varepsilon^g = 10^{-4}$	0.5358	—	—	—	—	0.7679	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.5188	0.4983	—	—	—	0.8362	0.8020	—	—	—
$\varepsilon^g = 10^{-6}$	0.5085	0.4915	0.4846	—	—	0.8396	0.8225	0.7952	—	—
$\varepsilon^g = 10^{-7}$	0.4710	0.4539	0.4505	0.4437	—	0.8430	0.8259	0.8123	0.7952	—
$\varepsilon^g = 10^{-8}$	0.4573	0.4505	0.4369	0.4334	0.4164	0.8464	0.8396	0.8191	0.8123	0.7747
`fmincon`										
$\varepsilon^g = 10^{-4}$	0.0034	—	—	—	—	0.4608	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.0137	0.0137	—	—	—	0.5631	0.4676	—	—	—
$\varepsilon^g = 10^{-6}$	0.0137	0.0137	0.0068	—	—	0.6041	0.5427	0.4744	—	—
$\varepsilon^g = 10^{-7}$	0.0137	0.0137	0.0137	0.0068	—	0.6314	0.5973	0.5529	0.4881	—
$\varepsilon^g = 10^{-8}$	0.0102	0.0102	0.0102	0.0102	0.0068	0.6485	0.6246	0.6007	0.5358	0.4778
`Ipopt`										
$\varepsilon^g = 10^{-4}$	0.3549	—	—	—	—	0.8874	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.3754	0.3857	—	—	—	0.8908	0.8874	—	—	—
$\varepsilon^g = 10^{-6}$	0.3891	0.3959	0.3857	—	—	0.8874	0.8840	0.7952	—	—
$\varepsilon^g = 10^{-7}$	0.3754	0.3891	0.3788	0.3891	—	0.8805	0.8771	0.7918	0.7713	—
$\varepsilon^g = 10^{-8}$	0.3618	0.3720	0.3618	0.3686	0.3686	0.8840	0.8840	0.7986	0.7747	0.7645
`Lancelot B`										
$\varepsilon^g = 10^{-4}$	0.4471	—	—	—	—	0.8532	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.4334	0.4437	—	—	—	0.8874	0.8703	—	—	—
$\varepsilon^g = 10^{-6}$	0.4437	0.4471	0.4471	—	—	0.8908	0.8771	0.8635	—	—
$\varepsilon^g = 10^{-7}$	0.4471	0.4471	0.4505	0.4505	—	0.9010	0.8840	0.8669	0.8464	—
$\varepsilon^g = 10^{-8}$	0.4471	0.4471	0.4471	0.4471	0.4369	0.9044	0.8908	0.8737	0.8567	0.8294
`L-BFGS-B`										
$\varepsilon^g = 10^{-4}$	0.3481	—	—	—	—	0.7713	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.3481	0.3481	—	—	—	0.7918	0.7611	—	—	—
$\varepsilon^g = 10^{-6}$	0.3379	0.3413	0.3413	—	—	0.7952	0.7782	0.7474	—	—
$\varepsilon^g = 10^{-7}$	0.3379	0.3413	0.3379	0.3447	—	0.8020	0.7850	0.7611	0.7440	—
$\varepsilon^g = 10^{-8}$	0.3140	0.3140	0.3106	0.3106	0.3072	0.7713	0.7645	0.7474	0.7372	0.7099
`SPG`										
$\varepsilon^g = 10^{-4}$	0.3174	—	—	—	—	0.6587	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.3208	0.3208	—	—	—	0.7952	0.7201	—	—	—
$\varepsilon^g = 10^{-6}$	0.3174	0.3174	0.3106	—	—	0.7952	0.7645	0.7201	—	—
$\varepsilon^g = 10^{-7}$	0.3140	0.3140	0.3106	0.3106	—	0.7816	0.7474	0.7235	0.7099	—
$\varepsilon^g = 10^{-8}$	0.3072	0.3072	0.3106	0.3072	0.2969	0.7679	0.7509	0.7270	0.7133	0.6792

Table 2: Efficiency and robustness rates for each combination of $\varepsilon^g \in \{10^{-4}, \dots, 10^{-8}\}$, $\varepsilon^f \in \{10^{-4}, \dots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$. In case the reader is able to see the table in colours, first, second and third places in the efficiency and robustness rankings are shown in blue, green and red, respectively.

Nowadays, Gencan incorporates second-order information in several ways: (i) the true Hessian-vector product can be used in CG, (ii) the Newton direction can be computed using a direct solver, and (iii) Gencan has embedded the trust-region approach of Betra [3] in the computation of an inner-to-the-face step. HSL direct solvers MA27 and MA57 can now be linked to Gencan in order to solve linear systems. In connection with MA27, either MC30 or MC77 can be used to scale such linear systems. One among the four possible options for taking inner-to-the-face steps is automatically chosen in Gencan depending on the dimension of the problem being solved. Naturally, the user has also the option of setting the strategy of their choice. Summing up, we believe that the differences in the comparison procedure, the set of test problems, and the updates in ASA and Gencan justify the discrepancies between the numerical experiments presented in [21] and the ones being reported here.

Returning to the matter of our comparison itself, there is a missing piece of information with respect to the performance profiles presented in Table 2, Figures 4a, 4b, and 5a that affects the computed efficiency of the methods. As we have already mentioned, none of the problems were excluded from the performance profiles by reason of being solved by all the methods in no more than one hundredth of a second. However, most of the methods had several short time intervals overestimated due to being replaced by 0.01 seconds, as suggested in Subsection 2.3. The number of times an overestimated CPU time was considered to build the fifteen performance profiles was, on average, 94 for Algencan, 66 for ASA, 0 for fmincon, 24 for Ipopt, 66 for Lancelot B, 11 for L-BFGS-B, and 13 for SPG. If we exclude fmincon from the comparison, then, on average, 55 problems are excluded from the performance profile calculation for being solved by all the remaining methods in at most one hundredth of a second. Considering the remaining problems (238 on average), Table 3 shows the efficiency and robustness rates of the fifteen performance profiles for each combination of ε^g and ε^f. The efficiency rates presented in Table 3, which does not include artificially added ties, are more realistic than the ones presented in Table 2. However, it is worth noting that the efficiency ranking induced by Table 3 is mostly identical to the one suggested by Table 2. On the other hand, the robustness rates appear to be smaller than they really are by the exclusion of, on average, 55 problems solved by all the methods in at most one hundredth of a second. The true robustness rates can be easily computed (as already shown in Section 2) and are presented on Table 4. These robustness rates are almost identical to the ones shown in Table 2 and the slight differences are justified by the exclusion of fmincon.

While performance profiles present each method in comparison to the other ones being tested and analyzed, Table 5 shows absolute information related to each method. In particular, for each method and each value of $\varepsilon^g \in \{10^{-4}, \dots, 10^{-8}\}$, Table 5 provides: Convergence – the number of problems in which a method declared to have satisfied the convergence stopping criterion (5); Unbounded – the number of problems in which the method declared the objective function to be unbounded from below within the feasible region; CPU Time – the number of problems in which the method exhausted the thirty-minutes time limit without fulfilling any other criteria; and Other – the number of times a method stopped due to an alternative stopping criteria. The last column of the table shows the "Individual robustness rate", computed as the fraction of problems in which a given method satisfied the stopping criterion (5) asked by the user or identified the objective function as unbounded. We say this rate is "individual" because it corresponds to the case in which a sole method being run by a user returns an unequivocal positive answer, without utilizing any additional information to check the quality of that solution. Algencan occupies the first place in the ranking of individual robustness rates irrespective of the value of ε^g. The second place in the ranking belongs to ASA, while Ipopt and Lancelot B share second and third places depending on the value of ε^g.

	Efficiency					Robustness				
$\varepsilon^f =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`Algencan`										
$\varepsilon^g = 10^{-4}$	0.4219	—	—	—	—	0.8270	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.4316	0.4188	—	—	—	0.8291	0.8034	—	—	—
$\varepsilon^g = 10^{-6}$	0.4213	0.4170	0.4255	—	—	0.8383	0.8255	0.7915	—	—
$\varepsilon^g = 10^{-7}$	0.4492	0.4449	0.4557	0.4515	—	0.8475	0.8347	0.8101	0.7932	—
$\varepsilon^g = 10^{-8}$	0.4730	0.4647	0.4835	0.4835	0.4797	0.8506	0.8423	0.8182	0.8017	0.7846
`ASA`										
$\varepsilon^g = 10^{-4}$	0.4262	—	—	—	—	0.7131	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.3974	0.3718	—	—	—	0.7949	0.7521	—	—	—
$\varepsilon^g = 10^{-6}$	0.3915	0.3702	0.3574	—	—	0.8000	0.7787	0.7447	—	—
$\varepsilon^g = 10^{-7}$	0.3475	0.3263	0.3249	0.3122	—	0.8051	0.7839	0.7679	0.7468	—
$\varepsilon^g = 10^{-8}$	0.3444	0.3361	0.3223	0.3182	0.3049	0.8133	0.8050	0.7810	0.7727	0.7317
`Ipopt`										
$\varepsilon^g = 10^{-4}$	0.2025	—	—	—	—	0.8608	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2222	0.2393	—	—	—	0.8632	0.8590	—	—	—
$\varepsilon^g = 10^{-6}$	0.2426	0.2553	0.2383	—	—	0.8596	0.8553	0.7447	—	—
$\varepsilon^g = 10^{-7}$	0.2331	0.2500	0.2405	0.2489	—	0.8517	0.8475	0.7426	0.7173	—
$\varepsilon^g = 10^{-8}$	0.2282	0.2407	0.2314	0.2397	0.2520	0.8589	0.8589	0.7562	0.7273	0.7195
`Lancelot B`										
$\varepsilon^g = 10^{-4}$	0.3165	—	—	—	—	0.8186	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2906	0.3034	—	—	—	0.8590	0.8376	—	—	—
$\varepsilon^g = 10^{-6}$	0.3064	0.3106	0.3106	—	—	0.8638	0.8468	0.8298	—	—
$\varepsilon^g = 10^{-7}$	0.3136	0.3136	0.3207	0.3207	—	0.8771	0.8559	0.8354	0.8101	—
$\varepsilon^g = 10^{-8}$	0.3278	0.3278	0.3306	0.3306	0.3293	0.8838	0.8672	0.8471	0.8264	0.7967
`L-BFGS-B`										
$\varepsilon^g = 10^{-4}$	0.1983	—	—	—	—	0.7173	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.1880	0.1880	—	—	—	0.7393	0.7009	—	—	—
$\varepsilon^g = 10^{-6}$	0.1745	0.1787	0.1787	—	—	0.7447	0.7234	0.6851	—	—
$\varepsilon^g = 10^{-7}$	0.1780	0.1822	0.1814	0.1899	—	0.7542	0.7331	0.7046	0.6835	—
$\varepsilon^g = 10^{-8}$	0.1660	0.1660	0.1653	0.1653	0.1748	0.7220	0.7137	0.6942	0.6818	0.6545
`SPG`										
$\varepsilon^g = 10^{-4}$	0.1561	—	—	—	—	0.5781	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.1496	0.1496	—	—	—	0.7436	0.6496	—	—	—
$\varepsilon^g = 10^{-6}$	0.1489	0.1489	0.1404	—	—	0.7447	0.7064	0.6511	—	—
$\varepsilon^g = 10^{-7}$	0.1483	0.1483	0.1477	0.1477	—	0.7288	0.6864	0.6582	0.6414	—
$\varepsilon^g = 10^{-8}$	0.1577	0.1577	0.1653	0.1612	0.1626	0.7178	0.6971	0.6694	0.6529	0.6179

Table 3: Efficiency and robustness rates for each combination of $\varepsilon^g \in \{10^{-4}, \dots, 10^{-8}\}$, $\varepsilon^f \in \{10^{-4}, \dots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$ disregarding fmincon. In case the reader is able to see the table in colours, first, second and third places in the efficiency and robustness rankings are shown in blue, green and red, respectively.

	Robustness				
$\varepsilon^f =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
Algencan					
$\varepsilon^g = 10^{-4}$	0.8601	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.8635	0.8430	—	—	—
$\varepsilon^g = 10^{-6}$	0.8703	0.8601	0.8328	—	—
$\varepsilon^g = 10^{-7}$	0.8771	0.8669	0.8464	0.8328	—
$\varepsilon^g = 10^{-8}$	0.8771	0.8703	0.8498	0.8362	0.8191
ASA					
$\varepsilon^g = 10^{-4}$	0.7679	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.8362	0.8020	—	—	—
$\varepsilon^g = 10^{-6}$	0.8396	0.8225	0.7952	—	—
$\varepsilon^g = 10^{-7}$	0.8430	0.8259	0.8123	0.7952	—
$\varepsilon^g = 10^{-8}$	0.8464	0.8396	0.8191	0.8123	0.7747
Ipopt					
$\varepsilon^g = 10^{-4}$	0.8874	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.8908	0.8874	—	—	—
$\varepsilon^g = 10^{-6}$	0.8874	0.8840	0.7952	—	—
$\varepsilon^g = 10^{-7}$	0.8805	0.8771	0.7918	0.7713	—
$\varepsilon^g = 10^{-8}$	0.8840	0.8840	0.7986	0.7747	0.7645
Lancelot B					
$\varepsilon^g = 10^{-4}$	0.8532	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.8874	0.8703	—	—	—
$\varepsilon^g = 10^{-6}$	0.8908	0.8771	0.8635	—	—
$\varepsilon^g = 10^{-7}$	0.9010	0.8840	0.8669	0.8464	—
$\varepsilon^g = 10^{-8}$	0.9044	0.8908	0.8737	0.8567	0.8294
L-BFGS-B					
$\varepsilon^g = 10^{-4}$	0.7713	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.7918	0.7611	—	—	—
$\varepsilon^g = 10^{-6}$	0.7952	0.7782	0.7474	—	—
$\varepsilon^g = 10^{-7}$	0.8020	0.7850	0.7611	0.7440	—
$\varepsilon^g = 10^{-8}$	0.7713	0.7645	0.7474	0.7372	0.7099
SPG					
$\varepsilon^g = 10^{-4}$	0.6587	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.7952	0.7201	—	—	—
$\varepsilon^g = 10^{-6}$	0.7952	0.7645	0.7201	—	—
$\varepsilon^g = 10^{-7}$	0.7816	0.7474	0.7235	0.7099	—
$\varepsilon^g = 10^{-8}$	0.7679	0.7509	0.7270	0.7133	0.6792

Table 4: Robustness rates for each combination of $\varepsilon^g \in \{10^{-4}, \ldots, 10^{-8}\}$, $\varepsilon^f \in \{10^{-4}, \ldots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$ disregarding fmincon. This robustness rates correspond to the ones in Table 3 with the inclusion of those problems that were solved by all the six methods in at most one hundredth of a second. In case the reader is able to see the table in colours, first, second and third places in the ranking of robustness rate are shown in blue, green and red, respectively.

	Stopping criteria				Individual robustness
	Convergence	Unbounded	CPU Time	Other	
`Algencan`					
$\varepsilon^g = 10^{-4}$	273	3	10	7	0.9420
$\varepsilon^g = 10^{-5}$	272	3	10	8	0.9386
$\varepsilon^g = 10^{-6}$	268	3	11	11	0.9249
$\varepsilon^g = 10^{-7}$	265	3	12	13	0.9147
$\varepsilon^g = 10^{-8}$	262	3	12	16	0.9044
`ASA`					
$\varepsilon^g = 10^{-4}$	276	—	11	6	0.9420
$\varepsilon^g = 10^{-5}$	275	—	11	7	0.9386
$\varepsilon^g = 10^{-6}$	269	—	16	8	0.9181
$\varepsilon^g = 10^{-7}$	264	—	20	9	0.9010
$\varepsilon^g = 10^{-8}$	260	—	21	12	0.8874
`fmincon`					
$\varepsilon^g = 10^{-4}$	63	—	28	202	0.2150
$\varepsilon^g = 10^{-5}$	53	—	29	211	0.1809
$\varepsilon^g = 10^{-6}$	58	—	29	206	0.1980
$\varepsilon^g = 10^{-7}$	52	—	32	219	0.1775
$\varepsilon^g = 10^{-8}$	58	—	39	196	0.1980
`Ipopt`					
$\varepsilon^g = 10^{-4}$	270	—	13	10	0.9215
$\varepsilon^g = 10^{-5}$	267	—	13	13	0.9113
$\varepsilon^g = 10^{-6}$	264	—	13	16	0.9010
$\varepsilon^g = 10^{-7}$	258	—	14	21	0.8805
$\varepsilon^g = 10^{-8}$	253	—	14	26	0.8635
`Lancelot B`					
$\varepsilon^g = 10^{-4}$	270	—	13	10	0.9215
$\varepsilon^g = 10^{-5}$	269	—	14	10	0.9181
$\varepsilon^g = 10^{-6}$	263	—	15	15	0.8976
$\varepsilon^g = 10^{-7}$	260	—	17	16	0.8874
$\varepsilon^g = 10^{-8}$	250	—	17	26	0.8532
`L-BFGS-B`					
$\varepsilon^g = 10^{-4}$	245	2	13	33	0.8430
$\varepsilon^g = 10^{-5}$	225	2	13	53	0.7747
$\varepsilon^g = 10^{-6}$	204	2	16	71	0.7031
$\varepsilon^g = 10^{-7}$	181	2	19	91	0.6246
$\varepsilon^g = 10^{-8}$	166	2	18	107	0.5734
`SPG`					
$\varepsilon^g = 10^{-4}$	269	2	22	—	0.9249
$\varepsilon^g = 10^{-5}$	259	2	32	—	0.8908
$\varepsilon^g = 10^{-6}$	255	2	36	—	0.8771
$\varepsilon^g = 10^{-7}$	241	2	50	—	0.8294
$\varepsilon^g = 10^{-8}$	232	2	59	—	0.7986

Table 5: “Individual robustness rate” related to each method. In case the reader is able to see the table in colours, first, second and third places in the ranking of individual robustness rate are shown in blue, green and red, respectively.

	Number of times all solvers converged to the same stationary point									
$\varepsilon^f =$	10^{-4}		10^{-5}		10^{-6}		10^{-7}		10^{-8}	
$\varepsilon^g = 10^{-4}$	145	(55/90)	—		—		—		—	
$\varepsilon^g = 10^{-5}$	178	(57/121)	161	(57/104)	—		—		—	
$\varepsilon^g = 10^{-6}$	167	(56/111)	162	(56/106)	134	(56/78)	—		—	
$\varepsilon^g = 10^{-7}$	151	(54/97)	145	(54/91)	123	(53/70)	117	(53/64)	—	
$\varepsilon^g = 10^{-8}$	129	(44/85)	129	(44/85)	108	(43/65)	104	(43/61)	97	(39/58)

Table 6: Number of problems considered to compute the efficiency rates.

4 Discussion

4.1 Seeking global minimizers or stationary points?

Performance profile $\Gamma_i(\tau)$ for method M_i, as defined in (1), considers only those problems P_j, $j \in \{1, \dots, p\}$, for which method M_i *found a solution*. According to (3–4), given a tolerance $\varepsilon^f > 0$, we say that method M_i found a solution for problem P_j whenever it found a feasible point with an objective function value f_{ij} that is close to the best value $f_j^{\min}$ found by the methods being evaluated. Roughly speaking, this "best value" is taken as if it were the global minimizer of the problem, and the computational efforts of the (more robust) methods that have found a reasonable approximation to the global minimizer are being compared to determine their efficiency. The other method that did not find an approximation to the global minimizer are considered as having failed on problem P_j (affecting their robustness rate in a negative way). The fact of a method having found or not a stationary point with tolerance $\varepsilon^g > 0$ according to (5) is completely ignored. This metric strongly reflects the authors' believe that the problem being solved is a minimization problem and that the stopping criterion (5) is just a tool commonly used to stop the methods.

However, many solvers are actually designed to find local minimizers or even stationary points. So, all the efforts are directed to satisfy the stopping criterion (5). We now evaluate the methods by their capacity to find stationary points, comparing the computational effort on those problems in which all solvers satisfied (5) with tolerance $\varepsilon^g > 0$ and found the same stationary point. We say that all solvers found the same stationary point if they all satisfy (3–4) with $\varepsilon^f \in \{10^{-4}, \dots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$. This comparison is useful to evaluate the solvers efficiency when considering that their objective is to find stationary points. The solvers capacity (effectiveness or robustness) to find stationary points, was already reported on Table 5. As it can be observed in the table, fmincon stopped most of the time satisfying alternative stopping criteria and failed to satisfy criterion (5). Therefore, it is excluded from the present efficiency comparison. Moreover, as the main objective of this comparison is to assess the efficiency of the methods, we excluded from the comparison those problems for which all solvers found the same stationary point in no more than one hundredth of a second. Otherwise, we would have artificially increased the efficiency rates of the methods as already explained in Subsection 2.3.

Considering all possible combinations of ε^g and ε^f, we obtained fifteen different performance profiles. Table 6 displays, for each combination of ε^g and ε^f, the number of problems being considered to compute the efficiency rates. In Table 6, $A(B/C)$ means that, among the 293 problems being considered, all solvers satisfied the stopping criterion (5) and converged to the same stationary point in A problems; and that among those A problems, in B problems all the methods stopped in no more than one hundredth of a second, leaving C problems to compute the efficiency rates displayed in Table 7. Table 7 shows that, independently of the tolerance ε^g used in the stopping criterion (5) and independently of the tolerance ε^f used to determine whether the stationary points are equivalent or not, ASA is the most efficient solver to find stationary points, followed by Algencan and Lancelot B.

	Efficiency to find stationary points				
$\varepsilon^f =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`Algencan`					
$\varepsilon^g = 10^{-4}$	0.4444	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.4628	0.5000	—	—	—
$\varepsilon^g = 10^{-6}$	0.4595	0.4717	0.4872	—	—
$\varepsilon^g = 10^{-7}$	0.4639	0.4835	0.4857	0.4688	—
$\varepsilon^g = 10^{-8}$	0.5176	0.5176	0.5231	0.5246	0.5172
`ASA`					
$\varepsilon^g = 10^{-4}$	0.7000	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.5620	0.5673	—	—	—
$\varepsilon^g = 10^{-6}$	0.5405	0.5283	0.5641	—	—
$\varepsilon^g = 10^{-7}$	0.4845	0.4615	0.5143	0.5313	—
$\varepsilon^g = 10^{-8}$	0.4824	0.4824	0.5538	0.5738	0.5862
`Ipopt`					
$\varepsilon^g = 10^{-4}$	0.0889	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.1322	0.0962	—	—	—
$\varepsilon^g = 10^{-6}$	0.1622	0.1698	0.1154	—	—
$\varepsilon^g = 10^{-7}$	0.1443	0.1429	0.1143	0.1250	—
$\varepsilon^g = 10^{-8}$	0.1765	0.1765	0.1385	0.1475	0.1207
`Lancelot B`					
$\varepsilon^g = 10^{-4}$	0.3444	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2727	0.2981	—	—	—
$\varepsilon^g = 10^{-6}$	0.2613	0.2642	0.3333	—	—
$\varepsilon^g = 10^{-7}$	0.2165	0.2308	0.2857	0.3125	—
$\varepsilon^g = 10^{-8}$	0.2588	0.2588	0.3385	0.3443	0.3448
`L-BFGS-B`					
$\varepsilon^g = 10^{-4}$	0.3222	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2314	0.2692	—	—	—
$\varepsilon^g = 10^{-6}$	0.2072	0.2170	0.2692	—	—
$\varepsilon^g = 10^{-7}$	0.1959	0.2088	0.2714	0.2969	—
$\varepsilon^g = 10^{-8}$	0.2471	0.2471	0.3231	0.3443	0.3448
`SPG`					
$\varepsilon^g = 10^{-4}$	0.1889	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.1736	0.2019	—	—	—
$\varepsilon^g = 10^{-6}$	0.1532	0.1604	0.1923	—	—
$\varepsilon^g = 10^{-7}$	0.1649	0.1758	0.2286	0.2500	—
$\varepsilon^g = 10^{-8}$	0.2235	0.2235	0.2923	0.3115	0.3103

Table 7: Efficiency rates for each combination of $\varepsilon^g \in \{10^{-4}, \dots, 10^{-8}\}$, $\varepsilon^f \in \{10^{-4}, \dots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$ disregarding fmincon. These efficiency rates correspond to the case in which we consider that the solvers objective is to find stationary points. In case the reader is able to see the table in colours, first, second and third places in the ranking of efficiency rate are shown in blue, green and red, respectively.

	Median sup-norm of projected gradient				
$\varepsilon^g =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`Algencan`	1.9D-06	2.3D-08	2.3D-09	2.3D-10	4.0D-11
`ASA`	6.6D-05	6.7D-06	6.7D-07	7.2D-08	6.5D-09
`fmincon`	1.1D-05	4.3D-07	5.9D-09	6.3D-10	7.5D-11
`Ipopt`	8.7D-11	8.3D-11	4.3D-11	1.9D-11	1.3D-11
`Lancelot B`	2.5D-05	1.6D-06	9.0D-08	9.6D-09	8.2D-10
`L-BFGS-B`	6.8D-05	6.6D-06	5.9D-07	4.6D-08	4.8D-09
`SPG`	7.2D-05	6.5D-06	5.9D-07	6.2D-08	6.2D-09

Table 8: Median sup-norm of the projected gradient at the final iterates provided by the solvers. Medians are taken over the problems in which each solver declared to have satisfied the convergence stopping criterion.

4.2 Benefits of using second-order information

As already recalled in the previous subsection, the methodology used in the numerical experiments of Section 3 to assess the solvers' performance is based on the objective function value at the final iterate. In theory, this methodology might favor second-order methods (like Algencan, fmincon, Ipopt and Lancelot B) in the following way. In second-order methods the norm of the gradient is reduced in big chunks when the methods approach a stationary point, while for gradient-type first-order methods the gradient's norm decreases slowly. Therefore, given a tolerance $\varepsilon^g > 0$, second-order methods usually stop at a final iterate that loosely satisfies the stopping criterion (5), in contrast to first-order methods whose final iterate tightly satisfies the stopping criterion (5). Table 8 illustrates this characteristic of the second-order methods in practice, displaying the median sup-norm of the projected gradient at the final iterate of each solver. The median is taken over the cases in which each solver declared to have satisfied the stopping criterion (5). ASA, L-BFGS-B, and SPG clearly show the expected behavior of the first-order methods; while the remaining solvers show the result of using second-order information. The highlights of the figures in Table 8 are the small projected gradient sup-norms reported by Ipopt. Accompanying the smaller gradients, the second-order solvers are supposed to obtain better approximations to an accumulation point of the generated sequence. Hence, they should have better chances of satisfying (3–4) for a given tolerance $\varepsilon^f > 0$; while first-order methods should have larger chances of receiving the label of "having failed to find a reasonable approximation to the best solution found".

In [18], p. 112, it is written *"It is often thought that modified Newton methods perform well only close to the solution. This belief has caused many 'hybrid' algorithms to appear in which a different method (usually steepest descent) is used until the iterates are close to the solution. It is our experience that there is little justification for a hybrid approach, since a carefully implemented modified Newton method will make good progress at points remote from the solution."* This assertion suggests that second-order methods should be used whenever second-order derivatives are available and/or affordable. This is certainly true if high precisions (small ε^g) are required to declare convergence; while it may not be the case if loose precisions are required. This dependence of the preferable method on the required tolerance to declare convergence is the reason for considering fifteen combinations of tolerances ε^f and ε^g, instead of fixing an arbitrary choice, in the numerical comparisons of the present work. Moreover, in practice, a bunch of characteristics of the underlying method and its implementation details, other than using only first or second-order information, may influence the solver's performance. In Tables 2 and 3, it can be seen that, for fixed values of ε^g, ASA ranks in third place of robustness when (3–4) is applied using small values of ε^f; while it is outperformed by other solvers if loose values of ε^f are considered, apparently contradicting the theoretical reasoning of the previous paragraph. This phenomenon is in fact related to

	Stopping criteria				Individual robustness
	Convergence	Unbounded	CPU Time	Other	
`Algencan without using 2nd-order derivatives`					
$\varepsilon^g = 10^{-4}$	260	2	11	20	0.8942
$\varepsilon^g = 10^{-5}$	256	2	13	22	0.8805
$\varepsilon^g = 10^{-6}$	254	2	14	23	0.8737
$\varepsilon^g = 10^{-7}$	250	2	16	25	0.8600
$\varepsilon^g = 10^{-8}$	247	2	16	28	0.8498

Table 9: "Individual robustness rate" related to Algencan without using second-order derivatives.

the performance of Ipopt, which has higher robustness rates when $\varepsilon^f \in \{10^{-4}, 10^{-5}\}$ (pushing down the ranking of the remaining solvers) and lower robustness rates for $\varepsilon^f \in \{10^{-6}, 10^{-7}, 10^{-8}\}$. This behavior of Ipopt, that uses second-order information, might be related to its difficulty in approaching a solution to high precision when it is situated in the boundary of the feasible region.

It was already mentioned on Section 3 that Algencan automatically chooses whether second-order information should be used or not, depending on the dimension of the problem being solved. On the other hand, the user can also select the desired strategy. To illustrate the influence of using second-order information, we finish this subsection by presenting a comparison among the first-order methods ASA, L-BFGS-B, and SPG, and Algencan without using second-order derivatives. To do this, it is enough to use Algencan with the keywords `truncated-newton-line-search-inner-solver` and `incremental-quotients-in-cg`, which reduce Algencan's underlying method to its first-order ancestor introduced in [5] and considered in [21].

Table 9 complements Table 5 by showing the individual robustness rates of Algencan without using second-order derivatives. Table 10 shows the efficiency and the robustness rates of each of the four first-order methods in all the fifteen possible performance profiles varying ε^g and ε^f. Table 10 (that is analogous to Table 2) reflects the results of the comparison based on the objective function value at the final iterate delivered by the methods. On average, 71 (out of the 293) problems were excluded from the performance profiles calculations for being solved by all the four methods in at most one hundredth of a second. On the remaining problems, the number of times an overestimated CPU time was considered to build the fifteen performance profiles was, on average, 27 for Algencan without using second-order derivatives, 13 for ASA, 3 for L-BFGS-B, and 2 for SPG. Figures in the table show that ASA is the most efficient and robust first-order method, independently of the combination of tolerances used to build the performance profiles. In the spirit of the comparison framework described in Subsection 4.1, Table 11 shows the efficiency rates corresponding to the case in which it is considered that the solvers' objective is to find stationary points. Considering the fifteen combinations of ε^g and ε^f, on average, the four solvers found equivalent stationary points in 156 out of the 293 problems. Among the problems in which the four solvers found equivalent stationary points, on average, 65 problems were eliminated from the efficiency rates computation because all methods found the stationary point in at most one hundredth of a second, leaving 91 problems for computing the efficiency rates. Once again, figures in Table 11 present ASA as the most efficient first-order method. These results are in agreement with the ones presented in [21], showing that they are valid for the test set being considered in the present work and for the current versions of ASA and Algencan without using second-order derivatives.

	Efficiency					Robustness				
$\varepsilon^f =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`Algencan without using 2nd-order derivatives`										
$\varepsilon^g = 10^{-4}$	0.3801	—	—	—	—	0.8145	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.3624	0.3624	—	—	—	0.8257	0.8028	—	—	—
$\varepsilon^g = 10^{-6}$	0.3891	0.3846	0.3901	—	—	0.8371	0.8190	0.8072	—	—
$\varepsilon^g = 10^{-7}$	0.4009	0.3919	0.3946	0.3946	—	0.8333	0.8198	0.8072	0.7982	—
$\varepsilon^g = 10^{-8}$	0.4286	0.4196	0.4222	0.4267	0.4311	0.8393	0.8259	0.8133	0.8089	0.8089
`ASA`										
$\varepsilon^g = 10^{-4}$	0.5475	—	—	—	—	0.7647	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.5413	0.5183	—	—	—	0.8578	0.8165	—	—	—
$\varepsilon^g = 10^{-6}$	0.5430	0.5294	0.5426	—	—	0.8552	0.8416	0.8206	—	—
$\varepsilon^g = 10^{-7}$	0.5135	0.5000	0.5022	0.4888	—	0.8559	0.8423	0.8296	0.8161	—
$\varepsilon^g = 10^{-8}$	0.5268	0.5268	0.5156	0.5156	0.4889	0.8616	0.8571	0.8356	0.8356	0.8089
`L-BFGS-B`										
$\varepsilon^g = 10^{-4}$	0.2217	—	—	—	—	0.7647	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2202	0.2339	—	—	—	0.7752	0.7477	—	—	—
$\varepsilon^g = 10^{-6}$	0.1946	0.2036	0.1839	—	—	0.7783	0.7602	0.7220	—	—
$\varepsilon^g = 10^{-7}$	0.2072	0.2162	0.2108	0.2108	—	0.7748	0.7613	0.7444	0.7175	—
$\varepsilon^g = 10^{-8}$	0.1786	0.1875	0.1822	0.1689	0.1689	0.7411	0.7366	0.7244	0.7111	0.6889
`SPG`										
$\varepsilon^g = 10^{-4}$	0.1222	—	—	—	—	0.5882	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.1147	0.1147	—	—	—	0.7615	0.6743	—	—	—
$\varepsilon^g = 10^{-6}$	0.1267	0.1267	0.1211	—	—	0.7602	0.7330	0.6816	—	—
$\varepsilon^g = 10^{-7}$	0.1396	0.1441	0.1300	0.1345	—	0.7477	0.7207	0.6906	0.6771	—
$\varepsilon^g = 10^{-8}$	0.1205	0.1250	0.1200	0.1200	0.1289	0.7277	0.7188	0.6933	0.6756	0.6622

Table 10: Efficiency and robustness rates of the first-order methods for each combination of $\varepsilon^g \in \{10^{-4}, \dots, 10^{-8}\}$, $\varepsilon^f \in \{10^{-4}, \dots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$. In case the reader is able to see the table in colours, first, second and third places in the efficiency and robustness rankings are shown in blue, green and red, respectively.

	Efficiency to find stationary points				
$\varepsilon^f =$	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`Algencan without using 2nd-order derivatives`					
$\varepsilon^g = 10^{-4}$	0.3176	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2895	0.3232	—	—	—
$\varepsilon^g = 10^{-6}$	0.2477	0.2523	0.2551	—	—
$\varepsilon^g = 10^{-7}$	0.2105	0.2222	0.2222	0.2299	—
$\varepsilon^g = 10^{-8}$	0.2593	0.2593	0.2692	0.2800	0.2778
`ASA`					
$\varepsilon^g = 10^{-4}$	0.7647	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.6667	0.6667	—	—	—
$\varepsilon^g = 10^{-6}$	0.7339	0.7290	0.7551	—	—
$\varepsilon^g = 10^{-7}$	0.7368	0.7222	0.7222	0.7126	—
$\varepsilon^g = 10^{-8}$	0.7531	0.7531	0.7436	0.7333	0.7361
`L-BFGS-B`					
$\varepsilon^g = 10^{-4}$	0.2235	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.2193	0.2121	—	—	—
$\varepsilon^g = 10^{-6}$	0.1835	0.1869	0.1837	—	—
$\varepsilon^g = 10^{-7}$	0.1684	0.1778	0.1778	0.1839	—
$\varepsilon^g = 10^{-8}$	0.1605	0.1605	0.1667	0.1733	0.1806
`SPG`					
$\varepsilon^g = 10^{-4}$	0.0706	—	—	—	—
$\varepsilon^g = 10^{-5}$	0.0702	0.0707	—	—	—
$\varepsilon^g = 10^{-6}$	0.0917	0.0935	0.0918	—	—
$\varepsilon^g = 10^{-7}$	0.1053	0.1111	0.1111	0.1149	—
$\varepsilon^g = 10^{-8}$	0.0988	0.0988	0.1026	0.1067	0.1111

Table 11: Efficiency rates of the first-order methods for each combination of $\varepsilon^g \in \{10^{-4}, \dots, 10^{-8}\}$, $\varepsilon^f \in \{10^{-4}, \dots, 10^{-8}\}$ and $\varepsilon^f \geq \varepsilon^g$. These efficiency rates correspond to the case in which we consider that the solvers' objective is to find stationary points. In case the reader is able to see the table in colours, first, second and third places in the ranking of efficiency rate are shown in blue, green and red, respectively.

5 Conclusions

We employed performance profiles in a robust way to evaluate the performance of some of the most well-known open-source software for bound-constrained minimization. Whether one of them "is better" than the others depends on several issues, like, for example, availability of second-order information, size of the problem being solved, the required accuracy on the solution, and on which between efficiency or robustness is being sought. Even the definition of robustness might be considered an open question, as there is no consensus on whether the capacity to find good local minimizers or stationary points should be evaluated.

The conclusions of the numerical experiments are restricted to the application of the evaluated software to the unconstrained and bound-constrained problems from the Cuter collection. It is very clear that the number of variables (as well as other characteristics of the considered problems) do exert an influence in the results of the comparison among the solvers performance. In the extreme cases, it is very clear that for very small problems using second-order information is profitable; while for very large problems only first-order methods can be applied. While SPG did not present itself as a preferable choice for bound-constrained minimization, numerical experiments in [7] showed that it may be the most adequate method for interesting classes of convex-constrained minimization problems. On the same line of reasoning, it should be noted that the underlying algorithm evaluated in fmincon was `trust-region-reflective`, which is fmincon's default choice for bound-constrained minimization because of historical reasons. Other options within fmincon, like `active-set`, `interior-point` and `sqp`, which would very likely outperform `trust-region-reflective`, were not tested due to not being the default one.

The present conclusions apply to the software being tested and are not necessarily extensible to the underlying optimization methods. Some of the software under consideration (Algencan, Ipopt and Lancelot B) lend themselves to solving NLP problems. Whether the conclusions of the present work about the similarity among their performance can be extrapolated to the NLP case or not still remains to be determined. Moreover, in order to tackle the NLP case, the present evaluation procedure needs to be extended in order to deal with the case of comparing solutions with different degrees of feasibility.

Regarding the comparison framework, many details were meticulously worked and analyzed. In the end, the impression is that while most of the possible decisions are reasonable choices, it is important to clearly state the exact calculations being made. On the other hand, it became clear that arbitrary choices on the stopping criteria of the methods and on the decision of whether a method found a solution or not may lead to partial conclusions, which do not provide an entirely truthful description of the whole picture.

Acknowledgements: We are indebted to two anonymous referees whose comments helped us to improve the paper. We are also indebted to Prof. Bill Hager, Editor-in-Chief of Computational Optimization and Applications, whose instigating comments motivated most of the material in Section 4.

References

[1] R. Andreani, E. G. Birgin, J. M. Martínez and M. L. Schuverdt, On Augmented Lagrangian Methods with general lower-level constraints, *SIAM Journal on Optimization* 18, pp. 1286–1309, 2007.

[2] R. Andreani, E. G. Birgin, J. M. Martínez and M. L. Schuverdt, Augmented Lagrangian methods under the Constant Positive Linear Dependence constraint qualification, *Mathematical Programming* 111, pp. 5–32, 2008.

[3] M. Andretta, E. G. Birgin and J. M. Martínez, Practical active-set Euclidian trust-region method with spectral projected gradients for bound-constrained minimization, *Optimization* 54, pp. 305–325, 2005.

[4] E. G. Birgin and J. M. Martínez, A box-constrained optimization algorithm with negative curvature directions and spectral projected gradients, *Computing [Suppl]* 15, pp. 49–60, 2001.

[5] E. G. Birgin and J. M. Martínez, Large-scale active-set box-constrained optimization method with spectral projected gradients, *Computational Optimization and Applications* 23, pp. 101–125, 2002.

[6] E. G. Birgin, J. M. Martínez and M. Raydan, Nonmonotone spectral projected gradient methods on convex sets, *SIAM Journal on Optimization* 10, pp. 1196–1211, 2000.

[7] E. G. Birgin, J. M. Martínez and M. Raydan, Algorithm 813: SPG – software for convex-constrained optimization, *ACM Transactions on Mathematical Software* 27, pp. 340–349, 2001.

[8] I. Bongartz, A. R. Conn, N. I. M. Gould and Ph. L. Toint, CUTE: constrained and unconstrained testing environment, *ACM Transactions on Mathematical Software* 21, pp. 123–160, 1995.

[9] R. H. Byrd, P. Lu and J. Nocedal, A Limited Memory Algorithm for Bound Constrained Optimization, *SIAM Journal on Scientific and Statistical Computing* 16, pp. 1190–1208, 1995.

[10] T. F. Coleman and Y. Li, On the Convergence of Reflective Newton Methods for Large-Scale Nonlinear Minimization subject to bounds, *Mathematical Programming* 67, pp. 189–224, 1994.

[11] T. F. Coleman and Y. Li, An Interior, Trust Region Approach for Nonlinear Minimization subject to bounds, *SIAM Journal on Optimization* 6, pp. 418–445, 1996.

[12] A. R. Conn, N. I. M. Gould and Ph. L. Toint, Global convergence of a class of trust region algorithms for optimization with simple bounds, *SIAM Journal on Numerical Analysis* 25, pp. 433–460, 1988.

[13] A. R. Conn, N. I. M. Gould and Ph. L. Toint, Testing a class of methods for solving minimization problems with simple bounds on the variables *Mathematics of Computation* 50, pp. 399–430, 1988.

[14] A. R. Conn, N. I. M. Gould and Ph. L. Toint, *Lancelot: A Fortran package for large scale nonlinear optimization*, Springer-Verlag, Berlin, 1992.

[15] A. R. Conn, N. I. M. Gould and Ph. L. Toint, *Trust Region Methods*, MPS/SIAM Series on Optimization, SIAM, Philadelphia, 2000.

[16] J. Corbet, Alessandro Rubini, and G. Kroah-Hartman, *Linux Device Drivers*, 3rd Edition, O'Reilly, USA, 2005.

[17] E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, *Mathematical Programming* 91, pp. 201–213, 2002.

[18] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, 1986.

[19] N. I. M. Gould, D. Orban and Ph. L. Toint, CUTEr and SifDec: A Constrained and Unconstrained Testing Environment, revisited, *ACM Transactions on Mathematical Software* 29, pp. 373–394, 2003.

[20] N. I. M. Gould, D. Orban, and Ph. L. Toint, GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization, *ACM Transactions on Mathematical Software* 29, pp. 353–372, 2004.

[21] W. W. Hager and H. Zhang, A new active set algorithm for box constrained optimization, *SIAM Journal on Optimization* 17, pp. 526–557, 2006.

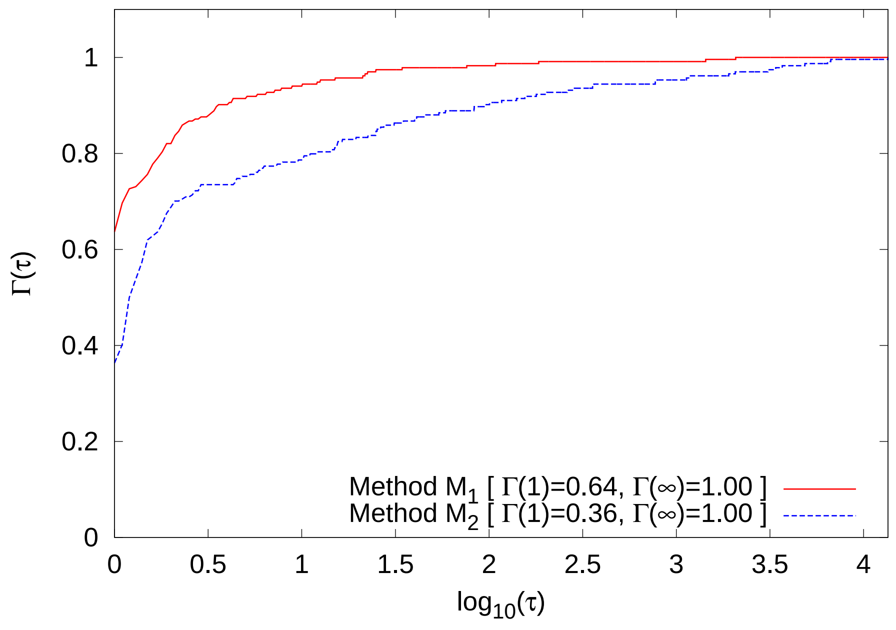
[22] J. L. Morales and J. Nocedal, Remark on "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained Optimization", *ACM Transactions on Mathematical Software*, to appear.

[23] J. J. Moré and S. M. Wild, Benchmarking derivative-free optimization algorithms, *SIAM Journal on Optimization* 20, pp. 172–191, 2009.

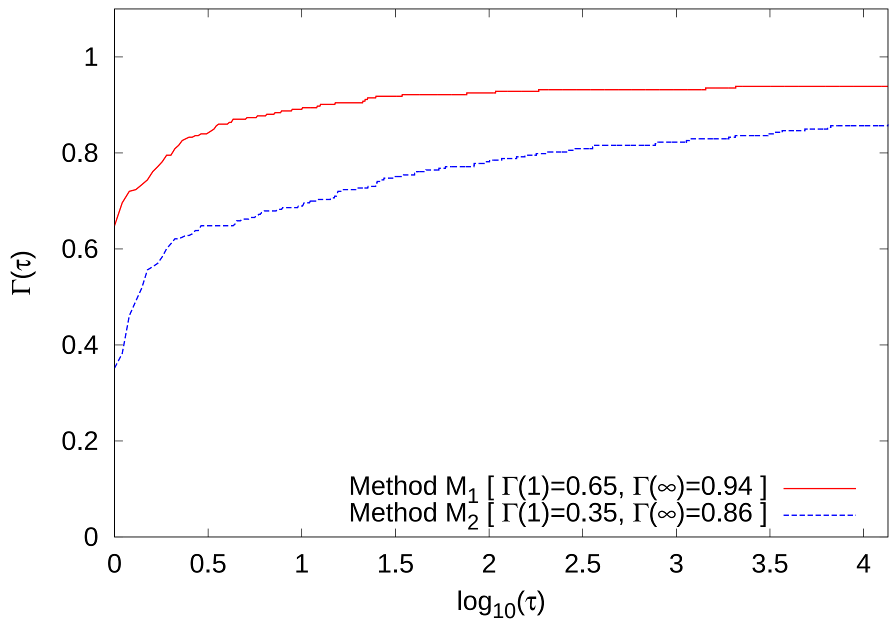
[24] A. Wächter and L. T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, *Mathematical Programming* 106, pp. 25–57, 2006.

[25] C. Zhu, R. H. Byrd and J. Nocedal, L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization, *ACM Transactions on Mathematical Software* 23, pp. 550–560, 1997.

[26] *CPU Time Inquiry*, The GNU C Library Manual, GNU Project, May 20, 2009, available at: `http://www.gnu.org/s/libc/manual/html_node/CPU-Time/html`.

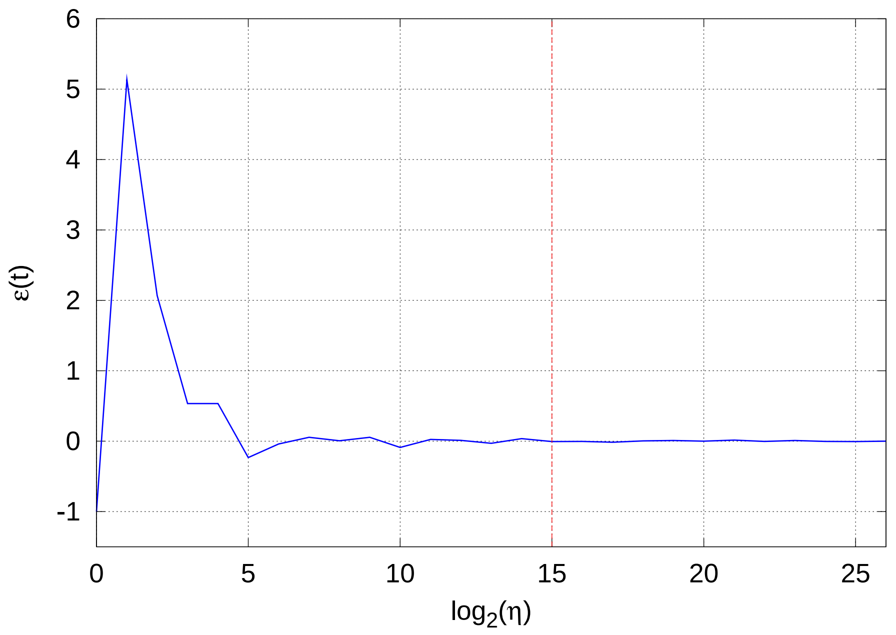


(a)

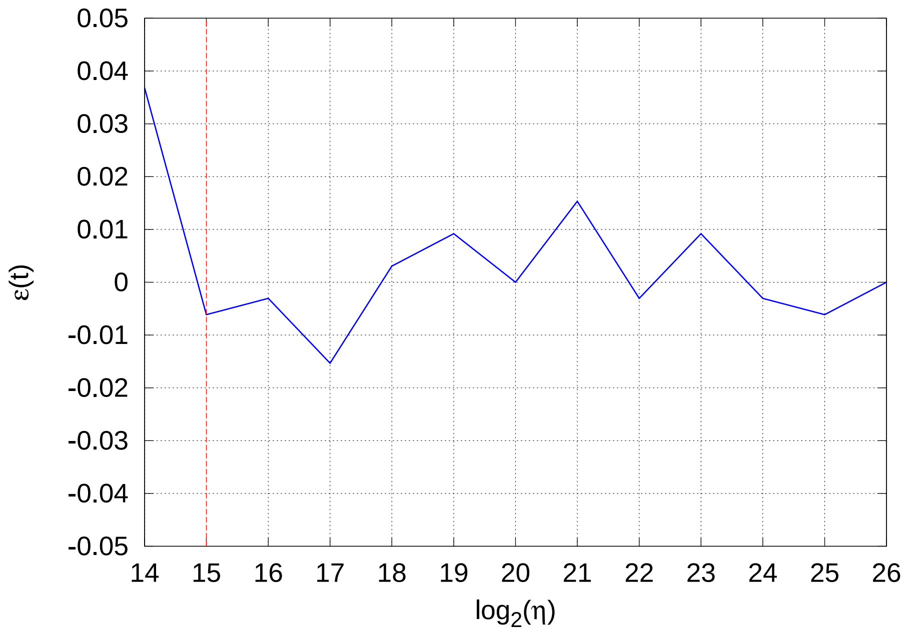


(b)

Figure 1: Performance profiles of methods M_1 and M_2. (a) Built ignoring the problems in which the methods found solutions of different quality. (b) Considering the whole set of test problems.

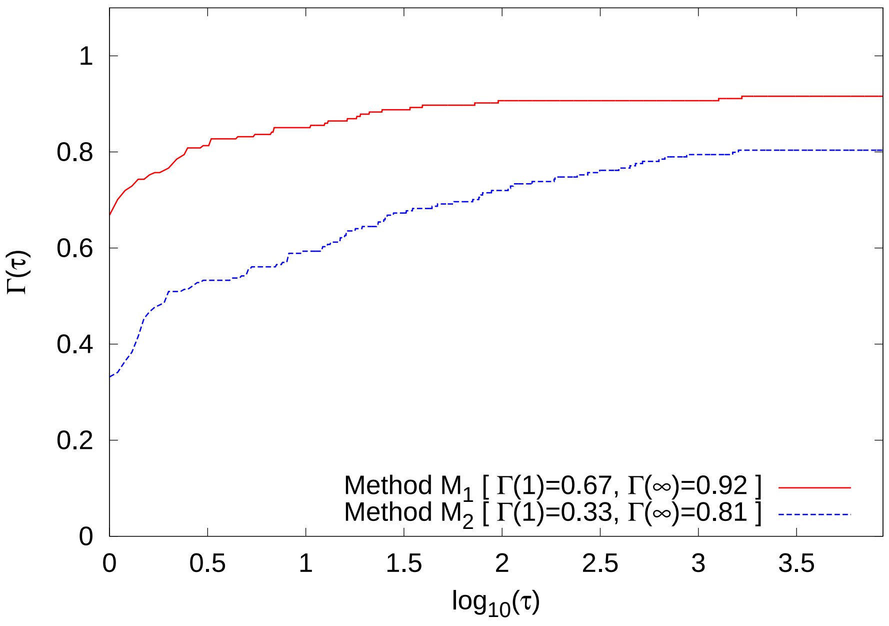


(a)

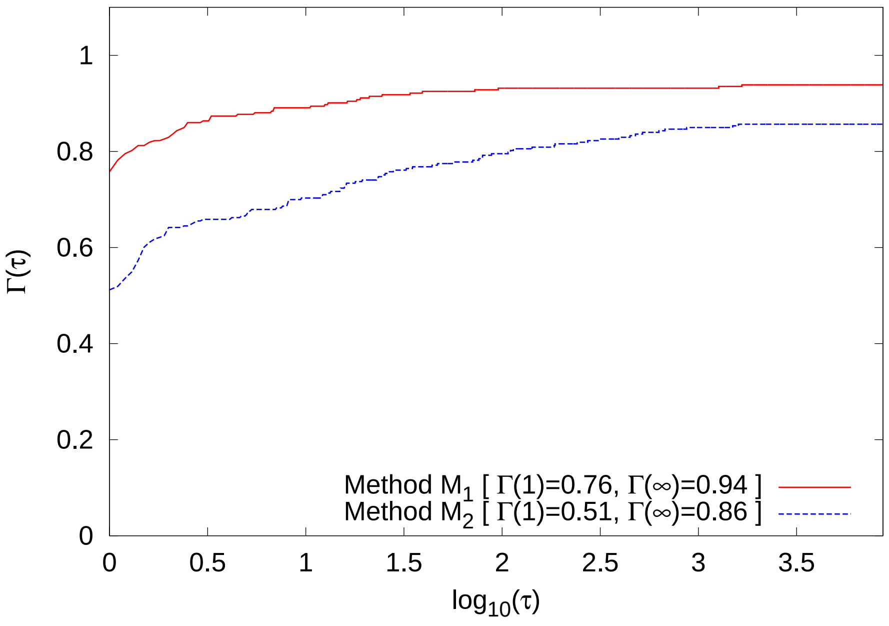


(b)

Figure 2: Problem BQP1VAR from the Cuter collection was solved η consecutive times using Gencan, for increasing values of η. The graphic (a) shows the relative error $\varepsilon(t(\eta)) = (t(\eta) - t^*)/t^*$. Graphic (b) is a zoom of the right hand side of graphic (a).

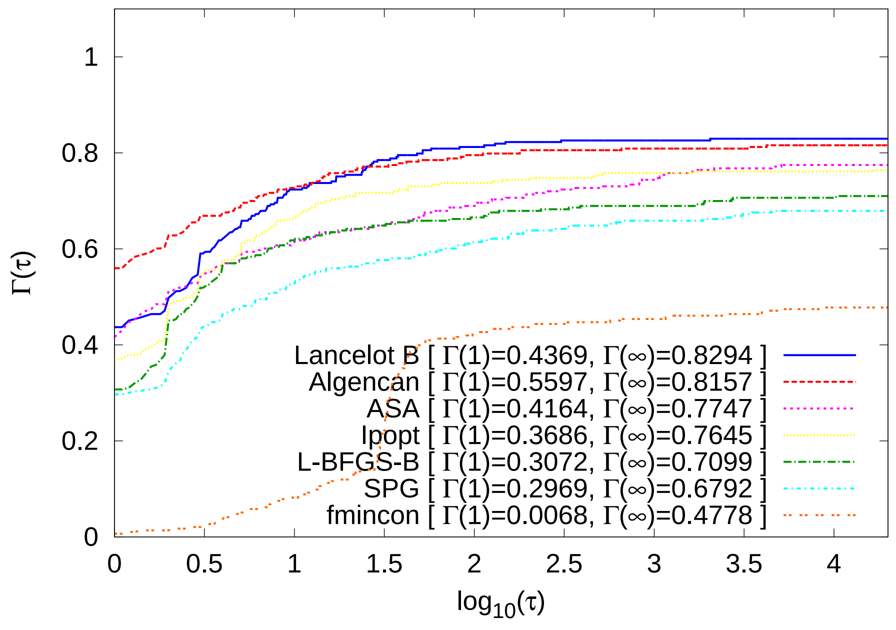


(a)

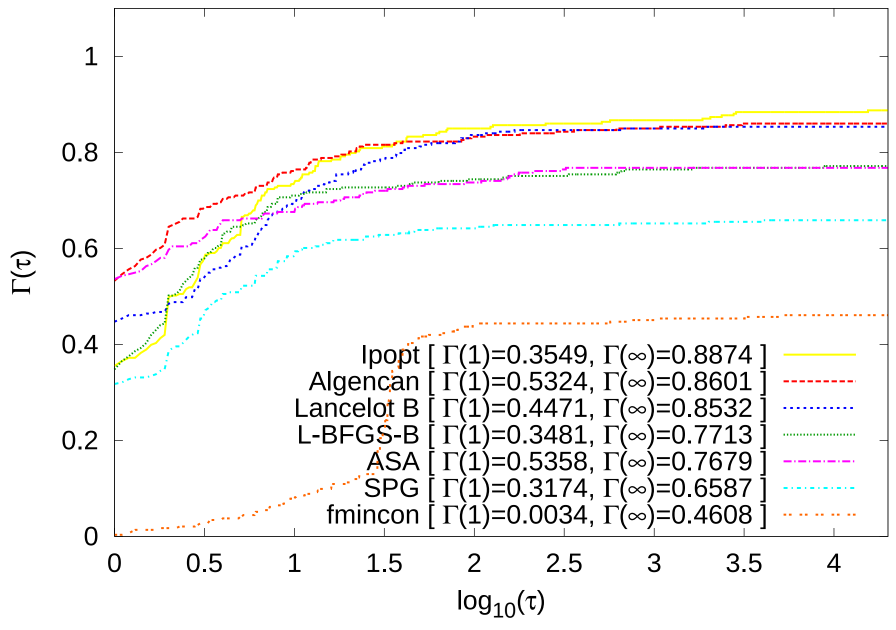


(b)

Figure 3: Performance profiles of methods M_1 and M_2 using a single-run CPU time measurement as performance metric. As in Figure 1b, problems are not eliminated when the solvers found different quality solutions. (a) Problems for which both methods found equivalent solutions in at most one hundredth of a second are disregarded. (b) The whole set of problems is used to draw the curves. In (a), as well as in (b), for every problem considered to build the curves, any measured time smaller than one hundredth of a second was replaced by 0.01 seconds.

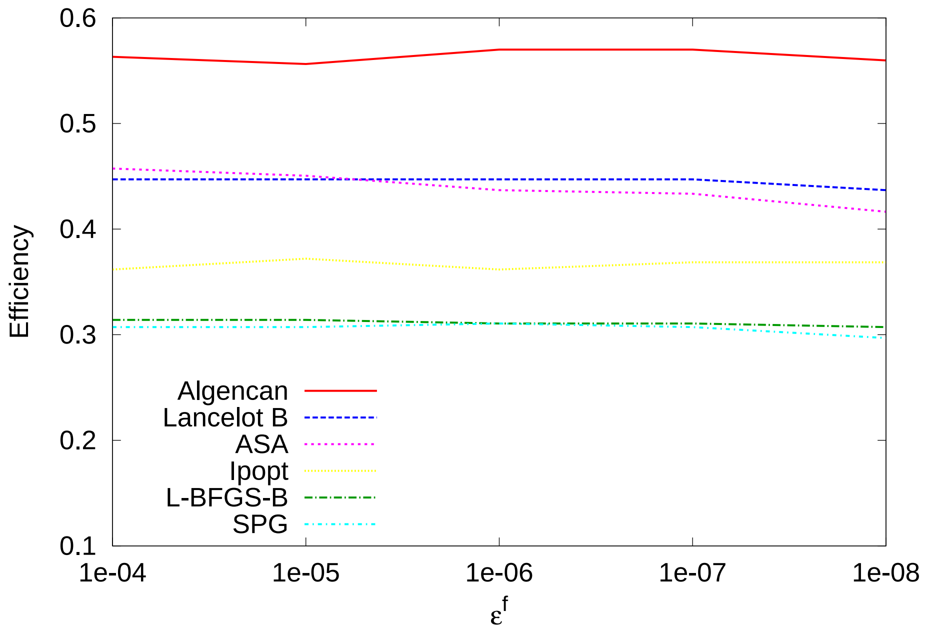


(a)

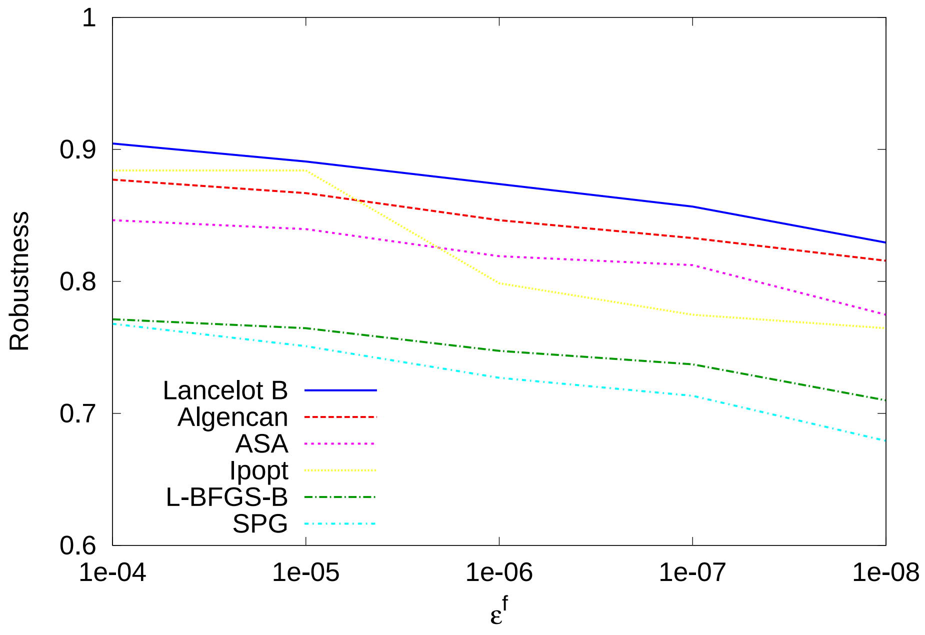


(b)

Figure 4: Performance profiles for the cases (a) $\varepsilon^g = \varepsilon^f = 10^{-8}$ and (b) $\varepsilon^g = \varepsilon^f = 10^{-4}$. They correspond to the extremes of the fifteen combinations of ε^g and ε^f presented in Table 2.



(a)



(b)

Figure 5: These graphics represent, for the case $\varepsilon^g = 10^{-8}$, the (a) efficiency and (b) robustness of the methods as a function of $\varepsilon^f \in \{10^{-4}, \ldots, 10^{-8}\}$. The curves representing the efficiency and robustness rates of fmincon are missing in these graphics because they are not visible in the scale adopted.