universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Unconstrained and bound-constrained optimization in high dimensions

verfasst von / submitted by

Morteza Kimiaei

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Naturwissenschaften (Dr.rer.nat.)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 796 605 405
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Mathematik
Betreut von / Supervisor:	Univ.-Prof. Dr. Arnold Neumaier

To my loves,

Somayeh, Fatemeh, and Narges

1 Abstract

There exist many optimization problems with applications in science, engineering, economics, industry, etc. According to whether the exact derivative is available or not, these problems can be classified into two classes: **black box optimization (BBO) problems** and **gradient-based optimization (GBO) problems**. It is very important to have algorithms that can be implemented with very little storage for both BBO and GBO problems in high dimensions. Accordingly, this thesis focuses on the design and test of a number of solvers using **subspace techniques** to handle BBO and GBO problems in high dimensions.

To compare the efficiency and robustness of our solvers with other state of the art (GBO) and (BBO) solvers, a new **test environment**, called **TestEnvironment**, is constructed with an automatic algorithm evaluation, significantly saving user time, performing statistics, and resulting in a summarized result as both pdf-file and tex-file including tables and figures. **TestEnvironment** uses two collections of unconstrained and bound constrained test problems by Gould et al. [80] and unconstrained least squares problems by Lukšan et al. [127].

The new solvers are called **LMBOPT**, **VSBBO**, **STBBO**, **VSBBON**, and **LMLS** and are available at the following URLs:

`http://www.mat.univie.ac.at/~neum/software/LMBOPT`,
`http://www.mat.univie.ac.at/~neum/software/VSBBO`,
`https://www.mat.univie.ac.at/~neum/software/STBBO`,
`https://www.mat.univie.ac.at/~kimiaei/software/VSBBON`,
`https://www.mat.univie.ac.at/~neum/software/LMLS`.

LMBOPT solves bound constrained GBO problems. It is an improved version of the active set method combined with the curved line search method by Neumaier & Azmi [136]. It is enriched by a new limited memory method and many practical enhancements that make **LMBOPT** very competitive in comparison with other state of the art local GBO solvers.

An active set trust region method for bound constrained optimization problems with the exact function value and gradient is discussed. It replaces the traditional trust region ratio with a variant of the sufficient descent condition – useful in finite precision

arithmetic and in strongly nonconvex regions. The reduced gradient is used as a critical measure to get a point which is never a spurious apparent local minimizer arisen because of cancellation in the calculation of a critical measure in double-precision arithmetic. The trust region radius is updated according to the reduced gradient. Under the positive semi-definiteness of approximated Hessian matrices restricted to the subspace of free variables, unlimited zigzagging could not occur. Hence all strongly active variables are found and fixed at finitely many iterations.

VSBBO solves unconstrained BBO problems. It is a randomized line search method with random directions. Complexity results with a given probability arbitrarily close to one are proven for all cases (nonconvex, convex, and strongly convex). The orders of our bounds are in all cases the same as those found by Bergou et al. [16], only valid in expectation while our bound is only, in the nonconvex case, the same as the one found by Gratton et al. [84]. The implemented version of **VSBBO** is enriched by many heuristic enhancements that do not affect the order of complexity results and turn **VSBBO** into a competitive solver in comparison with other state of the art local and global BBO solvers.

STBBO is a deterministic subspace method for unconstrained BBO problems. In the presence of inaccurate function values and gradients, the complexity bound for **STBBO** is found – independent of the choice of search directions enhancing the angle test – matching the order of bound found by Berahas et al. [15]. A significant decrease in the function value may be found when a successful prediction on a decrease in the model function and the model gradient norm along an efficient limited memory direction is achieved. Numerical results show that **STBBO** is competitive in comparison with known model-based solvers, **FMINUNC** using the traditional limited memory BFGS technique and standard BFGS technique.

VSBBON solves noisy unconstrained BBO problems. It is a randomized model-based line search method. In the presence of small noise level complexity results with a given probability arbitrarily close to one are proven in the nonconvex, convex, and strongly convex cases. Two effective techniques that make **VSBBON** competitive are to construct quadratic models in adaptively determined subspaces to handle problems in low and high dimensions and to find, update, and restart step sizes in a randomized line search algorithm. Numerical results show that the implemented version of **VSBBON** works well with any kind of noise which is not too large but in theory uniform random noise is matched on the assumption that the uncertainty of the function values is globally bounded by a small threshold.

LMLS solves nonlinear BBO least squares problems. It is a trust region method, but the complexity result is not investigated. **LMLS** is enriched by many enhancements: a non-monotone technique and adaptive radius strategy (useful in presence of narrow valley), a Broyden-like algorithm (useful when trust region is unsuccessful), a randomized finite difference approximation in an adaptive subspace for the Jacobian matrix estimation, either a Gauss-Newton or an improved dogleg method to solve the trust region

subproblem. Because of using these enhancements, **LMLS** is competitive for problems in low up to high dimensions in comparison with the traditional limited memory BFGS method and even standard BFGS method.

2 Zusammenfassung

Viele Anwendungen in Wissenschaft, Technik, Wirtschaft, Industrie usw. führen auf Optimierungsprobleme. Je nachdem, ob die genaue Ableitung verfügbar ist oder nicht, spricht man von gradientenbasierter Optimierung (GBO) oder Black-Box-Optimierung (BBO). Die vorliegende Arbeit behandelt das Design und den Test von Software, um BBO- und GBO-Probleme für Optimierungsprobleme, in denen höchstens die Varablen beschränkt sind, in hohen Dimensionen zu lösen. Unterraumtechniken ermöglichen Algorithmen, die trotz vieler Variablen mit sehr wenig Speicherplatz auskommen.

Um die Effizienz und Robustheit der Löser zu erhöhen, wird eine Benutzerzeit sparende Testumgebung namens **TestEnvironment** vorgestellt, die automatisch Algorithmen bewertet, dazu Tabellen, Abbildungen und Statistiken erstellt und diese zu einer Latex-Datei verarbeitet, die zu einer übersichtlichen Dokumentation in pdf führt. TestEnvironment verwendet eine Sammlung von Optimierungsproblemen von Gould et al. [80] und eine von kleinste-Quadrate-Problemen von Lukšan et al. [127].

Die neuen Löser heißen **LMBOPT**, **VSBBO**, **STBBO**, **VSBBON** und **LMLS** und sind unter den folgenden URLs verfügbar:

`http://www.mat.univie.ac.at/~neum/software/LMBOPT`,
`http://www.mat.univie.ac.at/~neum/software/VSBBO`,
`https://www.mat.univie.ac.at/~neum/software/STBBO`,
`https://www.mat.univie.ac.at/~kimiaei/software/VSBBON`,
`https://www.mat.univie.ac.at/~neum/software/LMLS`.

LMBOPT löst GBO-Probleme, bei denen die Variablen beschränkt sein können und beruht eine aktive-Mengen-Methode mit der Liniensuche von Neumaier & Azmi [136]. Eine neue Unterraummethode und viele praktische Verbesserungen machen **LMBOPT** im Vergleich zu den bekannten lokalen GBO-Lösern sehr wettbewerbsfähig.

VSBBO löst BBO-Probleme ohne Bedingungen an die Variablen. Es sucht entlang zufälligen Richtungen mit einer zufälligen Liniensuchmethode. Für nicht konvexe, konvexe und stark konvexe Zielfunktionen werden Komplexitätsergebnisse mit einer Wahrscheinlichkeit, die beliebig nahe bei eins liegt, im Einklang mit Ergebnissen von Gratton et al. [84] (nur für den nicht konvexen Fall) und denen von Bergou et al. [16] (die nur im Mittel gültige Ergebnisse erhalten). Viele heuristische Verbesserungen machen **VSBBO** im Vergleich zu den bekannten lokalen und globalen BBO-Lösern zu einem wettbewerbsfähigen Löser, ohne die Komplexitätsergebnisse zu beeinflussen.

STBBO ist ein deterministischer Löser für BBO-Probleme ohne Bedingungen an die Variablen, bei denen die Gradienten durch finite Differenzen geschätzt werden, mit derselben Komplexität wie bei Berahas et al. [15]. Es beruht auf einer neuen, deterministischen Unterraummethode. Die numerische Ergebnisse zeigen, dass **STBBO** mit bekannten modellbasierten Lösern, die die traditionelle BFGS-Technik oder LBFGS-Technik verwenden, sehr wettbewerbsfähig ist.

VSBBON löst ebenfalls BBO-Probleme ohne Bedingungen an die Variablen, bei denen die Funktionswerte verrauscht sind. Es ist eine randomisierte modellbasierte Liniensuchmethode, für die im Fall von uniform beschränktem Rauschen Komplexitätsergebnisse mit einer Wahrscheinlichkeit, die beliebig nahe bei eins liegt, bewiesen werden. Um **VSBBON** in niedrigen und hohen Dimensionen wettbewerbsfähig zu machen, werden quadratische Modelle in adaptiv bestimmten Teilräumen erstellt und einer randomisierten Liniensuche verwendet. Numerische Ergebnisse zeigen, dass **VSBBON** mit jeder Art von Rauschen gut funktioniert, das nicht zu großist.

LMLS ist ein gradientenfreies Vertrauensbereichs-Verfahren für nichtlineare kleinste-Quadrate-Probleme. Eine nichtmonotone Technik und eine adaptive Radiusstrategie (nützlich bei Vorhandensein eines engen Tals), ein Broyden-ähnlicher Algorithmus (nützlich, wenn die Vertrauensregion zu gross war), sowie eine randomisierte Finite-Differenzen-Näherung für die Jacobimatrix in einem adaptiven Unterraum machen **LMLS** für Probleme mit niedrigen bis hohen Dimensionen sehr wettbewerbsfähig mit herkömmlichen LBFGS-Methode und sogar zur Standard-BFGS-Methode. Die Komplexität wurde hier nicht untersucht.

3 Acknowledgement

There are many people at the University of Vienna whose supporting me made me deeply indebted.

Of all my wishes, the best goes to my supervisor – Prof. Arnold Neumaier – for his invaluable insight, encouragement, and continuous guidance to complete this research. Indeed, it was his mentoring this dissertation which would make it possible to pave the way. I feel very fortunate to have worked with him all these years long as a knowledgeable teacher and an outstanding mathematician.

I would like to appreciate the Doctoral Program Vienna Graduate School on Computational Optimization (VGSCO) for the financial support funded by the Austrian Science Foundation under Project No W1260-N3 in these four years and to thank the guest professors of our group Prof. Christian Blum, Prof. Claudia Sagastizabal, Prof. Luis Vicente, Prof. Alexander Shapiro, Prof. Arkadi Nemirovski, Prof. Daniel Kuhn, Prof. Heinz Bauschke, Prof. Petra Mutzel for their enthusiasm, energy, and great teaching during their intensive courses in the University of Vienna. In particular, I am grateful of Prof. Radu Ioan Bot and Prof. Georg Pflug for their support. It is also my pleasure to thank Prof. Stefano Lucidi for his support for six months of my research in the university of La Sapienza.

I feel very grateful to my friends Masoud Ahookhosh, Marefat Mansouri, Arash Ghaani Farashahi, Behzad Azmi, Mani Mesbah, Dang Khoa Nguyen, Marko Djukanovic, and Axel Böhem who have supported me throughout these four years in Vienna, having made my time very enjoyable.

I would like to appreciate my friend Mehdi Damaliamiri for proofreading the thesis.

The way I have paved so far has been hardy, and it was my lovely parents Ali and Chamar, my wife Somayeh, my daughters Fatemeh and Narges, my sisters Mahnaz, Maryam, Masoumeh, Mojdeh, and Mahtab, and my brothers Ruhollah, Mojtaba, and Masoud who made a great contribution to my success, and it is up to me to appreciate their pure and unconditional love, support, and inspiration during my life.

Contents

Part I

Bound-constrained gradient-based optimization

4 Introduction

4.1 Bound constrained optimization

Optimization is the study of mathematical problems with the goal of minimizing or maximizing a function, possibly subject to some constraints. Optimization problems include variables and their domains (only real values), an objective function depending on the variables and constraints as equations or/and inequalities to be minimized or maximized. There are applications in science, engineering, economics, industry, etc.

Feasible points are points satisfying all constraints. A **local solution** is a point with the lowest function value among nearby feasible points and a **global solution** is a point with the lowest function value among all feasible points in the domains. **Local methods** find the local solutions and **global methods** find the global solutions. **Optimality conditions** allow one to recognize whether a point is a solution or not.

Optimization problems are conventionally classified into different classes, depending on the form of the objective function and of the constraints. In terms of the kind of the constraints, they are classified as follows:
• **Unconstrained:** There is no constraint.
• **Bound constrained:** Variables are bounded.
• **Linearly constrained:** All constraints are linear equations or inequalities.
• **Nonlinearly constrained:** Some constraint is a nonlinear equation or inequality.

In this thesis, we only consider the problem where the objective function is **smooth** (continuously differentiable) and the only type of constraints are **bound constraint**. Thus we focus on solving bound constrained optimization problems (**BCOPT**) of the form

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in \mathbf{x}. \end{aligned} \tag{4.1}$$

Here, s.t. is short for "subject to" and the bounds on the variables are described by the **bounded** or **unbounded** box

$$\mathbf{x} := [\underline{x}, \overline{x}] := \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\},$$

where $-\infty \leq \underline{x}_i \leq \overline{x}_i \leq \infty$ for $i = 1, \cdots, n$. The objective function $f : \mathbf{x} \subseteq \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable with the **gradient**

$$g(x) := f'(x)^T \in \mathbb{R}^n$$

and the **Hessian**

$$B(x) := f''(x) \in \mathbb{R}^{n\times n},$$

which may be either **exact** or **inexact**. A special case is the unconstrained problem

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbb{R}^n, \end{array} \tag{4.2}$$

where the box is all of $\mathbb{R}^n$. A popular case of this problem is the unconstrained nonlinear least squares problem

$$\begin{array}{ll} \min & f(x) := \frac{1}{2}\|E(x)\|_2^2 \\ \text{s.t.} & x \in \mathbb{R}^n \end{array} \tag{4.3}$$

with continuously differentiable $E : \mathbb{R}^n \to \mathbb{R}^r$ $(r \geq n)$.

Optimization methods to be used for solving (4.1) can be classified according to whether the derivative information is available or not:
• **First-derivative methods** use exact function values and gradients.
• **Second-derivative methods** use exact function values, gradients and Hessians.
• **Derivative-free** or **black box optimization methods**, only use the exact function values, though they usually assume differentiability. Neither known gradients, known Lipschitz constants, nor known structural information about the objective function is assumed to be available. This problem (see, e.g., [42, 151]) is usually called black box optimization (BBO) or derivative-free optimization (DFO).
• **Noisy BBO** only uses inaccurate function values $\widetilde{f}(x) \approx f(x)$. The difference to the exact function value is called **noise**. Noise may be caused by either modelling, truncation, discretization errors, inaccurate measurements, or rounding errors. In some cases one knows the statistical properties of the noise; in other cases this is unknown.

Two efficient globalization methods are **line search methods** and **trust region methods**:
• Line search methods seek in each iteration for a point with better function value along a descent direction by generating a sequence of step sizes. Two inexact line search methods are the Wolfe conditions (Wolfe [162]) or Goldstein conditions (Goldstein [78]);
• Trust region methods construct a linear/quadratic model restricted to a region centered at the current point – called the **trust region** – to approximate the objective function, resulting in the **trust region step** which enforces a sufficient model function decrease. The achieved reduction in the function value over the predicate reduction in the model function is measured by a **trust region ratio**. If this ratio is sufficiently positive, a trial point with better function value is accepted as a new point and the trust region either remains fixed or is expanded. Otherwise the trust region is reduced until such a point is found; cf. Conn et al. [41].

Active set methods can be combined with either line search methods or trust region methods to be effective. They find the set of active bound constraints in each iteration and solve approximately an unconstrained subproblem, resulting in a local minimizer.

In exact precision arithmetic when an iterative bound constrained optimization algorithm finds a point whose reduced gradient vanishes it stops at a local minimizer. But in finite precision arithmetic such an algorithm may get stuck in nearly flat regions. Hence a theoretical criterion needs to be used to distinguish approximated local minimizers from spurious apparent local minimizers. For an iterative bound constrained optimization method a **complexity bound** is an upper bound on the number of iterations such that a point is found whose reduced gradient norm is below a given threshold and whose function value is as possible as smaller than the initial function value. In the unconstrained case, the reduced gradient is the same as the gradient. For an iterative BBO method, such a complexity bound is an upper bound on the number of function evaluations. The complexity bound of randomized BBO methods are valid either in expectation or a given probability arbitrarily close to 1. To get a complexity bound in the noisy BBO one can assume that the noise is bounded by

$$|\widetilde{f}(x) - f(x)| \leq \omega, \tag{4.4}$$

where ω is a fixed threshold unknown to the algorithm but it appears in the complexity bound. In practice, in the noisy unconstrained BBO, one can find an upper bound on the number of function evaluations and a point whose exact gradient norm is below ω.

4.2 Thesis overview

The first achievement of this thesis is a new test environment, called **TestEnviroment**, to compare selected solvers on a collection of test problems, save the necessary information for statistic, and perform statistic resulting in a summarized result as both pdf-file and tex-file including tables and figures. This is a joint work with Arnold Neumaier (cf. Kimiaei & Neumaier [116]), discussed in Section 5.

The reminding achievements of this thesis are the design and test of a number of solvers. These solvers use **subspace techniques** to handle problems in high dimensions and new **heuristic enhancements** turning them into the competitive solvers in comparison with other state of the art solvers.

• A gradient-based solver for the bound constrained problem (4.1) with the exact function value and gradient, called **LMBOPT**. **LMBOPT** is an improved version of active set method combined with a curved line search method by Neumaier & Azmi [136]. Its global convergence has been discussed in [136]. **LMBOPT** uses a new limited memory method and several practical enhancements which turns it into a competitive solver in comparison with other state of the art solvers. Using **TestEnviroment**, we show that **LMBOPT** is competitive for problems in low and large dimensions. This is a joint work with Arnold Neumaier and Behzad Azmi (cf. Kimiaei, Neumaier, & Azmi [118]), discussed in Section 6. **LMBOPT** is available at

`http://www.mat.univie.ac.at/~neum/software/LMBOPT`.

• An active set trust region method is introduced for solving the bound constrained problem (4.1) with the exact gradient. This method replaces the traditional trust region ratio by a variant of the sufficient descent condition – useful in finite precision arithmetic and in strongly nonconvex regions. It uses the reduced gradient as a critical measure to get a point which is never a spurious apparent local minimizer arisen because of cancellation in the calculation of a critical measure in double-precision arithmetic. The trust region radius is updated according to the reduced gradient. Under the positive semi-definiteness of approximated Hessian matrix restricted to the subspace of free variables, unlimited zigzagging could not occur. Hence all strongly active variables are found and fixed at finitely many iterations. This is my own work (cf. Kimiaei [111]), discussed in Section 7.

• A randomized gradient-free solver for the unconstrained optimization problem (4.2) with the exact function value and inexact gradient, called **VSBBO**. Our complexity results with a given probability arbitrarily close to 1 are proven in the nonconvex, convex, and strongly convex cases whose orders match the one found by Bergou et al. [16], only valid in expectation, and whose factors and orders match, only for the nonconvex case, the one found by Gratton et al. [84]. The implemented version of **VSBBO** is a randomized line search-based method enriched by heuristic enhancements that do not affect the order of the complexity results and turn **VSBBO** into an efficient global solver, although our theory guarantees only local minimizers. It even finds in most cases either a global minimizer or, where this could not be checked, at least a point of similar quality with the best competitive global solvers. Using **TestEnviroment**, we show that **VSBBO** is efficient and robust for medium and large scale problems in comparison with other state of the art local and global solvers. This is a joint work with Arnold Neumaier (cf. Kimiaei & Neumaier [117]), discussed in Section 8. **VSBBO** is obtainable from

`http://www.mat.univie.ac.at/~neum/software/VSBBO`.

• A deterministic gradient-free solver for the unconstrained optimization problem (4.2) with the exact function value and inexact gradient, called **STBBO**. In the presence of the inaccurate function value and gradient complexity result is proven in the general case – independent of the choice of search directions enhancing the angle test – whose order is the same as the one found by Berahas et al. [15]. An improved version of **STBBO** uses an efficient limited memory direction along which a successful prediction on a decrease in the model function and the model gradient norm might be achieved, resulting in a significant decrease in the function value. Using **TestEnviroment**, we show that **STBBO** is competitive for problems in low and high dimensions in comparison with other state of the art solvers. This is a joint work with Arnold Neumaier and Parvaneh Faramarzi (cf. Kimiaei, Neumaier, & Faramarzi [119]), discussed in Section 9. **STBBO** is available at

`https://www.mat.univie.ac.at/~neum/software/STBBO`.

• A noisy randomized model-based gradient-free solver for the unconstrained problem (4.2) with the inexact function value and gradient, called **VSBBON**. Complexity results

with a given probability arbitrarily close to 1 are proven in the presence of noise in the nonconvex, convex, and strongly convex cases. An improved version of **VSBBON** constructs quadratic models in adaptively determined subspaces for small and large scale problems and finds, updates, and restarts step sizes in a randomized line search algorithm so that its efficiency is increased. Using **TestEnviroment**, we show that **VSBBON** works well with any kind of noise which is not too large but in theory it works on uniform random noise satisfying (4.4). This is my own work (cf. Kimiaei [114]), discussed in Section 10. **VSBBON** is available at

`https://www.mat.univie.ac.at/~kimiaei/software/VSBBON`.

• A randomized gradient-free solver for the least squares problem (4.3) with the exact function value and inexact gradient, called **LMLS**. It does not include the global convergence or the complexity result. **LMLS** is the trust region algorithm with a nonmonotone technique and adaptive radius strategy (useful in presence of narrow valley), a Broyden-like algorithm (useful in the cases where trust region radius is small and iteration is unsuccessful), a randomized finite difference approximation in an adaptive subspace for the Jacobian matrix estimation, and either a Gauss-Newton or an improved dogleg method to solve the trust region subproblem. Using **TestEnviroment**, we show that **LMLS** is competitive for problems in low up to high dimensions in comparison with traditional limited memory BFGS method and even standard BFGS method. This is a joint work with Arnold Neumaier (cf. Kimiaei & Neumaier [115]), discussed in Section 11. **LMLS** is available at

`https://www.mat.univie.ac.at/~neum/software/LMLS`.

4.3 Basic notation

This section gives generalities for the problems (4.1)–(4.3).

4.3.1 Optimality condition for bound constrained optimization

We recall some notation from Neumaier & Azmi [136].

Let $x \in \mathbf{x}$ be the feasible point and i be the index. Then
(i) a bound $\underline{x}_i$ or $\overline{x}_i$ is called **active** if $x_i = \underline{x}_i$ or $x_i = \overline{x}_i$, respectively. In both cases, the index i and the component x_i are called **active** as well.
(ii) if $x_i \in]\underline{x}_i, \overline{x}_i[$, the index i, the component x_i, and the bounds $\underline{x}_i$ and $\overline{x}_i$ are called **nonactive** or **free**.
(iii) a point all of whose components are active is called **corner** of the box $\mathbf{x}$.

If the condition

$$g(\widehat{x})^T(x - \widehat{x}) \geq 0, \quad \text{for } x \in \mathbf{x} \tag{4.5}$$

holds, $\widehat{x} \in \mathbf{x}$ is a local solution of (4.1).

The **first order necessary optimality conditions** for the bound constrained problem (4.1) is

$$\begin{cases} g_i(\widehat{x}) \geq 0, & \text{if } \widehat{x}_i = \underline{x}_i, \\ g_i(\widehat{x}) \leq 0, & \text{if } \widehat{x}_i = \overline{x}_i, \\ g_i(\widehat{x}) = 0, & \text{if } \underline{x}_i < \widehat{x}_i < \overline{x}_i. \end{cases} \tag{4.6}$$

Here $\widehat{x}$ is called a **stationary point** for (4.1). Equivalently, at any local minimizer x of (4.1), the **reduced gradient** $g^{\text{red}}(x)$ at x, with components

$$g_i^{\text{red}}(x) := \begin{cases} 0 & \text{if } x_i = \underline{x}_i = \overline{x}_i, \\ \min(0, g_i) & \text{if } x_i = \underline{x}_i < \overline{x}_i, \\ \max(0, g_i) & \text{if } x_i = \overline{x}_i > \underline{x}_i, \\ g_i & \text{otherwise,} \end{cases} \tag{4.7}$$

vanishes, where $g_i := g_i(x)$ is the ith component of gradient vector at x. The **first order sufficient optimality conditions** for the bound constrained problem (4.1) is that every corner x of $\mathbf{x}$ such that $g_i(x) > 0$ at all active lower bounds and $g_i(x) < 0$ at all active upper bounds is a local minimizer of (4.1). If the inequalities in (4.6) hold strictly, the **strict complementarity** is said to hold at $\widehat{x}$. If at least one component of $\widehat{x}$ violates this strict complementarity, the stationary point $\widehat{x}$ is called **degenerate**; otherwise, it is called **nondegenerate**. Let f be twice continuously differentiable in a neighborhood of $\widehat{x}$. Then $\widehat{x}$ is called a **strong local minimizer** of f if the gradient of f at $\widehat{x}$ vanishes and the Hessian matrix of f at $\widehat{x}$ is positive definite.

4.3.2 Convex set and functions

The continuous optimization problems can also be classified into two different cases: **convex** or **nonconvex**. This classification includes both the objective function and constraint set. A local solution of the bound constrained optimization (4.1) is

• the global solution of (4.1) if f is convex;

• a unique solution of (4.1) if f is strictly convex.

We need the following statements:

• A set $C \subseteq \mathbb{R}^n$ is convex if the line segment $\overline{xy} := \{(1-t)x + ty \mid 0 \leq t \leq 1\}$ contained in C for all $x, y \in C$.

• A convex combination of $x_1, \cdots, x_m$ is a linear combination

$$x := \sum_{i=1}^{m} w_i x_i \text{ with } w_i \geq 0 \text{ and } \sum_{i=1}^{m} w_i = 1.$$

• An affine combination of $x_1, \cdots, x_m$ is a linear combination

$$x := \sum_{i=1}^{m} w_i x_i \text{ with } \sum_{i=1}^{m} w_i = 1.$$

• Let $f : C \to \mathbb{R}^n$ be a function on C. Then the function f convex if

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y), \quad \text{for } x, y \in C,\ t \in [0,1] \tag{4.8}$$

and is σ-strongly convex with $\sigma \geq 0$ if

$$f((1-t)x+ty)+t(1-t)\frac{\sigma}{2}\|x-y\|^2 \leq (1-t)f(x)+tf(y), \quad \text{for } x, y \in C,\ t \in [0,1]. \tag{4.9}$$

The condition (4.8) is the special case of the condition (4.9) when $\sigma = 0$.

4.4 The state of the art

This section discusses the state of the art solvers for solving the problems (4.1)–(4.3).

4.5 Gradient-based optimization methods

In the last few years, many algorithms have been developed for solving the **BCOPT** problem (4.1).

Active set methods are among the most effective methods. They consist of two main stages that alternate until a solution is found. In the first stage one identifies a good approximation for the set of optimal active bound constraints, defining a face likely to contain a stationary point of the problem. A second stage then explores this face of the feasible region by approximately solving an unconstrained subproblem.

A major reason of inefficiency of optimization methods is said **zigzagging**, arisen possibly through
• inefficient search directions, like steepest descent direction in the unconstrained optimization problems;
• changing active set strategies in the bound constrained optimization problems.
When a sequence of iteratively generated points of an optimization method moves along a valley (maybe either long, steep, or curved) to become close to a local minimizer, the effects of zigzagging are increased extremely and the desired accuracy may not be reached. Hence, to eliminate the effects of zigzagging one needs to take measures which theoretically and numerically show that all strongly active variables are found and fixed finitely, resulting in the fact that zigzagging cannot occur infinitely. Although the choice of efficient directions can remove zigzagging in the unconstrained case, their adaptation restricted to the subspace of non-active variables may not remove zigzagging in the bound-constrained case if a poor active set strategy is used (see e.g., the second and third examples in [136, Section 6]).

A classical reference on active set methods for bound constrained problems with a convex quadratic objective function (**QBOPT**) is the projected conjugate gradient method of

Polyak [145], which drops and adds only one constraint in each iteration. That is, at each step of this active set method, the dimension of the subspace of active variables is changed only by one. This fact implies that if there are n_1 constraints active at the starting point x^0 and n_2 constraints active on the solution of **QBOPT**, we need at least $|n_2 - n_1|$ iterations to reach the solution of **QBOPT**. This may be a serious drawback in the case of large scale problems. Dembo & Tulowitzki [54] introduced in 1983 methods for **QBOPT** that are able to add and drop many constrains at each iteration. Their basic idea was further developed by Yang & Tolle [164] into an algorithm guaranteed to identify in finitely many iterations the face containing a local solution of the **QBOPT**, even when the solution of the problem is degenerate. For further research on the **QBOPT** we refer the reader to [62, 63, 129, 130].

For **BCOPT** with a general nonlinear objective function, Bertsekas [17] proposed an active set algorithm that uses a **gradient projection method** to find the optimal active variables. He showed that this method is able to find very quickly the face containing a local solution. Further research on convergence and properties of projected gradient methods can be found in [17, 31, 64]. The idea of using gradient projections for identifying optimal active constraints was followed up by many researchers. Many of them [29, 40, 39] combined Newton type methods with the gradient projection method in order to accelerate the convergence. For example, **LBFGSB**, developed by Byrd, Lu, Zhu & Nocedal [29], performs the gradient projection method by computing the Cauchy point to determine the active variables. After the set of active variables is determined, the algorithm performs line searches along the search directions obtained by a **limited memory BFGS method** [30] to explore the subspace of nonactive variables. In fact, the use of limited memory BFGS matrices and the line search strategy are the main properties that distinguish this method from others, especially from the trust region type method proposed by Conn et al. [40, 39].

A **non-monotone line search** was first introduced for Newton methods by Grippo, Lampariello & Lucidi (**GLL**) in [86], in order to improve the ability to follow a curved valley with steep walls. Later several papers [49, 52, 75, 87, 157, 167] on non-monotone line search methods pointed out that in many cases these methods are more efficient than monotone line search methods. Other papers [18, 23, 50, 51, 77, 125, 154] indicate that gradient projection approaches based on a **Barzilai–Borwein step size** [10] have impressive performance in a wide range of applications. Recent work [20, 21, 23, 24, 25, 150] on Barzilai–Borwein gradient projection methods (**BBGP**) modified these by incorporating them with the **GLL** non-monotone line search: For instance, Raydan [150] developed the **BBGP** method for solving unconstrained optimization problems, and Dai & Fletcher [50, 51] proposed **BBGP** methods for large-scale bound constrained quadratic programming. The idea of Raydan [150] was developed to generate a convex constrained solver (**SPG**) by Birgin et al. [23, 24] and a bound constrained solver (**GENCAN**) by Birgin et al. [20, 21], enriched by an active set strategy.

The **GALAHAD** [79] package uses as bound-constrained solver **LANCELOT-B**, a

trust-region algorithm using truncated Newton directions. Recently, Burdakov et al. [28] constructed a family of limited memory quasi Newton methods for unconstrained optimization combined with either line search method or trust region one, called the **LMBFG** package.

To deal with negative curvature regions, Birgin & Martínez [20] used the second-order trust region algorithm of Zhang & Xu [168], and Birgin & Martínez [21] designed a new algorithm whose line search iteration is performed by means of backtracking and extrapolation. Hager & Zhang [92] developed an active set algorithm called **ASACG** for large scale bound constrained problems. **ASACG** consists of two main steps within a framework for branching between these two steps: a non-monotone gradient projection step, called **NGPA**, which is based on their research on cyclic Barzilai–Borwein method [52], and an unconstrained step that utilizes their developed conjugate gradient algorithms [89, 90, 91, 93]. **ASACG** version 3.0 has been updated by calling **CGdescent** version 6.0 which uses the variable `HardConstraint` to evaluate the function or gradient at a point that violates the bound constraints, so that it could improve performance by giving the code additional flexibility in the starting step size routine. In 2017, Cristofari et al. [43] proposed a two-stage active set algorithm for bound-constrained optimization, called **ASABCP**. **ASABCP** first finds an active set estimation with a guarantee that the function value is reduced. Then it uses a truncated-Newton technique in the subspace of the non-active variables.

A considerable amount of literature has been published on line search algorithms, most of which satisfy the Wolfe conditions (Wolfe [162]) or Goldstein conditions (Goldstein [78]). A problem of line search algorithms satisfying the Wolfe conditions is the need to **calculate a gradient at every trial point**. On the other hand, line search algorithms based on the Goldstein conditions are gradient-free, but they have **very poor behaviour in strongly nonconvex regions**. Neumaier & Azmi [136] introduced a new active set method **BOPT** (Algorithm 9.1 in [136] = Algorithm 4.5.1) using an efficient gradient-free curved line search **CLS** (Algorithm 3.3 in [136] = Algorithm 4.5.3). The active set strategy used in **BOPT** always enforces that the gradient reduction in the components restricted by non-active variables over the reduced gradient reduction is at least asymptotically bounded. This property of the active set can remove zigzagging, a possible source of inefficiency. On the other hand, **CLS** has good properties in theory, achieving a sensible decrease in the objective function.

Section 6 introduces an efficient version of **BOPT**, called **LMBOPT**, for bound constrained optimization problems with a continuously differentiable objective function. **LMBOPT** preserves the main structure of **BOPT** – the active set strategy and **CLS**. To get rid of getting stuck in nearly flat regions, **LMBOPT** uses **safeguards in finite precision arithmetic**, resulting in **an improved version of CLS** and a **regularized Krylov direction**. In addition, many other practical enhancements are used, one of which is a **new limited memory method**. A **solver choice** based on our finding from an extensive numerical results is made. It depends on the problems dimension, the presence and absence of constraints, the desired robustness, and the relative cost of

function and gradient evaluations.

Section 7 combines the trust region method with the active set method by NEUMAIER & AZMI [136]. Because of cancellation in the calculation of known critical measures in double-precision arithmetic, spurious apparent local minimizers may be considered to be a local minimizer. Hence the **reduced gradient** is a **reasonable critical measure** to decide whether the agreement of the objective function and the model function is good or not, **useful in finite precision arithmetic**. To prove our complexity bound, the **trust region radius** is updated based on the **reduced gradient**. All strongly active variables are found and fixed at finitely many iterations under the positive semi-definiteness of approximated Hessian matrix restricted to the subspace of non-active variables so that unlimited zigzagging cannot occur.

4.5.1 BOPT – an active set method for bound constrained optimization

Recently, NEUMAIER & AZMI [136] gave a comprehensive convergence theory for a generic algorithm for the bound constrained optimization problem (4.1) with a continuously differentiable objective function, called **BOPT**. At any point x during the iteration, a search direction is determined in a subspace obtained by varying the part indexed by a working set I, chosen either as the minimal set

$$I_-(x) := \{i \mid \underline{x}_i < x_i < \overline{x}_i\} \tag{4.10}$$

of **free indices** or as the maximal set

$$\begin{array}{rcl} I_+(x) & := & I_-(x) \cup \{i \mid g_i^{\rm red} \neq 0\} \\ & = & I_-(x) \cup \{i \mid \underline{x}_i = x_i < \overline{x}_i, g_i < 0 \text{ or } \underline{x}_i < x_i = \overline{x}_i, g_i > 0\} \end{array} \tag{4.11}$$

of **free or freeable indices**. To ensure the absence of severe zigzagging, **freeing iterations** in which

$$I = I_+(x) \neq I_-(x), \tag{4.12}$$

are restricted to cases where the choice $I = I_-(x)$ violates the inequality

$$\|g_I\|_*^2 \geq \rho \|g^{\rm red}\|_*^2, \quad \text{for some } \rho \in]0,1]. \tag{4.13}$$

Here $\|\cdot\|_*$ is the **dual norm** of a monotone norm $\|\cdot\|$, defined by $\|g\|_* := \sup_{p \neq 0} g^T p/\|p\|$, and g_I stands for the restriction of g to the index set I. More generally, we denote by x_I the subvector of a vector x with indices taken from the set I, by $A_{:k}$ the kth column of a matrix A, and by A_{II} the submatrix of A with row and column indices taken from I.

BOPT takes a starting point $x^0 \in \mathbb{R}^n$ and the feasible set $\mathbf{x}$ as input and returns an optimal point $x^{\rm best}$ and its function value $f^{\rm best} = f(x^{\rm best})$ as output. It uses the tuning parameters of a line search **CLS** (discussed in Section 4.5.2) and two tuning parameters:
$0 < \Delta^{\rm a} < 1$ (parameter for the angle condition),
$0 < \rho \leq 1$ (parameter for freeing iterations).

4.5.1 Algorithm. (**BOPT**, **bound constrained optimization algorithm**)
(BOP_0) Computes the initial gradient $g^0 := g(x^0)$ and **reduced gradient** $g^{\mathrm{red}}(x^0)$ and finds the initial working set $I^0 := I_+(x^0)$.
for $\ell = 0, 1, 2, \cdots$ **do**
(BOP_1) Stop if $g^{\mathrm{red}}(x^\ell) = 0$, resulting in $x^{\mathrm{best}} = x^\ell$ and $f^{\mathrm{best}} = f(x^\ell)$.
(BOP_2) Compute a search direction p^ℓ satisfying

$$p_i^\ell = 0 \quad \text{for } i \notin I^\ell, \tag{4.14}$$

$$\frac{(g_I^\ell)^T p_I^\ell}{\|g_I^\ell\|_* \|p_I^\ell\|} \le -\Delta^{\mathrm{a}} < 0, \tag{4.15}$$

$$\text{if (4.12) holds, } g_i^\ell p_i^\ell \le 0 \quad \text{for all } i. \tag{4.16}$$

Perform a line search **CLS** (discussed in Section 4.5.2) along the **bent search path**

$$x(\alpha) := \pi[x^\ell + \alpha^\ell p^\ell], \tag{4.17}$$

where $\pi[x^\ell + \alpha^\ell p^\ell]$ is the projection of the ray $x^\ell + \alpha^\ell p^\ell$ into the feasible set $\mathbf{x}$ with components

$$\pi[x^\ell + \alpha^\ell p^\ell]_i := \sup(\underline{x}_i, \inf(x_i + \alpha^\ell p_i^\ell, \overline{x}_i)) = \begin{cases} \underline{x}_i & \text{if } x_i^\ell + \alpha^\ell p_i^\ell \le \underline{x}_i, \\ \overline{x}_i & \text{if } x_i^\ell + \alpha^\ell p_i^\ell \ge \overline{x}_i, \\ x_i^\ell + \alpha^\ell p_i^\ell & \text{otherwise.} \end{cases} \tag{4.18}$$

Set $x^{\ell+1} := x(\alpha^\ell)$ and compute the gradient $g^{\ell+1} := g(x^{\ell+1})$ and the reduced gradient $g^{\mathrm{red}}(x^{\ell+1})$.
(BOP_3) Find $I^{\ell+1} := I_-(x^{\ell+1})$ by (4.10). If (4.13) is violated, update the new working set $I^{\ell+1} := I_+(x^{\ell+1})$ by (4.11).
end for

The bent line search and conditions (4.13)–(4.16) on the search direction are essential for the convergence analysis in [136, Sections 8 and 11], where the following is proved:

4.5.2 Theorem. *Let f be continuously differentiable, with Lipschitz continuous gradient g. Let x^ℓ denote the value of x in Algorithm 4.5.1 after its ℓth update. Then one of the following three cases holds:*
(i) The iteration stops after finitely many steps at a stationary point.
(ii) We have

$$\lim_{\ell\to\infty} f(x^\ell) = \widehat{f} \in \mathbb{R}, \quad \inf_{\ell\ge 0} \|g^{\mathrm{red}}(x^\ell)\|_* = 0.$$

Some limit point $\widehat{x}$ of the x^ℓ satisfies $f(\widehat{x}) = \widehat{f} \le f(x^0)$ and $g^{\mathrm{red}}(\widehat{x}) = 0$.
(iii) $\sup_{\ell\ge 0} \|x^\ell\| = \infty$.

Moreover, if **BOPT** *converges to a nondegenerate stationary point, all strongly active variables are ultimately fixed, so that zigzagging through changes of the active set cannot occur infinitely often.*

4.5.2 CLS – the curved line search of BOPT

CLS (Algorithm 3.3 in [136] = Algorithm 4.5.3 below), the line search used in **BOPT**, is an efficient gradient-free curved line search algorithm along a piecewise linear search path defined by directions guarded against zigzagging. It takes the ℓth point x^ℓ and its function value $f^\ell = f(x^\ell)$, the gradient vector $g^\ell = g(x^\ell)$, the search direction p^ℓ, the feasible set $\mathbf{x}$, and the initial step size α^{init} as input and returns the $(\ell+1)$th point $x^{\ell+1} = x(\alpha^\ell)$, its step size α^ℓ, and its function value $f^{\ell+1} = f(x^{\ell+1})$ as output. It uses several tuning parameters:
$\beta \in]0, \frac{1}{4}[$ (parameter for efficiency),
$q > 1$ (extrapolation factor),
the positive integer $l^{\max}$ (limit on the number of iterations),
$\alpha^{\max}$ (maximal value for the step size α).

4.5.3 Algorithm. (**CLS, a curved line search algorithm**)

(CLS_0) The lower and upper bound of the admissible step size interval are initialized to $\underline{\alpha} := 0$ and $\overline{\alpha} := \infty$, respectively. The initial step size is chosen as $\alpha^0 := \alpha^{\text{init}}$.
for $k = 0, \cdots, l^{\max}$ **do**
(CLS_1) Compute the point $x(\alpha^k)$ on the bent search path (4.17), its function value $f(x(\alpha^k))$, and the **Goldstein quotient** (Goldstein [78])

$$\mu(\alpha^k) := \frac{f(x(\alpha^k)) - f(x^\ell)}{\alpha (g^\ell)^T p^\ell} \quad \text{for } \alpha^k > 0 \tag{4.19}$$

(CLS_2) If the **sufficient descent condition**

$$\mu(\alpha^k)|\mu(\alpha^k) - 1| \geq \beta \ \text{ with fixed } \beta > 0 \tag{4.20}$$

holds, set $\alpha^\ell = \alpha^k$, $x^{\ell+1} = x(\alpha^\ell)$, and $f^{\ell+1} = f(x^{\ell+1})$, **CLS** is efficient and stops.
(CLS_3) If $\mu(\alpha^k) \geq \frac{1}{2}$, the lower bound of the interval is updated by $\underline{\alpha} := \alpha^k$; otherwise, if $\alpha^k = \alpha^{\max}$, **CLS** stops; otherwise, the upper bound of the interval is updated by $\overline{\alpha} := \alpha^k$.
(CLS_4) If $\underline{\alpha} > 0$ and $\overline{\alpha} < \infty$, go to ($\text{CLS}_5$); otherwise, go to ($\text{CLS}_6$).
(CLS_5) If $\mu(\alpha^k) < 1$, take as step size $\alpha^{k+1} := 0.5\alpha^k/(1-\mu(\alpha^k))$ (a step of the secant method for solving $\mu(\alpha^k) = 0.5$). Otherwise, expand the step size to $\alpha^{k+1} := q\alpha^k$. Then go to ($\text{CLS}_1$).
(CLS_6) If $\overline{\alpha} = \infty$, expand the step size to $\alpha^{k+1} := q\alpha^k$; otherwise, if $\underline{\alpha} = 0$, reduce the step size to $\alpha^{k+1} := \alpha^k/q$; otherwise, take as step size the geometric mean $\alpha^{k+1} := \sqrt{\underline{\alpha}\overline{\alpha}}$.
end for

• Condition (4.20) is an improved form of the Goldstein condition

$$0 < \mu'' \leq \mu(\alpha) \leq \mu' < 1.$$

It forbids step sizes which are too long or too small by enforcing that $\mu(\alpha)$ is sufficiently positive and not too close to one.
• According to Theorem 3.2 of [136], a step size satisfying (4.20) can be found by performing **CLS** if the objective function f is bounded below.
• If the objective function is quadratic and an exact line search results in that the secant step size is a minimizer of a convex quadratic function along the search ray and so the quadratic case is started optimally. Otherwise the function is far from quadratic and bounded.

4.5.3 Trust region methods

In the classical trust-region method, a **trust region** restricted to the subspace of free variables is defined by

$$\mathcal{T}^r := \Big\{p^\ell \in \mathbb{R}^n \mid \|p_I^\ell\|_\infty \leq \Delta^\ell, \quad x^\ell + p^\ell \in \mathbf{x}, \quad p_i^\ell = 0 \; \forall \; i \notin I\Big\}.$$

Here $\Delta^\ell > 0$ is called the **trust region radius**. A trust region method computes approximately a direction vector p^ℓ as a minimizer of the quadratic model function $\mathcal{Q}$ in $\mathcal{T}^r$, i.e.,

$$\begin{array}{ll} \min & \mathcal{Q}(x^\ell + p) \\ \text{s.t.} & p \in \mathcal{T}^r. \end{array} \tag{4.21}$$

A measure of disagreement between the model function $\mathcal{Q}$ and the objective function f is

$$\mathcal{R}^\ell := \frac{f(x^\ell + p^\ell) - f(x^\ell)}{\mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell)}. \tag{4.22}$$

Here, the numerator and the denominator of (4.22) are called the **actual** and **predicted** reduction, respectively. If a good agreement between the objective function and the estimated model is found, Δ^ℓ is expanded and the ℓth iteration is called **successful**; otherwise, it is reduced and the ℓth iteration is called **unsuccessful**; for more details see [41].

4.6 Unconstrained black box optimization (BBO) methods

An unconstrained BBO method solves the unconstrained problem (4.2), assuming only the availability of an oracle that returns for a given $x \in \mathbb{R}^n$ the function value $f(x)$. Neither gradients, Lipschitz constants, nor structural information about f are assumed to be available, though for convergence and/or complexity analysis one needs to make further assumptions. We denote by $\widetilde{g} := \widetilde{g}(x)$ the estimated gradient at x and by $\widetilde{f} := \widetilde{f}(x)$ the inaccurate function value at x. A huge deal of literature exists about the problem, and we only mention a few pointers to the literature. A thorough survey

for derivative-free optimization methods was given by LARSON et al. [122]. Another useful paper suggested by RIOS & SAHINIDIS [151] discusses the practical behaviour of derivative-free optimization softwares. The techniques for solving BBO problems fall into two classes, **deterministic** and **randomized** methods. We mainly discuss the randomized case; for deterministic methods see, e.g., the book by CONN et al. [42] and its many references. Randomized methods for BBO going back by POLYAK [146] and VAN LAARHOVEN & AARTS [158] were later discussed especially in the framework of evolutionary optimization [8, 96, 156]. There are also randomized BBO optimization algorithms for deterministic problems, e.g., HOLLAND [100] and BANDEIRA et al. [9].

Previous BBO software of the optimization group at the university of Vienna includes the deterministic algorithms **GRID** [65, 66] and **MCS** [102] and the randomized algorithms **SnobFit** [103] and **VXQR** [137]. Software by many others is mentioned in Section 8.11.2, where an extensive numerical comparison is discussed in Subsections 8.10.1–8.10.4.

Section 8 discusses a randomized line search method, called **VSBBO**, for unconstrained black box optimization. The local complexity results for nonconvex, convex, strongly convex are proven with probability arbitrarily close to one. Many new enhancements are introduced and added to **VSBBO** which makes it competitive for problems in high dimension in comparison with the state of the art local and global solvers.

Recently, researchers have shown an increased interest in applying quasi Newton methods using the finite difference for the estimation of the gradient to solve unconstrained black box optimization problems, e.g., [14, 117]. The **FMINUNC** solver has an option to use the standard quasi Newton method to estimate the inverse of the Hessian matrix and the finite difference technique to approximate the gradient. Although it is very useful for small scale problems, considerable savings are made due to full inverse matrices. Instead of such a standard method, a usual suggestion is to use a limited memory quasi Newton method, requiring a low memory, cf. LIU & NOCEDAL [124].

To estimate the gradient, BERAHAS et al. [15] discuss three bounds on the number of function evaluations, step sizes, the exact gradient norm for all existed randomized and deterministic methods so that the reduction in the error of gradient norm is asymptotically bounded by the reduction in the exact gradient norm.

Line search methods proceed iteratively by producing a sequence of estimated or exact step sizes in the hope of finding a good decrease in the function value. Their global convergence has been proven by AL-BAALI & FLETCHER [4], FLETCHER [74], and NOCEDAL & WRIGHT [140]. BERAHAS et al. [15] discuss complexity bounds for nonconvex, convex, and strongly convex cases on the accuracy of two various gradient estimations. Other useful references on the first and second order complexity bounds are CURTIS et al. [45, 46] and GRATTON et al. [83] for gradient-based trust region algorithms and GRAPIGLIA et al. [81] for gradient-based nonlinear step size control algorithms. CARTIS et al. [35] introduced a non-monotone gradient-related algorithm for nonconvex smooth

unconstrained optimization problems which finds at most $\mathcal{O}(\varepsilon^{-2})$ function and gradient evaluations a point x with the 2-norm gradient is below ε in the worst case.

Berahas et al. [14] estimated noise in the same way as Hamming [95] did. This estimation was added to the Armijo line search method and used to estimate the gradient by the finite difference technique. But the effect of noise in BFGS updating was not investigated. Recently, Xie et al. [163] showed that the Wolfe conditions with the exact function value and gradient can be satisfied if these conditions with the inaccurate function and gradient are satisfied and the exact gradient is not small. They modified the BFGS method with the Wolfe line search without estimating noise and showed that this method converges to a neighborhood of the solution while the effects of noise are take into account in BFGS updating.

Section 9 discusses a new deterministic subspace method for unconstrained black box optimization. Kimiaei et al. [118] suggested a limited memory for bound constrained optimization using the exact gradient. It included a limited memory quasi Newton method, which has minimal storage requirements. Our goal is to improve this method for unconstrained black box optimization by a subspace technique such that it tries to decrease both the model gradient norm and function value in the current iteration. In this case, the gradient is estimated by the finite difference technique. To get the complexity bound, it is important that we can assess the amount of reduction in the function value achievable by the Wolfe line search method [4] as discussed later. We improve this line search by a heuristic way when it fails in the presence of noise. Although our algorithm stops when the estimated gradient norm is below a given threshold, it finds a point whose exact gradient norm is below a given threshold. In this case, the Wolfe conditions with both the inaccurate function and gradient are satisfied numerically. Under assumptions that the discretization and function evaluation errors appear in estimation of the gradient and that the exact gradient norm is not small, we show in the same way as Xie et al. [163] did that the Wolfe conditions with the exact function value and gradient are satisfied but with the difference that these conditions are independent of the choice of search directions. In this section, the effects of noise are taken into account in an estimated Hessian update. But in contrast to Xie et al. [163] we enforce that our nontraditional limited memory quasi Newton direction enforces the angle condition. Hence we don't need to care about whether the Wolfe conditions with both the inaccurate function and gradient can generate a quasi Newton direction whose angle with the gradient is bounded away from 90°. Ignoring some parts of Hessian information reduces the efficiency of the traditional limited memory technique. We try to numerically show that the new subspace method is more efficient than the standard BFGS quasi Newton method even for problems in low and medium dimensions and than the traditional limited memory BFGS quasi Newton method for low and high dimensions.

4.7 Unconstrained noisy BBO methods

An unconstrained noisy BBO method solves the unconstrained problem (4.2), assuming the availability of a noisy oracle that returns for a given $x \in \mathbb{R}^n$ an approximation $\widetilde{f}(x)$ to the exact function value $f(x)$, contaminated by noise. The algorithm uses neither knowledge of the gradient, Lipschitz constants of f, information about the structure of f, nor knowledge of the statistical properties of the noise. The noise may be deterministic (caused by either modelling, truncation, and/or discretization errors) or stochastic (caused by either inaccurate measurements or rounding errors).

Some competitive solvers for noisy unconstrained and bound constrained black box optimization problem are **SDBOX** by Lucidi & Sciandrone [126], **NMSMAX** by Higham [99], **DSPFD** by Gratton et al. [84], **BFO** by Porcelli & Toint [147], **MCS** by Huyer & Neumaier [102], **BCDFO** by Gratton et al. [85], **UOBYQA** by Powell [149], **FMINUNC** by the Matlab Optimization Toolbox for small, medium, and large scale problems.

Section 10 discusses a randomized model-based line search method, called **VSBBON**, for noisy unconstrained optimization. Complexity results are proven with probability arbitrarily close to one for nonconvex, convex, and strongly convex in the presence of noise. Two main ingredients of **VSBBON** are
• to construct quadratic models in adaptively determined subspaces which can handle low up to large scale problems,
• to find, update, and restart step sizes in a randomized line search algorithm.

4.8 Unconstrained black box least squares methods

An unconstrained black box least squares method solves the unconstrained nonlinear least squares problem (4.3) with high-dimensional $x \in \mathbb{R}^n$. We assume that no derivative information is available.

In recent years, there has been a huge amount of literature on least squares and its applications. Here we just list a useful book and paper:
• Ortega & Rheinboldt [142] introduced an excellent book, both covering algorithms and their analysis.
• An excellent paper, both covering Levenberg-Marquardt algorithms, quasi-Newton algorithms, and trust region algorithms and their local analysis without non-singularity assumption, has been introduced by Yuan [166].

Derivative free unconstrained nonlinear black box least squares solvers can be classified in two ways according to how the Jacobian matrix is estimated, and according to whether they are based on line search or on trust region:
• Quasi Newton approximation. **FMINUNC** using the standard quasi Newton approx-

imation is an efficient solver for medium scale problems. It will be useful for small and medium and large scale problems. SORBER et al. [155] introduced **MINLBFGS** (a limited memory BFGS algorithm) and **MINLBFGSDL** (a trust region algorithm using a dogleg algorithm and limited memory BFGS approximation).
• Finite difference approximation. There are many trust region methods using the finite difference method for the Jacobian matrix estimation such as **CoDoSol** and **STRSCNE** by BELLAVIA et al. [12, 13], **NMPNTR** by KIMIAEI [112], **NATRN** and **NATRLS** by AMINI et al. [5, 6], **LSQNONLIN** from the Matlab Toolbox, **NLSQERR** (an adaptive trust region strategy) by DEUFLHARD [57], and **DOGLEG** by NIELSEN [138]. They are suitable for small and medium scale problems. Line search methods using the finite difference approximation are **NLEQ** (a damped affine invariant Newton method) by NOWAK & WEIMANN [141] and **MINFNCG** (a family of nonlinear conjugate gradient methods) by SORBER et al. [155].

To solve the least squares problem (4.3), trust region methods use linear approximations of the residual vectors to make surrogate quadratic models whose accuracy are increased by restricting their feasible points. These methods use a computational meseaure to identify whether an agreement between an actual reduction of the objective function and a predicate reduction of surrogate guadratic model function is good or not. If this agreement is good, the iteration is called successful and the trust region radius is expanded; otherwise, the iteration is called unsuccessful and the trust region radius is reduced, for more details see [41, 140].

The efficiency of trust region methods depends on how the trust region radius is updated (see, e.g., [3, 5, 6, 68, 67, 69, 71, 72, 73, 113, 165]) and whether non-monotone techniques are applied (see, e.g., [1, 2, 3, 5, 6, 55, 86, 88, 112, 113, 165]). Rounding errors may leads two problems:
(i) The model function may not decrease numerically for some iterations. In this case, if there is no decrease in the function value for such iterations, trust region radii are expanded possibly several times which is an unnecessary expansion for them,
(ii) The model function may decrease numerically but the objective function may not decrease in the cases where iterations are near a valley, deep with a small creek at the bottom and steep sides. In this case, trust region radii are reduced possibly many times, leading to the product of quite a small radius, or even a failure.

Non-monotone techniques can be used in the hope of overcoming the second problem.

Section 11 discusses a new limited memory method, called **LMLS**, for unconstrained black box least squares problem. **LMLS** is an improved trust region algorithm enriched by
• a new non-monotone technique and a new adaptive radius one, useful in the presence of narrow valley,
• a Broyden-like algorithm, useful in the cases where the function value cannot be improved,
• a randomized finite differences method to estimate the Jacobian matrix,

• either a Gauss-Newton method or an improved dogleg method in an adaptive subspace to solve the trust region subproblem.

5 Testing optimization methods

This section discusses a new test environment with automatic algorithm evaluation, saving significantly user time.

5.1 Test problems

• To compare gradient based solvers, we use all 1088 unconstrained and bound constrained problems with up to 100001 variables from the `CUTEst` collection of optimization problems by Gould et al. [80], in case of variable dimension problems for all allowed dimensions in this range.
• To compare black box solvers, all 568 unconstrained problems 1 up to 9000 variables from the `CUTEst` are used.
• To compare unconstrained black box least squares solvers, test problems from the collection by Lukšan et al. [127] are used.

5.2 Initial points

The initial point is $x^0 := 0$, but we shift the arguments by

$$\xi_i := (-1)^{i-1} \frac{2}{2+i}, \quad \text{for all } i = 1, \ldots, n \tag{5.1}$$

to avoid that a solver guesses the solution of toy problems with a simple solution (such as all zero or all one) – quite a number of these are in the `CUTEst` library. It means that the initial point is chosen by $x^0 := \xi$ and the initial function value is $f^0 := f(x^0)$ while the other function values are computed by $f^\ell := f(x^\ell + \xi)$ for all $\ell \geq 0$.

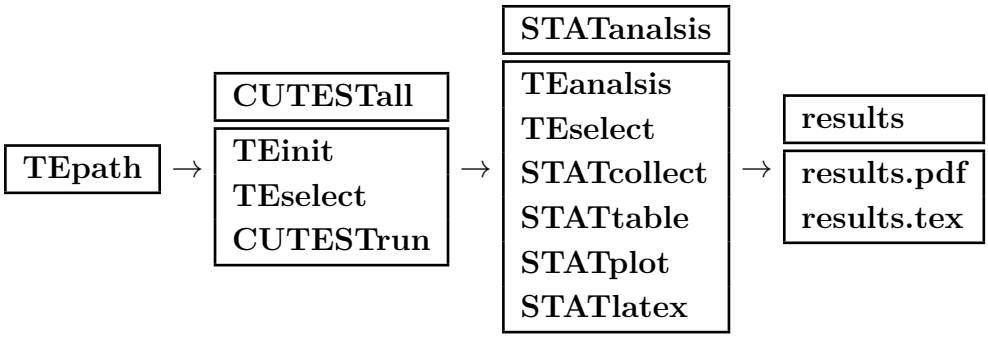
5.3 Automatic algorithm evaluation

There has been an increasing interest in automatic algorithm evaluation (**AAE**), e.g., Bischl et al. [26], Calheiros et al. [32], Cauwet et al. [36], He et al. [97], Hutter et al. [101], Kerschke et al. [110], Lindauer et al. [123], Malitsky [128], and Vermetten et al. [160].

Our automatic algorithm evaluation
• describes the results of a comprehensive experimental evaluation of all algorithms and identifies the best in a fully automatic principled and convenient way;
• calls all unconstrained and bound constrained `CUTEst` test problems by GOULD et al. [80] for all allowed dimensions and least squares problems by LUKŠAN et al. [127];
• runs any optimization software on two mentioned test collections;
• provides automatic solution statistics for all problems solved by at least one of the solvers;
• saves significant user time.

We consider the structure of **AAE** in Figure 5.1. Let us describe how to work it:

Figure 5.1: Automatic Algorithm Evaluation



• **TEpath** contains all necessary paths.
• **CUTESTall** uses
(1) **TEinit** to call all test problems for all allowed dimensions,
(2) **TEselect** to create problem info sorted by selected attributes,
(3) **CUTESTruns** to run multiple test problems from `CUTEst` on multiple solvers.
• **STATanalysis** uses
(1) **TEselect** to create problem info sorted by selected attributes,
(2) **TEanalsis** to analyze results from **CUTESTruns**,
(3) **STATcollect** to collect result analyzed by **TEanalsis**,
(4) **STATtable** to generate tables, **STATplot** to plot figures,
(5) **STATlatex** to give statistics as both **results.pdf** and **results.tex**.

5.4 Tools for refined statistics

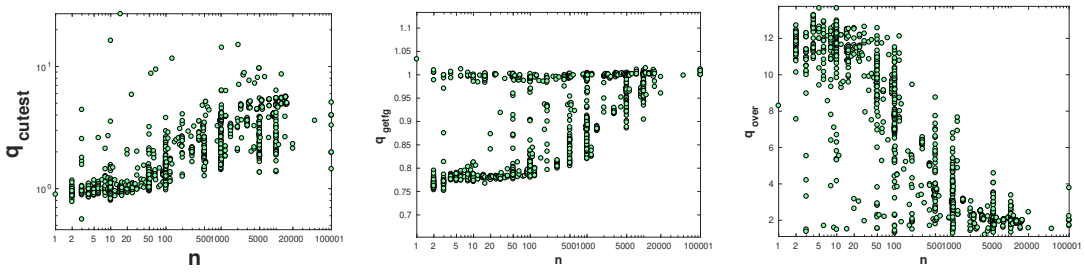
The **efficiency** of the solver *so* with respect to a cost measure c^{so} for solver *so* is defined by

$$e^{so} := \begin{cases} \dfrac{\min_{s\in\mathcal{S}} c^s}{c^{so}}, & \text{if } so \text{ solved the problem,} \\ 0, & \text{otherwise,} \end{cases} \tag{5.2}$$

where $\mathcal{S}$ is the list of solvers compared. The efficiency measures the ability of a solver $so \in \mathcal{S}$ relative to an ideal solver. Efficiency for all compared solvers with respect to various cost measures are summarized in the tables, called **efficiency tables**.

Cost measures are
- the number of function evaluations `nf`,
- the number of gradient evaluations `ng`,
- `nf2g` := `nf` + 2`ng`,
- time in milliseconds `msec`.

If the exact gradient is available, `ng` and `nf2g` are used. When we use the `CUTEst` test problems, the cost for computing the gradient is typically about twice that of the function value; hence `nf2g` is used as another cost measure. We now describe this. In [116], `getfg` have been introduced to compute the function value and gradient of function handle **fun** at x, collect statistics and enforce stopping tests. In `CUTEst`, both function value and gradient are computed by `cutest_obj` without returning any information about statistics. Subfigures (a) and (b) of Figure 5.2 show that the time for computing the gradient by `cutest_obj` and `getfg` are more than that of the function value, respectively. Hence, `nf2g` is a reasonable cost measure for the performance profile. In addition, Subfigure (c) of Figure 5.2 shows that `getfg` is more expansive than `cutest_obj` due to **collect statistics and enforce stopping tests**.



a) b) c)

Figure 5.2: Comparison of $q_{\texttt{cutest}} := \frac{t_g(\texttt{cutest})}{t_f(\texttt{cutest})}$, $q_{\texttt{getfg}} := \frac{t_g(\texttt{getfg})}{t_f(\texttt{getfg})}$ and $q_{\texttt{over}} := \frac{t_{f2g}(\texttt{getfg})}{t_{f2g}(\texttt{cutest})}$ versus dimensions, respectively, where t_f and t_g are considered the time to compute f and g by `cutest` or `getfg` and $t_{f2g} := t_f + 2t_g$.

In efficiency tables,
• efficiencies are given in percent; larger efficiencies in tables imply a better average behaviour while a zero efficiency indicates failure,
• all values are rounded (towards zero) to integers. Mean efficiencies are taken over all problems tried by all solvers and solved by at least one of them,
• if the exact gradient is available, #100 denotes the total number of test problems in which the solver needs the least number `nf2g`, and !100 the total number of test problems where the solver is the only one needing this many `nf2g`,
• otherwise, #100 denotes the total number of test problems in which the solver needs the least number `nf`, and !100 the total number of test problems where the solver is the only one needing this many `nf`,
• T_{mean} for a solver *so* is the average time needed to solve all test problems used, regardless of the time for unsolved problems. The columns titled "# of anomalies" report statistic on failure reasons:
(1) n indicates that either `nf2g` reaches its limit (`nf2gmax`) if gradient-based solvers are compared or `nf` reaches its limit (`nfmax`), otherwise.
(2) t indicates that `msec` reaches its limit (`secmax`).
(3) f indicates that the algorithm fails for other reasons.

5.5 Performance profiles and plots

To compare black box optimization solvers, we are interested in using performance profile and plot:
• The performance profile (Dolan & Moré [61]) displays the percentage of problems solved within a factor τ of the best solvers.
• The performance plot (Moré & Wild [131]) displays the percentage of problems solved within the number of function evaluations and time in milliseconds.
Both performance profile and plot are for `nf`/(`best nf`) and `msec`/(`best msec`) if the exact gradient is not available. Otherwise, the performance profile is used for `nf`/(`best nf`), `ng`/(`best ng`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`).

5.6 Stopping test

To compare black box solvers, the quotients

$$q^{so} := (f^{so} - f^{\text{opt}})/(f^0 - f^{\text{opt}}) \quad \text{for } so \in \mathcal{S} \tag{5.3}$$

are measures for identifying the convergence speed of solver *so* to reach a minimum of the smooth true function f (such quotients are not available in real applications). Note that this amounts to testing for finding the **global minimizer** to some reasonable accuracy. We did not check which of the test problems were multimodal, so that descent

algorithms might end up in a local minimum only. Here
• f^{so} is the best function value found by the solver *so*,
• f^0 is the function value at the starting point (common to all solvers),
• f^{opt} is the function value at the best point known to us (in most cases a global minimizer or at least a better local minimizer) found by performing a sequence of gradient-based and local/global gradient-free solvers; see Subsection 8.11.4,
• In the noisy optimization, ε depends on the dimension and the noise level because by increasing the noise level and the dimension the difficulty of problems is increased extremely. Hence ε is chosen slightly large for problems in medium and high dimensions in comparison with problems in low dimensions.

We consider a problem **solved** by the solver *so* if q^{so} is below a given threshold ε. Otherwise, the problem is called **unsolved** since either `nfmax` or `secmax` is exceeded. In this case, we run all black box solvers by monitoring in the function evaluation of a routine the number of function values and the time used until the bound of this number is met or an error occurs. We save time and number of function values at each improved function value and evaluated afterwards when q^{so} reaches ε. In order to get the above choices for `nfmax` and `secmax`, we run preliminarily to ensure that the best solver can solve most test problems. Both `nfmax` and `secmax` are input parameters for all black box solvers.

To compare gradient-based solvers, a solver *so* is stopped if the infinity norm of the reduced gradient at point x^{so} is below 10^{-6} and the problem is called **solved**. Otherwise, the problem is called **unsolved** since either `nf2gmax` or `secmax` is exceeded. In this case, we run all gradient-based solvers by monitoring in the function and gradient evaluation a routine the number of function and gradient evaluations and the time used until the bound of this number is met or an error occurs. We save time and number of function and gradient evaluations at each improved function value and evaluated afterwards when the infinity-norm of the reduced gradient at a point x^{so} is below 10^{-6}. In order to get the above choices for `nf2gmax` and `secmax`, we run preliminarily to ensure that the best solver can solve the most test problems. Both `nf2gmax` and `secmax` are input parameters for all gradient-based solvers.

Part II

New gradient-based optimization methods

6 A new limited memory method

This section discusses a new limited memory method for bound constrained optimization problems, called **LMBOPT**. This is a joint work with Arnold Neumaier and Behzad Azmi (cf. Kimiaei, Neumaier, & Azmi [118]). Recently, Neumaier & Azmi [136] gave a comprehensive convergence theory for a generic algorithm for bound constrained optimization problems with a continuously differentiable objective function. It combines an active set strategy with a gradient-free line search **CLS** along a piecewise linear search path defined by directions guarded against zigzagging. **LMBOPT** is an efficient implementation of this scheme. It employs a new limited memory techniques for computing the search directions, improves **CLS** by various safeguards relevant when finite precision arithmetic is used, and adds many practical enhancements in other details. The section compares **LMBOPT** and many other solvers on the unconstrained and bound constrained problems from the `CUTEst` collection and makes recommendations as to which solver should be used when. Depending on the problem class, the problem dimension, and the precise goal, the best solvers are found to be **LMBOPT**, **ASACG**, and **LMBFG-EIG-MS**.

6.1 LMBOPT – an efficient version of BOPT

We introduce a new **limited memory method for bound-constrained optimization** called **LMBOPT**. It conforms to the assumptions of **BOPT**, hence in exact precision arithmetic it converges and fixes all strongly active variables after finitely many iterations, but also takes care of various efficiency issues that are difficult to account for theory but need to be addressed in a robust and efficient implementation.

Important novelties compared to the literature include
• the useful trick of slightly moving starting point into the relative interior of the feasible domain $\mathbf{x}$,
• a useful starting direction based on the gradient signs,
• a new quadratic limited-memory model for progressing in a subspace,
• a numerically stable version of the descent direction proposed by Neumaier & Azmi [136] for removing zigzagging,
• a new regularized Krylov direction,
• safeguards for the curved line search accounting for effects due to finite precision

arithmetic,
• new heuristic methods for an initial step size, for a robust minimal step size, and for handling null steps without progress in the line search,
Taken together, these enhancements make **LMBOPT** very efficient and robust.

We describe how to compute search directions in Section 6.2:
• Subspace information is defined in Subsection 6.2.1.
• A new quasi Newton direction and a regularized Krylov step are introduced in Subsections 6.2.2 and 6.2.3, respectively.
• Some implementation details of these directions are given in Subsection 6.2.4.

Improvements in the line search are discussed in Section 6.3:
• Issues with finite arithmetic are described in Subsection 6.3.1.
• Ingredients of an improved version of **CLS** are introduced in Section 6.3.

We introduce the master algorithm and its implementation details in Section 6.4:
• A useful starting point is suggested in Subsection 6.4.1.
• What constitutes conditions on the acceptance of a new point is explained in Subsection 6.4.2.
• The master algorithm is introduced in Subsection 6.4.3.

Numerical results for unconstrained and bound constrained `CUTEst` problems [80] are summarized in Section 8.10:
• Details of test problems and a shifted starting point are discussed in Subsection 6.5.1. This subsection contains a list of all compared solvers and explains how unconstrained solvers turn into bound constrained solvers.
• The first numerical results are given in Subsection 6.5.2, resulting in the three best solvers (**LMBOPT**, **ASACG**, and **LMBFG-EIG-MS**). Then the second numerical results classified by constraints and dimensions are given, resulting in a solver choice in Subsection 6.5.4.
• The third numerical results for hard problems are given in Subsection 6.5.3.
• As a consequence, a solver choice based on our finding is made depending on the problem dimension, the presence and absence of constraints, the desired robustness, and the relative cost of function and gradient evaluations in Subsection 6.5.4.

The web site `http://www.mat.univie.ac.at/~neum/software/LMBOPT` contains public Matlab source code for **LMBOPT** together with more detailed documentation and an extensive list of tables and figures with numerical results and comparisons.

6.2 Search directions

In this section, we give a description of the search direction used at each iteration. In Subsection 6.2.1, subspace information is described. Accordingly, a new limited memory quasi Newton direction is discussed in Subsection 6.2.2. Then Subsection 6.2.3 describes

how Krylov directions are constructed and regularized. Finally, Subsection 6.2.4 contains the implementation details of our regularized Krylov direction.

6.2.1 Subspace information

After each iteration we form the differences

$$s = x - x^{\text{old}}, \quad y = g - g^{\text{old}},$$

where x, g are the current point and gradient, and $x^{\text{old}}, g^{\text{old}}$ are the previous point and gradient.

For some subspace dimension m, the matrix $S \in \mathbb{R}^{n\times m}$ has as columns (in the actual implementation a permutation of) m previous point differences s. A second matrix $Y \in \mathbb{R}^{n\times m}$ has as columns the corresponding gradient differences y.

If the objective function is quadratic with (symmetric) Hessian B and no rounding errors are made, the matrices $S, Y \in \mathbb{R}^{n\times m}$ satisfy the **quasi-Newton condition**

$$BS = Y. \tag{6.1}$$

Since B is symmetric,

$$H := S^T Y = S^T BS \tag{6.2}$$

must be symmetric. If we calculate $y = Bs$ at the direction $s \neq 0$, we have the consistency relations

$$h := S^T Bs = Y^T s = S^T y, \tag{6.3}$$

$$0 < \gamma := s^T Bs = y^T s, \tag{6.4}$$

for all $\alpha \in \mathbb{R}$ in exact precision arithmetic. If the columns of S (and hence those of Y) are linearly independent then $m \leq n$, and H is positive definite. Then the minimum of $f(x + Sz)$ with respect to $z \in \mathbb{R}^m$ is attained at

$$z^{\text{new}} := -H^{-1}c \;\text{ with } c := S^T g, \tag{6.5}$$

where the associated point and gradient are

$$x^{\text{new}} := x + Sz^{\text{new}}, \quad g^{\text{new}} := g(x^{\text{new}}) = g + Yz^{\text{new}},$$

and we have

$$S^T g(x^{\text{new}}) = 0. \tag{6.6}$$

If m reaches its limits, we use $\gamma := y^T s$ and form the augmented matrices

$$S^{\text{new}} := (\, S \;\; s \,), \quad Y^{\text{new}} := BS^{\text{new}} := (\, Y \;\; y \,),$$

$$H^{\text{new}} := (S^{\text{new}})^T BS^{\text{new}} := \begin{pmatrix} H & h \\ h^T & \gamma \end{pmatrix}, \tag{6.7}$$

the augmented vector $c^{\text{new}} := (S^{\text{new}})^T g^{\text{new}} = \begin{pmatrix} 0 \\ s^T g^{\text{new}} \end{pmatrix}$, and put

$$z^{\text{new}} := -(H^{\text{new}})^{-1} c^{\text{new}}. \tag{6.8}$$

But if the allowed memory for S and Y is used we delete the oldest column of S and Y and the corresponding row and column of H to make room for the new pair of vectors, and then augment as above.

The implementation contains a Boolean variable `updateH` as a tuning parameter to compute

$$h := \begin{cases} S^T y & \text{if } \texttt{updateH}, \\ Y^T s & \text{otherwise.} \end{cases} \tag{6.9}$$

If the objective is not quadratic, (6.1) does not hold exactly and $H := S^T Y$ does not need to be symmetric. However, the update (6.7) always produces a symmetric H, even in finite precision arithmetic.

6.2.2 A new quasi-Newton direction

We use S and Y to construct a Hessian approximation of the form

$$B = D + WXW^T, \tag{6.10}$$

for some symmetric matrix $W \in \mathbb{R}^{n\times m}$ and some matrix $X \in \mathbb{R}^{n\times m}$. Thus, temporarily, the additional assumption is made that B deviates from a diagonal matrix D by a matrix of rank at most m. Under these assumptions, we reconstruct the Hessian uniquely from the data S and $Y = BS$, in a manifestly symmetric form that can be used as a surrogate Hessian even when this structural assumption is not satisfied.

This provides an efficient alternative to the traditional L-BFGS-B formula [29], which needs twice as much storage and computation time.

6.2.1 Theorem. *Let $D \in \mathbb{R}^{n\times n}$ be diagonal, $\Sigma \in \mathbb{R}^{m\times m}$ and $U \in \mathbb{R}^{n\times m}$. If XW^TS is invertible then* (6.1) *and* (6.10) *imply*

$$B = D + U\Sigma^{-1}U^T, \tag{6.11}$$

where

$$U := Y - DS \tag{6.12}$$

and

$$\Sigma := U^T S \tag{6.13}$$

is symmetric. The solution of $Bp = -g$ is given in terms of the symmetric matrix

$$M := U^T D^{-1} Y = \Sigma^{-1}, \tag{6.14}$$

by the solution $p = D^{-1}(Uz - g)$ of $Mz = U^T D^{-1} g$.

Proof. The matrices $U := Y - DS$ and $\Sigma := U^T S$ are computable from S and Y, and we have

$$U = Y - DS = BS - DS = (B - D)S = WXW^T S,$$

and since B is symmetric, $\Sigma = S^T(B-D)S$ is symmetric, too. By assumption, the $m \times m$ matrix $Z := XW^T S$ is invertible, hence $W = UZ^{-1}$ and $Z = XZ^{-T}U^T S = XZ^{-T}\Sigma$. This product relation and the invertibility of Z imply that Σ is invertible, too, and we conclude that $X = Z\Sigma^{-1}Z^T$, hence

$$B = D + UZ^{-1}XZ^{-T}U^T = D + U\Sigma^{-1}U^T.$$

□

To apply it to the bound constrained case, we note that the first order optimality condition predicts the point $x + p$, where the nonactive part p_I of p solves the equation

$$B_{II}p_I = -g_I.$$

Noting that

$$B_{II} = D_{II} + U_{I:}\Sigma^{-1}U_{I:}^T,$$

we find $D_{II}p_I + U_{I:}\Sigma^{-1}U_{I:}^T p_I = -g_I$, hence

$$p_I = D_{II}^{-1}(U_{I:}z - g_I),$$

where $z := -\Sigma^{-1}U_{I:}^T p_I$. Now $-\Sigma z = U_{I:}^T p_I = U_{I:}^T D_{II}^{-1}(U_{I:}z - g_I)$, hence z solves the linear system

$$Mz = U_{I:}^T D_{II}^{-1} g_I.$$

Here $M := \Sigma + U_{I:}^T D_{II}^{-1} U_{I:}$ is equivalent to (9.28) by setting $Y = U + DS$ in (9.28) and using (6.13). With the symmetric matrix H defined by (6.2), we compute the symmetric $m \times m$ matrix M

$$\begin{aligned} M &= U_{I:}^T D_{II}^{-1} Y_{I:} = (Y_{I:} - D_{II}S_{I:})^T D_{II}^{-1} Y_{I:} = Y_{I:}^T D_{II}^{-1} Y_{I:} - S_{I:}^T Y_{I:} \\ &= Y_{I:}^T D_{II}^{-1} Y_{I:} - H \end{aligned} \tag{6.15}$$

and find

$$z = M^{-1}U_{I:}^T D_{II}^{-1} g_I; \tag{6.16}$$

hence

$$p_I = D_{II}^{-1}(U_{I:}z - g_I). \tag{6.17}$$

Here, for $i = 1, \cdots, n$,

$$D_{ii} := \sqrt{\sum_{j \in J} \mathtt{YY}_{ij} \Big/ \sum_{j \in J} \mathtt{SS}_{ij}} \tag{6.18}$$

with

$$\mathtt{YY} = Y_{IJ} \circ Y_{IJ}, \quad \mathtt{SS} = S_{IJ} \circ S_{IJ},$$

where J contains the indices of newest and oldest pair (s, y) and $\circ$ denotes componentwise multiplication.

Enforcing the angle condition. Due to rounding errors, a computed descent direction p need not satisfy the angle condition (4.15). We may add a multiple of the gradient to enforce the angle condition (4.15) for the modified direction

$$p^{\text{new}} := p - tg \tag{6.19}$$

with a suitable factor $t \geq 0$; the case $t = 0$ corresponds to the case where p already satisfies the bounded angle condition (4.15). The choice of t depends on the three numbers

$$\sigma_1 := g^T g > 0, \quad \sigma_2 := p^T p > 0, \quad \sigma := g^T p;$$

these are related by the Cauchy–Schwarz inequality

$$\sigma^{\text{new}} := \frac{\sigma}{\sqrt{\sigma_1 \sigma_2}} \in [-1, 1].$$

We want to choose t such that the angle condition (4.15) holds with p^{new} in the place of p. If $\sigma^{\text{new}} \leq -\Delta^{\text{a}}$, this holds for $t = 0$, and we make this choice. Otherwise we may enforce the equality (4.15) by choosing

$$t := \frac{\sigma + \Delta^{\text{a}}\sqrt{w}}{\sigma_1} \quad \text{with} \quad w := \frac{\sigma_1 \sigma_2 (1 - (\sigma^{\text{new}})^2)}{1 - (\Delta^{\text{a}})^2}. \tag{6.20}$$

The following proposition is a special case of Proposition 5.2 in [136].

6.2.2 Proposition. *Suppose that $g \neq 0$ and $0 < \Delta^{\text{a}} < 1$. Then if t is chosen by* (6.20)*, the search direction* (6.21) *satisfies the angle condition* (4.15)*.*

Given z by (6.16), we could compute the nonactive part of p from (6.17); however, this need not lead to a descent direction since B need not be positive definite. We therefore compute

$$u := U_{I:} z,$$

and choose

$$p_I = D_{II}^{-1}(u - tg_I), \tag{6.21}$$

where t is chosen analogous to (6.20) if this results in $t < 1$, and $t = 1$ otherwise. By Proposition 6.2.2, the direction (6.21) satisfies the angle condition (4.15).

6.2.3 A regularized Krylov direction

Neumaier & Azmi [136, Section 7] introduced a new nonlinear conjugate gradient method against zigzagging for unconstrained optimization which, applied to the working

subspace, may be used by **BOPT** to generate search directions as long as the active set does not change.

Any search direction p must satisfy $g^T p < 0$. In order to avoid zigzagging, [136] generated the search direction p as the vector with a fixed value $g^T p = -\overline{c} < 0$ closest (with respect to the 2-norm) to the previous search direction p^{old}. By Theorem 7.1 in [136] (applied for $B = I$),

$$p = \overline{\beta} p^{\text{old}} - \widehat{\lambda} g, \tag{6.22}$$

with

$$\overline{\beta} > 0, \quad \widehat{\lambda} = \frac{\overline{c} + \overline{\beta} g^T p^{\text{old}}}{g^T g}. \tag{6.23}$$

The resulting method has finite termination on quadratic objective functions, where it reduces to linear conjugate gradients.

[136, Theorem 7.3] shows that the bounded angle condition holds for sufficiently large ℓ if an efficient line search such as **CLS** is used and there are positive constants κ_1 and κ_2 such that either p^ℓ is parallel to the steepest descent direction $-g^\ell$ or the conditions

$$(g^\ell)^T g^\ell \leq \kappa_1 (y^{\ell-1})^T y^{\ell-1}, \tag{6.24}$$

$$(y^{\ell-1})^T p^{\ell-1} \leq \kappa_2 (g^{\ell-1})^T p^{\ell-1} \tag{6.25}$$

hold (where $y^{\ell-1} := g^\ell - g^{\ell-1}$). Convergence is locally linear when the sequence x^ℓ converges to a strong local minimizer.

As in [136, Theorem 7.5] the sequence generated by (6.22) and (6.23) can be rewritten as the nonlinear conjugate gradient method by Fletcher & Reeves [76] which is equivalent to the linear conjugate gradient method by Hestenes & Stiefel [98] when f is quadratic with the positive definite Hessian matrix and bounded. Hence it needs at most n steps to get a minimizer of f.

As a consequence of [136, Theorem 7.3], [136, Theorem 7.6] showed that the sequence x^ℓ of the conjugate gradient method generated by (6.22) and (6.23) satisfies

$$\inf_\ell \|g^\ell\|_* = 0 \text{ or } \lim_{\ell \to \infty} f^\ell = -\infty$$

and convergence is locally linear if the sequence x^ℓ converges to a strong local minimizer.

This section discusses a new Krylov method which is equivalent to the linear conjugate gradient method by Hestenes & Stiefel [98] in the cases where f is quadratic with the positive definite Hessian matrix and bounded. Theorems 7.3, 7.5, 7.6 in [136] are valid for our Krylov method.

In the subspace spanned by the columns of S, $x + Sz$ minimizes the quadratic model function. In the bigger subspace consisting of all spanned by the columns of S and

arbitrarily descent direction $\widetilde{p} \in \mathbb{R}^n$, $\widetilde{s} := -\zeta\widetilde{p} + S\widetilde{z}$ with $\zeta \in \mathbb{R}$ and $\widetilde{z} \in \mathbb{R}^m$, we find the optimal point $x_{\text{new}} = x + \widetilde{s}$ by minimizing the quadratic model function

$$f(x+\widetilde{s}) - f(x) \approx g^T\widetilde{s} + \frac{1}{2}\widetilde{s}^T B\widetilde{s} = Q(\zeta, \widetilde{z})$$

with

$$\begin{aligned} Q(\zeta,\widetilde{z}) &= -\zeta g(x)^T\widetilde{p} + c^T\widetilde{z} + \frac{1}{2}(\zeta^2\widetilde{p}^T B\widetilde{p} - 2\zeta\widetilde{z}^T Y^T\widetilde{p} + \widetilde{z}^T S^T BS\widetilde{z}). \\ &= -\zeta g(x)^T\widetilde{p} + c^T\widetilde{z} + \frac{1}{2}(\zeta^2\widetilde{\gamma} - 2\zeta q^T\widetilde{z} + \widetilde{z}^T H\widetilde{z}) \end{aligned}$$

using $\widetilde{\gamma} := \widetilde{p}^T B\widetilde{p}$, $q := Y^T\widetilde{p}$, and (6.2). In fact Q is minimized when its gradient vanishes, i.e.,

$$-g(x)^T\widetilde{p} + \zeta\widetilde{\gamma} - q^T\widetilde{z} = 0, \tag{6.26}$$

$$c - \zeta q + H\widetilde{z} = 0. \tag{6.27}$$

By multiplying the inverse matrix H in (6.27), setting $H^{-1}c = -z$ and $H^{-1}q = r$ in it, we get

$$\widetilde{z} := z + \zeta r. \tag{6.28}$$

Then multiplying the vector q in (6.28) results in

$$-q^T z - \zeta q^T r + q^T\widetilde{z} = 0 \tag{6.29}$$

Inserting (6.29) into (6.26) gives

$$\zeta := \frac{q^T z + g^T\widetilde{p}}{\widetilde{\gamma} - q^T r}. \tag{6.30}$$

We use the approximation

$$\widetilde{\gamma} = \widetilde{p}^T B\widetilde{p} \approx \frac{f(x+\alpha\widetilde{p}) - f - \alpha g^T\widetilde{p}}{\alpha^2/2}.$$

In finite precision arithmetic, a tiny denominator in (6.30) produces a vary inaccurate ζ. This drawback is overcome by regularization. The error made in $\widetilde{\gamma}$ is a tiny multiple of

$$\widetilde{\gamma}_e := \frac{|f(x+\alpha\widetilde{p}) - f| + \alpha|g|^T|\widetilde{p}|}{\alpha^2/2}. \tag{6.31}$$

We therefore shift the denominator in (6.30) away from zero to

$$\nu := \begin{cases} \widetilde{\gamma} - q^T r + \Delta^\nu(\widetilde{\gamma}_e/2 + |q|^T|r|) & \text{if } \widetilde{\gamma} \geq q^T r, \\ \widetilde{\gamma} - q^T r - \Delta^\nu(\widetilde{\gamma}_e/2 + |q|^T|r|) & \text{otherwise,} \end{cases} \tag{6.32}$$

where $\Delta^\nu \in (0,1)$ is a tiny factor and replace ζ by the regularized version

$$\zeta^{\text{reg}} := \frac{q^T z + g^T\widetilde{p}}{\nu}. \tag{6.33}$$

Hence the optimal point can be rewritten as the new point

$$x_{\text{new}} = x + \widetilde{s} = x + p^{\text{new}},$$

where

$$p^{\text{new}} := -\zeta^{\text{reg}}\widetilde{p} + S(z + \zeta^{\text{reg}} r) \tag{6.34}$$

is called the **regularized Krylov direction**. If the condition

$$g^T p^{\text{new}} = -\zeta^{\text{reg}} g^T \widetilde{p} + c^T (z + \zeta^{\text{reg}} r) \tag{6.35}$$

is negative, (6.34) is a descent direction of $f(x)$.

The above also holds if the subspace dimension m is not zero. Otherwise the formulas (6.30), (6.34)–(6.35) reduce to

$$p = \widetilde{p},\ \zeta = \zeta^{\text{reg}} = g^T \widetilde{p}/\widetilde{\gamma},\ p^{\text{new}} = -\zeta^{\text{reg}}\widetilde{p},\ g^T p^{\text{new}} = -\zeta^{\text{reg}} g^T \widetilde{p}.$$

We apply this in the bound-constraiend case to subvectors indexed by I, and limit the subspace dimension by picking only columns from S , Y, and H with index in a list `hist` determined by the iteration history.

6.2.4 Some implementation details

This section discusses how the working set I is implemented and $\widetilde{p}$ is computed which affect our regularized Krylov direction. Then it describes how to implement our regularized Krylov direction.

When the activity does not change, the step is called **local step**. Our regularized Krylov direction to be updated, restricted, and restarted needs what the subspace dimension is. We denote by `nlocal` the number of local steps and by `nwait` the number of local steps before the regularized Krylov direction is started, which will be a tuning parameter. We use `nlocal` and `nwait` to determine the subspace dimension.

In order to determine the working set I, the following are checked:
(1) The function value does not decreased.
(2) The size of new free index set is smaller than that of the old free index set (i.e., the activity is not fixed).
(3) The maximal number of local steps before the freeing iteration (which is a tuning parameter) is found is exceeded.
(4) Condition (4.13) is violated.

We use the algorithms **findFreePos** and **findFreeNeg** to get the working set. At the first iteration, **BOPT** calls **findFreePos** to find $I_+(x)$ by (4.11) and initializes the working set by $I(x) := I_+(x)$. Then if the statements (1)-(3) hold, **findFreeNeg** finds

$I_-(x)$ by (4.10) and **findFreePos** checks whether the statement (4) holds or not. If this statement does not hold, the working set is $I(x) := I_-(x)$; otherwise, **findFreePos** finds $I_+(x)$ by (4.11) and chooses it as a new working set; $I(x) := I_+(x)$.

Before our regularized Krylov direction is computed, $\widetilde{p}$ needs to be computed. Then the bigger subspace can be constructed. **searchDir** computes $\widetilde{p}$ by calling either **scaleDir**, **quasiNewtonDir**, or **AvoidZigzagDir**:
• In the first iteration the starting search direction makes use of the gradient signs only, and has nonzero entries in some components that can vary. Each **starting search direction** is computed by **scaleDir**. In this case, **scaleDir**, for $i = 1, \cdots, n$, computes

$$\mathtt{sc} := \min(1, \overline{x}_i - \underline{x}_i) \;\text{ and }\; \widetilde{p}_i := \begin{cases} \mathtt{sc} & \text{if } g_i < 0, \\ -\mathtt{sc} & \text{otherwise} \end{cases}$$

if $x_i = 0$; otherwise, it sets $\mathtt{sc} = |x_i|$ and computes

$$\widetilde{p}_i := \begin{cases} \mathtt{sc} & \text{if } x_i = \underline{x}_i, \\ -\mathtt{sc} & \text{elseif } x_i = \overline{x}_i, \\ \mathtt{sc} & \text{elseif } g_i < 0, \\ -\mathtt{sc} & \text{otherwise.} \end{cases}$$

• If $\mathtt{nlocal} \neq \mathtt{nwait}$, a modified direction is used to avoid zigzagging. $\widetilde{p}$ is computed by (6.22) using (6.23). Since $g_I^T \widetilde{p}_I = -\overline{c}$, the direction will be a descent direction. This direction is implemented by **AvoidZigzagDir** and enriched by a **new heuristic choice** of

$$\overline{\beta} := \theta \max_{i=1:n} \left\{ \left| \frac{g_i}{\widetilde{p}_i} \right| \right\}, \tag{6.36}$$

with tuning parameters $0 < \theta < 1$ and $\overline{c} > 0$.
• Otherwise, **quasiNewtonDir** is used in subspace. If $D_{ii} \in [(\Delta^D)^{-1}, \Delta^D]$ is violated, $D_{ii} = 1$, where $\Delta^D > 1$ is a tuning parameter.

Afterwards, **enforceAngle** is used if the angle condition (4.15) does not hold:
• If $g_I^T \widetilde{p}_I > 0$, $\widetilde{p}_I$ is chosen to be its opposite to move away from maximizer or saddle point.
• By changing the sign of g, it may enforce $g_I^T \widetilde{p}_I \leq 0$. Even though $g \neq 0$, cancellation may lead to a tiny $g_I^T \widetilde{p}_I$ (and even of the wrong sign). Given a tiny parameter Δ^{pg}, to overcome this weakness, subtract $\Delta^{pg} |g_I|^T |\widetilde{p}_I|$ can be a bound on the rounding error to have the theoretically correct sign. A **regularized directional derivative** is done if the condition

$$|g_I^T \widetilde{p}_I| \leq \Delta^{pg} |g_I|^T |\widetilde{p}_I| \tag{6.37}$$

holds, enforcing $g_I^T \widetilde{p}_I < 0$. In this case, if (6.37) holds, $\widetilde{p}_I$ is either $-g_I$ or $-\lambda^b g_I$. Here $\lambda^b := \max_{i \in I} \{D_{ii}\}$.
• If at least one of the conditions $w > 0$ and $0 \leq |t| < \infty$ does not hold, $\widetilde{p}_I$ is chosen to be $-\lambda^b g_I$.

If the statements (1)-(4) hold and the activity is fixed, a scaled Cauchy point is tried. It is computed in the same way as [104] but with the difference that the scaling matrix is computed by

$$\overline{D}_{ii} := \sqrt{\sum_{j=1}^{m} \mathtt{YY}_{ij} \Big/ \sum_{j=1}^{m} \mathtt{SS}_{ij}}, \quad \text{for } i = 1, \cdots, n,$$

with

$$\mathtt{YY} = Y \circ Y, \quad \mathtt{SS} = S \circ S,$$

where $\circ$ denotes componentwise multiplication, if at least once S and Y are updated. Otherwise, it is computed by

$$\overline{D} := \left| g \Big/ \widetilde{p} \right|,$$

where $\widetilde{p}$ is computed as discussed above.

findFreeNeg evaluates `nlocal` and **typeSubspace** determines the list `hist` of subspace basis indices and the subspace dimension. If the statement (2) holds (activity is fixed) `nlocal` is restarted with `nlocal` $= 0$. Otherwise we have one of two cases:
Case 1. The function value at the current point is less than that of the best point previously found. Then:
(i) If the function f is close to quadratic, the current iteration is **near a local minimizer**. In this case $\widetilde{p}$ is computed by **AvoidZigzagDir** against zigzagging and the regularized Krylov direction (6.34) is constructed effectively in the next iteration. If the subspace dimension does not reached its limit, before the regularized Krylov direction (6.34) is computed a **restricting procedure** is done by increasing `nlocal` by one and using only the newest columns of S, Y, `hist`, and rows and columns of H. Otherwise the regularized Krylov direction (6.34) is computed in the maximal subspace.
(ii) Otherwise, if the function f is not quadratic, the current iteration is **far from a local minimizer**. If `nlocal` does not reach `nwait`, `nlocal` is increased by one and $\widetilde{p}$ is computed by **AvoidZigzagDir**. Otherwise `nlocal` reaches `nwait`. A **restarting procedure** is done by permuting `hist` such that the newest columns of S, Y, `hist`, and rows and columns of H are used to compute $\widetilde{p}$ by **quasiNewtonDir**. Then the subspace dimension becomes zero and `hist` becomes empty for the next iteration. In both cases the regularized Krylov direction (6.34) is reduced to $p^{\text{new}} = -\zeta^{\text{reg}}\widetilde{p}$.
Case 2. The function value at the current point is not less than that of the best point previously found. In this case the current iteration may be either close to or far from a local minimizer. If the subspace dimension can be increased, a restricting procedure is done while $\widetilde{p}$ is computed by **AvoidZigzagDir** in the next iteration and the regularized Krylov direction (6.34) is computed accordingly. Otherwise `nlocal` reaches `nwait`. In this case a restarting procedure is done and the regularized Krylov direction (6.34) is reduced to $p^{\text{new}} = -\zeta^{\text{reg}}\widetilde{p}$.

Regardless of whether the activity changes or not, the subspace cannot be updated whenever very little progress is made, $y \approx 0$, while the gradient is still large., i.e., a new

pair (s, y) violates the condition

$$|g^T y| \geq \Delta^{po} g^T g, \tag{6.38}$$

where $\Delta^{po} \in (0, 1)$ is a tiny tuning parameter. Initially $m = 0$ and whenever a new pair (s, y) satisfies (6.38), m is increased by one by appending these vectors to S and Y, respectively. But once m reaches its limits, it is kept to be fixed and the oldest column of S and Y is replaced by s and y, respectively.

In summary, the implementation of our regularized Krylov direction, called **KrylovDir**, for computing p in (6.34) is given as follows: (i) $\widetilde{\gamma}_e$ is computed according to (6.31) by **getGam**, (ii) if the subspace dimension is not zero, our Krylov direction is used; otherwise, it reduces to $p^{\text{new}} = -\zeta^{\text{reg}}\widetilde{p}$, (iii) a regularization for the denominator of (6.33) is made according to (6.32) by **regDenom**, (iv) the condition (6.35) is computed to know whether the regularized Krylov direction is descent or not, (v) the new trial point, $x + p^{\text{new}}$, into $\mathbf{x}$, is projected, resulting in x^{new} and recomputing the direction by $p^{\text{new}} := x^{\text{new}} - x$.

6.3 Improvements in the line search

In this section, an improved version of **CLS**, called **CLS-new**, is introduced with **enhancements for numerical stability** (finding a **starting good step size**, a **target step size** and a **minimum step size** with safeguards in a finite precision arithmetic). The variable `eff` indicates what is the status of step in **CLS-new** – taking values 1 (efficient step), 2 (non-monotone step), 3 (inefficient decrease), and 4 (inefficient step).

6.3.1 Issues with finite precision arithmetic

Rounding errors prevent descent for step sizes that are too small.

6.3.1 Example. We consider the function

$$f(x) = x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120.$$

For $x = 5 + 3 \times 10^{-10}$ and $p = -1$, the plot $f(x + \alpha p)$ versus α in Figure 6.1 shows that one needs to find a sensible minimal step size.

In practice if the step size is too small, rounding errors will often prevent that the function value is strictly decreasing. Due to cancellation of leading digits, the Goldstein quotient can become very inaccurate, which may lead to a wrong bracket and then to a failure of the line search. The danger is particularly likely when the search direction is almost orthogonal to the gradient. Hence, before doing each line search method, we

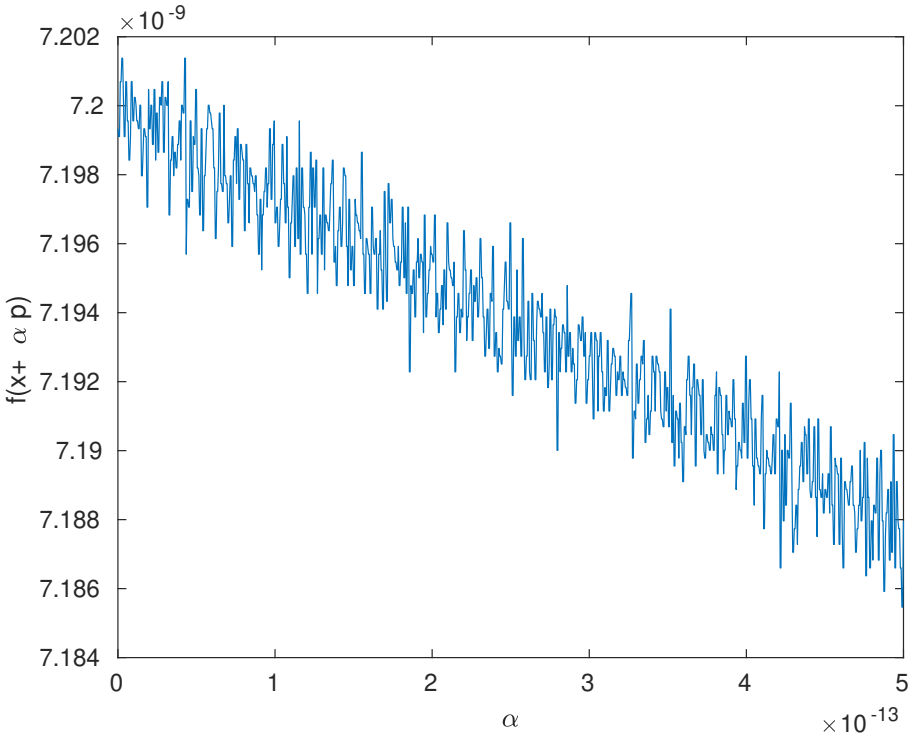


Figure 6.1: In the Example 6.3.1 points with step sizes $\alpha < 0.5 \times 10^{-13}$ have a high probability for having $f(x + \alpha p) \geq f(x)$.

need to produce a starting step size by a method we call **goodStep** to find the **starting good step size** α^{good}, the **target step size** α^{target}, and the **minimum step size** $\alpha^{\min}$ with safeguards in a finite precision arithmetic. **goodStep** computes the first and second breakpoint, respectively, by

$$\underline{\alpha}^{\text{break}} := \min\{(\underline{x}_i - x_i)/p_i \mid i \in \underline{\texttt{ind}}\}, \quad \overline{\alpha}^{\text{break}} := \min\{(\overline{x}_i - x_i)/p_i \mid i \in \overline{\texttt{ind}}\}.$$

Here $\underline{\texttt{ind}} := \{i \mid p_i < 0 \ \&\ x_i > \underline{x}_i\}$ is the indices of the first breakpoint and $\overline{\texttt{ind}} = \{i \mid p_i > 0 \ \&\ x_i < \overline{x}_i\}$ is the indices of the second breakpoint. Then it computes the **breakpoint** by $\alpha^{\text{break}} := \min(\underline{\alpha}^{\text{break}}, \overline{\alpha}^{\text{break}})$ in finite precision arithmetic and adjusts it by $\alpha^{\text{break}} := \alpha^{\text{break}}(1 + \Delta^b)$, where $\Delta^b \in (0, 1)$ is a tiny factor for adjusting a target step size. In the cases where $\underline{\texttt{ind}}$ and $\overline{\texttt{ind}}$ are empty, we set $\underline{\alpha}^{\text{break}} := +\infty$ and $\overline{\alpha}^{\text{break}} := +\infty$. Given a tiny factor $\Delta^\alpha \in (0, 1)$ and an index set $\texttt{indp} := \{i \mid p_i \neq 0\}$, the **minimal step size** is computed by a heuristic formula

$$\alpha^{\min} := \begin{cases} \min\left(1, \Delta^\alpha \left|\dfrac{f}{g^T p}\right|\right) & \text{if } x = 0 \text{ and } \texttt{indp} \neq \emptyset, \\ \min\left(1, \Delta^\alpha \min\left(\left|\dfrac{f}{g^T p}\right|, \min\limits_{i \in \texttt{ind}} \left\{\left|\dfrac{x_i}{p_i}\right|\right\}\right)\right) & \text{elseif } \texttt{indp} \neq \emptyset, \\ 1 & \text{otherwise,} \end{cases}$$

and the **target step size** is chosen by $\alpha^{\text{target}} := \max(\alpha^{\min}, \texttt{df}/|g^T p|)$. We discuss how `df` is computed in the next subsection. If an exact line search on quadratic is requested, α^{target} is restricted by

$$\alpha^{\text{target}} := \min(\alpha^{\text{target}}, \alpha^{\text{break}}).$$

In the special case, if $\alpha^{\min} = 1$, $\alpha^{\text{good}} := 1$ and **goodStep** ends due to being the zeros direction; otherwise, it computes the **good step size** by

$$\alpha^{\text{good}} := \begin{cases} \alpha^{\text{target}} & \text{if } q\alpha^{\text{target}} \leq \alpha^{\text{break}}, \\ \max(\alpha^{\min}, \alpha^{\text{break}}) & \text{otherwise;} \end{cases}$$

when it equals $\alpha^{\min}$, adverse finite precision effects are avoided. Here $q > 1$ is an input parameter for **goodStep** which is used to expand (reduce) step sizes by **CLS-new**.

The number of **stuck iterations** is the number of times for which the best point cannot be updated, denoted by `nstuck`. Its limits are `nstuckmax` (maximum number of all stuck iterations) and `nsmin` (how many stucks are allowed before a trial point is accepted), both of which will be tuning parameters. In the final step of **goodStep**, if $\texttt{nstuck} \geq \texttt{nsmin}$, α^{good} is increased by the factor $2 * \texttt{nstuck}$ to avoid remaining stuck.

6.3.2 CLS-new – an improved version of CLS

Before **CLS-new** tries to enforce the sufficient descent condition (4.20), the following steps need to be done:

• **LMBOPT** calls **enforceAngle** to enforce the angle condition (4.15).

• Once **LMBOPT** calls **initInfo** to initialize the best function value by $f^{\rm best} := f^0$ and to compute the factor for adjusting increases in f (discussed below) by

$$\delta_f := \begin{cases} \texttt{facf} * |f^0| & \text{if } f^0 \in (0, \infty), \\ 1 & \text{otherwise,} \end{cases} \tag{6.39}$$

where $\texttt{facf} > 0$ is a relative accuracy of f^0. We denote the list of acceptable increases in f by `Df` and its size by `mf` and the number of gradient evaluations by `ng`. Moreover, **initInfo** chooses, for $i = 1, \cdots, \texttt{mf} - 1$, $\texttt{Df}_i := -\infty$ and $\texttt{Df}_{\texttt{mf}} := \delta_f$. After the first call to **CLS-new**, **LMBOPT** always calls **updateInfo** to update
(1) the number of times that the best point is not updated by

$$\texttt{nstuck} := \begin{cases} 0 & \text{if } f^{\rm new} < f^{\rm best}, \\ \texttt{nstuck} + 1 & \text{otherwise;} \end{cases}$$

(2) the best point information by $f^{\rm best} := f^{\rm new}$ and $x^{\rm best} := x^{\rm new}$ if $\texttt{nstuck} = 0$;
(3) δ_f and `Df`. In this case, if $f^{\rm new} < f$, then $\delta_f := f - f^{\rm new}$ and $\texttt{nm} := \text{mod}(\texttt{ng}, \texttt{mf})$ are computed. Otherwise since the function value is not decreased, δ_f is expanded by $\delta_f := \max(\Delta_f \delta_f, \Delta^m(|f| + |f^{\rm new}|))$ and $\texttt{nm} := \text{mod}(\texttt{ng}, \texttt{mf})$ is updated. Here $\Delta^m \in (0, 1)$ is a tiny factor for adjusting δ_f and $\Delta_f > 1$ is a tuning parameter for expanding δ_f. If `nm` is zero, the last component of `Df` is replaced by δ_f; otherwise, the `nm`th component of `Df` is replaced by δ_f;
(4) f by $f^{\rm new}$ if $f^{\rm new} < f$ holds.
• If $\alpha^{\rm good} \geq 1$, q is updated by $q = \max\left(q^{\min}, q/\Delta^q\right)$, where $1 < q^{\min} < q$ and $0 < \Delta^q < 1$ are the tuning parameters. Whenever the term $q\alpha^{\rm good}$ is moderately large, this choice may helps **CLS-new** to prevent a failure. To get target step sizes which **should not become too tiny**, an acceptable increase in f (denoted by `df`) needs to be estimated in a heuristic way such that it becomes slowly small. Accordingly, at first, `df` is δ_f computed by (6.39). Next, it is either a multiple of the old δ_f value if the tuning parameter `mdf` divides `ng` or the maximum of `mf` old δ_f values, otherwise. In this case, target step sizes do not become too small.
• **goodStep** is used to find an initial step size. **CLS-new** tries to find a step size $\alpha > 0$ satisfying the sufficient descent condition (4.20).
• **CLS-new** ends once the sufficient descent condition holds, resulting in the line search being **efficient** and $\texttt{eff} = 1$.
• In the first iteration if the Goldstein quotient $\mu(\alpha) < 1$ an exact line search method results in the **secant step** $\frac{1}{2}\alpha/(1 - \mu(\alpha))$ for the quadratic objective function. In fact this ensures the finite termination of our Krylov method on the quadratic functions. Otherwise an **extrapolation** is done by the factor $q > 1$. In the next iteration, if the sufficient descent condition (4.20) does not hold, then the function is far from quadratic and bounded. In such a case, either an **interpolate** is performed if the lower bound for step size is zero or an **extrapolation** is done by the factor $q > 1$ until a bracket $[\underline{\alpha}, \overline{\alpha}]$ is found. Then, a **geometric** mean of $\underline{\alpha}$ and $\overline{\alpha}$ is used.
• A limit on the number of iterations is used.

• At the end, if **CLS-new** fails to give an improvement on the function values, **LMBOPT** calls **robustStep** to find a step size with corresponding lowest function value. Such a step size is called **robust**. Using a list of differences of the current best function value with the function values at trial points as **gains**, **robustStep** tries to find a point with **smallest robust change** if the minimum of gains is smaller than or equal to the acceptable increase in f (`df`). Otherwise, if the function is almost flat or flat; then a step with **largest gain** is chosen. Otherwise, a point with **nonrobust change** might be chosen provided that the minimum of gains $\leq \Delta^r$`df`, where $\Delta^r > 0$ is a tuning parameter.

After **LMBOPT** accepts a new point generated x^{new} and its step size α by **CLS-new**, the new step is defined by $s := x^{\text{new}} - x = \alpha\|p\|$. Due to inefficiency of **CLS-new**, α may be too small, so that $\|s\|$ goes to zeros. s with zero size is called a **null step**. If there have been too many null steps, **LMBOPT** cannot update the subspace information too many iterations, resulting a failure. To get rid of this weakness, **nullStep** is used, depending on the **CLS-new** is inefficient or not. If **CLS-new** is inefficient (`eff` $= 4$), the new point x^{new} is a multiple of the **current best point**. Otherwise, it is a multiple of the **current point** generated by **CLS-new**. Given a tiny tuning parameter `del`, x^{new} is adjusted by the factor of $1 -$ `del` and all of its zero components (if they exist) are replaced with `del` in both cases. Then it is projected into the feasible set **x**.

6.4 Starting point and master algorithm

6.4.1 projStartPoint – the starting point

Choosing the starting point too special may lead to inefficiencies. For example, consider minimizing the quadratic function $f(x) := (x_1-1)^2+\sum_{i=2}^{n}(x_i-x_{i-1})^2$ started from $x^0 = 0$. If a diagonal preconditioner is used, it is easy to see by induction that, for any method that chooses its search directions as linear combinations of the previously computed preconditioned gradients, the ith iteration point has zero in all coordinates $k > i$ and its gradient has zero in all coordinates $k > i+1$. Since the solution is the all-one vector, this implies that at least n iterations are needed to reduce the maximal error in components of x to below one.

Situations like this are likely to occur when both the Hessian and the starting point are sparse. To avoid this, **projStartPoint** moves a user-given starting point `x` slightly into the relative interior of the feasible domain.

6.4.2 getSuccess – a sufficient decrease in the function value

The goal of **getSuccess** is to test whether the sufficient descent condition (4.20) holds or not; only with the difference that the tuning parameter β is replaced by the other tuning parameter $\beta^{\texttt{CG}}$. The Goldstein quotient (4.19) is computed provided that all of the following hold:
• The regularized Krylov direction is descent, i.e., (6.35) is negative, but it is not zero.
• `nlocal` $\geq$ `nwait` or `nstuck` $\geq$ `nsmin`.

After the Goldstein quotient (4.19) is computed, the iteration will be successful if either line search is efficient, meaning the sufficient descent condition (4.20) with $\beta = \beta^{\texttt{CG}}$ holds, or there exists an improvement on the function value by at least δ_f and `nstuck` $\geq$ `nsmin`. In this case, the Boolean variable `success` is evaluated to be true; otherwise, it is evaluated to be false.

6.4.3 The master algorithm

We now recall the main ingredients for **LMBOPT**, the new **limited memory bound constrained optimization** method. The mathematical structure of **LMBOPT** is described in Section 1 of `suppMat.pdf`. **LMBOPT** first calls **projStartPoint** described in Subsection 6.4.1 to improve the starting point. Then the function value and gradient vector for such a point are computed and adjusted by **adjustGrad**; the same is done later in every such calculation. In practice, if the gradient is contaminated by NaN or $\pm\infty$, **adjustGrad** replaces these values by a tuning parameter. In the main loop,
• **LMBOPT** first computes the reduced gradient by **redGrad** in per iteration and then the working set is determined and updated by **findFreePos**.
• As long as the reduced gradient is not below a minimum threshold, it generates the direction $\widetilde{p}$ by **searchDir** to construct the subspace, and then constructs the regularized Krylov direction p by **KrylovDir** in the hope of achieving a successful iteration, provided that the activity is changed; otherwise the scaled Cauchy point is computed by **scaleCauchy** if the statements (1)-(4) hold, discussed earlier in Subsection 6.2.4. Such a successful iteration is determined by **getSuccess** and then the best point is updated.
• Otherwise it performs a gradient-free line search **CLS-new** along a regularized direction (**enforceAngle**) since the function is not near the quadratic case.
• Then if null steps are repeated at least `nnullmax` in a sequence, the point leading to such steps is replaced by **nullStep** with a point around the previous best point if **CLS-new** is not efficient; otherwise with the current point generated by **CLS-new**. This is repeated until no null step is found.
• Afterwards, the gradient at the new point is computed and adjusted by **adjustGrad**. In addition, the new free index set is found by **findFreeNeg**. At the end of every iteration, the subspace is updated provided that (i) there is no null step, (ii) either the condition (6.38) holds or the number of local steps exceeds its limit.

LMBOPT minimizes the bound constrained optimization problem (4.1). It takes the initial point x^0, the feasible set $\mathbf{x}$ and tuning parameters – detailed in Table 4 in `suppMat.pdf` – as input and returns an optimum point x^{best} and its function value f^{best} as output. For the convergence analysis of Algorithm 6.4.1 we refer to Theorem 4.5.2.

6.4.1 Algorithm. (**LMBOPT**, **limited memory bound-constrained optimization**)

(LMB_0) **Initialization**.
(1) Initialize the subspace information and other necessary information.
(2) Improve the starting point x^0 by **projStartPoint** and compute initial function value $f^0 := f(x^0)$ and its gradient $g^0 := g(x^0)$.
(3) Adjust the gradient by **adjustGrad**, and then initialize the necessary information by **initInfo**.
for $\ell = 0, 1, 2, \cdots$ **do**
(LMB_1) **Computing** $g^{\text{red}}(x^\ell)$ **and finding** $I_+(x^\ell)$**.** Compute the reduced gradient by **redGrad** and find the free indices by **findFreePos**.
(LMB_2) **Checking stopping tests.** If either the infinity norm of reduced gradient is below a given threshold or number of stuck iterations exceeds its limit, set $x^{\text{best}} = x^\ell$ and $f^{\text{best}} = f(x^\ell)$ and stop.
(LMB_3) **Computing the subspace direction.**
(1) Determine kind of the subspace by **typeSubspace**.
(2) Compute $\widetilde{p}^\ell$ by **searchDir**.
(3) Compute the regularized Krylov direction p^ℓ by **KrylovDir**.
(4) If the statements (1)-(4) hold (discussed earlier in Subsection 6.2.4) and the activity is fixed, a scaled Cauchy point by **scaleCauchy** is tried.
(5) Determine whether the iteration is successful (`success` is true) or not (`success` is false) by **getSuccess**.
(LMB_4) **A new trial point may be accepted.** If `success` is true, the $(\ell+1)$th iteration is successful and so set $x^{\ell+1} = x^\ell + p^\ell$; otherwise,
(1) regularize direction by **enforceAngle** and perform a line search along the regularized direction by **CLS-new** resulting in α^{new};
(2) project the trial point $x^\ell + \alpha^{\text{new}} p^\ell$ into $\mathbf{x}$, resulting in the accepted point $x^{\ell+1}$, and compute the step $s^{\ell+1} := x^{\ell+1} - x^\ell$;
(3) perform **nullStep** to check whether there exists a null step or not;
(4) if there is found a null step, increase the number of null steps and **LMBOPT** ends provided that maximum number of null steps is reached.
(LMB_5) **Computing the gradient at the new point.** If there is no null step, compute the gradient $g^{\ell+1} := g(x^{\ell+1})$, adjust it by **adjustGrad**, and set $y^{\ell+1} := g^{\ell+1} - g^\ell$.
(LMB_6) **Updating information, free indices, and subspace.** Update the information by **updateInfo**, find the new free indices set $I^{\ell+1} := I_-(x^{\ell+1})$ by **findFreeNeg**, update the subspace by **updateSubspace**.
end for

LMBOPT was implemented in Matlab; the source code is obtainable from

`http://www.mat.univie.ac.at/~neum/software/LMBOPT`.

6.5 Numerical results

In this section we compare our new solver **LMBOPT** with many other state-of-the-art solvers from the literature (see Subsection 6.6.2) on a large public benchmark. Only summary results are given; for supplementary information with much more detailed test results see `suppMat.pdf` from the **LMBOPT** web site. These solvers are

ASACG [89, 90, 91, 93],	**CGdescent** [89, 90, 93],	**ASABCP** [43],
SPG [23, 24],	**LBFGSB** [29],	**LMBFG-DDOGL** [28],
LMBFG-EIG-MS-2-2 [28],	**LMBFG-BWX-MS** [28],	**LMBFG-EIG-inf-2** [28],
LMBFGS-TR [28],	**LMBFG-MTBT** [28],	**LMBFG-MT** [28],
LMBFG-EIG-MS [28],	**LMBFG-EIG-curve-inf** [28].	

Details about the solvers and the options used can be found in Subsection 6.6.2. For some of the solvers we chose options different from the default to make them more competitive.

We only compare public software with an available Matlab interface. **LANCELOT-B** combines a trust region approach and projected gradient directions. But since there was no mex-file to run **LANCELOT-B** in Matlab, we could not call it to be run in our Matlab environment. Similarly, we could not find a version of **GENCAN**, the bound constrained version of **ALGENCAN** [22], which could be handled in Matlab. **GENCAN** is a combination of spectral projected gradient and an active set strategy. It is unlikely to introduce a significant bias into the comparison. Hence, we compare **LMBOPT** with many known solvers using various active set strategies and either projected conjugate gradient methods, projected truncated Newton methods, or projected quasi Newton methods, not including **LANCELOT-B** and **GENCAN**.

Unconstrained solvers were turned into bound-constrained solvers by pretending that the reduced gradient at the point $\pi[x]$ is the requested gradient at x. Therefore no theoretical analysis is available, the results show that **this is a simple and surprisingly effective strategy**.

6.5.1 Test problems used

We used all 1088 unconstrained and bound constrained problems with up to 100001 variables from `CUTEst`; see Subsection 6.6.4.

We limited the budget available for each solver by requiring

$$\texttt{nf2g} \leq \begin{cases} 20n + 10000 & \text{in the first and second runs,} \\ 50n + 200000 & \text{in the third run} \end{cases}$$

function evaluations plus two times gradient evaluations for a problem with n variables and

allowing at most

$$\begin{cases} 300 & \text{in the first run,} \\ 1800 & \text{in the second run,} \\ 7200 & \text{in the third run} \end{cases}$$

seconds of run time. A problem is considered solved by the solver *so* if infinity-norm of the gradient $\leq 10^{-6}$.

To identify the best solver under appropriate conditions on test problems and budgets, we made three different runs:
• In the first and second runs, the initial point is computed by (5.1). Compared to the standard starting point, this shift usually preserves the difficulty of the problem. In the second run, the three best solvers from the first run try to solve all test problems with an increased time limit of 1800 seconds.
• In the third run, the initial point x^0 is the standard starting point. The three best solvers from the first run try to solve the 98 test problems unsolved in the first run without the shift (5.1). Maximal time in seconds increased from 300 seconds to 7200 seconds and maximum number of `nf2g` increased from $20n + 10000$ to $50n + 200000$. In this case, the three best solvers from the first run succeeded to solve many of these unsolved problems. Test problems unsolved in third run could not be solved by any solver, even with a huge budget.

6.5.2 The results for stringent resources

Unconstrained and bound constrained optimization problems. We tested all 15 solvers for problems in dimension 1 up to 100001. A list of problems unsolved by all solvers can be found in Subsection 6.6.3.

As can be seen from Table 6.1 and Figure 6.4 (shown in Subsection 6.6.5), **LMBOPT** stands out as the most robust solver for unconstrained and bound constrained optimization problems; it is the best in terms of number of solved problems and the `ng` efficiency. Other best solvers in terms of the number of solved problems and the `nf2g` efficiency are **ASACG** and **LMBFG-EIG-MS**, respectively. **LBFGSB** is the best in terms of number of function evaluations #100 and !100, but it is not comparable in terms of the number of solved problems with other algorithms.

Classified by constraints and dimensions. Results for the three best solvers for all problems classified by dimension and constraint are given in Table 6.2 and Figures 6.5–6.6, Box plots 6.7–6.8 (shown in Subsection 6.6.5 and 6.6.6). These results show that,
• for **low-dimensional problems** ($1 \leq n \leq 30$), (1) **LMBOPT** is the best solver in terms of the `ng` and `nf2g` efficiencies and the number of solved problems, (2) **LMBFG-EIG-MS** is the best solver in terms of the `nf` efficiency (for both unconstrained and bound constrained problems), and (3) **ASACG** is the second best solver in terms of the number of solved problems (for both unconstrained and bound constrained problems);
• for **medium-dimensional problems** ($31 \leq n \leq 500$), (1) **LMBOPT** is the best in terms of the `ng` efficiency and the number of solved problems in the both unconstrained and bound constrained problems. It is the best is the best in terms of the `nf2g` efficiency in the unconstrained problems, (2) **LMBFG-EIG-MS** is the best in terms of `nf` in the both unconstrained and bound constrained problems and `nf2g` only in the bound constrained problems, (3) **ASACG** is the best solver in terms of the `nf2g` efficiency in the bound constrained problems;
• for **large-dimensional problems** ($501 \leq n \leq 100001$), (1) **LMBOPT** is the best solver in

Table 6.1: The summary results for all problems

stopping test:		$\|g\|_\infty \leq$ 1e-06,			sec $\leq$ 300,			nf2g $\leq 20 * n + 10000$			
990 of 1088 problems solved									mean efficiency in %		
dim∈[1,100001]						# of anomalies			for cost measure		
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	952	179	153	4310	87	49	0	59	70	43	13
ASACG	**asa**	935	164	28	1416	98	21	34	58	60	51	62
LMBFG-EIG-MS	**lt6**	924	103	45	2970	119	26	19	60	57	60	34
LMBFG-EIG-curve-inf	**lt4**	918	94	35	3330	118	25	27	60	56	59	34
ASABCP	**asb**	900	75	52	2404	142	25	21	41	36	44	46
LMBFG-DDOGL	**lt2**	896	112	49	2937	61	21	110	60	56	59	33
CGdescent	**cgd**	895	144	16	2559	77	17	99	54	56	47	55
LMBFG-EIG-MS-2-2	**lt7**	895	38	0	3390	112	21	60	50	45	57	34
LMBFG-BWX-MS	**lt1**	888	39	1	2694	56	21	123	51	45	58	32
SPG	**spg**	840	94	60	5901	182	58	8	34	34	31	9
LBFGSB	**lbf**	803	233	186	713	0	0	285	57	51	61	32
LMBFG-EIG-inf-2	**lt5**	753	81	23	3275	76	26	233	50	47	49	28
LMBFGS-TR	**ll3**	733	100	41	2904	242	92	21	48	44	48	36
LMBFG-MTBT	**ll2**	669	76	23	2257	55	14	350	45	41	46	26
LMBFG-MT	**ll1**	657	104	50	2677	57	14	360	45	39	48	32

terms of the ng efficiency for both unconstrained and bound constrained problems, (2) **LMBFG-EIG-MS** is the best solver in terms of the nf and nf2g efficiencies and the number of solved problems (for all problems) and is the best solver in terms of the number of solved problems (for bound constrained problems), (3) **ASACG** is the best solver in terms of the number of solved problems only in the unconstrained problems.

• for all problems ($1 \leq n \leq 100001$), (1) **LMBOPT** is the best in terms of the number of solved probelms and the ng efficiency in both unconstrained and bound constrained problems, (2) **LMBFG-EIG-MS** is the best solver in terms of the nf and nf2g efficiencies in both unconstrained and bound constrained problems.

Table 6.2: The summary results classified by dimension and constraint for all problems

stopping test:		$\|g\|_\infty \leq$ 1e-06,			sec $\leq$ 1800,			nf2g $\leq 20 * n + 10000$			
304 of 319 problems solved								mean efficiency in %			
dim∈[1,30]						# of anomalies			for cost measure		
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	303	104	85	198	16	0	0	74	82	60	21
ASACG	**asa**	285	116	90	28	20	0	14	72	70	66	81
LMBFG-EIG-MS	**lt6**	274	120	102	185	43	0	2	66	58	70	55
182 of 192 problems without bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	181	51	50	214	11	0	0	75	83	59	16
ASACG	**asa**	175	57	50	31	9	0	8	74	73	67	84
LMBFG-EIG-MS	**lt6**	166	81	75	259	26	0	0	72	62	77	57
122 of 127 problems with bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	122	53	35	173	5	0	0	73	79	62	28
ASACG	**asa**	110	59	40	23	11	0	6	68	64	66	78
LMBFG-EIG-MS	**lt6**	108	39	27	71	17	0	2	57	53	59	53
304 of 331 problems solved								mean efficiency in %			
dim∈[31,500]						# of anomalies			for cost measure		
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	299	93	88	482	32	0	0	70	79	55	18
ASACG	**asa**	293	89	80	154	28	0	10	69	68	64	83
LMBFG-EIG-MS	**lt6**	293	136	127	227	31	0	7	71	64	75	52
189 of 203 problems without bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	187	62	62	581	16	0	0	74	82	57	20
ASACG	**asa**	183	45	42	177	16	0	4	69	69	63	86
LMBFG-EIG-MS	**lt6**	183	85	82	266	18	0	2	73	63	78	56
115 of 128 problems with bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	112	31	26	317	16	0	0	65	75	51	16
ASACG	**asa**	110	44	38	114	12	0	6	68	67	67	79
LMBFG-EIG-MS	**lt6**	110	51	45	163	13	0	5	68	66	70	47
375 of 438 problems solved								mean efficiency in %			
dim∈[501,100001]						# of anomalies			for cost measure		
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBFG-EIG-MS	**lt6**	365	240	227	15202	60	2	11	73	65	76	38
ASACG	**asa**	358	76	63	5434	68	1	11	60	59	57	80
LMBOPT	**lmb**	354	81	71	17386	69	15	0	61	71	46	19
181 of 220 problems without bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
ASACG	**asa**	175	40	37	4021	42	1	2	62	62	57	78
LMBFG-EIG-MS	**lt6**	173	100	96	10696	43	2	2	65	57	69	32
LMBOPT	**lmb**	169	45	44	15044	40	11	0	58	65	44	15
194 of 218 problems with bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBFG-EIG-MS	**lt6**	192	140	131	19261	17	0	9	80	73	82	45
LMBOPT	**lmb**	185	36	27	19525	29	4	0	65	76	48	23
ASACG	**asa**	183	36	26	6786	26	0	9	58	55	57	82
983 of 1088 problems solved								mean efficiency in %			
dim∈[1,100001]						# of anomalies			for cost measure		
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	956	278	244	6651	117	15	0	68	76	53	19
ASACG	**asa**	936	281	233	2135	116	1	35	66	65	62	81
LMBFG-EIG-MS	**lt6**	932	496	456	6079	134	2	20	70	63	74	48
552 of 615 problems without bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	537	158	156	5009	67	11	0	68	75	53	17
ASACG	**asa**	533	142	129	1391	67	1	14	68	67	62	82
LMBFG-EIG-MS	**lt6**	522	266	253	3721	87	2	4	70	60	74	47
431 of 473 problems with bounds solved								mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf2g	ng	nf	msec
LMBOPT	**lmb**	419	120	88	8756	50	4	0	67	76	52	23
LMBFG-EIG-MS	**lt6**	410	230	203	9082	47	0	16	71	64	73	48
ASACG	**asa**	403	139	104	3119	49	0	21	64	60	62	80

6.5.3 Results for hard problems

All solvers have been run again on the hard problems defined as the 98 test problems unsolved in the first run. In this case, the standard starting point has been used instead of (5.1) and both `nfmax` and `secmax` have been increased. 41 test problems were not solved by all solvers for dimensions 1 up to 100001, given in Table 6.5 in Subsection 6.6.8

From Table 6.3 and Figure 6.9 (shown in Subsection 6.6.5), we conclude
• **LMBOPT** is the best in terms of the number of solved problems and the `ng` and `nf2g` efficiencies in the hard bound constrained problems.
• **ASACG** is the best in terms of the number of solved problems and the `ng` and `nf2g` efficiencies in the hard unconstrained problems.
• **LMBFG-EIG-MS** is the best in terms of the `ng` and `nf2g` efficiencies in the hard unconstrained problems.

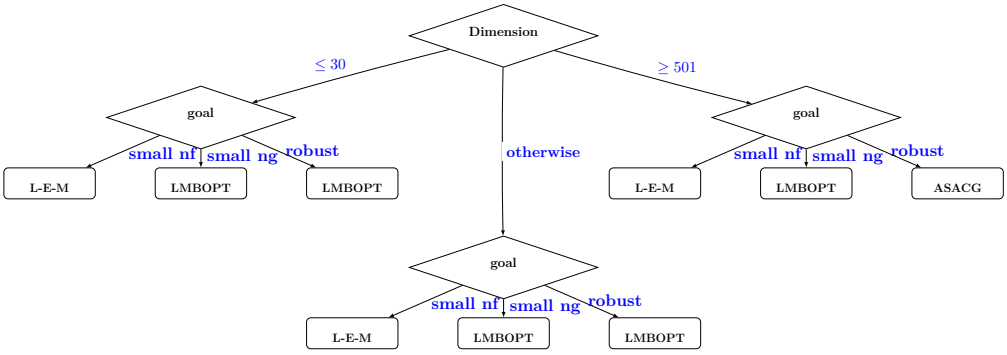
Table 6.3: The summary results for hard problems

stopping test:		$\|g\|_\infty \leq$ `1e-06`,			`sec` $\leq$ `7200`,			`nf2g` $\leq$ `50` $*$ `n` $+$ `200000`				
57 of 98 problems solved									mean efficiency in %			
dim∈[1,100001]						# of anomalies			for cost measure			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	`nf2g`	`ng`	`nf`	`msec`
LMBOPT	**lmb**	50	13	13	224127	44	4	0	36	41	28	15
ASACG	**asa**	50	20	20	104569	31	0	17	42	42	39	50
LMBFG-EIG-MS	**lt6**	46	24	24	157855	41	1	10	39	35	41	25
28 of 57 problems without bounds solved									mean efficiency in %			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	`nf2g`	`ng`	`nf`	`msec`
ASACG	**asa**	26	7	7	170316	22	0	9	38	35	35	44
LMBFG-EIG-MS	**lt6**	24	15	15	247233	26	1	6	38	35	39	21
LMBOPT	**lmb**	21	6	6	247561	33	3	0	27	31	20	11
stopping test:		$\|g\|_\infty \leq$ `1e-06`,			`sec` $\leq$ `7200`,			`nf2g` $\leq$ `50` $*$ `n` $+$ `200000`				
29 of 41 problems with bounds solved									mean efficiency in %			
dim∈[1,100001]						# of anomalies			for cost measure			
solver		solved	#100	!100	T_{mean}	#n	#t	#f	`nf2g`	`ng`	`nf`	`msec`
LMBOPT	**lmb**	29	7	7	207158	11	1	0	48	54	38	21
ASACG	**asa**	24	13	13	33344	9	0	8	47	46	45	58
LMBFG-EIG-MS	**lt6**	22	9	9	60353	15	0	4	40	36	45	31

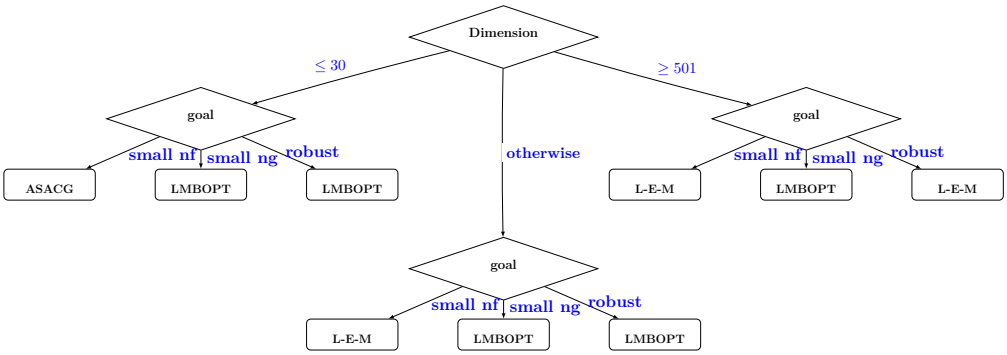
6.5.4 Recommendations

In this section, we recommend a solver choice based on our finding. The choice depends on the problem dimension, the presence and absence of constrains, the desired robustness, and the relative cost of function and gradient evaluations shown in Subfigures (a)-(c) of Figure 6.2.

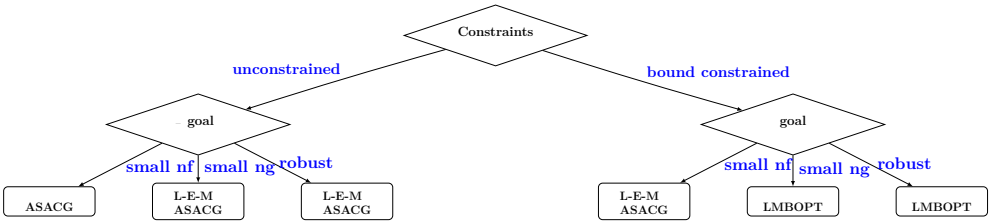
Figure 6.2: (a) Flow chart for unconstrained problems classified by problems, (b) Flow chart for bound constrained problems classified by the problem dimension, (c) Flow chart for hard problems classified by constraint. Here **L-E-M** stands for **LMBFG-EIG-MS**.



(a)



(b)



(c)

6.6 Additional material for LMBOPT

This section discusses additional material for **LMBOPT**.

6.6.1 Default tuning parameters for LMBOPT

For our tests we used for **LMBOPT** the following tuning parameters

`nsmin` $= 1$;	`nwait` $= 1$;	`rfac` $= 2.5$;	`nlf` $= 2$;	$\Delta^m = 10^{-13}$;	$\Delta^{pg} = \varepsilon_m$;
$\Delta^\nu = \varepsilon_m$;	$\Delta^\alpha = 5\varepsilon_m$;	$l^{\max} = 4$;	$\beta = 0.02$;	$\beta^{\texttt{CG}} = 0.001$;	$\Delta^{\mathrm{a}} = 10^{-12}$;
$\Delta^{reg} = 10^{-12}$;	$\Delta^w = \varepsilon_m$;	`facf` $= 10^{-8}$;	$\Delta^x = 10^{-20}$;	$m = 12$;	`mf` $= 2$;
`typeH` $= 0$;	`nnulmax` $= 3$;	`del` $= 10^{-10}$;	$\Delta^r = 20$;	$\Delta^g = 100$;	$\Delta^b = 10\varepsilon_m$;
$\Delta^u = 1000$;	$\theta = 10^{-8}$;	`exact` $= 0$;	$\Delta^{po} = \varepsilon_m$	`nstuckmax` $= +\infty$;	$\zeta^{\min} = -10^{50}$;
$\zeta^{\max} = -10^{-50}$;	$\Delta^D = 10^{10}$;	$q^{\min} = 2.5$;	$\Delta^q = 10$;	$\Delta_f = 2$;	$q = 25$;
`mdf` $= 20$.					

They are based on limited tuning by hand. How to find optimal tuning parameters [116] would be interesting and very important since the quality of **LMBOPT** depends on it.

6.6.2 Codes compared

We compare **LMBOPT** with the following solvers for unconstrained and bound constrained optimization. For some of the solvers we chose options different from the default to make them more competitive.

Bound constrained solvers:
• **ASACG** (asa), obtained from

`http://users.clas.ufl.edu/hager/papers/CG/Archive/ASA_CG-3.0.tar.gz`,

is an active set algorithm for solving a bound constrained optimization problem by Hager & Zhang [92]. The default parameters have been used. Only `memory` $= 12$ and other parameters have been chosen as default.
• **LBFGSB** (lbf), obtained from

`http://users.iems.northwestern.edu/~nocedal/Software/Lbfgsb.3.0.tar.gz`,

is a limited-memory quasi-Newton code for bound-constrained optimization by Byrd et al. [29]. Only $m = 12$ and other parameters have been chosen as default.
• **ASABCP** (asb), obtained from

`https://sites.google.com/a/dis.uniroma1.it/asa-bcp/download`,

is a two-stage active-set algorithm for bound-constrained optimization by Cristofari et al. [43]. The default parameters have been used.
• **SPG** (spg), obtained from

`https://www.ime.usp.br/~egbirgin/tango/codes.php`,

is a spectral projected gradient algorithm for solving a bound constrained optimization problem by BIRGIN et al. [23, 24]. The default parameters have been used.

Unconstrained solvers:
• **CGdescent** (cdg), obtained from

`http://users.clas.ufl.edu/hager/papers/CG/Archive/CG_DESCENT-C-6.8.tar.gz`,

is a conjugate gradient algorithm for solving an unconstrained minimization problem by HAGER & ZHANG [90, 91, 93, 94]. Only `memory` $= 12$ and other parameters have been chosen as default.
• **LMBFG**, obtained from

`http://gratton.perso.enseeiht.fr/LBFGS/index.html`,

is a limited memory quasi Newton package by BURDAKOV et al. [28]:
(a) **LMBFG-MT** (**ll1**) is a limited memory line-search algorithm **L-BFGS** based on the MORE-THUENTE line search. Only $m = 12$ and other parameters have been chosen as default.
(b) **LMBFG-MTBT** (**ll2**) is a limited memory line-search algorithm **L-BFGS** based on the MORE-THUENTE line search and the starting step is obtained using backtrack by BURDAKOV et al. [28]. Only $m = 12$ and other parameters have been chosen as default.
(c) **LMBFGS-TR** (**ll3**), is a limited memory line-search algorithm **L-BFGS** that takes a trial step along the quasi-Newton direction inside the trust region. Only $m = 12$ and the other parameters have been chosen as default.
(d) **LMBFG-BWX-MS** (**lt1**) is a limited memory trust-region algorithm BWX-MS. It applies the MORÉ & SORENSEN approach for solving the TR subproblem defined in the Euclidean norm. Only $m = 12$ and the other parameters have been chosen as default.
(e) **LMBFG-DDOGL** (**lt2**) is a limited memory trust-region algorithm **D-DOGL**. Only $m = 12$ and the other parameters have been chosen as default.
(f) **LMBFG-EIG-curve-inf** (**lt4**) is a limited memory trust-region algorithm `EIG`$(\infty, 2)$. Only $m = 12$ and the other parameters have been chosen as default.
(g) **LMBFG-EIG-inf-2** (**lt5**) is a limited memory trust-region algorithm `EIG`$(\infty, 2)$ based on the eigenvalue-based norm, with the exact solution to the TR subproblem in closed form. Only $m = 12$ and other parameters have been chosen as default.
(h) **LMBFG-EIG-MS** (**lt6**) is a limited memory trust-region algorithm **EIG-MS**. Only $m = 12$ and other parameters have been chosen as default.
(i) **LMBFG-EIG-MS-2-2** (**ll7**) is a limited memory trust-region algorithm `EIG` $-$ `MS`$(2, 2)$ based on the eigenvalue-based norm, with the MORÉ & SORENSEN approach for solving a low-dimensional TR subproblem. Only $m = 12$ and other parameters have been chosen as default.

6.6.3 Problems unsolved by all solvers

A list of problems unsolved by all solvers is given in Table 6.4.

Table 6.4: Problems unsolved by all solvers

BROWNBS	PALMER5E	PALMER5B	OSCIGRAD:10
OSCIPATH:10	STRATEC	SBRYBND:10	SCOSINE:10
SCURLY10:10	SCOND1LS	OSCIGRAD:15	OSCIGRAD:25
ANTWERP	NONMSQRT:49	HS110:50	SBRYBND:50
RAYBENDS	RAYBENDL:66	RAYBENDS:66	HYDC20LS
FLETCHBV:100	HS110:100	NONMSQRT:100	OSCIGRAD:100
SBRYBND:100	SCOSINE:100	SCURLY10:100	SSCOSINE:100
SCOND1LS:102	RAYBENDL:130	RAYBENDS:130	QR3DLS
GRIDGENA:170	DRCAV1LQ	HS110:200	SPMSRTLS:499
PENALTY2:500	SBRYBND:500	SCOND1LS:502	MSQRTALS:529
MSQRTBLS:529	NONMSQRT:529	GRIDGENA	QR3DLS:610
LINVERSE:999	CURLY20	CHENHARK	FLETCHBV:1000
PENALTY2:1000	SBRYBND	SCOSINE	SCURLY10
SSCOSINE	SPMSRTLS:1000	SCOND1LS:1002	MSQRTALS:1024
MSQRTBLS:1024	NONMSQRT:1024	RAYBENDL:1026	RAYBENDS:1026
DRCAV1LQ:1225	DRCAV2LQ:1225	DRCAV3LQ:1225	GRIDGENA:1226
LINVERSE:1999	RAYBENDL:2050	RAYBENDS:2050	GRIDGENA:2114
EIGENALS:2550	GRIDGENA:3242	DRCAV3LQ:4489	GRIDGENA:4610
MSQRTALS:4900	MSQRTBLS:4900	SPMSRTLS:4999	FLETCBV3:5000
FLETCHBV:5000	SBRYBND:5000	SCOSINE:5000	SPARSINE:5000
SSCOSINE:5000	SCOND1LS:5002	BRATU1D:5003	GRIDGENA:6218
CURLY10:10000	CURLY20:10000	CURLY30:10000	FLETCBV3:10000
FLETCHBV:10000	SCOSINE:10000	SCURLY10:10000	SPARSINE:10000
SPMSRTLS:10000	SSCOSINE:10000	DRCAV3LQ:10816	ODNAMUR
GRIDGENA:12482	SSCOSINE:100000		

6.6.4 Test problem selection

It is seen from Figure 6.3 that the number of unconstrained, bound constrained, and unconstrained and bound constrained optimization problems – solved at least by one of solvers – are 517, 375, and 990 respectively.

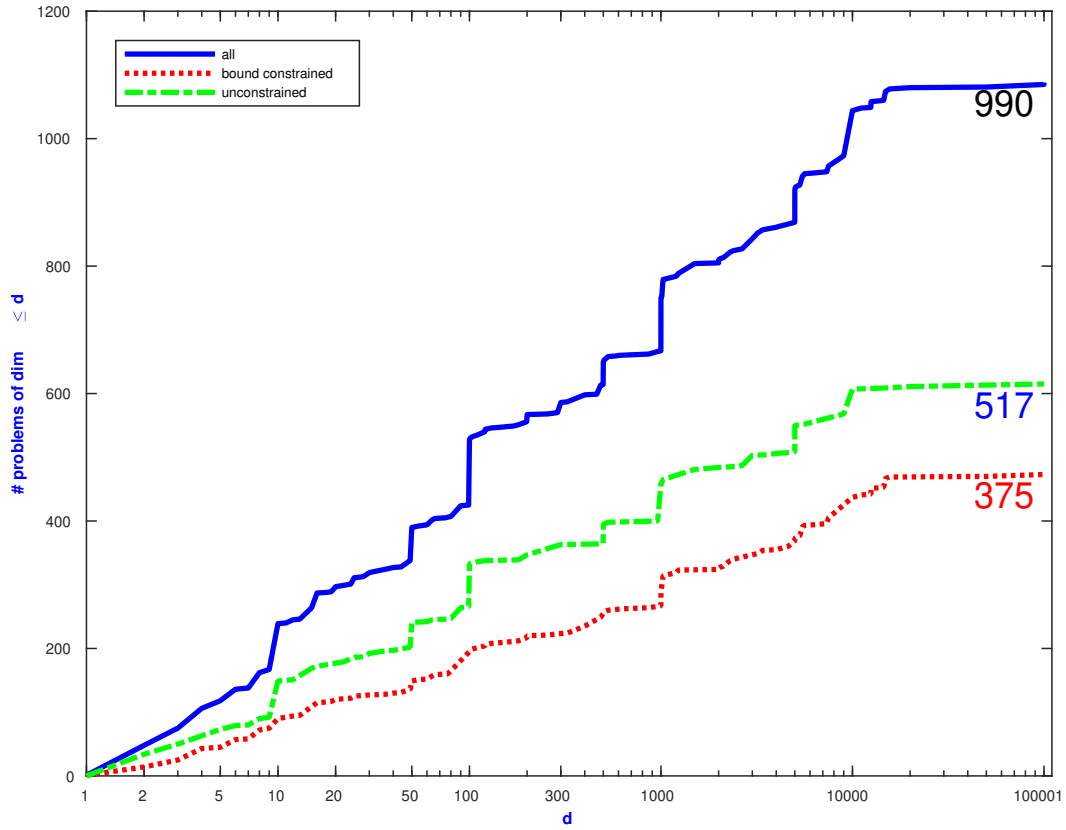


Figure 6.3: The number of problems with variables in a given range solved by at least one solver: 990 problems with dimensions 1 up to 100001

6.6.5 Performance profiles

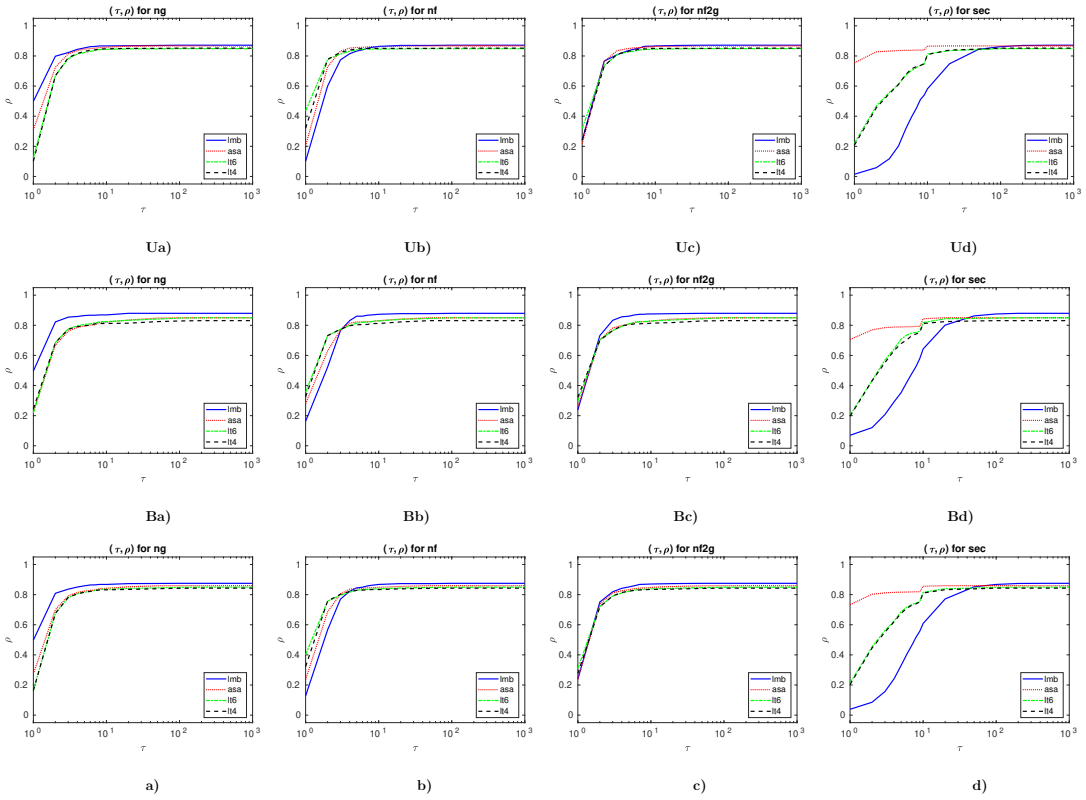


Figure 6.4: (Ua)-(Ud): Performance profiles for unconstrained problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. (Ba)-(Bd): Performance profiles for bound constrained problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. (a)-(d): Performance profiles for both unconstrained and bound constrained problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

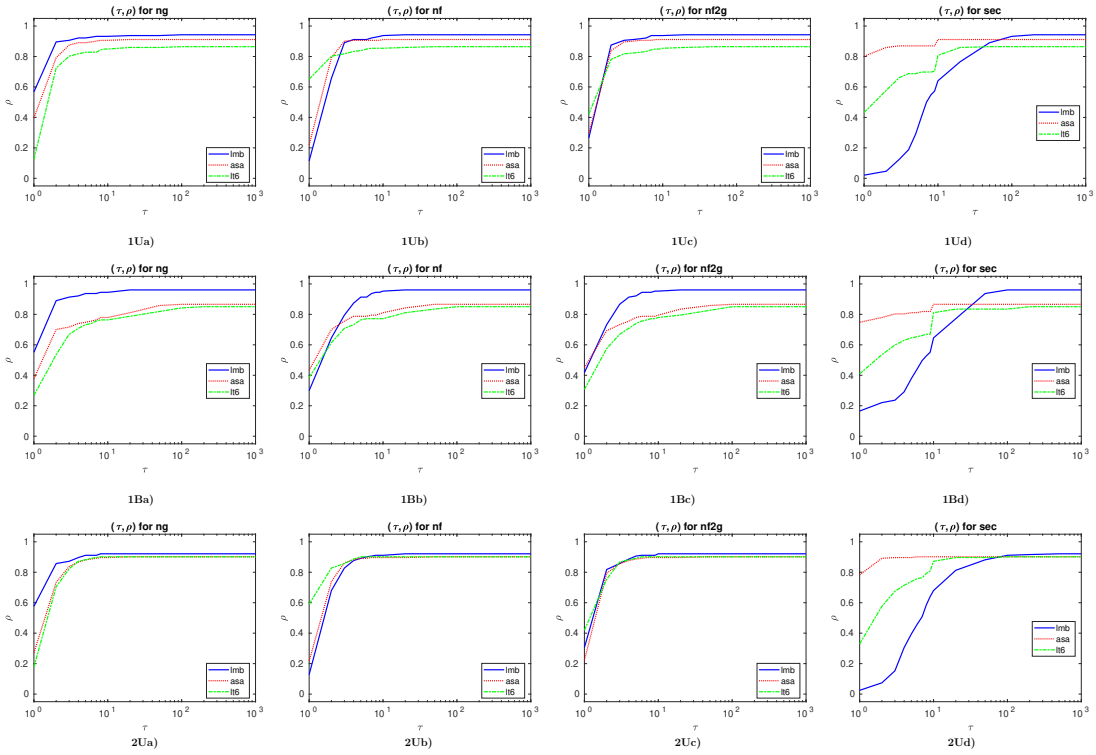


Figure 6.5: (1Ua)-(1Ud)/(1Ba)-(1Bd): Performance profiles for low-dimensional unconstrained/bound constrained problems ($1 \leq n \leq 30$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. (2Ua)-(2Ud): Performance profiles for medium-dimensional unconstrained problems ($31 \leq n \leq 500$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

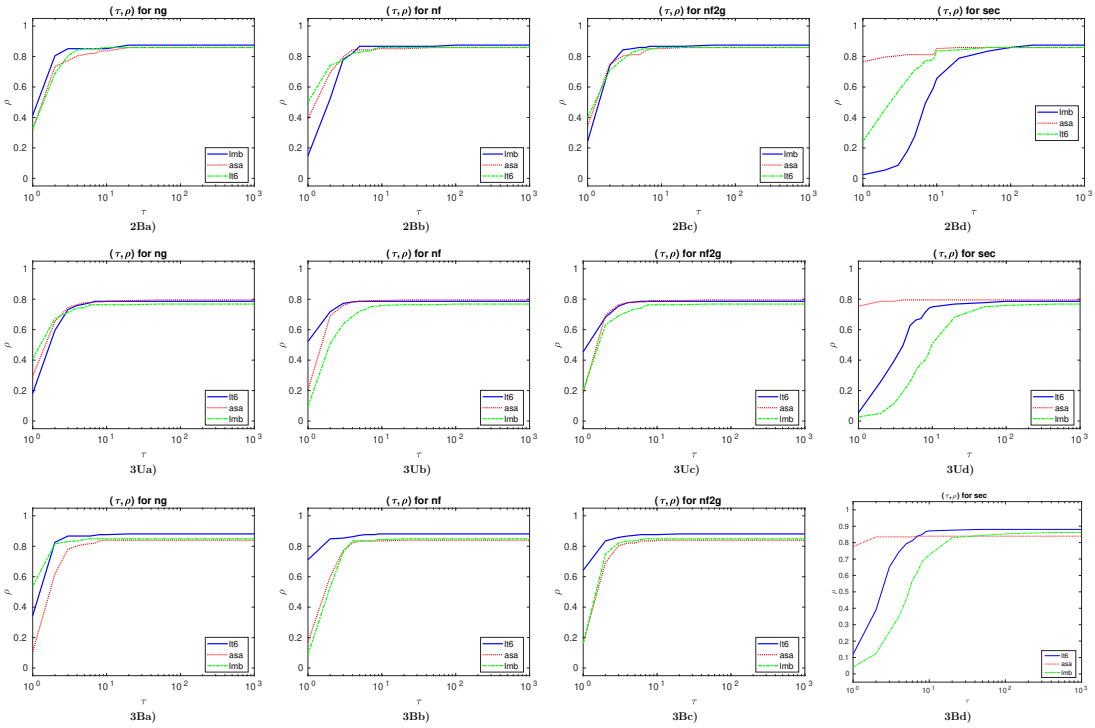


Figure 6.6: (2Ba)-(2Bd): Performance profiles for medium-dimensional bound constrained problems ($31 \leq n \leq 500$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. (3Ua)-(3Ud)/(3Ba)-(3Bd): Performance profiles for high-dimensional unconstrained and bound constrained problems ($501 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

6.6.6 Box plots

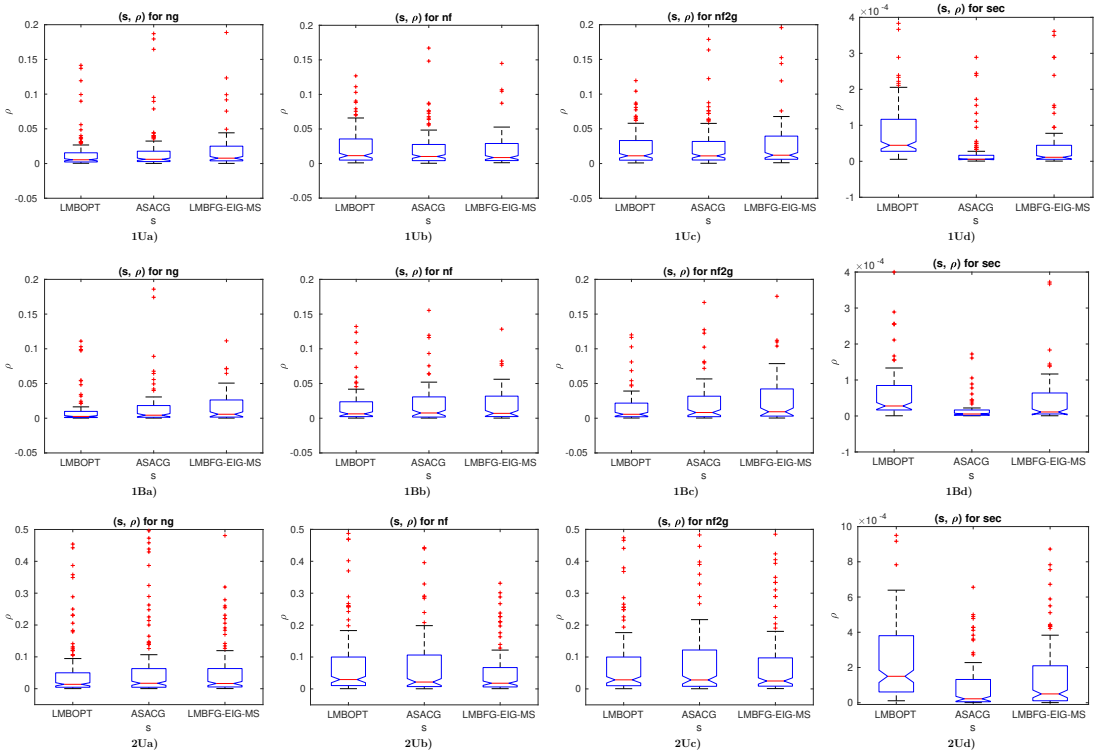


Figure 6.7: We show box plots for the data summarized in Table 6.2. Here ρ stands for `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, `sec`/`secmax` and s stands for the name of solvers. (1Ua)-(1Ud)/(1Ba)-(1Bd): Box plots for low-dimensional unconstrained and bound constrained problems ($1 \leq n \leq 30$) in terms of `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, and `sec`/`secmax`, respectively. (2Ua)-(2Ud): Box plots for medium-dimensional unconstrained problems ($31 \leq n \leq 500$) in terms of `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, and `sec`/`secmax`, respectively. Here `nfmax`, `ngmax`, `nf2gmax`, and `secmax` stand for maximal number of function evaluations, maximal number of gradient evaluations, maximal number of function evaluations plus two times gradient evaluations, and maximal time in seconds, respectively.

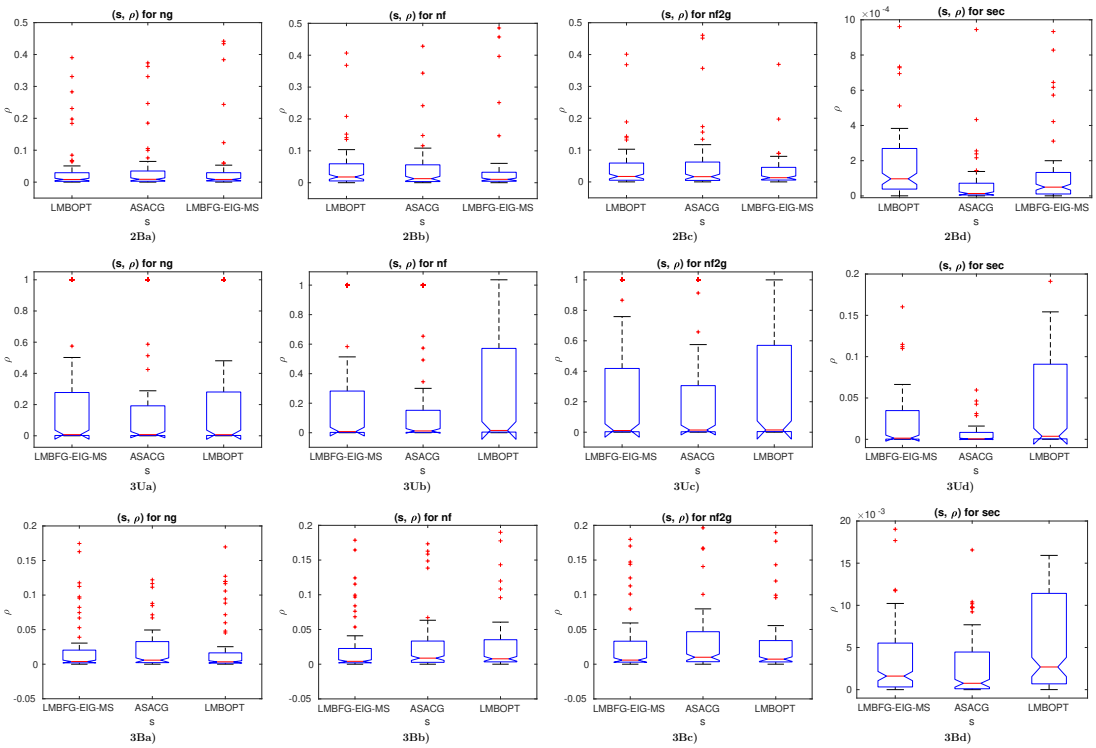


Figure 6.8: We show box plots for the data summarized in Table 6.2. Here ρ stands for `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, `sec`/`secmax` and s stands for the name of solvers. (2Ba)-(2Bd): Box plots for medium-dimensional bound constrained problems ($31 \leq n \leq 500$) in terms of `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, and `sec`/`secmax`, respectively. (3Ua)-(3Ud)/(3Ba)-(3Bd): Box plots for high-dimensional unconstrained and bound constrained problems ($501 \leq n \leq 100001$) in terms of `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, and `sec`/`secmax`, respectively. Here `nfmax`, `ngmax`, `nf2gmax`, and `secmax` stand for maximal number of function evaluations, maximal number of gradient evaluations, maximal number of function evaluations plus two times gradient evaluations, and maximal time in seconds, respectively.

6.6.7 The hard problems unsolved by all solvers

Table 6.5: The hard problems unsolved by all solvers

OSCIPATH:10	SCOSINE:10	SCOND1LS	ANTWERP
NONMSQRT:49	SBRYBND:50	HYDC20LS	FLETCHBV:100
NONMSQRT:100	SBRYBND:100	SCOSINE:100	SCURLY10:100
SCOND1LS:102	PENALTY2:500	SBRYBND:500	SCOND1LS:502
NONMSQRT:529	FLETCHBV:1000	PENALTY2:1000	SBRYBND
SCOSINE	SCURLY10	SSCOSINE	SCOND1LS:1002
NONMSQRT:1024	DRCAV1LQ:1225	DRCAV2LQ:1225	DRCAV3LQ:1225
DRCAV3LQ:4489	FLETCBV3:5000	FLETCHBV:5000	SBRYBND:5000
SCOSINE:5000	SCOND1LS:5002	BRATU1D:5003	FLETCBV3:10000
FLETCHBV:10000	SCOSINE:10000	SCURLY10:10000	DRCAV3LQ:10816
SSCOSINE:100000			

6.6.8 Performance profile for the hard problems

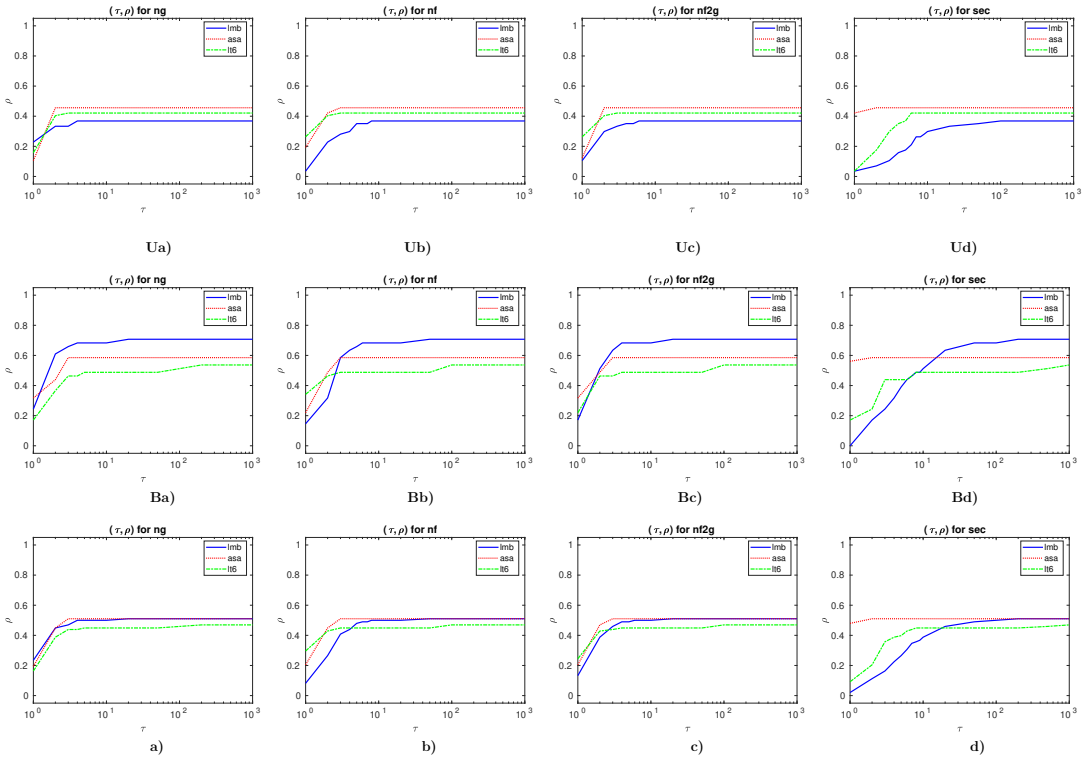


Figure 6.9: (Ua)-(Ud): Performance profiles for unconstrained hard problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. (Ba)-(Bd): Performance profiles for bound constrained hard problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. (a)-(d): Performance profiles for both unconstrained and bound constrained hard problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `msec`/(`best msec`) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

7 An active set trust region method

This section discusses an active set trust region method for bound constrained optimization problems. This is my own work (cf. Kimiaei [111]). A sufficient descent condition is used as a computational measure to identify whether the function value is reduced or not. A complexity result is proven for a critical measure which is computationally better than the other known critical measures. Under the positive semi-definiteness of approximated Hessian matrices restricted to the subspace of non-active variables, it will be shown that unlimited zigzagging cannot occur.

7.1 Overview of the new method

We construct an active set trust region algorithm for bound constrained optimization problems:
• The trust region ratio (4.22) is replaced by a variant of the sufficient descent condition (4.20). It is very useful in finite precision arithmetic whenever the function is very nonlinear.
• To get a complexity bound for our algorithm, the trust region radius is updated based on the reduced gradient.
• The reduced gradient is used as a critical measure to identify a good local minimizer from a spurious apparent local minimizer.
• Under the positive semi-definiteness of approximated Hessian matrices restricted to the subspace of free variables, it will be shown that unlimited zigzagging cannot occur. Hence all strongly active variables are found and fixed at finitely many iterations.

Section 7.2 describes the required assumptions on the problem and algorithm to prove an improved complexity result for bound constrained optimization. In Section 7.3, a sufficient descent condition is used instead of the trust region ratio. In Section 7.4, the new algorithm is described. The complexity result is proven in Section 7.5.

7.2 Complexity

In this section, we discuss the state of the art complexity results in Subsection 7.2.1 and our complexity results with a reasonable critical measure in Subsection 7.2.2.

7.2.1 Known complexity

This subsection discusses why the two known critical measures may accept numerically points as local minimizers while they are far from the local minimizers.

Two common critical measures for the problem (4.1) are

$$g^{(1)}(x) := \pi(x - g(x)) - x \quad \text{and} \quad \chi(x) := \min_{x+p\in\mathbf{x},\ \|p\|\leq 1} g(x)^T p.$$

Here the definition of π comes from (4.18). They were used to prove the global convergence by Hager & Zhang [90, 94] and Byrd et al. [29] and the complexity bound by Cartis et al. [34, 33].

Cartis et al. [34] showed that their algorithm, starting from a point $x^0 \in \mathbf{x}$, finds a point $x \in \mathbf{x}$ such that $\|g^{(1)}(x)\| \leq \varepsilon$ using at most $N(\varepsilon)$ iterations. For the one-dimensional problem [136]

$$\min_{0\leq x\leq\infty} x, \tag{7.1}$$

whose gradient of the objective function is $g(x) = 1$, the cancellation in the calculation of $g^{(1)}(x)$ in double precision arithmetic leads to the acceptance of the point $x = 10^{17}$ as the minimizer while such a point is far from the minimizer.

Cartis et al. [33] used another critical measure $|\chi(x)| \leq \varepsilon$. For the problem (7.1), this measure is $\chi(x) := -1$ and this critical measure fails since ε is assumed to be small.

7.2.2 Our complexity

This subsection introduces our reasonable critical measure and assumptions on the problem and algorithm which are needed to achieve our complexity bound in the nonconvex case.

To overcome the shortcoming of known critical measures, we are interested in finding an algorithm which needs at most $N(\varepsilon)$ iterations to get a point x^{best} satisfying

$$f(x^{\text{best}}) \leq \sup_{x\in\mathbf{x}}\{f(x) \mid f(x) \leq f(x^0) \ \text{ and } \|g^{\text{red}}(x)\|_\infty \leq \varepsilon\}.$$

It is clear that this measure overcomes the drawbacks of two previous critical measures since if it is used for the problem (7.1) it results in $g^{\text{red}}(0) = 0$ and $g^{\text{red}}(x) = 1$ for $x > 0$. But the reduced gradient does not preserve continuity. Hence, Neumaier & Azmi [136, Theorem 8.2] showed that if the sequence x^ℓ ($\ell \geq 0$) generated by **BOPT** converges to a point $\widehat{x}$ and $g^{\text{red}}(x^\ell)$ goes to zero then $g^{\text{red}}(\widehat{x}) = 0$ and all strongly active variables are found ultimately at the nondegenerate stationary point $\widehat{x}$. As a result, if $g^{\text{red}}(x^\ell)$ goes to zero for a bounded sequence $x^\ell \in \mathbf{x}$, there are some subsequences converging to $\widehat{x} \in \mathbf{x}$ such that $g^{\text{red}}(\widehat{x}) = 0$ (see [136, Corrolary 8.3]). Accordingly, we prove the main convergence result in Section 7.5 (see Theorem 7.5.7).

We prove that the complexity bound $\mathcal{O}(\varepsilon^{-2})$ holds in both unconstrained and bound constrained cases. In the unconstrained case, this complexity bound is the same as the known complexity bounds by Curtis [45, 46], Grapiglia et al. [81], and Gratton et al. [83]. In the bound constrained case, our complexity bound is the same as the known complexity results obtained by Cartis et al. [34, 33] provided that the trust region algorithms use the quadratic model but with the difference that it is numerically better than them.

The complexity results for the other optimization methods have been obtained, e.g., Nesterov [133], Cartis et al. [35], Birgin et al. [19] and Nesterov & Polyak [134].

As in [41], we need to make some assumptions on the problem and the algorithm in order to determine the complexity results. We abbreviate
• the upper bound on the norm of the model's Hessian to "umh",
• the upper bound on the norm of function's element Hessians to "ufh",
• the uniform norm equivalence to "une",
• the upper bound on the Hessians to "ubh".

Assumptions on the problem:
(GBO$_1$) The objective function $f(x)$ is twice continuously differentiable and has a lower bound on the level set $L(x^0) := \{x \in \mathbf{x} \mid f(x) \le f(x^0),\ x^0 \in \mathbf{x}\}$ of x^0.
(GBO$_2$) The Hessian G of the objective function is uniformly bounded, i.e., there exists a $\Gamma^{\rm ufh} > 0$ such that $\|G(x)\| \le \Gamma^{\rm ufh}$ for $x \in \mathbf{x}$.
Assumptions on the algorithm:
(GBO$_3$) The model function $\mathcal{Q}$ is twice differentiable on $\mathcal{T}^r$, $\mathcal{Q}(x) = f(x)$, $g(x) = \nabla f(x) = \nabla \mathcal{Q}(x)$, and $B(x) := \nabla^2 \mathcal{Q}(x)$ for all $x \in \mathbf{x}$.
(GBO$_4$) The Hessian of the model function $B(x)$ is bounded within the trust region $\mathcal{T}^r$, i.e., there exists a constant $\Gamma^{\rm umh} \ge 1$ such that $\|B(x)\| \le \Gamma^{\rm umh} - 1$ for $x \in \mathcal{T}^r$.

By (GBO$_4$) and the definition $\overline{\Gamma} := 1 + \max_{x \in \mathcal{T}^r}\{\|B(x)\|\} \ge 1$ an upper bound on B is $\overline{\Gamma} \le \Gamma^{\rm umh}$.

The following result is Proposition 10.1 in [136].

7.2.1 Proposition. *For nonzero q and $\alpha > 0$, $p_q(\alpha) := \dfrac{\pi[x + \alpha q] - x}{\alpha}$ satisfies (in any monotone norm)*

$$|p_q(\alpha)| \le |q|, \quad \|p_q(\alpha)\| \le \|q\|,$$

and with $p \in \mathbb{R}^n$ defined by

$$p_i := \begin{cases} 0 & \text{if } \underline{x}_i = x_i = \overline{x}_i, \\ \max(0, q_i) & \text{if } \underline{x}_i = x_i < \underline{x}_i, \\ \min(q_i, 0) & \text{if } \underline{x}_i < x_i = \overline{x}_i, \\ q_i & \text{if } \underline{x}_i < x_i < \overline{x}_i, \end{cases} \tag{7.2}$$

we have

$$p_q(\alpha) = p \quad \text{for sufficiently small } \alpha > 0. \tag{7.3}$$

7.2.2 Proposition. *The three following statements hold:*
(i) For $\alpha \in (0, 1)$

$$\|g^{(\alpha)}(x^\ell)\| \le \alpha \|g^{(1)}(x^\ell)\|. \tag{7.4}$$

(ii) For sufficiently small $\alpha \in (0, 1)$

$$g^{(\alpha)}(x^\ell) = \alpha p_{-g(x^\ell)}(\alpha) = -\alpha g^{\rm red}(x^\ell). \tag{7.5}$$

(iii) $\|g^{\rm red}(x^\ell)\| \le \|g^{(1)}(x^\ell)\| \le \|g(x^\ell)\|$.

Proof. (i) By Proposition 2.1 (P4) in [94], $\|g^{(\alpha)}(x^\ell)\|$ for all $x^\ell \in \mathbf{x}$ is a nondecreasing function of $\alpha \in (0, 1)$; hence (7.4) holds. (ii) By the definition of reduced gradient and Proposition

7.2.1, (7.5) is obtained. (iii) The statements (i)–(ii) and the nonexpansiveness of the projection operator result in (iii). □

The following result plays a key role in proving our complexity analysis. The same results have been obtained for bound constrained nonlinear systems, see [56, 107, 108, 112].

7.2.3 Proposition. *Given the ℓth iterate x^ℓ and the ℓth solution p^ℓ of the trust-region subproblem* (4.21)*, we have for any index set I^ℓ and for all $\ell \geq 0$*

$$\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell) \geq \frac{1}{2} \|g_I^{(1)}(x^\ell)\|_\infty \min\left\{ \frac{\|g_I^{(1)}(x^\ell)\|_\infty}{\overline{\Gamma}}, \Delta^\ell \right\}. \tag{7.6}$$

Proof. Because the convexity of $\mathbf{x}$, defined by $p_I^\ell := \zeta^\ell g_I^{(1)}(x^\ell) \neq 0$ with

$$\zeta^\ell := \min\left\{ 1, \frac{\Delta^\ell}{\|g_I^{(1)}(x^\ell)\|_\infty} \right\} \in [0,1],$$

is in $\mathcal{T}^r$; hence, p^ℓ is feasible for (4.21). Moreover, let $\widetilde{p}^\ell$ be the convex combination of vectors p^ℓ and zero both of which are feasible for (4.21). Then it is feasible for (4.21). A minimizer of the one-dimensional model function

$$\chi(t) := \mathcal{Q}(x^\ell + \widetilde{p}^\ell) - \mathcal{Q}(x^\ell) = \mathcal{Q}(x^\ell + tp^\ell) - \mathcal{Q}(x^\ell) = t(g_I^\ell)^T p_I^\ell + \frac{1}{2} t^2 (p_I^\ell)^T B_{II}^\ell p_I^\ell$$

over $[0,1]$ can be found in the two following cases:

Case 1: $(p_I^\ell)^T B_{II}^\ell p_I^\ell > 0$. Due to the strict convexity of quadratic function χ,

$$t^* := \operatorname*{argmin}_{t\in[0,1]} \chi(t) = \min\left\{ -\frac{(g_I^\ell)^T p_I^\ell}{(p_I^\ell)^T B_{II}^\ell p_I^\ell}, 1 \right\}$$

is the unique minimizer over $[0,1]$. This fact that p^ℓ is the minimum of the subproblem (4.21) results in

$$\mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell) \leq \chi(t^*) = \begin{cases} -\frac{1}{2} \frac{((g_I^\ell)^T p_I^\ell)^2}{(p_I^\ell)^T B_{II}^\ell p_I^\ell}, & \text{if } -(g_I^\ell)^T p_I^\ell \leq (p_I^\ell)^T B_{II}^\ell p_I^\ell, \\ \frac{1}{2}(g_I^\ell)^T p_I^\ell, & \text{otherwise.} \end{cases}$$

In the second case, $(p_I^\ell)^T B_{II}^\ell p_I^\ell < -(g_I^\ell)^T p_I^\ell$ results in

$$(g_I^\ell)^T p_I^\ell + \frac{1}{2}(p_I^\ell)^T B_{II}^\ell p_I^\ell < \frac{1}{2}(g_I^\ell)^T p_I^\ell.$$

From the projection theorem and the property of the Euclidean norm, we get

$$\begin{aligned} -(g_I^\ell)^T p_I^\ell &= -\zeta^\ell (g_I^\ell)^T g_I^{(1)}(x^\ell) = \zeta^\ell (-g_I^\ell)^T \big(\pi[x^\ell - g_I^\ell] - x^\ell\big) \\ &= \zeta^\ell \big((x^\ell - g_I^\ell) - \pi[x^\ell - g_I^\ell]\big)^T g_I^{(1)}(x^\ell) + \zeta^\ell \|g_I^{(1)}(x^\ell)\|^2 \\ &\geq \zeta^\ell \|g_I^{(1)}(x^\ell)\|^2 \geq \zeta^\ell \|g_I^{(1)}(x^\ell)\|_\infty^2, \end{aligned}$$

so that

$$\begin{aligned}
\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell) &\geq -\chi(t^*) \geq \min\left\{\frac{1}{2}\frac{((g_I^\ell)^T p_I^\ell)^2}{(p_I^\ell)^T B_{II}^\ell p_I^\ell}, -\frac{1}{2}(g_I^\ell)^T p_I^\ell\right\} \\
&\geq \frac{1}{2}\min\left\{\frac{(\zeta^\ell)^2\|g_I^{(1)}(x^\ell)\|^4}{(p_I^\ell)^T B_{II}^\ell p_I^\ell}, \zeta^\ell\|g_I^{(1)}(x^\ell)\|^2\right\} \\
&\geq \frac{1}{2}\min\left\{\frac{(\zeta^\ell)^2\|g_I^{(1)}(x^\ell)\|^4}{\overline{\Gamma}\|p_I^\ell\|^2}, \zeta^\ell\|g_I^{(1)}(x^\ell)\|^2\right\} \\
&= \frac{1}{2}\min\left\{\frac{(\zeta^\ell)^2\|g_I^{(1)}(x)\|^4}{(\zeta^\ell)^2\overline{\Gamma}\,\|g_I^{(1)}(x^\ell)\|^2}, \zeta^\ell\|g_I^{(1)}(x^\ell)\|^2\right\} \\
&= \frac{1}{2}\|g_I^{(1)}(x^\ell)\|\min\left\{\frac{\|g_I^{(1)}(x^\ell)\|}{\overline{\Gamma}}, \zeta^\ell\|g_I^{(1)}(x^\ell)\|\right\} \\
&\geq \frac{1}{2}\|g_I^{(1)}(x^\ell)\|_\infty\min\left\{\frac{\|g_I^{(1)}(x^\ell)\|_\infty}{\overline{\Gamma}}, \zeta^\ell\|g_I^{(1)}(x^\ell)\|_\infty\right\} \\
&= \frac{1}{2}\|g_I^{(1)}(x^\ell)\|_\infty\min\left\{\frac{\|g_I^{(1)}(x^\ell)\|_\infty}{\overline{\Gamma}}, \|g_I^{(1)}(x^\ell)\|_\infty, \Delta^\ell\right\} \\
&= \frac{1}{2}\|g_I^{(1)}(x^\ell)\|_\infty\min\left\{\frac{\|g_I^{(1)}(x^\ell)\|_\infty}{\overline{\Gamma}}, \Delta^\ell\right\}
\end{aligned}$$

by the definition of ζ^ℓ and since $\overline{\Gamma} \geq 1$.

Case 2: $(p_I^\ell)^T B_{II}^\ell p_I^\ell = 0$. Then the function $\chi(t) = t(g_I^\ell)^T p_I^\ell$ is linear. Using the definition of ζ^ℓ, the feasibility of x^ℓ, and the monotonicity of the projection operator, we have

$$\begin{aligned}
(g_I^\ell)^T p_I^\ell &= \zeta^\ell (g_I^\ell)^T g_I^{(1)}(x^\ell) \\
&= -\zeta^\ell(-g_I^\ell)^T\big(\pi[x^\ell - g_I^\ell] - x^\ell\big) \\
&= -\zeta^\ell(-g_I^\ell)^T\big(\pi[x^\ell - g_I^\ell] - \pi[x^\ell]\big) \\
&< 0
\end{aligned}$$

since $\pi[x^\ell - g_I^\ell] \neq \pi[x^\ell]$; otherwise, Algorithm 7.4.1 discussed in Section 7.4 is stopped due to Proposition 7.2.2(iii). Hence, p^ℓ is a descent direction, and $t^* := \operatorname{argmin}_{t\in[0,1]} \chi(t) = 1$ is the unique minimizer of $\chi(t)$ on $[0,1]$; since $-(g_I^\ell)^T p_I^\ell > 0 = (p_I^\ell)^T B_{II}^\ell p_I^\ell$, the minimum is attained at $t^* = 1$, so that

$$\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell) \geq \frac{1}{2}\|g_I^{(1)}(x^\ell)\|_\infty \min\big\{\|g_I^{(1)}(x^\ell)\|_\infty, \Delta^\ell\big\}.$$

□

A **Cauchy point** is a minimizer of the model function $\mathcal{Q}$ along the projected steepest descent direction $\widetilde{p}^\ell := \zeta^\ell g^{(1)}(x^\ell)$ with $\zeta^\ell \in [0,1]$, i.e.,

$$p^\ell := \widetilde{t}^\ell \widetilde{p}^\ell \quad \text{with} \quad \widetilde{t}^\ell := \operatorname{argmin}\{\mathcal{Q}(t\widetilde{p}^\ell) \mid t \geq 0,\ t\widetilde{p}^\ell \in \mathbf{x}\}.$$

Since the Cauchy point is not unique in the constrained case, the following assumption is valid for any global minimizer of (4.21).

(GBO$_5$) The reduction of the model function $\mathcal{Q}$ is at least as much as the Cauchy point

$$\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell) \geq \frac{1}{2}\|g_I^{(1)}(x^\ell)\|_\infty \ \min\left\{\frac{\|g_I^{(1)}(x^\ell)\|_\infty}{\overline{\Gamma}}, \Delta^\ell\right\}, \tag{7.7}$$

where p^ℓ is a solution of the trust-region subproblem (4.21).
(GBO$_6$) There exists a constant $\Gamma^{\text{une}} \geq 1$ such that

$$\frac{1}{\Gamma^{\text{une}}}\|x\|_k \leq \|x\| \leq \Gamma^{\text{une}}\|x\|_k \quad \text{for } x \in \mathbf{x},$$

where $\|x\|_k$ are uniformly equivalent to the Euclidean ℓ_2 norm.

Inspired by Algorithm 2 in [112], the following result is valid.

7.2.4 Corollary. *The ℓth inexact solution of* (4.21) *satisfies the inequality* (7.7).

Proof. By choosing

$$\zeta^\ell := \min\left\{1, \frac{\Delta^\ell}{\|g_I^{(1)}(x^\ell)\|_\infty}\right\}$$

and computing $\widetilde{p}^\ell := \zeta^\ell g_I^{(1)}(x^\ell)$, the ℓth inexact solution of (4.21) is computed by

$$p^\ell := t\widetilde{p}^\ell \quad \text{with} \quad t := \min\left\{1, -\frac{(g_I^\ell)^T\widetilde{p}_I^\ell}{(\widetilde{p}_I^\ell)^T B_{II}^\ell \widetilde{p}_I^\ell}\right\}$$

and satisfies (7.7). □

7.3 Enforcing a good agreement

Our achievement is to use the sufficient descent condition (4.19) in the trust region framework instead of (4.22) to enforce a sensible decrease in the function value provided that

$$(g_I^\ell)^T p_I^\ell < 0 \tag{7.8}$$

holds. Define

$$(\mu^1)^\ell := \mu^\ell(1) := \frac{f(x^\ell + p^\ell) - f(x^\ell)}{(g_I^\ell)^T p_I^\ell} \tag{7.9}$$

and then rewrite (4.20) as

$$(\mu^1)^\ell|(\mu^1)^\ell - 1| \geq \beta, \tag{7.10}$$

with fixed $\beta \in]0, \frac{1}{4}[$, equivalent to either

$$\mu' \leq (\mu^1)^\ell \leq \mu'' \tag{7.11}$$

or

$$(\mu^1)^\ell \geq \mu''' \tag{7.12}$$

where

$$0 < \mu' := \frac{2\beta}{1+\sqrt{1-4\beta}} < \mu'' := \frac{1+\sqrt{1-4\beta}}{2} < 1 \tag{7.13}$$

and

$$\mu''' := \frac{1+\sqrt{1+4\beta}}{2}. \tag{7.14}$$

Conversely, if (7.11) and (7.12) hold, then (7.10) holds with $\beta := \mu'(1-\mu'') > 0$.

As earlier defined, if the agreement between f and $\mathcal{Q}$ is good, i.e.,

$$\mathcal{R}^\ell \geq r, \quad \text{with } r \in (0,1), \tag{7.15}$$

the ℓth iteration is called successful; otherwise, it is called unsuccessful. Instead, if the condition (7.10) holds, the ℓth iteration is called successful; otherwise, it is called unsuccessful. In the next proposition, we show that if the condition (7.10) holds, then (7.15) is satisfied.

We use the following result to construct an improved trust region algorithm.

7.3.1 Proposition. *Given the ℓth iterate x^ℓ, the positive semidefinite $B^\ell_{II} := B_{II}(x^\ell)$, and the ℓth solution p^ℓ of the trust-region subproblem* (4.21) *satisfying* (7.8)*, if the condition* (7.10) *holds, there exists an $r \in (0,1)$ such that* (7.15) *holds.*

Proof. Let

$$\widehat{\theta}^\ell := \frac{\mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell)}{(g^\ell_I)^T p^\ell_I}. \tag{7.16}$$

We want to find the lower and upper bound for $\widehat{\theta}^\ell$. To do so, let us define

$$\Phi(t) := \mathcal{Q}(x^\ell + tp^\ell) - \mathcal{Q}(x^\ell) = t(g^\ell_I)^T p^\ell_I + \frac{1}{2}t^2 (p^\ell_I)^T B^\ell_{II} p^\ell_I$$

as the one-dimensional model function over all $t \in [0,1]$. We consider two cases:

Case 1. $(p^\ell_I)^T B^\ell_{II} p^\ell_I > 0$. In this case, the quadratic function Φ is strictly convex with a unique minimizer over $[0,1]$ defined by $t^* := \min\Big\{ -\dfrac{(g^\ell_I)^T p^\ell_I}{(p^\ell_I)^T B^\ell_{II} p^\ell_I}, 1 \Big\}$. Since

$$\Phi(1) = \mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell) \geq \Phi(t^*) = \begin{cases} -\dfrac{1}{2}\dfrac{((g^\ell_I)^T p^\ell_I)^2}{(p^\ell_I)^T B^\ell_{II} p^\ell_I}, & \text{if } -(g^\ell_I)^T p^\ell_I \leq (p^\ell_I)^T B^\ell_{II} p^\ell_I, \\ \dfrac{1}{2}(g^\ell_I)^T p^\ell_I, & \text{otherwise,} \end{cases}$$

we get $0 < \widehat{\theta}^\ell \leq \dfrac{1}{2}$; here the second case holds since $(p^\ell_I)^T B^\ell_{II} p^\ell_I < -(g^\ell_I)^T p^\ell_I$ results in

$$(g^\ell_I)^T p^\ell_I + \frac{1}{2}(p^\ell_I)^T B^\ell_{II} p^\ell_I < \frac{1}{2}(g^\ell_I)^T p^\ell_I.$$

Case 2. $(p^\ell_I)^T B^\ell_{II} p^\ell_I = 0$. In this case, the function $\Phi(t) = t(g^\ell_I)^T p^\ell_I$ is linear. Since p^ℓ_I is the descent direction, $t^* := 1$ is the unique minimizer of Φ over $[0,1]$; hence

$$\Phi(1) = \mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell) \geq \Phi(t^*) = (g^\ell_I)^T p^\ell_I,$$

so that $0 < \widehat{\theta}^\ell \leq 1$.

As a result, we conclude that $0 < \widehat{\theta}^\ell \leq 1$. By (4.19), (7.8), and (7.11), we now get either

$$\mathcal{R}^\ell = \frac{f(x^\ell + p^\ell) - f(x^\ell)}{\mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell)} = \frac{(\mu^1)^\ell}{\widehat{\theta}^\ell} \geq \mu',$$

or

$$\mathcal{R}^\ell = \frac{f(x^\ell + p^\ell) - f(x^\ell)}{\mathcal{Q}(x^\ell + p^\ell) - \mathcal{Q}(x^\ell)} = \frac{(\mu^1)^\ell}{\widehat{\theta}^\ell} \geq \mu''',$$

where μ' and μ''' satisfying (7.13) and (7.14), respectively. By setting $r = \max(\mu', \mu'') \in (0, 1)$, the proof is obtained. □

7.4 The improved trust region algorithm

This section describes how to work out our improved trust region algorithm whose radius is updated according to the reduced gradient norm.

To update the trust region radius Δ^ℓ for all $\ell \geq 0$, the parameter λ^ℓ is restricted into the interval $\boldsymbol{\lambda} := [\underline{\lambda}, \overline{\lambda}]$, where $0 < \underline{\lambda} < \overline{\lambda} < \infty$. It is done by projecting λ^ℓ into the interval $\boldsymbol{\lambda}$

$$P_{\boldsymbol{\lambda}}(\lambda^\ell) := \min(\overline{\lambda}, \max(\lambda^\ell, \underline{\lambda})). \tag{7.17}$$

Before we introduce the trust region radius, the associated factor is defined according to

$$\lambda^\ell := \begin{cases} P_{\boldsymbol{\lambda}}(\sigma_2 \lambda^{\ell-1}), & \text{if (7.10) holds,} \\ \sigma_1 \lambda^{\ell-1}, & \text{otherwise,} \end{cases} \tag{7.18}$$

where $0 < \sigma_1 < 1 < \sigma_2$ are suitable constants and $P_{\boldsymbol{\lambda}}(.)$ is computed by (7.17). The trust region radius

$$\Delta^\ell := \lambda^\ell \|g^{\text{red}}(x^\ell)\|_\infty \tag{7.19}$$

is updated according to the reduced gradient.

In fact, whenever the condition (7.10) holds, the trust region method is **efficient**. In this case, the ℓth iteration is successful and Δ^ℓ is updated according to the reduced gradient norm. Otherwise, since the condition (7.10) does not hold, the trust region method is not efficient and the ℓth iteration is unsuccessful; Δ^ℓ is reduced.

We describe how to perform our algorithm, bound constrained active set trust region algorithm, called **BCASTR**. It tries to figure out which constraints are probably active at the solution. It is done by enforcing the sufficient descent condition (7.10). In this case, the iteration is successful and the trust region radius is updated according to the reduced gradient norm while the corresponding factor is increased. If the mentioned condition does not hold, the iteration is unsuccessful and the trust region radius is reduced. Moreover, the trust region subproblem in the subspace of non-active variables is solved in the hope of getting the minimizer. Once the reduced gradient is below a given threshold, **BCASTR** ends.

BCASTR takes the initial point $x^0 \in \mathbf{x}$ the feasible set $\mathbf{x}$ as input and returns the minimum point $x^{\rm best} \in \mathbf{x}$ and its function value $f^{\rm best} = f(x^{\rm best})$ as output. It uses the following tuning parameters:
$0 < \underline{\lambda} < \overline{\lambda} < \infty$ (minimum and maximum values for λ),
$q > 1$ (parameter for expanding/reducing the step size),
$0 < \sigma_1 < 1 < \sigma_2$ (parameters for updating λ),
$0 < \beta < 0.25$ (parameter for efficiency),
$0 < \rho \leq 1$ (parameter for freeing).

7.4.1 Algorithm. (**BCASTR, bound constrained active set trust region method**)
(ASTR_0) Choose $\lambda^0 \in (\underline{\lambda}, \overline{\lambda})$ and $x^0 \in \mathbf{x}$. Then, compute the initial function value $f^0 := f(x^0)$ and its gradient $g^0 := g(x^0)$. Next, choose the positive definite matrix B^0.
for $\ell = 1, 2, 3, \cdots$ **do**
(ASTR_1) Compute $g^{\rm red}(x^\ell)$ by (4.7). Then if $\|g^{\rm red}(x^\ell)\|_\infty \leq \varepsilon$, set $x^{\rm best} := x^\ell$ and $f^{\rm best} := f^\ell$. Then **BCASTR** ends.
(ASTR_2) If $\ell = 0$, initialize the working set $I^\ell := I_+(x^\ell)$; otherwise if the condition (4.13) is violated, the working set $I^\ell := I_+(x^\ell)$ is changed.
(ASTR_3) Find an inexact solution of the trust region subproblem (4.21) restricted to free variables.
(ASTR_4) Compute $x^{\ell+1} := x^\ell + p^\ell$, $f^{\ell+1} := f(x^{\ell+1})$, and $(\mu^1)^\ell$ by (7.10). Then if the condition (7.10) holds, go to (ASTR_6); otherwise, go to (ASTR_5).
(ASTR_5) Reduce $\lambda^{\ell+1}$ to (7.18) and $\Delta^{\ell+1}$ to (7.19). Then go to (ASTR_2).
(ASTR_6) Expand $\lambda^{\ell+1}$ to (7.18) and $\Delta^{\ell+1}$ to (7.19) and compute the gradient $g^{\ell+1} := g(x^{\ell+1})$.
(ASTR_7) Compute $I^{\ell+1} := I_-(x^{\ell+1})$ and update the positive definite matrix $B^{\ell+1}$.
end for

7.5 Complexity analysis and limit accuracy

In this section, under the assumptions (GBO_1)–(GBO_6), we prove the complexity bound for our algorithm and show that all strongly active variables are found ultimately at the nondegenerate stationary point.

The following is the result of Theorem 6.3.1 in [41].

7.5.1 Proposition. *Suppose that (GBO_1)–(GBO_6) hold and p^ℓ is the ℓth inexact solution of* (4.21). *Then*

$$|f(x^\ell + p^\ell) - \mathcal{Q}(x^\ell + p^\ell)| \leq \Gamma^{\rm ubh} \|p_I^\ell\|^2, \quad \textit{for all } \ell \geq 0,$$

where $x^\ell + p^\ell \in \mathcal{T}^r$ and $\Gamma^{\rm ubh} := (\Gamma^{\rm une})^2 \max\{\Gamma^{\rm ufh}, \Gamma^{\rm umh}\} > 0$.

7.5.2 Proposition. *Let p^ℓ be the ℓth inexact solution of* (4.21) *and suppose that (A1)-(A6) hold. If $\lambda^\ell \leq \dfrac{1}{\Gamma}$, then*

$$\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell) \geq \frac{\lambda^\ell}{2} \|g^{\rm red}(x^\ell)\|_\infty^2, \quad \textit{for all } \ell \geq 0. \tag{7.20}$$

Proof. From (7.4)–(7.6) and the role of updating Δ, we get

$$\begin{aligned}
\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell) &\geq \frac{1}{2}\|g_I^{(1)}(x^\ell)\|_\infty \min\left\{\frac{\|g_I^{(1)}(x^\ell)\|_\infty}{\overline{\Gamma}}, \Delta^\ell\right\} \\
&\geq \frac{1}{2\alpha}\|g_I^{(\alpha)}(x^\ell)\|_\infty \min\left\{\frac{\|g_I^{(\alpha)}(x^\ell)\|_\infty}{\alpha\overline{\Gamma}}, \lambda^\ell\|g^{\mathrm{red}}(x^\ell)\|_\infty\right\} \\
&\geq \frac{1}{2}\|g^{\mathrm{red}}(x^\ell)\|_\infty \min\left\{\frac{\|g^{\mathrm{red}}(x^\ell)\|_\infty}{\overline{\Gamma}}, \lambda^\ell\|g^{\mathrm{red}}(x^\ell)\|_\infty\right\} \\
&= \frac{\lambda^\ell}{2}\|g^{\mathrm{red}}(x^\ell)\|_\infty^2.
\end{aligned}$$

□

7.5.3 Proposition. *Suppose that* (GBO_1)*–*(GBO_6) *hold and let* p^ℓ *be the* ℓ*th inexact solution of* (4.21)*. Then*

$$\lambda^{\ell+1} \geq \lambda^{\min} := \min\left\{\underline{\lambda}, \frac{\sigma_1}{\overline{\Gamma}}, \frac{\sigma_1(1-\beta)}{2\Gamma^{\mathrm{ubh}}}\right\}, \quad \textit{for all } \ell \geq 0.$$

Proof. The proof is done by induction. With the choice of λ^0 in (ASTR_0) of Algorithm 7.4.1, the proof holds for $\ell = 0$; $\lambda^0 \geq \underline{\lambda} \geq \lambda^{\min}$. Let us assume that the induction hypothesis holds, i.e., $\lambda^\ell \geq \lambda^{\min}$. Then, we show that it is valid for $\lambda^{\ell+1}$. To do so, we consider two cases:

Case 1. If $\lambda^\ell > \min\left\{\frac{\sigma_1}{\overline{\Gamma}}, \frac{\sigma_1(1-\beta)}{2\Gamma^{\mathrm{ubh}}}\right\}$, then, by (7.19), we get

$$\lambda^{\ell+1} \geq \min\{\underline{\lambda}, \sigma_1\lambda^\ell\} \geq \lambda^{\min}.$$

Case 2. As earlier defined by (7.16), if $\lambda^\ell \leq \min\left\{\frac{\sigma_1}{\overline{\Gamma}}, \frac{\sigma_1(1-\beta)}{2\Gamma^{\mathrm{ubh}}}\right\}$, then we get

$$|\frac{(\mu^1)^\ell}{\widehat{\theta}^\ell} - 1| = |\mathcal{R}^\ell - 1| = \frac{|f(x^\ell + p^\ell) - \mathcal{Q}(x^\ell + p^\ell)|}{\mathcal{Q}(x^\ell) - \mathcal{Q}(x^\ell + p^\ell)} \leq \frac{2(\lambda^\ell)^2\Gamma^{\mathrm{ubh}}\|g^{\mathrm{red}}(x^\ell)\|_\infty^2}{\lambda^\ell\|g^{\mathrm{red}}(x^\ell)\|_\infty^2} = 2\Gamma^{\mathrm{ubh}}\lambda^\ell,$$

so that

$$\widehat{\theta}^\ell(1 - 2\Gamma^{\mathrm{ubh}}\lambda^\ell) \leq (\mu^1)^\ell \leq \widehat{\theta}^\ell(1 + 2\Gamma^{\mathrm{ubh}}\lambda^\ell).$$

Without loss of generality we choose from (7.13)

$$\mu' = \frac{2\beta}{1 + \sqrt{1 - 4\beta}} = \widehat{\theta}^\ell(1 - 2\Gamma^{\mathrm{ubh}}\lambda^\ell)$$

and

$$\mu'' = \frac{1 + \sqrt{1 - 4\beta}}{2} = \widehat{\theta}^\ell(1 + 2\Gamma^{\mathrm{ubh}}\lambda^\ell).$$

The product of μ' and μ'' results in $\widehat{\theta}^\ell := \dfrac{\beta}{1 - 4(\Gamma^{\mathrm{ubh}})^2(\lambda^\ell)^2} > 0$; hence

$$\mu' = \widehat{\theta}^\ell(1 - 2\Gamma^{\mathrm{ubh}}\lambda^\ell) = \frac{\beta}{1 + 2\Gamma^{\mathrm{ubh}}\lambda^\ell} \geq \frac{\beta}{2 - \beta} > 0$$

and

$$\mu'' = \widehat{\theta}^{\ell}(1 + 2\Gamma^{\mathrm{ubh}}\lambda^{\ell}) = \frac{\beta}{1 - 2\Gamma^{\mathrm{ubh}}\lambda^{\ell}} < 1,$$

by the assumption $\lambda^{\ell} \leq \dfrac{\sigma_1(1-\beta)}{2\Gamma^{\mathrm{ubh}}} < \dfrac{1-\beta}{2\Gamma^{\mathrm{ubh}}}$. As a result, $0 < \mu' < \mu'' < 1$ holds. Finally, the choice $\beta := \mu'(1-\mu'') > 0$ comes to the fact that (7.10) holds; hence the iteration is successful and so $\lambda^{\ell+1} \geq \underline{\lambda} \geq \lambda^{\min}$. □

Propositions 7.5.2 and 7.5.3 result in the following result.

7.5.4 Corollary. *Let p^{ℓ} be the ℓth inexact solution of* (4.21) *and suppose that* (GBO_3)–(GBO_6) *hold. Then*

$$\mathcal{Q}(x^{\ell}) - \mathcal{Q}(x^{\ell} + p^{\ell}) \geq \frac{\lambda^{\min}}{2}\|g^{\mathrm{red}}(x^{\ell})\|_{\infty}^2, \quad \text{for all } \ell \geq 0, \tag{7.21}$$

where $\lambda^{\min}$ comes from Proposition 7.5.3.

To prove the complexity analysis, let us define

$$\mathcal{K} := \{\ell \in N_0 \mid \|g^{\mathrm{red}}(x^{\ell})\|_{\infty} > \varepsilon\},$$
$$\mathcal{S}(\varepsilon) := \{\ell \in N_0 \mid \|g^{\mathrm{red}}(x^{\ell})\|_{\infty} > \varepsilon \text{ and (7.10) holds}\}.$$

Here $\mathcal{S}(\varepsilon)$ contains all successful iterations generated by (7.10) and $\mathcal{U}(\varepsilon)$ contains the unsuccessful iterations.

The following result gives an upper bound on the unsuccessful iterations.

7.5.5 Proposition. *Given $\sigma_1 \in (0,1)$ and $\overline{\lambda} > \underline{\lambda} > 0$, the number of unsuccessful iterations is bounded by*

$$|\mathcal{U}(\varepsilon)| \leq \left\lceil \log_{\sigma_1} \frac{\lambda^{\min}}{\overline{\lambda}} \right\rceil, \tag{7.22}$$

where $\lambda^{\min}$ comes from Proposition 7.5.3.

Proof. By the role of updating λ and Proposition 7.5.3, we get

$$\lambda^{\min} \leq \sigma_1^{|\mathcal{U}(\varepsilon)|}\overline{\lambda},$$

so that (7.22) is obtained. □

The following result is our main complexity result for nonconvex case.

7.5.6 Theorem. *Suppose that* (GBO_1)–(GBO_6) *hold and let p^{ℓ} be the ℓth inexact solution of* (4.21). *Moreover, let f^0 be the initial function value and $\mu^{\min} := \min(\mu', \mu''')$. Given any $\varepsilon \in (0,1)$,* **BCASTR** *uses at most*

(i) $\dfrac{f^0 - \widehat{f}}{\mu^{\min}\lambda^{\min}\varepsilon^2}$ *successful iterations (gradient evaluations) due to* (7.10)*, i.e.,*

$$|\mathcal{S}(\varepsilon)| \leq \mathcal{O}(\varepsilon^{-2}),$$

where $\widehat{f} := \inf_{\ell} f^{\ell}$ *is finite due to* (GBO$_1$) *and* $\lambda_{\min}$ *comes from Proposition 7.5.3.*

(ii) $\left\lceil \log_{\sigma_1} \dfrac{\lambda^{\min}}{\overline{\lambda}} \right\rceil \left(\dfrac{f^0 - \widehat{f}}{\mu^{\min}\lambda^{\min}\varepsilon^2} \right)$ *iterations, i.e.,*

$$|\mathcal{K}(\varepsilon)| \leq \mathcal{O}(\varepsilon^{-2}).$$

Proof. (i) Let $g_I := g_I(x^{\ell})$ and $g^{\mathrm{red}} := g^{\mathrm{red}}(x^{\ell})$. Then we conclude from Corollary 7.5.4 that

$$f^0 - \widehat{f} \geq \mu^{\min} \sum_{\ell \in \mathcal{S}(\varepsilon)} (\mathcal{Q}(x^{\ell}) - \mathcal{Q}(x^{\ell} + p^{\ell})) \geq \mu^{\min}\lambda^{\min}\varepsilon^2 |\mathcal{S}(\varepsilon)|, \tag{7.23}$$

so that

$$|\mathcal{S}(\varepsilon)| \leq \frac{f^0 - \widehat{f}}{\mu^{\min}\lambda^{\min}\varepsilon^2} := \mathcal{O}(\varepsilon^{-2}). \tag{7.24}$$

□

According to [136, Theorem 8.2] we show that when $\inf_{\ell} \|g^{\mathrm{red}}(x^{\ell})\| = 0$ and the sequence generated by our method converges to a point $\widehat{x} \in \mathbf{x}$ then $g^{\mathrm{red}}(\widehat{x}) = 0$ and all strongly active variables are found ultimately at the nondegenerate stationary point $\widehat{x}$ and unlimited zigzagging cannot occur.

7.5.7 Theorem. *Assume that the approximated Hessian matrix* B_{II} *restricted to the subspace of free variables is positive semidefinite matrix. Under assumptions Theorem 7.5.6, for sufficiently large* ℓ*, we have*

$$\inf_{\ell} f^{\ell} = \widehat{f} < \infty \quad \text{and} \quad \inf_{\ell} \|g^{\mathrm{red}}(x^{\ell})\|_{\infty} = 0. \tag{7.25}$$

Moreover, unlimited zigzagging cannot occur at the nondegenerate stationary point $\widehat{x}$*.*

Proof. For sufficiently large ℓ, $\inf_{\ell} f^{\ell} = \widehat{f}$ and $\widehat{f}$ is finite due to (GBO$_1$). For sufficiently large ℓ, we have

$$f^{\ell} - f^{\ell+1} \geq \mu^{\min}(\mathcal{Q}(x^{\ell}) - \mathcal{Q}(x^{\ell} + p^{\ell})) \geq \mu^{\min}\lambda^{\min}\|g^{\mathrm{red}}(x^{\ell})\|_{\infty}^2$$

by Corollary 7.5.4, so that (7.25) is obtained. We define freeing iterations with respect to any index set I by

$$\mathcal{F}_I := \{\ell \mid I = I^{\ell} = I_+(x^{\ell}) \neq I_-(x^{\ell})\}.$$

Whether there is a limited zigzagging or not makes the proof followed in two cases:

Case 1. Unlimited zigzagging. In such a case, $\mathcal{F}_I$ is infinite. Since I is often changed infinitely, $\#I$ is not finite, (4.21) is solved infinitely often for different I. By Proposition 7.5.5, the number of unsuccessful iterations is finite; hence, the number of successful iterations is infinite. For $I \subset I_+(x^{\ell})$, we have $\|g_I(x^{\ell})\|_{\infty} < \|g_{I_+}(x^{\ell})\|_{\infty} = \|g^{\mathrm{red}}(x^{\ell})\|_{\infty} = 0$ for sufficiently large $\ell \in \mathcal{F}_I$. Hence, $g_I = g_I(x^{\ell}) = 0$ and $g_I^T p_I^{\ell} = 0$ for sufficiently large ℓ. By the positive semidefiniteness of B_{II} and Corollary 7.5.4, we get a contradiction

$$0 < Q(x^{\ell}) - \mathcal{Q}(x^{\ell} + p^{\ell}) = -\frac{1}{2}(p_I^{\ell})^T B_{II} p_I^{\ell} \leq 0.$$

Hence, unlimited zigzagging cannot occur at the nondegenerate stationary point $\widehat{x}$.

Case 2. Limited zigzagging. In this case, $\mathcal{F}_I$ is finite, meaning that the number of freeing iterations is finite, i.e., there exists a number $\underline{\ell}$ such that $I := I^\ell := I_-(x^\ell)$ for $\ell > \underline{\ell}$. It means that the working set is fixed and all bounds are not fixed. In other words, the trust region algorithm, like the unconstrained case, solves the subproblem (4.21) in the subspace $I = I^\ell = I_-(x^\ell)$ for $\ell > \underline{\ell}$, so that there are some successful iterations; once the optimal activities are identified. Hence, for $\ell > \underline{\ell}$, (7.25) holds. According to [136, Theorem 8.2] when $\inf_\ell \|g^{\mathrm{red}}(x^\ell)\| = 0$ and the sequence generated by our method converges to a point $\widehat{x} \in \mathbf{x}$ then $g^{\mathrm{red}}(\widehat{x}) = 0$ and all strongly active variables are found ultimately at the nondegenerate stationary point $\widehat{x}$. □

Part III

New black box optimization methods

8 A new randomized method

This section discusses a randomized line search method for unconstrained black box optimization, called **VSBBO**. This is a joint work with Arnold Neumaier (cf. KIMIAEI & NEUMAIER [117]). In practice, **VSBBO** matches the quality of other state of the art algorithms for finding, in small and large dimensions, a local minimizer with reasonable accuracy. Although our theory guarantees only local minimizers our heuristic techniques turn **VSBBO** into an efficient global solver. In very thorough numerical experiments, we found in most cases either a global minimizer, or where this could not be checked, at least a point of similar quality with the best competitive global solvers. For smooth, everywhere defined functions, it is proved that, with probability arbitrarily close to 1, the basic version of our algorithm finds with $\mathcal{O}(nR\varepsilon^{-2})$ function evaluations a point with gradient 2-norm is blow ε, where n is the dimension and $R \geq \mathcal{O}(\log\log(n/\varepsilon^2))$ is the number of random directions used in each iteration. In the smooth convex case, this number improves to $\mathcal{O}(nR\varepsilon^{-1})$ and in the smooth (strongly) convex case to $\mathcal{O}(R\log n\varepsilon^{-1})$. This matches known recent complexity results for reaching a slightly different goal, namely the expected gradient 2-norm is below ε.

8.1 Overview of the new method

Our achievement is to describe a new, practically very efficient randomized method, called **Vienna randomized black box optimization** algorithm (**VSBBO**), for which good local complexity results can be proved, and which is competitive with the state of the art solvers for global black box optimization.

An algorithm loosely related to **VSBBO** (but without complexity guarantees) is the **Hit-and-Run** algorithm by BÉLISLE [11].

VSBBO contains many new useful techniques, in particular:
• Several kinds of search directions ensure the good practical performance.
• Adaptive heuristic estimations for the Lipschitz constant are used.
• A sensible scaling vector is estimated.
• The gradient vector is estimated by a randomized finite difference approach.

In summary,
• Section 8.2 introduces a theoretical criterion in finite precision arithmetic to terminate **VSBBO** in an approximated local minimizer.
• Section 8.3 gives an overview of state of the art complexity bounds of BBO methods.
• In Section 8.4, a randomized line search, called **RLS**, is constructed.
• Section 8.6 first introduces a basic version of our fixed decrease search algorithm, called **FDS-basic** to hopefully get a decrease in the function value. It has repeated calls to **RLS** until the

function value is decreased. Then the basic version of our algorithm is introduced while having repeated calls to **FDS-basic** until the function value is decreased.
• In Section 8.7, the complexity of the basic version of **VSBBO** for reaching a given accuracy with probability arbitrarily close to 1 matches the known recent complexity results for reaching a slightly different goal, namely the expected gradient 2-norm is below ε.
• In Section 8.8, we introduce heuristic techniques to improve the performance in practice, leading to the **VSBBO** implemented documentation in Section 8.9.
• The improved version of our algorithm including all heuristic techniques and directions is introduced in Section 8.9.
• The numerical results of Section 8.10 show that the the improved version of **VSBBO** matches the quality of other state of the art algorithms for finding, with reasonable accuracy, a global minimizer in small and large dimensions, or at least in the majority of cases a point of a quality comparable with the best competing algorithms, even the basic version of our algorithm is more efficient than algorithms suggested by Bergou et al. [16] and Gratton et al. [84].

8.2 Finite iteration goal

We use a scaled 2-norm $\|p\|$ and its dual norm $\|g\|_*$ of $p, g \in \mathbb{R}^n$, defined by

$$\|p\| := \sqrt{\sum_i p_i^2/s_i^2}, \quad \|g\|_* := \sqrt{\sum_i s_i^2 g_i^2} \quad (\text{all } s_i > 0) \tag{8.1}$$

in terms of a positive scaling vector $s \in \mathbb{R}^n$. This scaling vector will be estimated later in Subsection 8.8.5.

In exact precision arithmetic, the exact gradient vanishes at a good local minimizer whose function value is less than the initial function value. But in finite precision arithmetic optimization methods may get stuck in nearly flat regions. Hence they need a theoretical criterion to distinguish good local minimizers from spurious apparent local minimizers.

Our theoretical criterion

$$f(x^{\text{best}}) \le \sup_{x \in \mathbb{R}^n} \{f(x) \mid \|g(x)\|_* \le \varepsilon \text{ and } f(x) \le f(x^0)\}. \tag{8.2}$$

satisfies with a given probability arbitrarily close to one at the point x^{best} and **VSBBO** starting from a point x^0 terminates at x^{best} which is an approximated local minimizer. Here $g(x)$ denotes the exact gradient of f at x, only when the gradient exists.

Clearly, any local minimizer whose function value is not greater than the initial function value satisfies the condition (8.2) for every $\varepsilon > 0$. However, for fixed $\varepsilon > 0$, this may also be satisfied far away from a local minimizer at paths a very flat part of the graph of f. To see the meaning of the condition (8.2), we consider the following result.

8.2.1 Proposition. *If $\widehat{x}$ denotes the minimizer of a σ-strongly convex quadratic function $f(x)$*

$$f(\widehat{x}) \ge f(x) + g(x)^T(\widehat{x} - x) + \frac{\sigma}{2}\|\widehat{x} - x\|^2 \quad \text{for all } x \in \mathbb{R}^n. \tag{8.3}$$

then (8.2) *implies $f(x) - f(\widehat{x}) \le \varepsilon^2/(2\sigma)$ for $x \in \mathbb{R}^n$.*

Proof. For fixed x, the right-hand side of (8.3) is a convex quadratic function of $\widehat{x}$, minimal when its gradient vanishes. By (8.1), this is the case iff $\widehat{x}_i$ takes the value $x_i - \dfrac{s_i}{\sigma} g_i(x)$ for $i = 1, \cdots, n$, so that

$$f(\widehat{x}) \geq f(x) - \frac{1}{2\sigma}\|g(x)\|_*^2 \text{ for } x \in \mathbb{R}^n.$$

Therefore from (8.2) we conclude that

$$f(x) - f(\widehat{x}) \leq \frac{1}{2\sigma}\|g(x)\|_*^2 \leq \frac{\varepsilon^2}{2\sigma} \quad \text{for } x \in \mathbb{R}^n.$$

□

In fact when σ is tiny the function f is very flat. In this case, the speed of convergence of the optimization methods may be extremely slowed down and a spurious apparent local minimizer is found.

8.3 Complexity

We assume the standard assumptions for the complexity analysis of BBO algorithms:
(BBO_1) The function f is continuously differentiable on $\mathbb{R}^n$, and its gradient is Lipschitz continuous with Lipschitz constant L.
(BBO_2) The level set $\mathcal{L}(x^0) := \{x \in \mathbb{R}^n \mid f(x) \leq f(x^0)\}$ of f at x^0 is compact. In practice $\widehat{f} := \min\limits_{x \in \mathbb{R}^n} f(x)$ is finite.

Under these assumptions complexity bounds limit the size of the number $N(\varepsilon)$ of function evaluations needed to reach the goal (8.2) with a given probability (or a related goal). The appropriate asymptotic form for the expression $N(\varepsilon)$, found by Vicente [161], Dodangeh & Vicente [59], Dodangeh, Vicente & Zhang [60], Gratton et al. [84], Bergou, Gorbunov & Richtárik [16], and Nesterov & Spokoiny [133, 135], depends on the properties (smooth, smooth convex, or smooth strongly convex) of f; cf. Subsection 8.4.1 below.

case	goal	complexity
nonconvex	$\mathbf{E}(\|g\|_*) \leq \varepsilon$	$\mathcal{O}(n\varepsilon^{-2})$
convex	$\mathbf{E}(\|g\|_*) \leq \varepsilon$	$\mathcal{O}(n\varepsilon^{-1})$
convex	$\mathbf{E}(f - \widehat{f}) \leq \varepsilon$	$\mathcal{O}(n\varepsilon^{-1})$
strongly convex	$\mathbf{E}(\|g\|_*) \leq \varepsilon$	$\mathcal{O}(n \log \varepsilon^{-1})$
strongly convex	$\mathbf{E}(f - \widehat{f}) \leq \varepsilon$	$\mathcal{O}(n \log \varepsilon^{-1})$

Table 8.1: Complexity results for randomized BBO in expectation (Bergou et al. [16] for all cases)

Bergou et al. [16] and Nesterov & Spokoiny [135] generalized this result to give algorithms with complexity results for the nonconvex, convex and strongly convex cases shown in Table 8.1.

In each case, the bounds are better by a factor of n than the best known complexity results for deterministic algorithms (by Dodangeh & Vicente [59], Vicente [161] and Konečný & Richtárik [38]) given in Table 8.2. Of course, being a randomized algorithm, the performance guarantee obtained by Bergou et al. is slightly weaker, only valid in expectation. Moreover, they generated step sizes without testing whether the function value is improved or not. This is the reason why the algorithms proposed by Bergou et al. [16] are numerically poor, see Section 8.10.

case	goal	complexity
nonconvex	$\|g\|_* \leq \varepsilon$	$\mathcal{O}(n^2\varepsilon^{-2})$
convex	$\|g\|_* \leq \varepsilon$	$\mathcal{O}(n^2\varepsilon^{-1})$
convex	$f - \widehat{f} \leq \varepsilon$	$\mathcal{O}(n^2\varepsilon^{-1})$
σ-strongly convex	$\|g\|_* \leq \varepsilon$	$\mathcal{O}(n^2 \log \varepsilon^{-1})$
σ-strongly convex	$f - \widehat{f} \leq \varepsilon$	$\mathcal{O}(n^2 \log \varepsilon^{-1})$

Table 8.2: Complexity results for deterministic BBO (Vicente [161] for the nonconvex case, Dodangeh & Vicente [59] for the convex and the strongly convex cases, Konečný & Richtárik [38] for all cases)

The best complexity bound for a direct search with probabilistic (rather than expectation) guarantees has been found by Gratton et al. [84], only for nonconvex case. They used the Chernoff bounds to prove that a complexity bound $\mathcal{O}(nR\varepsilon^{-2})$ holds, R is the number of random directions, uniformly independently distributed on the unit sphere, used in each iteration.

case	goal	complexity
nonconvex	$\Pr(\|g\|_* \leq \varepsilon) \geq 1 - \eta$	$\mathcal{O}(nR\varepsilon^{-2})$
convex	$\Pr(\|g\|_* \leq \varepsilon) \geq 1 - \eta$	$\mathcal{O}(nR\varepsilon^{-1})$
convex	$\Pr(f - \widehat{f} \leq \varepsilon) \geq 1 - \eta$	$\mathcal{O}(nR\varepsilon^{-1})$
σ-strongly convex	$\Pr(\|g\|_* \leq \varepsilon) \geq 1 - \eta$	$\mathcal{O}(R \log n\varepsilon^{-1})$
σ-strongly convex	$\Pr(f - \widehat{f} \leq \varepsilon) \geq 1 - \eta$	$\mathcal{O}(R \log n\varepsilon^{-1})$

Table 8.3: Complexity results for randomized BBO with probability $1 - \eta$, for fixed $0 < \eta < 1$ (Gratton et al. [84] for the nonconvex case, present paper for all cases). Here $R \geq \Omega\Big(\log\log(n/\varepsilon^2) + \log \eta^{-1}\Big)$ is the number of random directions used in each iteration for a given $0 < \eta < 1$.

Our complexity bound for nonconvex is the same as the one found by GRATTON et al. [84] using the Chernoff bounds – in terms of both order and factor – but with the difference that its constant factor is proven by PINELIS [143] and its optimal value is obtained by a numerical simulation by us (see Subsection 8.11.3). Both complexity results are better by the factor of R/n than those given in Table 8.1 and are more reasonable than those given in Table 8.2.

Our complexity bounds for the convex and strongly convex cases are proven with probability arbitrarily close to 1, which are new results and are more reasonable than those given in Table 8.2, only valid in expectation. Table 8.3 summarizes our complexity results for all cases, matching GRATTON et al. [84] for the nonconvex case. GRATTON et al.'s results for the nonconvex case allow $R = 2$, while **VSBBO** requires

$$R \geq \Omega\Big(\log\log\frac{n}{\varepsilon^2} + \log\eta^{-1}\Big) \text{ for a given } 0 < \eta < 1.$$

But $\log\log\frac{n}{\varepsilon^2}$ and $\log\eta^{-1}$ cannot be large for reasonable values of n, ε, and η.

8.4 Line search techniques for BBO

In this section, we describe methods that try to achieve a decrease in the function value using line searches along specially chosen random directions. Random directions are used to exploit the fact that randomized black box optimization methods have a worst case complexity by a factor of n superior to those of deterministic algorithms (see cf. [9]).

A line search then polls one or more points along the lines in each chosen direction starting at the current best point. Several such line searches are packaged together into a randomized line search (multi-line search later), for which strong probabilistic results can be proved.

The details are chosen in such a way that failure to achieve the desired descent implies that, with probability arbitrarily close to one, a bound on the gradient is obtained.

8.4.1 Probing a direction

For every $x, p \in \mathbb{R}^n$, $f(x) - f(x \pm p)$ is called the **gain** along $\pm p$ and $\Delta \geq 0$ is called a threshold for the gain. First we give a theoretical test that either results in a gain of Δ or more in function value, or gives a small upper bound for the norm of at least one of the exact gradients encountered though our algorithm never calculates ones.

Assumption (BBO_1) implies that for every $x, p \in \mathbb{R}^n$, we have

$$f(x+p) - f(x) = g(x)^T p + \frac{1}{2}\gamma\|p\|^2, \tag{8.4}$$

where γ depends on x and p and satisfies one of

$$|\gamma| \leq L, \quad \text{(general case)} \tag{8.5}$$

$$0 \leq \gamma \leq L, \quad \text{(convex case)} \tag{8.6}$$

$$0 < \sigma \leq \gamma \leq L. \quad \text{(strongly convex case)} \tag{8.7}$$

Here σ comes from Proposition (8.3). In all three cases,

$$g(x)^T p - \frac{1}{2}L\|p\|^2 \leq f(x+p) - f(x) \leq g(x)^T p + \frac{1}{2}L\|p\|^2. \tag{8.8}$$

Continuity and condition (BBO_2) imply that a minimizer $\widehat{x}$ exists and

$$r^0 := \sup_{x \in \mathbb{R}^n} \left\{ \|x - \widehat{x}\| \mid f(x) \leq f(x^0) \right\} < \infty. \tag{8.9}$$

(It is enough that this holds with x^0 replaced by some point found during the iteration, which is then taken as x^0).

8.4.1 Proposition. *Let $x, p \in \mathbb{R}^n$ and $\Delta \geq 0$. Then (BBO_1) implies that*

$$L \geq \frac{|f(x+p) + f(x-p) - 2f(x)|}{\|p\|^2}, \tag{8.10}$$

and at least one of the following holds:
(i) $f(x+p) < f(x) - \Delta$,
(ii) $f(x+p) > f(x) + \Delta$ and $f(x-p) < f(x) - \Delta$,
(iii) $|g^T p| \leq \Delta + \frac{1}{2}L\|p\|^2$.

Proof. Taking the sum of (8.8) and the formula obtained from it by replacing p with $-p$ gives (8.10).

Assume that (iii) is violated, so that $\Delta + \frac{1}{2}L\|p\|^2$. Then by (8.4) with $\mp p$ in place of p, we have for an appropriate choice of the sign

$$f(x \mp p) - f(x) \leq \mp g(x)^T p + \frac{1}{2}L\|p\|^2 = -|g^T p| + \frac{1}{2}L\|p\|^2 < -\Delta.$$

If the lower sign applies we conclude that (i) holds. If the upper sign applies we get the second half of (ii), and the first half follows from $f(x+p) - f(x) \geq g(x)^T p - \frac{1}{2}L\|p\|^2 > \Delta$. □

Proposition 8.4.1 will play a key role in the construction of our randomized line search **RLS** detailed in Subsection 8.5:
• It exploits the well-known (Evtushenko [70], Pintér [144], Kvasov & Sergeyev [120]) lower bound (8.10) for the Lipschitz constant L which can be used to find reasonable estimates for L.
• If (i) holds, then the step p gives a gain of at least Δ, called the **sufficient gain**.
• If (ii) holds, then the step $-p$ gives a sufficient gain of at least Δ.
• If neither (i) nor (ii) holds (no sufficient gain is found along $\pm p$) then (iii) holds, giving a useful upper bound for the directional derivative.

In particular, this allows us to prove statements about the exact gradient even though our algorithm never calculates one.

8.4.2 Random search directions

Random directions are uniformly independent and identically distributed (**i.i.d**) in $[-\frac{1}{2}, \frac{1}{2}]^n$, computed by

$$p = \text{rand}(n, 1) - 0.5, \tag{8.11}$$

where rand generates a uniformly distributed random vector. We are interested in scaling the random directions by

$$p := p(\delta/\|p\|), \tag{8.12}$$

where

$$\delta = \max\left(\delta^{\min}, \min\left(\sqrt{\alpha^e \gamma^\delta \Delta/\lambda}, \delta^{\max}\right)\right). \tag{8.13}$$

Here the tuning parameter $\gamma^\delta > 0$ is to adjust δ, the sensible positive lower and upper bounds $0 < \delta^{\min} < \delta^{\max} < +\infty$ are the tuning parameters to safeguard δ, and α^e is a step size, and λ is an approximation for the Lipschitz constant L. As discussed earlier in Section 8.3, **VSBBO** tries to find a point satisfying (8.2). In practice the goal of the scaling of the search direction (8.12) is to approximately minimize the bound $\sqrt{cn}\Gamma(\delta)$ with

$$\Gamma(\delta) := L\delta + \frac{2\Delta}{\delta}; \tag{8.14}$$

to find a point satisfying (8.2). For fixed Δ, the scale-dependent factor (8.14) is smallest for the choice

$$\widehat{\delta} := \sqrt{2\Delta/L}. \tag{8.15}$$

However, in practice, L is unknown and we replace later it by λ. Proposition 8.4.1 implies that

$$\lambda^0 \le \lambda \le \max(\lambda^0, L) \le \lambda^0 + L, \tag{8.16}$$

where λ^0 is the initial value of λ.

As a guarantee for our complexity results, we need that sufficiently many search directions p satisfy an **angle condition** of the form

$$\sup \frac{g^T p}{\|g\|_* \|p\|} \le -\Delta^{\mathrm{a}} < 0. \tag{8.17}$$

Here p and g come from (8.1) and $\Delta^{\mathrm{a}} > 0$ is a tuning parameter for the angle condition. The following variant of the angle condition (8.17) plays a key role to get our complexity bounds.

8.4.2 Proposition. *For random search directions generated by* (8.11) *and scaled by* (8.12) *satisfies* $\|p\| = \delta$ *and, with probability* $\ge \frac{1}{2}$,

$$\|g(x)\|_* \|p\| \le 2\sqrt{cn}|g(x)^T p| \tag{8.18}$$

with a positive constant $c \approx 4/7$.

Proof. As defined earlier in Section 8.3, $s \in \mathbb{R}^n$ is a scaling vector. Define $\overline{p}_i := p_i/s_i$ and $\overline{g}_i := s_i g_i$. Then by (8.1), $g^T p = \overline{g}^T \overline{p}$ and $\|g\|_* = \|\overline{g}\|_2$ and $\|p\| = \|\overline{p}\|_2$; so the results of Subsection 8.11.3 apply after scaling and give $c = c_0/4 \approx 4/7$. □

8.5 A randomized line search

In this section, we construct a randomized line search algorithm, called **RLS**. It polls in random directions (satisfying (8.18), with probability $\geq \frac{1}{2}$, generated by (8.11), and scaled by (8.12)) in a line search fashion a few objective function values each in the hope of finding sufficient gains by more than a multiple of Δ.

8.5.1 An extrapolation step

Extrapolation speeds up reaching a minimizer by expanding step sizes and computing the corresponding trial points and their function values as long as sufficient gains are found. We discuss how to construct extrapolation steps, called **extrapolationStep**, trying to hopefully find sufficient gains. **extrapolationStep** may perform extrapolation along with either the search direction p or its opposite direction.

Care must be taken to ensure that the book-keeping needed for the evaluation of the lower bound for the Lipschitz constant comes out correctly. To ensure this during an extrapolation step, we always use `xm` for the best point found, and rescale p by (8.12) such that the next evaluation is always at $\texttt{xm}+p$ and a former third evaluation point is at $\texttt{xm}-p$. The function values immediately after the next evaluation are then

$$\texttt{fl} := f(\texttt{xm} - p), \quad \texttt{fm} := f(\texttt{xm}), \quad \texttt{fr} := f(\texttt{xm} + p). \tag{8.19}$$

At this stage, we can compute the lower bound

$$\lambda := \max(\lambda^{\text{old}}, |\texttt{fl} + \texttt{fr} - 2\texttt{fm}|/\delta^2) \tag{8.20}$$

for the Lipschitz constant L, valid by (8.10). Note that the initial λ^{old} is the tuning parameter $\lambda^{\max}$, however, it is updated by **extrapolationStep** and may be estimated by a heuristic formula.

Afterwards, whenever $\texttt{fr} < \texttt{fm}$, the best point is updated by overwriting $\texttt{xm}+p$ over `xm`, with the consequence that in this case

$$\texttt{fl} := f(\texttt{xm} - 2p), \quad \texttt{fm} := f(\texttt{xm} - p), \quad \texttt{fr} := f(\texttt{xm}). \tag{8.21}$$

As defined earlier in Subsection 8.4.1, $\texttt{df} := \texttt{fm} - \texttt{fr}$ is the gain and if the condition

$$\texttt{df} > \min(\alpha^e, 1)\Delta \tag{8.22}$$

holds, we say that the **sufficient gain** is found.

R denotes the number of the random search directions used in **RLS** and **a** denotes the list of R **extrapolation step sizes**. All components of the initial **a** are one; each of its components is expanded or reduced according to whether a sufficient gain is found or not. Let `ne` be the number of iterations generated by **extrapolationStep** to exceed sufficient gains. If the counter `ne` stays zero, **extrapolationStep** cannot find a sufficient gain; in this case it is called **unsuccessful**. Otherwise, a sufficient gain is found and **extrapolationStep** is called **successful**. The Boolean variable `good` takes 0 if no sufficient gain is found and 1 otherwise. In other words, if `good` is zero, **extrapolationStep** is unsuccessful; otherwise, **extrapolationStep** is successful.

t is a counter for R taking $0, \cdots, R$. It does not change inside **extrapolationStep**, but it is updated later outside **extrapolationStep** (inside **RLS**).

We must be careful to make sure that the estimation of the Lipschitz constant is correct, especially when an extrapolation step – improving the function value – is tried. This estimation is obtained when either the extrapolation step is performed (the first iteration was tried and the second iteration may be tried) or the opposite direction is tried. Let `fe` be the function value improved by an extrapolation step in each iteration. Instead of the previous best function value `fm`, `fe` must be used to estimate the Lipschitz constant λ by (8.20). In this case, the previous best function value `fm` is restored in `ft`. Then, after we estimate λ by (8.20), `fm` is replaced by `ft`. Finally, whenever the extrapolation step ends, `fm` is replaced by `fe` and hence the best point and its function value are updated.

Denote by α^e the step sizes used in **extrapolationStep**. **extrapolationStep** first takes the initial step size $\alpha^e = 1$, which is necessary to find an approximated lower bound for the Lipschitz constant L, gets an upper bound for the gradient norm, and then takes the second step size from $\mathbf{a}^t$ while expanding it until the function value is decreased. One of the following cases is happened:
(i) A sufficient gain is found along the direction p.
(ii) A sufficient gain is found along the direction $-p$.
(iii) No sufficient gain is found along $\pm p$.
If either (i) or (ii) holds, **extrapolationStep** is successful. But if (iii) holds **extrapolationStep** is unsuccessful.

Throughout the paper, $A_{:k}$ denotes the kth column of a matrix A.

extrapolationStep takes the old best point `xm` and its function value `fm`, the search direction p, the threshold for good improvement $\Delta > 0$, the old approximation $\lambda \geq 0$ for the Lipschitz constant L, the tth extrapolation step size $\mathbf{a}^t$, and maximum number of function evaluations `nfmax` as input and uses the following tuning parameters:
$\gamma^e > 1$ (the factor for extrapolation test),
$E \geq 1$ (maximum number of extrapolations).
It returns a newest best point `xm` and its function value `fm`, a new approximation λ for the Lipschitz constant L, the Boolean variable `good`, and the tth extrapolation step size $\mathbf{a}^t$ as output.

8.5.1 Algorithm. (**extrapolationStep, an extrapolation step**)
(ES_0) **Initialization.** Initialize the step size $\alpha^e := 1$, which is necessary to estimate λ below.
(ES_1) **Trying either an extrapolation step along** p **or** $-p$.
for `ne` $= 0, \cdots, E$ **do**
(1) Compute the trial point `xr` $:=$ `xm` $+ \alpha^e p$, its function value `fr` $:= f($`xr`$)$, and its gain `df` $:=$ `fm` $-$ `fr`.
(2) If `nfmax` is reached, set `xm` $:=$ `xr` and `fm` $:=$ `fr` provided that `df` > 0 and end **extrapolationStep**.
(3) If `ne` $= 1$ temporarily save the old best function value `fm` in `ft` and replace `fm` by the new best function value `fe` (found so far) – only for the estimation λ. Then estimate λ by (8.20) and the old best function value `fm` is replaced by `ft`.
(4) If the condition (8.22) holds a sufficient gain is found. Then if `ne` $= 1$ save the old best function value `fm` in `fl` and choose $\mathbf{a}^t$ as α^e; otherwise, expand the step size to $\alpha^e := \gamma^e \alpha^e$ and save the current function value `fr` in `fe`. Otherwise if the condition (8.22) does not hold no gain is found and the for loop ends.

end for

(ES_2) **Updating the best point, its function value, its step size if a gain is found**.
If `ne` > 0 a sufficient gain along p is found. Since the last point generated by the extrapolation could not improve the function value,
(1) reduce the step size to $\alpha^e := \alpha^e/\gamma^e$,
(2) update the best point by `xm` $:=$ `xm` $+ \alpha^e p$,
(3) evaluate the Boolean variable `good` to be true,
(4) save the step size α^e of the current best point `xm` in $\mathbf{a}^t$ and `fe` in `fm`.
Otherwise, once try another extrapolation by
(1) replacing the search direction p by its opposite direction $-p$,
(2) saving the current function value `fr` in the third former function value `fl`,
(3) going to (ES_1).
Otherwise,
(1) evaluate the Boolean variable `good` to be false,
(2) terminate **extrapolationStep** since no sufficient gain is found along $\pm p$.

8.5.2 RLS, a randomized line search method

For each random direction generated, a randomized line search, called **RLS**, using **extrapolationStep** is performed where the following happens:
• A step in the current direction is tried.
• If a sufficient gain is found, a sequence of extrapolations is tried.
• If sufficient negative gain is found, a step in the opposite direction is tried.
• If a sufficient gain is found in the opposite direction, a sequence of extrapolations is tried.
• If no sufficient gain along $\pm p$ is found, the step size is reduced.

RLS takes the old best point `xm` and its function value `fm`, the threshold for good improvement $\Delta > 0$, the old approximation $\lambda \geq 0$ for the Lipschitz constant L, maximal number of function evaluations `nfmax`, and the list $\mathbf{a}$ of extrapolation step sizes as input and uses the following tuning parameters:
$\gamma^e > 1$ (the factor for extrapolation test),
$E \geq 1$ (maximum number of extrapolations),
$\delta^{\min}/\delta^{\max}$ (minimum/maximum norm of trial steps),
γ^λ (factor for adjusting δ),
$\alpha^{\min} \in (0, 1)$ (minimum threshold for step sizes),
$R > 0$ (the number of random search directions).
It returns a newest best point `xm` and its function value `fm`, a new approximation $\lambda \geq 0$ for the Lipschitz constant L, and an updated list $\mathbf{a}$ of extrapolation step sizes as output.

8.5.2 Algorithm. (**RLS, a randomized line search**)
for $t = 1, \cdots, R$ **do**
(RLS_1) Compute the random direction p^t by (8.11) and δ^t by (8.13). Then scale it by (8.12).
(RLS_2) Perform **extrapolationStep** to hopefully get sufficient gains, resulting in λ and possibly the newest best point `xm` and its function value `fm`. Once `nfmax` is reached, **RLS** ends.
(RLS_3) If there is no sufficient gain, `ne` $= 0$, reduce the tth step size to

$$\mathbf{a}^t := \max(\mathbf{a}^t/\gamma^e, \alpha^{\min}). \tag{8.23}$$

end for

In (8.23), the extrapolation step sizes need to be reduced whenever there is no sufficient gain. Hence, they need to be controlled by the tuning parameter $\alpha^{\min}$.

We now prove that one obtains either a sufficient gain of multiple of Δ or, with probability arbitrarily close to 1, an upper bound of $\|g\|_*$ for at least one of the exact gradients encountered though our algorithm never calculates ones.

8.5.3 Theorem. *Assume that* (BBO_1) *holds and let* `nf` *be the counter for the number of function evaluations, R be the number of random search directions, and Δ_f be the improvement on the function value in* **RLS**. *Moreover, let $\overline{\alpha} := \min(\alpha^e, 1)$ and $\overline{\Delta} := \overline{\alpha}\Delta$, where α^e is the step size generated by* **extrapolationStep**. *Here* `nfmax` *is assumed to be sufficiently large.*

(i) f decreases by at least

$$\overline{\Delta}_f := \overline{\Delta}\max(\mathtt{nf} - 2R - 1, 0) \tag{8.24}$$

(Note that $\overline{\Delta}_f$ may be zero, catering for the case of no strict decrease).

(ii) Suppose that $0 < \eta < 1$ and $R := \lceil \log_2 \eta^{-1} \rceil$. If f does not decrease by more than a multiple of Δ then, with probability $\geq 1 - \eta$, the original point or one of the points evaluated with better function values has a gradient g with

$$\|g\|_* \leq \sqrt{cn}\Gamma(\delta), \tag{8.25}$$

where c is the constant in Proposition 8.4.2 and $\Gamma(\delta)$ is computed by (8.14).

Proof. (i) Clearly, the function value of the best point does not increase. Thus (i) holds if $\mathtt{nf} - 2R - 1 \leq 0$. If this is not the case, then $\mathtt{nf} \geq 2R + 2$. But in the for loop of **RLS**, R directions p are generated and at most two function values are computed, unless an extrapolation step is performed. In the latter case, at least $\mathtt{nf} - 2R - 1$ additional function values are computed during the extrapolation stage, each time with a sufficient gain of at least $\overline{\Delta}$. Thus the total sufficient gain is at least (8.24).

(ii) Assume that f does not decrease by more than $\overline{\Delta}$. For $t = 1, \ldots, R$, let p^t be the tth random direction generated by (8.12), and let x^t be the best point obtained before searching in direction p^t. Then, from Proposition 8.4.1, we get

$$|g(x^t)^T p^t| \leq \overline{\Delta} + \frac{L}{2}\|p^t\|^2 \leq \Delta + \frac{L}{2}\|p^t\|^2 = \frac{\delta}{2}\Gamma(\delta), \quad \text{for all } t = 1, \ldots, R.$$

Since the random direction is generated by (8.11), Proposition 8.4.2 implies that

$$\|g(x^t)\|_* = \|g(x^t)\|_*\|p^t\|/\delta \leq \left(2\sqrt{cn}|g(x^t)^T p^t|\right)/\delta \leq \sqrt{cn}\Gamma(\delta), \quad \text{for all } t = 1, \ldots, R,$$

holds with probability $\dfrac{1}{2}$ or more. Therefore $\|g(x^t)\|_* \leq \sqrt{cn}\Gamma(\delta)$ fails with a probability $\Pr_t < \dfrac{1}{2}$ for all $t = 1, \ldots, R$. Therefore, the probability Pr that (8.25) holds for at least one of the gradients $g = g(x^t)$ ($t \in \{1, \ldots, R\}$) is $\Pr = 1 - \prod_{t=1}^{R-1} \Pr_t \geq 1 - 2^{-R}$. □

8.6 A randomized descent algorithm for BBO

In this section, we first consider a fixed decrease search for which an upper bound of the exact gradient norm for at least one of the points generated by the **extrapolationStep** or of the total number of function evaluations is found. Then the primary version of our algorithm is given.

8.6.1 Probing for fixed decrease

Based on the preceding results, we introduce the basic version of a fixed decrease search algorithm, called **FDS-basic**, whose goal is to use repeated calls to the randomized line search **RLS** to find a sufficient gain each time by a multiple of Δ.

FDS-basic takes `xm` and `fm` the old best point `xm` and its function value `fm`, the threshold for good improvement $\Delta > 0$, the old approximation $\lambda \geq 0$ for the Lipschitz constant L, maximum number of function evaluations `nfmax`, and the list of extrapolation step sizes **a** as input and uses the following tuning parameters:
$\gamma^e > 1$ (the factor for extrapolation test),
$E \geq 1$ (maximum number of extrapolations),
$\delta^{\min}/\delta^{\max}$ (minimum/maximum norm of trial steps),
γ^λ (factor for adjusting δ),
$\alpha^{\min} \in (0,1)$ (minimum threshold for step sizes),
$R > 0$ (the number of random search directions).
It returns a new best point `xm` and its function value `fm`, a new approximation $\lambda \geq 0$ for the Lipschitz constant L, and an updated list **a** of extrapolation step sizes as output.

8.6.1 Algorithm. (**FDS-basic, a basic fixed decrease search**)
while `good` is true **do**
(FDS_0) Perform **RLS** to hopefully get sufficient gains resulting in `good`.
(FDS_1) If `nfmax` is reached, **FDS-basic** ends.
end while

8.6.2 Theorem. *Assume that* (BBO_1) *and* (BBO_2) *hold and denote by* f^0 *the initial value of* f. *Then:*
(i) The number of function evaluations of **FDS-basic** *is bounded by*

$$2R + (2R+1)\frac{f^0 - \widehat{f}}{\alpha^{\min}\Delta},$$

where $\widehat{f} := \inf_k f^k$ *is finite due to* (BBO_1) *and* (BBO_2) *and* $\alpha^{\min} \in (0,1)$.
(ii) Denote by K_f *the number of calls to* **RLS** *by* **FDS-basic** *and assume that*

$$0 < \eta < 1, \quad R := \lceil \log_2 \eta^{-1}(K_f + 1) \rceil, \quad 0 < \delta^{\min} < \delta^{\max} < \infty.$$

Then **FDS-basic** *finds a point* x, *with probability* $\geq 1 - \eta$, *satisfying*

$$\|g(x)\|_* \leq \sqrt{cn} \min_{t=0:K_f} \Gamma(\delta^t) \leq \sqrt{cn}\Big(L\delta^{\min} + \sqrt{L'\Delta} + \frac{2\Delta}{\delta^{\max}}\Big). \tag{8.26}$$

Here c is the constant from Proposition 8.4.2 and, if λ^0 denotes the value of λ before the first execution of **FDS-basic**,

$$L' := \frac{L^2\gamma^\delta}{\lambda^0} + 4L + 4\frac{\lambda^0 + L}{\gamma^\delta} \quad \textit{with } \gamma^\delta > 0. \tag{8.27}$$

Proof. By (BBO_2), $\widehat{f}$ is finite. Denote by f^{k+1} the result of the $(k+1)$th execution of **FDS-basic**. In the worst case in each iteration $\ell \in \{1, \cdots, k\}$ of **FDS-basic** a sufficient gain is found, i.e., the condition

$$f^\ell \le f^{\ell-1} - \min(1, (\alpha^e)^\ell)\Delta \text{ for } \ell \in \{1, \cdots, k\}$$

holds. But in the $(k+1)$th iteration **FDS-basic** cannot find any sufficient gain and ends. We then conclude that

$$\widehat{f} \le f^k \le f^0 - \sum_{i=1}^{k} \min(\alpha_i^e, 1)\Delta \le f^0 - k\min(\alpha^{\min}, 1)\Delta = f^0 - k\alpha^{\min}\Delta$$

by (8.23), so that $k \le (f^0 - \widehat{f})/(\alpha^{\min}\Delta)$.

Since a sufficient gain is found in each iteration $\ell = 1, \cdots, k$, $2R+1$ function evaluations are used. But in the $(k+1)$th iteration, $2R$ function evaluations are used since there is no sufficient gain. Hence (i) follows.

(ii) K_f is finite due to (i) and we have $2^{-R} \le \eta/(K_f+1)$. Hence by Theorem 8.5.3 with probability $\ge 1 - (K_f+1)2^{-R} \ge 1 - \eta$

$$\|g\|_* \le \sqrt{cn} \min_{t=0:K_f} \Gamma(\delta^t)$$

holds. Thus it is sufficient to show that

$$\Gamma(\delta) \le L\delta^{\min} + \sqrt{L'\Delta} + \frac{2\Delta}{\delta^{\max}}. \tag{8.28}$$

By the definition of δ in (8.13), we have one of the following three cases:

Case 1: $\delta = \sqrt{\dfrac{\gamma^\delta\Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta + \frac{2\Delta}{\delta} = L\sqrt{\frac{\gamma^\delta\Delta}{\lambda}} + 2\sqrt{\frac{\lambda\Delta}{\gamma^\delta}} = \Lambda\sqrt{\Delta},$$

where

$$\Lambda := L\sqrt{\frac{\gamma^\delta}{\lambda}} + 2\sqrt{\frac{\lambda}{\gamma^\delta}}. \tag{8.29}$$

Case 2: $\delta = \delta^{\min} \ge \sqrt{\dfrac{\gamma^\delta\Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta^{\min} + \frac{2\Delta}{\delta^{\min}} \le L\delta^{\min} + 2\sqrt{\frac{\lambda\Delta}{\gamma^\delta}} \le L\delta^{\min} + \Lambda\sqrt{\Delta}.$$

CASE 3: $\delta = \delta^{\max} \le \sqrt{\frac{\gamma^\delta \Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta^{\max} + \frac{2\Delta}{\delta^{\max}} \le L\sqrt{\frac{\gamma^\delta \Delta}{\lambda}} + \frac{2\Delta}{\delta^{\max}} \le \Lambda\sqrt{\Delta} + \frac{2\Delta}{\delta^{\max}}.$$

Thus in each case,

$$\Gamma(\delta) \le L\delta^{\min} + \Lambda\sqrt{\Delta} + \frac{2\Delta}{\delta^{\max}}.$$

Now (8.28) follows since by (8.16) and (8.29), $\Lambda^2 = \frac{L^2\gamma^\delta}{\lambda} + 4L + \frac{4\lambda}{\gamma^\delta} \le L'$. □

8.6.2 The basic VSBBO algorithm

We now have all ingredients to formulate the basic version of **Vienna randomized black box optimization** algorithm **VSBBO-basic**. It uses in each iteration the fixed decrease search algorithm to update the best point. If no sufficient gain is found in the corresponding **FDS-basic** call, Δ is reduced by a factor of Q. Once either Δ is below a minimum threshold, **VSBBO-basic** stops.

VSBBO-basic takes the initial point x^0, and maximum number of function evaluations `nfmax` as input and uses the following tuning parameters:
$\gamma^e > 1$ (the factor for extrapolation test),
$E \ge 1$ (maximum number of extrapolations),
$\delta^{\min}/\delta^{\max}$ (minimum/maximum norm of trial steps),
γ^λ (factor for adjusting δ),
$Q > 1$ (factor for adjusting Δ),
$\Delta^{\min}$ (minimal threshold for Δ),
$\Delta^{\max}$ (maximum threshold for Δ),
$\alpha^{\min} \in (0, 1)$ (minimum threshold for step sizes),
$\lambda^{\max}$ (the initial Lipschitz constant),
$R > 0$ (the number of random search directions).
It returns an optimum point `xm` and its function value `fm` as output.

8.6.3 Algorithm. (**VSBBO-basic, Vienna basic randomized BBO**)
(VSb_0) Set $\lambda^0 := \lambda^{\max}$, $\delta^0 := \delta^{\max}$, $\Delta^0 := \Delta^{\max}$, and `xm` $:= x^0$. Then compute `fm` $:= f($`xm`$)$.
for $k = 1, 2, 3, \cdots$ **do**
(VSb_1) Perform **FDS-basic** to hopefully find sufficient gains. If `nfmax` is reached, **VSBBO-basic** ends.
(VSb_2) If $\Delta^k \le \Delta^{\min}$, **VSBBO-basic** ends.
(VSb_3) Reduce $\Delta^{k+1} := \Delta^k/Q$.
end for

8.7 Complexity analysis of VSBBO

We now prove the complexity results for the nonconvex, convex and strongly convex objective functions.

8.7.1 The general (nonconvex) case

8.7.1 Proposition. *Assume that (*BBO_1*) and (*BBO_2*) hold and let* f^0 *be the initial function value. If* $\Delta^{\max} > 0$*, the number of function evaluations needed up to iteration* k *by* **VSBBO-basic***, started at* x^0*, is*

$$N^k \le 1 + 2R + (2R+1)\frac{f^0 - \widehat{f}}{\alpha^{\min}\Delta^{\max}}\frac{Q^k - 1}{Q-1},$$

where $\widehat{f} := \inf_{k\ge 0} f^k$ *is finite due to (*BBO_1*) and (*BBO_2*).*

Proof. By construction, the kth call to **FDS-basic** uses $\Delta^k = Q^{1-k}\Delta^{\max}$, hence needs by Theorem 8.6.2(i) at most

$$2R + (2R+1)\frac{f^0 - \widehat{f}}{\alpha^{\min}Q^{1-k}\Delta^{\max}}$$

function evaluations. Then, the total number of function evaluations up to iteration k is

$$N^k \le 1 + \sum_{j=1}^{k}\Big(2R + (2R+1)\frac{f^0 - \widehat{f}}{\alpha^{\min}Q^{1-j}\Delta^{\max}}\Big) = 1 + 2Rk + (2R+1)\frac{f^0 - \widehat{f}}{\alpha^{\min}\Delta^{\max}}\frac{Q^k - 1}{Q-1}.$$

□

8.7.2 Theorem. *Assume that (*BBO_1*) and (*BBO_2*) hold and the parameters*

$$0 < \eta < 1,\ \Delta_{\max} > 0,\ \delta_{\max} > 0,\ \varepsilon > 0$$

are given. If the parameters are chosen such that

$$\Delta^{\min} := \Theta(\varepsilon^2/n), \tag{8.30}$$

$$K := \Big\lceil \frac{\log(\Delta^{\max}/\Delta^{\min})}{\log Q} \Big\rceil, \tag{8.31}$$

$$R := \Big\lceil \log_2 \frac{K+1}{\eta} \Big\rceil, \tag{8.32}$$

$$\delta^{\min} := \mathcal{O}(\varepsilon/\sqrt{n}), \tag{8.33}$$

then **VSBBO-basic** *finds after at most* $\mathcal{O}(nR\varepsilon^{-2})$ *function evaluations with probability* $\ge 1-\eta$ *a point* x *with*

$$\|g(x)\|_* = \mathcal{O}(\varepsilon). \tag{8.34}$$

Here c *comes from Proposition 8.4.2.*

Proof. By the rule of updating Δ in **VSBBO-basic**, $\Delta^k = Q^{1-k}\Delta^{\max} \le \Delta^{\min}$ for $k \ge K$. Hence at most K steps of **FDS-basic** are performed. By (8.32), we have $\eta_1 = 2^{-R} \le \eta/(K+1)$. Thus by Theorem 8.6.2(ii), we have, with probability $\ge 1-(K+1)\eta_1 \ge 1-\eta$, for at least one of the function values encountered,

$$\|g\|_* \le \min_{j=0:K}\Gamma(\delta^j) \le \sqrt{cn}\Big(L\delta^{\min} + \sqrt{L'\Delta^{\min}} + \frac{2\Delta^{\min}}{\delta^{\max}}\Big) = \mathcal{O}(\varepsilon) \tag{8.35}$$

by (8.30). Moreover, from Proposition 8.7.1 and by setting (8.30) in (8.31), we conclude that

$$N^K \quad \leq \quad 1 + 2RK + (2R+1)\frac{f^0 - \widehat{f}}{\alpha^{\min}\Delta^{\max}}\frac{Q^K - 1}{Q-1} = \mathcal{O}\Big(nR\varepsilon^{-2}\Big).$$

□

Choosing $\Delta^{\min} = \mathcal{O}(\varepsilon^2/n)$ with (8.30) is possible, and k, R, $\delta^{\min}$ can clearly be chosen to satisfy (8.31)–(8.33) and displays

$$K = \mathcal{O}(\log \frac{n}{\varepsilon^2}), \quad R = \mathcal{O}(\log \frac{n}{\varepsilon^2} + \log \eta^{-1}).$$

8.7.2 The convex case

8.7.3 Theorem. *Let f be convex on $\mathcal{L}(x^0)$ and assume that (*BBO_1*) and (*BBO_2*) hold. Given $0 < \eta < 1$, for any $\varepsilon > 0$, if* (8.32)–(8.33) *hold then* **VSBBO-basic** *finds after at most $\mathcal{O}\Big(nR\varepsilon^{-1}\Big)$ function evaluations with probability $\geq 1 - \eta$ a point x satisfying* (8.34) *and*

$$f(x) - \widehat{f} = \mathcal{O}(\varepsilon r^0), \tag{8.36}$$

where r^0 is given by (8.9) *and $\widehat{f}$ comes from Proposition 10.3.3.*

Proof. By (BBO_2), f has a global minimizer $\widehat{x}$ and $r^0 < \infty$. By Theorem 8.7.2, at most K steps of **FDS-basic** are performed. Let x^k be the result of the kth execution of **VSBBO-basic** satisfying (8.34); hence $k \leq K$. The convex case is characterized by (8.6), so that

$$\widehat{f} \geq f^k + (g^k)^T(\widehat{x} - x^k).$$

We know from Theorem 8.7.2 that, with probability $\geq 1 - \eta$, (8.34) holds and hence

$$f^k - f^{k+1} \leq f^k - \widehat{f} \leq (g^k)^T(x^k - \widehat{x}) \leq \|g^k\|_* \|x^k - \widehat{x}\| = \mathcal{O}(r^0\varepsilon) \tag{8.37}$$

by (8.9), resulting in (8.36).

Before **FDS-basic** is terminated, each iteration of **FDS-basic** has a sufficient gain, resulting in a finite sequence f^j $(j = 1, \ldots, K)$ of good function values. Then we conclude from (8.30), (8.31), and (8.37) that

$$\begin{aligned}\sum_{\ell=0}^{K} \frac{f^\ell - f^{\ell+1}}{\alpha^{\min}\Delta^\ell} &= \frac{\mathcal{O}(r^0\varepsilon)}{\alpha^{\min}} \sum_{\ell=0}^{K} (\Delta^\ell)^{-1} \leq \frac{\mathcal{O}(r^0\varepsilon)}{\alpha^{\min}\Delta_{\max}} \sum_{\ell=0}^{K} Q^{\ell-1} \\ &= \mathcal{O}(\varepsilon)\frac{Q^K - 1}{Q-1} = \mathcal{O}(\varepsilon)\mathcal{O}(n\varepsilon^{-2}) = \mathcal{O}(n\varepsilon^{-1})\end{aligned}$$

so that the bound for the number of function evaluations is

$$N^K \leq 1 + 2RK + (2R+1)\sum_{\ell=0}^{K} \frac{f^\ell - f^{\ell+1}}{\alpha^{\min}\Delta^\ell} = \mathcal{O}(nR\varepsilon^{-1})$$

by Theorem 8.6.2. □

8.7.3 The strongly convex case

8.7.4 Theorem. *Let f be convex on $\mathcal{L}(x^0)$ and assume that (*BBO_1*) and (*BBO_2*) hold. Under the assumptions of Theorem 8.7.2,* **VSBBO-basic** *finds after at most*

$$\mathcal{O}\Big(R\log n\varepsilon^{-1}\Big)$$

function evaluations with probability $\geq 1-\eta$ a point x satisfying (8.34) *and*

$$f(x)-\widehat{f}=\mathcal{O}\Big(\frac{\varepsilon^2}{2\sigma}\Big),\quad \|x-\widehat{x}\|=\mathcal{O}\Big(\frac{\varepsilon}{\sigma^2}\Big), \tag{8.38}$$

where $\widehat{f}$ comes from Proposition 8.7.1.

Proof. By Theorem 8.7.2, at most K steps of **FDS-basic** are performed. Let x^k be the results of the kth execution of **VSBBO** satisfying (8.34); hence $k\leq K$. The strongly convex case is characterized by (8.7), so that f has a global minimizer $\widehat{x}$ and

$$f(y)\geq f(x)+g(x)^T(y-x)+\frac{1}{2}\sigma\|y-x\|^2$$

for any x and y in $\mathcal{L}(x^0)$. For fixed x, the right-hand side of this inequality is a convex quadratic function of y, minimal when its gradient vanishes. By (8.1), this is the case iff y_i takes the value $x_i-\dfrac{s_i}{\sigma}g_i(x)$ for $i=1,\cdots,n$, and we conclude that $f(y)\geq f(x)-\dfrac{1}{2\sigma}\|g(x)\|_*^2$ for $y\in\mathcal{L}(x^0)$. Therefore

$$\widehat{f}\geq f(x)-\frac{1}{2\sigma}\|g(x)\|_*^2. \tag{8.39}$$

The replacement of x by x^k in (8.39) and (8.34) gives, with probability $\geq 1-\eta$,

$$f^k-f^{k+1}\leq f^k-\widehat{f}\leq\frac{\|g^k\|_*^2}{2\sigma}=\mathcal{O}\Big(\frac{\varepsilon^2}{2\sigma}\Big). \tag{8.40}$$

Since the gradient vanishes at the optimal point, we get from Theorem 8.7.2 and (8.40)

$$\|\widehat{x}-x^k\|^2\leq\frac{2}{\sigma}(f^k-\widehat{f})=\mathcal{O}\Big(\frac{\varepsilon}{\sigma^2}\Big) \tag{8.41}$$

with probability $\geq 1-\eta$.

Before **FDS-basic** is terminated, each iteration of **FDS-basic** has a sufficient gain, resulting in a finite sequence f^j $(j=1,\ldots,K)$ of good function values. Then we conclude from (8.30), (8.31), and (8.40) that

$$\begin{aligned}\sum_{\ell=0}^{K}\frac{f^\ell-f^{\ell+1}}{\alpha^{\min}\Delta^\ell} &\leq \frac{\mathcal{O}\Big(\frac{\varepsilon^2}{2\sigma}\Big)}{\alpha^{\min}}\sum_{\ell=0}^{K}(\Delta^\ell)^{-1}\leq\frac{\mathcal{O}(\varepsilon^2)}{\alpha^{\min}\Delta_{\max}}\sum_{\ell=0}^{K}Q^{\ell-1}\\ &= \mathcal{O}(\varepsilon^2)\frac{Q^K-1}{Q-1}=\mathcal{O}(\varepsilon^2)\mathcal{O}(n\varepsilon^{-2})=\mathcal{O}(n)\end{aligned}$$

Now Theorem 8.6.2 implies

$$N^K\leq 1+2RK+(2R+1)\sum_{\ell=0}^{K}\frac{f^\ell-f^{\ell+1}}{\alpha^{\min}\Delta^\ell}=\mathcal{O}(R\log n\varepsilon^{-1}).$$

□

8.8 Some new heuristic techniques

In this section, we describe several heuristic techniques that improve the basic Algorithm 8.6.3. While only convergence to a local minimizer is guaranteed, the addition of these heuristic techniques results in an efficient global solver. Indeed, in our comprehensive numerical experiment, reported in Section 8.10, the resulting **VSBBO** algorithm found in most cases either a global minimizer or, where this could not be checked, at least a point of similar quality with the best competitive global solvers.

More specifically, we discuss the occasional use of alternative search directions (two cumulative directions and a random subspace direction) and heuristics for estimating key parameters unspecified by the general theory – the initial desired gain, the Lipschitz constant, and the scaling vector. Moreover, we discuss the use of approximate gradients estimated by finite differences with steps extracted from the extrapolation steps. In Section 6, we combine Algorithm 3.3 with these heuristic techniques, resulting in the global solver **VSBBO**.

8.8.1 Cumulative directions

We consider two possibilities to accumulate past directional information into a cumulative search direction:
(i) With `xm` and `fm` defined in Section 8.5.1 the first cumulative direction is model independent, computed by

$$p = \texttt{xm} - x^{\text{init}}, \tag{8.42}$$

where x^{init} the initial point of the current improved version of **RLS**. Here the idea is that many small improvement steps accumulate to a direction pointing from the starting point into a valley, so that more progress can be expected by going further into this cumulative direction.
(ii) The second cumulative direction assumes a separable quadratic model of the form

$$f\Big(\texttt{xm} + \sum_{i\in I} \alpha_i p_i\Big) \approx \texttt{fm} - \sum_{i\in I} e_i(\alpha_i) \tag{8.43}$$

with quadratic univariate functions $e_i(\alpha)$ vanishing at $\alpha = 0$. Here I is the set of directions polled at least twice, and p_i is the corresponding direction as rescaled by an improved version of **RLS**.

By construction, we have for any $i \in I$ three function values at equispaced arguments. We write the quadratic interpolant as

$$f(\texttt{xm} + \alpha p) = \texttt{fm} - \frac{\alpha}{2} d + \frac{\alpha^2}{2} h = \texttt{fm} - e(\alpha),$$

where $e(\alpha) := \dfrac{\alpha}{2}(d - \alpha h)$. Let us recall the function values `fl`, `fm`, and `fr` satisfying either (8.19) or (8.21). If `fr` $<$ `fm`, the last evaluated point was the best one, so `fr` $\le \min($`fl`, `fm`$)$. In this case, (8.21) holds and we have

$$d := 4\texttt{fm} - 3\texttt{fr} - \texttt{fl}, \tag{8.44}$$

and

$$h := \texttt{fr} + \texttt{fl} - 2\texttt{fm}. \tag{8.45}$$

Otherwise, the last evaluated point was not the best one, so $\texttt{fm} \leq \min(\texttt{fl}, \texttt{fr})$. In this case, (8.19) holds and we compute d by

$$d := \texttt{fl} - \texttt{fr} \tag{8.46}$$

and h by (8.45).

Given the tuning parameter $a > 0$, the minimizer of the quadratic interpolant restricted to the interval $[-a, a]$ is

$$\alpha := \begin{cases} a & \text{if } d \geq 0, \\ -a & \text{if } d < 0 \end{cases} \tag{8.47}$$

in case $h \leq 0$. Otherwise, we have

$$\alpha := \begin{cases} \min(a, d/2h) & \text{if } d \geq 0, \\ \max(-a, d/2h) & \text{if } d < 0. \end{cases} \tag{8.48}$$

Assuming the validity of the quadratic model (8.43), we find the model optimizer by additively accumulating the estimated steps αp and gains e into a cumulative step q with anticipated gain r.

8.8.2 Random subspace directions

When sufficient gains are found, the trial points are accepted as the new best points and saved in X and their function values are saved in F. Denote by m^s the maximum number of points saved in X and by b the index of newest best point.

Random subspace directions point into the low-dimensional affine subspace spanned by a number of good points kept from previous iterations. They are computed by

$$\alpha^{\text{rand}} := \text{rand}(m^s - 1, 1) - 0.5, \quad \alpha^{\text{rand}} := \alpha^{\text{rand}} / \|\alpha^{\text{rand}}\|, \quad p := \sum_{i=1, i \neq b}^{m^s} \alpha_i^{\text{rand}} (X_{:i} - X_{:b}). \tag{8.49}$$

8.8.3 Choosing the initial Δ

First of all, we compute

$$\texttt{dF} = \underset{i=1:m^s}{\text{median}} |F_i - F_b|. \tag{8.50}$$

Then if $\texttt{dF}$ is not zero we estimate the initial desired gain

$$\Delta := \gamma^{\max} \min(\texttt{dF}, 1) \tag{8.51}$$

where $\gamma^{\max} > 0$ is a tuning parameter. Otherwise $\Delta := \Delta^{\max}$, where $\Delta^{\max} > 0$ is the initial gain.

8.8.4 Choosing the initial λ

The initial value for λ is $\lambda^{\max}$ which is the tuning parameter, however, it is updated by (8.20) provided that the best point is updated by **extrapolationStep**. Our achievement is to estimate it by a heuristic formula based on the previous best function values restored in F.

Let λ^{old} be the old estimation for the Lipschitz constant and $\gamma^\lambda > 0$ be a factor for adjusting λ. We compute λ by

$$\lambda := \begin{cases} \dfrac{\gamma^\lambda}{\sqrt{n}} & \text{if } \mathtt{dF} = 0 \text{ and } \lambda^{\text{old}} = 0, \\ \lambda^{\text{old}} & \text{if } \mathtt{dF} = 0 \text{ and } \lambda^{\text{old}} \neq 0, \\ \gamma^\lambda \sqrt{\dfrac{\mathtt{dF}}{n}} & \text{otherwise,} \end{cases} \tag{8.52}$$

where `dF` is computed by (8.50).

8.8.5 Choosing the scaling vector

The idea is to estimate a sensible scaling vector s with the goal of adjusting the search direction scaled by (8.12). We compute

$$\mathtt{dX}_{:i} = X_{:i} - X_{:b} \quad \text{for all } i = 1, \cdots, m^s.$$

and estimate the scaling vector

$$s := \sup_{i=1:m^s} (\mathtt{dX}_{:i}), \quad J = \{i \mid s_i = 0\}, \quad s_J = 1. \tag{8.53}$$

Finally, the formula (8.12) is rewritten as

$$p = s \circ p \quad \text{and} \quad p = p(\delta/\|p\|) \tag{8.54}$$

where $\circ$ denotes componentwise multiplication and δ is computed by (8.13).

8.8.6 Estimating the gradient

Denote by $\widetilde{g}$ the estimated gradient. With `xm` and `fm` defined in Section 8.5.1, finite difference quasi-Newton methods estimate the gradient with components

$$\widetilde{g}_i := \frac{f(\mathtt{xm} + \alpha_i e_i) - \mathtt{fm}}{\alpha_i},$$

where e_i is the ith coordinate vector. The most popular choice for α is the constant choice

$$\alpha_i := \max\{1, \|\mathtt{xm}\|_\infty\}\sqrt{\varepsilon_m}, \tag{8.55}$$

where ε_m is the machine precision; another choice for α is made now. After generating each coordinate search direction, we estimate each component of the gradient in a way that is a little different from the forward finite difference approach. The step size generated by **extrapolationStep** is used instead of the general choice (8.55). The reason of this change is that we don't need to estimate the gradient by another algorithm due to the additional cost. Let describe how to compute the gradient. If **extrapolationStep** cannot find a sufficient gain in the tth iteration ($\mathtt{ne} = 0$), `fr` is computed and $\mathbf{a}^t$ is unchanged. Given the old best point $\mathtt{fm}^{\text{old}}$, the tth component of the gradient is computed by

$$\widetilde{g}_t = (\mathtt{fr} - \mathtt{fm}^{\text{old}})/\mathbf{a}^t; \tag{8.56}$$

otherwise, it is computed by

$$\widetilde{g}_t = (\mathtt{fm} - \mathtt{fm}^{\text{old}})/\mathbf{a}^t, \tag{8.57}$$

where both $\mathtt{fm}^{\text{old}}$ and $\mathbf{a}^t$ are updated by **extrapolationStep**. We will add later this computation to an improved version of **RLS**.

8.9 The implemented version of VSBBO

In this section, we discuss the implementation of **VSBBO** with the improvements which are of a heuristic nature, very important for efficiency, and do not change the order of our complexity results. Thus **VSBBO** gives the same order of complexity as the one by Bergou et al. but with a guarantee that holds with probability arbitrarily close to 1; see Table 8.3. Numerical results in Section 8.10 show that, in practice, **VSBBO** matches the quality of all state of the art algorithms for unconstrained black box optimization problems. **VSBBO** is implemented in Matlab; the source code is obtainable from

`http://www.mat.univie.ac.at/~neum/software/VSBBO`.

It includes many subalgorithms described earlier in Sections 8.5.2–8.6.3. The others have a simpler structure; hence we skip their details (which can be found at the above website) and only state their goals and those tuning parameters which have not been defined yet. These subalgorithms are **identifyDir**, **lbfgsDir**, **updateSY**, **updateXF**, **updateCum**, **enforceAngle**, **direction**, **MLS**, **FDS**, and **setScales**, which are described below.

Before we compute the direction, the type of direction needs to be identified by **identifyDir**. **VSBBO** calls **direction** to generate 5 kinds of direction vectors: coordinate directions, limited memory quasi-Newton directions, random subspace directions, random directions, and cumulative directions, as pointed out earlier in more detail in Subsection 8.8:
• Coordinate directions are the coordinate axes e_i, $i = 1, \cdots, n$, in a cyclic fashion. The coordinate direction values enhance the global search properties, decreasing on average with the number of function evaluations used. Moreover, they are used to estimate the gradient by the finite difference approach.
• Limited memory quasi-Newton directions are computed by **lbfgsDir** (standard limited memory BFGS direction [140]). Due to rounding errors, the computed direction may not satisfy the angle condition (8.17); hence it needs to be modified by **enforceAngle** discussed in [118].
• **updateSY**, **updateXF**, and **updateCum** are auxiliary routines for updating the data needed for calculating, limited memory quasi-Newton steps, random subspace steps and cumulative steps, respectively.
These subalgorithms have the following tuning parameters:
`cum` (the cumulative step type),
$\texttt{ms}^{\max}$ (the maximum number of best points kept),
$\texttt{mq}^{\max}$ (the memory for L-BFGS approximation),
$0 < \gamma^w < 1$ and $0 < \gamma^{\mathrm{a}} < 1$ (tiny parameters for the angle condition),
`scSub` (random subspace direction scale?),
`scCum` (cumulative direction scale?).

We denote by C the number of coordinate directions, by R the number of random directions, and by S the number of subspace directions in each repeated call to a multi-line search algorithm – an improved version of **RLS**, called **MLS**; once the cumulative direction and L-BFGS direction are computed. Here T is the number of 5 kinds of directions satisfying $1 \le T \le C + S + R + 2$. C, R, and S are the tuning parameters.

Denote by **FDS** the improved version of **FDS-basic** and by **MLS**. Both **setScales** and **FDS** work by making repeated calls to **MLS**. **MLS** polls in a number of suitable chosen directions (implemented by **direction**) in a line search fashion a few objective function values each in the hope of reducing the objective by more than a multiple of Δ. Schematically, it works as follows:

• At first, at most C iterations with coordinate directions are used. They are used to estimate the gradient.
• Then, the L-BFGS direction is used only once since the gradient has been estimated by the finite difference technique using the coordinate directions.
• Next, except in the final iteration, at most S iterations with subspace directions are used. These directions are very useful, especially after performing the coordinate directions and L-BFGS, due to our numerical experiments.
• After generating $T-1$ directions without finding a sufficient gain, a cumulative direction is used as final, Tth direction in the hope of finding a model-based gain.

MLS calls an improved version of **extrapolationStep**, which is the same as **extrapolationStep**, except that it updates the cumulative step q and the cumulative gain r by **updateCum** whenever the second cumulative direction is used.

VSBBO initially calls the algorithm **setScales** to estimate a good scaling of norms, step lengths, and related control parameters. Then it uses in each iteration **FDS**, aimed at repeatedly reducing the function value by an amount of at least a multiple of Δ to update the best point. If no sufficient gain is found in a call to **FDS**, Δ is reduced by a factor of Q. Once Δ is below a minimum threshold or `nfmax` is reached, **VSBBO** stops.

An important question is the ordering of the search directions. In Section 8.10, it will be shown that after coordinate directions using the random subspace direction is very preferable. Changing the ordering other direction is not very effective on the efficiency of our algorithm. However, using all directions is useful, especially using coordinate directions increase the efficiency of **VSBBO** provided that some random and random subspace directions are tried after it.

The statement (i) of Theorem 8.5.3 remains valid when R is replaced by T, and the statement (ii) of it remains valid with probability $\geq 1-2^{C+S+2-T} = 1-2^{-R}$.

Let T^0 be the maximal number of multi-line searches in **setScales** as the tuning parameter. Then, **setScales** uses $(2T+1)T^0$ function evaluations which does not affect on the order of the complexity bounds. Theorems 8.7.2, 8.7.3, and 8.7.4 are valid with probability $\geq 1-2^{2+C+S-T} = 1-2^{-R}$, where 5 kinds of directions are used. Given the tuning parameter `alg` $\in \{0,1,3,4,5\}$ (algorithm type), we now discuss the factor of bounds depending on the number of search directions used in **MLS**. We would have the following cases:
• In the first case (`alg` $= 0$), $T = R < n$ random directions are used. Then complexity results considered as Table 8.3 are valid. This variant of **VSBBO** is denoted by **VSBBO-basic1**.
• In the second case (`alg` $= 1$), $T = R \geq n$ random directions are used. Then complexity results considered as Table 8.3 are valid but with the factor of n^2. This variant of **VSBBO** is denoted by **VSBBO-basic2**.
• In the third case (`alg` $= 2$), random, random subspace, and cumulative directions are used whose total number is $T = S + R + 1 < n$. The complexity results considered as Table 8.3 are valid. This variant of **VSBBO** is denoted by **VSBBO-C-Q**, ignoring the coordinate and limited memory quasi Newton directions.
• In the fourth case (`alg` $= 3$), coordinate, random, random subspace, and cumulative directions are used whose total number is $T = C + S + R + 1 > n$. The complexity results considered as Table 8.3 are valid but with the factor of n^2. This variant of **VSBBO** is denoted by **VSBBO-Q**, ignoring the limited memory quasi Newton directions.
• In the fifth case (`alg` $= 4$), only subspace directions are ignored. The total number of directions used is $T = C + R + 2 > n$; hence the complexity results are valid but with the factor n^2. This variant of **VSBBO** is denoted by **VSBBO-S**.

• In the sixth case (`alg` = 5), coordinate, L-BFGS, random, random subspace, and cumulative directions are used successively whose total number is $T = C + S + R + 2 > n$. The complexity results considered as Table 8.3 are valid but with the factor of n^2. This variant of **VSBBO** is the default version.

This defines six versions of **VSBBO**, the full algorithm and 5 simplified variants. In Section 8.10, we compare them and show that each simplification degrades the algorithm. This means that all heuristic components of **VSBBO** are necessary for the best performance.

8.10 Numerical results

In this section, we compare 28 competitive solvers from the literature on all 549 unconstrained problems from the `CUTEst` test problems for optimization with up to 5000 variables, in case of variable dimension problems for all allowed dimensions in this range. Default parameters for **VSBBO** can be found in Subsections 8.11.1 and for other solvers in Subsection 8.11.2. For problems in dimension $n > 100$, only the most robust and fast solvers were compared. To avoid guessing the solution of toy problems with a simple solution (such as all zero or all one), we shifted the arguments by (5.1) for all $i = 1, \ldots, n$.

We limited the budget available for each solver by allowing at most

$$\texttt{secmax} := \begin{cases} 180 & \text{if } 1 \le n \le 100, \\ 700 & \text{if } 101 \le n \le 1000 \\ 1500 & \text{if } 1001 \le n \le 5000 \end{cases}$$

seconds of run time and at most

$$\texttt{nfmax} := \begin{cases} 2n^2 + 1000n + 5000 & \text{if } 1 \le n \le 100, \\ 100n & \text{if } 101 \le n \le 5000 \end{cases}$$

function evaluations for a problem with n variables.

As discussed earlier in Subsection 5.4, a problem with dimension n is considered solved by the solver *so* if the target accuracy satisfies

$$q^{so} \le \begin{cases} 10^{-4} & \text{if } 1 \le n \le 100, \\ 10^{-3} & \text{if } 101 \le n \le 5000. \end{cases}$$

The `nf` efficiency and `msec` efficiency have been defined in Subsection 5.4. They are used as the two cost measures for performance profile and plot discussed in Subsection 5.5. Performance profiles and plots are shown in Figures 8.2–8.9 and the other results are give all results in Tables 8.4–8.7.

8.10.1 Results for small dimensions ($n \le 20$)

The low dimensional test problems ($n \le 20$) unsolved by all 28 solvers are HATFLDFL, FLETCBV3, and FLETCHBV. In summary, the results for the remaining problems are summarized in Table 8.4.

Table 8.4: The summary results for small dimensions $n \leq 20$

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq 2n^2 + 1000n + 5000$										
174 of 177 problems solved										mean efficiency in %
dim∈[1,20]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
MCS	**mcs**	162	22	21	428	3	0	12	41	13
GCES	**gecs**	159	5	3	1684	1	0	17	29	9
VSBBO	**vsbb**	159	3	2	325	18	0	0	25	18
BCDFO	**bcd**	159	41	37	3796	0	1	17	53	21
NMSMAX	**nmsm**	157	10	9	193	12	0	8	35	41
NELDER	**neld**	157	14	11	190	0	0	20	36	42
PSM	**psm**	156	7	6	1222	10	2	9	37	14
SDBOX	**sdb**	144	5	4	205	33	0	0	25	43
FMINUNC	**func**	142	54	51	84	0	0	35	52	53
CMAES	**cma**	142	3	2	457	35	0	0	10	10
BFO	**bfo**	140	1	1	269	0	0	37	19	19
SDS	**sds**	132	3	1	285	0	0	45	16	22
AHDS	**ahds**	130	2	0	283	12	0	35	11	14
ADSMAX	**adsm**	126	1	1	662	37	0	14	14	10
DSDS	**dsds**	125	2	0	345	12	0	40	8	11
MDSMAX	**mdsm**	123	3	2	318	53	0	1	12	18
PSWARM	**psw**	122	8	5	491	33	0	22	9	6
HOOKE	**hook**	120	0	0	196	13	0	44	18	24
DSPFD	**dspf**	113	4	4	995	0	0	64	15	11
FMINSEARCH	**fmin**	97	0	0	575	20	0	60	7	6
GLOBAL	**glo**	93	3	2	166	21	0	63	7	12
DE	**de**	84	0	0	769	68	0	25	0	2
DESTRESS	**dest**	80	0	0	284	33	0	64	6	8
STP-vs	**svs**	57	2	0	258	120	0	0	4	8
PSTP	**pst**	55	2	1	913	122	0	0	2	3
STP-fs	**sfs**	52	0	0	308	125	0	0	3	5
ACRS	**acr**	50	1	0	330	83	0	44	3	4
MDS	**mds**	13	3	1	133	96	0	68	2	3
166 of 177 problems solved										mean efficiency in %
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	159	36	31	325	18	0	0	66	60
VSBBO-Q	**vsbbq**	158	56	50	336	19	0	0	68	66
VSBBO-S	**vsbbs**	148	34	28	414	29	0	0	54	55
VSBBO-basic1	**vsbb1**	129	26	24	632	46	0	2	39	39
VSBBO-C-Q	**vsbcq**	128	18	14	831	46	0	3	32	28
VSBBO-basic2	**vsbb2**	127	13	10	636	47	0	3	36	39
137 of 177 problems solved										mean efficiency in %
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-basic1	**vsbb1**	129	59	58	632	46	0	2	55	56
DSPFD	**dspf**	113	64	63	995	0	0	64	49	43
STP-vs	**svs**	57	4	4	258	120	0	0	10	20
PSTP	**pst**	55	6	6	913	122	0	0	6	7
STP-fs	**sfs**	52	5	5	308	125	0	0	8	15

From Table 8.4 and Figures 8.2–8.3 (shown in Subsection 8.11.5), we see that on problems with few variables,
• **VSBBO** and **VSBBO-Q** have above average performance and can compete in robustness with the best solvers **MCS**, **GCES**, **NELDER**, and **BCDFO**.
• **VSBBO** and **VSBBO-Q** are the best solvers in terms of the number of solved problems and the `nf` efficiency, respectively in comparison with **VSBBO-basic1**, **VSBBO-basic2**, and **VSBBO-C-Q**. A question that may be asked here is why **VSBBO** and **VSBBO-Q** are the best. The answer is that using the coordinate directions is very effective provided that, after these directions, random subspace directions are tried. Note that **VSBBO-C-Q** uses the random subspace directions after performing random directions but it is weaker than **VSBBO** and **VSBBO-Q**.
• **VSBBO-basic1**, the basic version of **VSBBO**, for which complexity results considered as Table 8.3 are valid is more robust than solvers proposed by Gratton et al. [84] (**DSPFD**) and Bergou et al. [16] (**STP-vs**, **PSTP**, and **STP-fs**), discussed earlier in Section 8.3. As pointed out earlier, there is no plan to test whether the function value is decreased or not; this is a reason why three solvers proposed by Bergou et al. are numerically poor.

8.10.2 Results for medium dimensions ($21 \leq n \leq 100$)

We tested all 28 solvers on medium dimensional test problems ($21 \leq n \leq 100$). The problems unsolved by all solvers are NONMSQRT:49, CURLY10:100, NONMSQRT:100 and OSCIGRAD:100.

Table 8.5: The summary results for medium dimensions 21–100

stopping test: $q_f \le$ `0.0001`, `sec` $\le$ `180`, `nf` $\le 2n^2+1000n+5000$										
151 of 156 problems solved										mean efficiency in %
dim∈[21,100]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	133	11	11	2473	23	0	0	39	25
MCS	**mcs**	133	5	5	3852	3	0	20	22	14
FMINUNC	**func**	128	76	74	911	0	0	28	62	68
SDBOX	**sdb**	126	19	15	1615	30	0	0	37	45
ADSMAX	**adsm**	115	9	6	5021	36	0	5	25	14
CMAES	**cma**	101	1	0	14852	52	3	0	7	2
PSWARM	**psw**	100	3	2	11142	51	0	5	5	3
SDS	**sds**	97	2	0	3304	0	3	56	6	11
NMSMAX	**nmsm**	94	3	3	5214	60	2	0	13	10
MDSMAX	**mdsm**	90	1	0	4172	66	0	0	4	9
NELDER	**neld**	90	1	1	25525	5	29	32	13	4
DSPFD	**dspf**	89	8	8	21346	0	31	36	18	3
BFO	**bfo**	83	0	0	4781	0	4	69	5	5
HOOKE	**hook**	76	1	1	4616	11	0	69	11	7
DE	**de**	73	0	0	3670	48	0	35	1	4
GLOBAL	**glo**	36	1	1	1925	31	0	89	4	5
DESTRESS	**dest**	34	0	0	1757	75	0	47	1	2
PSM	**psm**	31	4	4	28431	0	125	0	8	1
DSDS	**dsds**	31	2	0	2190	89	4	32	3	4
AHDS	**ahds**	30	2	0	2503	113	4	9	1	2
GCES	**gecs**	27	0	0	42836	0	124	5	4	0
BCDFO	**bcd**	20	11	10	29203	0	135	1	7	1
STP-vs	**svs**	18	0	0	6906	137	1	0	0	0
PSTP	**pst**	17	2	1	4408	138	1	0	2	2
STP-fs	**sfs**	15	0	0	7302	140	1	0	0	1
FMINSEARCH	**fmin**	14	0	0	6433	136	2	4	0	0
MDS	**mds**	7	0	0	4691	146	2	1	0	0
ACRS	**acr**	6	0	0	27262	84	66	0	0	0
138 of 156 problems solved										mean efficiency in %
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-Q	**vsbbq**	135	45	41	2188	21	0	0	73	70
VSBBO	**vsbb**	133	40	35	2473	23	0	0	70	66
VSBBO-S	**vsbbs**	124	37	36	3478	32	0	0	59	62
VSBBO-basic1	**vsbb1**	94	6	5	5096	61	1	0	23	21
VSBBO-basic2	**vsbb2**	93	7	7	5156	63	0	0	23	23
VSBBO-C-Q	**vsbcq**	42	9	8	9087	113	1	0	10	10
100 of 156 problems solved										mean efficiency in %
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-basic1	**vsbb1**	94	20	18	5096	61	1	0	41	58
DSPFD	**dspf**	89	76	74	21346	0	31	36	53	18
STP-vs	**svs**	18	0	0	6906	137	1	0	1	4
PSTP	**pst**	17	5	5	4408	138	1	0	4	5
STP-fs	**sfs**	15	1	1	7302	140	1	0	1	3

Table 8.6: The summary results for for large dimensions 101–1000

stopping test:		$q_f \le$ `0.001`,		`sec` $\le$ `700`,		`nf` $\le$ `100*n`				
104 of 126 problems solved									mean efficiency in %	
dim∈[101,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	97	21	20	9471	28	1	0	44	30
SDBOX	**sdb**	95	9	8	5240	30	1	0	39	51
FMINUNC	**func**	75	45	42	2688	28	0	23	46	50
ADSMAX	**adsm**	72	25	24	11076	49	1	4	32	19
DSPFD	**dspf**	18	4	3	140555	0	71	37	6	1
MCS	**mcs**	17	7	4	24057	7	1	101	6	3
101 of 126 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	97	39	23	9471	28	1	0	67	60
VSBBO-S	**vsbbs**	94	40	28	8166	31	1	0	63	65
VSBBO-Q	**vsbbq**	94	38	24	9598	31	1	0	63	57
VSBBO-basic2	**vsbb2**	43	3	2	13033	82	1	0	13	12
VSBBO-basic1	**vsbb1**	42	6	5	13712	83	1	0	15	14
VSBBO-C-Q	**vsbcq**	22	3	2	12359	103	1	0	7	7
104 of 126 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	97	46	45	9471	28	1	0	55	42
FMINUNC	**func**	75	59	58	2688	28	0	23	54	56
43 of 126 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-basic1	**vsbb1**	42	29	28	13712	83	1	0	28	33
DSPFD	**dspf**	18	15	14	140555	0	71	37	13	2

From Table 8.5 and Figures 8.4–8.5 (shown in Subsection 8.11.5), we see that on problems with a medium number of variables,
• **VSBBO-Q** and **VSBBO** are outstanding in robustness, but **FMINUNC** is the best in terms of `nf` and `sec` efficiency.
• **VSBBO-Q** and **VSBBO** are the best version of **VSBBO** in terms of the number of solved problems, #100, !100, the `nf` and `msec` efficiencies, respectively.
• **VSBBO-basic1** is more robust than than solvers proposed by Gratton et al. [84] (**DSPFD**) and Bergou et al. [16] (**STP-vs**, **PSTP**, and **STP-fs**).

8.10.3 Results for large dimensions ($101 \le n \le 1000$)

We tested the 6 most robust solvers from Table 8.5 on large dimensional test problems ($101 \le n \le 1000$). A list of the problems unsolved by the 6 solvers are reported in Subsection 8.11.6.

Table 8.7: The summary results for very large dimensions 1001–5000

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 1500,		nf $\leq$ 100*n				
77 of 90 problems solved									mean efficiency in %	
dim∈[1001,5000]						# of anomalies			for cost measure	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
VSBBO	**vsbb**	74	18	18	74585	14	2	0	60	42
SDBOX	**sdb**	72	29	28	61545	13	5	0	54	57
FMINUNC	**func**	49	31	30	36280	30	0	11	47	44
75 of 90 problems solved									mean efficiency in %	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
VSBBO	**vsbb**	74	37	24	74585	14	2	0	75	74
VSBBO-Q	**vsbbq**	74	36	22	86023	14	2	0	75	74
VSBBO-S	**vsbbs**	73	23	11	96244	15	2	0	72	61
VSBBO-basic1	**vsbb1**	19	1	0	197927	67	4	0	6	5
VSBBO-basic2	**vsbb2**	17	1	0	163034	70	3	0	6	5
VSBBO-C-Q	**vsbbq**	11	2	1	299000	77	2	0	4	3

A comparison of 6 solvers in Table 8.6 and Figures 8.5–8.6 (shown in Subsection 8.11.5) shows that **VSBBO** is the most robust solver. This solver is better than **SDBOX** in terms of the `nf` efficiency and than **FMINUNC** in terms of the number of solved problems. **VSBBO-basic1**, the basic version of **VSBBO**, is more robust than **MCS** and **DSPFD**. Due to the use of L-BFGS direction, **VSBBO** is better than **VSBBO-Q** but the converse was true for small scale and medium problems. Moreover, **VSBBO-S** and **VSBBO-Q** have the same quality and are more efficient than **VSBBO-basic1**, **VSBBO-basic2**, and **VSBBO-C-Q**.

8.10.4 Results for very large dimensions ($1001 \leq n \leq 5000$)

We tested the 3 most robust solvers from Table 8.6 on very large dimensional test problems ($101 \leq n \leq 1000$). A list of problems unsolved by the three best solvers can be found in Subsection 8.11.7.

In Table 8.7 and Figures 8.8–8.9 (shown in Subsection 8.11.5), we give a comparison of best solvers **SDBOX**, **VSBBO**, and **FMINUNC** for very large dimensions 1001 up to 5000. The high quality of **VSBBO** is clearly visible. **VSBBO** is the best in terms of the `nf` efficiency, #100 and !100. **VSBBO** and **VSBBO-Q** have the same behaviour, are a little better than **VSBBO-S**, and are more efficient than **VSBBO-basic1**, **VSBBO-basic2**, and **VSBBO-C-Q**.

8.11 Additional material for VSBBO

This section discusses additional material for **VSBBO**.

8.11.1 Default parameters for VSBBO

For our tests we used the following parameter choices:

Common tuning parameters: $E = 10$; $\delta^{\min} = 10^{-4}\sqrt{n}$; $\delta^{\max} = 0.1\sqrt{n}$; $\Delta^{\min} = 0$; $\Delta^{\max} = 10^{-3}$; $\gamma^{\delta} = 10^{6}$; $\gamma^{e} = 2$; $Q = 2$; $\lambda^{\max} = 1$;
VSBBO-basic1: $R = \texttt{fix}(n/2) + 1$
VSBBO-basic2: $R = n$
VSBBO-C-Q: $\texttt{ms}^{\max} = 5$; $R = \texttt{fix}(n/2) + 1$; $S = \texttt{fix}(n/5)$; $T^0 = 2 * \texttt{ms}^{\max}$; $\texttt{scCum} = 0$; $\texttt{scSub} = 0$; $\gamma^{\max} = 10^{-3}$; $\texttt{cum} = 1$;
VSBBO-S: $C = n$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $\texttt{scCor} = 0$; $\texttt{cum} = 1$; $\gamma^{w} = \varepsilon_m$; $\gamma^{\mathrm{a}} = 10^{-20}$; $\texttt{mq}^{\max} = 5$
VSBBO-Q: $\texttt{ms}^{\max} = 5$; $C = n$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $S = \min(\texttt{fix}(n/10), 5)$; $\texttt{scCor} = 0$; $\texttt{scSub} = 0$; $\texttt{cum} = 1$; $\texttt{ms}^{\max} = 5$; $T^0 = 2 * \texttt{ms}^{\max}$; $\gamma^{\max} = 10^{-3}$;
VSBBO: $\texttt{ms}^{\max} = 5$; $\texttt{mq}^{\max} = 5$; $T^0 = 2 * \texttt{ms}^{\max}$; $C = n$; $S = \min(\texttt{fix}(n/10) + 1, 5)$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $\texttt{scCum} = 0$; $\texttt{scCor} = 0$; $\texttt{scSub} = 0$; $\texttt{cum} = 1$; $\gamma^{\max} = 10^{-3}$; $\gamma^{w} = \varepsilon_m$; $\gamma^{\mathrm{a}} = 10^{-20}$;

Table 8.8: The values of the tuning parameters

Although the best theoretical complexity is obtained for $\Omega(\log\log n)$, the best numerical result are obtained for much larger $R \sim n$.

$\Delta^{\min} = 0$ implies that the algorithm stops due to `nfmax` or `secmax`.

In recent years, there has been an increasing interest in finding the best tuning parameters configuration for derivative-free solvers with respect to a benchmark problem set; see, e.g., [7, 148, 147]. In Table 8.8, there are 7 integral, 2 binary, 2 ternary, and 12 continuous tuning parameters, giving a total of 23 parameters for tuning our algorithm. A small amount of tuning

was done by hand. Automatic tuning of **VSBBO** will be considered elsewhere.

8.11.2 Codes compared

• **UOBYQA**, available at

`http://mat.uc.pt/~zhang/software.html`,

is an algorithm that forms quadratic models by interpolation by Powell [149]. Tuning parameters are default.
• **STP-fs, STP-vs**, and **PSTP**, obtained from the authors of Bergou et al. [16], are three versions of a randomized direct search method with complexity guarantees.
• **BFO**, obtained from

`https://github.com/m01marpor/BFO`,

is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by Porcelli & Toint [147].
• **CMAES**, obtained from

`http://cma.gforge.inria.fr/count-cmaes-m.php?Down=cmaes.m`,

is the stochastic covariance matrix adaptation evolution strategy by Auger & Hansen [8]. We used **CMAES** with the tuning parameters

```
oCMAES.MaxFunEvals = nfmax,  oCMAES.DispFinal = 0,      oCMAES.DispModulo = 0,
oCMAES.LogModulo = 0,        oCMAES.SaveVariables = 0,  oCMAES.MaxIter = nfmax,
oCMAES.Restarts = 7.
```

• **GLOBAL**, obtained from

`http://www.mat.univie.ac.at/~neum/glopt/contrib/global.f90`,

is a stochastic multistart clustering global optimization method by Csendes et al. [44]. We used **GLOBAL** with the tuning parameters `oGLOBAL.MAXFNALL` = `nfmax`, `oGLOBAL.MAXFN` = `nfmax`/5, `oGLOBAL.DISPLAY` =‘`off`’, `oGLOBAL.N100` = 300,
`oGLOBAL.METHOD` =‘`unirandi`’, and `oGLOBAL.NG0` = 2.
• **DE**, obtained from

`http://www.icsi.berkeley.edu/~storn/code.html`,

is the stochastic differential evolution algorithm by Storn & Price [156].
• **MCS**, obtained from

`https://www.mat.univie.ac.at/~neum/software/mcs/`,

is the deterministic global optimization by multilevel coordinate search by Huyer & Neumaier [102]. We used **MCS** with the tuning parameters `iinit` = 1, `nfMCS` = `nfmax`, `smax` = $5n + 10$, `stop` = $3n$, `local` = 50, `gamma` = `eps`;
`hess` = `ones`(n, n), and `prt` = 0.

• **BCDFO**, obtained from Anke Troeltzsch (personal communication), is a deterministic model-based trust-region algorithm for derivative-free bound-constrained minimization by Gratton et al. [85].
• **PSM**, obtained from

`http://ferrari.dmat.fct.unl.pt/personal/alcustodio`,

is a deterministic pattern search method guided by simplex derivatives for use in derivative-free optimization proposed by Custódio & Vicente [48, 47].
• **FMINUNC**, obtained from the Matlab Optimization Toolbox at

`https://ch.mathworks.com/help/optim/ug/fminunc.html`,

is a deterministic quasi-Newton or trust-region algorithm. We use **FMINUNC** with the options set by `optimoptions` as follows:

```
opts = optimoptions(@fminunc),'Algorithm','quasi-newton', 'Display',
'Iter','MaxIter', Inf, 'MaxFunEvals', limits.nfmax, 'TolX', 0,'TolFun',
0,'ObjectiveLimit',-1e-50);
```

It is the standard quasi Newton method while finding step sizes by Wolfe condition.
• **FMINSEARCH**, obtained from the Matlab Optimization Toolbox at

`https://ch.mathworks.com/help/matlab/ref/fminsearch.html`,

is the deterministic Nelder-Mead simplex algorithm by Lagarias et al. [121]. We use **fminseach** with the options set by

```
opts = optimset('Display','Iter', 'MaxIter', Inf,'MaxFunEvals',
limits.nfmax,'TolX', 0, 'TolFun',0,'ObjectiveLimit',-1e-50);
```

• **GCES** is a globally convergence evolution strategy presented by Diouane et al. [59, 60]. The default parameters are used.
• **PSWARM**, obtained from

`http://www.norg.uminho.pt/aivaz`

is Particle swarm pattern search algorithm for global optimization presented by Vaz & Vicente [159].
• **MDS**, **NELDER** and **HOOKE**, obtained from

`https://ctk.math.ncsu.edu/matlab_darts.html`

are Multidirectional search, Nelder-Mead and Hooke-Jeeves algorithms, respectively, presented by Kelley [109]. The default parameters are used.
• **MDSMAX**, **NMSMAX**, and **ADSMAX**, obtained from

`http://www.ma.man.ac.uk/~higham/mctoolbox/`

are Multidirectional search, Nelder-Mead simplex and alternating directions method for direct search optimization algorithms, respectively, presented by Higham [99].
• **ACRS**, obtained from

`http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3`

is a global optimization algorithm presented by Brachetti et al. [27].
• **SDBOX**, obtained from

`http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3`

is a derivative-Free algorithm for bound constrained optimization problems presented by Lucidi & Sciandrone [126].
• **DSPFD**, available at

`pages.discovery.wisc.edu/%7Ecroyer/codes/dspfd_sources.zip`,

is a direct-search MATLAB code for derivative-free optimization by Gratton et al. [84]. The default parameters are used.
• **DESTRESS**, downloaded from

`pages.discovery.wisc.edu/%7Ecroyer/codes/destress_sources.zip`,

is a trust-region method in MATLAB for smooth, unconstrained optimization problems, with second-order convergence guarantees, by Gratton et al. [83]. The default parameters are used.
• **SDS**, **AHDS** and **DSDS**, downloaded from

`pages.discovery.wisc.edu/%7Ecroyer/codes/sounds_sources.zip`,

are symmetrized direct search, approximate Hessian direct search and decoupled step direct search, respectively. **SDS** and **AHDS** suggested by Gratton et al. [82] and **DSDS** proposed by Royer [152]. The default parameters are used for all.

8.11.3 Estimation of c

The following theorem was recently proved by Pinelis [143].

8.11.1 Theorem. *There is a universal constant c_0 such that for any fixed nonzero real vector q of any dimension n and any random vector p of the same dimension n with independent components uniformly distributed in $[-1, 1]$, we have*

$$(p^T p)(q^T q) \le c_0 n (p^T q)^2 \tag{8.58}$$

with probability $\ge 1/2$.

More specifically, Pinelis proved the bounds $0.73 < c_0 < 50$ for the optimal value of the constant c_0. The true optimal value seems to be approximately $16/7$. This is suggested by numerical simulation. To estimate c_0, we executed three times the Matlab commands

```
% run PinConst
N=10000;
nlist=[2:10,20,50,100,200,500,1000,2000,5000,10000,20000,50000,100000];
c0=PinConst(N,nlist);
```

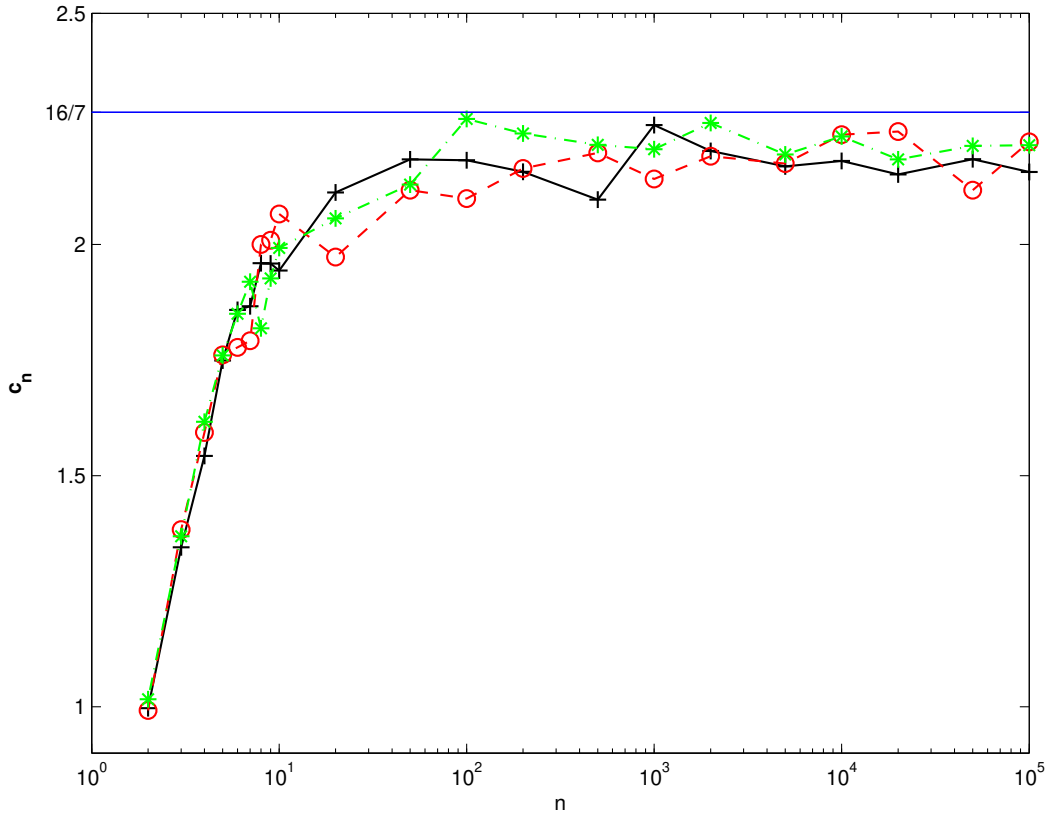


Figure 8.1: The plot of c_n versus the dimension n suggests that $c_0 \approx 16/7$.

using the algorithm **PinConst** below. All three outputs,

$$c_0 = 2.2582, c_0 = 2.2444, c_0 = 2.2714$$

are slightly smaller than $16/7 = 2.2857....$

8.11.2 Algorithm. (**PinConst, estimating the Pinelis constant**)
Purpose: Estimate c_0 satisfying (8.58) with probability $\geq 1/2$
Input: N (the total number of gradient evaluations), D (vector of dimensions used)
Output: c_0
(PC_0) Set $M = |D|$ and $i = 1$.
(PC_1) Replace i by $i + 1$ and set $k = 1$.
(PC_2) Replace k by $k + 1$.
(PC_3) Generate random g^k and p^k with length D_i and compute

$$\texttt{gain}(k) = \frac{\|g^k\|_2 \|p^k\|_2}{|(g^k)^T p^k|}.$$

(PC_4) If k exceeds N, go to (PC_2); otherwise, compute

$$\texttt{medgain}(i) = \texttt{median}(\texttt{gain}) \quad \text{and} \quad c(i) = \frac{(\texttt{medgain}(i))^2}{D_i}.$$

(PC_5) If i exceeds M, **PinConst** ends; otherwise go to (PC_1).

8.11.4 A list of test problems with f^{opt}

Here we list the `CUTEst` test problems for which our best point did not satisfy the condition

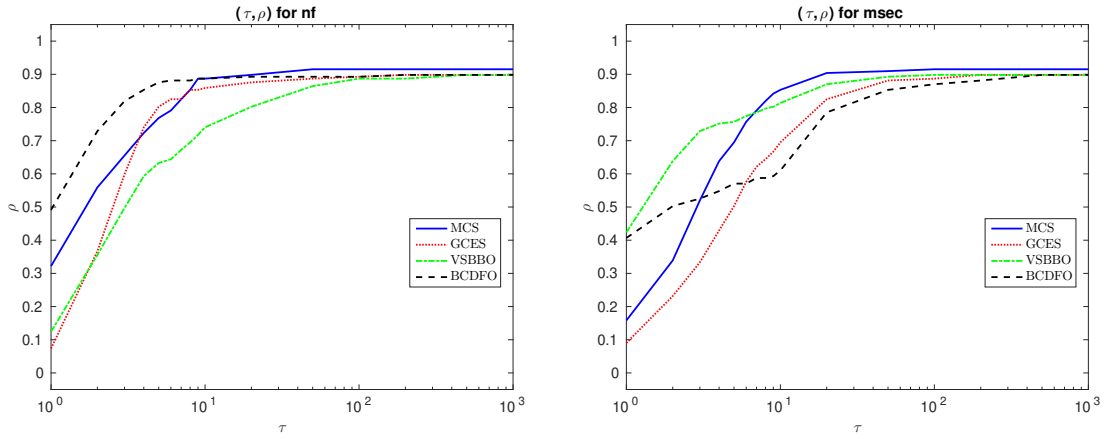
$$\|g^k\|_\infty \leq 10^{-5}.$$

problem	dim	f^{opt}	$\|g\|_\infty$	$\|g\|_2$
BROWNBS	2	$-2.80e+00$	$2.08e-05$	$2.08e-05$
DJTL	2	$-8.95e+03$	$1.44e-04$	$1.44e-04$
STRATEC	10	$2.22e+03$	$8.10e-05$	$1.42e-04$
SCURLY10:10	10	$-1.00e+03$	$5.34e-04$	$5.50e-04$
OSBORNEB	11	$2.40e-01$	$3.47e-02$	$3.47e-02$
ERRINRSM:50	50	$3.77e+01$	$1.89e-05$	$1.89e-05$
ARGLINC:50	50	$1.01e+02$	$1.29e-05$	$5.28e-05$
HYDC20LS	99	$1.12e+01$	$5.54e-01$	$8.79e-01$
PENALTY3:100	100	$9.87e+03$	$2.01e-03$	$4.68e-03$
SCOSINE:100	100	$-9.30e+01$	$1.95e-02$	$3.58e-02$
SCURLY10:100	100	$-1.00e+04$	$5.74e-02$	$1.56e-01$
NONMSQRT:100	100	$1.81e+01$	$3.42e-05$	$6.51e-05$
PENALTY2:200	200	$4.71e+13$	$3.85e-04$	$1.07e-03$
ARGLINB:200	200	$9.96e+01$	$3.27e-04$	$2.68e-03$
SPMSRTLS:499	499	$1.69e+01$	$1.08e-05$	$3.59e-05$
PENALTY2:500	500	$1.14e+39$	$1.97e+26$	$4.08e+26$
MSQRTBLS:529	529	$1.13e-02$	$1.44e-05$	$1.03e-04$
NONMSQRT:529	529	$6.13e+01$	$2.17e-05$	$1.76e-04$
SCOSINE	1000	$-9.21e+02$	$3.38e-03$	$9.32e-03$
SCURLY10	1000	$-1.00e+05$	$5.49e+01$	$3.37e+02$
COSINE	1000	$-9.99e+02$	$5.00e-05$	$6.34e-05$
PENALTY2:1000	1000	$1.13e+83$	$2.53e+77$	$3.41e+77$
SINQUAD:1000	1000	$-2.94e+05$	$1.21e-05$	$1.52e-05$
SPMSRTLS:1000	1000	$3.19e+01$	$9.75e-05$	$2.26e-04$
NONMSQRT:1024	1024	$9.01e+01$	$1.73e-04$	$1.28e-03$
MSQRTALS:4900	4900	$7.60e-01$	$1.88e-03$	$3.56e-02$
SPMSRTLS:4999	4999	$2.05e+02$	$2.36e-03$	$9.27e-03$

Continued on next page

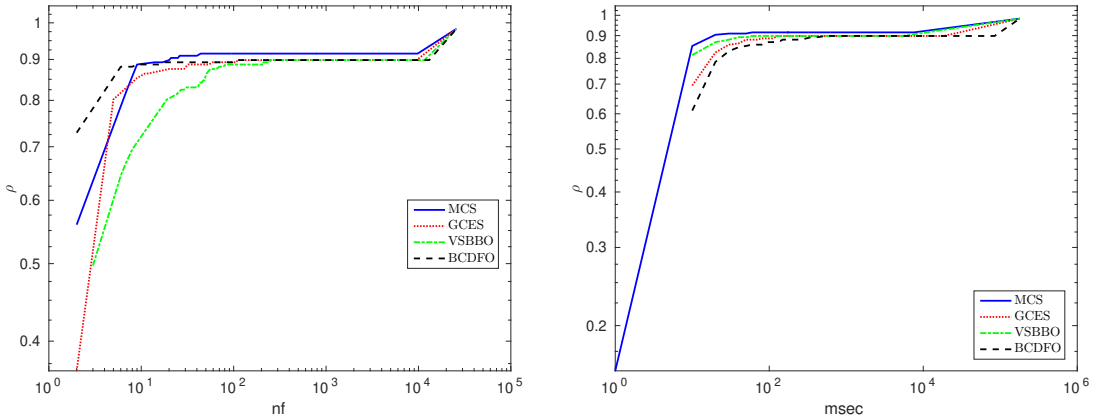
INDEFM:5000	5000	$-5.02e+05$	$1.43e-05$	$2.00e-05$
SBRYBND:5000	5000	$2.58e-10$	$3.73e-04$	$3.50e-03$
SCOSINE:5000	5000	$-4.60e+03$	$6.32e-03$	$2.72e-02$
NONCVXUN:5000	5000	$1.16e+04$	$3.94e-05$	$7.19e-04$

8.11.5 Performance profiles and plots



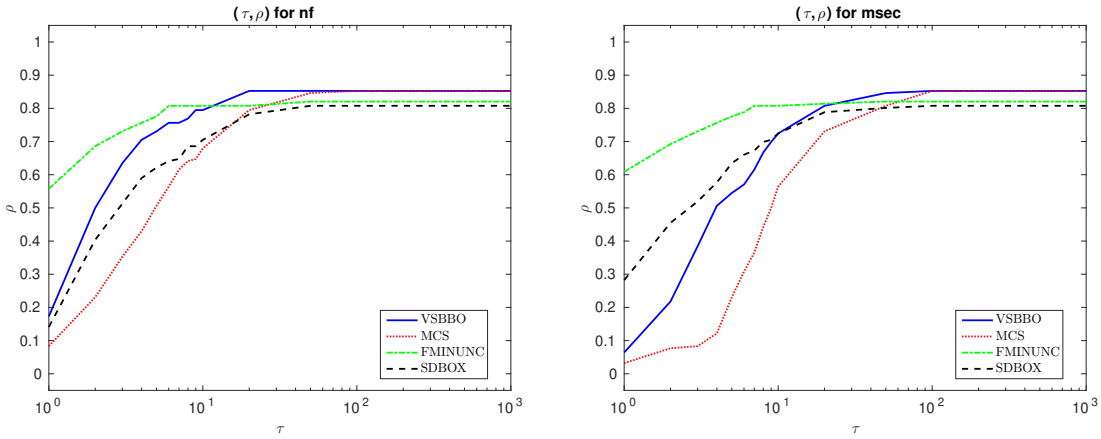
a) b)

Figure 8.2: Small dimensions 2–20: Performance profiles for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver.



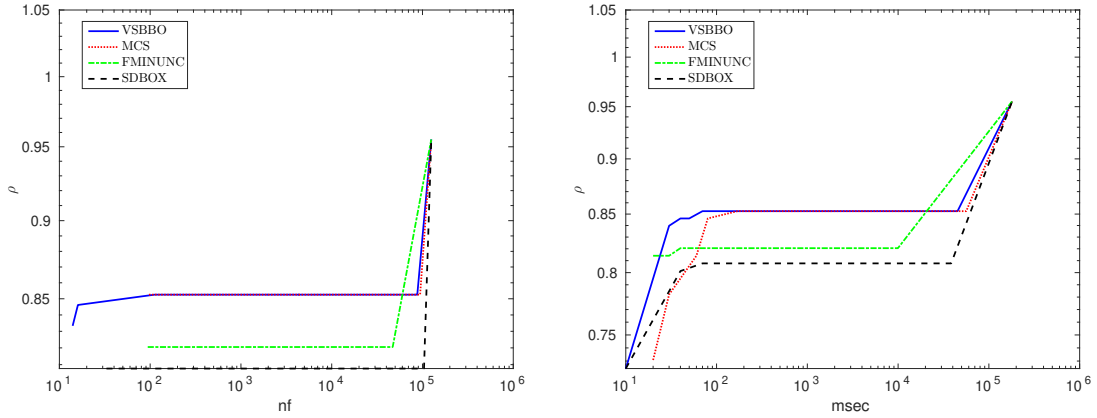
a) b)

Figure 8.3: Small dimensions 2–20: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.



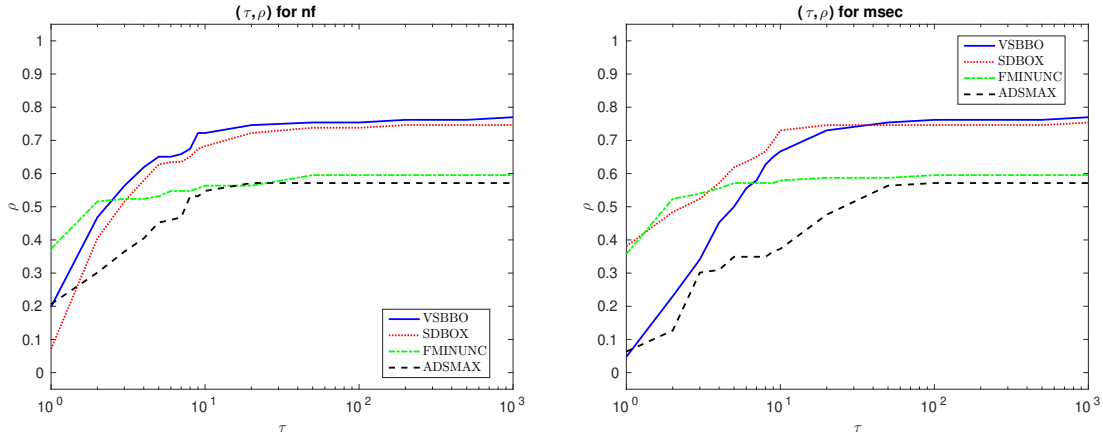
a) b)

Figure 8.4: Medium dimensions 21–100: Performance profiles for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.



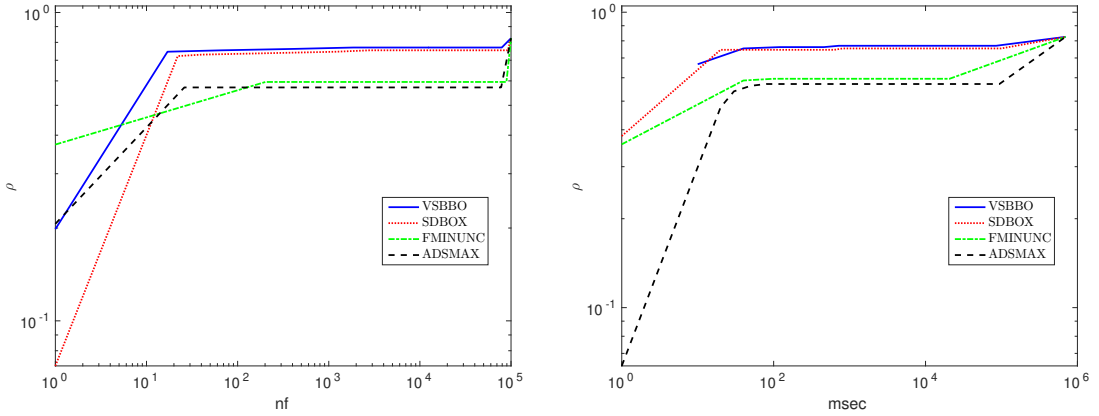
a) b)

Figure 8.5: Medium dimensions 21–100: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.



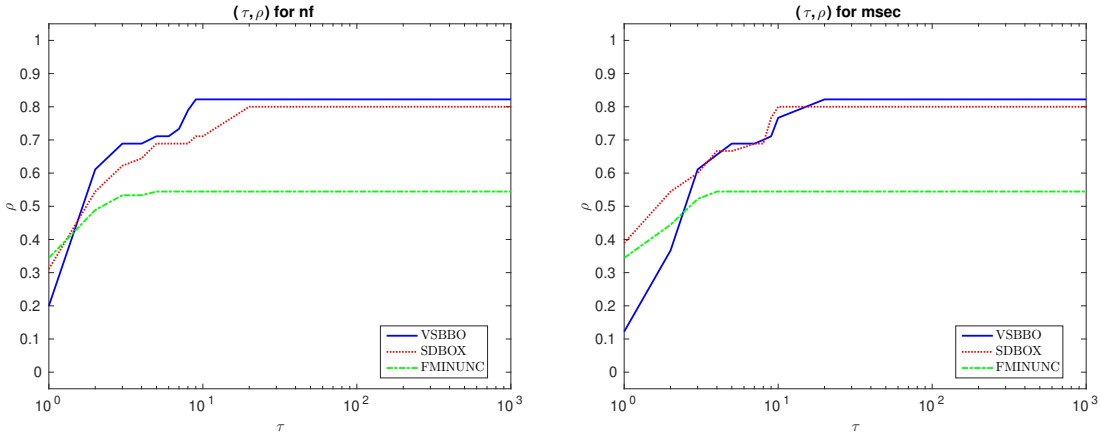
a) b)

Figure 8.6: Large dimensions 101–1000: Performance profiles for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.



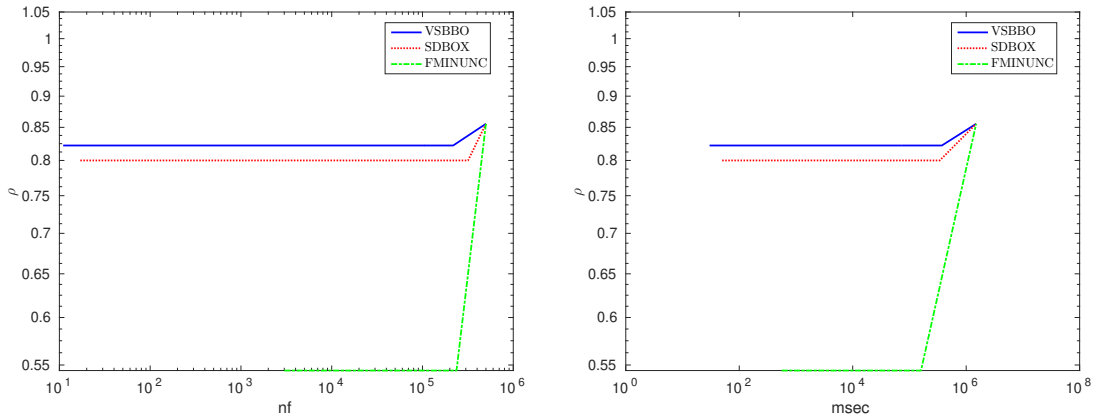
a) b)

Figure 8.7: Large dimensions 101–1000: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.



a) b)

Figure 8.8: Very large dimensions 1001–5000: Performance profiles for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.



a) b)

Figure 8.9: Very large dimensions 1001–5000: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

8.11.6 A list of large unsolved problem

EIGENALS	EIGENBLS	EIGENCLS	SPMSRTLS:499
GENROSE:500	PENALTY2:500	SINQUAD:500	MSQRTALS:529
MSQRTBLS:529	NONMSQRT:529	CURLY10	CURLY20
CURLY30	DIXON3DQ:1000	FLETCBV2:1000	FLETCBV3:1000
FLETCHCR:1000	NONCVXU2	SENSORS:1000	SINQUAD:1000
SPMSRTLS:1000			

8.11.7 A list of very large unsolved problem

MSQRTALS:1024	MSQRTBLS:1024	NONMSQRT:1024	EIGENALS:2550
EIGENBLS:2550	EIGENCLS:2652	MSQRTALS:4900	MSQRTBLS:4900
SPMSRTLS:4999	FLETCBV2:5000	NONCVXU2:5000	NONCVXUN:5000
SINQUAD:5000			

9 A new subspace method

This section discusses a deterministic subspace method for unconstrained black box optimization, called **STBBO**. This is a joint work with Arnold Neumaier and Parvaneh Faramarzi (Kimiaei, Neumaier, & Faramarzi [119]). **STBBO** improves subspace technique for enhancing a decrease in the model function value and the model gradient norm. If the exact objective function has Lipschitz continuous gradient, a complexity bound is found for a particular algorithm based on this technique in the presence of the inexact function value. Although the inexact function values affect the estimation of the gradient and subspace updating as well, the subspace directions will not be inefficient because the angle condition is enforced. Numerical results show that **STBBO** is competitive compared to solvers using the standard and limited memory BFGS methods.

9.1 Overview of the new method

This paper constructs a new deterministic solver, called **STBBO**, for unconstrained black box optimization problems. The basic version of **STBBO** is a quasi Newton algorithm using the limited memory direction by Kimiaei et al. [118], the finite difference technique for the approximation of the gradient, and the Wolfe line search method by Al-Baali & Fletcher [4] finding step sizes.

Under the assumptions discussed in the introduction, we prove that the Wolfe line search method [4] with both the exact gradient and function value can be satisfied while this line search is satisfied numerically with both the inaccurate function and gradient. In this case, we prove the complexity result for both the Wolfe line search algorithm [4] and the basic version of **STBBO** for the general case independent of the choice of search directions. The order of our complexity bound matches the one found by Berahas et al. [15].

The enhancements which make **STBBO** robust and competitive are:
• The new subspace direction is computed by finding the solution of a linear problem in the subspace and then by solving a simple program in the hope of finding **a decrease in the model function value and the model gradient norm**.
• In the presence of rounding errors, the search direction may not satisfy the angle condition; hence it needs to be modified. However after this modification the search direction may be contaminated by NaN or $\pm\infty$. In this case, a **diagonally preconditioned descent direction** is used.
• The Wolfe line search algorithm [4] may fail in finite precision arithmetic. In this case, the step size is generated by a **heuristic formula**.

STBBO is available at `https://www.mat.univie.ac.at/~neum/software/STBBO`

In Section 9.2, a basic version of our algorithm, called **STBBO-basic**, is introduced using the

Wolfe line search algorithm [4], called **WLS-basic**. In Section 9.3, complexity results for **WLS-basic** are discussed in Subsection 9.3.1 and for **STBBO-basic** in Subsection 9.3.2. Section 9.4 discusses
• in Subsection 9.4.1 the subspace information and how a basic version of subspace direction is computed and the subspace information is updated,
• in Subsection 9.4.2 how a decrease in the model function value and the gradient norm can be achieved,
• in Subsection 9.4.3 how an improved version of subspace direction can be computed and the angle condition can be enforced,
• in Subsection 9.4.4 an improved version of the Wolfe line search algorithm [4],
• in Subsection 9.4.5 an improved version of **STBBO-basic**.
Numerical results are given in Section 9.5.

The extensive numerical results show that **STBBO** by the present authors is effective and favorable in low up to high dimensions ($1 \leq n \leq 9000$) on all 568 unconstrained problems from the `CUTEst` test problems collection by Gould et al. [80] in comparison with **UOBYQA** by Powell [149], **BCDFO** by Gratton et al. [85], **BFO** by Porcelli & Toint [147], **NMSMAX** by Higham [99], **NELDER** by Kelley [109], **VSBBO** by Kimiaei & Neumaier [117], and **VSBBON** by Kimiaei [114].

9.2 A basic version of STBBO

Our basic algorithm is called **STBBO-basic**, a quasi-Newton algorithm for unconstrained black box optimization problems. As long as **STBBO-basic** does not converge, the direction is computed by a quasi-Newton method along which a line search is tried. If the search direction does not satisfy the angle condition, it needs to be improved. In **STBBO-basic**, the gradient vector is estimated by the forward finite difference technique.

STBBO-basic takes the initial point x^0 as input and returns an optimum point $x^{\rm best}$ and its inexact function value $\widetilde{f}^{\rm best}$ as output. It uses the following tuning parameters:
$\Delta^{\rm a} \in (0,1)$ (tiny parameter for the angle condition),
$0 < \widetilde{\sigma} < \widetilde{\rho} < 1$ (parameters for line search),
$0 < \varepsilon_h < 1$ (parameter for adjusting the finite difference step size h).
$0 < \varepsilon_g < 1$ (minimum threshold for the stopping test).

9.2.1 Algorithm. (**STBBO-basic, a quasi-Newton method for unconstrained BBO**)

($\rm Sb_0$) Compute the initial function value $\widetilde{f}^0 := \widetilde{f}(x^0)$.
for $\ell = 1, 2, 3, \cdots$ **do**
($\rm Sb_1$) Compute

$$h_i^\ell := \sqrt{\varepsilon_h}\,\mathrm{sign}(x^\ell)\max\{|x_i^\ell|, 1\}, \quad \text{for } i = 1, \cdots, n \tag{9.1}$$

and

$$\widetilde{g}_i^\ell := \widetilde{g}(x^\ell)_i := \frac{\widetilde{f}(x^\ell + h_i^\ell e_i) - \widetilde{f}(x^\ell)}{h_i^\ell}. \tag{9.2}$$

Here e_i is the ith coordinate direction and $\widetilde{f}$ is an inexact function value. If $\|\widetilde{g}(x^\ell)\| \leq \varepsilon_g$, set $x^{\rm best} := x^\ell$ and $\widetilde{f}^{\rm best} := \widetilde{f}^\ell$. Then **STBBO-basic** ends.

(Sb_2) Update the symmetric matrix $B^{\ell+1}$ by a quasi-Newton formula.
(Sb_3) Compute p^ℓ by solving the linear system $B^\ell p^\ell = -\widetilde{g}^\ell$.
(Sb_4) If the sufficient descent condition $(\widetilde{g}^\ell)^T p^\ell < 0$ does not hold, modify p^ℓ such that the angle condition

$$\frac{(\widetilde{g}^\ell)^T p^\ell}{\|\widetilde{g}^\ell\|\|p^\ell\|} \leq -\Delta^{\mathrm{a}} < 0, \tag{9.3}$$

holds.
(Sb_5) Find a step size α^ℓ satisfying

$$\widetilde{f}(x^\ell + \alpha^\ell p^\ell) \leq \widetilde{f}^\ell + \widetilde{\rho}\alpha^\ell (\widetilde{g}^\ell)^T p^\ell, \tag{9.4}$$
$$|\widetilde{g}(x^\ell + \alpha^\ell p^\ell)^T p^\ell| \leq -\widetilde{\sigma}(\widetilde{g}^\ell)^T p^\ell, \tag{9.5}$$

resulting in the new point $x^{\ell+1} := x^\ell + \alpha^\ell p^\ell$ and its function value $\widetilde{f}^{\ell+1} := \widetilde{f}(x^{\ell+1})$. The **sufficient descent** (9.4) guarantees $\widetilde{f}(x^\ell + \alpha^\ell p^\ell) < \widetilde{f}^\ell$ while the **curvature condition** (9.5) guarantees that α^ℓ is not too far from a minimizer of the function f.
end for

9.3 Complexity

We prove the complexity bounds for **WLS-basic** in Subsection 9.3.1 and **STBBO-basic** in Subsection 9.3.2 under the assumptions (BBO_1) and (BBO_2) discussed in Subsection 8.3 and the following assumption:
(BBO_3) The uncertainty of the function value $\widetilde{f}(x)$ obtained by a noisy oracle is globally bounded by a small threshold $\omega > 0$, i.e., the condition (4.4) holds.

9.3.1 Theorem. *Assume that* (BBO_1)–(BBO_3) *hold. If* $\varepsilon_h := \Theta(\omega)$, *then the gradient approximation defined by* (9.1)–(9.2) *satisfies*

$$\|\widetilde{g}(x) - g(x)\| = \mathcal{O}(\sqrt{\omega}). \tag{9.6}$$

Proof. By applying $f - \omega \leq \widetilde{f} \leq f + \omega$ into (9.2), we get

$$\widetilde{g}(x) = h^{-1}(\widetilde{f}(x + hp) - \widetilde{f}(x)) \leq h^{-1}(f(x + hp) - f(x)) + 2h^{-1}\omega,$$

resulting in

$$g(x) = \widetilde{g}(x) + \mathcal{O}(h) = h^{-1}(f(x + hp) - f(x)) + \mathcal{O}(\omega/h + h)$$

by Taylor expansion. Here the first and second term of $\mathcal{O}(\omega/h + h)$ are the function evaluation error and the discretization error. The optimal value $\sqrt{\omega}$ of $\Delta(h) := \omega/h + h$ is achieved at $\widehat{h} := \sqrt{\omega}$. If $h = \Theta(\sqrt{\omega})$, the best magnitude of the error is obtained and (9.6) holds. □

Berahas et al. [15] gave a nice summary to guarantee the condition (9.6) for various deterministic/stochastic techniques and to find bounds on the number of function evaluations to estimate the gradient, the step size h, and the exact gradient norm; for more details see Tables 2 and 3.

Under the assumptions (BBO_1)–(BBO_3), we find a complexity bound of **STBBO-basic** in the general case. We denote by $N(\omega)$ the number of function evaluations that **STBBO-basic** needs to find a point x^{best} satisfying

$$f(x^{\mathrm{best}}) \leq \sup_{x \in \mathbb{R}^n} \{f(x) \mid f(x) \leq f(x^0) \ \text{ and } \ \|g(x)\| \leq \omega\}.$$

9.3.1 Complexity of a line search

In this section, we discuss how to get the complexity result for the Wolfe line search by Al-Baali & Fletcher [4], called **WLS-basic**. It has a key role in proving the complexity bound for **STBBO-basic**. In the presence of rounding errors, when it fails, we enrich it by a heuristic formula, discussed later in Subsection 9.4.4.

As defined earlier, $\widetilde{g}(x)$ denotes the finite difference approximation of $g(x)$ at x. An efficient line search algorithm based on Wolfe conditions has been constructed by Al-Baali & Fletcher [4] and Fletcher [74]. It includes an inner loop (a bracketing phase, called **bracketingPhase**) and an outer loop (an interpolation phase, **sectioningPhase**). **bracketingPhase** finds an interval with suitable step sizes, and **sectioningPhase** finds a good step size within the interval. **sectioningPhase** has repeated calls to **bracketingPhase** until the sufficient descent (9.4) is not satisfied. Afterwards, it tries to satisfy the curvature condition (9.5).

The following result follows from Theorem 2.2 in [4] which is valid with the exact gradient.

9.3.2 Proposition. *Given the finite threshold value $\overline{f}$, we define*

$$\overline{\alpha}^\ell := \frac{\overline{f} - f^\ell}{\rho (g^\ell)^T p^\ell}, \quad \text{for all } \ell \geq 0. \tag{9.7}$$

If $\sigma > \rho$, then one of the following statements is valid:
(i) **WLS-basic** *ends with $f(x^\ell + \alpha_j^\ell p^\ell) < \overline{f}$ for some $\alpha_j^\ell \in (0, \overline{\alpha}^\ell]$;*
(ii) **WLS-basic** *ends with α_j^ℓ satisfying*

$$f(x^\ell + \alpha^\ell p^\ell) \leq f^\ell + \rho \alpha^\ell (g^\ell)^T p^\ell, \tag{9.8}$$
$$|g(x^\ell + \alpha^\ell p^\ell)^T p^\ell| \leq -\sigma (g^\ell)^T p^\ell. \tag{9.9}$$

In particular, if $\sigma = \rho$, **WLS-basic** *ends, each interval $I_j^\ell := (a_j^\ell, b_j^\ell)$, $j = 1, 2, \cdots, \infty$, contains an interval of acceptable points and there exists a limit point $\widehat{\alpha} \in I_j^\ell$, for all j, such that, for sufficiently large j, the a_j^ℓ are monotonically (not strictly) increasing, $\lim_{j\to\infty} a_j^\ell = \widehat{\alpha}$ and the b_j^ℓ are monotonically (not strictly) decreasing, $\lim_{j\to\infty} b_j^\ell = \widehat{\alpha}$, and $\widehat{\alpha}$ is an acceptable point as in (ii).*

The following result describes how the complexity bound for **WLS-basic** can be obtained.

9.3.3 Proposition. *Assume that (BBO$_1$)–(BBO$_3$) hold, the gradient of f is estimated by the forward finite differences (9.2) with the step sizes generated by (9.1), the direction p^ℓ satisfies the angle condition (9.3), and the step size α^ℓ satisfies the strong Wolfe conditions (9.4) and (9.5). Denote by $\widetilde{\theta}$ the angle between $\widetilde{g}^\ell$ and p^ℓ and by θ the angle between g^ℓ and p^ℓ. Then if $\varepsilon_h := \Theta(\omega)$:*
(i) For all $x^\ell, x^{\ell'} \in \mathbb{R}^n$ we have

$$\|\widetilde{g}(x^\ell) - \widetilde{g}(x^{\ell'})\| = \mathcal{O}(\sqrt{\omega}) + L\|x^\ell - x^{\ell'}\|.$$

(ii) p^ℓ satisfies the angle condition for the exact gradient

$$\frac{(g^\ell)^T p^\ell}{\|g^\ell\| \|p^\ell\|} < -\frac{\Delta^{\mathrm{a}}}{2}. \tag{9.10}$$

(iii) Under the assumptions

$$\|\widetilde{g}^\ell\| > \frac{2\sqrt{\omega}}{\Delta^{\rm a}(1-\sigma)}, \tag{9.11}$$

$$\|g^\ell\| \geq \max\Big\{\frac{4\widetilde{\rho}\sqrt{\omega}}{\rho\Delta^{\rm a}}, \frac{(1+\widetilde{\sigma})\sqrt{\omega}}{\sigma\Delta^{\rm a}}\Big\}, \tag{9.12}$$

and

$$\|\widetilde{g}^\ell\|\|g^\ell\| \geq \frac{16L\omega}{\rho(1-\widetilde{\sigma})(\Delta^{\rm a})^2}, \tag{9.13}$$

α^ℓ satisfies the strong Wolfe conditions

$$f(x^\ell + \alpha^\ell p^\ell) \leq f^\ell + \rho_1\alpha^\ell (g^\ell)^T p^\ell, \tag{9.14}$$

$$|g(x^\ell + \alpha^\ell p^\ell)^T p^\ell| \leq -\sigma_1 (g^\ell)^T p^\ell, \tag{9.15}$$

where $0 < \rho_1 := \widetilde{\rho} - \rho < \widetilde{\rho} < \widetilde{\sigma} < \sigma_1 := \widetilde{\sigma} + \sigma < 1$.
(iv) Given $0 < \rho_1 < \sigma_1 < 1$, the minimal threshold $\alpha^{\min} \in (0,1)$ for the step size, we define

$$\overline{\alpha}_1^\ell := \frac{\overline{f} - f^\ell}{\rho_1 (g^\ell)^T p^\ell} \quad \text{for all } \ell \geq 0. \tag{9.16}$$

and choose the step size $\alpha^\ell \in (0, \overline{\alpha}_1^\ell]$, for $\ell \geq 1$, and $\tau \in (0,1)$. Then **WLS-basic** *needs at most*

$$K^\ell := \Big\lceil \log_\tau \frac{\alpha^{\min}}{\overline{\alpha}_1^\ell} \Big\rceil \tag{9.17}$$

iterations and at most

$$N_f^\ell := n(K^\ell)^2 \tag{9.18}$$

function evaluations to satisfy (9.14) *and* (9.15) *(The factor n comes from the approximated gradient evaluations needed in the Wolfe line search).*

Proof. (i) From ($\rm BBO_2$) and (9.6), we have

$$\|\widetilde{g}(x^\ell) - \widetilde{g}(x^{\ell'})\| = \|\widetilde{g}(x^\ell) - g(x^{\ell'}) + g(x^\ell) - g(x^{\ell'}) + g(x^{\ell'}) - \widetilde{g}(x^{\ell'})\| = \mathcal{O}(\sqrt{\omega}) + L\|x^\ell - x^{\ell'}\|.$$

(ii) By [163, Lemma 3.11], since $\cos\widetilde{\theta} \geq \Delta^{\rm a} > 0$ by (9.3) and (9.6) holds, $\cos\theta \geq \Delta^{\rm a}/2$ and so (9.10) is obtained.
(iii) By (i), Cauchy–Schwarz inequality, and (9.5), we obtain

$$\begin{aligned}(\mathcal{O}(\sqrt{\omega}) + \alpha^\ell L\|p^\ell\|)\|p^\ell\| &= \|\widetilde{g}(x^\ell + \alpha^\ell p^\ell) - \widetilde{g}^\ell\|\|p^\ell\| \\ &\geq (\widetilde{g}(x^\ell + \alpha^\ell p^\ell) - \widetilde{g}^\ell)^T p^\ell \geq (\widetilde{\sigma} - 1)(\widetilde{g}^\ell)^T p^\ell,\end{aligned}$$

resulting in

$$\alpha^\ell \geq \frac{1-\widetilde{\sigma}}{L}\Big(\frac{-(\widetilde{g}^\ell)^T p^\ell}{\|p^\ell\|^2}\Big) - \frac{\mathcal{O}(\sqrt{\omega})}{L\|p^\ell\|} \geq \frac{(1-\widetilde{\sigma})\Delta^{\rm a}\|\widetilde{g}^\ell\| - \sqrt{\omega}}{L\|p^\ell\|} \geq \frac{(1-\widetilde{\sigma})\Delta^{\rm a}\|\widetilde{g}^\ell\|}{2L\|p^\ell\|}.$$

(iii) Under the condition (9.12), the statements (ii) and (iii) result in

$$\begin{aligned}-\rho\alpha^\ell g^\ell p^\ell &\geq \frac{\rho(1-\widetilde{\sigma})\Delta^{\rm a}\|\widetilde{g}^\ell\|}{2L\|p^\ell\|} g^\ell p^\ell \geq \frac{\rho(1-\widetilde{\sigma})\Delta^{\rm a}\|\widetilde{g}^\ell\|}{2L\|p^\ell\|}\|g^\ell\|\|p^\ell\|\cos\theta \\ &\geq \frac{\rho(1-\widetilde{\sigma})(\Delta^{\rm a})^2}{4L}\|\widetilde{g}^\ell\|\|g^\ell\| \geq 4\omega.\end{aligned}$$

On the other hand, (9.12) gives

$$-\rho\alpha^\ell g^\ell p^\ell \geq \frac{\rho\Delta^{\mathrm{a}}}{2}\alpha^\ell\|p^\ell\|\|g^\ell\| \geq \frac{\rho\Delta^{\mathrm{a}}}{2}\alpha^\ell\|p^\ell\|\Big(\frac{4\widetilde{\rho}\sqrt{\omega}}{\rho\Delta^{\mathrm{a}}}\Big) = 2\widetilde{\rho}\alpha^\ell\|p^\ell\|\sqrt{\omega}. \tag{9.19}$$

By summing both sides of (9.19) with (9.19), it results in

$$-\rho\alpha^\ell g^\ell p^\ell \geq 2\omega + \widetilde{\rho}\alpha^\ell\|p^\ell\|\sqrt{\omega}.$$

From (9.4) and (BBO_3), we get

$$\begin{aligned} f(x^\ell+\alpha^\ell p^\ell) &\leq f^\ell + 2\omega + \widetilde{\rho}\alpha^\ell(\widetilde{g}^\ell)^T p^\ell \leq f^\ell + 2\omega + \widetilde{\rho}\alpha^\ell\Big((g^\ell)^T p^\ell + \|p^\ell\|\sqrt{\omega}\Big) \\ &= f^\ell + 2\omega + \widetilde{\rho}\alpha^\ell\|p^\ell\|\sqrt{\omega} + \widetilde{\rho}\alpha^\ell(g^\ell)^T p^\ell \\ &\leq f^\ell + (\widetilde{\rho}-\rho)\alpha^\ell(g^\ell)^T p^\ell = f^\ell + \rho_1\alpha^\ell(g^\ell)^T p^\ell; \end{aligned}$$

hence (9.14) holds. From (9.5) we conclude that

$$g(x^\ell+\alpha^\ell p^\ell)^T p^\ell + \|p^\ell\|\sqrt{\omega} \geq \widetilde{g}(x^\ell+\alpha^\ell p^\ell)^T p^\ell \geq \widetilde{\sigma}(\widetilde{g}^\ell)^T p^\ell \geq \widetilde{\sigma}\Big((g^\ell)^T p^\ell - \|p^\ell\|\sqrt{\omega}\Big),$$

so that

$$g(x^\ell+\alpha^\ell p^\ell)^T p^\ell \geq \widetilde{\sigma}(g^\ell)^T p^\ell - (1+\widetilde{\sigma})\|p^\ell\|\sqrt{\omega} \geq (\widetilde{\sigma}+\sigma)(g^\ell)^T p^\ell = \sigma_1(g^\ell)^T p^\ell$$

by (9.12); hence (9.15) is obtained.
(iv) **sectioningPhase** chooses initially $\ell := 1$ and calls **bracketingPhase**. In the worst case, the sufficient descent condition (9.4) may be satisfied by **bracketingPhase** in the last iteration. **bracketingPhase** uses an interpolating which generates a decreasing sequence of the intervals $I^{\ell,j} := [a^{\ell,j}, b^{\ell,j}] \subset I^{\ell,j-1} := [a^{j-1}, b^{j-1}]$ for $j \geq 1$ and $\ell \geq 1$, resulting in

$$|b^{\ell,j} - a^{\ell,j}| \leq (1-\tau_1)|b^{\ell,j-1} - a^{\ell,j-1}|$$

with $\tau_1 \in (0,1)$; this condition is [4, (16)]. For sufficiently large j, $b^{\ell,j} - a^{\ell,j}$ goes to zero and α^ℓ satisfies (9.4). Hence, there exists a positive integer $\widehat{j}^\ell$ such that

$$\alpha^{\ell,j} \leq \alpha^\ell := \alpha^{\ell,\widehat{j}^\ell} \quad \text{for } j \geq \widehat{j}^\ell := \Big\lceil \log_{1-\tau_1}\frac{\alpha^\ell}{\overline{\alpha}_1^\ell}\Big\rceil \text{ and for } \ell \geq 1.$$

sectioningPhase chooses initially $\overline{I}^1 := [\overline{a}^1, \overline{b}^1] := I^{1,\widehat{j}}$. Then, for all $\ell \geq 1$, it uses an interpolating which gives

$$|\overline{b}^\ell - \overline{a}^\ell| \leq (1-\tau_1)|\overline{b}^{\ell-1} - \overline{a}^{\ell-1}|$$

with $\tau_1 \in (0,1)$ and an extrapolating which results in

$$|\overline{b}^\ell - \overline{a}^\ell| \leq (1-\tau_2)|\overline{b}^{\ell-1} - \overline{a}^{\ell-1}|,$$

with $\tau_2 \in (0,1)$. It generates the decreasing sequence $\overline{I}^\ell$ of the intervals so that, for sufficiently large ℓ, $\overline{b}^\ell - \overline{a}^\ell$ goes to zero while $\alpha^{\min}$ satisfies (9.5); this condition is [4, (17)]. Hence

$$|\overline{b}^\ell - \overline{a}^\ell| \leq \tau|\overline{b}^{\ell-1} - \overline{a}^{\ell-1}|$$

with $\tau := \min\{1-\tau_1, 1-\tau_2\} \in (0,1)$. Then there exists a positive integer $\widehat{\ell}$ such that

$$\alpha^\ell \le \alpha^{\min} \quad \text{for } \ell \ge \widehat{\ell} := \Big\lceil \log_\tau \frac{\alpha^{\min}}{\overline{\alpha}_1^\ell} \Big\rceil.$$

As a result, the number of iterations is bounded by (9.17) and so the number of function evaluations is bounded by (9.18) since n function evaluations are needed to estimate the gradient in each iteration, $\alpha^{\ell,\widehat{j}^\ell} \ge \alpha^{\min}$ for $\ell = 1, \cdots, \widehat{\ell}$, and $\tau \in (0,1)$. □

9.3.2 Complexity of STBBO-basic

9.3.4 Theorem. *Suppose that assumptions* (BBO_1)–(BBO_3) *hold and let* f^0 *be the initial function value of* f. *Define* $\overline{\alpha}^{\max} := \max\{\overline{\alpha}_1^\ell \mid \ell \ge 0\}$. *Then* **STBBO-basic** *needs at most* $\mathcal{O}(\omega^{-1})$ *iterations and* $\mathcal{O}(nK^2\omega^{-1})$ *function evaluations to find a point* x *with* $\|g(x)\| = \mathcal{O}(\omega)$. *Here* $K := \lceil \log_\tau(\alpha^{\min}/\overline{\alpha}^{\max}) \rceil$, $\alpha^{\min}$ *and* τ *come from Proposition 9.3.3, and the factor* n *comes from the approximated gradient evaluations needed in the Wolfe line search method.*

Proof. $\widehat{f} := \inf_\ell f^\ell$ is finite by (BBO_2) and (BBO_2). Let $\mathcal{S}$ be the set of iterations generated by **WLS-basic** such that

$$\|g^\ell\| > \max\Big\{\frac{4\widetilde{\rho}\sqrt{\omega}}{\rho\Delta^{\text{a}}}, \frac{(1+\widetilde{\sigma})\sqrt{\omega}}{\sigma\Delta^{\text{a}}}\Big\} \quad \text{for all } \ell \in \mathcal{S};$$

$\mathcal{S}$ is not empty by Proposition 9.3.2. In other words, all iterations of **STBBO-basic** decrease the function value; hence, they are successful. Based on Proposition 9.3.2, we consider two cases: Case 1. Suppose that the statement (i) in Proposition 9.3.2 holds. As defined earlier $\overline{\alpha}_1^\ell$ by (9.16), we get

$$f^{\ell+1} := f(x^\ell + \alpha^\ell p^\ell) \le \overline{f}, \quad \text{for some } \alpha^\ell \in (0, \overline{\alpha}_1^\ell];$$

hence

$$f^\ell - f^{\ell+1} \ge f^\ell - \overline{f} \ge 0. \tag{9.20}$$

Case 2. Suppose that the statement (ii) in Proposition 9.3.2 holds. Then, by the angle condition (9.3), we get from Proposition (9.3.3)

$$f^\ell - f^{\ell+1} \quad \ge \quad -\rho_1\alpha^\ell(g^\ell)^T p^\ell \ge \frac{4\rho_1\omega}{\rho}. \tag{9.21}$$

Then (9.20) and (9.21) result in $f^\ell - f^{\ell+1} \ge \dfrac{4\rho_1\omega}{\rho}$. By taking a sum from both sides of the above inequality, we get

$$f^0 - \widehat{f} \ge \sum_{k\in\mathcal{S}}(f^\ell - f^{\ell+1}) \ge |\mathcal{S}|\frac{4\rho_1\omega}{\rho},$$

leading to $|\mathcal{S}| \le \dfrac{\rho(f^0 - \widehat{f})}{4\rho_1\omega}$. Denote by N_f^ℓ the number of function evaluations in each call to **WLS-basic** for $\ell \ge 1$. By Proposition 9.3.3(iii), the total number of function evaluations by **STBBO-basic** is bounded by

$$\sum_{\ell\in\mathcal{S}} N_f^\ell = n\sum_{\ell\in\mathcal{S}}(K^\ell)^2 \le n|\mathcal{S}|K^2 = \mathcal{O}\Big(nK^2\omega^{-1}\Big).$$

□

The order of our bound is the same as the one found by Berahas et al. [15].

9.4 New enhancements

This section discusses how new enhancements make **STBBO-basic** very competitive:
• In Subsection 9.4.1 subspace information, limited memory quasi Newton direction, and subspace updating are given, resulting in a basic algorithm for computing subspace directions.
• In Subsection 9.4.2 under what conditions a decrease in the model function and gradient norm can be found is discussed.
• In Subsection 9.4.3 an improved algorithm for computing subspace directions is described.
• In Subsection 9.4.4 an improved Wolfe line search algorithm enriched by a heuristic formula is discussed.
• In Subsection 9.4.5 a new subspace algorithm is explained.

9.4.1 Subspace information

In this subsection, we give some details about subspace information, the limited memory quasi Newton direction, and how to update the subspace information.

Let S^ℓ be the $n \times m$ matrix whose columns are (in the actual implementation a permutation of) the previous m search directions,

$$S^\ell := \{s^{\ell-m+1}, \cdots, s^\ell\} = \{x^{\ell-m+1} - x^{\ell-m}, \cdots, x^\ell - x^{\ell-1}\}, \tag{9.22}$$

and $Y^\ell \in \mathbb{R}^{n\times m}$ be the corresponding gradient differences,

$$Y^\ell := \{y^{\ell-m+1}, \cdots, y^\ell\} = \{\widetilde{g}^{\ell-m+1} - \widetilde{g}^{\ell-m}, \cdots, \widetilde{g}^\ell - \widetilde{g}^{\ell-1}\}. \tag{9.23}$$

subspaceDir-basic solves $B^\ell p^\ell = -\widetilde{g}^\ell$, where

$$B^\ell := D^\ell + U^\ell(\Sigma^\ell)^{-1}(U^\ell)^T$$

is an approximate Hessian B^ℓ discussed in [118]. Here the diagonal matrix D^ℓ and U^ℓ are defined below and $\Sigma^\ell := (U^\ell)^T S^\ell$ is symmetric. This approximation is determined by unique S^ℓ and Y^ℓ and the **quasi-Newton condition**

$$B^\ell S^\ell = Y^\ell. \tag{9.24}$$

Another approximation is defined by

$$H^\ell := (S^\ell)^T B^\ell S^\ell = (S^\ell)^T Y^\ell. \tag{9.25}$$

At the ℓth iteration, **subspaceDir-basic** takes the matrices S^ℓ, Y^ℓ, H^ℓ and the estimated gradient $\widetilde{g}^\ell$ as input and returns the subspace direction p^ℓ as output. It only uses the tuning parameter m as the subspace size.

9.4.1 Algorithm. (**subspaceDir-basic, a basic version of the subspace direction**)
(SDb_1) Compute the diagonal matrix

$$D^\ell := \mathrm{diag}(d^\ell), \quad \text{with } d^\ell := \sqrt{(\sum_{i=1}^{m} \mathtt{YY}_{:i})//(\sum_{i=1}^{m} \mathtt{SS}_{:i})}, \tag{9.26}$$

where YY and SS are the componentwise squares of S^ℓ and Y^ℓ, respectively, and // is the componentwise division.
(SDb_2) Compute the matrix

$$U^\ell := Y^\ell - D^\ell S^\ell. \tag{9.27}$$

(SDb_3) Compute the matrix

$$M^\ell := (Y^\ell)^T (D^\ell)^{-1} Y^\ell - H^\ell, \tag{9.28}$$

which is equivalent to

$$M^\ell = (U^\ell)^T (D^\ell)^{-1} Y^\ell = (\Sigma^\ell)^{-1}; \tag{9.29}$$

[118, (25)]. (9.28) is numerically better than (9.29).
(SDb_4) Solve the linear system $M^\ell z^\ell = (U^\ell)^T (D^\ell)^{-1} \widetilde{g}^\ell$.
(SDb_5) Compute the direction by

$$p^\ell := (D^\ell)^{-1}(U^\ell z^\ell - \widetilde{g}^\ell). \tag{9.30}$$

By Theorem 1 in [118], p^ℓ is the solution of $B^\ell p^\ell = -\widetilde{g}^\ell$.

In the same way as [118], we update the matrices

$$S^\ell := (\, S^{\ell-1} \;\; s^\ell \,), \quad Y^\ell := (\, Y^{\ell-1} \;\; y^\ell \,), \quad H^\ell := \begin{pmatrix} H^{\ell-1} & (S^{\ell-1})^T y^\ell \\ (y^\ell)^T S^{\ell-1} & (y^\ell)^T s^\ell \end{pmatrix} \tag{9.31}$$

until m does not exceed its bound $m^{\max}$. It is done by **updateSYH**.

9.4.2 Decreasing model function value and gradient norm

In this subsection, we try to find a decrease in the model function value and gradient norm by a subspace technique. This can be done by solving a linear problem and a one-dimensional bound constrained problem in a cheap way.

We show how to minimize the maximum norm of the model gradient in the subspace spanned by the columns of the matrix $S^\ell \in \mathbb{R}^{n\times m}$, where typically $m \ll n$. We define by $\widetilde{f}^\ell := \widetilde{f}(x^\ell)$ the inexact function value at the current point x^ℓ, by $\widetilde{g}^\ell := \widetilde{g}(x^\ell)$ the estimated gradient function at x^ℓ, and by

$$c^\ell := (S^\ell)^T \widetilde{g}^\ell \tag{9.32}$$

the estimated gradient function restricted to S^ℓ at x^ℓ. Then the quadratic model

$$\widetilde{f}(x^\ell - S^\ell z) - \widetilde{f}^\ell \approx q(z) := -(c^\ell)^T z + \frac{1}{2} z^T H^\ell z$$

is constructed whose gradient is

$$q'(z) := \partial q(z)/\partial z := -c^\ell + H^\ell z.$$

Here c^ℓ by (9.32) and H^ℓ by (9.25) were defined earlier. The stationary point is at

$$\widehat{z} := (H^\ell)^{-1} c^\ell. \tag{9.33}$$

To find a point that reduces not only the model function value q but if possible also the model gradient norm $\|q\|$, we do on the model function an exact line search along $\widehat{z}$ by solving the simpler univariate problem

$$\begin{aligned} \min_{\beta} \quad & \|q'(\beta\widehat{z})\|_2 = |1-\beta|\|c^\ell\|_2 \\ \text{s.t.} \quad & q(\beta) = \gamma_1\beta + \gamma_2\beta^2 \le 0, \end{aligned} \tag{9.34}$$

where

$$\gamma_1 := -(c^\ell)^T\widehat{z}, \quad \gamma_2 := \frac{1}{2}\widehat{z}^T H^\ell \widehat{z}. \tag{9.35}$$

In three situations we do not want to improve the model gradient norm:
• If γ_1 or γ_2 is contaminated by NaN or $\pm\infty$, the constraint is not satisfied.
• If $\gamma_2 \le 0$, $q(\beta)$ is flat or unbounded below.
• If $\gamma_2 > 0$ but $\gamma_1 \ge 0$, the minimal q is attained at $\beta = 0$.
In these cases, the problem (9.34) can not be solved and p^ℓ is computed by **subspaceDir-basic**. In the remaining case, $\gamma_1 < 0 < \gamma_2$ and $\beta^* := -\gamma_1/(2\gamma_2) > 0$. Since $q(\beta) \le 0$ iff $0 \le \beta \le 2\beta^*$. The constraint in (9.34) is equivalent to $0 \le \beta \le 2\beta^*$. This case is only acceptable when β^* is a real value (otherwise p^ℓ is computed by **subspaceDir-basic**). Hence, (9.34) reduces to

$$\begin{aligned} \min \quad & |1-\beta| \\ \text{s.t.} \quad & 0 \le \beta \le 2\beta^* \end{aligned} \tag{9.36}$$

with the solution

$$\widehat{\beta} := \min(1, 2\beta^*). \tag{9.37}$$

We denote by $\widehat{q} := q(\widehat{\beta})$ the optimal model function value and by $\widehat{q}' := |1-\widehat{\beta}|\|c^\ell\|_2$ the optimal model gradient norm. In fact, $\widehat{\beta}$ does not necessarily decrease both the model function value q and the model gradient norm q' in finite precision arithmetic. To check this, we define two computational measures $\mathtt{df}^\ell$ and $\mathtt{dg}^\ell$, which are initialized as

$$\mathtt{df}^0 := \varepsilon_1|\widetilde{f}^0|, \quad \mathtt{dg}^0 := \varepsilon_2\|\widetilde{g}^0\| \quad \text{with } \varepsilon_1, \varepsilon_2 > 0 \tag{9.38}$$

(if $\widetilde{f}^0 = 0$ set $\mathtt{df}^0 := 1$) and updated based on the information of function value and estimated gradient, respectively,

$$\mathtt{df}^\ell := \begin{cases} \gamma_f^1(\widetilde{f}^\ell - \widetilde{f}^{\ell-1}) & \text{if } \widetilde{f}^\ell < \widetilde{f}^{\ell-1} + \mathtt{df}^{\ell-1}, \\ \max\{\gamma_f^2 \mathtt{df}^{\ell-1}, \gamma_f^3(|\widetilde{f}^\ell| + |\widetilde{f}^{\ell-1}|)\} & \text{otherwise} \end{cases} \tag{9.39}$$

and

$$\mathtt{dg}^\ell := \begin{cases} \gamma_g^1\|\widetilde{g}^\ell - \widetilde{g}^{\ell-1}\|_\infty & \text{if } \|\widetilde{g}^\ell\|_\infty < \|\widetilde{g}^{\ell-1}\|_\infty + \mathtt{dg}^{\ell-1}, \\ \max\{\gamma_g^2 \mathtt{dg}^{\ell-1}, \gamma_g^3(\|\widetilde{g}^\ell\|_\infty + \|\widetilde{g}^{\ell-1}\|_\infty)\} & \text{otherwise.} \end{cases} \tag{9.40}$$

Here the parameters $\gamma_f^1, \gamma_g^1 \in (0,1)$, $\gamma_f^2, \gamma_g^2 > 1$, and $\gamma_f^3, \gamma_g^3 \in (0,1)$ are tuning parameters. At the ℓth iteration, if

$$\widehat{q} \le -\mathtt{df}^\ell \ \text{ and } \ \widehat{q}' \le \|\widetilde{g}^{\ell-1}\|_\infty - \gamma_g^4 \mathtt{dg}^\ell, \tag{9.41}$$

we have a good decrease in both the model function value and gradient norm in the current subspace, where $\gamma_g^4 > 1$. Then we can compute the search direction by

$$p^\ell := -\widehat{\beta} S^\ell \widehat{z}. \tag{9.42}$$

9.4.3 An improved version of subspaceDir-basic

This subsection discusses **subspaceDir**, an improved version of **subspaceDir-basic**. It takes the matrices S^ℓ, Y^ℓ, H^ℓ, the estimated gradient $\widetilde{g}^\ell$, two computational measures $\mathtt{df}^\ell$ and $\mathtt{dg}^\ell$ as input and returns the subspace direction p^ℓ as output. It uses the following tuning parameter:
$m > 0$ (memory for the subspace).

9.4.2 Algorithm. (**subspaceDir, an improved subspace direction**)
(SD_1) Solve the linear problem (9.33).
(SD_2) To get the optimal model function and gradient:
(1) compute h^ℓ, γ_1, and γ_2 by (9.35).
(2) compute $\widehat{\beta}$ by (9.37)
Then $\widehat{q} = q(\widehat{\beta})$ and $\widehat{q}' = q'(\widehat{\beta})$ are the model function and gradient, respectively.
(SD_3) If the condition (9.41) holds,
(1) find $\widehat{z}$ by (9.33),
(2) compute p^ℓ by (9.42).
Otherwise, p^ℓ is computed in the same way as **subspaceDir-basic**.

Let $\Delta^{\mathrm{a}} \in (0, 1)$ be the tiny tuning parameter. When the search direction p^ℓ computed by **subspaceDir** does not satisfy the angle condition (9.3), it needs to be modified by setting

$$\eta_1 := (\widetilde{g}^\ell)^T \widetilde{g}^\ell, \quad \eta_2 := (p^{\ell-1})^T p^{\ell-1}, \quad \text{and} \quad \eta := (\widetilde{g}^\ell)^T p^{\ell-1},$$

finding t enforcing

$$\frac{\eta - t\eta_1}{\sqrt{\eta_1(\eta_2 - 2t\eta + t^2\eta_1)}} \leq -\Delta^{\mathrm{a}},$$

and computing $p^\ell = p^{\ell-1} - t\widetilde{g}^\ell$. But it is not possible sometimes due to rounding errors. In this case, our idea is to replace $p^\ell = -\widetilde{g}^\ell$ by the **diagonally preconditioned steepest descent direction** $p^\ell = -(D^\ell)^{-1}\widetilde{g}^\ell$. This is done by **enforceAngle**. This is different from the **enforceAngle** of [118].

9.4.4 An improved Wolfe line search

The statements of (i) and (ii) in Proposition 9.3.2 may be not satisfied numerically due to rounding errors. In this case, **WLS-basic** needs to be improved.

An improved version of **WLS-basic** by Al-Baali & Fletcher [4] is called **WLS**. It uses the following tuning parameters:
$0 < \widetilde{\sigma} < \widetilde{\rho} < 1$ (parameters for line search),
$0 < \Delta^\alpha < 1$ (tiny parameter for adjusting the heuristic step size),
$0 < \varepsilon_h < 1$ (parameter for adjusting the finite difference step size h).

9.4.3 Algorithm. (**WLS, an improved Wolfe line search**)
(WLS_1) Call **WLS-basic** to find a step size satisfying (9.4) and (9.5).
(WLS_2) If **WLS-basic** does not enforce the Wolfe conditions in finite precision arithmetic, find the step size by a heuristic way:
(1) Find $\mathtt{ind} := \{i \mid p_i^\ell \neq 0\}$. If both x^ℓ and p^ℓ are not zero, compute $\alpha^\ell := \Delta^\alpha |\widetilde{f}^\ell / (\widetilde{g}^\ell)^T p^\ell|$.
(2) Otherwise if only p^ℓ is not zero, compute

$$\alpha^\ell := \Delta^\alpha \max\left(|\widetilde{f}^\ell / (\widetilde{g}^\ell)^T p^\ell|, \min\left\{|x_i^\ell / p_i^\ell| \mid i \in \mathtt{ind}\right\}\right).$$

(3) Otherwise choose $\alpha^\ell := 1$.
Then compute the new point $x^{\ell+1} := x^\ell + \alpha^\ell p^\ell$, its inexact function value $\widetilde{f}^{\ell+1} := \widetilde{f}(x^{\ell+1})$, and its estimated gradient $\widetilde{g}^{\ell+1} := \widetilde{g}(x^{\ell+1})$.

9.4.5 The new subspace algorithm

Our new algorithm is called **STBBO**, **subspace technique for unconstrained black box optimization**. It takes advantage of the new subspace technique which tries to significantly decrease not only the model function value but also the model gradient norm. As long as **STBBO** does not converge, the direction is computed by **subspaceDir** along which **WLS** is tried. Then, some necessary information about the subspace is updated. Note that if the ℓth search direction does not satisfy the descent condition $(\widetilde{g}^\ell)^T p^\ell < 0$, then it needs to be improved by calling **enforceAngle**.

STBBO takes the initial point x^0 and maximal function evaluations (`nfmax`) as input and returns the best point x^{best} and its inexact function value $\widetilde{f}^{\text{best}}$ as output. It uses the following tuning parameters:
$\varepsilon_1, \varepsilon_2 > 0$ (parameters for adjusting the initial `df` and `dg`),
$\gamma_f^2, \gamma_g^2 > 1$ (parameters for expanding `df` and `dg`),
$\gamma_f^1, \gamma_g^1, \gamma_f^3, \gamma_g^3 \in (0,1)$, $\gamma_g^4 > 1$ (parameters for reducing `df` and `dg`),
$m > 0$ (memory for the subspace),
$\Delta^{\text{a}} \in (0,1)$ (tiny parameter for the angle condition),
$0 < \Delta^\alpha < 1$ (tiny parameter for adjusting the heuristic step size),
$0 < \widetilde{\sigma} < \widetilde{\rho} < 1$ (parameters for line search),
$0 < \varepsilon_h < 1$ (parameter for adjusting the finite difference step size h).

9.4.4 Algorithm. (**STBBO, subspace method for black box optimization**)
(ST_0) Initialization:
(1) Compute the initial inexact function value $\widetilde{f}^0$.
(2) Estimate the gradient $\widetilde{g}^0$ by (9.1) and (9.2).
(3) Initialize $\texttt{df}^0$ and $\texttt{dg}^0$ by (9.38).
(4) Set $x^{\text{best}} := x^0$ and $\widetilde{f}^{\text{best}} := \widetilde{f}^0$.
for $\ell = 1, 2, 3, \cdots$ **do**
(ST_1) Compute the subspace direction by **subspaceDir**. If $(\widetilde{g}^\ell)^T p^\ell \geq 0$, perform **enforceAngle**.
(ST_2) Find step size α^ℓ:
(1) Perform **WLS** to find α^ℓ resulting in $x^{\ell+1}$, $\widetilde{f}^{\ell+1}$, and $\widetilde{g}^{\ell+1}$ by (9.1) and (9.2).
(2) If `nfmax` is reached, set $x^{\text{best}} := x^{\ell+1}$ and $\widetilde{f}^{\text{best}} := \widetilde{f}^{\ell+1}$. Then **stop**.
(3) Update $\texttt{df}^{\ell+1}$ by (9.39) and $\texttt{dg}^{\ell+1}$ by (9.40).
(ST_3) Update the subspace information by setting $s^\ell := x^{\ell+1} - x^\ell$ and $y^\ell := \widetilde{g}^{\ell+1} - \widetilde{g}^\ell$ and calling **updateSYH**.
end for

9.5 Numerical results

In this section, we give a comprehensive comparison of **STBBO** with several robust and fast solvers on all 568 unconstrained problems from the `CUTEst` [80] collection of test problems for optimization with up to 9000 variables. The arguments are shifted by (5.1) to prevent guessing the solution of toy problems with a simple solution (such as all zero or all one).

We choose

$$\texttt{nfmax} \in \begin{cases} \{100n, 500n, 1000n\} & \text{if } 1 \leq n \leq 30 \\ \{100n, 500n, 1000n\} & \text{if } 31 \leq n \leq 1000 \\ \{100n, 500n, 1000n\} & \text{if } 1001 \leq n \leq 9000 \end{cases}$$

We are interested in using the test environment suggested by Kimiaei & Neumaier [116]. In order to compare **STBBO** with known solvers, two cost measures `nf` and `msec` are used. The **efficiency** of the solver *so* with respect to a cost measure c^{so} for solver *so* has been defined earlier by (5.2).

We consider a problem solved by the solver *so* if $q^{so} \leq 10^{-4}$ holds. Otherwise, the problem is unsolved since either `nfmax` or `secmax` is exceeded. Here q^{so} has been defined by (5.3).

Default parameters for **STBBO** are discussed in Subsection 9.6.1.

We compare **STBBO** with **UOBYQA** by Powell [149], **BCDFO** by Gratton et al. [85], **BFO** by Porcelli & Toint [147], **NMSMAX** by Higham [99], **NELDER** by Kelley [109], **VSBBO** by Kimiaei & Neumaier [117], and **VSBBON** by Kimiaei [114]. Details of code compared can be found in Subsection 8.11.2.

We summarize the numerical results, described in details in Subsection 9.6.2. In a comparison among solvers using the standard BFGS and limited memory BFGS methods (**FMINUNC**, **FMINUNC1**, **FMINUNC2**) and **STBBO**, it is shown that **STBBO** is very robust and efficient for problems in low up to high dimensions.
• The main results is that **STBBO** is more efficient than **FMINUNC** which uses the standard BFGS method while **FMINUNC** is more efficient than **FMINUNC1** and **FMINUNC2** which use the standard limited memory BFGS method proposed by Liu & Nocedal [124]. The new subspace technique remains efficient and robust though it collects the Hessian information only in a subspace.
• **STBBO** has the best performance in comparison with **BCDFO** using surrogate quadratic models and the two efficient versions of Nelder–Mead (**NELEDER** and **NMSMAX**) for dimensions $n \leq 30$. Moreover, **STBBO** and **UOBYQA** have somewhat behaviour for dimensions $n \leq 30$ but **STBBO** is more efficient than **UOBYQA** for dimensions $11 \leq n \leq 30$. **STBBO** is more efficient than two recent randomized solvers **VSBBO** and **VSBBON** and the new version of **BFO** for dimensions $n \leq 30$.
• In contrast to this, **VSBBO** and **VSBBON** are better than **STBBO** only in terms of the number of solved problems for dimensions $31 \leq n \leq 1000$.
• For very large problems, **STBBO** is better than **VSBBO** and **VSBBON** in terms of the number of solved problems and `nf` efficiency.
• Finally, in a generic comparison for dimensions $n \leq 9000$, **STBBO** is more efficient than **VSBBO** and **FMINUNC**.

9.6 Additional material for STBBO

This section discusses additional material for **STBBO**.

9.6.1 Default tuning parameters for STBBO

STBBO with the memory $m := \min(10, n)$ and $m := \min(20, n)$ is denoted by **STBBO1** and **STBBO2**, respectively. The tuning parameters for **STBBO** are:

$$\begin{array}{|l|}\hline \varepsilon_h = \varepsilon_m, \quad \Delta^{\mathrm{a}} = 10^{-8}, \quad \widetilde{\sigma} = 10^{-4}, \quad \widetilde{\rho} = 0.9, \quad \Delta^{\alpha} = \varepsilon_m, \quad \gamma_g^4 = 10, \\ \gamma_f^1 = \gamma_g^1 = 0.5, \quad \gamma_f^2 = \gamma_g^2 = 2, \quad \gamma_f^3 = \gamma_g^3 = 10^{-12}, \quad \varepsilon_1 = \varepsilon_2 = 10^{-8} \\ \hline \end{array}$$

9.6.2 Tables and plots

Table 9.1 and Figure 9.1 compare the summary statistics for $1 \leq n \leq 30$ with the small budget $\texttt{nfmax} = 100n$:
• For $1 \leq n \leq 30$ **UOBYQA** is the best in terms of the number of solved problems and the `nf` efficiency. **STBBO1** and **STBBO2** have the same behaviour and are the best solvers in terms of the `nf` efficiency.
• For $1 \leq n \leq 2$ **UOBYQA**, **STBBO1**, **STBBO2**. and **BCDFO** are the best in terms of the number of solved problems and **UOBYQA** is the best the `nf` efficiency.
• For $3 \leq n \leq 5$ and $6 \leq n \leq 10$ **UOBYQA** is the best in terms of the number of solved problems and the `nf` efficiency.
• For $11 \leq n \leq 30$ **FMINUNC** and **VSBBO** are the best in terms of the number of solved problems and **STBBO1** is the best in terms of the `nf` efficiency.

Table 9.1: Results for $1 \leq n \leq 30$ with $\texttt{nfmax} = 100n$

stopping test:	$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 100*n					
176 of 192 problems without bounds solved									mean efficiency in %	
dim∈[1,30]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	161	33	29	226	30	0	1	52	49
STBBO2	**stbbo2**	157	38	2	53	35	0	0	52	54
STBBO1	**stbbo1**	157	38	1	58	35	0	0	52	52
FMINUNC	**func**	144	42	20	96	15	0	33	46	33
BCDFO	**bcd**	141	36	31	3441	0	11	40	41	13
NELDER	**neld**	137	9	7	138	47	0	8	29	30
VSBBO	**vsbb**	134	5	4	118	58	0	0	21	23
NMSMAX	**nmsm**	133	6	6	73	59	0	0	27	35
VSBBON	**vsbbn**	131	19	16	366	61	0	0	27	10
FMINUNC2	**func2**	128	29	2	78	58	0	6	38	30
FMINUNC1	**func1**	127	27	0	69	59	0	6	38	27
BFO	**bfo**	99	0	0	199	0	0	93	14	6
34 of 34 problems without bounds solved									mean efficiency in %	
dim∈[1,2]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	33	13	10	17	1	0	0	73	79
STBBO2	**stbbo2**	33	5	0	22	1	0	0	59	55
STBBO1	**stbbo1**	33	5	0	47	1	0	0	59	42
BCDFO	**bcd**	33	9	5	54	0	0	1	65	27
NMSMAX	**nmsm**	32	1	1	20	2	0	0	43	54
NELDER	**neld**	32	5	3	20	0	0	2	56	56
FMINUNC	**func**	31	5	0	57	0	0	3	54	24
VSBBON	**vsbbn**	30	5	2	82	4	0	0	38	15
FMINUNC1	**func1**	29	6	0	39	4	0	1	51	20
FMINUNC2	**func2**	29	6	0	48	4	0	1	51	29
VSBBO	**vsbb**	29	2	1	31	5	0	0	26	29
BFO	**bfo**	28	0	0	129	0	0	6	31	7
36 of 39 problems without bounds solved									mean efficiency in %	
dim∈[3,5]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	34	12	11	35	5	0	0	67	65
NELDER	**neld**	33	0	0	46	4	0	2	38	50
FMINUNC	**func**	32	9	5	73	0	0	7	50	37
STBBO2	**stbbo2**	31	6	0	43	8	0	0	47	45
STBBO1	**stbbo1**	31	6	0	41	8	0	0	47	46
BCDFO	**bcd**	31	7	7	144	0	0	8	51	19
NMSMAX	**nmsm**	29	1	1	34	10	0	0	34	47
VSBBO	**vsbb**	25	0	0	54	14	0	0	18	22
VSBBON	**vsbbn**	25	3	3	183	14	0	0	20	10
FMINUNC1	**func1**	24	4	0	54	14	0	1	31	21
FMINUNC2	**func2**	24	4	0	50	14	0	1	31	24
BFO	**bfo**	23	0	0	129	0	0	16	18	8
68 of 76 problems without bounds solved									mean efficiency in %	
dim∈[6,10]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	60	8	8	183	15	0	1	49	44
STBBO2	**stbbo2**	58	12	0	67	18	0	0	44	48
STBBO1	**stbbo1**	58	12	0	64	18	0	0	44	51
BCDFO	**bcd**	50	16	15	2351	0	0	26	37	11
FMINUNC	**func**	45	16	11	91	9	0	22	40	34
VSBBO	**vsbb**	44	1	1	130	32	0	0	20	21
VSBBON	**vsbbn**	43	8	8	435	33	0	0	27	10
NMSMAX	**nmsm**	43	3	3	81	33	0	0	22	28
FMINUNC1	**func1**	41	10	0	81	31	0	4	32	28
FMINUNC2	**func2**	41	10	0	81	31	0	4	32	29
NELDER	**neld**	41	4	4	114	31	0	4	21	18
BFO	**bfo**	30	0	0	195	0	0	46	10	7
38 of 43 problems without bounds solved									mean efficiency in %	
dim∈[11,30]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
FMINUNC	**func**	36	12	4	154	6	0	1	47	34
VSBBO	**vsbb**	36	2	2	218	7	0	0	24	22
STBBO2	**stbbo2**	35	15	2	67	8	0	0	63	71
STBBO1	**stbbo1**	35	15	1	74	8	0	0	64	66
UOBYQA	**uob**	34	0	0	693	9	0	0	26	21
FMINUNC2	**func2**	34	9	2	119	9	0	0	45	38
FMINUNC1	**func1**	33	7	0	91	10	0	0	44	37
VSBBON	**vsbbn**	33	3	3	674	10	0	0	25	6
NELDER	**neld**	31	0	0	387	12	0	0	14	11
NMSMAX	**nmsm**	29	1	1	157	14	0	0	19	20
BCDFO	**bcd**	27	4	4	13385	0	11	5	22	2
BFO	**bfo**	18	0	0	404	0	0	25	4	4

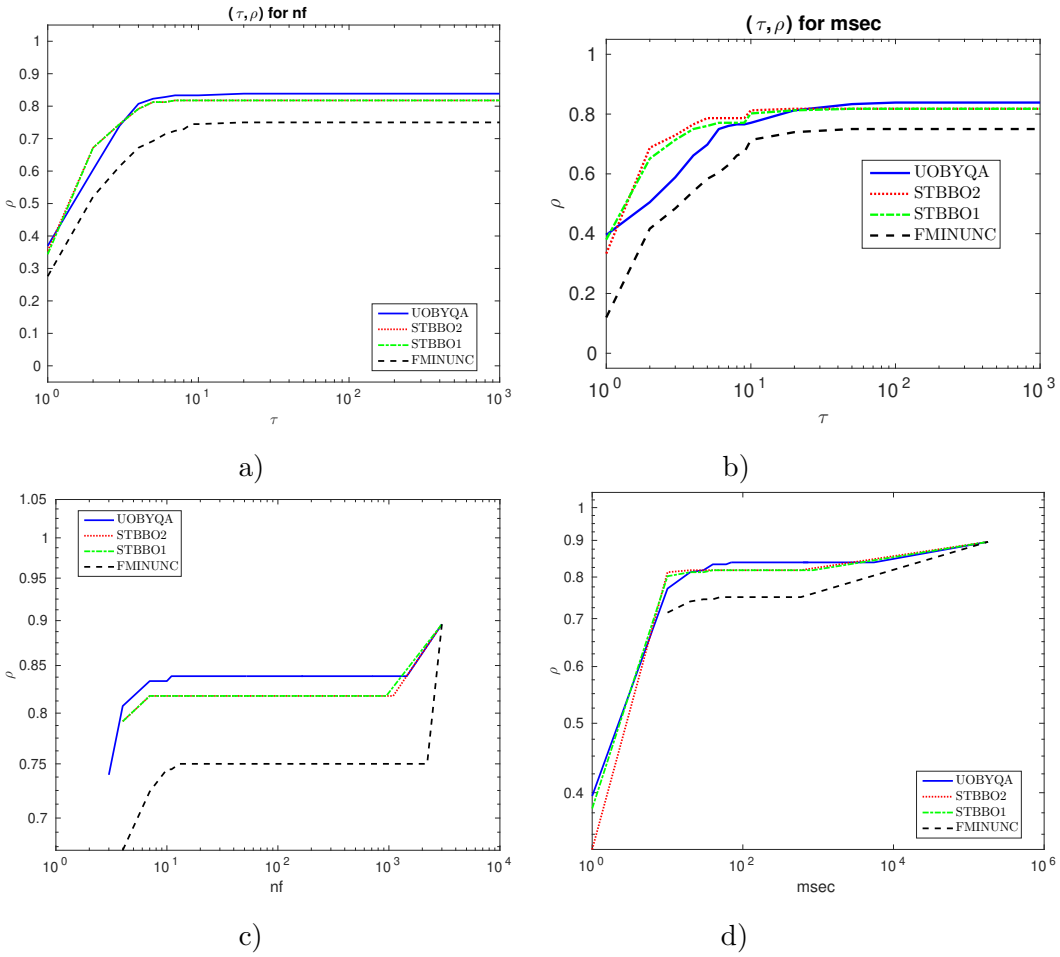


Figure 9.1: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

The results obtained for $1 \leq n \leq 30$ with the medium budget $\texttt{nfmax} = 500n$ are summarized in Table 9.2 and shown in Figure 9.2:
• For $1 \leq n \leq 30$ **STBBO1**, **STBBO2**, and **UOBYQA** are the best in terms of the `nf` efficiency and the number of solved problems.
• For $1 \leq n \leq 2$ **UOBYQA** and **BCDFO** are the two best solvers.
• For $3 \leq n \leq 5$ and $6 \leq n \leq 10$ **UOBYQA** and **STBBO1** (**STBBO2**) are the two best solvers.
• For $11 \leq n \leq 30$ **VSBBO** is the best in terms of the number of solved problems and **STBBO1** is the best solvers in terms of `nf` efficiency.

Table 9.2: Results for $1 \leq n \leq 30$ with $\texttt{nfmax} = 500n$

stopping test:		$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 500*n				
188 of 192 problems without bounds solved									mean efficiency in %	
dim∈[1,30]						# of anomalies			for cost measure	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
UOBYQA	**uob**	176	36	31	840	13	0	3	56	51
STBBO2	**stbbo2**	176	37	2	100	15	0	1	56	61
STBBO1	**stbbo1**	176	39	3	105	15	0	1	56	58
VSBBO	**vsbb**	167	8	6	246	25	0	0	27	28
NELDER	**neld**	164	11	9	336	11	0	17	32	32
VSBBON	**vsbbn**	163	19	16	853	29	0	0	30	12
NMSMAX	**nmsm**	159	6	6	162	29	0	4	30	38
BCDFO	**bcd**	157	36	31	5754	0	12	23	44	11
FMINUNC	**func**	154	45	24	156	1	0	37	50	36
FMINUNC1	**func1**	149	26	0	139	32	0	11	41	30
FMINUNC2	**func2**	149	28	2	141	32	0	11	41	30
BFO	**bfo**	137	0	0	321	0	0	55	16	11
34 of 34 problems without bounds solved									mean efficiency in %	
dim∈[1,2]						# of anomalies			for cost measure	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
UOBYQA	**uob**	34	13	10	33	0	0	0	74	84
BCDFO	**bcd**	34	9	5	82	0	0	0	66	20
STBBO2	**stbbo2**	33	5	0	22	0	0	1	60	53
STBBO1	**stbbo1**	33	5	0	25	0	0	1	60	46
VSBBON	**vsbbn**	33	5	2	117	1	0	0	37	18
NMSMAX	**nmsm**	33	1	1	26	0	0	1	43	58
BFO	**bfo**	32	0	0	72	0	0	2	32	20
FMINUNC1	**func1**	32	6	0	41	0	0	2	53	27
FMINUNC2	**func2**	32	6	0	47	0	0	2	53	25
NELDER	**neld**	32	6	4	17	0	0	2	57	60
FMINUNC	**func**	31	5	0	35	0	0	3	54	30
VSBBO	**vsbb**	30	1	0	38	4	0	0	27	31
38 of 39 problems without bounds solved									mean efficiency in %	
dim∈[3,5]						# of anomalies			for cost measure	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
UOBYQA	**uob**	37	13	11	62	2	0	0	73	61
STBBO2	**stbbo2**	37	6	0	65	2	0	0	53	57
STBBO1	**stbbo1**	37	6	0	68	2	0	0	53	52
VSBBON	**vsbbn**	35	3	3	372	4	0	0	25	11
NMSMAX	**nmsm**	35	1	1	75	4	0	0	41	55
NELDER	**neld**	35	0	0	59	1	0	3	43	48
BCDFO	**bcd**	34	7	7	249	0	0	5	56	17
FMINUNC	**func**	33	10	6	100	0	0	6	54	35
VSBBO	**vsbb**	33	1	0	94	6	0	0	23	29
BFO	**bfo**	32	0	0	201	0	0	7	20	15
FMINUNC1	**func1**	30	4	0	101	7	0	2	35	23
FMINUNC2	**func2**	30	4	0	103	7	0	2	35	21
73 of 76 problems without bounds solved									mean efficiency in %	
dim∈[6,10]						# of anomalies			for cost measure	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
UOBYQA	**uob**	66	10	10	414	7	0	3	54	47
STBBO2	**stbbo2**	66	11	0	99	10	0	0	48	57
STBBO1	**stbbo1**	66	11	0	97	10	0	0	48	55
NELDER	**neld**	62	5	5	408	2	0	12	26	22
VSBBO	**vsbb**	61	3	3	273	15	0	0	26	26
BCDFO	**bcd**	59	16	15	3640	0	0	17	42	9
NMSMAX	**nmsm**	58	3	3	237	15	0	3	25	30
VSBBON	**vsbbn**	57	7	7	1019	19	0	0	29	11
FMINUNC	**func**	50	17	13	172	1	0	25	44	38
FMINUNC1	**func1**	49	9	0	168	21	0	6	34	28
FMINUNC2	**func2**	49	9	0	168	21	0	6	34	30
BFO	**bfo**	41	0	0	280	0	0	35	12	9
43 of 43 problems without bounds solved									mean efficiency in %	
dim∈[11,30]						# of anomalies			for cost measure	
solver		solved	#100	!100	Tmean	#n	#t	#f	nf	msec
VSBBO	**vsbb**	43	3	3	471	0	0	0	33	28
STBBO2	**stbbo2**	40	15	2	199	3	0	0	69	77
STBBO1	**stbbo1**	40	17	3	217	3	0	0	71	76
FMINUNC	**func**	40	13	5	275	0	0	3	53	41
UOBYQA	**uob**	39	0	0	3000	4	0	0	30	20
FMINUNC1	**func1**	38	7	0	212	4	0	1	49	44
FMINUNC2	**func2**	38	9	2	215	4	0	1	49	44
VSBBON	**vsbbn**	38	4	4	1685	5	0	0	29	9
NELDER	**neld**	35	0	0	776	8	0	0	15	11
NMSMAX	**nmsm**	33	1	1	261	10	0	0	19	20
BFO	**bfo**	32	0	0	742	0	0	11	6	5
BCDFO	**bcd**	30	4	4	22578	0	12	1	21	1

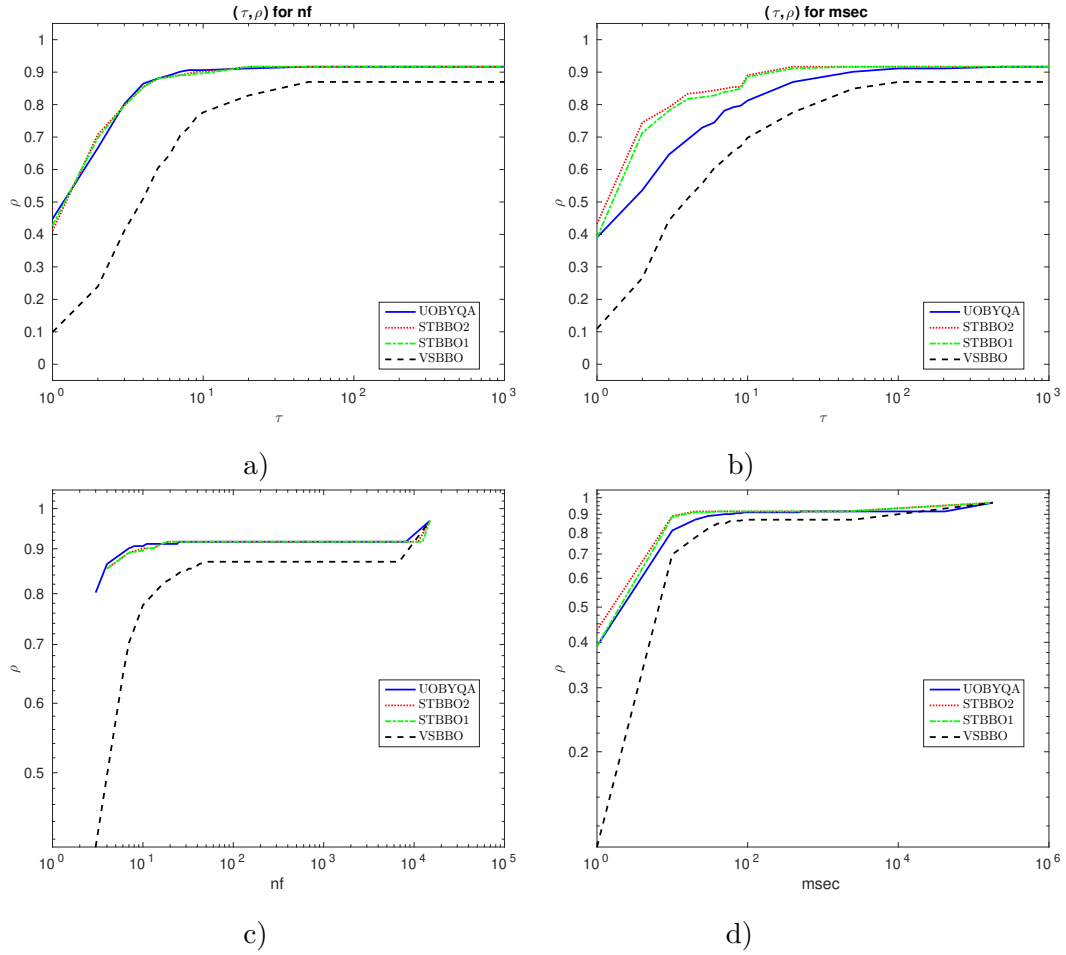


Figure 9.2: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 9.3 and Figure 9.3 provide the results for $1 \leq n \leq 30$ with the large budget $\texttt{nfmax} = 1000n$:
• For $1 \leq n \leq 30$ **STBBO1**, **STBBO2**, and **UOBYQA** are the best in terms of the `nf` efficiency and the number of solved problems.
• For $1 \leq n \leq 2$ **UOBYQA** and **BCDFO** are the two best solvers.
• For $3 \leq n \leq 5$ **UOBYQA** and **STBBO1** (**STBBO2**) are the two best solvers in terms of the number of solved problems and the `nf` efficiency.
• For $6 \leq n \leq 10$ **STBBO1** (**STBBO2**) are the two best solvers in terms of the number of solved problems and **UOBYQA** is the best in terms of the `nf` efficiency.
• For $11 \leq n \leq 30$ **VSBBO** is the best in terms of the number of solved problems and **STBBO1** is the best solvers in terms of `nf` efficiency.

What is interesting in Tables 9.1-9.3 and Figures 9.1-9.3 is that **STBBO1** and **STBBO2** are more efficient than **FMINUNC** using the standard BFGS method and **FMINUNC1** and **FMINUNC2** using the limited memory BFGS in terms of the number of solved problem and the `nf` efficiency.

Table 9.3: Results for $1 \leq n \leq 30$ with $\texttt{nfmax} = 1000n$

stopping test:	$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 1000*n					
188 of 192 problems without bounds solved									mean efficiency in %	
dim∈[1,30]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	179	35	31	1339	9	1	3	56	50
STBBO2	**stbbo2**	177	38	2	147	14	0	1	56	59
STBBO1	**stbbo1**	177	40	3	148	14	0	1	56	60
VSBBON	**vsbbn**	174	16	13	1295	18	0	0	29	10
VSBBO	**vsbb**	169	12	11	248	23	0	0	28	28
NELDER	**neld**	166	11	9	447	3	0	23	32	31
NMSMAX	**nmsm**	163	6	6	254	22	0	7	31	38
BCDFO	**bcd**	160	35	30	6603	0	14	18	45	12
FMINUNC1	**func1**	157	27	0	237	13	0	22	41	30
FMINUNC2	**func2**	156	29	2	211	14	0	22	41	30
FMINUNC	**func**	154	45	23	161	0	0	38	50	36
BFO	**bfo**	142	0	0	420	0	0	50	16	12
34 of 34 problems without bounds solved									mean efficiency in %	
dim∈[1,2]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	34	13	10	24	0	0	0	74	69
BCDFO	**bcd**	34	9	5	66	0	0	0	66	23
STBBO2	**stbbo2**	33	5	0	23	0	0	1	60	51
STBBO1	**stbbo1**	33	5	0	23	0	0	1	60	50
VSBBON	**vsbbn**	33	5	2	122	1	0	0	37	14
NMSMAX	**nmsm**	33	1	1	26	0	0	1	43	60
BFO	**bfo**	32	0	0	75	0	0	2	32	20
FMINUNC1	**func1**	32	6	0	44	0	0	2	53	20
FMINUNC2	**func2**	32	6	0	44	0	0	2	53	22
VSBBO	**vsbb**	32	1	0	52	2	0	0	28	24
NELDER	**neld**	32	6	4	18	0	0	2	57	52
FMINUNC	**func**	31	5	0	35	0	0	3	54	31
38 of 39 problems without bounds solved									mean efficiency in %	
dim∈[3,5]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
UOBYQA	**uob**	38	12	11	95	1	0	0	73	69
STBBO2	**stbbo2**	36	6	0	70	3	0	0	50	47
STBBO1	**stbbo1**	36	6	0	67	3	0	0	50	52
VSBBON	**vsbbn**	36	2	2	428	3	0	0	25	10
NMSMAX	**nmsm**	36	1	1	92	3	0	0	42	55
BCDFO	**bcd**	35	6	6	353	0	0	4	55	18
NELDER	**neld**	35	0	0	59	1	0	3	41	49
FMINUNC	**func**	33	9	5	101	0	0	6	52	37
VSBBO	**vsbb**	33	4	4	132	6	0	0	28	31
BFO	**bfo**	32	0	0	203	0	0	7	20	14
FMINUNC1	**func1**	31	4	0	133	0	0	8	34	25
FMINUNC2	**func2**	31	4	0	140	0	0	8	34	20
73 of 76 problems without bounds solved									mean efficiency in %	
dim∈[6,10]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO2	**stbbo2**	68	12	0	215	8	0	0	49	58
STBBO1	**stbbo1**	68	12	0	214	8	0	0	49	59
UOBYQA	**uob**	67	10	10	576	6	0	3	54	48
VSBBON	**vsbbn**	65	6	6	1581	11	0	0	28	10
VSBBO	**vsbb**	62	3	3	334	14	0	0	24	27
NELDER	**neld**	62	5	5	416	0	0	14	26	25
BCDFO	**bcd**	61	16	15	6269	0	1	14	42	11
NMSMAX	**nmsm**	61	3	3	467	9	0	6	26	29
FMINUNC1	**func1**	55	10	0	349	10	0	11	35	31
FMINUNC2	**func2**	55	10	0	342	10	0	11	35	31
FMINUNC	**func**	50	18	13	178	0	0	26	45	37
BFO	**bfo**	45	0	0	520	0	0	31	12	10
43 of 43 problems without bounds solved									mean efficiency in %	
dim∈[11,30]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	42	4	4	361	1	0	0	34	32
UOBYQA	**uob**	40	0	0	4918	2	1	0	31	19
STBBO2	**stbbo2**	40	15	2	202	3	0	0	69	76
STBBO1	**stbbo1**	40	17	3	211	3	0	0	71	77
FMINUNC	**func**	40	13	5	285	0	0	3	53	39
VSBBON	**vsbbn**	40	3	3	2579	3	0	0	28	6
FMINUNC1	**func1**	39	7	0	321	3	0	1	49	40
FMINUNC2	**func2**	38	9	2	221	4	0	1	49	43
NELDER	**neld**	37	0	0	1236	2	0	4	15	9
BFO	**bfo**	33	0	0	829	0	0	10	6	6
NMSMAX	**nmsm**	33	1	1	264	10	0	0	19	21
BCDFO	**bcd**	30	4	4	21985	0	13	0	23	1

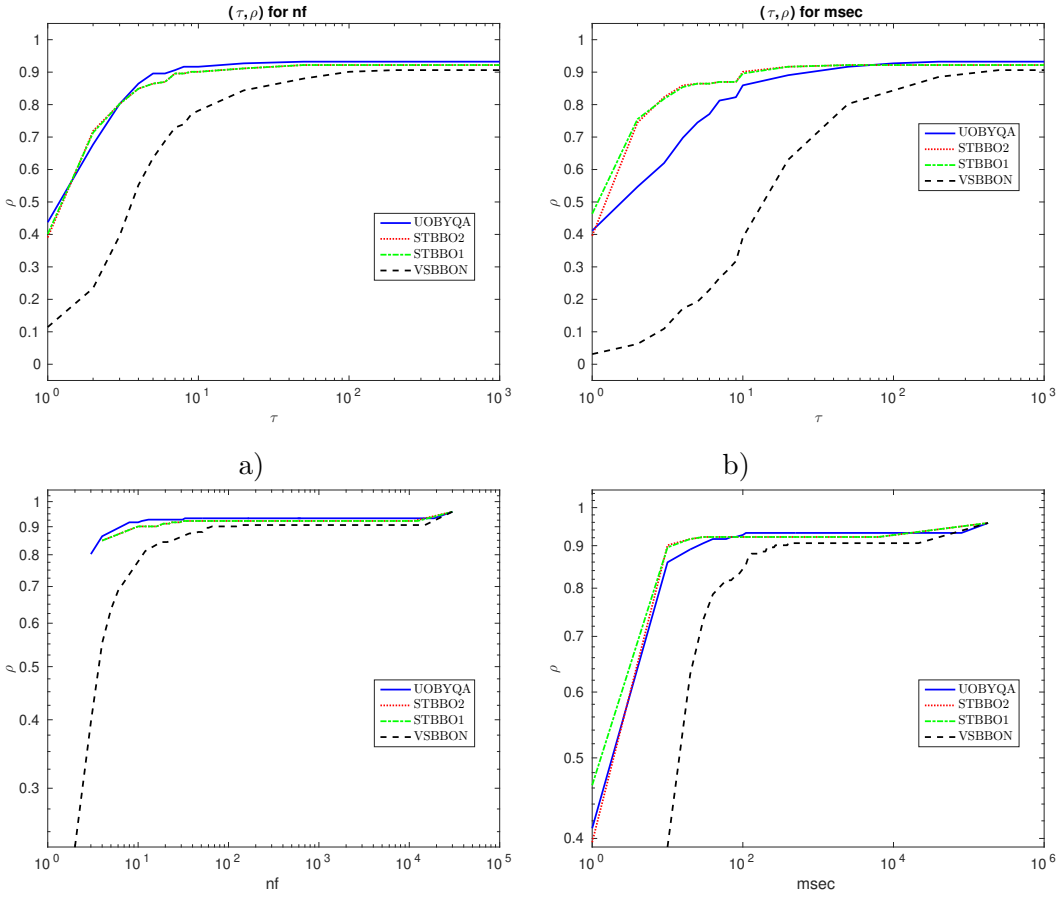


Figure 9.3: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 9.4 and Figure 9.4 report the results for $31 \leq n \leq 1000$ with the small budget `nfmax` $= 100n$:
• **STBBO1** and **STBBO2** are the two best solvers in terms of the `nf` efficiency.
• **VSBBO** and **VSBBON** are the two best solvers in terms of the number of solved problems.
• Further analysis shows that **STBBO1** and **STBBO2** are comparable with **FMINUNC** using the standard BFGS method and more efficient than **FMINUNC1** and **FMINUNC2** using the limited memory BFGS in terms of the number of solved problems and the `nf` efficiency.

Table 9.4: Results for $31 \leq n \leq 1000$ with `nfmax` $= 100n$

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq$ `100*n`										
215 of 267 problems without bounds solved										mean efficiency in %
dim∈[31,1000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	184	20	18	4700	81	2	0	32	27
VSBBON	**vsbbn**	176	58	57	6290	89	2	0	40	14
STBBO2	**stbbo2**	170	66	9	1189	96	1	0	45	49
STBBO1	**stbbo1**	166	59	2	1106	100	1	0	43	48
FMINUNC	**func**	161	68	46	1709	59	0	47	43	36
FMINUNC1	**func1**	155	37	0	1255	98	1	13	34	37
FMINUNC2	**func2**	153	38	1	1338	101	1	12	33	38

Our results, for $31 \leq n \leq 1000$ with the medium budget `nfmax` $= 500n$, are set out in Table 9.5 and Figure 9.5:
• **STBBO1** is the best solver in terms of the `nf` efficiency
• **VSBBO** is the best solver in terms of the number.
• **STBBO1** and **STBBO2** are slightly better than **FMININC** using the standard BFGS method in terms of the number of solved problems and the `nf` efficiency.
• They are more efficient than **FMINUNC1** and **FMINUNC2** using the traditional limited memory BFGS method.

Table 9.5: Results for $31 \leq n \leq 1000$ with `nfmax` $= 500n$

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq$ `500*n`										
237 of 267 problems without bounds solved										mean efficiency in %
dim∈[31,1000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	214	32	30	7564	29	24	0	38	30
FMINUNC	**func**	201	80	59	4845	13	1	52	49	41
STBBO2	**stbbo2**	200	67	10	3042	66	1	0	49	56
VSBBON	**vsbbn**	197	54	53	11371	26	44	0	42	15
STBBO1	**stbbo1**	194	59	3	2469	72	1	0	47	54
FMINUNC1	**func1**	181	37	0	2719	64	1	21	36	41
FMINUNC2	**func2**	178	39	2	2554	67	1	21	36	41

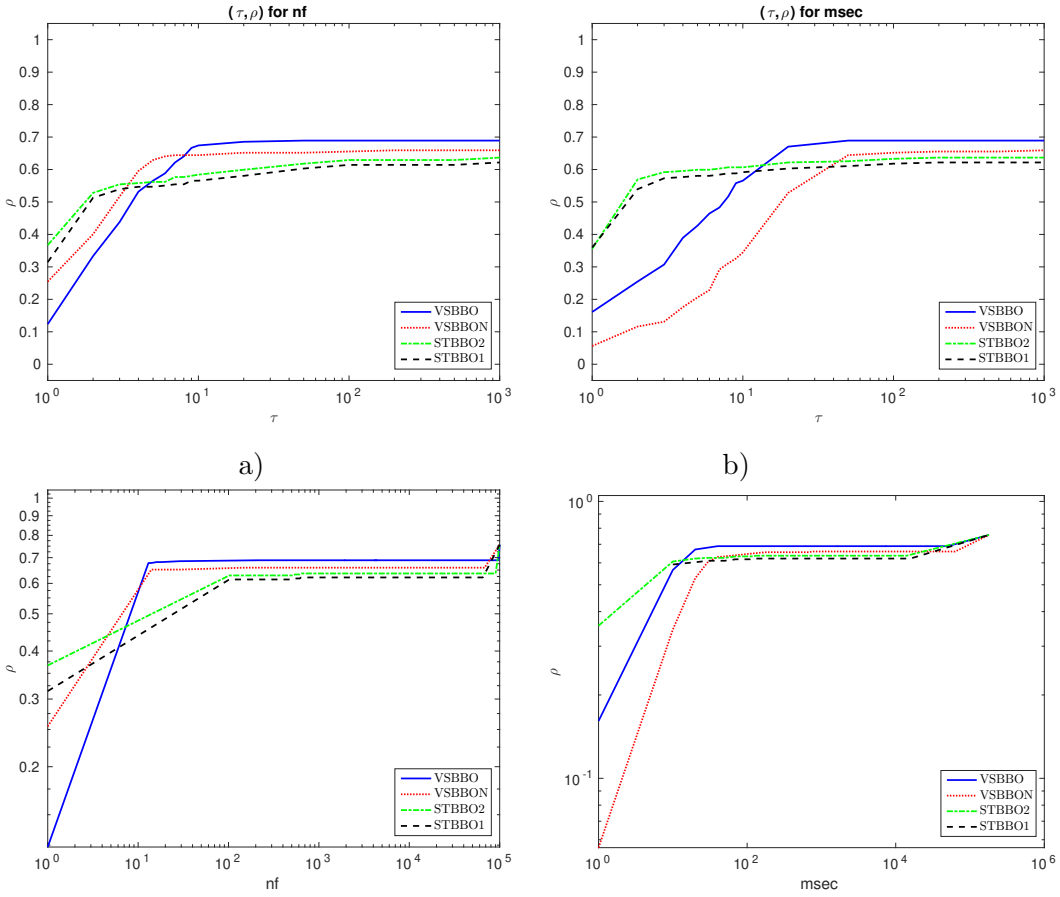


Figure 9.4: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

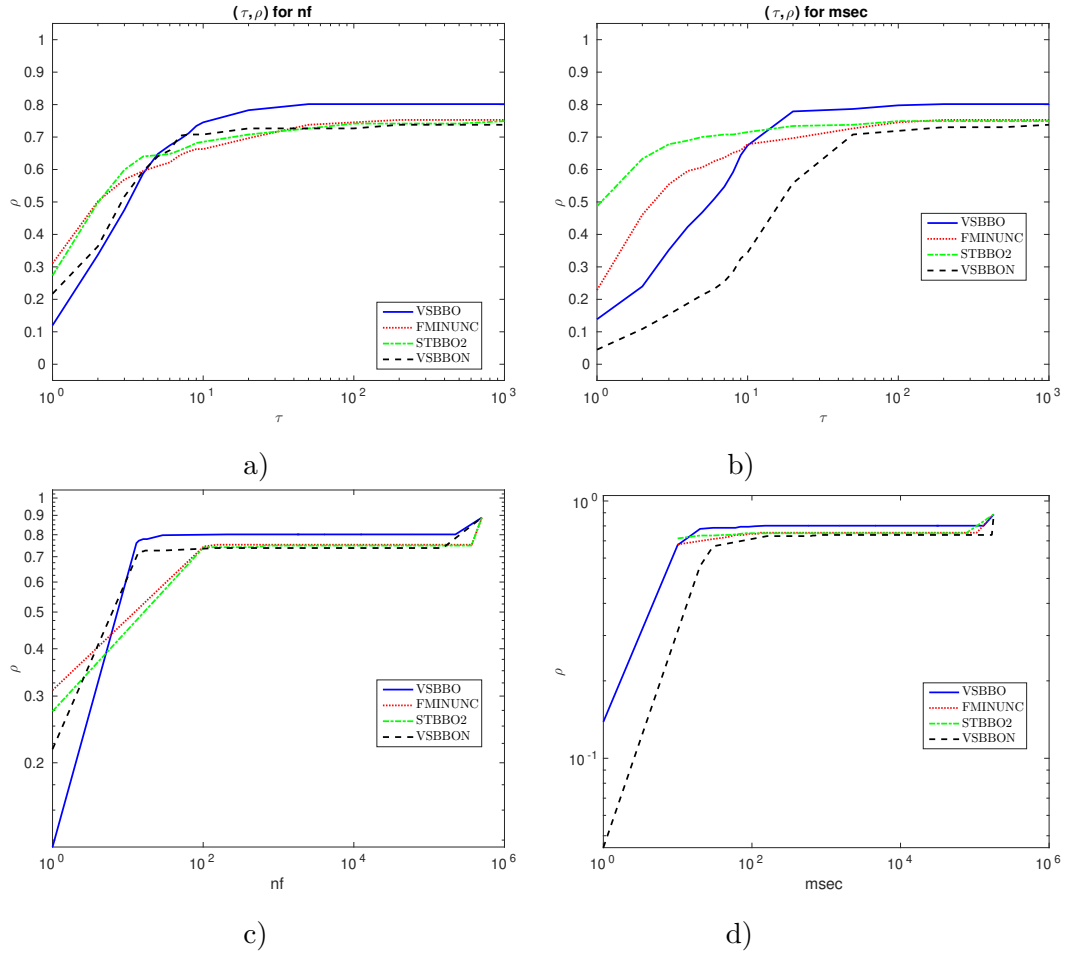


Figure 9.5: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

The results, as Table 9.6 and Figure 9.6 for $31 \leq n \leq 1000$ with the large budget $\texttt{nfmax} = 1000n$, indicate that
• **STBBO1** is the best solver in terms of the `nf` efficiency while **VSBBO** is the best solver in terms of the number of solved problems,
• **STBBO1** and **STBBO2** have approximately the same behaviour as **FMINUNC** using the standard BFGS method in terms of the number of solved problems and the `nf` efficiency,
• **STBBO2** are more efficient than **FMINUNC1** and **FMINUNC2** using the standard limited memory BFGS method.

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq$ `1000*n`										
239 of 267 problems without bounds solved										mean efficiency in %
dim∈[31,1000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	214	30	29	6226	23	30	0	37	32
STBBO2	**stbbo2**	211	68	13	4099	54	1	1	50	57
STBBO1	**stbbo1**	203	58	3	3609	62	2	0	47	56
FMINUNC	**func**	203	80	59	5729	4	6	54	50	42
VSBBON	**vsbbn**	199	57	56	10542	21	47	0	42	14
FMINUNC1	**func1**	189	37	1	4171	34	1	43	37	41
FMINUNC2	**func2**	188	37	1	4588	45	2	32	36	41

Table 9.6: Results for $31 \leq n \leq 1000$ with $\texttt{nfmax} = 1000n$

The results, for $1001 \leq n \leq 9000$ with the small budget $\texttt{nfmax} = 100n$, are summarized in Table 9.7 and Figure 9.7 with the goal of identifying the best solver:
• **STBBO1** and **STBBO2** are the two best solvers in terms of the `nf` efficiency and the number of solved problems,
• They are more efficient than **FMINUNC** using the standard BFGS method in terms of the number of solved problems and the `nf` efficiency.

Table 9.7: Results for $1001 \leq n \leq 9000$ with $\texttt{nfmax} = 100n$

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq$ `100*n`										
83 of 109 problems without bounds solved										mean efficiency in %
dim∈[1001,9000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO1	**stbbo1**	76	49	10	31910	5	28	0	59	62
STBBO2	**stbbo2**	75	43	4	30378	5	29	0	58	63
VSBBO	**vsbb**	59	12	12	68356	12	38	0	25	18
FMINUNC	**func**	43	22	18	36889	19	35	12	29	25

Table 9.8 and Figure 9.8 report the results for $1001 \leq n \leq 9000$ with the medium budget $\texttt{nfmax} = 500n$:
• **STBBO1** and **STBBO2** are the two best solvers in terms of the `nf` efficiency and the number of solved problems.
• They are more efficient than solver using the standard BFGS method in terms of the number of solved problems and the `nf` efficiency.

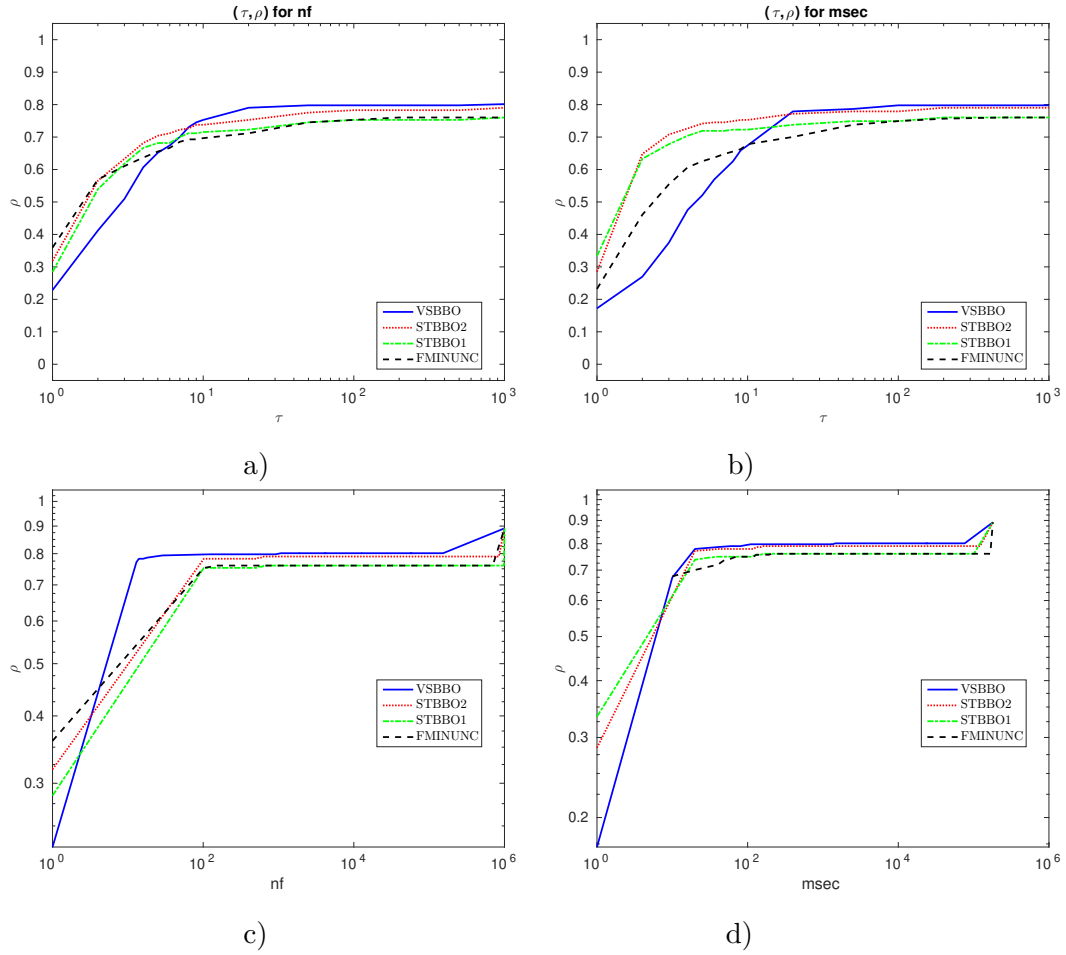


Figure 9.6: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

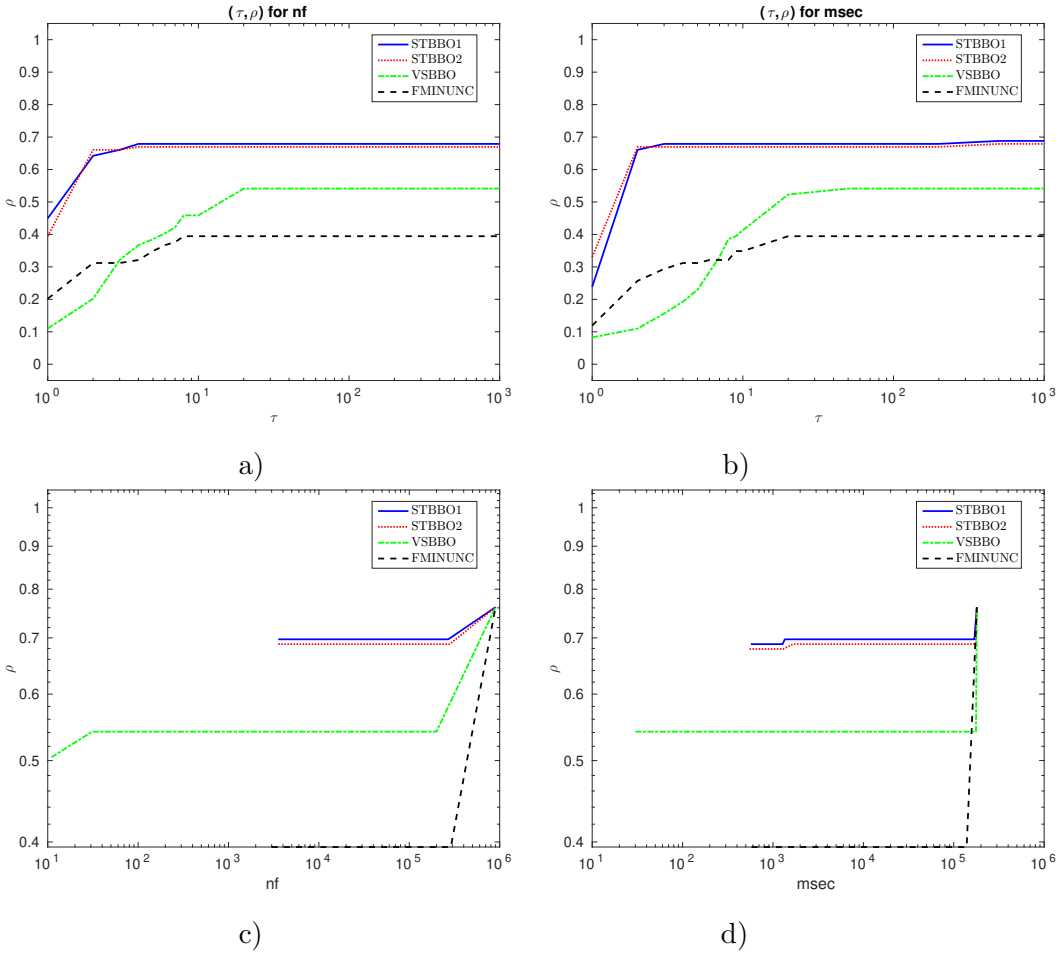


Figure 9.7: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

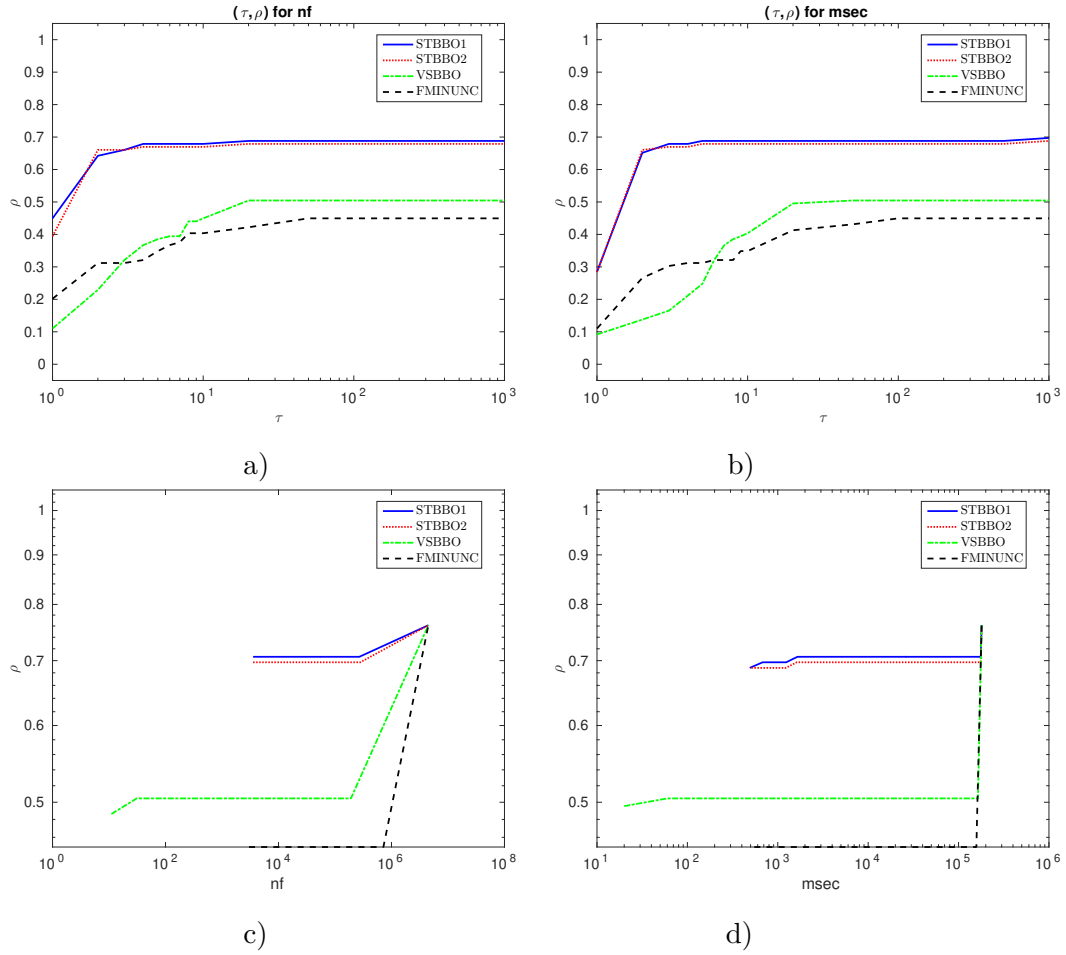


Figure 9.8: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 9.8: Results for $1001 \leq n \leq 9000$ with $\texttt{nfmax} = 500n$

stopping test:	$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 500*n					
83 of 109 problems without bounds solved									mean efficiency in %	
dim∈[1001,9000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO1	**stbbo1**	77	49	10	32248	3	29	0	59	62
STBBO2	**stbbo2**	76	43	4	31221	3	30	0	58	62
VSBBO	**vsbb**	55	12	12	55052	2	52	0	25	18
FMINUNC	**func**	49	22	18	42801	4	42	14	29	25

Summarized in Table 9.9 and shown in Figure 9.9 are the results for $1001 \leq n \leq 9000$ with the large budget $\texttt{nfmax} = 1000n$:
• **STBBO2** and **STBBO1** are the two best solvers in terms of the nf efficiency and the number of solved problems.
• They are more efficient than **FMINUNC** using the standard BFGS method in terms of the number of solved problems.

Table 9.9: Results for $1001 \leq n \leq 9000$ with $\texttt{nfmax} = 1000n$

stopping test:	$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 1000*n					
83 of 109 problems without bounds solved									mean efficiency in %	
dim∈[1001,9000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO1	**stbbo1**	77	48	9	31925	0	32	0	59	63
STBBO2	**stbbo2**	76	43	4	30615	0	33	0	58	63
VSBBO	**vsbb**	61	13	13	65181	0	48	0	27	19
FMINUNC	**func**	49	22	18	43160	0	46	14	29	25

Table 9.10 and Figure 9.10 show the results for $1 \leq n \leq 9000$ with the small budget $\texttt{nfmax} = 100n$:
• **STBBO1** and **STBBO2** are the two best solvers in terms of the nf efficiency and the number of solved problems.
• Interestingly, they are more efficient than **FMINUNC** using the standard BFGS method.

Table 9.11 and Figure 9.11 provide the results for $1 \leq n \leq 9000$ with the medium budget $\texttt{nfmax} = 500n$:
• **STBBO2** and **STBBO1** are the two best solvers in terms of the nf efficiency and the number of solved problems.
• As a consequence, they are more efficient than **FMINUNC** using the standard BFGS method.

Table 9.12 and Figure 9.12 show the results for $1 \leq n \leq 9000$ with the large budget $\texttt{nfmax} = 1000n$. **STBBO1** and **STBBO2** are the two best solvers in terms of the nf efficiency and the number of solved problems.

From this result, **STBBO1** and **STBBO2** are more robust than **FMINUNC** using the standard BFGS method in terms of the number of solved problems and the nf efficiency.

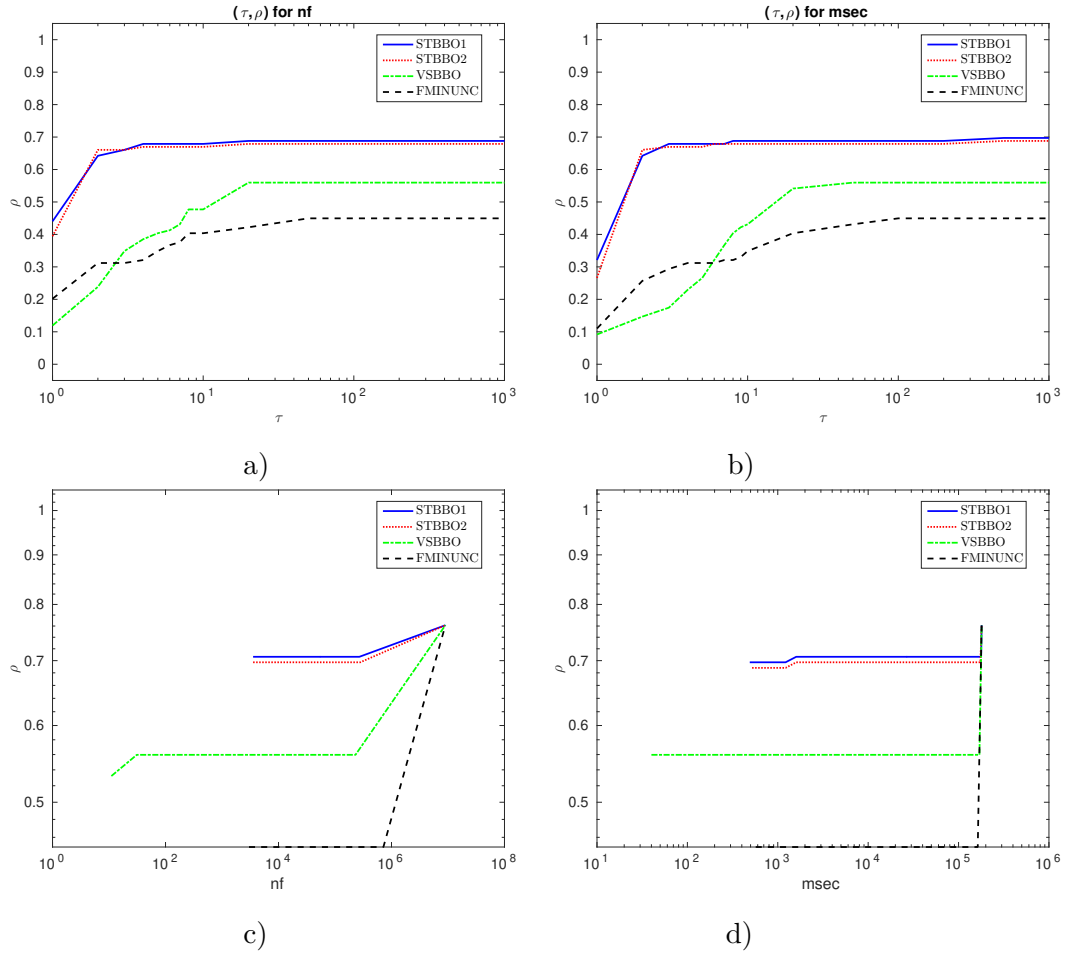


Figure 9.9: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 9.10: Results for $1 \leq n \leq 9000$ with $\texttt{nfmax} = 100n$

stopping test:		$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 100*n				
472 of 568 problems without bounds solved										mean efficiency in %
dim∈[1,9000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO2	**stbbo2**	402	222	19	6191	136	30	0	58	61
STBBO1	**stbbo1**	399	222	18	6561	140	29	0	57	59
VSBBO	**vsbb**	377	84	82	13033	151	40	0	34	28
FMINUNC	**func**	348	186	147	5388	93	35	92	48	38

Table 9.11: Results for $1 \leq n \leq 9000$ with $\texttt{nfmax} = 500n$

stopping test:		$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 500*n				
505 of 568 problems without bounds solved										mean efficiency in %
dim∈[1,9000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO2	**stbbo2**	452	222	21	6635	84	31	1	62	67
STBBO1	**stbbo1**	447	222	21	6668	90	30	1	61	65
VSBBO	**vsbb**	436	98	96	10752	56	76	0	39	31
FMINUNC	**func**	404	200	165	7661	18	43	103	52	41

Table 9.12: Results for $1 \leq n \leq 9000$ with $\texttt{nfmax} = 1000n$

stopping test:		$q_f \leq$ 0.0001,		sec $\leq$ 180,		nf $\leq$ 1000*n				
508 of 568 problems without bounds solved										mean efficiency in %
dim∈[1,9000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
STBBO2	**stbbo2**	464	226	23	6934	68	34	2	62	67
STBBO1	**stbbo1**	457	223	19	7039	76	34	1	61	67
VSBBO	**vsbb**	444	101	100	12050	46	78	0	39	33
FMINUNC	**func**	406	200	162	8134	4	52	106	52	42

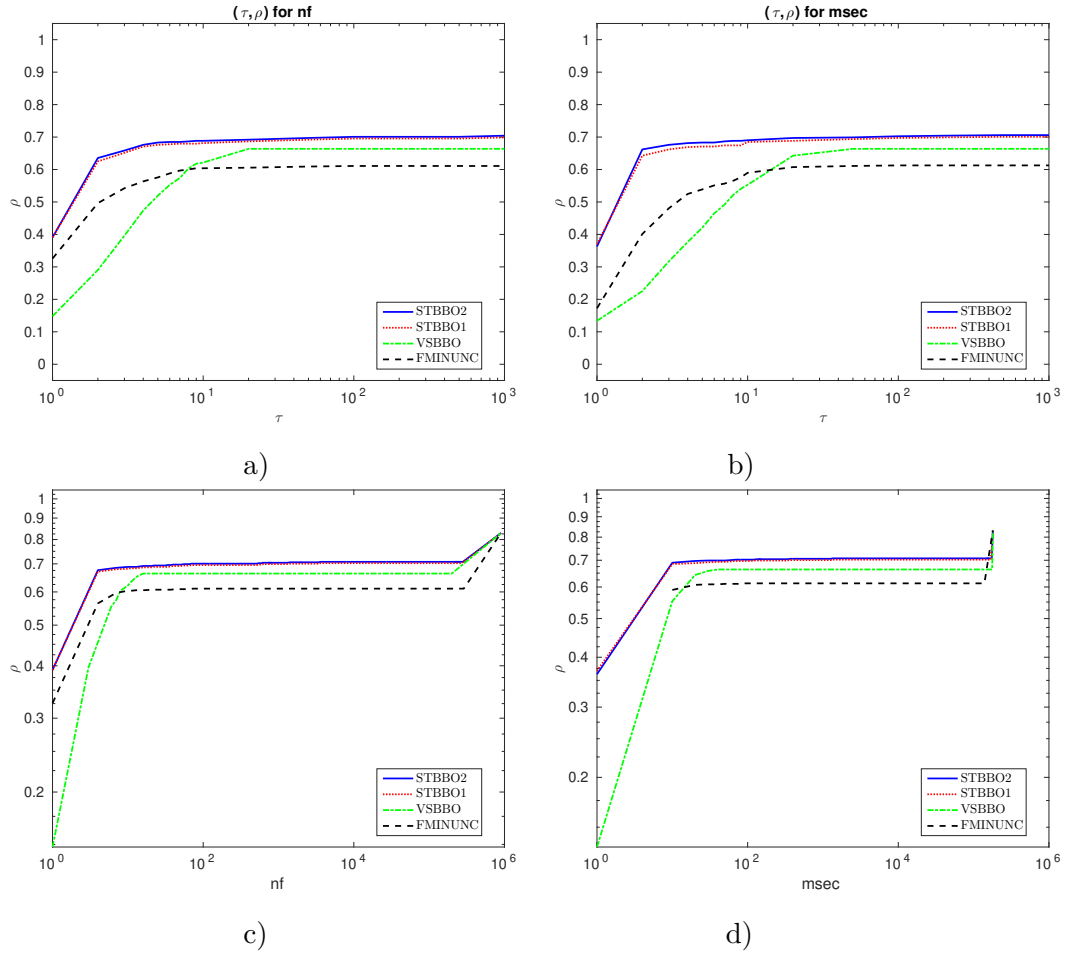


Figure 9.10: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

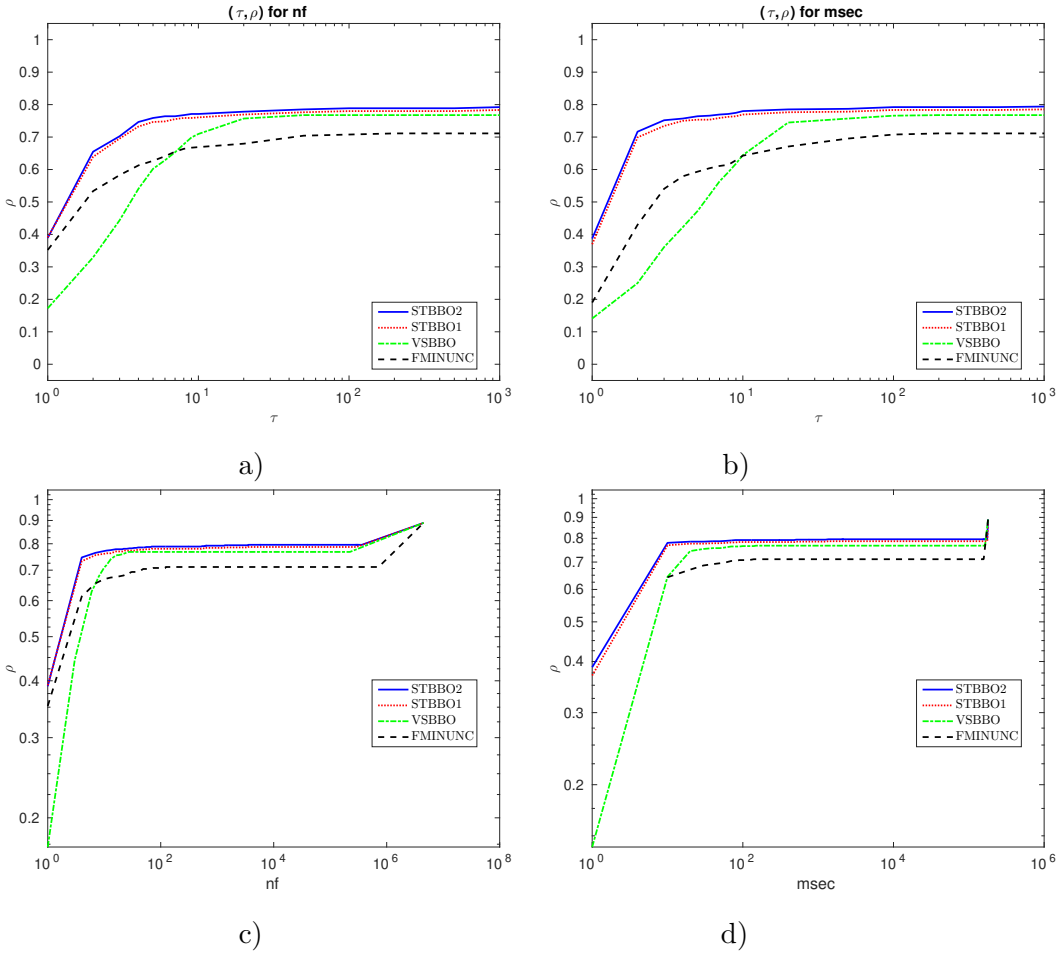


Figure 9.11: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

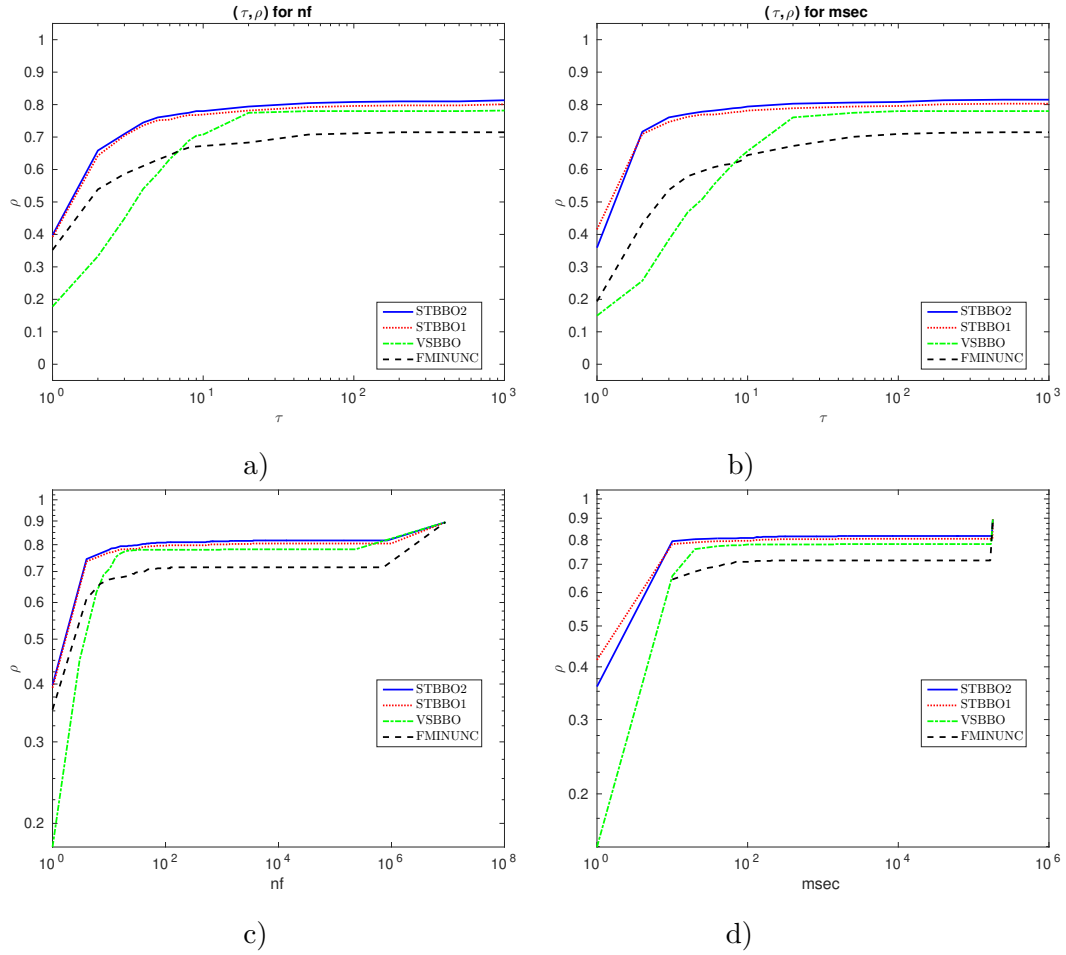


Figure 9.12: (a) and (b): Performance profiles for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored. (c) and (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

10 A new noisy black box optimization methods

This section discusses a noisy randomized line search method for unconstrained black box optimization, called **VSBBON**. This is my own work (cf. Kimiaei [114]). Complexity bounds are investigated in the presence of noise for nonconvex, convex, and strongly convex functions. Two effective ingredients of **VSBBON** are an improved derivative-free line search algorithm with many heuristic enhancements and quadratic models in adaptively determined subspaces. Numerical results are given showing that **VSBBON** is robust and efficient.

10.1 Overview of the new method

We propose a new solver for noisy unconstrained black box optimization – called **Vienna noisy randomized black box optimization** (**VSBBON**). In terms of classifications done by Larson et al. [122] and Rios & Sahinidis [151], our new solver **VSBBON** is a local, model-based, and randomized solver. It is an adaptation of our recent solver **VSBBO** (Kimiaei & Neumaier [117]) to the noisy case preserving the main structure of **VSBBO**, namely, a multi-line search algorithm and standard random subspace directions. But unlike **VSBBO**, **VSBBON** constructs surrogate quadratic models in adaptively determined subspaces and finds step sizes by variants that are more robust in the presence of substantial amounts of noise.

VSBBON constructs surrogate quadratic models in adaptively determined subspaces that can handle medium and large scale problems. Although these models have lower accuracy in higher dimensions, their usefulness in the presence of a high level of noise is confirmed in extensive numerical experiments. Moreover, **VSBBON** makes repeated calls to an improved line search algorithm which is likely to decrease the function value, using heuristics to find, update, and restart step sizes. Improved line searches are done along either random approximate coordinate, perturbed random or improved trust region directions. For randomized black box optimization methods, the worst case complexity is known [9] to be better by a factor of n than that of deterministic methods; hence using random directions seems preferable to using deterministic ones. Even better directions are random approximate coordinate directions. Improved trust region directions are found by minimizing surrogate quadratic models in adaptively determined subspaces inside a trust region. Perturbed random directions are perturbations of random directions by scaled descent directions in adaptively determined subspaces. These directions are useful in the presence of a high level of noise.

Our numerical results show that **VSBBON** is more robust and efficient than competitive global solvers, although it cannot guarantee to find a global optimum.

In Section 2, we describe a basic version of our new algorithm. Complexity bounds for nonconvex, convex, and strongly convex functions are proved in Section 3. Section 4 defines the final version

of our new algorithm, enriched by some new heuristic techniques. Summarized in Section 5 is a comparison among **VSBBON** with **VSBBO** by Kimiaei & Neumaier [117], **SDBOX** by Lucidi & Sciandrone [126], **NMSMAX** by Higham [99], **DSPFD** by Gratton et al. [84], **BFO** by Porcelli & Toint [147], **MCS** by Huyer & Neumaier [102], **BCDFO** by Gratton et al. [85], **UOBYQA** by Powell [149], and **FMINUNC** by the Matlab Optimization Toolbox for small, medium and large scale problems. The **VSBBON** package is publicly available at

`https://www.mat.univie.ac.at/~kimiaei/software/VSBBON`.

This package includes supplemental material, called `supplMat.pdf`, and data structure, called `dataStruct.pdf`.

10.2 A randomized algorithm for the noisy case

This section introduces in Subsection 10.2.1 a basic randomized line search algorithm, called **ILS-basic**, in Subsection 10.2.2 a basic randomized descent algorithm, called **DS-basic**, and in Subsection 10.2.3 a basic version of **VSBBON**, called **VSBBON-basic**. **VSBBON-basic** has repeated calls to **DS-basic** which alternates calls to **ILS-basic** using scaled random directions.

10.2.1 A basic randomized line search algorithm

In this subsection, we introduce a basic version of our improved line search algorithm – called **ILS-basic** – using **scaled random directions**. The standard random direction is the random direction p uniformly **i.i.d.** in $[-\frac{1}{2},\frac{1}{2}]^n$. Here i.i.d. stands for the independent and identically distributed. The **scaled random direction** is a standard random direction p scaled by $\gamma^{\rm rd}/\|p\|$, where $0<\gamma^{\rm rd}<1$ is a tiny tuning parameter, resulting in $\|p\|=\gamma^{\rm rd}$. Here given a positive scaling vector $s\in\mathbb{R}^n$ (fixed in throughout the paper), we define the scaled 2-norm $\|p\|$ of $p\in\mathbb{R}^n$ and the dual norm $\|g\|_*$ of $g\in\mathbb{R}^n$ by

$$\|p\|:=\sqrt{\sum_i p_i^2/s_i^2}\ \text{ and }\ \|g\|_*:=\sqrt{\sum_i s_i^2 g_i^2}.$$

Essential for our complexity bounds is the following result (Proposition 6.2.2) for the unknown exact gradient $g(x)$ of $f(x)$ at $x\in\mathbb{R}^n$.

10.2.1 Proposition. *Scaled random directions p satisfy with probability $\geq \frac{1}{2}$ the inequality* (8.18)*; that is*

$$\|g(x)\|_*\|p\|\leq 2\sqrt{cn}|g(x)^Tp|,$$

with a positive constant $c\approx 4/7$.

Other kinds of directions, improved trust region, perturbed random, and random subspace directions, are also used in **VSBBON**. They are described in Section 10.4 since they play no role in the complexity analysis.

At the rth iteration of **ILS-basic**, we denote by $\widetilde{f}^{\rm best}:=\widetilde{f}(x^{\rm best})$ the inexact best function value at the current best point $x^{\rm best}$, by

$$x^{\rm trial}:=x^{\rm best}+\alpha^r p^r \tag{10.1}$$

the current trial point, and by $\widetilde{f}^{\rm trial} := \widetilde{f}(x^{\rm trial})$ its current inexact function value. Then $\widetilde{f}^{\rm best} - \widetilde{f}^{\rm trial}$ is called the **gain** along the rth search direction p^r. In fact, the current trial point can be chosen as the new best point if the condition

$$\widetilde{f}^{\rm best} - \widetilde{f}^{\rm trial} > \Delta_f^r := \gamma(\alpha^r)^2 \tag{10.2}$$

holds, where $0 < \gamma < 1$ is a tuning parameter. Then we say that the **sufficient gain** is found at the rth iteration.

Before describing the structure of our randomized line search algorithm, we explain the **extrapolation** whose goal is to expand step sizes along a fixed direction, compute the corresponding trial points and inexact function values as long as sufficient gains are found. Once no sufficient gain is found, the extrapolation ends. These step sizes are called **extrapolation step sizes**. In fact, extrapolations increase the convergence speed to reach a minimizer. In each **ILS-basic**, they are initialized by another step size δ; discussed later in Section 10.2.3.

We now describe how **ILS-basic** works in each iteration. During an iteration of **ILS-basic**, one of the following is done:
• There is no sufficient gain along directions $\pm p$; hence the corresponding step size is reduced.
• There exists a sufficient gain along the direction p or its opposite direction; extrapolation is tried while expanding its step sizes.

ILS-basic tries to find sufficient gains along scaled random directions. It takes the oldest best point $x^{\rm best}$ and its inexact function value $\widetilde{f}^{\rm best}$, step size δ, and maximal number of function evaluations (`nfmax`) as input and returns a newest best point $x^{\rm best}$ and its inexact function value $\widetilde{f}^{\rm best}$ as output. It uses the following tuning parameters:
$R \geq 1$ (number of scaled random directions),
$0 < \gamma^{\rm rd} < 1$ (parameter for adjusting scaled random directions),
$\gamma^e > 1$ (factor for adjusting step sizes),
$0 < \gamma < 1$ (factor for extrapolation test),
$0 < \alpha^{\rm min} < 1$ (threshold for extrapolation step sizes).

10.2.2 Algorithm. (**ILS-basic, a basic randomized line search algorithm**)
($\rm ILSb_0$) Initialize the step size by $\alpha^1 := \delta$.
for $r = 1, \cdots, R$ **do**
($\rm ILSb_1$) Compute the **scaled random direction** p^r.
while true **do**
($\rm ILSb_2$) Compute the new trial point $x^{\rm trial}$ by (10.1) and its inexact function value $\widetilde{f}^{\rm trial}$. If `nfmax` is exceeded, **ILS-basic** ends.
($\rm ILSb_3$) If the condition (10.2) holds, a sufficient gain is found and try extrapolation by saving $\widetilde{f}^{\rm trial}$ in $\widetilde{f}^e$, expanding the step size to $\alpha^r := \gamma^e \alpha^r$; otherwise, break the while loop.
end while
($\rm ILSb_4$) If a sufficient gain is found in ($\rm ILSb_3$), replace $x^{\rm best}$ by $x^{\rm trial}$ and $\widetilde{f}^{\rm best}$ by $\widetilde{f}^e$;
otherwise, if extrapolation along $-p^r$ has not been tried already, set $p^r := -p^r$ and go to ($\rm ILSb_2$),
otherwise, reduce the step size to $\alpha^r := \max(\alpha^r/\gamma^e, \alpha^{\rm min})$ since no sufficient gain is found along $\pm p^r$.
end for

10.2.2 A randomized descent algorithm

We introduce a basic version of randomized decrease search algorithm, called **DS-basic**, to significantly improve the function value. **DS-basic** performs repeated calls (T^0 times) to **ILS-basic**. Regardless of whether sufficient gains are found or not, **ILS-basic** using R scaled random search directions is performed in the hope of finding sufficient gains.

DS-basic tries repeatedly to find sufficient gains. It takes the oldest best point x^{best} and its inexact function value $\widetilde{f}^{\text{best}}$ obtained by **ILS-basic** and the step size δ as input and returns a newest best point x^{best} and its inexact function value $\widetilde{f}^{\text{best}}$ as output. It uses the tuning parameters introduced in **ILS-basic** and maximum number of iterations ($T^0 \geq 1$).

10.2.3 Algorithm. (**DS-basic, a randomize basic decrease search algorithm**)
for $t = 1, \cdots, T^0$ **do**
(DSb_1) Perform **ILS-basic** using R **scaled random directions**.
end for

The decrease search is called **successful** if **DS-basic** finds at least a sufficient gain. Otherwise, it is called **unsuccessful**. We cannot guarantee that a sufficient gain is found in each iteration; however, we find an upper bound for the number of such successful iterations in Section 10.3.

10.2.3 The basic version of VSBBON

Using the ingredients discussed, we now formulate a basic version of **Vienna noisy randomized black box optimization** algorithm, called **VSBBON-basic**. As long as a sufficient gain is found in each **ILS-basic**, the best point is updated by **DS-basic**; otherwise, **DS-basic** stops and δ is reduced. Once δ is below a minimum threshold $\delta^{\min}$, **VSBBON-basic** ends.

VSBBON-basic solves noisy black box optimization problem (**NBBOP**) . It takes the initial point x^0 and maximal number of function evaluations (`nfmax`) as input and returns the overall best point x^{best} and its inexact function value $\widetilde{f}^{\text{best}}$ as output. It uses the tuning parameters introduced in **DS-basic** and the following tuning parameters:
$Q > 1$ (factor for reducing δ),
$\delta^{\max} > 0$ (initial value for δ),
$0 \leq \delta^{\min} < \alpha^{\min}$ (minimum threshold for δ).

10.2.4 Algorithm. (**VSBBON-basic, a basic randomized method for NBBOP**)
(Vb_1) Compute $\widetilde{f}^0 := \widetilde{f}(x^0)$. Then set $\delta := \delta^{\max}$, $x^{\text{best}} = x^0$, and $\widetilde{f}^{\text{best}} = \widetilde{f}^0$.
for $k = 1, 2, 3, \cdots$ **do**
(Vb_2) Call **DS-basic** in the hope of finding sufficient gains.
(Vb_3) If $\delta^k \leq \delta^{\min}$ or `nfmax` is reached, **VSBBON-basic** ends.
(Vb_4) If **DS-basic** cannot find a sufficient gain, reduce δ^k by the factor of Q. Otherwise, $\delta^k = \delta^{k-1}$.
end for

10.3 Limit accuracy and complexity bounds

In order to investigate the achievable limit accuracy and complexity bounds, we assume that the assumptions (BBO_1) and (BBO_2) discussed in Subsection 8.3, and (BBO_3) discussed in Subsection 9.3 hold. Note that in the noiseless case $\omega = 0$, (BBO_3) implies $\widetilde{f} = f$.

10.3.1 Known results

In the noiseless case, a summary of known worst case complexity results and corresponding references can be found in Larson et al. [122, Table 8.1]. To get $\|g(x)\|_* \leq \varepsilon$ (under the assumptions (BBO_1) and (BBO_2)), one needs
• $\mathcal{O}(\varepsilon^{-2})$ function evaluations for general case,
• $\mathcal{O}(\varepsilon^{-1})$ function evaluations for convex case,
• $\mathcal{O}(\log \varepsilon^{-1})$ function evaluations for strongly convex case. In all cases, the factors are ignored. As already mentioned in the introduction, all randomized algorithms have complexity bounds better by a factor of n than those of deterministic algorithms.

In the presence of noise, the limit accuracy of some algorithms has been investigated:
• For the **unconstrained case**, Berahas et al. [14] proved convergence results for the problem (4.2) when f is **strongly convex**. Assuming the strong convexity of f and the boundedness of noise in the approximate gradient, they proved that a quasi-Newton method with a fixed step size has the linear convergence to a neighborhood of the solution; the gradient is estimated by the forward or central finite differences. Under the additional assumption (BBO_3), they showed that a quasi-Newton method with step sizes found by a relaxed Armijo line search, called **FDLM**, has the asymptotic accuracy

$$f - \widehat{f} = \mathcal{O}(L\omega). \tag{10.3}$$

Chen [37] suggested a randomized algorithm with Gaussian directions and estimated step sizes, called **STRRS** for different kinds of noise, one of which is discussed here. Under the assumptions
(i) $\widetilde{f}(x) - f(x) = \omega(x;\zeta)$ is a stochastic noise component, where ζ is a random vector with probability distribution $\mathbb{P}(\zeta)$,
(ii) For all $x \in \mathbb{R}^n$, ω is i.i.d with bounded variance $\mathrm{var}(\omega) > 0$,
(iii) For all $x \in \mathbb{R}^n$, the noise is unbiased, i.e., $\mathbb{E}_\zeta(\omega) = 0$,
(iv) f is **convex** and (BBO_1) holds,
(v) $\mathrm{var}(\omega) \leq \mathcal{O}(\varepsilon/n)$,
STRRS needs at most $\mathcal{O}(nL\varepsilon^{-1})$ to ensure that

$$x^N := \operatorname*{argmin}_x \{f(x) \mid x \in \{x^0, \cdots, x^N\}\}$$

satisfies $\mathbb{E}[f(x^N)] - \widehat{f} \leq \varepsilon$.
• For the **bound constrained case**, Elster & Neumaier [65] introduced a grid algorithm, called **Grid**. Here we restrict their results to the unconstrained case. Under the assumptions (BBO_1)–(BBO_3), [65, Theorem 2] ensures that there exists a constant C_n such that

$$\|g(x^k)\|_* \leq C_n(2\omega/h^k + Lh^k) \quad \text{for } x^k \in \mathbb{R}^n$$

at the end of the kth refinement step, where h^k denotes the kth grid size. If $h^k := \Theta(\sqrt{\omega})$, then the best order of magnitude can be obtained for at least a point x^{best} with

$$\|g(x^{\mathrm{best}})\|_* = \mathcal{O}(C_n\sqrt{\omega}). \tag{10.4}$$

The dependence of C_n on n is not specified. Under the same assumptions, LUCIDI & SCIANDRONE [126] discuss a derivative-free line search algorithm, called **SDBOX**, only using the coordinate directions. They proved that, for any k,

$$\|g(x^k)\|_* = \mathcal{O}\Big(n^{3/2} L \mathbf{a}^k_{\max} + \frac{n\omega}{\mathbf{a}^k_{\min}}\Big), \quad \text{for } x^k \in \mathbb{R}^n.$$

Here $\mathbf{a}^k_{\min}$ and $\mathbf{a}^k_{\max}$ are minimum and maximum values for the n step sizes used along coordinate directions in the iteration k, respectively. If $\mathbf{a}^k_{\max} := \Theta(\sqrt{\omega})$, the best order of magnitude can be obtained for at least a point x^{best} with

$$\|g(x^{\text{best}})\|_* = \mathcal{O}(n^{3/2}\sqrt{\omega}). \tag{10.5}$$

The order in this bound is the same as that in (10.4).

10.3.2 Bounds for VSBBON

Under the assumptions (BBO_1)–(BBO_3), **VSBBON-basic** finds after at most
• $\mathcal{O}(R\omega^{-1})$ function evaluations for general case
• $\mathcal{O}(\sqrt{n}R\omega^{-1/2})$ function evaluations for convex case
• $\mathcal{O}(R\log\omega^{-1})$ function evaluations for strongly convex case
a point x^{best}, with a given probability arbitrarily close to 1, satisfying

$$f(x^{\text{best}}) \le \sup_{x\in\mathbb{R}^n}\{f(x) \mid f(x) \le f(x^0) \ \text{ and } \ \|g(x)\|_* = \mathcal{O}(\sqrt{n\omega})\}.$$

The following results are generalizations of Proposition 8.4.1. It is shown that if none of the search directions $\pm p$ finds a sufficient gain of at least Δ_f (threshold for the progress on f) then a useful bound for the directional derivative can be found.

10.3.1 Proposition. *Suppose that* (BBO_1)–(BBO_3) *hold. Then, for all* $x, p \in \mathbb{R}^n$ *and all* $\Delta_f \ge 0$, *at least one of the following holds:*
(i) $\widetilde{f}(x+p) < \widetilde{f}(x) - \Delta_f$,
(ii) $\widetilde{f}(x+p) > \widetilde{f}(x) + \Delta_f$ *and* $\widetilde{f}(x-p) < \widetilde{f}(x) - \Delta_f$,
(iii) $|g(x)^T p| \le \Delta_f + 2\omega + \frac{1}{2}L\|p\|^2$.

Proof. (BBO_1) results in

$$g(x)^T p - \frac{1}{2}L\|p\|^2 \le f(x+p) - f(x) \le g(x)^T p + \frac{1}{2}L\|p\|^2. \tag{10.6}$$

We assume that (iii) is violated, so that

$$|g(x)^T p| > \Delta_f + 2\omega + \frac{1}{2}L\|p\|^2. \tag{10.7}$$

By replacing p by $\mp p$ in (10.6) and using (4.4) and (10.7), we get for an appropriate choice of the sign

$$\begin{aligned}
\widetilde{f}(x \mp p) - \widetilde{f}(x) &\le f(x \mp p) - f(x) + 2\omega \\
&\le \mp g(x)^T p + \frac{1}{2}L\|p\|^2 + 2\omega \\
&= -|g(x)^T p| + \frac{1}{2}L\|p\|^2 + 2\omega < -\Delta_f.
\end{aligned}$$

Hence, by applying the lower sign, (i) is satisfied while by applying the upper sign the second half of (ii) is satisfied, and by (4.4), (10.6), and (10.7) the first half

$$\widetilde{f}(x+p) - \widetilde{f}(x) \geq f(x+p) - f(x) - 2\omega \geq g(x)^T p - \frac{1}{2}L\|p\|^2 - 2\omega > \Delta_f$$

is obtained. □

The following result is a generalization of Theorem 8.5.3 for **ILS-basic** to the noisy case. It is proven that either a gain of at least Δ_f is found or an upper bound for the exact gradient norm of at least one of points generated by **ILS-basic** is found with a given probability arbitrarily close to one though our algorithm never calculates the exact gradient.

10.3.2 Theorem. *Assume that (*BBO_1*)–(*BBO_3*) hold,* `nfmax` *is sufficiently large, and* Δ_f *is the sufficient gain (if one exists).*
(i) $\widetilde{f}$ *decreases by at least*

$$\Delta_f \max(\mathtt{nf} - 2R - 1, 0), \tag{10.8}$$

where `nf` *is the number of function evaluations in* **ILS-basic**.
(ii) Assume that $0 < \eta < 1$ *and* $R := \lceil \log_2 \eta^{-1} \rceil$. *If* $\widetilde{f}$ *does not decrease by more than* ρ*, then, with probability* $\geq 1 - \eta$*, either the original point or one of the points evaluated with better function value has a gradient* g *with*

$$\|g\|_* \leq \sqrt{cn}\Gamma(\delta), \tag{10.9}$$

where c *comes from Proposition 10.2.1 and* $\Gamma(\delta) := (2\gamma + L)\delta + 4(\gamma^e)^{R-1}\omega/\delta$.

Proof. Let $\mathcal{R} := \{1, \ldots, R\}$. Since the function value of the best point does not increase, (i) holds if $\mathtt{nf} - 2R - 1 \leq 0$. If this is not the case, then $\mathtt{nf} \geq 2R + 2$. But in the for loop of **ILS-basic**, directions p^r ($r \in \mathcal{R}$) are generated and at most two function values are computed along each direction p^r, unless an extrapolation step is performed along the last opposite search direction. In the latter case, the extrapolation step needs at least $\mathtt{nf} - 2R - 1$ additional function values, each time with a sufficient gain of at least Δ_f. Thus the total sufficient gain is at least (10.8).

To prove (ii) we assume that $\widetilde{f}$ does not decrease by more than Δ_f. We denote the rth scaled random search direction by p^r, the best point obtained before searching in direction p^r by x^r, the rth step step size by $\alpha^r = (\gamma^e)^{1-r}\delta$ and define $\Gamma_0(\alpha^r) := (2\gamma + L)\alpha^r + 4\omega/\alpha^r$. Since $\Gamma_0(\alpha^r)$ for $\alpha^r > 0$ is a convex function, we get for $r \in \mathcal{R}$

$$\Gamma^0(\alpha^r) \leq \max\{\Gamma^0(\alpha^1), \Gamma^0(\alpha^r)\} \leq \Gamma(\delta) := (2\gamma + L)\delta + 4(\gamma^e)^{R-1}\omega/\delta,$$

where $\alpha^1 := \max_{r\in\mathcal{R}}\{\alpha^r\} = \delta$ and $\alpha^R := \min_{r\in\mathcal{R}}\{\alpha^r\} = (\gamma^e)^{1-R}\delta$. Then we get from Proposition 10.3.1, for all $r \in \mathcal{R}$,

$$|g(x^r)^T p^r| \leq \gamma(\alpha^r)^2 + 2\omega + \frac{L}{2}\|p^r\|^2 < \gamma(\alpha^r)^2 + \frac{L}{2}(\alpha^r)^2 + 2\omega,$$

so that, for all $r \in \mathcal{R}$, the inequality

$$\begin{aligned} \|g(x^r)\|_* &= \|g(x^r)\|_*\|p^r\|/\alpha^r \leq 2\sqrt{cn}|g(x^r)^T p^r|/\alpha^r \\ &\leq \sqrt{cn}\Big((2\gamma + L)\alpha^r + \frac{4\omega}{\alpha^r}\Big) \leq \sqrt{cn}\Gamma(\delta) \end{aligned}$$

holds with probability $\frac{1}{2}$ or more by Proposition 10.2.1. In other words, $\|g(x^r)\|_* \le \sqrt{cn}\Gamma(\delta)$ fails with a probability $\Pr_r < \frac{1}{2}$ for any fixed $r \in \mathcal{R}$. Therefore, we find at least one of the gradients $g = g(x^r)$ $(r \in \mathcal{R})$, so that (10.9) holds, i.e., $\Pr = 1 - \prod_{r=1}^{R-1} \Pr_r \ge 1 - 2^{-R}$. □

By S^k and U^k we denote the index sets of successful and unsuccessful iterations generated by **VSBBON-basic** up to iteration k and their sizes by $|S^k|$ and $|U^k|$, respectively. S^k is finite by (BBO_1) and U^k is finite by the updating rule for δ, discussed in (Vb_4) of **VSBBON-basic**. Hence

$$\delta^{k+1} = \delta^k/Q, \quad \text{for some } Q > 1 \text{ and for } k \in U^k, \tag{10.10}$$

but $\delta^{k+1} = \delta^k$ for $k \in S^k$; hence

$$\sum_{k\in S^k} (\delta^k)^{-2} = \sum_{k\in U^k} (\delta^k)^{-2}, \tag{10.11}$$

which will be used in Subsection 10.3.2.

10.3.3 Proposition. *Assume that* (BBO_1)*–*(BBO_3) *hold,* $\delta^{\max} > 0$, $0 < \delta^{\min} < 1$, $Q > 1$, *the integer* $T^0 \ge 1$ *and let* f^0 *be the initial value of* f. N^k, S^k, *and* U^k *denote the number of function values, successful iterations, and unsuccessful iterations in* **VSBBON-basic** *up to iteration* k. *If the condition* (10.10) *holds, then:*
(i) The number of unsuccessful iterations up to iteration k *by* **VSBBON-basic** *is at most*

$$|U^k| \le 1 + \Big\lfloor \frac{\log(\delta^{\max}/\delta^k)}{\log Q} \Big\rfloor \le K := 1 + \Big\lfloor \frac{\log(\delta^{\max}/\delta^{\min})}{\log Q} \Big\rfloor. \tag{10.12}$$

(ii) The number of successful iterations up to iteration k *by* **VSBBON-basic** *is at most*

$$|S^k| \le \gamma^{-1}(\delta^{\max})^{-2}(f^0 - \widehat{f})\frac{Q^{2|U^k|} - 1}{Q^2 - 1},$$

where $\widehat{f}$ *is finite by* (BBO_1) *and* (BBO_2) *discussed in Section 10.3.*
(iii) Each iteration of **VSBBON-basic** *needs at most* $(2R+1)T^0$ *function evaluations.*
(iv) The number N^k *of function evaluations needed up to iterate* k *by* **VSBBON-basic**, *started at* x^0, *satisfies*

$$N^k \le 1 + (2R+1)T^0(|U^k| + |S^k|). \tag{10.13}$$

(v) If $0 < \eta < 1$ *and* $R := \lceil \log_2 \eta^{-1}(k+1) \rceil$ *then* **VSBBON-basic** *finds with probability* $\ge 1-\eta$ *a point* x *with*

$$\|g(x)\|_* \le \sqrt{cn} \min_{\ell=0:k} \Gamma(\delta^\ell), \tag{10.14}$$

where c *and* $\Gamma(\delta)$ *come from Proposition 10.2.1 and Theorem 10.3.2.*

Proof. (i) By (10.10), we get

$$|U^k| \le 1 + \Big\lfloor \frac{\log(\delta^{\max}/\delta^k)}{\log Q} \Big\rfloor.$$

(ii) By (10.11), we get

$$|S^k| \;=\; \sum_{\ell\in S^k} \frac{f^{\ell+1} - f^\ell}{\gamma(\delta^\ell)^2} \le \gamma^{-1}(f^0 - \widehat{f}) \sum_{\ell\in S^k} (\delta^\ell)^{-2}$$

$$= \gamma^{-1}(f^0 - \widehat{f}) \sum_{\ell \in U^k} (\delta^\ell)^{-2} = \gamma^{-1}(\delta^{\max})^{-2}(f^0 - \widehat{f}) \sum_{\ell \in U^k} Q^{2\ell - 2}$$
$$= \gamma^{-1}(\delta^{\max})^{-2}(f^0 - \widehat{f}) \frac{Q^{2|U^k|} - 1}{Q^2 - 1}.$$

(iii) In the worst case, the for loops of **DS-basic** and **ILS-basic** are repeated T^0 and R times, respectively. Using RT^0 scaled random directions and RT^0 corresponding opposite ones, **ILS-basic** cannot find a sufficient gain; however, if a sufficient gain along the last opposite direction is found, an extrapolation step is tried at most two function evaluations, only with a sufficient gain. As a result, in each call to **DS-basic**, at most $(2R+1)T^0$ function evaluations are used.
(iv) We have $N^0 := 1$. Then (i)–(iii) result in (10.13).
(v) In this case, **VSBBON-basic** cannot find a sufficient gain after k repeated calls to **DS-basic**, resulting in

$$\|g(x^\ell)\|_* \le \sqrt{cn}\Gamma(\delta^\ell), \quad \text{for all } \ell = 1, \cdots, k$$

for a point satisfying (10.14) with probability $\ge 1 - (k+1)2^{-R} \ge 1 - \eta$. □

The following result is the complexity bound for the nonconvex case.

10.3.4 Theorem. *Assume that* $0 < \eta < 1$, $R := \lceil \log_2 \eta^{-1}(K+1) \rceil$, $\delta^{\max} > 0$, $0 < \delta^{\min} < 1$, *and*

$$\delta^{\min} := \Theta(\sqrt{\omega}). \tag{10.15}$$

If (10.10) *holds,* **VSBBON-basic** *finds after at most* $\mathcal{O}\Big(R\omega^{-1}\Big)$ *function evaluations with probability* $\ge 1 - \eta$ *a point* x *with*

$$\|g(x)\|_* = \mathcal{O}(\sqrt{n\omega}). \tag{10.16}$$

The order in the bound (10.16) is the same as the orders in (10.4) and (10.5). But the factor of (10.16) is better than that in (10.5).

Proof. By (10.12), we have $\delta^K = Q^{1-K}\delta^{\max} \le \delta^{\min}$ and Proposition 10.3.3(i) and (10.15) result in

$$\begin{aligned} \Gamma(\delta^K) &= (2\gamma + L)\delta^K + 4(\gamma^e)^{R-1}\omega/\delta^K \\ &\le (2\gamma + L)Q^{1-K}\delta^{\max} + 4(\gamma^e)^{R-1}Q^{K-1}\omega/\delta^{\max} = \mathcal{O}(\sqrt{\omega}); \end{aligned}$$

hence (10.16) is obtained with probability $\ge 1 - (K+1)2^{-R} \ge 1 - \eta$ from Proposition 10.3.3(v). From (BBO$_1$) and (BBO$_2$), $\widehat{f} := \inf_k f^k$ is finite. We conclude from Propositions 10.3.3(iii) and (10.10) that

$$N^K \le \mathcal{O}(R\log(\delta^{\min})^{-1}) + \mathcal{O}(R(\delta^{\min})^{-2}) = \mathcal{O}\Big(R\omega^{-1}\Big).$$

□

The following result gives the complexity bounds for the convex and strongly convex cases.

10.3.5 Theorem. *In addition to the assumptions of Theorem 10.3.4, assume that* f *satisfies*

$$f(y) \ge f(x) + g(x)^T(y - x) + \frac{1}{2}\sigma\|y - x\| \quad \text{for } x, y \in \mathcal{L}(x^0) \tag{10.17}$$

and let $0 < \eta < 1$.

(i) If $\sigma = 0$ *(i.e., if* f *is convex),* **VSBBON-basic** *finds after at most* $\mathcal{O}\Big(\sqrt{n}R\omega^{-1/2}\Big)$ *function evaluations with probability* $\geq 1-\eta$ *a point* x *satisfying* (10.16) *and*

$$f(x) - \widehat{f} = \mathcal{O}(r^0\sqrt{n\omega}), \tag{10.18}$$

where r^0 *is given by*

$$r^0 := \sup_{x\in\mathbb{R}^n} \Big\{ \|x - \widehat{x}\| \mid f(x) \leq f(x^0) \Big\} < \infty. \tag{10.19}$$

Here $\widehat{x}$ *and* $\widehat{f} = f(\widehat{x})$ *are the global minimizer and its function value discussed in Section 10.3.*
(ii) If $\sigma > 0$ *(i.e., if* f *is strongly convex),* **VSBBON-basic** *finds after at most* $\mathcal{O}\Big(R\log\omega^{-1}\Big)$ *function evaluations with probability* $\geq 1-\eta$ *a point* x *satisfying* (10.16) *and*

$$f(x) - \widehat{f} = \frac{\mathcal{O}(n\omega)}{2\sigma}, \quad \|x - \widehat{x}\| = \frac{\mathcal{O}(\sqrt{n\omega})}{\sigma^2}. \tag{10.20}$$

The condition is the same as (10.3) found by Berahas et al. [14].

Proof. Denote $f^k = f(x^k)$, $\widetilde{f}^k = \widetilde{f}(x^k)$, and $g^k = g(x^k)$. We terminate **VSBBON** after at most K steps to satisfy (10.16). After this termination we denote by U^K and S^K the index sets of unsuccessful and successful iterations. Assume that x^k is the overall best point at the end of the kth execution of **VSBBON** satisfying (10.16). From (BBO_1) and (BBO_2), r^0 is finite. We now prove the statements (i) and (ii).

(i) The convexity of f results in $\widehat{f} \geq f^k + (g^k)^T(\widehat{x} - x^k)$. By Theorem 10.3.4, with probability $\geq 1-\eta$, we get from (10.18)

$$f^k - \widehat{f} \leq (g^k)^T(x^k - \widehat{x}) \leq \|g^k\|_*\|x^k - \widehat{x}\| = \mathcal{O}(\sqrt{n\omega}), \tag{10.21}$$

so that $f^k - f^{k+1} \leq f^k - \widehat{f} = \mathcal{O}(\sqrt{n\omega})$. Then

$$\begin{aligned}
\sum_{\ell\in S^K} \frac{\widetilde{f}^\ell - \widetilde{f}^{\ell+1}}{\gamma(\delta^\ell)^2} &\leq \sum_{\ell\in S^K} \frac{f^\ell - f^{\ell+1} + 2\omega}{\gamma(\delta^\ell)^2} \leq \frac{\mathcal{O}(\sqrt{n\omega}) + 2\omega}{\gamma} \sum_{\ell\in S^K} (\delta^\ell)^{-2} \\
&= \mathcal{O}(\sqrt{n\omega}) \sum_{\ell\in U^K} (\delta^\ell)^{-2} = \mathcal{O}(\sqrt{n\omega}) \sum_{\ell\in U^K} Q^{2\ell-2} \\
&= \mathcal{O}(\sqrt{n\omega}) \frac{Q^{2K}-1}{Q^2-1} = \mathcal{O}(\sqrt{n\omega}Q^{2K}) = \mathcal{O}(\sqrt{n}\omega^{-1/2})
\end{aligned}$$

by (10.11) and since $\mathcal{O}(\sqrt{n\omega}) + \omega = \mathcal{O}(\sqrt{n\omega})$ as ω goes to zero. Hence, we conclude from Proposition 10.3.3 that

$$N^K \leq 1 + (2R+1)T^0\Big(\mathcal{O}(\log\omega^{-1}) + \mathcal{O}(\sqrt{n}\omega^{-1/2})\Big) = \mathcal{O}(\sqrt{n}R\omega^{-1/2}).$$

(ii) When x is assumed to be fixed, the right hand side of (10.17) is a convex quadratic function in terms of y whose gradient in the components vanishes at $y_i = x_i - s_i\sigma^{-1}g_i(x)$ for $i = 1, \cdots, n$, resulting in

$$f(y) \geq f(x) - \frac{1}{2\sigma}\|g(x)\|_*^2.$$

Here as mentioned earlier $s \in \mathbb{R}^n$ is a scaling vector. By applying (10.16) in this inequality, we get

$$f^k - \widehat{f} \leq \frac{1}{2\sigma}\|g(x^k)\|_*^2 = \mathcal{O}(n\omega). \tag{10.22}$$

By replacing x by $\widehat{x}$ and y by x^k in (10.17), we get

$$f(x^k) \geq f(\widehat{x}) + \frac{\sigma}{2}\|x^k - \widehat{x}\|^2,$$

so that

$$\|x^k - \widehat{x}\|^2 \leq \frac{2}{\sigma}(f^k - \widehat{f}) = \mathcal{O}(n\omega)$$

by (10.22). From (10.22), we conclude that $f^k - f^{k+1} = \mathcal{O}(n\omega)$, resulting in

$$\begin{aligned}\sum_{\ell \in S^K} \frac{\widetilde{f}^\ell - \widetilde{f}^{\ell+1}}{\gamma(\delta^\ell)^2} &\leq \sum_{\ell \in S^K} \frac{f^\ell - f^{\ell+1} + 2\omega}{\gamma(\delta^\ell)^2} \leq \frac{\mathcal{O}(n\omega) + 2\omega}{\gamma} \sum_{\ell \in S^K} (\delta^\ell)^{-2} \\ &= \mathcal{O}(n\omega) \sum_{\ell \in U^K} (\delta^\ell)^{-2} = \mathcal{O}(n\omega) \sum_{\ell \in U^K} Q^{2\ell-2} \\ &= \mathcal{O}(n\omega)\frac{Q^{2K}-1}{Q^2-1} = \mathcal{O}(n\omega)\mathcal{O}(\omega^{-1}) = \mathcal{O}(n),\end{aligned}$$

by (10.11). Finally, we conclude from Proposition 10.3.3 that

$$N^K \leq 1 + (2R+1)T^0\Big(\mathcal{O}(\log \omega^{-1}) + \mathcal{O}(n)\Big) = \mathcal{O}(R \log \omega^{-1}).$$

□

10.4 Heuristic enhancements

In this section, we first describe how to construct **surrogate quadratic models in adaptively determined subspaces** reducing the influence of noise. Next, we describe how step sizes are updated in an improved version of **ILS-basic**. Next, an improved version of **DS-basic** performs an improved version of **ILS-basic** with four different directions to save some best points and its inexact function values. Then if a sufficient gain cannot be found in several repeated calls to an improved version of **ILS-basic**, step sizes are **restricted** and **recomputed** in a heuristic way. Finally, we introduce an improved version of **VSBBON-basic** for unconstrained noisy black box optimization problems making repeated calls to an improved version of **DS-basic**.

10.4.1 Random approximate coordinate directions

As discussed earlier in the introduction, using random directions is preferable to using deterministic ones (see [9]). On the other hand, it is well known that coordinate directions are useful to estimate the gradient since at least one of these directions has a good angle with the gradient (see [53]). Inspired by these, we construct an effective version of scaled random directions, called **random approximate coordinate direction**, inheriting advantages of both scaled random and coordinate directions. The random coordinate direction p multiplies a standard random direction by a scaling vector, one of whose components is $1/\|p\|$ and whose randomly chosen remaining components are $\gamma^{\mathrm{rd}}/\|p\|$, where $0 < \gamma^{\mathrm{rd}} < 1$ is a tiny tuning parameter.

10.4.2 Subspace information

Points X_i with best inexact function values are saved as columns of a matrix X, their inexact function values $\widetilde{f}(X_i)$ in the vector F, and their step sizes α_i in the vector Y, by an improved line search algorithm discussed later in Subsection 10.4.7. **adjustX** adjusts the matrix X with value `NaN` or $\pm\infty$ by replacing a huge positive tuning parameter $\gamma^X > 0$.

We refer to points saved in X as **sample points** whose maximum number is defined by $m^{\max} := \min\left\{\overline{m}, \frac{1}{2}n(n+3)\right\}$, where $\overline{m}$ is a tuning parameter. The **number of sample points** is denoted by $m \in [2, m^{\max}]$ and the **subspace size** m^{o} is defined as the largest integer number satisfying $\frac{1}{2}m^{\mathrm{o}}(m^{\mathrm{o}}+3) \leq m$.

If m exceeds $m^{\max}$, **updateXFY** updates X, F, Y by replacing the worst point, its inexact function value, and its step size by the current best point, its inexact function value, and its step size, respectively. Otherwise, it appends the current best point to X, its inexact function value to F, and its step size to Y.

10.4.3 Random subspace directions

In [117], it has been shown by extensive numerical results that after a derivative-free line search algorithm with coordinate directions is used, the use of random subspace directions by such an algorithm is very useful. Inspired by this, after random coordinate directions are used by an improved version of **ILS-basic**, random subspace directions are used by such an algorithm in the hope of reducing the influence of noise.

We write $A_{II} := (A_{ij})_{i\in I, j\in I}$ for the submatrix of A with row and column indices from I by A_{II}, $A_{:k}$ for the kth column of a matrix A, and b for the index of the best point.

Random subspace directions are constructed based on the information of sample points with good function values. As in [117], we generate a $(m-1)\times 1$ standard random vector α^{rs} and then scale it by $\alpha^{\mathrm{rs}} := \alpha^{\mathrm{rs}}/\|\alpha^{\mathrm{rs}}\|$. Then we compute the **random subspace direction** by

$$p := \sum_{i=1, i\neq b}^{m} \alpha_i^{\mathrm{rs}}(X_{:i} - X_{:b}).$$

10.4.4 Reduced quadratic models

It is well-known [103] that model-based algorithms need at least

$$N := n + \frac{1}{2}n(n+1) = \frac{1}{2}n(n+3)$$

sample points and $\mathcal{O}(N^3)$ operations for the estimation of the gradient vector and Hessian matrix of an objective quadratic model. For medium or large scale problems, this is prohibitively expensive. To overcome this problem, we construct quadratic models in adaptively determined subspaces called **reduced quadratic models**, one of which is a fully quadratic model when $m = N$.

For all $i = 1, \cdots, m$, let x_i and $\widetilde{f}_i := \widetilde{f}(x_i)$ be the sample points and their inexact function values, stored in X and F, respectively, and let $\overline{s}_i := x_i - X_{:b}$. Before defining model errors, we need to know whether the number of sample points m is allowable to construct a full or reduced quadratic model or not. Hence, all subspace sizes allowed to construct quadratic models are computed by **sizeSample**. The subspace size defined in Subsection 10.4.2 is computed by

$$m^{\mathrm{o}} := \Big\lfloor \frac{1}{2}(-3 + \sqrt{9 + 8m}) \Big\rfloor. \tag{10.23}$$

It is clear that if $m < N$ a fully quadratic model cannot be constructed; instead, reduced quadratic models are constructed. If the dimension is greater than $m^{\max}$, the $n \times m^{\max}$ matrix X and the $n \times 1$ vector x^m are reduced to the $m^{\max} \times m^{\mathrm{o}}$ matrix X^{o} and the $m^{\mathrm{o}} \times 1$ vector x^{o}, respectively. In other words, the restriction on entries of X and x^m is done by picking a random subset with the size m^{o}.

Some components of each best point stored in X may be ignored. To remedy this shortcoming, reduced quadratic models are constructed several times before m exceeds $m^{\max}$, each of which chooses X^{o} (x^{o}) by taking a random subset of entries of X (x^{best}).

We write $\widetilde{g}^{\mathrm{o}}$ and $\widetilde{B}^{\mathrm{o}}$ for the estimated gradient vector and symmetric Hessian matrix in a subspace, respectively. Let $M := \frac{1}{2}m^{\mathrm{o}}(m^{\mathrm{o}} + 3)$ and $K := \min(2M, m - 1)$. To assess the inexact $\widetilde{g}^{\mathrm{o}}$ and $\widetilde{B}^{\mathrm{o}}$, we define the model errors by

$$\varepsilon_i := \frac{\widetilde{f}_i - F_b - (\widetilde{g}^{\mathrm{o}})^T \overline{s}_i - \frac{1}{2}\overline{s}_i^T \widetilde{B}^{\mathrm{o}} \overline{s}_i}{\mathtt{sc}_i} \quad \text{for all } i = 1, \cdots, K, \tag{10.24}$$

where $\mathtt{sc}$ is a suitable scaling vector. As in [58], if numerical methods preserve the affine invariant, they generally can perform much better. The following choice ensures the affine invariance of the fitting procedure.

The nominator of (10.24) is $\mathcal{O}(\|\overline{s}_i\|^3)$ if $m = N$; otherwise it is $\mathcal{O}(\|\overline{s}_i\|^2)$. To get ε_i of a uniform magnitude, we choose $\mathtt{sc}$ as follows:
(1) We form the matrix $S := (\,\overline{s}_1, \cdots, \overline{s}_K\,)^T$, where $\overline{s}_i := x_i^{\mathrm{o}} - X_{:b}^{\mathrm{o}}$ for all $i = 1, \cdots, K$.
(2) We compute the matrix

$$H^{\mathrm{o}} := \Big(\sum_l \overline{s}_l \overline{s}_l^T \Big)^{-1} \tag{10.25}$$

by constructing a reduced QR factorization $S = QR$, where $Q \in \mathbb{R}^{K \times m^{\mathrm{o}}}$ is an orthogonal matrix and $R \in \mathbb{R}^{m^{\mathrm{o}} \times m^{\mathrm{o}}}$ is a square upper triangular matrix.
(3) We compute the scaling vector $\mathtt{sc}$

$$\mathtt{sc}_i := (\overline{s}_i^T H^{\mathrm{o}} \overline{s}_i)^{e/2} \quad \text{for } i = 1, \cdots, K, \tag{10.26}$$

using $\overline{s}_i^T H^{\mathrm{o}} \overline{s}_i = \|R^{-T} \overline{s}_i\|^2$ for $i = 1, \cdots, K$. In (10.26), $e = 3$ if $m = N$ holds (full quadratic model) and $e = 2$ otherwise (reduced quadratic model).

Both H^{o} and $\mathtt{sc}$ are computed by **scaleModel** in the same way as **SNOBFIT** [103] by making a reduced QR factorization but the difference that they are made in adaptively determined subspaces.

We have $\varepsilon = Ay - a$, where, for $i = 1, \cdots, K$,

$$a_i := \frac{F_b - \widetilde{f}_i}{\mathtt{sc}_i} \quad \text{and} \quad A_{ij} := \begin{cases} \dfrac{\overline{s}_i^j}{\mathtt{sc}_i} & \text{if } j \in \{1, \cdots, m^{\mathrm{o}}\}, \\ \dfrac{(\overline{s}_i^{j-m^{\mathrm{o}}})^2}{2\mathtt{sc}_i} & \text{if } j \in \{m^{\mathrm{o}}+1, \cdots, 2m^{\mathrm{o}}\}, \\ \dfrac{\overline{s}_i^{j'}\overline{s}_i^{j''}}{\mathtt{sc}_i} & \text{if } j \in \{2m^{\mathrm{o}}+1, \cdots, M\}. \end{cases} \tag{10.27}$$

Here j' and j'' are the remainders of the division of $j-2m^{\mathrm{o}}$ and $j-2m^{\mathrm{o}}+1$ by m^{o}, respectively, and $\overline{s}_i^j$ is the jth component of the vector $\overline{s}_i$. To find the entries of inexact $\widetilde{g}^{\mathrm{o}}$ and $\widetilde{B}^{\mathrm{o}}$ we solve the linear least squares problem

$$\min_{y \in \mathbb{R}^M} \|Ay - a\|_2^2. \tag{10.28}$$

In the finite precision IEEE arithmetic, any of a, $\mathtt{sc}$, y, and hence $\widetilde{g}^{\mathrm{o}}$, $\widetilde{B}^{\mathrm{o}}$ may have entries with value `NaN` or $\pm\infty$. **adjustVec** replaces components of vectors a, $\mathtt{sc}$, and y with value `NaN` or $\pm\infty$ by a huge positive tuning parameter $\gamma^v > 0$.

getMultiplier constructs the vector a and the matrix A by (10.27), adjusts a by calling **adjustVec**, and finds all multipliers by solving (10.28). **getGg** adjusts y by calling **adjustVec** and defines $\widetilde{g}^{\mathrm{o}}$ through the first m^{o} components of y, the diagonal entries of $\widetilde{B}^{\mathrm{o}}$ through the next m^{o} components of y, and off-diagonal entries of $\widetilde{B}^{\mathrm{o}}$ symmetrically through the remaining entries of y.

In summary, reduced quadratic models are constructed by **fit**. Two advantages of such models are to use limited sample points and to be constructed several times by increasing the subspace size from 2 up to $m^{\max}$. To get a robust model, **fit** uses **adjustX** to adjust X and **adjustVec** to adjust the vector y whenever they are contaminated by `NaN` or $\pm\infty$. Second, it computes $\mathtt{sc}$ by **scaleModel** and adjusts it by **adjustVec**. Third, it computes y by **getMultiplier** and adjusts it by **adjustVec**. Finally, it estimates $\widetilde{g}^{\mathrm{o}}$ and $\widetilde{B}^{\mathrm{o}}$ by **getGg**.

10.4.5 Perturbed random directions

After an improved version of **ILS-basic** is used with random subspace directions, possibly sometimes without any sufficient gain in the presence of noise, the use of other good directions is needed to hopefully find sufficient gains. In this case, **perturbed random directions** are used if $\widetilde{B}^{\mathrm{o}}$ is not computable, meaning that at least one entry of $\widetilde{B}^{\mathrm{o}}$ is contaminated by NaN or $\pm\infty$. Otherwise, reduced quadratic models are constructed and improved trust region directions are computed, as discussed later in Subsections 10.4.4 and 10.4.6, respectively.

A perturbed random direction $\overline{p}$ is a perturbation of a standard random direction p^{o} by the approximated direction $-\widetilde{g}^{\mathrm{o}}$. Both $\widetilde{g}^{\mathrm{o}}$ and p^{o} are in a subspace which is a random subset $\mathcal{J}$ of $\{1, \cdots, n\}$ with the size m^{o}. For all $i \notin \mathcal{J}$, $\overline{p}_i = 0$. To be numerically appropriate, both of them are scaled by the heuristic step size $\alpha^{\mathrm{o}} := (1 + \kappa(\widetilde{g}^{\mathrm{o}})^T p^{\mathrm{o}})/\|\widetilde{g}^{\mathrm{o}}\|^2$ and the decreasing sequence $\kappa := 1/(1+\mathtt{nf})^{\gamma^\kappa}$ with the tuning parameter $0 < \gamma^\kappa < 1$ and the number of function evaluations $\mathtt{nf}$, respectively. It can be easily shown that

$$\overline{p}^T \widetilde{g} = (p^{\mathrm{o}})^T \widetilde{g}^{\mathrm{o}} = (\kappa p^{\mathrm{o}} - \alpha^{\mathrm{o}} \widetilde{g}^{\mathrm{o}})^T \widetilde{g}^{\mathrm{o}} = -1 < 0;$$

hence perturbed random directions are descent ones.

10.4.6 An improved trust region direction

The goal of trust region methods is to restrict steps inside a trust region to increase the accuracy of surrogate models. Hence, trust region directions can be very useful, even in the presence of noise.

We now solve the trust region subproblem in a subspace

$$\begin{aligned} \min \quad & \zeta^T \widetilde{g}^{\mathrm{o}} + \frac{1}{2}\zeta^T \widetilde{B}^{\mathrm{o}} \zeta \\ \text{s.t.} \quad & \|\zeta - x^{\mathrm{o}}\|_\infty \leq d, \end{aligned}$$

whose solution is denoted by ζ^{best}. Here d is called the **trust region radius** which will be discussed later in Algorithm 10.4.2. The trust region direction is $p_{\mathrm{tr}}^{\mathrm{o}} := \zeta^{\mathrm{best}} - x^{\mathrm{o}}$ in a subspace which is a random subset of $\{1, \cdots, n\}$ with the size m^{o}. The idea is to construct the **improved trust region direction** by scaling $p_{\mathrm{tr}}^{\mathrm{o}}$ with the positive tuning parameter γ^p and perturbing it by $p^{\mathrm{mean}} := x^{\mathrm{mean}} - X_b$, where x^{mean} is the mean of X. This perturbation converts $p_{\mathrm{tr}}^{\mathrm{o}}$ to a full subspace direction, enriched by p^{mean}.

10.4.7 An improved version of ILS-basic

The efficiency of line search methods depends on how their step sizes are updated. The line search algorithms discussed in [126] and [117] find their step sizes in a way that it seems not to be effective, especially for large scale problems, since step size updates are independent; only depending on the corresponding step size generated by the previous executions. To overcome this shortcoming, we construct an improved version of **ILS-basic** whose step sizes are generated and updated in a new way.

• At first, **initStepSize** finds an initial step size. It initially chooses a positive tuning parameter $\delta^{\max}$ as an initial value for an extrapolation step size α and the step size δ.
• **direction** is called to generate 4 kinds of search directions: random coordinate direction, random subspace direction, improved trust region direction, perturbed random direction.
• In each iteration, after a new trial point and its function value were computed, **initInterval** saves sufficient gain – defined earlier in Subsection 10.2.1 – in `dF` and its step size in **a**.
• After a sufficient gain is found, an extrapolation step is tried:
(1) If in an extrapolation step, some points with best inexact function values are found **robustPoint** stores the inexact function values and their step sizes in $\overline{F}$ and $\overline{\mathbf{a}}$, respectively. Once an extrapolation step ends, an index of the best point, its function value, and its step size are found by

$$\mathtt{ind} := \underset{i=1,\cdots,m}{\operatorname{argmax}}(\overline{F}_i),\ \mathtt{ind} := (\mathtt{ind})_1,\ \overline{f}^{\mathrm{best}} := \overline{F}_{\mathtt{ind}},\ \text{and}\ \overline{\alpha}^{\mathrm{best}} := \overline{\mathbf{a}}_{\mathtt{ind}};$$

(2) Afterwards, **updateXFY** stores the information found by **robustPoint**;
(3) Next, **initInterval** finds an index set of the saved points with the decreasing inexact function values by $\mathtt{ind} := \{i \mid \mathtt{dF}_i < 0\}$. If `ind` is not empty, the lower bound of the interval $[\underline{\alpha}, \overline{\alpha}]$ is found by $\underline{\alpha} := \max(\mathbf{a}_{\mathtt{ind}})$; if $\underline{\alpha} = \underline{\alpha}^{\mathrm{init}}$ is given as a positive tuning parameter, then $\underline{\alpha}$ is updated

by $\underline{\alpha} := \min(\underline{\alpha}, \max(\mathbf{a}_{\mathtt{ind}}))$. Next, it finds an index set of positive gains or of the corresponding step sizes which are strictly greater than $\overline{\alpha}$ by

$$\overline{\mathtt{ind}} := \{i \mid \mathtt{dF}_i \geq 0 \text{ or } \mathbf{a}_i > \overline{\alpha}\}.$$

If $\overline{\mathtt{ind}}$ is not empty, an upper bound of the initial interval is found by $\overline{\alpha} := \min(\mathbf{a}_{\overline{\mathtt{ind}}})$; if $\overline{\alpha} = \overline{\alpha}^{\text{init}}$ is given as a positive tuning parameter, the upper bound of the initial interval is updated by $\overline{\alpha} := \max(\overline{\alpha}, \min(\mathbf{a}_{\overline{\mathtt{ind}}}))$.
(4) One of the lower or upper bound of the interval $[\underline{\alpha}, \overline{\alpha}]$ is updated by **updateInterval**. In practice, if $\overline{\alpha}^{\text{best}} > \underline{\alpha}$ holds, $\overline{\alpha} = \overline{\alpha}^{\text{best}}$; otherwise, $\underline{\alpha} = \overline{\alpha}^{\text{best}}$.
• If the interval $[\underline{\alpha}, \overline{\alpha}]$ is found, **initStepSize** finds a new step size by choosing the maximum of δ and $\alpha^{\text{bis}} := \sqrt{\underline{\alpha}\overline{\alpha}}$ (**geometric** mean of $\underline{\alpha}$ and $\overline{\alpha}$).
• If no sufficient gain is found, **reducedStepSize** is used to reduce the step size. If the initial interval has been updated, **reducedStepSize** calls **bisection** to generate α^{bis}. Then a new step size α^{new} is the projection of α^{bis} into $[\alpha^{\min}, \alpha/\gamma^e]$. Otherwise, α^{new} is the maximum of $\alpha^{\min}$ and α/γ^e. Then **updateInterval** updates the interval whenever $\overline{\alpha} = \underline{\alpha}$. In this case, if $\alpha^{\text{new}} < \overline{\alpha}$ holds, $\underline{\alpha} = \alpha^{\text{new}}$; otherwise, $\overline{\alpha} = \alpha^{\text{new}}$.
• If no sufficient gain is found, the best point, its inexact function value, and its step size are updated if the inexact function value at the current trial point is smaller than that of at the current best point. In particular, this may happen in very flat regions of the feasible domain which contain no nearby stationary point.

Let T be the number of trial steps. Then an improved version of **ILS-basic** tries to hopefully get sufficient gains while using T suitable directions so that all step sizes are chosen inside such an interval.

ILS tries to hopefully find sufficient gains along multiple directions. It takes m, X, F, Y, b, T, δ, $\overline{\alpha}$, $\underline{\alpha}$, and `nfmax` as input and returns m, X, F, Y, b, $\overline{\alpha}$, $\underline{\alpha}$ as output. It uses all tuning parameters discussed in **ILS-basic** and the following tuning parameters:
$\gamma^p > 0$ (parameter for adjusting trust region directions),
$\gamma^v > 0$ (parameter for polishing the vectors a, `sc`, y),
$\gamma^X > 0$ (parameter polishing X),
$\overline{m} \geq 2$ (limit for m).
Here $x^{\text{best}} = X_{:b}$ and $\widetilde{f}^{\text{best}} = F_b$.

10.4.1 Algorithm. (**ILS, an improved version of ILS-basic**)
(ILSb_0) Call **initStepSize** to get an initial step size, resulting in α^1.
for $t = 1, \cdots, T$ **do**
(ILSb_1) Call **direction** to generate the new direction p^t.
while true **do**
(ILSb_2) A new trial point and its inexact function value:
(1) Compute $x^{\text{trial}} := x^{\text{best}} + \alpha^t p^t$ and $\widetilde{f}^{\text{trial}} := \widetilde{f}(x^{\text{trial}})$.
(2) If `nfmax` is exceeded, **ILS** ends.
(3) If the initial interval has not been found or updated by **initInterval**, save the step size α^t and $\widetilde{f}^{\text{trial}} - \widetilde{f}^{\text{best}}$ in $\mathbf{a}$ and `dF`, respectively.
(ILSb_3) If $\widetilde{f}^{\text{trial}} < \widetilde{f}^{\text{best}} - \gamma(\alpha^t)^2$ holds, a sufficient gain is found and so call **robustPoint** to update the best point, its inexact function value, its step size, expand the step size to $\alpha^t := \gamma^e \alpha^t$.
end while
(ILSb_4) If a sufficient gain is found in (ILSb_3), call

(1) **robustPoint** and set $\widetilde{f}^{\text{best}} := \overline{f}^{\text{best}}$, $\alpha^t := \overline{\alpha}^{\text{best}}$ and $x^{\text{best}} := x^{\text{best}} + \alpha^t p^t$;
(2) **updateXFY** to store x^{best}, $\widetilde{f}^{\text{best}}$, and α in X, F, and Y, respectively;
(3) **initInterval** to find or update it if the initial interval is not found;
(4) **updateInterval** to update the interval if the initial interval is found.
otherwise, if extrapolation along $-p^t$ has not been tried already, set $p^t := -p^t$ and go to (ILSb_2),
otherwise, no sufficient gain is found along $\pm p^t$:
(1) If $\widetilde{f}^{\text{trial}} < \widetilde{f}^{\text{best}}$, set $x^{\text{best}} := x^{\text{trial}}$ and $\widetilde{f}^{\text{best}} := \widetilde{f}^{\text{trial}}$, store x^{best}, $\widetilde{f}^{\text{best}}$, and α^t by **updateXFY**.
(2) Regardless of whether $\widetilde{f}^{\text{trial}} < \widetilde{f}^{\text{best}}$ or not, call **reducedStepSize** to reduce the step size and **updateInterval** to update the interval $[\underline{\alpha}, \overline{\alpha}]$.
end for

10.4.8 An improved version of DS-basic

As was described earlier in Subsection 10.2.1, **DS-basic** alternated T^0 calls to **ILS-basic** with R scaled random directions regardless of whether or not a sufficient gain is found. In this subsection, an improved version of **DS-basic**, called **DS**, preserves this procedure with the goal of restoring and updating the m sample points and their inexact function values. Then **DS** constructs quadratic or linear model to generate improved trust region direction or perturbed random direction and alternates T^0 calls to **ILS** along these directions as long as sufficient gains are found. If there is no sufficient gain, the current interval $[\underline{\alpha}, \overline{\alpha}]$ is not possibly effective; hence it needs to be restarted and reconstructed in a heuristic way, using the information of the m sample points. It is done by **resInterval** described below inside **DS**.

DS tries to hopefully find sufficient gains by performing **ILS** along multiple directions. It has the same input and output as **ILS**. It uses the tuning parameters used in **ILS** and the following tuning parameters:
$\gamma^a > 0$ (parameter for adjusting heuristic step size),
$\gamma^{d_1} > 1$ and $\gamma^{d_2} \in (0, 1)$ (parameters for updating trust region radius),
$0 < d^{\min} < d^{\max} < \infty$ (control parameters for the trust region radius).

10.4.2 Algorithm. (**DS, an improved version of DS-basic**)
for $t = 1, \cdots, T^0$ **do**
(DS_1) Perform **ILS** using R **scaled random directions**.
(DS_2) If $m \geq 3$, perform **ILS** using **random subspace directions** until sufficient gains are found.
(DS_3) If $m \geq 2$, either reduced quadratic or linear model is constructed:
if both estimations are computable (not contaminated by either `NaN` or $\pm\infty$) **then**
(1) call **fit** to construct and minimize a reduced quadratic model, resulting in an improved trust region direction;
(2) call **genRadius** to generate the trust region radius by $d^t := \|x^{\text{mean}} - X_{:b}\|$ (x^{mean} is the mean of X) and restrict it by $d^t := \max(d^{\min}, \min(d^{\max}, \gamma^{d_1} d^t))$;
(3) call **ILS** using improved trust region directions until sufficient gains are found while using $d^t := (\gamma^{d_2} + \text{rand}) d^t$, where $\text{rand} \in (0, 1]$ is a random value.
else
perform **ILS** using perturbed random directions until sufficient gains are found.
end if

(DS_4) If a sufficient gain is not found in the current iteration, call **resInterval** to do the following:
For $i = 1, \cdots, n$
(1) compute $\texttt{dx}_i := X_{:i} - X_{:b}$,
(2) find $\mathcal{I}_i := \{j \mid \texttt{dx}_j \neq 0 \text{ and } (X_{:b})_j \neq 0\}$,
(3) compute $\beta_i^t := \min_{j \in \mathcal{I}_i} |(X_{:b})_j / \texttt{dx}_j|$.

Then generate two random values μ^1 and μ^2 satisfying $0 < \mu^1 < \mu^2 < 1$ and reconstruct the new interval $[\underline{\alpha}, \overline{\alpha}]$ as

$$[\gamma^a \mu^1 \beta_{\min}^t, \gamma^a \mu^2 \beta_{\min}^t] \text{ with } \beta_{\min}^t := \min_{i=1,\cdots,n} \beta_i^t.$$

Otherwise, the interval is not restarted.
end for

10.4.9 The implemented version of VSBBON

An improved version of **VSBBON-basic**, called **VSBBON**, using all prepared ingredients is introduced. As long as **DS** cannot find a sufficient gain, δ is updated by **updateStepSize**. Once δ is below a minimum threshold $\delta^{\min}$, **VSBBON** ends.

VSBBON solves randomized noisy black box optimization. It takes the initial point x^0 and `nfmax` as input and returns the overall best point x^{best} and its inexact function value $\widetilde{f}^{\text{best}}$ as output. It uses the tuning parameters used in **DS** and the following tuning parameters:
$0 \leq \underline{\alpha}^{\text{init}} < \overline{\alpha}^{\text{init}} \leq \infty$ (the lower and upper of an initial interval),
$0 < \delta^{\min} < 1$ (minimum threshold for δ),
$\delta^{\max} > 0$ (initial step size),
$Q > 1$ (factor for reducing the step size).

10.4.3 Algorithm. (**VSBBON, an implemented version of VSBBON-basic**)
(V_1) Compute the initial inexact function value $\widetilde{f}^0$ and set $F := \widetilde{f}^0$, $X := x^0$, $Y := 1$, $b := 1$, and $m := 1$. Next, initialize $\underline{\alpha} := \underline{\alpha}^{\text{init}}$ and $\overline{\alpha} := \overline{\alpha}^{\text{init}}$.
for $k = 1, 2, 3, \cdots$ **do**
(V_2) Call **DS** to hopefully find sufficient gains.
(V_3) If $\delta^k \leq \delta^{\min}$ or `nfmax` is reached, set $x^{\text{best}} := X_{:b}$ and $\widetilde{f}^{\text{best}} := F_b$. Then **VSBBON** ends.
(V_4) If **DS** cannot find a sufficient gain, **updateStepSize** reduces the step size δ^k by the factor of Q; otherwise, it finds α^{bis} by calling **bisection** and updates δ^k by $\delta^k := \max(\delta^k, \alpha^{\text{bis}})$ provided that $[\underline{\alpha}, \overline{\alpha}] \subseteq (0, \infty)$.
end for

Theorem 10.3.2, Proposition 10.3.3, Theorem 10.3.4, and Theorem 10.3.5 remain valid with the following changes
• In Theorem 10.3.2(i), R (number of scaled random steps) must be replaced by T (number of trial steps), and the statement (ii) of it remains valid with probability $\geq 1 - 2^{-R}$.
• In Proposition 10.3.3(i), the number of function evaluations of **VSBBON** up to iteration k is bounded by

$$1 + (2R + 5)T^0(|U^k| + |S^k|);$$

because in the worst case **ILS** with RT^0 scaled random directions and RT^0 corresponding opposite ones, T^0 random subspace directions and T^0 corresponding opposite ones, and T^0 improved trust region directions (or T^0 perturbed random directions) and T^0 corresponding opposite ones cannot find a sufficient gain; however, if there exists a sufficient gain along the opposite direction of the last improved trust region direction (or the last random perturbed direction), an extrapolation step is tried with at most two function evaluations. Then the best point is updated. The statement (ii) of it remains valid.
• In Proposition 10.3.3, Theorem 10.3.4, and Theorem 10.3.5, $2RT^0$ must be replaced by $(2R+5)T^0$.

Note that random approximate coordinate directions are numerically better than random scaled directions but Proposition 10.2.1 is valid for random scaled directions. Hence, if **VSBBON** uses random approximate coordinate directions instead of random scaled directions, it has no complexity result.

10.5 Numerical results

In this section, we report numerical experiments with the test environment constructed by Kimiaei & Neumaier [116] on the 503 unconstrained `CUTEst` test problems from the collection by Gould et al. [80], described in in the supplemental material `supplMat.pdf` with detailed tables and performance plots.

In the numerical results reported here, uniform random noise matching the assumption (BBO_3)is used. The function values are computed by $\widetilde{f} = f + \widetilde{\omega}$, where f is the true function value and $\widetilde{\omega} = (2 * \text{rand} - 1)\omega$. Here rand stands for the uniformly distributed random number. However, further results for relative uniform noise and relative/absolute Gaussian noise reported in `supplMat.pdf` show that **VSBBON** works well with any kind of noise that is not too large, though our complexity results are only valid under the assumption (A3).

VSBBON1 is the model-based version of **VSBBON** and **VSBBON2** is the model-free version of **VSBBON**. **VSBBON1** and **VSBBON2** are compared with **VSBBO** by Kimiaei & Neumaier [117], **SDBOX** by Lucidi & Sciandrone [126], **NMSMAX** by Higham [99], **DSPFD** by Gratton et al. [84], **BFO** by Porcelli & Toint [147], **MCS** by Huyer & Neumaier [102], **BCDFO** by Gratton et al. [85], **UOBYQA** by Powell [149], and **FMINUNC** by the Matlab Optimization Toolbox for small, medium, and large scale problems. Unfortunately, software for **FDLM** by Berahas et al. [14], **STRRS** by Chen, and **Grid** by Elster & Neumaier [65] was not available to us.

As discussed in Subsection 5.6, we consider a problem solved by the solver *so* if $q^{so} \leq \varepsilon$. Otherwise, the problem is unsolved since either `nfmax` or `secmax` is exceeded. Here ε depends on the dimension and the noise level

$$\varepsilon := \begin{cases} 10^{-3} & \text{if } \omega \in \{10^{-4}, 10^{-3}\} \text{ and } n \in [1, 30], \\ 10^{-2} & \text{if } \omega \in \{0.1, 0.9\} \text{ and } n \in [1, 30], \\ 10^{-3} & \text{if } \omega = 10^{-4} \text{ and } n \in [31, 300], \\ 0.05 & \text{if } \omega \in \{0.1, 0.01, 0.001\} \text{ and } n \in [31, 300], \\ 0.05 & \text{if } \omega \in \{10^{-5}, 10^{-4}, 10^{-3}\} \text{ and } n \in [301, 3000]; \end{cases}$$

because by increasing the noise level and the dimension the difficulty of problems is increased

extremely. Hence ε is chosen to slightly be large for problems in medium and high dimensions in comparison with problems in low dimensions.

The efficiency of the solver *so* with respect to a cost measure c^{so} for solver *so* was discussed in Subsection 5.4.

10.5.1 Small scale: $1 \leq n \leq 30$

Subfigures (a), (c) and (e) of Figure 10.1 plot the number of solved problems versus the noise level ω, while Subfigures (b), (d) and (f) of this figure plot the `nf` efficiency versus the noise level ω, respectively.

In Figure 10.1,
• Subfigures (a) and (b) are the two different comparisons among solvers using the line search methods (**VSBBON1**, **VSBBON2**, **VSBBO**, **FMINUNC**, and **SDBOX**).
• Subfigures (c) and (d) are the two different comparisons among **VSBBON1** and **VSBBON2** with four famous direct search methods (**NMSMAX**, **DSPFD**, **BFO**, and **MCS**).
• Subfigures (e) and (f) are the two different comparisons among **VSBBON1** and **VSBBON2** with two competitive solvers using the full quadratic model (**BCDFO** and **UOBYQA**)

In summary, we conclude from Figure 10.1 that, in the absolute uniform noise case,
• **VSBBON1** and **VSBBON2** are more robust than **VSBBO**, **FMINUNC**, and **SDBOX** in terms of the number of solved problems and the `nf` efficiency, shown in Subfigures (a) and (b), respectively.
• **VSBBON1** and **VSBBON2** are more robust than **NMSMAX**, **DSPFD**, **BFO**, and **MCS** in terms of the number of solved problems shown in Subfigure (c). **NMSMAX** is the first best solver in terms of the `nf` efficiency while **VSBBON1** and **VSBBON2** are the second and third best solvers, shown in Subfigure (d). Moreover, **VSBBON1** is the best solver in the high level of noise $\omega = 0.9$.
• **VSBBON1** and **VSBBON2** are more robust than **BCDFO** and **UOBYQA** in terms of the number of solved problems shown in Subfigure (e), while **UOBYQA** and **BCDFO** are the first and second best solvers in terms of the `nf` efficiency, respectively, except in the presence of the high noise level $\omega = 0.9$; in this case, **VSBBON1** outperforms others.

From the result, we conclude that **VSBBON1** is more robust than others, in the absolute uniform noise case, for small scale problems $n \in [1, 30]$. Moreover, the model-based **VSBBON1** is preferable to the model-free **VSBBON2**.

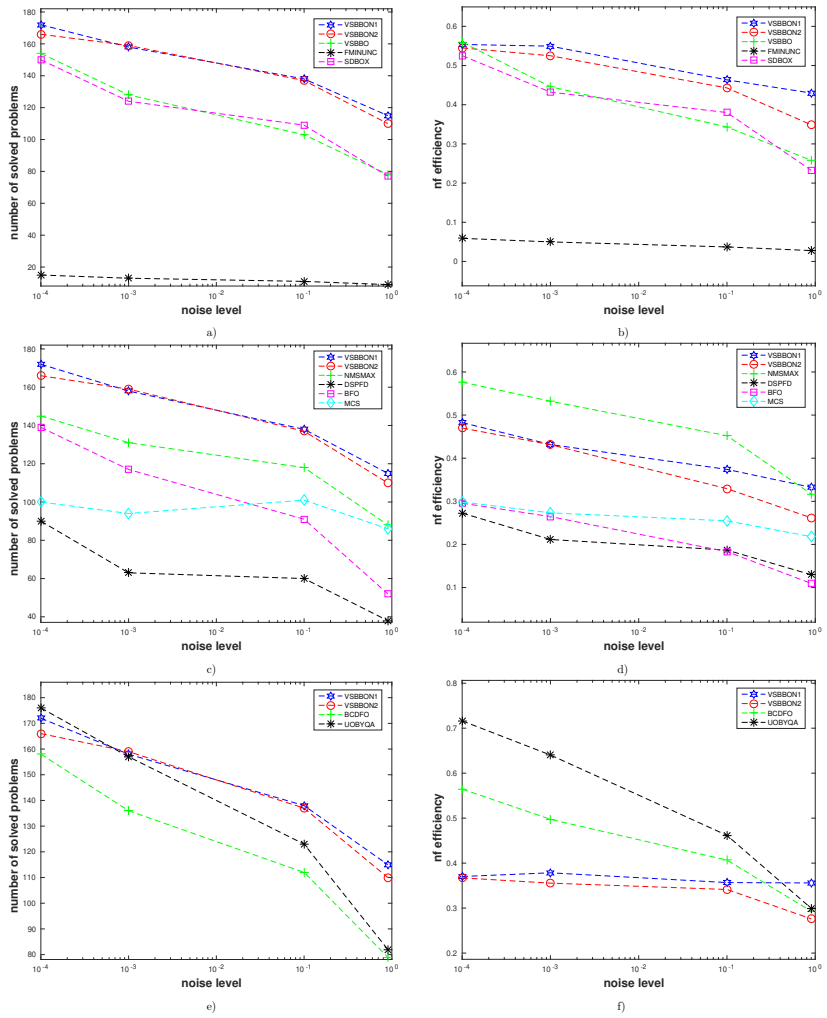


Figure 10.1: For the noise levels $\omega \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.9\}$. Subfigures (a), (c) and (e) plot the number of solved problems versus the noise level ω, while Subfigures (b), (d) and (f) plot the `nf` efficiency versus the noise level ω, respectively.

10.5.2 Medium scale: $31 \leq n \leq 300$

Subfigures (a) and (c) of Figure 10.2 plot the number of solved problems versus the noise level ω, while Subfigures (b) and (d) of this figure plot the `nf` efficiency versus the noise level ω, respectively.

In Figure 10.2,
• Subfigures (a) and (b) are the two different comparisons among solvers using the line search methods (**VSBBON1**, **VSBBON2**, **VSBBO**, and **SDBOX**).
• Subfigures (c) and (d) are the two different comparisons among **VSBBON1** and **VSBBON2** with some famous direct search methods (**NMSMAX** and **BFO**).

In summary, we conclude from Figure 10.2 that
• **VSBBON1** and **VSBBON2** are more robust than **VSBBO** and **SDBOX** in terms of the number of solved problems and the `nf` efficiency, shown in Subfigures (a) and (b), respectively.
• **VSBBON1** and **VSBBON2** are more robust than **NMSMAX** and **BFO** in terms of the number of solved problems and the `nf` efficiency, shown in Subfigures (c) and (d).

From the result, we conclude that **VSBBON1** and **VSBBON2** are more robust than others for medium scale problems $n \in [31, 300]$.

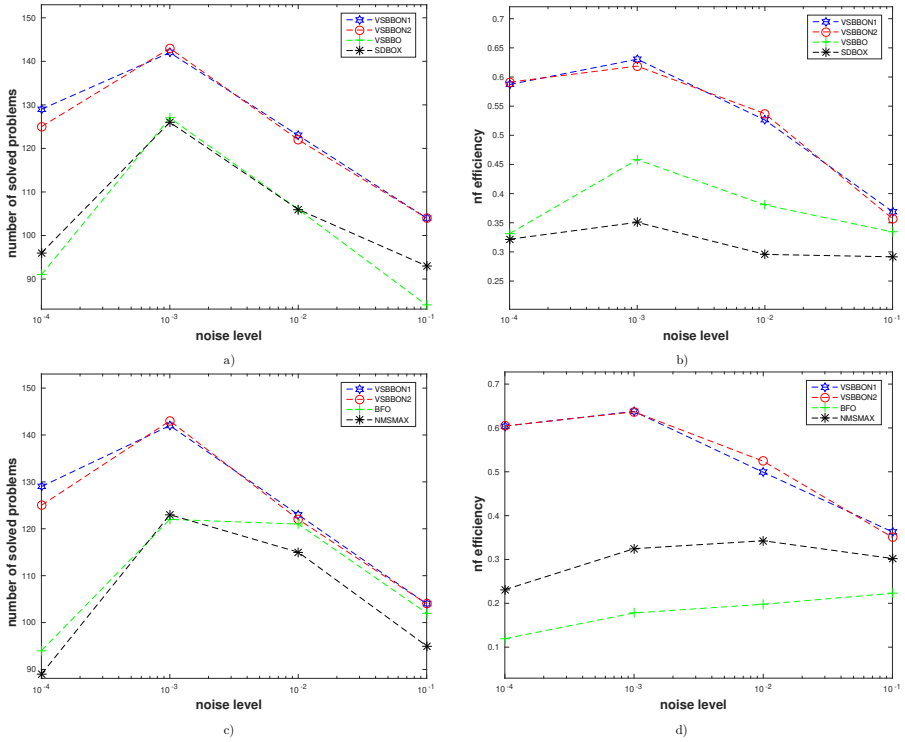


Figure 10.2: For the noise levels $\omega \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Subfigures (a) and (c) plot the number of solved problems versus the noise level ω, while Subfigures (b) and (d) plot the `nf` efficiency versus the noise level ω, respectively.

10.5.3 Large scale: $301 \leq n \leq 3000$

Subfigure (a) of Figure 10.3 plots the number of solved problems versus the noise level ω in the absolute uniform noise case, while Subfigure (b) of this figure plots the `nf` efficiency versus the noise level ω.

In Figure 10.3, Subfigures (a) and (b) are comparisons among solvers using the line search methods (**VSBBON1**, **VSBBON2**, and **VSBBO**).

From the result, we conclude from Figure 10.3 that **VSBBON1** and **VSBBON2** are more robust than **VSBBO** in terms of the number of solved problems and the `nf` efficiency, except for $\omega = 10^{-4}$; in this case **VSBBO** has the best performance in terms of the number of solved problems (Subfigures (a) and (b)), respectively. Moreover, the model-based version of **VSBBON1** is preferable to the model-free version of **VSBBON2**.

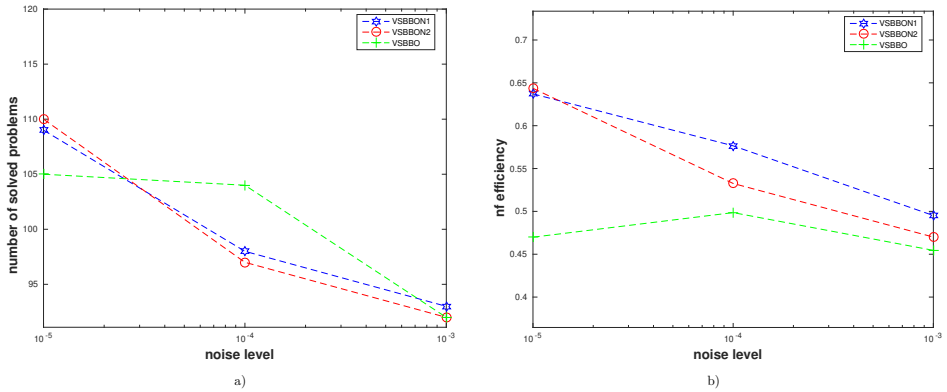


Figure 10.3: For the noise levels $\omega \in \{10^{-5}, 10^{-4}, 10^{-3}\}$. Subfigure (a) plots the number of solved problems versus the noise level ω, while Subfigure (b) plots the `nf` efficiency versus the noise level ω.

10.6 Additional material for VSBBON

This section discusses additional material for **VSBBON**.

10.6.1 Default tuning parameters for VSBBON

VSBBON was written in Matlab; the source code is available at

`http://www.mat.univie.ac.at/~kimiaei/software/VSBBON`.

The values of tuning parameters for both cases, noiseless and noisy, are chosen as

$$\begin{array}{|l|}\hline \overline{m}=230;\ m^{\max}=\min(0.5n(n+3),\overline{m});\ T^0=n;\ R=n;\ \delta^{\min}=0;\\ \delta^{\max}=1;\ \gamma^X=100;\ \gamma^r=10^{-30};\ \gamma^{d_1}=2;\ Q=1.5;\ d^{\min}=10^{-4};\\ \gamma^v=100;\ \gamma=10^{-6};\ \gamma^e=3;\ d^{\max}=10^3;\ \underline{\alpha}^{\text{init}}=0.01;\ \overline{\alpha}^{\text{init}}=0.99;\\ \gamma^k=0.85;\ \texttt{model}=1;\ \gamma^{d_2}=0.5;\ \gamma^p=0.25;\\ \hline\end{array}$$

The values of other tuning parameters are

$$\gamma^a:=\begin{cases}10^{-5} & \text{if } \omega=0,\\ 2.5 & \text{otherwise}\end{cases}$$

and

$$\alpha^{\min}:=\begin{cases}10^{-30} & \text{if } \omega=0,\\ 10^{-3} & \text{otherwise.}\end{cases}$$

There has almost been no effort to find the optimal tuning parameters. Probably, the efficiency of **VSBBON** will be increased by properly optimizing them. This is still a work in progress [116]. However, we identify what the best value is for the tuning parameter `model`. To do so, **VSBBON** with `model` $=1$ and `model` $=0$ are denoted by **VSBBON1** (the model-based) and **VSBBON2** (the model-free), respectively.

The trust region subproblem is solved by `minq8` [104] whose tuning parameters are chosen

$$\texttt{minqmax}=10000,\quad \texttt{minqeps}=10^{-8};$$

available at

`https://www.mat.univie.ac.at/~neum/software/minq/`

11 A new method for least squares problems

This section discusses a new method for least squares problems, called **LMLS**. This is a joint work with Arnold Neumaier (Kimiaei & Neumaier [115]). Main features of **LMLS** are a new non-monotone technique, a new adaptive radius strategy, a new Broyden-like algorithm based on the previous good points, and a heuristic estimation for the Jacobian matrix in an adaptive subspace. Our numerical results show that **LMLS** is robust and efficient, especially in comparison with solvers using traditional limited memory and standard quasi Newton approximations.

11.1 Overview of the new method

We suggest in Section 11.2 a new trust region-based limited memory algorithm for unconstrained black box least squares, called **LMLS**. This algorithm uses
• a non-monotone ratio and an adaptive radius formula to quickly reach the minimizer when the valley is narrow;
• a Broyden-like algorithm to get a decrease in the function value when the trust region radius is so small and iteration is unsuccessful;
• a finite difference approximation in an adaptive subspace to estimate the Jacobian matrix;
• either a Gauss-Newton in an adaptive subspace or a dogleg algorithm in an adaptive subspace to solve the trust region subproblems.

Numerical results for small, medium, and large scale problems are given in Section 11.3 showing the fact that the new method is suitable for large scale problems and is more robust and efficient than solvers using limited memory and standard quasi Newton approximations.

11.2 The trust region method

In this section, we construct an improved trust region algorithm handling for problems in high dimensions:
• In Subsection 11.2.1 a new subspace Gauss-Newton direction is introduced.
• In Subsection 8.7.1 a non-monotone term and an adaptive technique are constructed to quickly reach the minimizer in the presence of a narrow valley.
• In Subsection 11.2.3 a subspace dogleg algorithm is discussed.
• In Subsection 11.2.4 a Broyden-like technique is suggested based on the old best points.
• In Subsection 11.2.5 our algorithm using new enhancements is introduced.

We write $J(x)$ for the Jacobian matrix of the residual vector E at x. Then the gradient vector is $g(x) := \nabla f(x) := J(x)^T E(x)$ and the Hessian matrix is

$$G(x) := J(x)^T J(x) + \nabla^2 E(x)^T E(x)$$

If the residual vector $E(x)$ is small, the second term in $G(x)$ is small. Hence, we approximate $G(x)$ by the Gauss-Newton Hessian matrix $J(x)^T J(x)$. We define the quadratic surrogate objective function

$$\mathcal{Q}(p) := \frac{1}{2}\|E + Jp\|^2 := f + p^T g + \frac{1}{2}(Jp)^T Jp, \tag{11.1}$$

where $f := f(x)$, $E := E(x)$, $J := J(x)$, $g := g(x) := J^T E$. We denote by $A_{:k}$ the kth column of a matrix A.

A trust region method finds a minimizer of the constrained problem

$$\begin{aligned} &\min\ \mathcal{Q}(p) \\ &\text{s.t}\ \ p \in \mathbb{R}^n \ \text{ and } \ \|p\| \le \Delta, \end{aligned} \tag{11.2}$$

whose constraint restricts feasible points by the **trust region radius** $\Delta > 0$. This problem is called the **trust region subproblem**. Given a solution p of (11.2), we define the **actual reduction** in the objective function by

$$df := f - f(x + p) \tag{11.3}$$

and the **predicted reduction** in the model function by

$$dq := \mathcal{Q}(0) - \mathcal{Q}(p). \tag{11.4}$$

What constitutes an agreement between the actual and predicted reduction around the current iterate x must be measured by the **monotone trust region ratio**

$$\rho := \frac{df}{dq}. \tag{11.5}$$

If such an agreement is good according to a heuristic formula discussed in Subsection 8.7.1, the iteration is called **successful**, $x + p$ is accepted as a new point, and the radius is expanded; otherwise, the iteration is called **unsuccessful** and so the radius is reduced.

11.2.1 A new subspace Gauss-Newton method

In this subsection, we have two goals: estimating the Jacobian matrix in an adaptive subspace and constructing a new subspace Gauss-Newton direction.

Let m^{sn} be the subspace size. The Jacobian matrix in an adaptive subspace is estimated by a new **subspace random finite difference** called **SRFD** using the following steps:
(1) At first, an initial subspace is a random subset of $\{1, \cdots, n\}$ consisting of m^{sn} members and its complementary is $S^c := \{1, \cdots, n\} \setminus S$.
(2) Next, if the complementary of old subspace S^c_{old} is not empty, a new index set $\mathcal{I}$ needs to be identified before a new subspace is determined. In this case, if $\mathcal{I}$ consists of at least m^{sn} members, $\mathcal{I}$ is a random subset of $\{1, \cdots, m^{\text{sn}}\}$ with the $|S^c_{\text{old}}|$ members; otherwise, it is a permutation

of $\{1, \cdots, |S^c_{\rm old}|\}$. Then, a new subspace is determined by $S := S^c_{\rm old}(\mathcal{I})$ and its complementary is found by $S^c := S^c_{\rm old} \setminus P$. But if $S^c_{\rm old}$ is empty, a new subspace and its complementary are restarted and chosen in the same way as the initial subspace and its complementary, respectively.
(3) For any $i \in S$,
• the step size is computed by

$$h_i := \begin{cases} \gamma^s & \text{if } x_i = 0, \\ \gamma_s(\operatorname{sign} x_i) \max\left\{|x_i|, \dfrac{\|x\|_1}{n}\right\} & \text{otherwise,} \end{cases}$$

where $0 < \gamma^s < 1$ is a tiny factor and $\operatorname{sign} x_i$ identifies the sign of x_i, taking one of values -1 (if $x_i < 0$), 0 (if $x_i = 0$), and 1 (if $x_i > 0$).
• the random approximation coordinate direction p discussed in [114] is used with the difference that its ith component is updated by $p_i = p_i + h_i$.
• the new trial residual $E(x + \alpha p)$ and the new column $(E(x + \alpha p) - E)/h_i$ of the Jacobian matrix are computed.

It is well known that standard quasi Newton methods are more robust than limited memory quasi Newton ones, but they cannot handle for problems in high dimensions; for standard quasi Newton methods, see [105, 106, 139, 153], and for limited memory quasi Newton methods, see [124, 132, 139]. On the other hand, finite difference methods are more efficient than standard quasi Newton ones. Hence, if used in an adaptive subspace, they can be more efficient than limited memory quasi Newton methods for small up to large scale problems.

Using S and S^c generated and updated by **SRFD**, we construct a new **subspace Gauss-Newton direction** by

$$(p_{\rm sn})_i := 0 \quad \text{for } i \in S^c \text{ and } \quad (p_{\rm sn})_S := -(J_{:S}^T J_{:S})^{-1} J_{:S}^T E. \tag{11.6}$$

11.2.2 New non-monotone and adaptive strategies

In this subsection, a new non-monotone term – stronger than the objective function f – is constructed and a new adaptive radius formula to update Δ is derived from it. They help **LMLS** in finite precision arithmetic to quickly reach the minimizer in the cases where the valley is deep with a small creek at the bottom and steep sides.

Our non-monotone term is updated not only for successful iterations but also for **unsuccessful** iterations that may happen before a successful iteration is found. This choice is based on an estimated increase in f defined below which is updated according to whether a decrease in f is found or not. It helps us to generate a somewhat strong non-monotone term when a decrease in f is not found and a somewhat weak non-monotone term otherwise. Somewhat strong non-monotone terms increase the chance of finding a point with better function value or at least a point with a little progress in the function value instead of solving trust region subproblems with high computational costs.

We denote by X a list of best points and by F a list of corresponding function values. Let $m^{\rm rs}$ be the maximum number of good points saved in X. In order to update X and F, we use **updateXF**. Here we describe how to work it. If $m^{\rm rs}$ is not exceeded, points with good function values are saved in X and their function values in F. Otherwise, the worst point and its function value are found and replaced by the best point and its function value, respectively.

Let $\gamma^t \in (0,1)$, $\underline{\gamma} \in (0,1)$, $\gamma^{\text{init}} > 0$, and $\overline{\gamma} > 1$ be the tuning parameters and let

$$f^k_{\max} := \max_{i=1:m^{\text{rs}}} \{F^k_i\} \quad \text{for all } k. \tag{11.7}$$

Before a new non-monotone term is constructed, an estimated increase in f needs to be estimated by

$$\delta^k_f := \begin{cases} \gamma^{\text{init}}|f^0| & \text{if } k = 0,\ f^0 \in (0,\infty), \\ 1 & \text{if } k = 0,\ f^0 \in \{-\infty, 0, \infty\}, \\ \dfrac{1}{\overline{\gamma}}(f^{k-1} - f(x^{k-1} + p^{k-1})) & \text{if } k \geq 1,\ f(x^{k-1} + p^{k-1}) < f^{k-1}, \\ \max(\overline{\gamma}\delta^{k-1}_f, \underline{\gamma}(|f(x^{k-1} + p^{k-1})| + |f^k_{\max}|)) & \text{if } k \geq 1,\ f(x^{k-1} + p^{k-1}) \geq f^{k-1}. \end{cases} \tag{11.8}$$

Accordingly, the new non-monotone formula is defined by

$$f^k_{\text{nm}} := \begin{cases} f^0 & \text{if } k = 0, \\ f^k + \delta^k_f & \text{if } k \geq 1 \end{cases} \tag{11.9}$$

and the new adaptive radius is constructed by

$$\Delta^k_{\text{nm}} := \lambda^k \sqrt{f^k_{\text{nm}}}, \tag{11.10}$$

where $\Delta^0_{\text{nm}} > 0$ is a tuning parameter and λ^k is updated according to

$$\lambda^k := \begin{cases} \sigma_1 \lambda^{k-1}, & \text{if } \rho^{k-1}_{\text{nm}} < \gamma^t, \\ \min(\overline{\lambda}, \max(\sigma_2 \lambda^{k-1}, \underline{\lambda})), & \text{otherwise.} \end{cases} \tag{11.11}$$

Here $\lambda^0 > 0$, $0 < \sigma_1 < 1 < \sigma_2$, and $\overline{\lambda} > \underline{\lambda} > 0$ are the tuning parameters and the new non-monotone trust region ratio is defined by

$$\rho^{k-1}_{\text{nm}} := \frac{f^{k-1}_{\text{nm}} - f(x^{k-1} + p^{k-1})}{\widetilde{\mathcal{Q}}^{k-1}(0) - \widetilde{\mathcal{Q}}^{k-1}(p^{k-1})}, \tag{11.12}$$

where p^{k-1} is a solution of the following trust region subproblem in the subspace S by **SRFD**

$$\begin{aligned} &\min\ \widetilde{\mathcal{Q}}^{k-1}(p) := \frac{1}{2}\|E^{k-1} + J^{k-1}_{:S} p\|^2 := f^{k-1} + p^T g^{k-1}_S + \frac{1}{2}(J^{k-1}_{:S} p)^T J^{k-1}_{:S} p \\ &\text{s.t}\ \ p \in \mathbb{R}^{m^{\text{sn}}} \ \text{ and } \ \|p\| \leq \Delta^{k-1}_{\text{nm}} \end{aligned} \tag{11.13}$$

with $f^{k-1} := f(x^{k-1})$, $E^{k-1} := E(x^{k-1})$, $J^{k-1}_{:S} := J_{:S}(x^{k-1})$, and $g^{k-1}_S := (J^{k-1}_{:S})^T E^{k-1}$.

11.2.3 A subspace dogleg algorithm

We define the **Cauchy step** by

$$p_{\text{c}} := -t^* g_S, \quad t^* := \text{argmin}\{\widetilde{\mathcal{Q}}(-t g_S) \mid t \geq 0,\ \|t g_S\| \leq \Delta_{\text{nm}}\}. \tag{11.14}$$

The goal is to solve the trust region subproblem (11.13) such that

$$\|p\| \leq \Delta_{\text{nm}} \ \text{ and } \ \widetilde{\mathcal{Q}}(p) \leq \widetilde{\mathcal{Q}}(p_{\text{c}}) \tag{11.15}$$

hold. After the subspace Gauss-Newton direction is computed by (11.6), if it is outside a trust region, a **subspace dogleg algorithm**, called **subDogleg**, is used resulting in an estimated step enforcing (11.15).

The model function $\widetilde{\mathcal{Q}}$ is reduced by (11.14) if $dq_{\rm sn} := \widetilde{\mathcal{Q}}(0) - \widetilde{\mathcal{Q}}(p_{\rm sn}) > 0$. **subDogleg** first identifies whether $dq_{\rm sn} > 0$ or not. Then we have one of the following cases:

CASE 1. If $dq_{\rm sn} > 0$, the **scaled steepest descent step**

$$p_{\rm sd} := -\frac{g_S^T g_S}{(J_S g_S)^T (J_S g_S)} g_S \tag{11.16}$$

is computed. If it is outside the trust region, an estimated solution of (11.2) is either the Cauchy step computed by (11.14) or the **dogleg step**

$$p_{\rm dg} := p_{\rm sd} + t(p_{\rm sn} - p_{\rm sd}); \tag{11.17}$$

both of (11.16) and (11.17) are on the trust region boundary. Here t is found by solving the equation $\|p_{\rm sd} + t(p_{\rm sn} - p_{\rm sd})\| = \Delta_{\rm nm}$. If the condition $dp := (p_{\rm sd})^T(p_{\rm sn} - p_{\rm sd}) \le 0$ holds, a positive root is computed by

$$t := \frac{-dp + \sqrt{dp^2 + \|p_{\rm sn} - p_{\rm sd}\|(\Delta_{\rm nm}^2 - \|p_{\rm sd}\|^2)}}{\|p_{\rm sn} - p_{\rm sd}\|^2} \in (0, 1). \tag{11.18}$$

Otherwise, t is computed by

$$t := \frac{\Delta_{\rm nm}^2 - \|p_{\rm sd}\|^2}{dp + \sqrt{dp^2 + \|p_{\rm sn} - p_{\rm sd}\|(\Delta_{\rm nm}^2 - \|p_{\rm sd}\|^2)}} \in (0, 1); \tag{11.19}$$

e.g., see [138].

CASE 2. If $dq_{\rm sn} \le 0$, the model function $\widetilde{\mathcal{Q}}$ is convex since the matrix $(J_S g_S)^T (J_S g_S)$ is symmetric and positive semidefinite. An estimated solution of (11.2) is either $p_{\rm sd}$ computed by (11.16) if it is inside the trust region or the Cauchy step $p := \Delta_{\rm nm}(p_{\rm sd}/\|p_{\rm sd}\|)$ according to $p_{\rm sd}$, otherwise.

11.2.4 A Broyden-like technique

Before a successful iteration is found by a trust region algorithm, the trust region subproblems may be solved many times with high computational cost. Instead, our idea is to use a new algorithm based on the previous best points in the hope of finding a point with good function value.

Whenever **LMLS** cannot decrease the function value, a new Broyden-like technique, called **BroydenLike**, is used in the hope of getting a decrease in the function value. Let $x_1, \cdots, x_{m^{\rm rs}}$ be the $m^{\rm rs}$ best point stored in X. Then a point in the affine space spanned by such points has the following form

$$x^z := Xz, \quad z \in \mathbb{R}^{m^{\rm rs}}, \quad e^T z = 1, \tag{11.20}$$

where $e \in \mathbb{R}^{m^{\rm rs}}$ is a vector all of whose components are one. Given $B := \begin{pmatrix} X \\ e \end{pmatrix}$, the linear approximation $E(x^z) \approx Bz$ is used to replace (4.3) by the surrogate problem

$$\begin{aligned} \min \quad & \frac{1}{2}\|Bz\|_2^2 \\ \text{s.t.} \quad & e^T z = 1. \end{aligned} \tag{11.21}$$

This is a quality constrained convex quadratic problem in $m^{\rm rs}$ variables; hence can be solved in closed form. Then a QR factorization is made in the form $B = QR$, where Q is an orthogonal matrix and R is a square upper triangular matrix. By setting $Z := R^{-1}$, we make the substitution $z := Zy$, define $a^T := e^T Z$, and obtain the $m^{\rm rs}$-dimensional minimal norm linear feasibility problem

$$\begin{array}{ll} \min & \frac{1}{2}\|y\|_2^2 \\ \text{s.t.} & a^T y = 1 \end{array} \tag{11.22}$$

whose solution is $y := a/\|a\|_2$. Hence, a new trial point for the next algorithm is

$$x^{\rm trial} := x^z := XR^{-1}a/\|a\|_2. \tag{11.23}$$

BroydenLike tries to find a point with better function value when no decrease in f is found along p. It takes X, F, x, E, B, f, δ_f, and $f_{\rm nm}$ as input and uses the following tuning parameter:
$\underline{\gamma} \in (0,1)$ (tiny factor for adjusting δ_f),
$\overline{\gamma} > 1$ (parameter for expanding δ_f),
γ^s (tiny parameter for the finite difference step size),
$m^{\rm rs}$ (memory for affine space),
$0 < \gamma^r < 1$ (tiny parameter for adjusting the scaled random directions).
It returns a new δ_f and $f_{\rm nm}$ (and f, X, F, E, S, and J_S if a decrease in f is found) as output.

11.2.1 Algorithm. (**BroydenLike, a Broyden-like method**)
(BL$_0$) Make a QR factorization for the matrix B and get the square upper triangular matrix R.
(BL$_1$) **A new trial point.** Compute the new trial point $x^{\rm trial}$ by (11.23) and $E^{\rm trial} := E(x^{\rm trial})$. Then set $f^{\rm trial} := 0.5\|E^{\rm trial}\|^2$. If $f^{\rm trial} \le f_{\rm nm}$, then
(1) set $x := x^{\rm trial}$, $E := E^{\rm trial}$, and $f := f^{\rm trial}$;
(2) save x in X and f in F by **updateXF**;
(3) compute S and $J_S := J_{:S}(x)$ by **SRFD**;
Then update δ_f by (11.8) and compute $f_{\rm nm}$ by (11.9).

Since the Jacobian matrix may be singular or indefinite, a new point may move toward a maximizer. To remedy this disadvantage, **BroydenLike** does not lead to accept such a point with largest function value; like a maximizer.

11.2.5 A limited memory trust region algorithm

We describe all steps of a new **limited memory algorithm**, called **LMLS** using the new subspace direction (11.6), the new non-monotone technique (11.9), the new adaptive radius strategy (11.10), and **BroydenLike**.

In each iteration, an estimated solution of the trust region subproblem (11.13) is found. Whenever the condition $\rho_{\rm nm} \ge \gamma^t$ holds, the iteration is called **successful** while updating both the non-monotone term (11.9) and adaptive radius formula (11.10), and estimating the Jacobian matrix in an adaptive subspace by **SRFD**. Otherwise, the iteration is called unsuccessful. In this case, **BroydenLike** is performed in the hope of finding a decrease in the function value. If a decrease in the function value is found, the iteration becomes successful; otherwise, it remains unsuccessful

while reducing the radius and updating the non-monotone term (11.9) until a decrease in the function value is found and the iteration becomes successful.

LMLS solves a unconstrained nonlinear black box least squares problems. This algorithm takes the initial point x^0, and maximal number of function evaluations (`nfmax`). It uses the following tuning parameters:
$m^{\rm sn}$ (subspace size),
$\gamma^t \in (0,1)$ (parameter for trust region),
$0 < \sigma_1 < 1 < \sigma_2$ (parameters for updating λ),
$\gamma^{\rm init}$ (parameter for updating the initial δ_f),
$\underline{\gamma} \in (0,1)$ (tiny factor for adjusting δ_f),
$\overline{\gamma} > 1$ (parameter for expanding δ_f),
$\underline{\lambda}$ (lower bound for λ),
$\overline{\lambda}$ (upper bound for λ),
γ^s (tiny parameter for adjusting finite difference step sizes),
$0 < \gamma^r < 1$ (tiny parameter for adjusting random approximation coordinate directions).
It returns a solution $x_{\rm best}$ of a nonlinear least squares problem as output.

11.2.2 Algorithm. (**LMLS, limited memory method for least squares problems**)

($\rm LM_0$) **Initialization:**
(1) Choose $\lambda^0 \in (\underline{\lambda}, \overline{\lambda})$.
(2) Compute $E^0 := E(x^0)$ and set $f^0 := 0.5\|E^0\|^2$. Then set $X := x^0$ and $F := f^0$.
(3) Compute the initial subspace S and $J_{:S}(x^0)$ by **SRFD**.
(4) Compute $g_S^0 := J_{:S}^T(x^0)E^0$ and δ_f^0 by (11.8).
for $\ell = 0, 1, 2, \cdots$ **do**
($\rm LM_1$) Compute the subspace Gauss-Newton direction $p^\ell := p_{\rm sn}^\ell$ by (11.6).
($\rm LM_2$) If $\|p_{\rm sn}^\ell\| > \Delta_{\rm nm}^\ell$, **obtain an approximated solution** p^ℓ **of** (11.13) by calling **subDogleg**, which is a variant of **dogleg.m**, available at
`http://www.math.ubc.ca/~loew/m604/mfiles.htm`
but with the difference that
(1) t is recomputed by (11.19) when t computed by (11.18) is not positive,
(2) it can be handled for problems in high dimensions,
(3) the concavity of $\widetilde{\mathcal{Q}}$ is ignored,
(4) an adaptive subspace is used.
($\rm LM_3$) If $(g^\ell)^T p^\ell \geq 0$, which may happen, e.g., due to rounding errors, we modify the search direction according to $\texttt{ind} := \{i \mid g_i^\ell p_i^\ell > 0\}$ and $p_{\texttt{ind}}^\ell := -p_{\texttt{ind}}^\ell$ so that the **descent condition** $(g^\ell)^T p^\ell < 0$ holds.
($\rm LM_4$) **A new trial point:**
(1) Compute $x^{\rm trial} := x^\ell + p^\ell$, $E^{\rm trial} := E(x^{\rm trial})$, and $f^{\rm trial} := 0.5\|E^{\rm trial}\|^2$.
(2) If `nfmax` is exceeded, **LMLS** ends, resulting $x^{\rm best} = x^\ell$.
(3) Otherwise, compute $\rho_{\rm nm}^\ell$ by (11.12); if $\rho_{\rm nm}^\ell < \gamma^t$, go to ($\rm LM_5$); otherwise, go to ($\rm LM_6$).
($\rm LM_5$) **Unsuccessful iteration:**
(1) Call **BroydenLike**. If a decrease in f is found go to ($\rm LM_6$).
(2) Update $\delta_f^{\ell+1}$ by (11.8) and $f_{\rm nm}^{\ell+1}$ by (11.9).
(3) Reduce $\lambda^{\ell+1}$ to (11.11) and the radius $\Delta_{\rm nm}^{\ell+1}$ to (11.10). Then go to ($\rm LM_2$).
($\rm LM_6$) **Successful iteration:**
(1) Set $x^{\ell+1} := x^{\rm trial}$, $f^{\ell+1} := f^{\rm trial}$, and $E^{\ell+1} := E^{\rm trial}$.
(2) Compute the new subspace S and $J_{:S}(x^{\ell+1})$ by **SRFD** and compute $g_S^{\ell+1} := J_{:S}(x^{\ell+1})^T E^{\ell+1}$.
(3) Update $\delta_f^{\ell+1}$ by (11.8) and $f_{\rm nm}^{\ell+1}$ by (11.9).

(4) Expand $\lambda^{\ell+1}$ to (11.11) and $\Delta_{\text{nm}}^{\ell+1}$ to (11.10).
(5) Store $x^{\ell+1}$ in X and $f^{\ell+1}$ in F by **updateXF**.
end for

LMLS was implemented in Matlab; the source code is available at

`http://www.mat.univie.ac.at/~neum/software/LMLS`.

11.3 Numerical results

We updated the test environment discussed in Section 5 to use test problems by Lukšan et al. [127], classified in Table 11.2 according to whether they are **overdetermined** ($r > n$) or not ($r = n$). A shifted point for these problems is done by (5.1). **LMLS** is compared with unconstrained least squares and unconstrained optimization solvers (discussed in Subsection 11.4.3), for some of which we had to choose options different from the default to make them competitive.

`nfmax` and `secmax` are chosen as

$$\texttt{nfmax} \in \begin{cases} \{10n, 50n, 100n, 500n\} & \text{if } 1 \le n \le 100, \\ \{10n, 50n, 100n\} & \text{if } 101 \le n \le 1000 \\ \{10n, 100n\} & \text{if } 1001 \le n \le 10000 \end{cases}$$

and

$$\texttt{secmax} := \begin{cases} 300 & \text{if } 1 \le n \le 100, \\ 800 & \text{if } 101 \le n \le 10000. \end{cases}$$

If the condition

$$q^{so} \le \begin{cases} 10^{-8} & \text{if } 1 \le n \le 100, \\ 10^{-3} & \text{if } 101 \le n \le 10000, \end{cases}$$

holds then the problem is solved. Otherwise, the problem is unsolved; either `nfmax` or `secmax` is exceeded, or the solver fails. Here q^{so} comes from (5.3) and the `nf` efficiency comes from Subsection 5.4.

11.3.1 Small scale: $1 \le n \le 100$

A comparison among **LMLS1**, **LMLS2**, **LMLS3**, **LMLS4**, and solvers using quasi Newton is shown in Subfigures (a) and (b) of Figure 11.1, so that
• **LMLS4** using the full estimated Jacobian matrix is the best in terms of the number of solved problems and the `nf` efficiency;
• **LMLS3**, **LMLS2**, and **LMLS1** are more efficient than solvers using quasi Newton approximation (**FMINUNC**, **FMINUNC1**, **MINFLBFGS1**, and **MINFLBFGSDL1**) in terms of the `nf` efficiency;
• **FMINUNC** and **MINFLBFGSDL1** are comparable with **LMLS3** – only for very large budget – in terms of the number of solved problems but **LMLS3**, **LMLS2**, and **LMLS1** are more efficient than **FMINUNC** and **MINFLBFGSDL1** in terms of the `nf` efficiency not only for very large budget but also for small up to large budgets.

To determine whether our new non-monotone and adaptive radius strategies are effective or not, we compare **LMLS4** with solvers using other non-monotone and adaptive radius strategies, shown in Subfigures (c) and (d) of Figure 11.1. All solvers use the full Jacobian matrix and the trust region subproblems are solved by the same algorithm. As can be seen, **LMLS4** is much more efficient and robust than **NATRLS1**, **NMPGTR2**, and **NATRN1** in terms of the number of solved problems and the `nf` efficiency.

We compare **LMLS4** with four famous solvers **LSQNONLIN1**, **CoDoSol1**, **NLEQ1**, and **DOGLEG1** shown in Subfigures (e) and (f) of Figure 11.1. It is seen that **LMLS4** and **CoDoSol1** are the two best solvers in terms of the `nf` efficiency while **LMLS4** and **DOGLEG1** are the two best solvers in terms of the number of solved problems.

Another comparison is among **LMLS3**, **LMLS2**, and **LMLS1** using the Jacobian matrix in adaptive subspace and **LSQNONLIN1** and **NLEQ1** using the full Jacobian matrix. We conclude from Subfigures (g) and (h) of Figure 11.1 that
• **LMLS3** is the best in terms of the number of solved problems and the `nf` efficiency;
• **LMLS2** is the second best solver in terms of the number of solved problems and the `nf` efficiency for medium, large, and very large budgets;
• **LMLS1** with lowest subspace size is more efficient than **LSQNONLIN1** in terms of the number of solved problems and the `nf` efficiency; even it is more efficient than **NLEQ1** for very large budget in terms of the `nf` efficiency.

As a result, **LMLS** is competitive for small scale problems in comparison with the state-of-the-art solvers.

All efficiency tables and performance plots are given in Subsection 11.4.4.

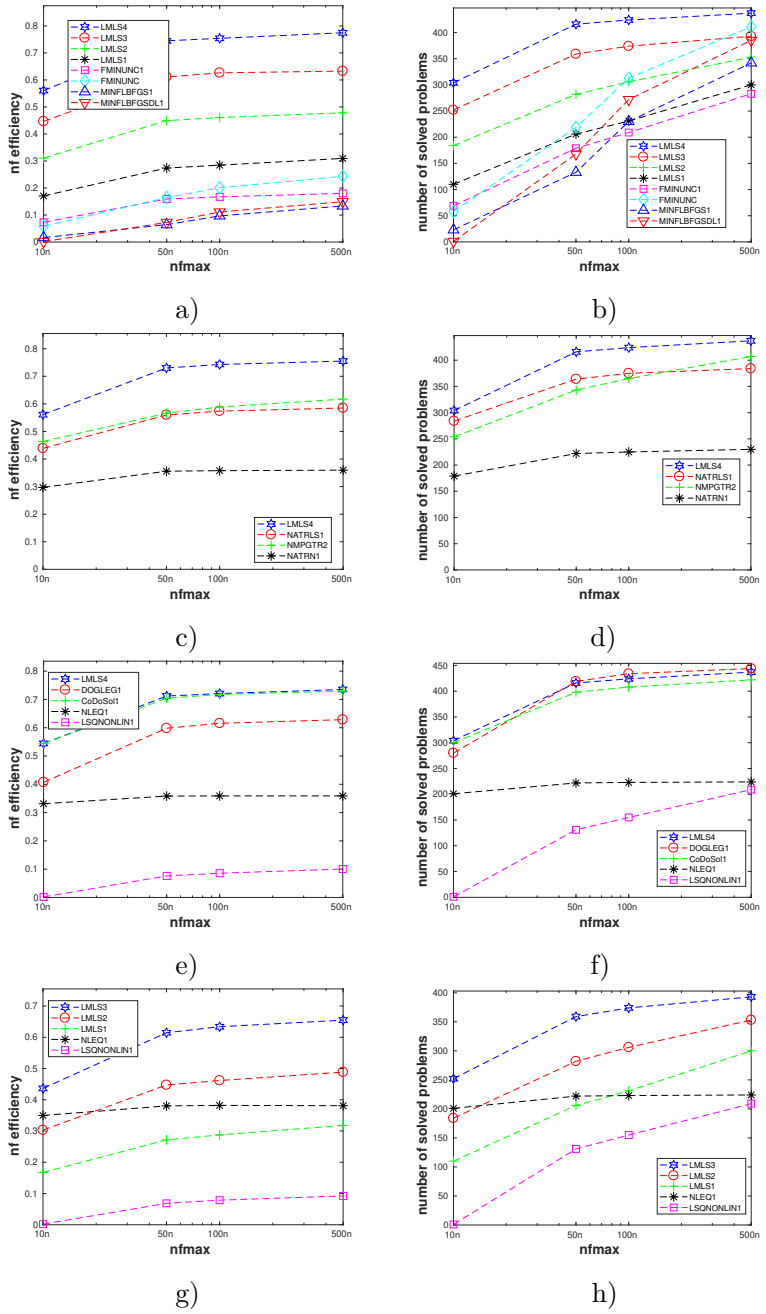


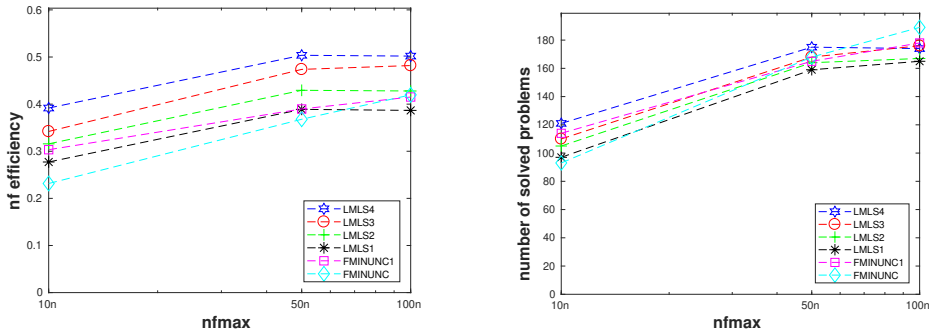
Figure 11.1: Performance plots for small scale problems. (a) – (b): A comparison of limited memory solvers, (c) – (d): A comparison among **LMLS** in a full subspace and solvers using other non-monotone and adaptive radius techniques, (e) – (f): A comparison among **LMLS** in a full subspace and other famous solvers, (g) – (h): A comparison among low-dimensional **LMLS1**, **LMLS2**, **LMLS3** and **NLEQ1** and **LSQNONLIN1** using full estimated Jacobian.

11.3.2 Medium scale: $101 \leq n \leq 1000$

In this subsection, we compare **LMLS1**, **LMLS2**, **LMLS3**, **LMLS4** using the estimated Jacobian matrices in an adaptive subspace with **FMINUNC** using standard BFGS approximations and **FMINUNC1** using limited memory BFGS ones.

From Subfigures (a) and (b) of Figure 11.2, we conclude that
• **LMLS4**, **LMLS3**, and **LMLS2** are the three best solvers in terms of the `nf` efficiency, respectively;
• **LMLS4** is the best solver in terms of the number of solved problems; only **FMINUNC** is the best for large budget.

All efficiency tables and performance plots are given in Subsection 11.4.4.



a) b)

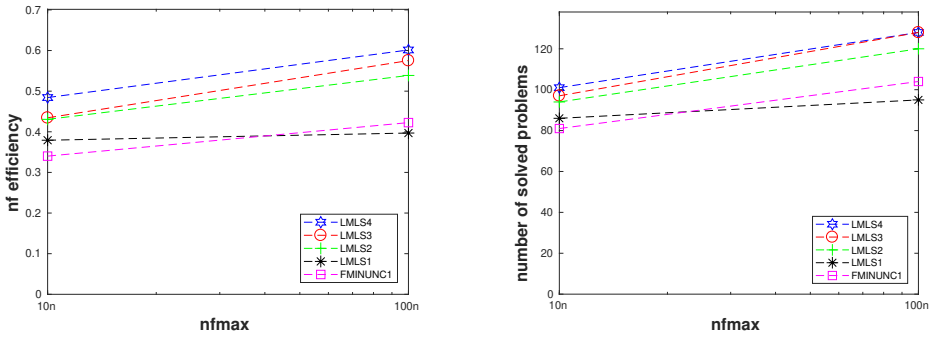
Figure 11.2: (a) – (b): Performance plots for medium scale problems

11.3.3 Large scale: $1001 \leq n \leq 10000$

In this subsection, we compare **LMLS1**, **LMLS2**, **LMLS3**, **LMLS4** using the estimated Jacobian matrices in an adaptive subspace with **FMINUNC1** using limited memory BFGS approximations.

In terms of the `nf` efficiency and the number of solved problems, Subfigures (a) and (b) of Figure 11.3 result in the fact that
• **LMLS4**, **LMLS3**, and **LMLS2** are the three best solvers, respectively;
• **LMLS1** with lowest subspace size is more efficient than **FMINUNC1** for small budget while **FMINUNC1** is more efficient than **LMLS1** for large budget.

All efficiency tables and performance plots are given in Subsection 11.4.4.



a) b)

Figure 11.3: (a) – (b): Performance plots for large scale problems

11.4 Additional material for LMLS

This section discusses additional material for **LMLS**.

11.4.1 Default tuning parameters for LMLS

The tuning parameters for **LMLS** are chosen as

$$m^{\text{nm}} = 10;\ \gamma^r = 10^{-30};\ \gamma^p = 5\varepsilon_m;\ \gamma^s = \sqrt{\varepsilon_m};\ \gamma^t = 0.1;\ \sigma_1 = 0.5;$$
$$\sigma_2 = 1;\ \gamma^{\text{init}} = 10^{-8};\ \gamma^m = 10^{-30};\ \underline{\lambda} = 10^{-4};\ \Delta^0 = 10;\ \Delta^{\min} = 10^{-6};$$
$$\lambda^0 = 1;\ \overline{\lambda} = 10^5;$$

11.4.2 Test problem selection

Table 11.2: A classification of test problems

Dimensions (n)	3	5	10	16	30	50	100	300	500	1000	5000	10000
Total number of least squares problems ($r \geq n$)	49	69	76	83	83	83	83	83	83	83	83	83
Number of square problems ($r = n$)	43	53	55	62	62	62	62	62	62	62	62	62
Number of least squares problems with $r > n$	6	16	21	21	21	21	21	21	21	21	21	21

11.4.3 Codes compared

• **CoDoSol** is a solver for constrained nonlinear systems of equations, obtained from

`http://codosol.de.unifi.it`.

It combines Newton method and a trust-region method, see [13]. The following option used

$$\texttt{parms} = [\texttt{maxit}, \texttt{maxnf}, \texttt{tr}, \texttt{delta}, \texttt{scaling}, \texttt{outflag}] = [\texttt{inf}, \texttt{nfmax}, 1, -1, 0, 2].$$

Note that $\texttt{delta} = -1$ means that $\Delta_0 = 1$. According to our numerical results, **CoDoSol** was not sensitive for the initial radius; hence, the default used.
• **STRSCNE** is a solver for constrained nonlinear systems of equations, obtained from

`http://codosol.de.unifi.it`.

It combines Newton method and a trust-region procedure, see [12]. The option

$$\texttt{parms} = [\texttt{maxit}, \texttt{maxnf}, \texttt{delta}, \texttt{outflag}] = [\texttt{inf}, \texttt{nfmax}, -1, 0]$$

used. Note that $\texttt{delta} = -1$ means that $\Delta_0 = 1$. According to our numerical results, **STRSCNE** was not sensitive for the initial radius; hence, the default used for Δ_0.
• **NLEQ1** is a damped affine invariant Newton method for nonlinear systems, obtained from

`http://elib.zib.de/pub/elib/codelib/nleq1_m/nleq1.m`

and suggested by Deuflhard [57] and written by Nowak & Weimann [141]. The default tuning parameters used; only we selected $\texttt{iopt.jacgen} = 2$, $\texttt{iopt.qrank1} = 1$ and $\texttt{wk.fmin} = \texttt{1e} - \texttt{50}$.
• **NLEQ2** is the same as **NLEQ1**; only we selected $\texttt{iopt.qrank1} = 0$.
• **MINLBFGS** is a L-BFGS with line search algorithm, obtained from

`https://www.tensorlab.net/`

and written by Sorber et al. [155]. The default parameters used, except for m. **MINLBFGS1** and **MINLBFGS2** denote **MINLBFGS** with $m = \min(n, 30)$ and $m = \min(n, 100)$, respectively.
• **MINLBFGSDL** is a L-BFGS with dogleg trust region algorithm with the option set

$$\texttt{MaxIter} = \texttt{nfmax}, \quad \texttt{TolFun} = \texttt{1e} - \texttt{50}, \quad \texttt{TolX} = \texttt{1e} - \texttt{50}.$$

The default parameters used for `PlaneSearch` and M. **MINLBFGSDL1**, **MINLBFGSDL2**, **MINLBFGSDL3**, and **MINLBFGSDL4** denoted **MINLBFGSDL** with
(1) `PlaneSearch` = `false` and $M = \min(30, n)$,
(2) `PlaneSearch` = `false` and $M = \min(100, n)$,
(3) `PlaneSearch` = `true` and $M = \min(30, n)$,
(4) `PlaneSearch` = `true` and $M = \min(100, n)$.
According to our results, **MINLBFGSDL1** was the best.
• **MINFNCG** is a nonlinear conjugate gradient solver, obtained from

`https://www.tensorlab.net/`

and written by Sorber et al. [155]. It used the following option set

$$\texttt{MaxIter} = \texttt{nfmax};\ \ \texttt{TolFun} = \texttt{1e}-\texttt{50};\ \ \texttt{TolX} = \texttt{1e}-\texttt{50}.$$

The other tuning parameter was `Beta` $\in \{$`HS`, `HSm`, `PR`, `FR`, `PRm`, `SD`$\}$. **MINFNCG1**, **MINFNCG2**, **MINFNCG3**, **MINFNCG4**, **MINFNCG5**, and **MINFNCG6** denoted **MINFNCG** with `Beta` = `HS`, `Beta` = `HSm`, `Beta` = `PR`, `Beta` = `FR`, `Beta` = `PRm`, and `Beta` = `SD`, respectively.
• **NLSQERR** is a global unconstrained Gauss-Newton method with error oriented convergence criterion and adaptive trust region strategy [57], obtained from

`http://elib.zib.de/pub/elib/codelib/NewtonLib/index.html`

The following options used

$$\begin{aligned}&\texttt{iniscalx} = 0;\ \ \texttt{rescalx} = 0;\ \ \texttt{xthrsh} = \texttt{ones}(n, 1);\\ &\texttt{xtol} = \texttt{1.e}-\texttt{50};\ \ \texttt{ftol} = \texttt{1.e}-\texttt{50};\ \ \texttt{kmax} = \texttt{nfmax};\\ &\texttt{printmon} = 2;\ \ \texttt{printsol} = 1;\ \ \texttt{fid} = 1;\ \ \texttt{numdif} = 1;\\ &\texttt{lambda0} = \texttt{eps};\ \ \texttt{lambdamin} = \texttt{1e}-\texttt{50};\ \ \texttt{ftol} = \texttt{1.e}-\texttt{50}.\end{aligned}$$

• **NMPNTR**, nonmonotone projected Newton trust region method, is a bound constrained solver [112]. **NMPNTR1**, **NMPNTR2**, **NMPNTR3**, and **NMPNTR4** denoted **NMPNTR** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. According to our results, **NMPNTR2** was the best.
• **NATRN** is a nonmonotone trust region algorithm [5] using the full finite difference approximation. The subproblem solved in the same way as **LMLS**. **NATRN1**, **NATRN2**, **NATRN3**, and **NATRN4** denoted **NATRN** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. According to our results, **NATRN1** had the best performance.
• **NATRLS** is a nonmonotne line search and trust region algorithm [6] using the full finite difference approximation. The subproblem solved in the same way as **LMLS**. **NATRLS1**, **NATRLS2**, **NATRLS3**, and **NATRLS4** denoted **NATRLS** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. According to our results, **NATRLS1** had the best performance.
• **LSQNONLIN1**, obtained from the Matlab Optimization Toolbox at,

`https://de.mathworks.com/help/optim/ug/lsqnonlin.html`

is nonlinear least-squares solver with the following options:

`options = optimoptions(@lsqnonlin,'Algorithm', 'levenberg-marquardt',`
`'FiniteDifferenceType','forward', 'MaxIter', Inf,'MaxFunEvals', nfmax,`

'TolX', 0,'SpecifyObjectiveGradient','false').
• **LSQNONLIN2**, obtained from the Matlab Optimization Toolbox at,

https://de.mathworks.com/help/optim/ug/lsqnonlin.html

is nonlinear least-squares solver with the following options:

options = optimoptions(@lsqnonlin,'Algorithm', 'trust-region reflective', 'FiniteDifferenceType','forward','MaxIter', Inf,'MaxFunEvals', nfmax, 'TolX', 0,'SpecifyObjectiveGradient','false').
• **DOGLEG** is Powell's dogleg method for least squares problems, which is the best algorithm from the toolbox of immoptibox.zip [138], available at

http://www2.imm.dtu.dk/projects/immoptibox/

The following option used

$$\texttt{opts} = [\Delta_0, \texttt{tolg}, \texttt{tolx}, \texttt{tolr}, \texttt{maxeval}] = [\Delta_0, \texttt{1e}-\texttt{50}, \texttt{1e}-\texttt{50}, \texttt{1e}-\texttt{50}, \texttt{nfmax}].$$

DOGLEG1, **DOGLEG2**, **DOGLEG3**, and **DOGLEG4** denoted **DOGLEG** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. The best version was **DOGLEG1**.

Unconstrained solvers:
• **FMINUNC**, obtained from the Matlab Optimization Toolbox at

https://ch.mathworks.com/help/optim/ug/fminunc.html,

is a standard quasi-Newton algorithm. We used **FMINUNC** with the options set

opts = optimoptions(@fminunc),'Algorithm','quasi-newton', 'Display', 'Iter', 'MaxIter',Inf,'MaxFunEvals', nfmax, 'TolX', 0,'TolFun',0,'ObjectiveLimit',-1e-50).
• **FMINUNC1** is **FMINUNC** with the limited memory quasi Newton approximation by Liu & Nocedal [124]. It has been added to **FMINUNC** by the present authors. The option set for it is the same as **FMINUNC**; only the memory $m = 10$ added to the option set.

11.4.4 Tables and plots

Table 11.3: Results for small scale and small budget

stopping test: $q_f \leq$ 1e-08, sec $\leq$ 300, nf $\leq$ 10*n										
367 of 526 problems without bounds solved									mean efficiency in %	
dim∈[1,100]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmtr4**	304	204	15	52	222	0	0	53	46
CoDoSol1	**codo1**	300	197	30	56	219	0	7	52	40
NATRLS1	**natrs1**	284	63	8	67	242	0	0	40	30
DOGLEG1	**dogleg1**	280	70	11	98	246	0	0	38	22
NMPGTR2	**nmpg2**	254	172	8	55	272	0	0	43	31
LMLS3	**lmtr3**	252	156	4	53	274	0	0	42	35
NLEQ1	**nleq1**	201	42	30	77	145	0	180	33	16
LMLS2	**lmtr2**	184	91	7	75	342	0	0	29	22
NATRN1	**natrn1**	179	54	11	89	347	0	0	26	18
LMLS1	**lmtr1**	110	47	20	112	416	0	0	16	10
STRSCNE1	**strs1**	70	69	0	28	107	0	349	13	7
FMINUNC1	**func1**	69	2	0	123	455	0	2	6	4
FMINUNC	**func**	58	2	0	139	468	0	0	5	4
MINFLBFGS1	**lbfgs1**	23	0	0	227	449	0	54	1	0
LSQNONLIN1	**lsqn1**	1	1	1	50	525	0	0	0	0
MINFLBFGSDL1	**minlbfgs1**	1	0	0	30	525	0	0	0	0
MINFNCG1	**MINFNCG1**	0	0	0	–	514	0	12	0	0

Table 11.4: Results for small scale and medium budget

stopping test: $q_f \leq$ 1e-08, sec $\leq$ 300, nf $\leq$ 50*n										
474 of 526 problems without bounds solved									mean efficiency in %	
dim∈[1,100]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
DOGLEG1	**dogleg1**	419	80	17	148	107	0	0	55	30
LMLS4	**lmtr4**	416	228	17	108	110	0	0	66	59
CoDoSol1	**codo1**	398	226	52	90	107	0	21	65	54
NATRLS1	**natrs1**	364	79	21	81	162	0	0	51	39
LMLS3	**lmtr3**	359	177	8	124	167	0	0	55	46
NMPGTR2	**nmpg2**	343	191	22	79	183	0	0	53	40
LMLS2	**lmtr2**	282	111	16	197	244	0	0	40	29
NATRN1	**natrn1**	222	63	17	117	304	0	0	31	22
NLEQ1	**nleq1**	222	42	30	77	122	0	182	35	18
FMINUNC	**func**	219	4	3	167	292	0	15	13	11
LMLS1	**lmtr1**	206	49	21	253	320	0	0	24	15
FMINUNC1	**func1**	179	8	7	177	345	0	2	13	10
MINFLBFGSDL1	**minlbfgs1**	168	0	0	286	358	0	0	5	5
MINFLBFGS1	**lbfgs1**	133	0	0	200	339	0	54	5	4
LSQNONLIN1	**lsqn1**	131	1	1	314	395	0	0	5	4
STRSCNE1	**strs1**	72	69	0	28	1	0	453	13	8
MINFNCG1	**MINFNCG1**	47	0	0	698	458	0	21	1	1

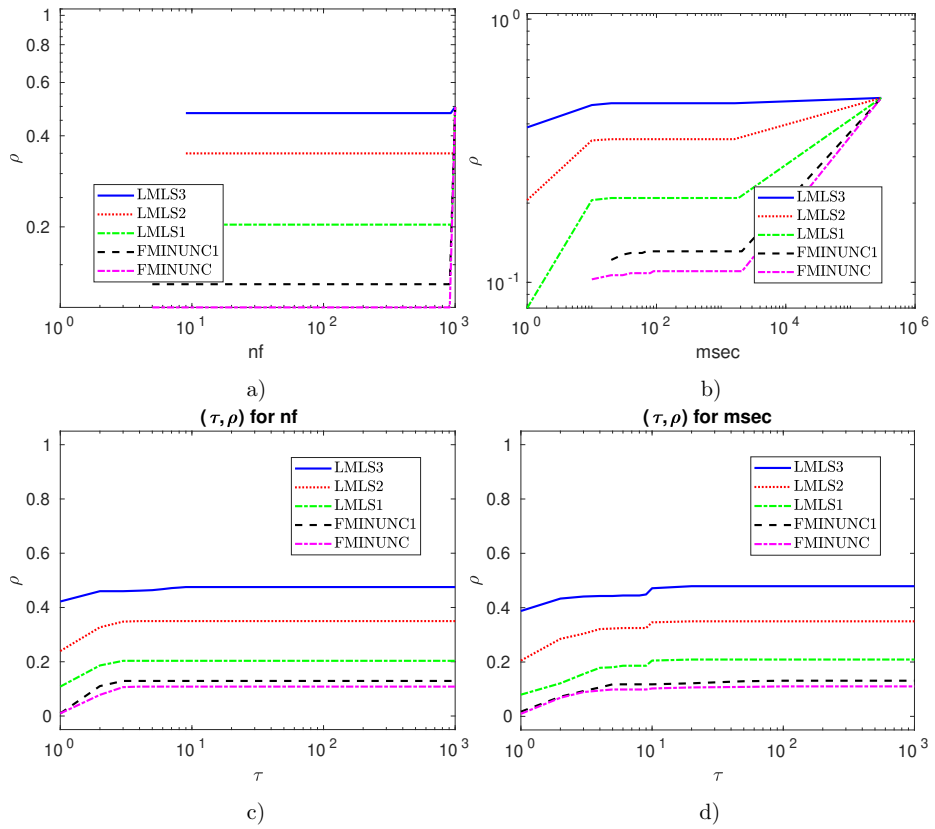


Figure 11.4: (a) and (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

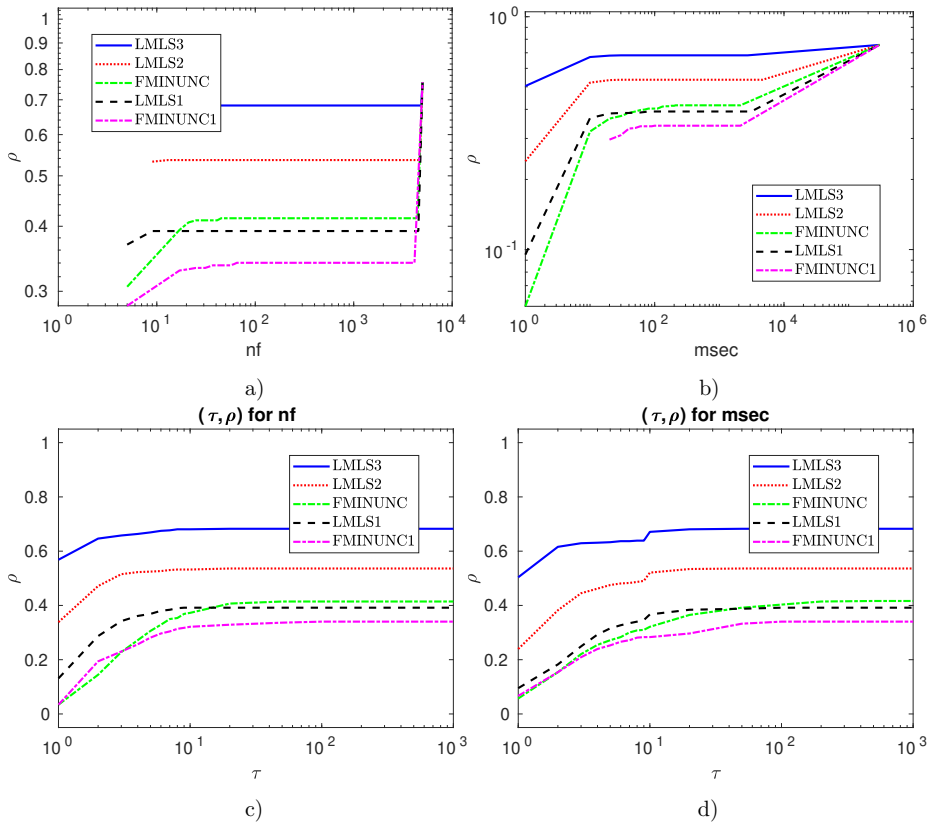


Figure 11.5: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 11.5: Results for small scale and large budget

stopping test: $q_f \leq$ 1e-08, sec $\leq$ 300, nf $\leq$ 100*n										
493 of 526 problems without bounds solved										mean efficiency in %
dim∈[1,100]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
DOGLEG1	**dogleg1**	434	82	19	167	92	0	0	56	31
LMLS4	**lmtr4**	424	229	17	113	102	0	0	67	59
CoDoSol1	**codo1**	408	227	53	103	88	0	30	66	54
NATRLS1	**natrs1**	375	87	27	104	151	0	0	53	39
LMLS3	**lmtr3**	374	177	9	150	152	0	0	56	48
NMPGTR2	**nmpg2**	365	196	25	103	160	0	1	54	40
FMINUNC	**func**	313	6	4	351	170	0	43	16	14
LMLS2	**lmtr2**	306	106	12	257	220	0	0	41	31
MINFLBFGSDL1	**minlbfgs1**	272	2	2	453	244	0	10	9	8
LMLS1	**lmtr1**	231	53	24	437	295	0	0	25	17
MINFLBFGS1	**lbfgs1**	230	0	0	349	232	0	64	8	6
NATRN1	**natrn1**	225	65	19	121	301	0	0	32	22
NLEQ1	**nleq1**	223	43	30	79	121	0	182	35	17
FMINUNC1	**func1**	209	8	6	327	297	0	20	14	12
LSQNONLIN1	**lsqn1**	155	1	1	374	371	0	0	6	4
MINFNCG1	**MINFNCG1**	136	0	0	524	365	0	25	2	2
STRSCNE1	**strs1**	72	69	0	28	1	0	453	13	7

Table 11.6: Results for small scale and very large budget

stopping test: $q_f \leq$ 1e-08, sec $\leq$ 300, nf $\leq$ 500*n										
509 of 526 problems without bounds solved										mean efficiency in %
dim∈[1,100]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
DOGLEG1	**dogleg1**	444	79	18	220	82	0	0	57	31
LMLS4	**lmtr4**	437	229	18	159	89	0	0	68	58
CoDoSol1	**codo1**	422	226	54	172	61	0	43	67	55
FMINUNC	**func**	411	8	7	761	20	0	95	19	17
NMPGTR2	**nmpg2**	407	194	25	378	110	1	8	56	42
LMLS3	**lmtr3**	393	178	9	343	133	0	0	56	47
MINFLBFGSDL1	**minlbfgs1**	385	6	6	723	69	0	72	11	12
NATRLS1	**natrs1**	384	89	31	143	142	0	0	54	41
LMLS2	**lmtr2**	353	107	14	998	173	0	0	42	31
MINFLBFGS1	**lbfgs1**	342	4	4	973	56	0	128	10	9
LMLS1	**lmtr1**	300	53	25	1302	226	0	0	27	17
MINFNCG1	**MINFNCG1**	283	0	0	990	194	0	49	3	4
FMINUNC1	**func1**	283	8	7	534	205	0	38	15	13
NATRN1	**natrn1**	230	63	17	178	296	0	0	32	22
NLEQ1	**nleq1**	224	42	30	82	120	0	182	35	18
LSQNONLIN1	**lsqn1**	209	1	1	690	316	1	0	6	5
STRSCNE1	**strs1**	72	69	0	29	1	0	453	13	6

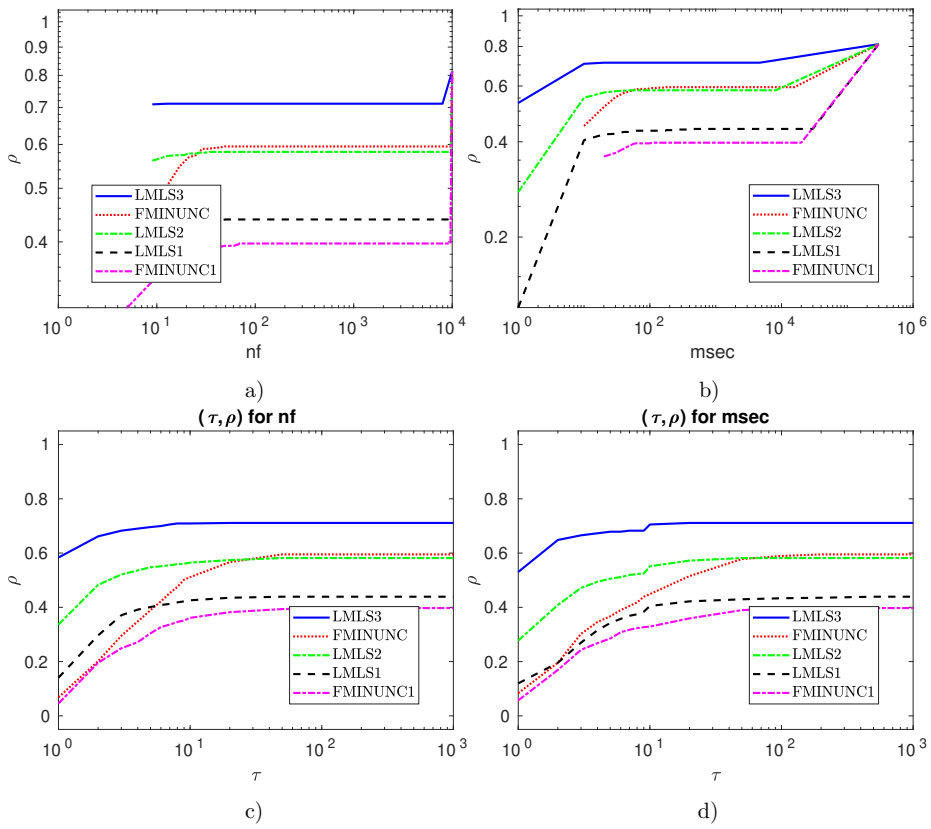


Figure 11.6: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

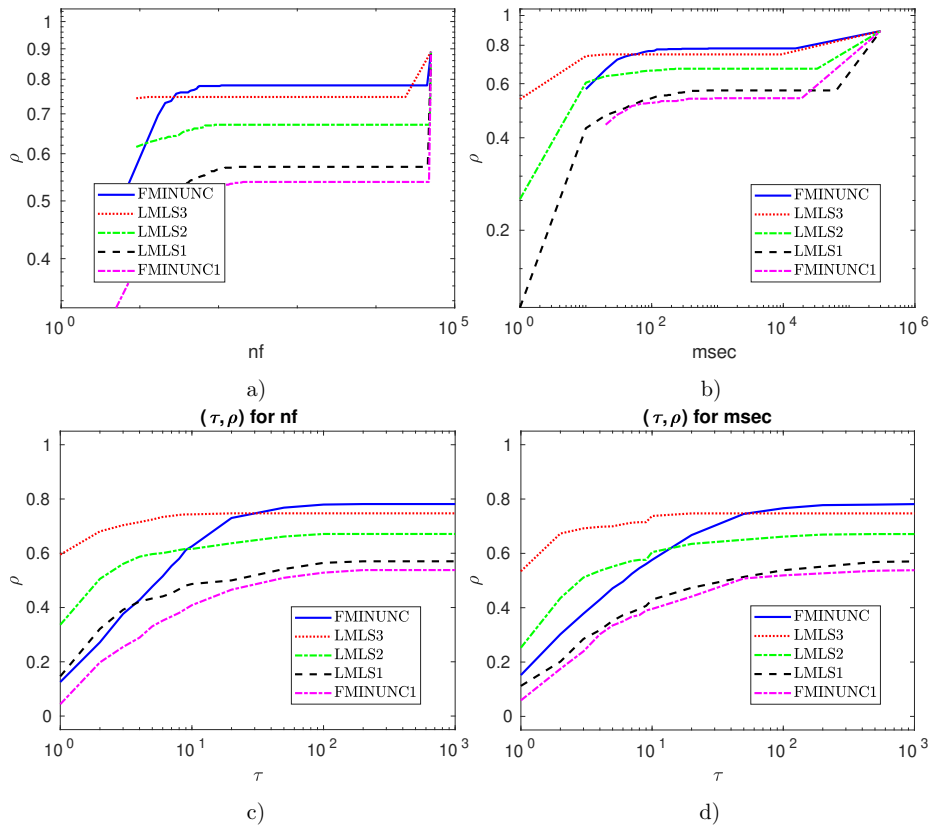


Figure 11.7: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 11.7: Results for medium scale and small budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 10*n				
154 of 249 problems without bounds solved									mean efficiency in %	
dim∈[101,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmls4**	119	49	48	13738	129	1	0	37	37
FMINUNC1	**func1**	114	40	14	11271	133	2	0	29	35
LMLS3	**lmtr3**	110	30	28	15401	138	1	0	34	28
LMLS2	**lmtr2**	105	22	21	15006	143	1	0	31	20
LMLS1	**lmtr1**	97	15	15	18207	151	1	0	27	21
FMINUNC	**func**	93	26	0	13702	154	2	0	22	24

Table 11.8: Results for medium scale and budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 50*n				
188 of 249 problems without bounds solved									mean efficiency in %	
dim∈[101,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmtr4**	169	58	56	11582	78	2	0	50	46
LMLS3	**lmtr3**	168	48	47	12980	79	2	0	47	37
FMINUNC	**func**	168	32	5	11064	78	3	0	36	42
FMINUNC1	**func1**	165	43	16	10503	81	3	0	39	50
LMLS2	**lmtr2**	164	16	15	15614	84	1	0	43	25
LMLS1	**lmtr1**	159	20	20	15399	88	2	0	38	28

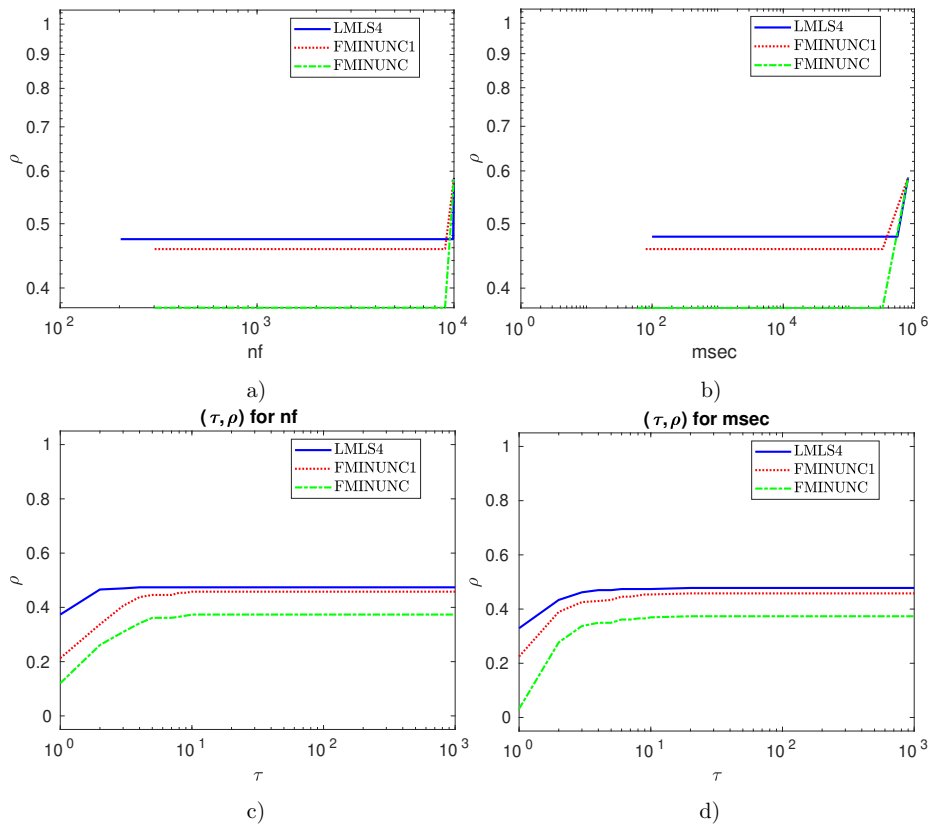


Figure 11.8: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

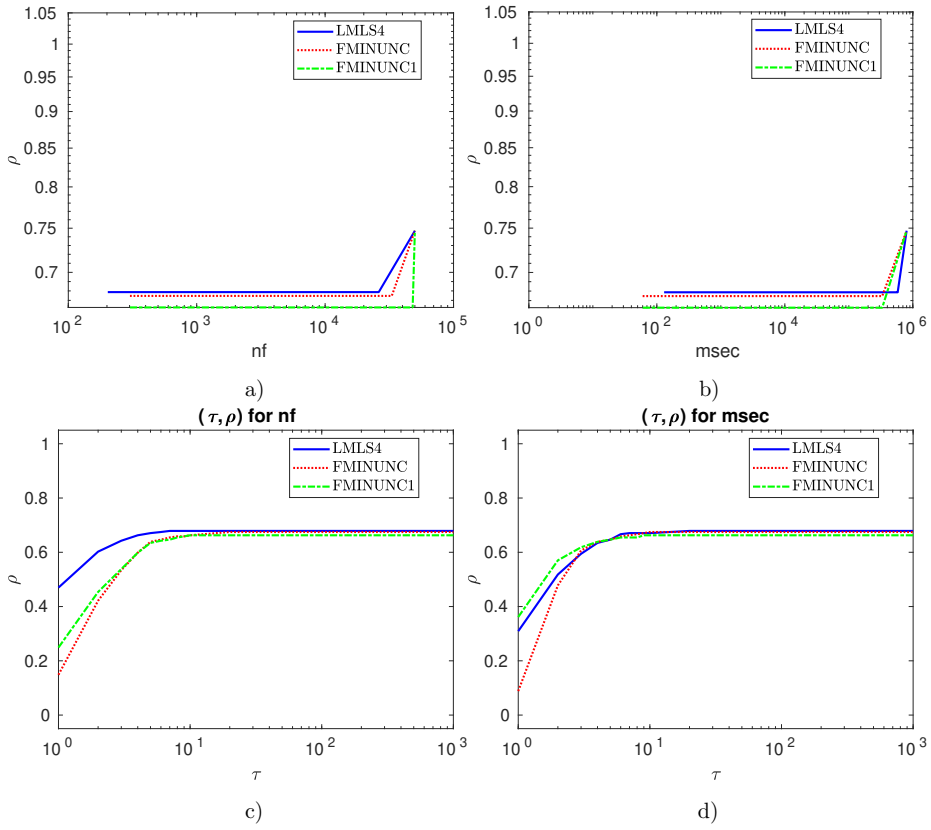


Figure 11.9: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

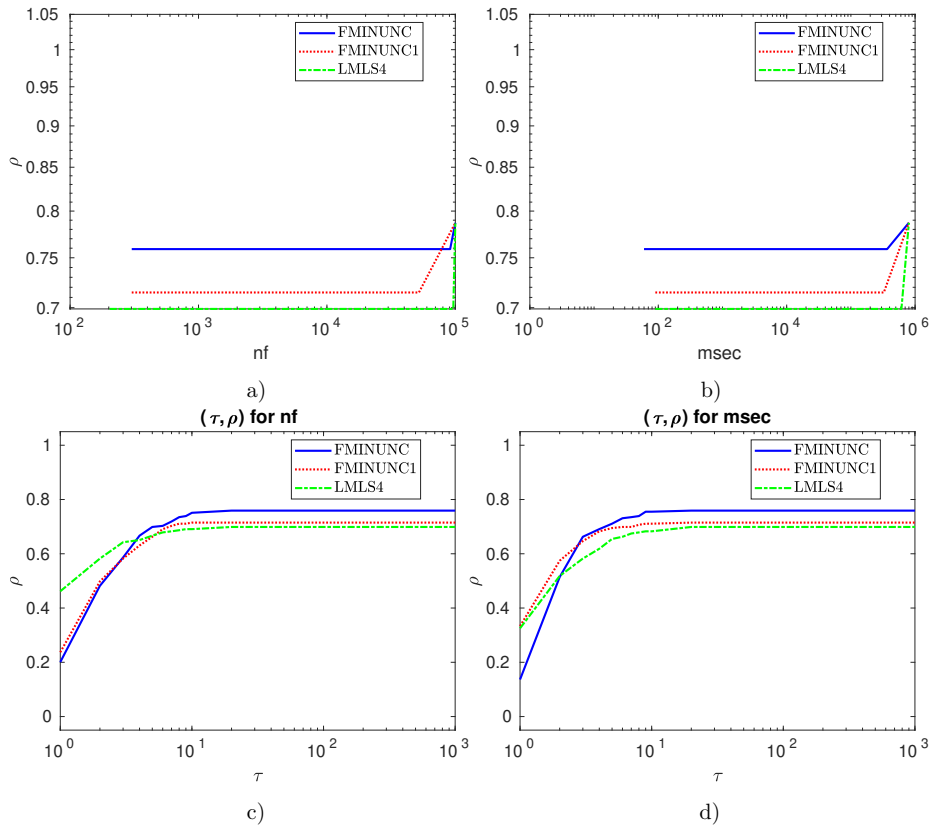


Figure 11.10: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 11.9: Results for medium scale and large budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 100*n				
198 of 249 problems without bounds solved									mean efficiency in %	
dim∈[101,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
FMINUNC	**func**	189	44	18	14603	55	3	2	41	48
FMINUNC1	**func1**	178	46	20	12133	66	3	2	41	52
LMLS3	**lmtr3**	176	39	37	16832	71	2	0	48	36
LMLS4	**lmtr4**	174	53	51	18938	73	2	0	50	47
LMLS2	**lmtr2**	167	24	24	18182	81	1	0	42	24
LMLS1	**lmtr1**	165	20	20	28711	84	0	0	38	26

Table 11.10: Results for large scale and small budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 10*n				
132 of 166 problems without bounds solved									mean efficiency in %	
dim∈[1001,10000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmtr4**	101	47	44	265873	47	17	1	48	45
LMLS3	**lmtr3**	97	22	19	242590	57	10	2	43	42
LMLS2	**lmtr2**	94	11	10	262404	60	8	4	43	41
LMLS1	**lmtr1**	86	22	21	302069	60	14	6	37	32
FMINUNC1	**func1**	81	36	35	197750	60	24	1	34	37

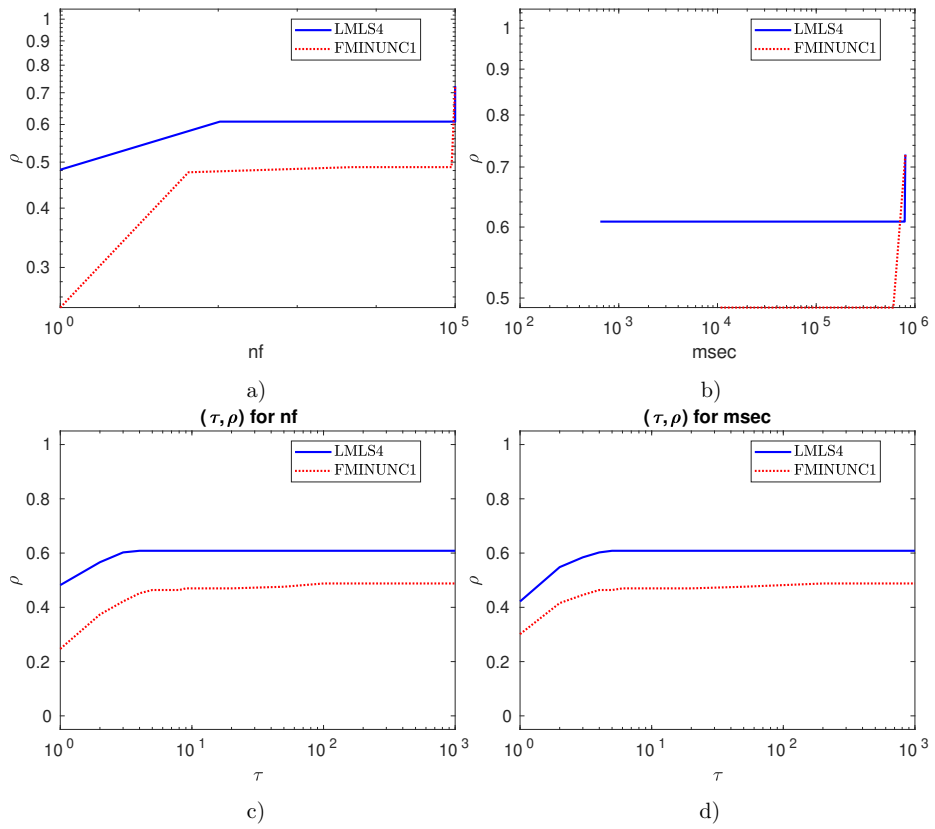


Figure 11.11: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 11.11: Results for large scale and budget

stopping test:		$q_f \leq$ 0.001,			sec $\leq$ 800,	nf $\leq$ 100*n				
147 of 166 problems without bounds solved									mean efficiency in %	
dim∈[1001,10000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS3	**lmtr3**	128	32	29	313348	0	38	0	57	55
LMLS4	**lmtr4**	128	49	47	325880	0	38	0	60	55
LMLS2	**lmtr2**	120	17	15	306190	0	46	0	53	49
FMINUNC1	**func1**	104	37	36	236336	0	61	1	42	47
LMLS1	**lmtr1**	95	18	17	332592	0	71	0	39	33

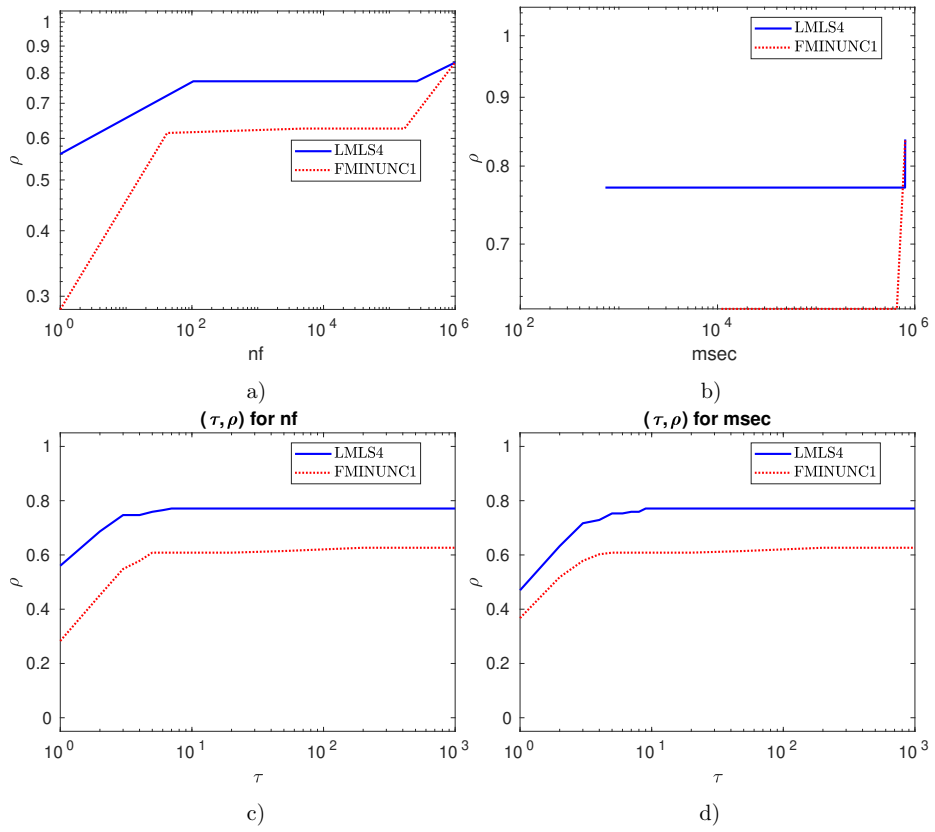


Figure 11.12: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Part IV

Conclusion

This thesis designed and tested a number of solvers using subspace methods to handle unconstrained and bound constrained optimization problems in high dimensions. These solvers used many practical enhancements that do not affect the order of our complexity results.

Section 5 constructed a new test environment with an automatic algorithm evaluation, significantly saving user time, performing statistics, and resulting in a summarized result as both pdf-file and tex-file with efficiency tables and performances plots.

Section 6 discussed a new limited memory method for the bound constrained optimization problem (4.1) with the exact function value and gradient – called **LMBOPT**. It was a joint work with Arnold Neumaier and Behzad Azmi (cf. Kimiaei, Neumaier, & Azmi [118]). **LMBOPT** was the improved version of the active set strategy by Neumaier & Azmi [136]. The following enhancements turned **LMBOPT** into a competitive solver:
• A starting direction according to the gradient signs.
• Safeguards for a curved line search method **CLS** by Neumaier & Azmi [136] because of finite precision arithmetic.
• A new regularized Krylov direction.
• A new limited memory quasi Newton direction.
According to the solver choice discussed in Subsection 6.5.4,
• for low-dimensional problems $1 \leq n \leq 30$, **LMBOPT** was recommended.
• for medium-dimensional problems $31 \leq n \leq 500$, **LMBOPT** was recommended.
• for large-dimensional problems $501 \leq n \leq 100001$, **LMBFG-EIG-MS** was recommended.

Section 7 described an active set trust region method for the bound constrained optimization problem (4.1) with the exact function value and gradient. It was my own work (Kimiaei [114]). This method
• replaced the traditional trust region ratio by a variant of the sufficient descent condition (4.20), useful in finite precision arithmetic and in strongly nonconvex regions;
• used the reduced gradient as a critical measure to get a point which is never a spurious apparent local minimizer arisen because of cancellation in the calculation of a critical measure in double-precision arithmetic;
• updated the trust region radius according to the reduced gradient, resulting in under the positive semi-definiteness of approximated Hessian matrices restricted to the subspace of free variables, unlimited zigzagging could not occur; hence all strongly active variables were found and fixed at finitely many iterations.

Section 8 discussed an efficient randomized algorithm for the unconstrained black box optimization problem with the exact function value and inexact gradient, called **VSBBO**. It was a joint work with Arnold Neumaier (cf. Kimiaei & Neumaier [117]):
• For the basic version of **VSBBO** with only random directions, the complexity bound for the nonconvex case, with probability arbitrarily close to 1, matched the one found by Gratton et al. [84] for another algorithm.
• We also proved the complexity bounds for **VSBBO** for the convex and strongly convex cases, with probability arbitrarily close to 1, essentially matching the bounds found by Bergou et al. [16], only valid in expectation.
• Numerical results showed that the basic version of **VSBBO** (**VSBBO-basic1**) was more efficient than the randomized direct search solvers proposed by Gratton et al. [84] (**DSPFD**) and Bergou et al. [16] (**STP-vs**, **STP-vf**, and **PSTP**) whose complexity results were discussed in Section 8.3.

• An improved version of our algorithm had additional heuristic techniques that did not affect the order of the complexity results and which turned **VSBBO** into an efficient global solver, although our theory guarantees only local minimizers. This version even found in most cases either a global minimizer or, where this could not be checked, at least a point of similar quality with the best competitive global solvers. Other versions of **VSBBO** were **VSBBO-C-Q**, **VSBBO-S**, and **VSBBO-Q**. **VSBBO-C-Q** used random directions, random subspace directions, cumulative direction and additional heuristic techniques but was not better than **VSBBO-basic1**. Although **VSBBO-S** did not use additional heuristic techniques and random subspace directions, it was more efficient than **VSBBO-basic1** and **VSBBO-C-Q** due to using coordinate directions and then random directions. **VSBBO-Q** used additional heuristic techniques and all directions except the L-BFGS direction. It was slightly weaker than **VSBBO** and **VSBBO-S** and was more efficient than **VSBBO-basic1** and **VSBBO-C-Q**. As a consequence, **VSBBO** was the best in comparison with the others versions:
• In terms of the number of solved problems **VSBBO** was for $1 \leq n \leq 20$ more robust than the based-model trust region solvers (**DESTRESS** and **BCDFO**), the direct search solvers (**BFO**, **GCES**, **SDS**, **AHDS**, **DSDS**, and **DSPFD**), the basic deterministic line search solver with only coordinate directions (**SDBOX**), the standard quasi Newton solver (**FMINUNC**), the Nelder–Mead solvers (**FMINSEARCH**, **NELDER**, and **NMSMAX**), the multidirectional search solvers (**MDSMAX** and **MDS**), the Hooke–Jeeves solver (**HOOKE**), the alternating directions solver (**ADSMAX**), the global solvers (**CMAES**, **GLOBAL**, and **ACRS**), and the pattern search solvers (**PSM** and **PSWARM**). The quality of **VSBBO** was improved by increasing the dimension.
• For $21 \leq n \leq 100$, **VSBBO** was the best solvers in the comparison with the best competitive local and global solvers in terms of the number of solved problems and was the second best solvers in terms of the `nf` efficiency.
• For $101 \leq n \leq 1000$, **VSBBO** was the best solver in terms of the number of solved problems and the second best solver in terms of the `nf` efficiency. Finally, for $1001 \leq n \leq 5000$, **VSBBO** was more efficient and robust than **SDBOX** and **FMINUNC** which were the two good solvers for large scale problems.
As a result, when the problem size grew, the performance of **VSBBO** became competitive.

Section 9 introduced an efficient and robust deterministic subspace algorithm (**STBBO**) was introduced for unconstrained box optimization problems (4.2) with the inaccurate function value and gradient in low up to high dimensions. It was a joint work with Arnold Neumaier and Parvaneh Faramarzi (cf. Kimiaei, Neumaier, & Faramarzi [119]). It was able to get a significant decrease in the function value due to the construction of an efficient limited memory direction along which a successful prediction on a decrease in the model function and the model gradient norm might be achieved. In the presence of the inaccurate function value and gradient, the complexity bound for our algorithm found – independent of the choice of search directions enhancing the angle test – matching the order of bound found by Berahas et al. [15]. The numerical results confirmed that
• **STBBO** and **UOBYQA** have the same quality and are the two best solvers for problem in low dimension ($1 \leq n \leq 30$);
• **VSBBO** and **VSBBON** are the two best solvers in terms of the number of solved problems while **STBBO** is the best in terms of the `nf` efficiency for problems with dimensions $31 \leq n \leq 1000$;
• **STBBO** is the best in terms of the number of solved problems and the `nf` efficiency in comparison with **VSBBO** and **VSBBON**;
As a result, the new subspace method is more efficient than the standard quasi Newton and traditional limited memory methods, even it is comparable with the model-based methods.

Section 10 focused on making a randomized model-based line search solver for noisy unconstrained optimization problems, called **VSBBON**. It was my own work (cf. Kimiaei [114]). Two effective techniques which turned **VSBBON** into a competitive solver were
• to construct quadratic models not only for small scale problems but for medium and large scale problems also;
• to find, update, and restart step sizes in a randomized line search algorithm so that its efficiency was increased.
Numerical results confirmed that **VSBBON** worked well with any kind of noise which was not too large but in theory uniform random noise was matched on the assumption (BBO_3).

Section 11 generated a limited memory solver for unconstrained black box least squares problem (4.3), called **LMLS**. It was a joint work with Arnold Neumaier (cf. Kimiaei & Neumaier [115]). **LMLS** was the trust region-based algorithm whose main ingredients were
• using a non-monotone technique and adaptive radius strategy, useful in presence of narrow valley,
• using a Broyden-like algorithm, useful in the cases where trust region radius is small and iteration is unsuccessful,
• using a randomized finite difference approximation in an adaptive subspace for the Jacobian matrix estimation,
• using either a Gauss-Newton or an improved dogleg method to solve the trust region subproblem.
Because of using these enhancements, **LMLS** was competitive for problems in low up to high dimensions in comparison with traditional limited memory BFGS method and even standard BFGS method.

Bibliography

[1] M. Ahookhosh and K. Amini. An efficient nonmonotone trust-region method for unconstrained optimization. *Numer. Algorithms*, 59(4):523–540, September 2011.

[2] M. Ahookhosh, K. Amini, and M. Kimiaei. A globally convergent trust-region method for large-scale symmetric nonlinear systems. *Number. Func. Anal. Opt.*, 36(7):830–855, May 2015.

[3] M. Ahookhosh, H. Esmaeili, and M. Kimiaei. An effective trust-region-based approach for symmetric nonlinear systems. *Int. J. Comput. Math.*, 90(3):671–690, March 2013.

[4] M. Al-Baali and R. Fletcher. An efficient line search for nonlinear least squares. *J. Optim. Theory Appl.*, 48(3):359–377, March 1986.

[5] K. Amini, H. Esmaeili, and M. Kimiaei. A nonmonotone trust-region-approach with nonmonotone adaptive radius for solving nonlinear systems. *IJNAO*, 6(1), February 2016.

[6] K. Amini, M. A. K. Shiker, and M. Kimiaei. A line search trust-region algorithm with nonmonotone adaptive radius for a system of nonlinear equations. *4OR-Q J. Oper. Res.*, 14(2):133–152, January 2016.

[7] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. Optim*, 17(3):642–664, January 2006.

[8] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*. IEEE, 2005.

[9] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM J. Optim*, 24(3):1238–1264, January 2014.

[10] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA J. Numer. Anal.*, 8(1):141–148, 198.

[11] C. J. P. Bélisle, H. E. Romeijn, and R. L. Smith. Hit-and-run algorithms for generating multivariate distributions. *Math. Oper. Res.*, 18(2):255–266, May 1993.

[12] S. Bellavia, M. Macconi, and B. Morini. STRSCNE: A scaled trust-region solver for constrained nonlinear equations. *Comput. Optim. Appl.*, 28(1):31–50, April 2004.

[13] S. Bellavia, M. Macconi, and S. Pieraccini. Constrained dogleg methods for nonlinear systems with simple bounds. *Comput. Optim. Appl.*, 53(3):771–794, March 2012.

[14] A. S. Berahas, R. H. Byrd, and J. Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM J. Optim.*, 29(2):965–993, January 2019.

[15] A. S. Berahas, L. Cao, K. Choromanski, and K. Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization, 2020.

[16] E. H. Bergou, E. Gorbunov, and P. Richtárik. Stochastic three points method for unconstrained smooth minimization. *CoRR*, abs/1902.03591, 2019.

[17] D. P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM J. Control Opim.*, 20(2):221–246, 1982.

[18] E. G. Birgin, I. Chambouleyron, and J. M. Martínez. Estimation of the optical constants and thickness of thin films using unconstrained optimization. *J. Comput. Phys.*, 151:862–880, 1999.

[19] E. G. Birgin, J. L. Gardenghi, J. M. Martínez, S. A. Santos, and Ph. L. Toint. Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. *Math. Program.*, 163(1-2):359–368, August 2016.

[20] E. G. Birgin and J. M. Martínez. A box-constrained optimization algorithm with negative curvature directions and spectral projected gradients. In G. Alefeld and X. Chen, editors, *Topics in Numerical Analysis*, volume 15 of *Computing Supplementa*, pages 49–60. Springer Vienna, 2001.

[21] E. G. Birgin and J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Comput. Optim. Appl.*, 23(1):101–125, 2002.

[22] E. G. Birgin and J. M. Martínez. On the application of an augmented lagrangian algorithm to some portfolio problems. *EURO J. Comput. Optim.*, 4(1):79–92, October 2015.

[23] E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. Optim.*, 10(4):1196–1211, 1999.

[24] E. G. Birgin, J. M. Martínez, and M. Raydan. Algorithm 813: Spg-software for convex-constrained optimization. *ACM Trans. Math. Softw.*, 27(3):340–349, 2001.

[25] E. G. Birgin, J. M. Martínez, and M. Raydan. Inexact spectral projected gradient methods on convex sets. *IMA J. Numer. Anal.*, 23:539–559, 2003.

[26] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and et al. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 273:41–58, 2016.

[27] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, and S. Lucidi. A new version of the price's algorithm for global optimization. *J. Glob. Optim.*, 10(2):165–184, 1997.

[28] O. Burdakov, L. Gong, S. Zikrin, and Y. Yuan. On efficiently combining limited-memory and trust-region techniques. *Math. Program. Comput.*, 9:101–134, 2017.

[29] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190, 1995.

[30] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Math. Program.*, 63(1-3):129–156, 1994.

[31] P. Calamai and J. Moré. Projected gradient methods for linearly constrained problems. *Math. Program.*, 39(1):93–116, 1987.

[32] R. N. Calheiros, R. B. César, and A. F. De Rose. Building an automated and self–configurable emulation testbed for grid applications. *software: practice and experience*, 40:405–429, 2010.

[33] C. Cartis, N. I. M. Gould, and Ph. L Toint. Evaluation complexity bounds for smooth constrained nonlinear optimisation using scaled kkt conditions, high-order models and the criticality measure χ, 2017.

[34] C. Cartis, N. I. M. Gould, and Ph. L. Toint. Evaluation complexity bounds for smooth constrained nonlinear optimization using scaled KKT conditions and high-order models. In *Approximation and Optimization*, pages 5–26. Springer International Publishing, 2019.

[35] C. Cartis, Ph. R. Sampaio, and Ph. L. Toint. Worst-case evaluation complexity of non-monotone gradient-related algorithms for unconstrained optimization. *Optimization*, 64(5):1349–1361, January 2014.

[36] M. L. Cauwet, J. Liu, R. Baptiste, and O. Teytaud. Algorithm portfolios for noisy optimization. *Annals of Mathematics and Artificial Intelligence*, 76:143–172, 2016.

[37] R. Chen. *Stochastic Derivative-Free Optimization of Noisy Functions*. PhD thesis, Lehigh University, 2015. Theses and Dissertations. 2548.

[38] J. Konečný and P. Richtárik. Simple complexity analysis of simplified direct search. *CoRR*, abs/1410.0390, 2014.

[39] 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 J. Numer. Anal.*, 25(2):433, 1988.

[40] 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(182):399–430, 1988.

[41] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics, January 2000.

[42] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, January 2009.

[43] A. Cristofari, M. De Santis, S. Lucidi, and F. Rinaldi. A two-stage active-set algorithm for bound-constrained optimization. *J. Optim. Theory Appl.*, 172:369–401, 2017.

[44] T. Csendes, L. Pál, J. O. H. Sendín, and J. R. Banga. The GLOBAL optimization method revisited. *Optim. Lett.*, 2(4):445–454, November 2007.

[45] F. E. Curtis, Z. Lubberts, and D. P. Robinson. Concise complexity analyses for trust region methods. *Optim. Lett.*, 12(8):1713–1724, June 2018.

[46] F. E. Curtis, D. P. Robinson, and M. Samadi. A trust region algorithm with a worst-case iteration complexity of $\mathcal{O}(\epsilon^{-3/2})$ for nonconvex optimization. *Math. Program.*, 162(1-2):1–32, May 2016.

[47] A. L. Custódio, H. Rocha, and L. N. Vicente. Incorporating minimum frobenius norm models in direct search. *Comput. Optim. Appl.*, 46(2):265–278, August 2009.

[48] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM J. Optim*, 18(2):537–555, 2007.

[49] Y. H. Dai. On the nonmonotone line search. *J. Optim. Theory Appl.*, 112(2):315–330, 2002.

[50] Y. H. Dai and R. Fletcher. Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming. *Numer. Math.*, 100(1):21–47, 2005.

[51] Y. H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Math. Program.*, 106(3):403–421, 2006.

[52] Y. H. Dai, W. W. Hager, K. Schittkowski, and H. Zhang. The cyclic Barzilai–Borwein method for unconstrained optimization. *IMA J. Numer. Anal.*, 26(3):604–627, 2006.

[53] C. Davis. Theory of positive linear dependence. *Amer. J. Math.*, 76(4):733, October 1954.

[54] R. S. Dembo and U. Tulowitzki. On the minimization of quadratic functions subject to box constraints. Technical report, School of Organization and Management, Yale University, New Haven, CT, 1983.

[55] N. Y. Deng, Y. Xiao, and F. J. Zhou. Nonmonotonic trust region algorithm. *J. Optim. Theory Appl.*, 76(2):259–285, February 1993.

[56] J. E. Dennis and L. N. Vicente. Trust-region interior-point algorithms for minimization problems with simple bounds. In *Applied Mathematics and Parallel Computing*, pages 97–107. Physica-Verlag HD, 1996.

[57] P. Deuflhard. *Newton Methods for Nonlinear Problems*. Springer Berlin Heidelberg, 2011.

[58] P. Deuflhard and G. Heindl. Affine invariant convergence theorems for newton's method and extensions to related methods. *SIAM J. Numer. Anal.*, 16(1):1–10, February 1979.

[59] M. Dodangeh and L. N. Vicente. Worst case complexity of direct search under convexity. *Math. Program.*, 155(1-2):307–332, November 2014.

[60] M. Dodangeh, L. N. Vicente, and Z. Zhang. On the optimal order of worst case complexity of direct search. *Optim. Lett.*, 10(4):699–708, June 2015.

[61] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, January 2002.

[62] Z. Dostál. Box constrained quadratic programming with proportioning and projections. *SIAM J. Optim*, 7(3):871–887, 1997.

[63] Z. Dostál. A proportioning based algorithm with rate of convergence for bound constrained quadratic programming. *Numer. Algorithms*, 34(2-4):293–302, 2003.

[64] J. C. Dunn. On the convergence of projected gradient processes to singular critical points. *J. Optim. Theory Appl.*, 55:203–216, 1987.

[65] C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA J. Numer. Anal.*, 15(4):585–608, 1995.

[66] C. Elster and A. Neumaier. A method of trust region type for minimizing noisy functions. *Computing*, 58(1):31–46, March 1997.

[67] H. Esmaeili and M. Kimiaei. An efficient adaptive trust-region method for systems of nonlinear equations. *Int. J. Comput. Math.*, 92(1):151–166, April 2014.

[68] H. Esmaeili and M. Kimiaei. A new adaptive trust-region method for system of nonlinear equations. *Appl. Math. Model.*, 38(11-12):3003–3015, June 2014.

[69] H. Esmaeili and M. Kimiaei. A trust-region method with improved adaptive radius for systems of nonlinear equations. *Math. Meth. Oper. Res.*, 83(1):109–125, November 2015.

[70] Yu. G. Evtushenko. Numerical methods for finding global extrema (case of a non-uniform mesh). *USSR USSR Comput. Math. & Math. Phys.*, 11(6):38–54, January 1971.

[71] J. Fan. Convergence rate of the trust region method for nonlinear equations under local error bound condition. *Comput. Optim. Appl.*, 34(2):215–227, March 2006.

[72] J. Fan and J. Pan. An improved trust region algorithm for nonlinear equations. *Comput. Optim. Appl.*, 48(1):59–70, February 2009.

[73] J. Fan and J. Pan. A modified trust region algorithm for nonlinear equations with new updating rule of trust region radius. *Int. J. Comput. Math.*, 87(14):3186–3195, October 2010.

[74] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Ltd, May 2000.

[75] R. Fletcher. On the Barzilai-Borwein method. *Optimization and Control with Applications*, pages 235–256, 2005.

[76] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer J,*, 7(2):149–154, February 1964.

[77] W. Glunt, T. L. Hayden, and M. Raydan. Molecular conformations from distance matrices. *J. Comput. Chem.*, 14(1):114–120, 1993.

[78] A. Goldstein and J. Price. An effective algorithm for minimization. *Numer. Math.*, 10:184–189, 1967.

[79] 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 Trans. Math. Softw. (TOMS)*, 29(4):353–372, December 2003.

[80] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.*, 60:545–557, 2015.

[81] G. N. Grapiglia, J. Yuan, and Y. Yuan. Nonlinear stepsize control algorithms: Complexity bounds for first- and second order optimality. *J. Optim. Theory Appl.*, 171(3):980–997, September 2016.

[82] S. Gratton, C. W. Royer, and L. N. Vicente. A second-order globally convergent direct-search method and its worst-case complexity. *Optimization*, 65(6):1105–1128, December 2015.

[83] S. Gratton, C. W. Royer, and L. N. Vicente. A decoupled first/second-order steps technique for nonconvex nonlinear unconstrained optimization with improved complexity bounds. *Math. Program.*, 179(1-2):195–222, September 2018.

[84] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM J. Optim*, 25(3):1515–1541, January 2015.

[85] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.*, 26(4-5):873–894, October 2011.

[86] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for newton's method. *SIAM J. Numer. Anal.*, 23(4):707–716, 1986.

[87] L. Grippo and M. Sciandrone. Nonmonotone globalization techniques for the Barzilai-Borwein gradient method. *Comput. Optim. Appl.*, 23(2):143–169, 2002.

[88] L. Grippo and M. Sciandrone. Nonmonotone derivative-free methods for nonlinear equations. *Comput. Optim. Appl.*, 37(3):297–328, March 2007.

[89] W. W. Hager and H. Zhang. CG_DESCENT user's guide. Technical report, Department of Mathematics, University of Florida, Gainesville, FL, 2004.

[90] W. W. Hager and H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM J. Optim.*, 16(1):170–192, 2005.

[91] W. W. Hager and H. Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Trans. Math. Softw.*, 32(1):113–137, 2006.

[92] W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM J. Optim.*, 17(2):526–557, 2006.

[93] W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pac. J. Optim.*, 2(1):35–58, 2006.

[94] W. W. Hager and H. Zhang. The limited memory conjugate gradient method. *SIAM J. Optim.*, 23(4):2150–2168, 2013.

[95] R. W. Hamming. *Introduction to Applied Numerical Analysis*. Taylor & Francis/Hemisphere, USA, 1989.

[96] N. Hansen. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, pages 75–102. Springer Berlin Heidelberg, 2006.

[97] Y. He, S.Y. Yuen, Y. Lou, and X. Zhang. A sequential algorithm portfolio approach for black box optimization. *Swarm and Evolutionary Computation*, pages 559–570, 2018.

[98] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[99] N. J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(2):317–333, April 1993.

[100] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Optim*, 2(2):88–105, June 1973.

[101] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In A. Lodi, M. Milano, and P. Toth (eds), editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science,*, Berlin, Heidelberg, 2010. Springer.

[102] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *J. Glob. Optim.*, 14(4):331–355, 1999.

[103] W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.*, 35(2):1–25, July 2008.

[104] W. Huyer and A. Neumaier. MINQ8: general definite and bound constrained indefinite quadratic programming. *Comput. Optim. Appl.*, 69(2):351–381, October 2017.

[105] Jr. J. E. Dennis and J. J. Moré. Quasi-newton methods, motivation and theory. *SIAM Rev.*, 19(1):46–89, January 1977.

[106] Jr. J. E. Dennis and H. F. Walker. Convergence theorems for least-change secant update methods. *SIAM J. Numer. Anal.*, 18(6):949–987, December 1981.

[107] C. Kanzow and A. Klug. On affine-scaling interior-point newton methods for nonlinear minimization with bound constraints. *Comput. Optim. Appl.*, 35(2):177–197, June 2006.

[108] C. Kanzow and A. Klug. An interior-point affine-scaling trust-region method for semismooth equations with box constraints. *Comput. Optim. Appl.*, 37(3):329–353, March 2007.

[109] C. T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, January 1999.

[110] P. Kerschke, H.H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 1(27):3–45, 2009.

[111] M. Kimiaei. An active set trust region method for bound constrained optimization.

[112] M. Kimiaei. A new class of nonmonotone adaptive trust-region methods for nonlinear equations with box constraints. *Calcolo*, 54(3):769–812, October 2016.

[113] M. Kimiaei. Nonmonotone self-adaptive levenberg–marquardt approach for solving systems of nonlinear equations. *Number. Func. Anal. Opt.*, 39(1):47–66, July 2017.

[114] M. Kimiaei. Line search in noisy unconstrained black box optimization. `http://www.optimization-online.org/DB_HTML/2020/09/8007.html`, Sep 2020.

[115] M. Kimiaei and A. Neumaier. A new limited memory method for unconstrained nonlinear least squares.

[116] M. Kimiaei and A. Neumaier. Testing and tuning optimization algorithm. Preprint, Vienna University, Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria, 2019.

[117] M. Kimiaei and A. Neumaier. Efficient global unconstrained black box optimization. `http://www.optimization-online.org/DB_HTML/2018/08/6783.html`, Jul 2020.

[118] M. Kimiaei, A. Neumaier, and B. Azmi. LMBOPT – a limited memory method for bound-constrained optimization. `http://www.optimization-online.org/DB_HTML/2020/11/8089.html`, Nov 2020.

[119] M. Kimiaei, A. Neumaier, and P. Faramarzi. New subspace method for unconstrained black box optimization.

[120] D. E. Kvasov and Y. D. Sergeyev. Lipschitz gradients for global optimization in a one-point-based partitioning scheme. *J. Comput. Appl. Math.*, 236(16):4042–4054, October 2012.

[121] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM J. Optim*, 9(1):112–147, January 1998.

[122] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numer.*, 28:287–404, May 2019.

[123] M. Lindauer, J. N. Rijn, and L. Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

[124] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1-3):503–528, August 1989.

[125] W. Liu and Y. H. Dai. Minimization algorithms based on supervisor and searcher cooperation. *J. Optim. Theory Appl.*, 111(2):359–379, 2001.

[126] S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Comput. Optim. Appl.*, 21(2):119–142, 2002.

[127] L. Lukšan, C. Matonoha, and J. Vlček. Problems for nonlinear least squares and nonlinear equations. Technical Report V-1259, ICS CAS, 2018.

[128] Y. Malitsky. *Instance-Specific Algorithm Configuration 2014th Edition, Kindle Edition.* Springer, Cham, 2014.

[129] J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numer. Math.*, 55(4):377–400, 1989.

[130] J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optim.*, 1(1):93–113, 1991.

[131] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20(1):172–191, January 2009.

[132] L. Nazareth. A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms. *SIAM J. Numer. Anal.*, 16(5):794–800, October 1979.

[133] Y. Nesterov. *Introductory Lectures on Convex Optimization*. Springer US, 2004.

[134] Y. Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Math. Program.*, 108(1):177–205, April 2006.

[135] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Found. Comput. Math.*, 17(2):527–566, November 2015.

[136] A. Neumaier and B. Azmi. Line search and convergence in bound-constrained optimization. `http://www.optimization-online.org/DB_FILE/2019/03/7138.pdf`, 2019.

[137] A. Neumaier, H. Fendl, H. Schilly, and Thomas Leitner. VXQR: derivative-free unconstrained optimization based on QR factorizations. *Soft Comput.*, 15(11):2287–2298, September 2010.

[138] H. B. Nielsen. immoptibox – a matlab toolbox for optimization and data fitting, 2012. version 2.2.

[139] J. Nocedal. Theory of algorithms for unconstrained optimization. *Acta Numer.*, 1:199–242, January 1992.

[140] J. Nocedal and S. J. Wright, editors. *Numerical Optimization*. Springer-Verlag, 1999.

[141] U. Nowak and L. Weimann. A family of newton codes for systems of highly nonlinear equations. Technical Report TR 91–10, Zuse Institute Berlin (ZIB), 1990.

[142] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, January 2000.

[143] I. Pinelis. A probabilistic angle inequality. MathOverflow.

[144] J. Pintér. Globally convergent methods for n-dimensional multiextremal optimization. *Optimization*, 17(2):187–202, January 1986.

[145] B. T. Polyak. The conjugate gradient method in extremal problems. *USSR Comput. Math. Math. Phys.*, 9:94–112, 1969.

[146] Boris T Polyak. Introduction to optimization. 1987. *Optimization Software, Inc, New York*.

[147] M. Porcelli and P. Toint. Global and local information in structured derivative free optimization with BFO. *arXiv: Optimization and Control*, 2020.

[148] M. Porcelli and Ph. L. Toint. A note on using performance and data profiles for training algorithms. *ACM Transactions on Mathematical Software*, 45(2):1–10, June 2019.

[149] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.*, 92(3):555–582, May 2002.

[150] M. Raydan. The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM J. Optim.*, 7(1):26–33, 1997.

[151] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Global. Optim.*, 56(3):1247–1293, July 2012.

[152] C. Royer. *Derivative-free optimization methods based on probabilistic and deterministic properties : complexity analysis and numerical relevance*. Theses, Université Paul Sabatier - Toulouse III, November 2016.

[153] R. B. Schnabel. Sequential and parallel methods for unconstrained optimization. In M. Iri and eds. K. Tanabe, editors, *In Mathematical Programming, Recent Developments and Applications*, pages 227–261. Kluwer Academic Publishers, 1989.

[154] T. Serafini, G. Zanghirati, and L. Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optim. Methods Softw.*, 20(2):353–378, 2005.

[155] L. Sorber, M. V. Barel, and L. D. Lathauwer. Unconstrained optimization of real functions in complex variables. *SIAM J. Optim.*, 22(3):879–898, January 2012.

[156] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[157] Ph. L. Toint. An assessment of nonmonotone linesearch techniques for unconstrained optimization. *SIAM J. Sci. Comput.*, 17(3):725–739, 1996.

[158] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Springer Netherlands, 1987.

[159] A. I. F. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*, 39(2):197–219, February 2007.

[160] D. Vermetten, H. Wang, C. Doerr, and T. Bäck. Towards dynamic algorithm selection for numerical black-box optimization: Investigating bbob as a use case, Jun 2020.

[161] L. N. Vicente. Worst case complexity of direct search. *EURO J. Comput. Optim.*, 1(1-2):143–153, December 2012.

[162] P. Wolfe. Convergence conditions for ascent methods. *SIAM Rev.*, 11:226–235, 1969.

[163] Y. Xie, R. H. Byrd, and J. Nocedal. Analysis of the BFGS method with errors. *SIAM J. Optim.*, 30(1):182–209, January 2020.

[164] E. K. Yang and J. W. Tolle. A class of methods for solving large, convex quadratic programs subject to box constraints. *Math. Program.*, 51:223–228, 1991.

[165] Z. Yu and D. Pu. A new nonmonotone line search technique for unconstrained optimization. *J. Comput. Appl. Math.*, 219(1):134–144, September 2008.

[166] Y. Yuan. Recent advances in numerical methods for nonlinear equations and nonlinear least squares. *Numer. Algebra, Control. Optim.*, 1(1):15–34, 2011.

[167] H. Zhang and W. W. Hager. A nonmonotone line search technique and its application to unconstrained optimization. *SIAM J. Optim.*, 14(4):1043–1056, 2004.

[168] J. Zhang and C. Xu. A class of indefinite dogleg path methods for unconstrained minimization. *SIAM J. Optim.*, 9(3):646–667, 1999.

List of Figures

List of Tables

CURRICULUM VITAE

Morteza Kimiaei

PERSONAL INFORMATION

Birth	December 25, 1983, Tehran, Iran
Cell	+918 714 16 43
Email	morteza.kimiaei@gmail.com, kimiaeim83@univie.ac.at
Homepage	https://www.mat.univie.ac.at/∼kimiaei
Google scholar	https://scholar.google.at/citations?user=79302hwAAAAJ&hl=en
VGSCO page	https://vgsco.univie.ac.at/people/phd-students/morteza-kimiaei/

EDUCATION & CERTIFICATION

2017-2021 **PhD Student in Computational Optimization**
Vienna Graduate School on Computational Optimization (VGSCO)

2006-2008 **Master of Science in Applied Mathematics (Optimization)**
Department of Mathematics, Razi University, Kermanshah, Iran

2002-2006 **Bachelor of Science in pure Mathematics**
Department of Mathematics, Bu-Ali Sina University, Hamedan, Iran

1998-2001 **Diploma in Mathematics and Physics**
Engelab High School, Asadabad, Iran

RESEARCH INTERESTS

- *Nonsmooth Optimization*
- *Convex Optimization*
- *Nonlinear Optimization*
- *Global Optimization*

- *Computational Mathematics*
- *Large-Scale Structured Optimization*
- *Black box optimization*

PUBLICATIONS

- PUBLISHED:

1. **M. Kimiaei**, F. Rahpaymaii, A fixed point method for convex systems, *Applied Mathematics*, 3 (2012), 1327–1333.
2. M. Ahookhosh, H. Esmaeili, **M. Kimiaei**, An effective trust-region-based approach for symmetric nonlinear systems, *International Journal of Computer Mathematics* (Taylor & Francis), 90 (3) (2013), 671–690.
3. H. Esmaeili, **M. Kimiaei**, An improved adaptive trust-region method for unconstrained optimization, *Mathematical Modelling and Analysis* (Taylor & Francis), 19(4) (2014), 469–490.
4. H. Esmaeili, **M. Kimiaei**, A new adaptive trust-region method for systems of nonlinear equations, *Applied Mathematical Modelling* (Elsevier), 38(11–12) (2014), 3003–3015.
5. H. Esmaeili, **M. Kimiaei**, An efficient implementation of a trust region method for box constrained optimization, *Journal of Applied Mathematics and Computing* (Springer), 48 (2015), 495–517.
6. M. Ahookhosh, K. Amini, **M. Kimiaei**, A globally convergent trust-region method for large-scale symmetric nonlinear systems, *Numerical Functional Analysis and Optimization* (Taylor & Francis), 36 (2015), 830–855.
7. H. Esmaeili, **M. Kimiaei**, An efficient adaptive trust-region method for systems of nonlinear equations, *International Journal of Computer Mathematics* (Taylor & Francis), 92(1) (2015), 151–166.
8. M. Ahookhosh, K. Amini, **M. Kimiaei**, M.R. Peyghami, A limited memory trust-region method with adaptive radius for large-scale unconstrained optimization, *Bulletin of the Iranian Mathematical Society* (Iranian Mathematical Society), 42(4) (2015), 819–837.
9. K. Amini, **M. Kimiaei**, M.A.K. Shiker, A line search trust-region algorithm with nonmonotone adaptive radius for solving systems of nonlinear equations, *4OR* (Springer), 14(2) (2016), 133–152.
10. K. Amini, H. Esmaeili, **M. Kimiaei**, A nonmonotone trust-region-approach with nonmonotone adaptive radius for nonlinear systems, *Iranian Journal of Numerical Analysis and Optimization*, 6(1) (2016), 101–121.
11. F. Rahpeymaii, **M. Kimiaei**, A. Bagheri, A limited memory quasi-Newton trust-region method for box constrained optimization, *Computational and Applied Mathematics* (Elsevier), 303 (2016), 105–118.
12. H. Esmaeili, **M. Kimiaei**, A trust-region method with improved adaptive radius for systems of nonlinear equations, *Mathematical Methods of Operations Research* (Springer), 83 (2016), 109–105.
13. **M. Kimiaei**, H. Esmaeili, A trust-region approach with novel filter adaptive radius for systems of nonlinear equations, *Numerical Algorithms* (Springer), 73(4) (2016), 999–1016.
14. **M. Kimiaei**, A new class of nonmonotone adaptive trust-region method for nonlinear equations with box constrained, *Calcolo* (Springer), 54(3) (2017), 769–812.
15. **M. Kimiaei**, M. Rostami, Impulse noise removal based on new hybrid spectral conjugate gradient approach, *KYBERNETIKA* (The Czech Academy of Sciences), 52(5) (2016), 791–823.

16. **M. Kimiaei**, Nonmonotone self-adaptive levenberg-marquardt approach for solving systems of nonlinear equations, *Numerical Functional Analysis and Optimization* (Taylor & Francis), 39(1) (2018), 47–66.
17. H. Esmaeili, M. Rostami, **M. Kimiaei**, Combining line search and trust-region methods for ℓ_1-minimization, *International Journal of Computer Mathematics* (Taylor & Francis), 95(10) (2018), 1950–1972.
18. **M. Kimiaei**, S. Ghaderi, A new restarting adaptive trust-region method for unconstrained optimization, *Journal of the Operations Research Society of China* (Springer), 5(4) (2017), 487–507.
19. F. Rahpeymaii, **M. Kimiaei**, A Barzilai Borwein Adaptive Trust-Region Method for Solving Systems of Nonlinear Equation, *International Journal of Research in Industrial Engineering*, 6(4) (2017), 339–349.
20. K. Amini, **M. Kimiaei**, H. Khotanlou, A nonmonotone pattern search approach for systems of nonlinear equations, *International Journal of Computer Mathematics* (Taylor & Francis), 96(1) (2019), 33–50.
21. H. Esmaeili, S. shaebani, **M. Kimiaei**, A new conjugate gradient methods for compressive sensing problems, *Calcolo* (Springer), 56 (1) (2019).
22. **M. Kimiaei**, F. Rahpeymaii, A new nonmonotone line search adaptive trust region for nonlinear systems, *TOP* (Springer), 27(2) (2019), 192–232.
23. **M. Kimiaei**, F. Rahpeymaii, Impulse noise removal by an adaptive trust-region method, *Soft Computing* (Springer), 23 (2019), 11901–11923.
24. **M. Kimiaei**, H. Esmaeili, F. Rahpeymaii, A Trust-region Method using Extended Nonmonotone Technique for Unconstrained Optimization, *Iranian Journal of Mathematical Sciences and Informatics* (ACECR at Tarbiat Modares University, ISI) (2020).

- Under review:
  1. **M. Kimiaei**, A. Neumaier, B. Azmi, LMBOPT - a limited memory method for bound-constrained optimization, revised (2020).
  2. **M. Kimiaei**, A. Neumaier, VSBBO – Efficient unconstrained black box optimization, revised (2020).
  3. **M. Kimiaei**, VSBBON – Line search in noisy unconstrained black box optimization, submitted (2021).
  4. **M. Kimiaei**, An active set trust region method for bound constrained optimization, submitted (2021).
  5. **M. Kimiaei**, A. Neumaier, LMLS – Limited memory for unconstrained nonlinear least squares, submitted (2020).
  6. **M. Kimiaei**, A. Neumaier, P. Faramarzi, STBBO – A new subspace tecnique for unconstrained black box optimization, submitted (2021).

- Work in progress:
  1. **M. Kimiaei**, A. Neumaier, Testing and tuning optimization algorithm, (2020).

Softwares

- LMBOPT: *A limited memory for bound constrained optimization in Matlab*
- VSBBO: *An efficient stochastic algorithm unconstrained black box optimization in Matlab*
- VSBBON: *An efficient stochastic algorithm for unconstrained noisy black box optimization in Matlab*

– LMLS: *A limited memory method for unconstrained least squares in Matlab*
– STBBO: *A subspace technique for unconstrained black box optimization in Matlab*
– GSCG: *A new generalized shrinkage conjugate gradient method for sparse recovery in Matlab*

TEACHING

2008-2016 Razi University, Department of Mathematics, Teacher Math
Operation Research, Calculus and Analytic Geometry, Differential Equation

2009-2016 Azad Islamic University, Asadabad, Teacher Math
Operation Research, Calculus and Analytic Geometry, Differential Equation

2009-2016 Payame Noor University, Asadabad, Teacher Math
Operation Research, Calculus and Analytic Geometry, Differential Equation

2012-2016 Seyyed Jamaleddin Asadabadi University, Teacher Math
Operation Research, Calculus and Analytic Geometry, Differential Equation

SELECTED SKILLS

– Computer Skills:

Programming Language	C, C++
Mathematical Software	MATLAB
Operating System	Linux, Windows
Applied Software	Latex, Office

– Language Skills:

Persian	Native Language
English	Advanced Knowledge

EXTRACURRICULAR ACTIVITIES

– Sport: football, Hiking, Climbing, Fishing
– Reading books, Watching historical movies.