MPC manuscript No.
(will be inserted by the editor)

LMBOPT – a limited memory method for bound-constrained optimization

Morteza Kimiaei[1] · **Arnold Neumaier** · **Behzad Azmi**

the date of receipt and acceptance should be inserted later

Abstract Recently, Neumaier and Azmi gave a comprehensive convergence theory for a generic algorithm for bound constrained optimization problems with a continuously differentiable objective function. The algorithm combines an active set strategy with a gradient-free line search **CLS** along a piecewise linear search path defined by directions chosen to reduce zigzagging.

This paper describes **LMBOPT**, an efficient implementation of this scheme. It employs new limited memory techniques for computing the search directions, improves **CLS** by adding various safeguards relevant when finite precision arithmetic is used, and adds many practical enhancements in other details.

The paper compares **LMBOPT** and many other solvers on the unconstrained and bound constrained problems from the `CUTEst` collection and makes recommendations on which solver to use and when. Depending on the problem class, the problem dimension, and the precise goal, the best solvers are **LMBOPT**, **ASACG**, and **LMBFG-EIG-MS**.

1. The first author acknowledges the financial support of the Doctoral Program *Vienna Graduate School on Computational Optimization* (*VGSCO*) funded by the Austrian Science Foundation under Project No W1260-N35

M. Kimiaei
Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
E-mail: kimiaeim83@univie.ac.at
`WWW:http://www.mat.univie.ac.at/~kimiaei/`

A. Neumaier
Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
`WWW:http://www.mat.univie.ac.at/~neum/`
E-mail: Arnold.Neumaier@univie.ac.at

B. Azmi
Johann Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences, Altenbergerstraße 69, A-4040 Linz, Austria
E-mail: behzad.azmi@ricam.oeaw.ac.at

Keywords bound constrained optimization · exact gradient · limited memory technique · robust line search method

2000 AMS Subject Classification: primary 90C30, 90C06, 65K05.

4, November

1 Introduction

The bound constrained optimization problem (**BCOPT**) is the task of minimizing a function subject to a feasible region defined by simple bounds on the variables. In this paper we describe the implementation and numerical evaluation of a new active set method for solving the bound constrained optimization problem

$$\begin{array}{l} \min\ f(x) \\ \text{s.t.}\ \ x \in \mathbb{R}^n, \quad \underline{x} \le x \le \overline{x}, \end{array} \tag{1}$$

where $\mathbf{x} = [\underline{x}, \overline{x}]$ is a bounded or unbounded box in $\mathbb{R}^n$ describing the bounds on the variables and the **objective function** $f : \mathbf{x} \to \mathbb{R}$ is continuously differentiable with **gradient**

$$g(x) := \partial f(x)/\partial x \in \mathbb{R}^n.$$

Problems with naturally given bounds appear in a wide range of applications including the optimal design problem [4], contact and friction in rigid body mechanics [50], the obstacle problem [53], journal bearing lubrication and flow through a porous medium [48]. Often variables of an optimization problem can only be considered meaningful within a particular interval [31]. Some approaches [1] reduce the solution of variational inequalities and complementarity problems to bound constrained problems. The bound constrained optimization problem also arises as an important subproblem in algorithms for solving general constrained optimization problems based on augmented Lagrangians and penalty methods. Numerous research papers [17, 27, 39, 38, 51] deal with the development of efficient numerical algorithms for solving bound constrained optimization problems, especially when the number of variables is large.

1.1 Past work

In the last few years, many algorithms have been developed for solving the **BCOPT** problem (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 classical reference for active set methods for bound constrained problems with a convex quadratic objective function (**QBOPT**) is the projected conjugate gradient method of Polyak [55], 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 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 [23] introduced methods for **QBOPT** in 1983 that are able to add and drop many constraints at each iteration. Their basic idea was further developed by Yang & Tolle [60] into an algorithm guaranteed to identify the face containing a local solution of **QBOPT** in finitely many iterations, even when the solution of the problem is degenerate. For further research on **QBOPT** we refer the reader to [25, 26, 52, 53].

For **BCOPT** with a general nonlinear objective function, Bertsekas [3] 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 the face containing a local solution very quickly. Further research on convergence and properties of projected gradient methods can be found in [3, 14, 28]. The idea of using gradient projections for identifying optimal active constraints was followed up by many researchers. Most of them [12, 16, 15] combined Newton type methods with the gradient projection method to speed up the convergence. For example, **LBFGSB**, developed by Byrd et al. [12], performs the gradient projection method by computing the Cauchy point to determine the active variables. After determining the set of active variables, the algorithm performs line searches along the search directions obtained by a **limited memory BFGS method** [13] 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. [16, 15].

A **non-monotone line search** was first introduced by Grippo, Lampariello & Lucidi (**GLL**) in [36] for Newton methods, to improve the ability to follow a curved valley with steep walls. Later several works [19, 22, 29, 37, 58, 61] on non-monotone line search methods pointed out that these methods are more efficient than monotone line search methods in many cases. Other papers [4, 8, 20, 21, 32, 49, 57] indicate that gradient projection approaches based on a **Barzilai–Borwein step size** [2] have impressive performance in a wide range of applications. Recent works [5, 6, 8, 9, 10, 56] on Barzilai–Borwein gradient projection methods (**BBGP**) have modified them by incorporating them with the **GLL** non-monotone line search: For instance, Raydan [56] developed the **BBGP** method for solving unconstrained optimization problems, and Dai & Fletcher [20, 21] proposed **BBGP** methods for large-scale bound constrained quadratic programming. The idea of Raydan [56] was further developed to generate a convex constrained solver (**SPG**) by Birgin et al. [8, 9] and a bound constrained solver (**GENCAN**) by Birgin et al. [5, 6], enriched by an active set strategy.

The **GALAHAD** package [34] uses as bound-constrained solver **LANCELOT-B**, a trust-region algorithm using truncated Newton directions. Recently, Burdakov et al. [11] constructed a family of limited memory quasi Newton methods for un-

constrained optimization combined with line searches or trust regions, called the **LMBFG** package.

To deal with negative curvature regions, Birgin & Martínez [5] used the second-order trust region algorithm of Zhang & Xu [62], and Birgin & Martínez [6] designed a new algorithm whose line search iteration is performed by means of backtracking and extrapolation. Hager & Zhang [43] 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 the cyclic Barzilai–Borwein method [22], and an unconstrained step that utilizes their developed conjugate gradient algorithms [40, 41, 42, 44]. **ASACG** version 3.0 was 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 it could improve performance by giving the code additional flexibility in the starting step size routine. In 2017, Cristofari et al. [18] 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 non-active variables.

A considerable amount of literature has been published on line search algorithms, which enforce the Wolfe conditions (Wolfe [59]) or the Goldstein conditions (Goldstein [33]). One problem with line search algorithms satisfying the Wolfe conditions is the need to **calculate a gradient at each trial point**. On the other hand, line search algorithms based on the Goldstein conditions are gradient-free, but they have **very poor behaviour in severely nonconvex regions**. Neumaier & Azmi [54] introduced a new active set method **BOPT** (Algorithm 9.1 in [54] = Algorithm 1) using an efficient gradient-free curved line search **CLS** (Algorithm 3.3 in [54] = Algorithm 2). 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 and achieves a reasonable reduction of the objective function.

This paper 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 conjugate gradient direction**. In addition, many other practical enhancements are used, including a **new limited memory method**. A **solver choice** is based on our findings from an extensive numerical results. It depends on the problem dimension, the presence or absence of constraints, the desired robustness, and the relative costs of the function and gradient evaluations.

1.2 **BOPT** – an active set method for bound constrained optimization

Recently, Neumaier & Azmi [54] gave a comprehensive convergence theory for a generic algorithm for the bound constrained optimization problem (1) with a continuously differentiable objective function, called **BOPT**. The **reduced gradient** $g^{\mathrm{red}}(x)$ at a point x, whose components are

$$g^{\mathrm{red}}(x)_i := \begin{cases} 0 & \text{if } x_i = \underline{x}_i = \overline{x}_i, \\ \min(0, g_i) & \text{if } x_i = \underline{x}_i < \overline{x}_i, \\ \max(0, g_i) & \text{if } x_i = \overline{x}_i > \underline{x}_i, \\ g_i & \text{otherwise,} \end{cases} \tag{2}$$

(where $g_i := g_i(x)$ is the ith component of gradient vector at x) vanishes at a local minimizer. At each 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

$$I_-(x) := \{i \mid \underline{x}_i < x_i < \overline{x}_i\} \tag{3}$$

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

$$\begin{aligned} I_+(x) &:= I_-(x) \cup \{i \mid g_i^{\mathrm{red}} \neq 0\} \\ &= I_-(x) \cup \{i \mid \underline{x}_i = x_i < \overline{x}_i, g_i < 0 \text{ or } \underline{x}_i < x_i = \overline{x}_i, g_i > 0\} \end{aligned} \tag{4}$$

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

$$I = I_+(x) \neq I_-(x), \tag{5}$$

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

$$\|g_I\|_*^2 \geq \rho \|g^{\mathrm{red}}\|_*^2, \quad \text{for some } \rho \in]0, 1]. \tag{6}$$

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^{best} and its function value $f^{\mathrm{best}} = f(x^{\mathrm{best}})$ as output. It uses the tuning parameters of a line search **CLS** (discussed in Section 1.3) and two tuning parameters:
$0 < \Delta_{\mathrm{a}} < 1$ (parameter for the angle condition),
$0 < \rho \leq 1$ (parameter for freeing iterations).

The bent line search and conditions (6)–(9) on the search direction are essential for the convergence analysis in [54, Sections 8 and 11], where the following is proved:

Algorithm 1 BOPT, bound constrained optimization algorithm

Initialization

Compute the initial gradient $g^0 := g(x^0)$.
Compute the initial **reduced gradient** $g^{\rm red}(x^0)$.
Find the initial working set $I^0 := I_+(x^0)$. ▷ the maximal set is chosen
for $\ell = 0, 1, 2, \cdots$ **do**

Stopping test

  if $g^{\rm red}(x^\ell) = 0$ **then**
    Set $x^{\rm best} = x^\ell$ and $f^{\rm best} = f(x^\ell)$. Then terminate **BOPT**.
  end if

Computing search direction

  Compute a search direction p^ℓ satisfying

$$\text{▷ Restriction on } p^\ell \text{ to the subspace of } I\text{:} \quad p_i^\ell = 0 \quad \text{for } i \notin I^\ell, \tag{7}$$

$$\text{▷ Angle condition:} \quad \frac{(g_I^\ell)^T p_I^\ell}{\|g_I^\ell\|_* \|p_I^\ell\|} \le -\Delta_{\rm a} < 0, \tag{8}$$

$$\text{▷ For freeing iterations:} \quad \text{if (5) holds, } g_i^\ell p_i^\ell \le 0 \quad \text{for all } i. \tag{9}$$

Finding the step size

  Perform a line search **CLS** (discussed in Section 1.3) along the **bent search path**

$$x(\alpha) := \pi[x^\ell + \alpha^\ell p^\ell], \tag{10}$$

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

$$\pi[x^\ell + \alpha^\ell p^\ell]_i := \sup(\underline{x}_i, \inf(x_i + \alpha^\ell p_i^\ell, \overline{x}_i)) = \begin{cases} \underline{x}_i & \text{if } x_i^\ell + \alpha^\ell p_i^\ell \le \underline{x}_i, \\ \overline{x}_i & \text{if } x_i^\ell + \alpha^\ell p_i^\ell \ge \overline{x}_i, \\ x_i^\ell + \alpha^\ell p_i^\ell & \text{otherwise.} \end{cases} \tag{11}$$

  Set $x^{\ell+1} := x(\alpha^\ell)$, compute $g^{\ell+1} := g(x^{\ell+1})$ and $g^{\rm red}(x^{\ell+1})$.

Updating the working set

  Find $I^{\ell+1} := I_-(x^{\ell+1})$ by (3). ▷ the working set changes to the minimal set
  if (6) is violated **then** ▷ the working set changes to the maximal set
    Find $I^{\ell+1} := I_+(x^{\ell+1})$ by (4).
  end if
end for

Theorem 1 *Let f be continuously differentiable, with Lipschitz continuous gradient g. Let x^ℓ denote the value of x in Algorithm 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

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

Some limit point $\widehat{x}$ of the x^ℓ satisfies $f(\widehat{x}) = \widehat{f} \le f(x^0)$ and $g^{\rm red}(\widehat{x}) = 0$.
(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.*

1.3 **CLS** – the curved line search of **BOPT**

CLS (Algorithm 3.3 in [54] = Algorithm 2 below), the line search used in **BOPT**, is an efficient gradient-free curved line search algorithm. It searches for a piecewise linear search path defined by directions chosen to reduce 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).

Algorithm 2 CLS, a curved line search algorithm

Initialization

Set $\underline{\alpha} := 0$. ▷ the lower bound of the admissible step size
Set $\overline{\alpha} := \infty$. ▷ the upper bound of the admissible step size
Set $\alpha^0 := \alpha^{\text{init}}$. ▷ the initial step size
for $k = 0, \cdots, l^{\max}$ **do**

Compute the Goldstein quotient

Compute the point $x(\alpha^k)$ on the bent search path (10), its function value $f(x(\alpha^k))$.
Compute the **Goldstein quotient** (Goldstein [33])

$$\mu(\alpha^k) := \frac{f(x(\alpha^k)) - f(x^\ell)}{\alpha(g^\ell)^T p^\ell} \quad \text{for } \alpha^k > 0. \tag{12}$$

Stopping test

if the **sufficient descent condition**

$$\mu(\alpha^k)|\mu(\alpha^k) - 1| \geq \beta \ \text{ with fixed } \beta > 0 \tag{13}$$

holds **then** ▷ **CLS** is efficient
Set $\alpha^\ell = \alpha^k$, $x^{\ell+1} = x(\alpha^\ell)$, and $f^{\ell+1} = f(x^{\ell+1})$, **CLS** stops.
end if

Update the interval

if $\mu(\alpha^k) \geq \dfrac{1}{2}$ **then** ▷ update the interval
Set $\underline{\alpha} := \alpha^k$. ▷ the lower bound of the interval is updated
else if α^k reaches $\alpha^{\max}$ **then**
CLS stops.
else
Set $\overline{\alpha} := \alpha^k$. ▷ the upper bound of the interval is updated
end if

Update the step size

if k is zero **then** ▷ try to optimally handle the quadratic case by solving $\mu(\alpha^k) = \dfrac{1}{2}$
if $\mu(\alpha^k) < 1$ **then**
Take as step size $\alpha^{k+1} := \dfrac{1}{2}\alpha^k/(1 - \mu(\alpha^k))$. ▷ interpolant is done
else
Expand the step size to $\alpha^{k+1} := q\alpha^k$. ▷ extrapolation is done
end if
else ▷ update the step size
if $\overline{\alpha}$ is ∞ **then**
Expand the step size to $\alpha^{k+1} := q\alpha^k$. ▷ extrapolation is done
else if $\underline{\alpha}$ is zero **then**
Reduce the step size to $\alpha^{k+1} := \alpha^k/q$.
else ▷ the interval was found
Compute $\alpha^{k+1} := \sqrt{\underline{\alpha}\overline{\alpha}}$. ▷ the geometric mean of $\underline{\alpha}$ and $\overline{\alpha}$ is computed
end if
end if
end for

• Condition (13) is an improved form of the Goldstein condition

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

It forbids step sizes too large or too small by enforcing that $\mu(\alpha)$ is sufficiently positive and not too close to one.
• According to Theorem 3.2 of [54], a step size satisfying (13) 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 reveals that the secant step size is a minimizer of a convex quadratic function along the search ray and so the quadratic case is optimally started. Otherwise the function is far from quadratic and bounded.

1.4 **LMBOPT** – an efficient version of **BOPT**

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

Important novelties compared to the literature are
• the useful trick of moving the starting point slightly into the relative interior of the feasible domain **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 [54] for removing zigzagging,
• a new regularized conjugate gradient direction,
• safeguards for the curved line search taking into account 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 2:
• Subspace information is defined in Subsection 2.1.
• A new quasi Newton direction and a regularized conjugate gradient step are introduced in Subsections 2.2 and 2.3, respectively.
• Some implementation details of these directions are given in Subsection 2.4.

Improvements in the line search are discussed in Section 3:
• Issues with finite arithmetic are described in Subsection 3.1.
• Ingredients of an improved version of **CLS** are introduced in Section 3.

We introduce the master algorithm and its implementation details in Section 4:
• A useful starting point is suggested in Subsection 4.1.

• The conditions for accepting a new point are explained in Subsection 4.2.
• Some implementation details are given in Subsection 4.3.
• The master algorithm is introduced in Subsection 4.4.

Numerical results for unconstrained and bound constrained `CUTEst` problems [35] are summarized in Section 5:
• Details of test problems and a shifted starting point are discussed in Subsection 5.1.
• Default parameters for **LMBOPT** are given in Subsection 5.2.
• Subsection 5.3 contains a list of all compared solvers and explains how unconstrained solvers turn into bound constrained solvers.
• First numerical results are given in Subsection 5.4.1, resulting in the three best solvers (**LMBOPT**, **ASACG**, and **LMBFG-EIG-MS**).
• Additional numerical results classified by constraints and dimensions are given in Subsection 5.4.2, resulting in a solver choice in Subsection 5.6.
• Further numerical results for hard problems are given in Subsection 5.5.
• As a consequence, a solver choice is based on our findings depending on the problem dimension, the presence or absence of constraints, the desired robustness, and the relative costs of the function and gradient evaluations in Subsection 5.6.

The website `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.

2 Search directions

In this section, we describe the search direction used at each iteration. In Subsection 2.1, subspace information is described. Accordingly, a new limited memory quasi Newton direction is discussed in Subsection 2.2. Then Subsection 2.3 describes how conjugate gradient directions are constructed and regularized. Finally, Subsection 2.4 contains the implementation details of our regularized conjugate gradient direction.

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**

$$BS = Y. \tag{14}$$

Since B is symmetric,

$$H := S^T Y = S^T BS \tag{15}$$

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

$$h := S^T Bs = Y^T s = S^T y, \tag{16}$$

$$0 < \gamma := s^T Bs = y^T s, \tag{17}$$

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

$$z^{\text{new}} := -H^{-1}c \ \text{ with } c := S^T g, \tag{18}$$

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

$$S^T g(x^{\text{new}}) = 0. \tag{19}$$

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

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

$$H^{\text{new}} := (S^{\text{new}})^T BS^{\text{new}} := \begin{pmatrix} H & h \\ h^T & \gamma \end{pmatrix}, \tag{20}$$

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

$$z^{\text{new}} := -(H^{\text{new}})^{-1} c^{\text{new}}. \tag{21}$$

But when the allowed memory for S and Y is full 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 described above.

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

$$h := \begin{cases} S^T y & \text{if } \texttt{updateH}, \\ Y^T s & \text{otherwise.} \end{cases} \tag{22}$$

If the objective is not quadratic, (14) does not hold exactly and $H := S^T Y$ need not be symmetric. However, the update (20) always produces a symmetric H, even in finite precision arithmetic.

2.2 A new quasi-Newton direction

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

$$B = D + WXW^T, \tag{23}$$

for some symmetric matrix $W \in \mathbb{R}^{n\times m}$ and some matrix $X \in \mathbb{R}^{n\times m}$. Thus, the additional assumption is temporarily 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 [12], which needs twice as much storage and computation time.

Theorem 1 *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* (14) *and* (23) *imply*

$$B = D + U\Sigma^{-1}U^T, \tag{24}$$

where

$$U := Y - DS \tag{25}$$

and

$$\Sigma := U^TS \tag{26}$$

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

$$M := U^TD^{-1}Y = \Sigma^{-1}, \tag{27}$$

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

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

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

and since B is symmetric, $\Sigma = S^T(B - D)S$ is symmetric, too. By assumption, the $m \times m$ matrix $Z := XW^TS$ is invertible, hence

$$W = UZ^{-1} \text{ and } Z = XZ^{-T}U^TS = XZ^{-T}\Sigma.$$

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 (27) by setting $Y = U + DS$ in (27) and using (26). With the symmetric matrix H defined by (15), we compute the symmetric $m \times m$ matrix M

$$\begin{aligned} M = U_{I:}^T D_{II}^{-1} Y_{I:} &= (Y_{I:} - D_{II}S_{I:})^T D_{II}^{-1} Y_{I:} = Y_{I:}^T D_{II}^{-1} Y_{I:} - S_{I:}^T Y_{I:} \\ &= Y_{I:}^T D_{II}^{-1} Y_{I:} - H \end{aligned} \tag{28}$$

and find

$$z = M^{-1} U_{I:}^T D_{II}^{-1} g_I; \tag{29}$$

hence

$$p_I = D_{II}^{-1}(U_{I:}z - g_I). \tag{30}$$

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

$$D_{ii} := \sqrt{\sum_{j\in J} \mathtt{YY}_{ij} \Big/ \sum_{j\in J} \mathtt{SS}_{ij}} \tag{31}$$

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 (8). We may add a multiple of the gradient to enforce the angle condition (8) for the modified direction

$$p^{\text{new}} := p - tg \tag{32}$$

with a suitable factor $t \geq 0$; the case $t = 0$ corresponds to the case where p already satisfies the bounded angle condition (8). 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 (8) holds with $p^{\rm new}$ in the place of p. If $\sigma^{\rm new} \leq -\Delta_{\rm a}$, this holds for $t = 0$, and we make this choice. Otherwise we may enforce the equality (8) by choosing

$$t := \frac{\sigma + \Delta_{\rm a}\sqrt{w}}{\sigma_1} \quad \text{with} \quad w := \frac{\sigma_1\sigma_2(1 - (\sigma^{\rm new})^2)}{1 - \Delta_{\rm a}^2}. \tag{33}$$

The following proposition is a special case of Proposition 5.2 in [54].

Proposition 1 *Suppose that $g \neq 0$ and $0 < \Delta_{\rm a} < 1$. Then if t is chosen by* (33)*, the search direction* (34) *satisfies the angle condition* (8)*.*

Given z by (29), we could compute the nonactive part of p from (30); 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

$$p_I = D_{II}^{-1}(u - tg_I), \tag{34}$$

where t is chosen analogous to (33) if this results in $t < 1$, and $t = 1$ otherwise. By Proposition 1, the direction (34) satisfies the angle condition (8).

2.3 A conjugate gradient step

Neumaier & Azmi [54, Section 7] introduced a new nonlinear conjugate gradient method chosen to reduce 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 $g^T p < 0$. To avoid zigzagging, [54] 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 $p^{\rm old}$. By Theorem 7.1 in [54] (applied for $B = I$),

$$p = \overline{\beta}p^{\rm old} - \widehat{\lambda}g, \tag{35}$$

with

$$\overline{\beta} > 0, \quad \widehat{\lambda} = \frac{\overline{c} + \overline{\beta}g^T p^{\rm old}}{g^T g}. \tag{36}$$

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

[54, 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

$$(g^\ell)^T g^\ell \leq \kappa_1 (y^{\ell-1})^T y^{\ell-1}, \tag{37}$$

$$(y^{\ell-1})^T p^{\ell-1} \leq \kappa_2 (g^{\ell-1})^T p^{\ell-1} \tag{38}$$

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 [54, Theorem 7.5] the sequence generated by (35) and (36) can be rewritten into the nonlinear conjugate gradient method of Fletcher & Reeves [30] equivalent to the linear conjugate gradient method of Hestenes & Stiefel [45] when f is quadratic with the positive definite Hessian matrix and bounded below. Therefore it requires at most n steps to obtain a minimizer of f.

As a consequence of [54, Theorem 7.3], [54, Theorem 7.6] showed that the sequence x^ℓ of the conjugate gradient method generated by (35) and (36) 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.

In this section, we discuss a new conjugate gradient method equivalent to the linear conjugate gradient method of Hestenes & Stiefel [45] in the cases where f is quadratic with the positive definite Hessian matrix and bounded. Theorems 7.3, 7.5, 7.6 in [54] are valid for our conjugate gradient method.

The **conjugate gradient method** chooses the direction s in the subspace generated by **typeSubspace** and enforces the conjugacy relation

$$h = Y^T s = 0; \tag{39}$$

thus making H diagonal. Both conditions together determine s up to a scaling factor: s must be a multiple of

$$p := Sr + p^{\text{init}} \tag{40}$$

for some $r \in \mathbb{R}^m$, in which p^{init} is a descent direction, computed by **searchDir** (discussed later in Subsection 2.4).

Then (39) requires

$$Hr = q := -Y^T p^{\text{init}}. \tag{41}$$

`hist` denotes the subspace basis index set (discussed later in Subsection 2.4). By restricting S, Y, and H to `hist`, we define

$$c_h := \sum_{i\in\texttt{hist}} g_I \circ S_{I,i}, \quad q_h := \sum_{i\in\texttt{hist}} p_I^{\text{init}} \circ Y_{I,i}, \quad H_h := H_{\texttt{hist},\texttt{hist}}.$$

Then we solve the linear systems $H_h z_h := -c_h$ and $H_h r_h := q_h$ according to (21) and (41), respectively. Afterwards, we construct the conjugate gradient direction depending on whether the subspace is possible ($m_0 > 0$) or not ($m_0 = 0$) by

$$p_I := \begin{cases} -\zeta p_I^{\text{init}} + S_{I,\texttt{hist}}(z_h + \zeta r_h) & \text{if } m_0 > 0, \\ -\zeta p_I^{\text{init}} & \text{if } m_0 = 0, \end{cases} \tag{42}$$

with

$$\zeta := \begin{cases} \dfrac{g_I^T p_I^{\text{init}} + q^T z_h}{\gamma - q_h^T r_h} & \text{if } m_0 > 0, \\ \dfrac{g_I^T p_I^{\text{init}}}{\gamma} & \text{if } m_0 = 0. \end{cases} \tag{43}$$

Here γ is computed by

$$\gamma := \frac{f(x + \alpha p^{\text{init}}) - f - \alpha g_I^T p_I^{\text{init}}}{\alpha^2/2}$$

where α is found by a heuristic way (**goodStep**; discussed later in Section 3).

In finite precision arithmetic, a tiny denominator in (43) produces a very inaccurate γ. This drawback is overcome by regularization. The error made in γ_{reg} is a tiny multiple of

$$\gamma_{\text{reg}} := \frac{|f(x + \alpha p^{\text{init}}) - f| + \alpha |g_I|^T |p_I^{\text{init}}|}{\alpha^2/2}. \tag{44}$$

We therefore shift the denominator in (43) away from zero to

$$\texttt{denom} := \begin{cases} \gamma - q_h^T r_h + \Delta_H(\gamma_{\text{reg}}/2 + |q_h|^T |r_h|) & \text{if } \gamma \geq q_h^T r_h, \\ \gamma - q_h^T r_h - \Delta_H(\gamma_{\text{reg}}/2 + |q_h|^T |r_h|) & \text{otherwise,} \end{cases} \tag{45}$$

where $\Delta_H \in (0, 1)$ is a tiny factor. Then the **regularized conjugate gradient direction** is computed by

$$p_I^{\text{new}} := \begin{cases} -\zeta_{\text{reg}} p_I^{\text{init}} + S_{I,\texttt{hist}}(z_h + \zeta_{\text{reg}} r_h) & \text{if } m_0 > 0, \\ -\zeta_{\text{reg}} p_I^{\text{init}} & \text{if } m_0 = 0, \end{cases} \tag{46}$$

with

$$\zeta_{\text{reg}} := \begin{cases} \dfrac{g_I^T p_I^{\text{init}} + q_h^T z_h}{\texttt{denom}} & \text{if } m_0 > 0, \\ \dfrac{g_I^T p_I^{\text{init}}}{\gamma} & \text{if } m_0 = 0. \end{cases} \tag{47}$$

If

$$\texttt{cosine} := \begin{cases} -\zeta_{\text{reg}} g_I^T p_I^{\text{init}} + c_h^T z_h & \text{if } m_0 > 0, \\ -\zeta_{\text{reg}} g_I^T p_I^{\text{init}} & \text{if } m_0 = 0 \end{cases} \tag{48}$$

is negative, (46) is a descent direction.

2.4 Some implementation details

In this section, we first discusses how to determine the ingredients of the subspace. e.g., the subspace dimension, the subspace basis index set, the subspace type, and p^{init}. Then we describe how to implement our regularized conjugate gradient direction.

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

$$|g^T y| \geq \Delta_{po} g^T g, \tag{49}$$

where $\Delta_{po} \in (0, 1)$ is a tiny tuning parameter. Initially $m = 0$ and whenever a new pair (s, y) satisfies (49), m is increased by one by appending these vectors to S and Y, respectively. But once m reaches its limit, it is kept to be fixed and the oldest column of S and Y is replaced by s and y, respectively.

When the activity does not change, the step is called a **local step**. For the construction of our regularized conjugate gradient direction, it is important which subspace basis index set is present and what is subspace dimension. We denote by `nlocal` the number of local steps and by `nwait` the number of local steps before starting the regularized conjugate gradient direction, which will be a tuning parameter. We use `nlocal` and `nwait` to determine the subspace dimension.

To encode which subspace should be used, **typeSubspace** updates three variables m_0 (subspace dimension), `hist` (subspace basis index set), and `CG` (subspace type).

Given the number of updated subspace `nh`, **typeSubspace** defines $\widehat{m} := \min(m, \texttt{nh})$ and identifies `CG`, m_0, and `hist` as follows:

• If $\texttt{nlocal} < \texttt{nwait}$, the **ordinary subspace step** is used since the full subspace direction may be contaminated by nonactive components and so lead to premature freeing if used directly. In this case, $\texttt{CG} := 0$, $m_0 := \min(\texttt{ng} - 1, \widehat{m})$, and $\texttt{hist} := \{1, \cdots, m_0\}$.

• If $\texttt{nlocal} = \texttt{nwait}$, the **quasi-Newton step** generated by **quasiNewtonDir** is used if (34) holds, and the subspace basis index set is permuted by

$$\texttt{perm} := \{\texttt{ch} + 1, \cdots, \widehat{m}, 1, \cdots, \texttt{ch}\}$$

so that the oldest columns are shifted with the newest, i.e.,

$$S := S_{:\texttt{perm}}, \; Y := Y_{:\texttt{perm}}, \text{ and } H := H_{\texttt{perm},\texttt{perm}};$$

here `ch` is a counter for m. Then set $\texttt{ch} := 0$. In this case, $\texttt{CG} := 1$, $m_0 := 0$, and $\texttt{hist} := \emptyset$. Since $m_0 = 0$, there is a premature replacement of s (y) by the first column of the subspace matrix S (Y) before `ch` exceeds m.

• If $\texttt{nlocal} < \texttt{nwait}+\widehat{m}$, the **conjugacy relation** (39) is preserved by restricting the subspace. In such a case, $\texttt{CG} := 2$, $m_0 := \texttt{nlocal} - \texttt{nwait}$, and $\texttt{hist} := \{1, \cdots, m_0\}$.

• Otherwise, the **full subspace step** preserves the **conjugacy**. In this case, $\texttt{CG} := 3$, $m_0 := \widehat{m}$, and $\texttt{hist} := \{1, \cdots, \widehat{m}\}$.

The value of γ depends on the search direction. Here **searchDir** is used to compute p^{init} including **scaleDir**, **quasiNewtonDir**, and **AvoidZigzagDir**. It works as follows:

• 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

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

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

$$p_i^{\text{init}} := \begin{cases} \texttt{sc} & \text{if } x_i = \underline{x}_i, \\ -\texttt{sc} & \text{elseif } x_i = \overline{x}_i, \\ \texttt{sc} & \text{elseif } g_i < 0, \\ -\texttt{sc} & \text{otherwise.} \end{cases}$$

• If $\texttt{nlocal} \neq \texttt{nwait}$, a modified direction is used to avoid zigzagging. p^{init} is computed by (35) using (36). Since $g_I^T p_I^{\text{init}} = -\overline{c}$, the direction will be a descent direction. This direction is implemented by **AvoidZigzagDir** and enriched by a **new heuristic choice** of

$$\overline{\beta} := \theta \max_{i=1:n} \left\{ \left| \frac{g_i}{p_i^{\text{init}}} \right| \right\}, \tag{50}$$

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 (8) does not hold:

• If $g_I^T p_I^{\text{init}} > 0$, p_I^{init} is chosen to be its opposite to move away from maximizer or saddle point.

• By changing the sign of g, it may enforce $g_I^T p_I^{\text{init}} \leq 0$. Even though $g \neq 0$, cancellation may lead to a tiny $g_I^T p_I^{\text{init}}$ (and even with the wrong sign). Given a tiny parameter Δ_{pg}, to overcome this weakness, the subtract $\Delta_{pg}|g_I|^T|p_I^{\text{init}}|$ can be a bound on the rounding error to get the theoretically correct sign. A **regularized directional derivative** is done if the condition

$$|g_I^T p_I^{\text{init}}| \leq \Delta_{pg}|g_I|^T|p_I^{\text{init}}| \tag{51}$$

holds, enforcing $g_I^T p_I^{\text{init}} < 0$. In this case, if (51) holds, p_I^{init} 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, p_I^{init} is chosen to be $-\lambda_b g_I$.

In summary, the implementation of our regularized conjugate gradient direction, called **ConjGradDir**, for computing p in (46) is given as follows: (i) γ_{reg} is computed by **getGam** according to (44), (ii) if the subspace dimension is nonzero, our conjugate gradient direction is used; otherwise, it reduces to $p^{\text{new}} = -\zeta_{\text{reg}} p^{\text{init}}$ and the subspace basis index set is restarted, (iii) a regularization for the denominator of (47) is made according to (45) by **regDenom**, (iv) the condition (48) is computed to know whether the regularized conjugate gradient direction is descent or not, (v) the new trial point, $x + p^{\text{new}}$, is projected into $\mathbf{x}$, resulting in x^{new} and the direction is recomputed by $p^{\text{new}} := x^{\text{new}} - x$.

3 Improvements in the line search

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

3.1 Issues with finite precision arithmetic

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

Example 1 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 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 failure of the line search. The danger is particularly likely when the search direction is almost orthogonal to the gradient. Hence, before each line search method, we need to produce a starting step size by a method, called **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 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

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

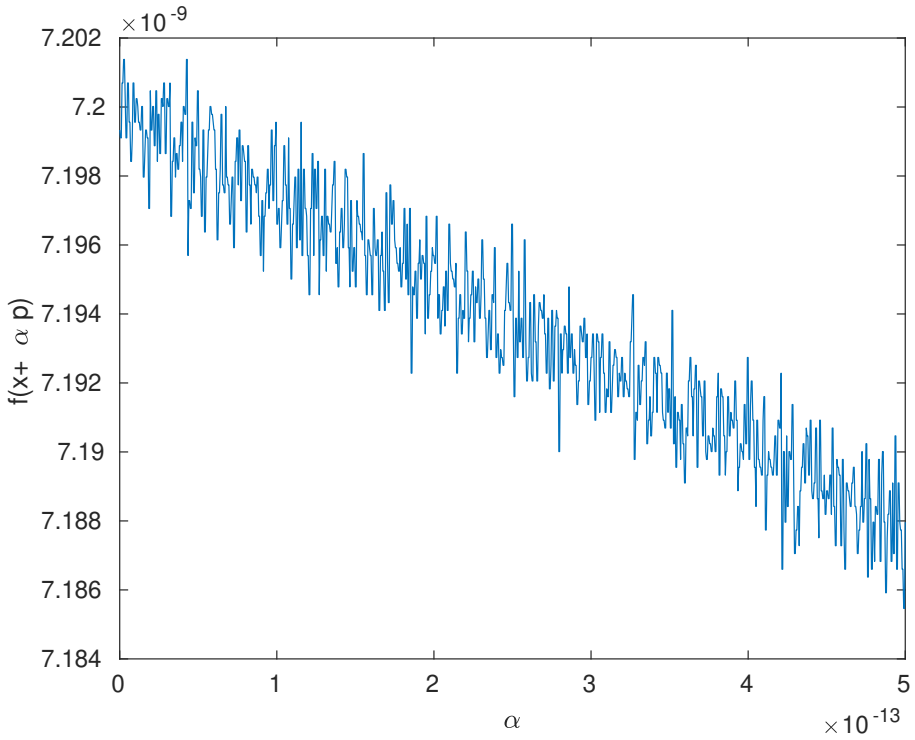


Fig. 1: In the Example 1 points with step sizes $\alpha < 0.5 \times 10^{-13}$ have a high probability for having $f(x + \alpha p) \geq f(x)$.

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 `nstuck` of times that **LMBOPT** cannot update the best point. 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.

3.2 **CLS-new** – an improved version of **CLS**

Before **CLS-new** tries to enforce the sufficient descent condition (13), the following steps must be taken:
• **LMBOPT** calls **enforceAngle** to enforce the angle condition (8).
• Once **LMBOPT** calls **initInfo** to initialize the best function value $f^{\text{best}} := f^0$ and to compute the factor for adjusting increases in f (discussed below)

$$\delta_f := \begin{cases} \texttt{facf} * |f^0| & \text{if } f^0 \in (0, \infty), \\ 1 & \text{otherwise,} \end{cases} \tag{52}$$

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

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

(2) the best point information by $f^{\text{best}} := f^{\text{new}}$ and $x^{\text{best}} := x^{\text{new}}$ if $\texttt{nstuck} = 0$;
(3) δ_f and `Df`. In this case, if $f^{\text{new}} < f$, then $\delta_f := f - f^{\text{new}}$ and $\texttt{nm} := \text{mod}(\texttt{ng}, \texttt{mf})$ are computed. Otherwise since the function value is not decreased, δ_f is expanded by $\delta_f := \max(\Delta_f \delta_f, \Delta_m(|f| + |f^{\text{new}}|))$ 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^{new} if $f^{\text{new}} < f$ holds.
• If $\alpha^{\text{good}} \geq 1$, 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 $q\alpha^{\text{good}}$ is moderately large, this choice may helps **CLS-new** to prevent a failure. To get target step sizes which **should not become too small**, an acceptable increase in f (denoted by `df`) must be estimated in a heuristic way such that it becomes slowly small. Accordingly, at first, `df` is δ_f computed by (52). Next, it is a multiple of the old δ_f value if the tuning parameter `mdf` divides `ng`. Otherwise it is the maximum of the `mf` old δ_f valuesİn 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 (13).
• **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 satisfies $\mu(\alpha) < 1$ an exact line search uses the **secant step** $\frac{1}{2}\alpha/(1 - \mu(\alpha))$ for the quadratic objective function. In fact this ensures finite termination of our conjugate gradient method for the quadratic functions. Otherwise **extrapolation** is done by the factor $q > 1$. In the next iteration, if the sufficient descent condition (13) does not hold, then the function is far from quadratic and bounded. In such a case, either **interpolate** is performed

if the lower bound on the step size is zero or **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** ends up providing no improvement in 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 between the current best function value and 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 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 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 the 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 whether **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 a factor of $1 -$ `del` and all its zero components (if any) are replaced by `del` in both cases. Then it is projected into the feasible set `x`.

4 Starting point and master algorithm

4.1 **projStartPoint** – the starting point

A poor choice of the starting point can lead to inefficiencies. For example, consider minimizing the quadratic function

$$f(x) := (x_1 - 1)^2 + \sum_{i=2}^{n} (x_i - x_{i-1})^2$$

that starts with $x^0 = 0$. 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 preconditioned gradients computed earlier, 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 the 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.

4.2 **getSuccess** – successful iteration

The goal of **getSuccess** is to test whether the sufficient descent condition (13) holds or not; the only difference being that the tuning parameter β is replaced by the other tuning parameter $\beta^{\texttt{CG}}$. The Goldstein quotient (12) is computed provided that all of the following hold:
• The regularized conjugate gradient direction is descent, i.e., (48) is negative, but it is not zero.
• `nlocal` $\geq$ `nwait` or `nstuck` $\geq$ `nsmin`.

After computing the Goldstein quotient (12), the iteration will be successful if either line search is efficient, meaning the sufficient descent condition (13) with $\beta = \beta^{\texttt{CG}}$ holds, or there exists an improvement in the function value by at least δ_f and `nstuck` $\geq$ `nsmin`. In this case, the Boolean variable `success` is evaluated as true; otherwise, it is evaluated as false.

4.3 Some implementation details

To determine the working set I, it is checked if one of the following holds:
(1) The function value cannot be decreased.
(2) The size of the 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 finding the freeing iteration (which is a tuning parameter) is exceeded.
(4) Condition (6) 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) and initializes the working set with $I(x) := I_+(x)$. Then if the statements (1)-(3) are true, **findFreeNeg** finds $I_-(x)$ by (3) and **findFreePos** checks whether the statement (4) is true or not. If this statement is not true, the working set is $I(x) := I_-(x)$; otherwise, **findFreePos** finds $I_+(x)$ by (4) and chooses it as the new working set $I(x) := I_+(x)$.

If at least one of the statements (1)-(4) holds, a scaled Cauchy point is tried. It is computed in the same way as [46] but with the difference that the scaling matrix is computed by

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

with

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

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

$$\overline{D}_{ii} := \left| \frac{g_i}{p_i^{\text{init}}} \right|, \quad \text{for } i = 1, \cdots, n,$$

where p^{init} is computed as discussed above.

4.4 The master algorithm

We now recall the main ingredients of **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 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 computation happens later for other points. 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 each 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 p^{init} by **searchDir** to construct the subspace, and then constructs the regularized conjugate gradient direction p by **ConjGradDir** to achieve a successful iteration, provided the activity is changed; otherwise the scaled Cauchy point is computed by **scaleCauchy** if at least one of the statements (1)-(4) holds, discussed earlier in Subsection 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 at least `nnullmax` null steps are repeated 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 by 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 more null step, (ii) either the condition (49) holds or the number of local steps exceeds its limit.

LMBOPT minimizes the bound constrained optimization problem (1). It takes the initial point x^0, the feasible set $\mathbf{x}$ and the 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 3 we refer to Theorem 1.

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

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

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 5.3) on a large public benchmark. Only

Algorithm 3 LMBOPT, limited memory bound-constrained optimization

Initialization

Initialize the subspace information and other necessary information.
Improve the starting point x^0 by **projStartPoint**.
Compute initial function value $f^0 := f(x^0)$ and its gradient $g^0 := g(x^0)$.
Check whether g^0 needs to be adjusted by **adjustGrad** or not.
Initialize the necessary information by **initInfo**.
for $\ell = 0, 1, 2, \cdots$ **do** ▷ main loop
  Compute the reduced gradient by **redGrad**.

For $\ell > 0$ check whether the condition (6) is violated or not

  Find the free indices by **findFreePos**. ▷ $I^\ell = I_-(x^\ell)$ or $I^\ell = I_+(x^\ell)$?

Stopping test

  if $\|g^{\rm red}(x^\ell)\|_\infty \le \varepsilon$ or number of stuck iterations exceeds its limit **then**
    Set $x^{\rm best} = x^\ell$ and $f^{\rm best} = f(x^\ell)$ and **LMBOPT** stops.
  end if

Construct the regularized conjugate gradient direction

  Identify kind of the subspace by **typeSubspace**. ▷ spanned by the columns of S
  Compute $(p^{\rm init})^\ell$ by **searchDir**. ▷ needed to construct a bigger subspace
  Compute p^ℓ by **ConjGradDir**. ▷ the regularized conjugate gradient direction

Try to decrease in function value by a scaled Cauchy point

  if at least one of the statements (1)-(4) is true (discussed in Subsection 4.3) **then**
    A scaled Cauchy point by **scaleCauchy** is tried.
  end if

Determine whether the iteration is successful (`success` = 1) or not (`success` = 0)

  Perform **getSuccess** to determine the Boolean variable `success`.
  if `success` is true **then** ▷ the iteration is successful
    Set $x^{\ell+1} = x^\ell + p^\ell$.
  else ▷ the iteration is unsuccessful
    Regularize the direction p^ℓ by **enforceAngle**.
    Perform **CLS-new**. ▷ along the regularized direction resulting in $x^{\ell+1}$
    Compute the step $s^{\ell+1} := x^{\ell+1} - x^\ell$.
    if $\|s^\ell\|$ is zero **then** ▷ a null step is found
      Perform **nullStep**. ▷ to check whether a null step is removed or not
    end if
    if $\|s^\ell\|$ is zero **then** ▷ **nullStep** cannot remove the null step
      Increase the number of null steps.
      if the maximum number of null steps is reached **then**
        LMBOPT ends.
      end if
    end if
  end if

Update the subspace spanned by the columns of S and the working set

  if $\|s^\ell\| \neq 0$ **then**
    Compute the gradient $g^{\ell+1} := g(x^{\ell+1})$.
    Check whether $g^{\ell+1}$ needs to be adjusted by **adjustGrad** or not.
    Set $y^{\ell+1} := g^{\ell+1} - g^\ell$ and 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 if
end for

summary results are given; supplementary information with much more detailed test results can be found in `suppMat.pdf` from the **LMBOPT** web site.

5.1 Test problems used

As test problems we used all 1088 unconstrained and bound constrained problems with up to 100001 variables from the CUTEst collection of optimization problems by Gould et al. [35], in case of variable dimension problems for all allowed dimensions in this range; see Section 5 of `suppMat.pdf`.

`nf`, `ng`, and `sec` denote the number of function evaluations, the number of gradient evaluations, and the time in seconds, respectively. Since the cost of computing the gradient is typically about twice the cost of the function value (see Section 3 of `suppMat.pdf`), we also use the cost measure $\texttt{nf2g} := \texttt{nf} + 2\texttt{ng}$. These measures are used as the cost measures to do performance profiles [24] shown in Figures 2-5.

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

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

sec of run time. A problem is considered solved if $\|g^k\| \leq 10^{-6}$.

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 $x^0 := 0$, but we shift the arguments by

$$\xi_i := (-1)^{i-1} \frac{2}{2+i}, \quad \text{for all } i = 1, \ldots, n \tag{53}$$

to avoid a solver guessing the solution of toy problems with a simple solution (such as all zero or all one) – there are quite a few of these in the `CUTEst` library. This 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 $\ell \geq 0$. 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 sec.

• 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 (53). Maximal time in sec increased from 300 sec to 7200 sec 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.

5.2 Default parameters for **LMBOPT**

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

`nsmin` $= 1$;	`nwait` $= 1$;	`rfac` $= 2.5$;	`nlf` $= 2$;	$\Delta_m = 10^{-13}$;	$\Delta_{pg} = \varepsilon_m$;
$\Delta_H = \varepsilon_m$;	$\Delta_\alpha = 5\varepsilon_m$;	$l^{\max} = 4$;	$\beta = 0.02$;	$\beta^{\texttt{CG}} = 0.001$;	$\Delta_{\mathrm{a}} = 10^{-12}$;
$\Delta_{reg} = 10^{-12}$;	$\Delta_w = \varepsilon_m$;	`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 a limited tuning by hand. In a further release we plan to find optimal tuning parameters [47], as the quality of **LMBOPT** depends on it.

5.3 Codes compared

We compare **LMBOPT** with competitive solvers for unconstrained and bound constrained optimization. These solvers are

ASACG [40,41,42,44],	**CGdescent** [40,41,44],	**ASABCP** [18],
SPG [8,9],	**LBFGSB** [12],	**LMBFG-DDOGL** [11],
LMBFG-EIG-MS-2-2 [11],	**LMBFG-BWX-MS** [11],	**LMBFG-EIG-inf-2** [11],
LMBFGS-TR [11],	**LMBFG-MTBT** [11],	**LMBFG-MT** [11],
LMBFG-EIG-MS [11],	**LMBFG-EIG-curve-inf** [11].	

Details about the solvers and options used can be found in Section 1 of `suppMat.pdf`. For some solvers, we have chosen options other than the default ones to make them more competitive.

We only compare public software with an available Matlab interface. **LANCELOT-B** combines a trust region approach with projected gradient directions. But since there was no mex-file to run **LANCELOT-B** in Matlab, we could not call and run it in our Matlab environment. Similarly, we could not find a version of **GENCAN**, the bound constrained version of **ALGENCAN** [7], which could be handled in Matlab. **GENCAN** is a combination of spectral projected gradient and an active set strategy. It is unlikely to introduce significant bias in the comparison. Hence, we compare **LMBOPT** to many known solvers using various active set strategies and either projected conjugate gradient methods, projected truncated Newton methods, or projected quasi Newton methods.

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, but the results show that **this is a simple and surprisingly effective strategy**.

5.4 The results for stringent resources

5.4.1 Unconstrained and bound constrained optimization problems

We tested all 15 solvers for problems in dimension 1 to 100001. A list of problems unsolved by all solvers can be found in Section 4 of `suppMat.pdf`.

For more refined statistics, we use our test environment (Kimiaei & Neumaier [47]) for comparing optimization routines on the `CUTEst` test problem collection.

For a given collection S of solvers and a collection $\mathcal{P}$ of problems, the **efficiency** of the solver so for solving the problem $j \in \mathcal{P}$ with respect to the cost measure c_s is the strength of the solver $so \in \mathcal{S}$ – relative to an ideal solver that matches on each problem the best solver. It is measured by

$$e^j_{so} := \begin{cases} \min\limits_{s\in S} c_s/c_{so}, & \text{if the solver } so \text{ solved the problem } j \in \mathcal{P}, \\ 0, & \text{otherwise.} \end{cases}$$

The **total mean efficiency** of the solver so with respect to c_s is defined by

$$e_{so} = \operatorname*{mean}_{j\in\mathcal{P}}(e^j_{so}).$$

T_{mean} is the mean of the time in seconds needed by a solver to solve the test problems chosen from the list of test problems $\mathcal{P}$, ignoring the times for unsolved problems. #100 is the total number of problems for which the solver so was best with respect to `nf2g` ($e^j_{so} = 1 = 100\%$). !100 is the total number of problems solved for which the solver so was better than all other solvers with respect to `nf2g`.

In the tables, efficiencies are given in percent. Larger efficiencies in the table imply a better average behaviour; a zero efficiency indicates failure. All values are rounded (towards zero) to whole integers. Mean efficiencies are taken over the 990 problems tried by all solvers and solved by at least one of them, out of a total of 1088 problems. The columns titled "# of anomalies" report statistic on failure reasons:

- n indicates that `nf2g` $\geq 20n + 10000$ `was reached`.
- t indicates that `sec` ≥ 300 `was reached`.
- f indicates that the `algorithm failed` for other reasons.

As can be seen from Table 1 and Figure 2, **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 to other algorithms in terms of the number of solved problems.

Table 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	sec
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

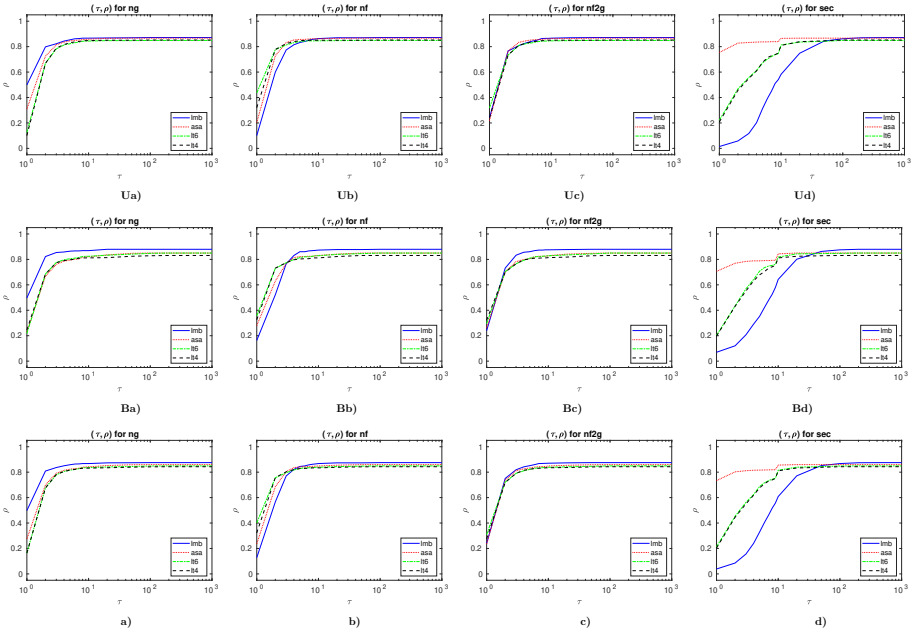


Fig. 2: (Ua)-(Ud): Performance profiles for unconstrained problems ($1 \leq n \leq 100001$) in terms of the ng/(best ng), nf/(best nf), nf2g/(best nf2g), and sec/(best sec) efficiencies, respectively. (Ba)-(Bd): Performance profiles for bound constrained problems ($1 \leq n \leq 100001$) in terms of the ng/(best ng), nf/(best nf), nf2g/(best nf2g), and sec/(best sec) 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 sec/(best sec) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

5.4.2 Classified by constraints and dimensions

Results for the three best solvers for all problems classified by dimension and constraint are given in Table 2, Figure 3, and Box plot 4. 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 for the unconstrained problems, (2) **LMBFG-EIG-MS** is the best in terms of `nf` for the both unconstrained and bound constrained problems and `nf2g` for the bound constrained problems only, (3) **ASACG** is the best solver in terms of the `nf2g` efficiency for the bound constrained problems;

• for **large-dimensional problems** ($501 \leq n \leq 100001$), (1) **LMBOPT** is the best solver in 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 for the unconstrained problems only.

• for all problems ($1 \leq n \leq 100001$), (1) **LMBOPT** is the best in terms of the number of solved problems 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 for both unconstrained and bound constrained problems.

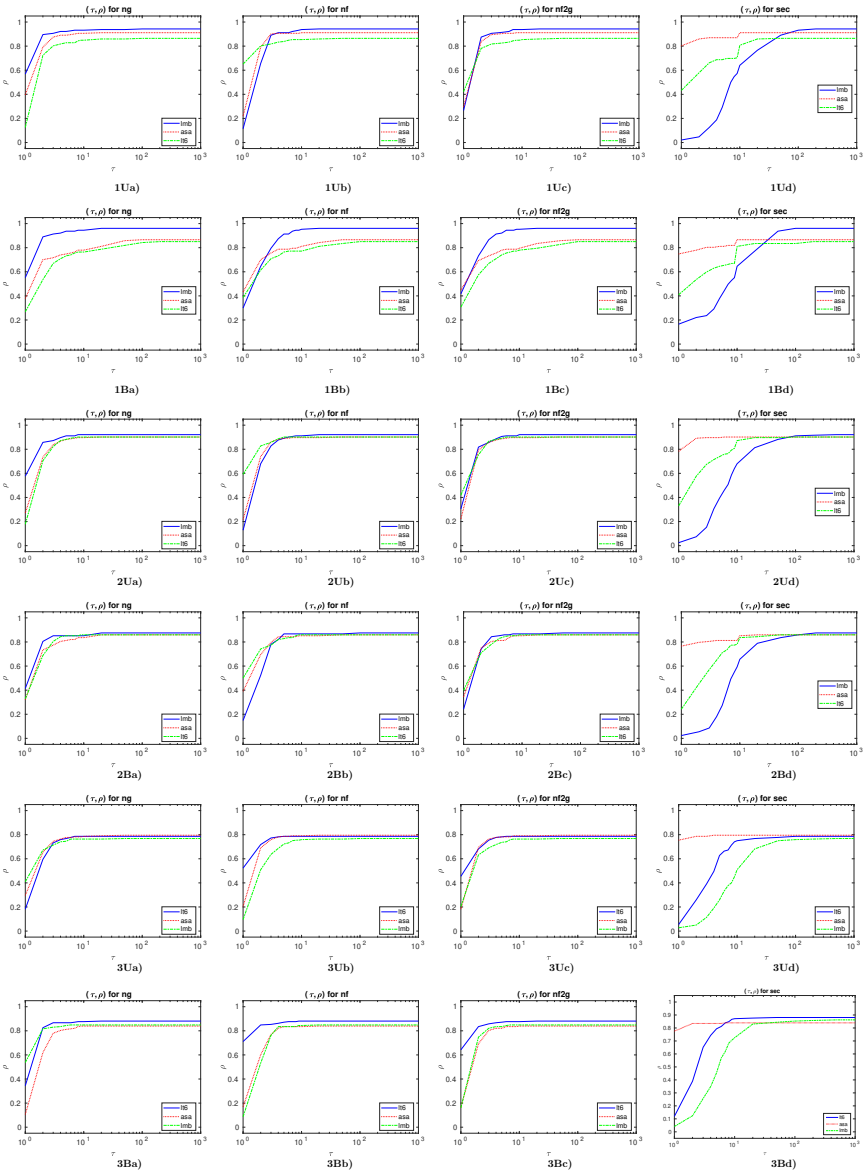


Fig. 3: (1Ua)-(1Ud)/(1Ba)-(1Bd): Performance profiles for low-dimensional unconstrained/bound constrained problems ($1 \leq n \leq 30$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `sec`/(`best sec`) efficiencies, respectively. (2Ua)-(2Ud)/(2Ba)-(2Bd): Performance profiles for medium-dimensional unconstrained/bound constrained problems ($31 \leq n \leq 500$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `sec`/(`best sec`) efficiencies, respectively. (3Ua)-(3Ud)/(3Ba)-(3Bd): Performance profiles for high-dimensional unconstrained/bound constrained problems ($501 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `sec`/(`best sec`) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

Table 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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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	sec
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

5.5 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 was used instead of (53) and both `nfmax` and `secmax` were increased. 41 test problems were not solved by all solvers for dimensions 1 up to 100001, given in Table 3.

From Table 4 and Figure 5, we conclude

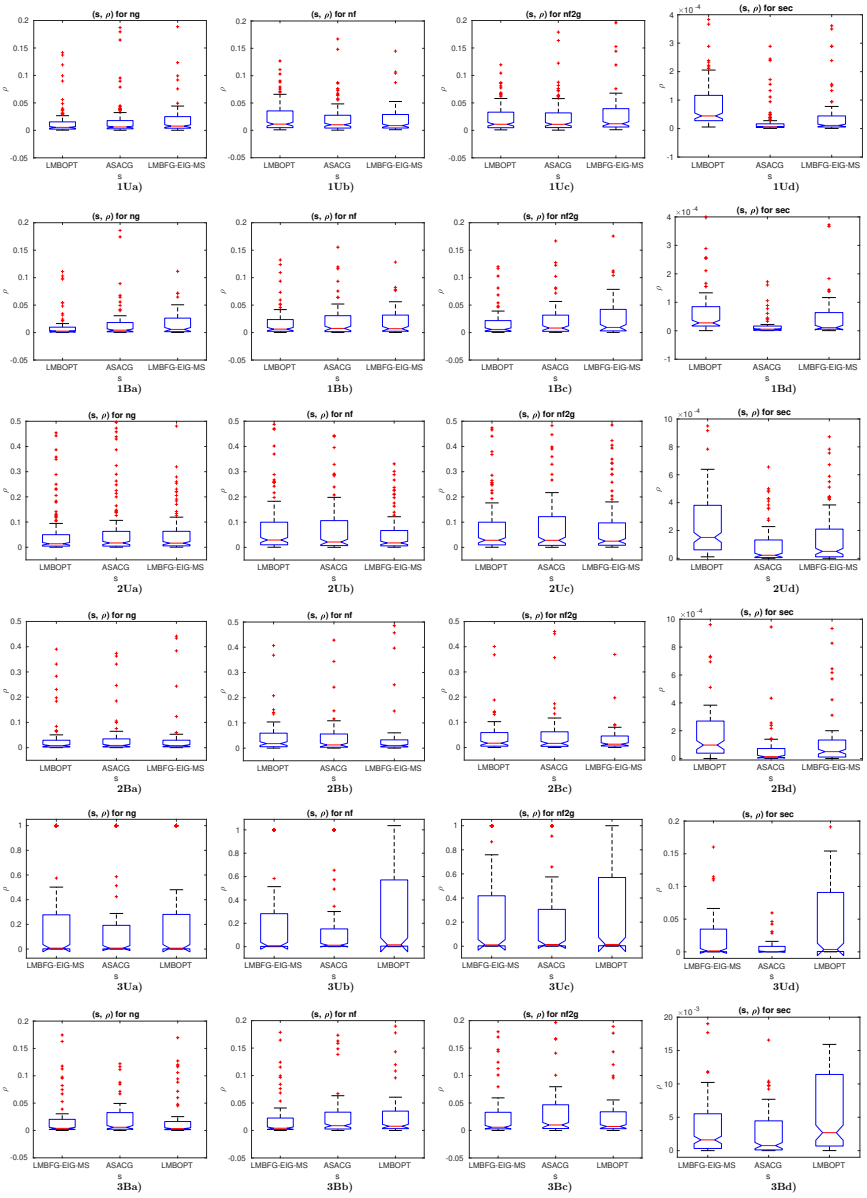


Fig. 4: We show box plots for the data summarized in Table 2. Here ρ stands for `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, `sec`/`secmax` and s stands for the name of solvers. (1Ua)-(1Ud)/(1Ba)-(1Bd): Box plots for low-dimensional unconstrained/bound constrained problems ($1 \leq n \leq 30$) in terms of `ng`/`ngmax`, `nf`/`nfmax`, `nf2g`/`nf2gmax`, and `sec`/`secmax`, respectively. (2Ua)-(2Ud)/(2Ba)-(2Bd): Box plots for medium-dimensional unconstrained/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/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.

• **LMBOPT** is the best in terms of the number of solved problems and the `ng` and `nf2g` efficiencies for the hard bound constrained problems

• **ASACG** is the best in terms of the number of solved problems and the `ng` and `nf2g` efficiencies for the hard unconstrained problems.

• **LMBFG-EIG-MS** is the best in terms of the `ng` and `nf2g` efficiencies for the hard unconstrained problems.

Table 3: 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			

Table 4: 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`	`sec`
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`	`sec`
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`	`sec`
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

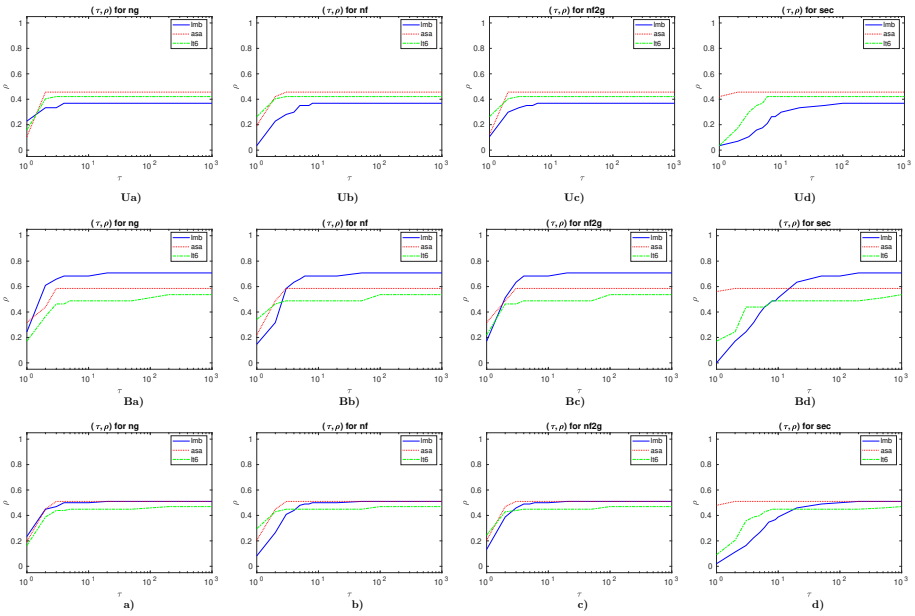
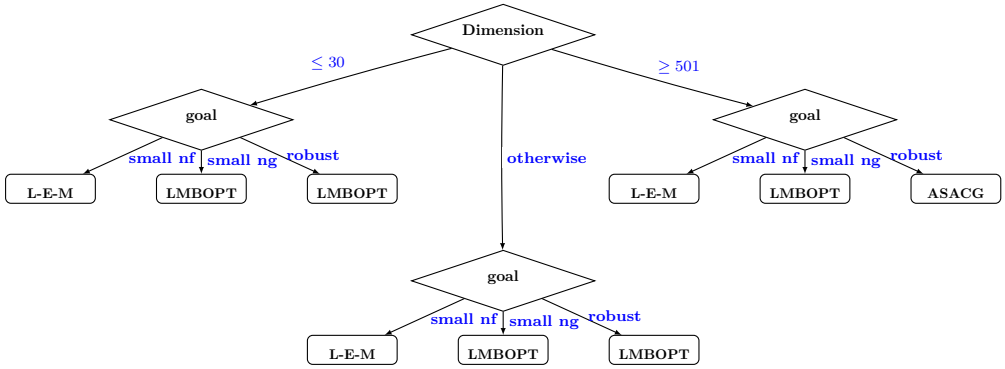


Fig. 5: (Ua)-(Ud): Performance profiles for unconstrained hard problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `sec`/(`best sec`) efficiencies, respectively. (Ba)-(Bd): Performance profiles for bound constrained hard problems ($1 \leq n \leq 100001$) in terms of the `ng`/(`best ng`), `nf`/(`best nf`), `nf2g`/(`best nf2g`), and `sec`/(`best sec`) 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 `sec`/(`best sec`) efficiencies, respectively. ρ designates the percentage of problems solved within a factor τ of the best solver. Problem solved by no solver are ignored.

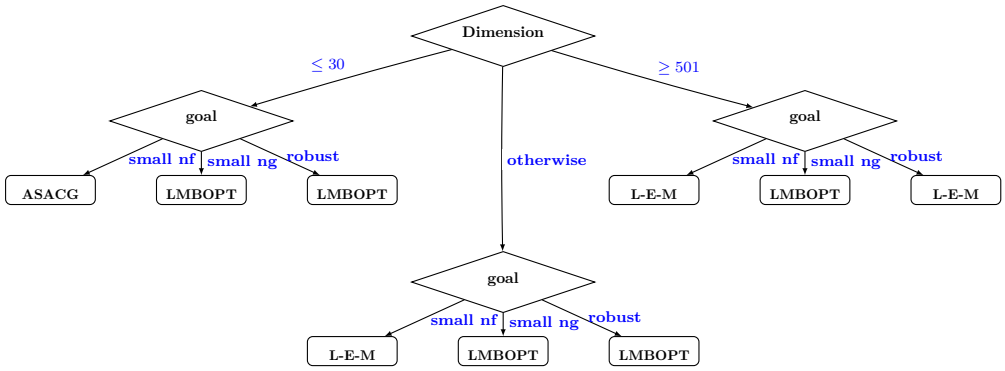
5.6 Recommendations

In this section, we recommend a solver choice based on our findings. The choice depends on the problem dimension, the presence or absence of constrains, the desired robustness, and the relative costs of the function and gradient evaluations shown in Subfigures (a)-(c) of Figure 6.

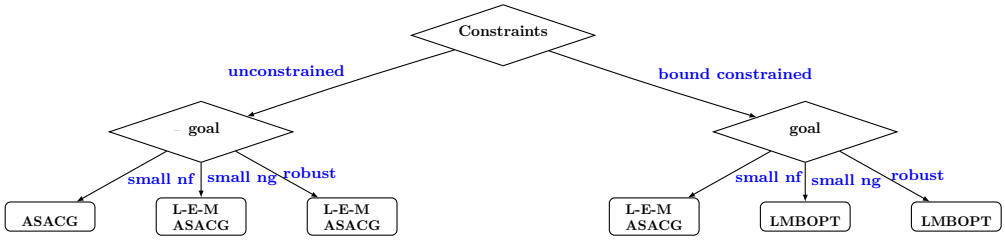
Fig. 6: (a) Flow chart for unconstrained problems classified by problems, (b) Flow chart for bound constrained problems classified by the problem dimension, (c) Flow chart for hard problems classified by constraint. Here **L-E-M** stands for **LMBFG-EIG-MS**.



(a)



(b)



(c)

References

1. R. Andreani, A. Friedlander, and J. M. Martínez. On the solution of finite-dimensional variational inequalities using smooth optimization with simple bounds. *J. Optim. Theory Appl.* **94** (1997), 635–657.
2. J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA J. Numer. Anal.* **8** (198), 141–148.

3. D. P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM J. Control Opim.* **20** (1982), 221–246.
4. 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** (1999), 862–880.
5. E. G. Birgin and J. M. Martínez. A box-constrained optimization algorithm with negative curvature directions and spectral projected gradients. In *Topics in Numerical Analysis* (G. Alefeld and X. Chen, eds.), Vol. 15 of *Computing Supplementa*, pp. 49–60. Springer Vienna (2001).
6. E. G. Birgin and J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Comput. Optim. Appl.* **23** (2002), 101–125.
7. E. G. Birgin and J. M. Martínez. On the application of an augmented lagrangian algorithm to some portfolio problems. *EURO J. Comput. Optim.* **4** (October 2015), 79–92.
8. E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. Optim.* **10** (1999), 1196–1211.
9. E. G. Birgin, J. M. Martínez, and M. Raydan. Algorithm 813: Spg-software for convex-constrained optimization. *ACM Trans. Math. Softw.* **27** (2001), 340–349.
10. E. G. Birgin, J. M. Martínez, and M. Raydan. Inexact spectral projected gradient methods on convex sets. *IMA J. Numer. Anal.* **23** (2003), 539–559.
11. O. Burdakov, L. Gong, S. Zikrin, and Y. Yuan. On efficiently combining limited-memory and trust-region techniques. *Math. Program. Comput.* **9** (2017), 101–134.
12. R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.* **16** (1995), 1190.
13. R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Math. Program.* **63** (1994), 129–156.
14. P. Calamai and J. Moré. Projected gradient methods for linearly constrained problems. *Math. Program.* **39** (1987), 93–116.
15. 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** (1988), 433.
16. 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** (1988), 399–430.
17. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM J. Numer. Anal.* **28** (1991), 545–572.
18. 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** (2017), 369–401.
19. Y. H. Dai. On the nonmonotone line search. *J. Optim. Theory Appl.* **112** (2002), 315–330.
20. Y. H. Dai and R. Fletcher. Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming. *Numer. Math.* **100** (2005), 21–47.
21. Y. H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Math. Program.* **106** (2006), 403–421.
22. Y. H. Dai, W. W. Hager, K. Schittkowski, and H. Zhang. The cyclic Barzilai–Borwein method for unconstrained optimization. *IMA J. Numer. Anal.* **26** (2006), 604–627.
23. 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).
24. E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2001), 13.
25. Z. Dostál. Box constrained quadratic programming with proportioning and projections. *SIAM J. Optim* **7** (1997), 871–887.
26. Z. Dostál. A proportioning based algorithm with rate of convergence for bound constrained quadratic programming. *Numer. Algorithms* **34** (2003), 293–302.
27. Z. Dostál, A. Friedlander, and S. A. Santos. Solution of coercive and semicoercive contact problems by feti domain decomposition. *Contemp. Math.* **218** (1998), 82–93.

28. J. C. Dunn. On the convergence of projected gradient processes to singular critical points. *J. Optim. Theory Appl.* **55** (1987), 203–216.
29. R. Fletcher. On the Barzilai-Borwein method. *Optimization and Control with Applications* (2005), 235–256.
30. R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer J,* **7** (February 1964), 149–154.
31. P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London (1981).
32. W. Glunt, T. L. Hayden, and M. Raydan. Molecular conformations from distance matrices. *J. Comput. Chem.* **14** (1993), 114–120.
33. A. Goldstein and J. Price. An effective algorithm for minimization. *Numer. Math.* **10** (1967), 184–189.
34. 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** (December 2003), 353–372.
35. 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** (2015), 545–557.
36. L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for newton's method. *SIAM J. Numer. Anal.* **23** (1986), 707–716.
37. L. Grippo and M. Sciandrone. Nonmonotone globalization techniques for the Barzilai-Borwein gradient method. *Comput. Optim. Appl.* **23** (2002), 143–169.
38. W. W. Hager. Dual techniques for constrained optimization. *J. Optim. Theory Appl.* **55** (1987), 37–71.
39. W. W. Hager. Analysis and implementation of a dual algorithm for constrained optimization. *J. Optim. Theory Appl.* **79** (1993), 427–462.
40. W. W. Hager and H. Zhang. CG_DESCENT user's guide. Technical report, Department of Mathematics, University of Florida, Gainesville, FL (2004).
41. W. W. Hager and H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM J. Optim.* **16** (2005), 170–192.
42. W. W. Hager and H. Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Trans. Math. Softw.* **32** (2006), 113–137.
43. W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM J. Optim.* **17** (2006), 526–557.
44. W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pac. J. Optim.* **2** (2006), 35–58.
45. M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.* **49** (1952), 409–436.
46. W. Huyer and A. Neumaier. MINQ8: general definite and bound constrained indefinite quadratic programming. *Comput. Optim. Appl.* **69** (October 2017), 351–381.
47. 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).
48. Y. Lin and C. W. Cryer. An alternating direction implicit algorithm for the solution of linear complementarity problems arising from free boundary problems. *Appl. Math. Optimization* **13** (1987), 1–17.
49. W. Liu and Y. H. Dai. Minimization algorithms based on supervisor and searcher cooperation. *J. Optim. Theory Appl.* **111** (2001), 359–379.
50. P Lötstedt. Solving the minimal least squares problem subject to bounds on the variables. *BIT* **24** (1984), 206–224.
51. J. M. Martínez. BOX-QUACAN and the implementation of augmented lagrangian algorithms for minimization with inequality constraints. *Comput. Appl. Math.* **19** (2000), 31–36.
52. J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numer. Math.* **55** (1989), 377–400.

53. J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optim.* **1** (1991), 93–113.
54. 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).
55. B. T. Polyak. The conjugate gradient method in extremal problems. *USSR Comput. Math. Math. Phys.* **9** (1969), 94–112.
56. M. Raydan. The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM J. Optim.* **7** (1997), 26–33.
57. T. Serafini, G. Zanghirati, and L. Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optim. Methods Softw.* **20** (2005), 353–378.
58. Ph. L. Toint. An assessment of nonmonotone linesearch techniques for unconstrained optimization. *SIAM J. Sci. Comput.* **17** (1996), 725–739.
59. P. Wolfe. Convergence conditions for ascent methods. *SIAM Rev.* **11** (1969), 226–235.
60. E. K. Yang and J. W. Tolle. A class of methods for solving large, convex quadratic programs subject to box constraints. *Math. Program.* **51** (1991), 223–228.
61. H. Zhang and W. W. Hager. A nonmonotone line search technique and its application to unconstrained optimization. *SIAM J. Optim.* **14** (2004), 1043–1056.
62. J. Zhang and C. Xu. A class of indefinite dogleg path methods for unconstrained minimization. *SIAM J. Optim.* **9** (1999), 646–667.