A new limited memory method for unconstrained nonlinear least squares

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

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

October 16, 2020

Abstract. This paper suggests a new limited memory trust region algorithm for large unconstrained black box least squares problems, called **LMLS**. Main features of **LMLS** are a new non-monotone technique, a new adaptive radius strategy, a new Broyden-like algorithm based on the previous good points, and a heuristic estimation for the Jacobian matrix in an adaptive subspace. Our numerical results show that **LMLS** is robust and efficient, especially in comparison with solvers using traditional limited memory and standard quasi Newton approximations.

Keywords: Black box least squares, limited memory method, trust region method, non-monotone technique

2010 MSC Classification: primary 90C56

Contents

1 Introduction

In this paper, we consider the unconstrained nonlinear least squares problem

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

with high-dimensional $x \in \mathbb{R}^n$ and continuously differentiable $E : \mathbb{R}^n \to \mathbb{R}^r$ $(r \geq n)$, possibly expensive. However, we assume that no derivative information is available.

1.1 Related work

In recent years, there has been a huge amount of literature on least squares and its applications. Here we just list a useful book and paper:

• Ortega & Rheinboldt [35] introduced an excellent book, both covering algorithms and their analysis.

• An excellent paper, both covering Levenberg-Marquardt algorithms, quasi-Newton algorithms, and trust region algorithms and their local analysis without non-singularity assumption, has been introduced by Yuan [39].

Derivative free unconstrained nonlinear black box least squares solvers can be classified in two ways according to how the Jacobian matrix is estimated, and according to whether they are based on line search or on trust region:

• Quasi Newton approximation. **FMINUNC** using the standard quasi Newton approximation is an efficient solver for medium scale problems. It will be useful for small and medium and large scale problems. Sorber et al. [37] introduced **MINLBFGS** (a limited memory BFGS algorithm) and **MINLBFGSDL** (a trust region algorithm using a dogleg algorithm and limited memory BFGS approximation).

• Finite difference approximation. There are many trust region methods using the finite difference method for the Jacobian matrix estimation such as **CoDoSol** and **STRSCNE** by Bellavia et al. [6, 7], **NMPNTR** by Kimiaei [22], **NATRN** and **NATRLS** by Amini et al. [4, 5], **LSQNONLIN** from the Matlab Toolbox, **NLSQERR** (an adaptive trust region strategy) by Deuflhard [10], and **DOGLEG** by Nielsen [31]. They are suitable for small and medium scale problems. Line search methods using the finite difference approximation are **NLEQ** (a damped affine invariant Newton method) by Nowak & Weimann [34] and **MINFNCG** (a family of nonlinear conjugate gradient methods) by Sorber et al. [37].

To solve the least squares problem (1), trust region methods use linear approximations of the residual vectors to make surrogate quadratic models whose accuracy are increased by restricting their feasible points. These methods use a computational meseaure to identify whether an agreement between an actual reduction of the objective function and a predicate reduction of surrogate guadratic model function is good or not. If this agreement is good, the iteration is called successful and the trust region radius is expanded; otherwise, the iteration is called unsuccessful and the trust region radius is reduced, for more details see [8, 33].

The efficiency of trust region methods depends on how the trust region radius is updated (see, e.g., [3, 4, 5, 13, 12, 14, 15, 16, 17, 23, 38]) and whether non-monotone techniques are applied (see, e.g., [1, 2, 3, 4, 5, 9, 18, 19, 22, 23, 38]). Rounding errors may leads two problems:

(i) The model function may not decrease numerically for some iterations. In this case, if there is no decrease in the function value for such iterations, trust region radii are expanded possibly several times which is an unnecessary expansion for them,

(ii) The model function may decrease numerically but the objective function may not decrease in the cases where iterations are near a valley, deep with a small creek at the bottom and steep sides. In this case, trust region radii are reduced possibly many times, leading to the product of quite a small radius, or even a failure.

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

1.2 Our contribution

In this paper, we suggest in Section 2 a new trust region-based limited memory algorithm for unconstrained black box least squares, called **LMLS**. This algorithm uses

• a non-monotone ratio and an adaptive radius formula to quickly reach the minimizer when the valley is narrow;

• a Broyden-like algorithm to get a decrease in the function value when the trust region radius is so small and iteration is unsuccessful;

• a finite difference approximation in an adaptive subspace to estimate the Jacobian matrix;

• either a Gauss-Newton in an adaptive subspace or a dogleg algorithm in an adaptive subspace to solve the trust region subproblems.

Numerical results for small, medium, and large scale problems are given in Section 5 showing the fact that the new method is suitable for large scale problems and is more robust and efficient than solvers using limited memory and standard quasi Newton approximations.

2 The trust region method

In this section, we construct an improved trust region algorithm handling for problems in high dimensions. In Subsection 2.1, a new subspace Gauss-Newton direction is introduced. In Subsection 2.2, a non-monotone term and an adaptive technique are constructed to quickly reach the minimizer in the presence of a narrow valley. A subspace dogleg algorithm is discussed in Subsection 2.3. In Subsection 2.4, a Broyden-like technique is suggested based on the old best points. Our algorithm using new enhancements is introduced in Subsection 2.5.

We write $J(x)$ for the Jacobian matrix of the residual vector E at x. Then the gradient vector is $g(x) := \nabla f(x) := J(x)^T E(x)$ and the Hessian matrix is

$$G(x) := J(x)^T J(x) + \nabla^2 E(x)^T E(x)$$

If the residual vector $E(x)$ is small, the second term in $G(x)$ is small. Hence, we approximate $G(x)$ by the Gauss-Newton Hessian matrix $J(x)^T J(x)$. We define the quadratic surrogate objective function by

$$\mathcal{Q}(p) := \frac{1}{2}\|E + Jp\|^2 := f + p^T g + \frac{1}{2}(Jp)^T Jp, \tag{2}$$

where

$$f := f(x),\ E := E(x),\ J := J(x),\ g := g(x) := J^T E.$$

We denote the kth column of a matrix A by $A_{:k}$.

A trust region method finds a minimizer of the constrained problem

$$\begin{aligned} &\min\ \mathcal{Q}(p) \\ &\text{s.t}\ \ p \in \mathbb{R}^n \ \text{ and } \ \|p\| \leq \Delta, \end{aligned} \tag{3}$$

whose constraint restricts feasible points by the **trust region radius** $\Delta > 0$. This problem is called the **trust region subproblem**. Given the solution p of (3), we define the **actual reduction** in the objective function by

$$df := f - f(x + p), \tag{4}$$

the **predicted reduction** in the model function by

$$dq := \mathcal{Q}(0) - \mathcal{Q}(p). \tag{5}$$

What constitutes an agreement between the actual and predicted reduction around the current iterate x must be measured by the **monotone trust region ratio**

$$\rho := \frac{df}{dq}. \tag{6}$$

If such an agreement is good according to a heuristic formula discussed in Subsection 2.2, the iteration is called **successful**, $x + p$ is accepted as a new point, and the radius is expanded; otherwise, the iteration is called **unsuccessful** and so the radius is reduced.

2.1 A new subspace Gauss-Newton method

In this subsection, we have two goals: estimating the Jacobian matrix in an adaptive subspace and constructing a new subspace Gauss-Newton direction.

Let m_{sn} be the subspace size. The Jacobian matrix in an adaptive subspace is estimated by a new **subspace random finite difference** called **SRFD** using the following steps:

(1) At first, an initial subspace is a random subset of $\{1, \cdots, n\}$ consisting of m_{sn} members and its complementary is $S^c := \{1, \cdots, n\} \setminus S$.

(2) Next, if the complementary of old subspace S^c_{old} is not empty, a new index set $\mathcal{I}$ needs to be identified before a new subspace is determined. In this case, if $\mathcal{I}$ consists of at least m_{sn} members, $\mathcal{I}$ is a random subset of $\{1, \cdots, m_{\text{sn}}\}$ with the $|S^c_{\text{old}}|$ members; otherwise, it is a permutation of $\{1, \cdots, |S^c_{\text{old}}|\}$. Then, a new subspace is determined by $S := S^c_{\text{old}}(\mathcal{I})$ and its complementary is found by $S^c := S^c_{\text{old}} \setminus P$. But if S^c_{old} is empty, a new subspace and its complementary are restarted and chosen in the same way as the initial subspace and its complementary, respectively.

(3) For any $i \in S$,

• the step size is computed by

$$h_i := \begin{cases} \gamma_s & \text{if } x_i = 0, \\ \gamma_s(\operatorname{sign} x_i) \max\left\{|x_i|, \dfrac{\|x\|_1}{n}\right\} & \text{otherwise,} \end{cases}$$

where $0 < \gamma_s < 1$ is a tiny factor and $\operatorname{sign} x_i$ identifies the sign of x_i, taking one of values -1 (if $x_i < 0$), 0 (if $x_i = 0$), and 1 (if $x_i > 0$).

• the random coordinate direction p discussed in [24] is used with the difference that its ith component is updated by $p_i = p_i + h_i$.

• the new trial residual $E(x + \alpha p)$ and the new column $(E(x + \alpha p) - E)/h_i$ of the Jacobian matrix are computed.

It is well known that standard quasi Newton methods are more robust than limited memory quasi Newton ones, but cannot handle for problems in high dimensions; for standard quasi Newton methods, see [20, 21, 32, 36], and for limited memory quasi Newton methods, see [27, 30, 32]. On the other hand, finite difference methods are more efficient than standard quasi Newton ones. Hence, if used in an adaptive subspace, they can be more efficient than limited memory quasi Newton methods for small up to large scale problems.

Using S and S^c generated and updated by **SRFD**, we construct a new **subspace Gauss-Newton direction** by

$$p_i^{\text{sn}} := 0 \quad \text{for } i \in S^c \text{ and } \quad p_S^{\text{sn}} := -(J_{:S}^T J_{:S})^{-1} J_{:S}^T E. \tag{7}$$

2.2 New non-monotone and adaptive strategies

In this subsection, a new non-monotone term – stronger than the objective function f – is constructed and a new adaptive radius formula to update Δ is derived from it. They help **LMLS** in finite precision arithmetic to quickly reach the minimizer in the cases where the valley is deep with a small creek at the bottom and steep sides.

Our non-monotone term is updated not only for successful iterations but also for **unsuccessful** iterations that may be happened before a successful iteration is found. This choice is based on an estimated increase in f defined below which is updated according to whether a decrease in f is found or not. It helps us to generate a somewhat strong non-monotone term when a decrease in f is not found and a somewhat weak non-monotone term otherwise. Somewhat strong non-monotone terms increase the chance of finding a point with better function value or at least a point with a little progress in the function value instead of solving trust region subproblems with high computational costs.

We denote a list of best points by X and a list of corresponding function values by F. Let m_{rs} be the maximum number of good points saved in X. In order to update X and F, we use **updateXF**. If m_{rs} is not exceeded, points with good function values are saved in X and their function values in F. Otherwise, the worst point and its function value are found and replaced by the best point and its function value, respectively.

Let $\gamma_t \in (0,1)$, $\underline{\gamma} \in (0,1)$, $\gamma_{\text{init}} > 0$, and $\overline{\gamma} > 1$ be the tuning parameters and let

$$f_k^{\max} := \max_{i=1:m_{\text{rs}}} \{F_i^k\} \quad \text{for all } k. \tag{8}$$

Before a new non-monotone term is constructed, an estimated increase in f needs to be estimated by

$$\delta_f^k := \begin{cases} \gamma_{\text{init}}|f_0| & \text{if } k = 0 \text{ and } f_0 \in (0,\infty), \\ 1 & \text{if } k = 0 \text{ and } f_0 \in \{-\infty, 0, \infty\}, \\ \dfrac{1}{\overline{\gamma}}(f_{k-1} - f(x_{k-1}+p_{k-1})) & \text{if } k \geq 1 \text{ and } f(x_{k-1}+p_{k-1}) < f_{k-1}, \\ \max(\overline{\gamma}\delta_f^{k-1}, \underline{\gamma}(|f(x_{k-1}+p_{k-1})| + |f_k^{\max}|)) & \text{if } k \geq 1 \text{ and } f(x_{k-1}+p_{k-1}) \geq f_{k-1}. \end{cases} \tag{9}$$

Accordingly, the new non-monotone formula is defined by

$$f_k^{\text{nm}} := \begin{cases} f_0 & \text{if } k = 0, \\ f_k + \delta_f^k & \text{if } k \geq 1 \end{cases} \tag{10}$$

and the new adaptive radius is constructed by

$$\Delta_k^{\text{nm}} := \lambda_k \sqrt{f_k^{\text{nm}}}, \tag{11}$$

where $\Delta_0^{\text{nm}} > 0$ is a tuning parameter and λ_k is updated according to

$$\lambda_k := \begin{cases} \sigma_1 \lambda_{k-1}, & \text{if } \rho_{k-1}^{\text{nm}} < \gamma_t, \\ \min(\overline{\lambda}, \max(\sigma_2 \lambda_{k-1}, \underline{\lambda})), & \text{otherwise.} \end{cases} \tag{12}$$

Here $\lambda_0 > 0$, $0 < \sigma_1 < 1 < \sigma_2$ and $\overline{\lambda} > \underline{\lambda} > 0$ are the tuning parameters and the new non-monotone trust region ratio is defined by

$$\rho_{k-1}^{\text{nm}} := \frac{f_{k-1}^{\text{nm}} - f(x_{k-1}+p_{k-1})}{\widetilde{\mathcal{Q}}_{k-1}(0) - \widetilde{\mathcal{Q}}_{k-1}(p_{k-1})}, \tag{13}$$

where p_{k-1} is a solution of the following trust region suproblem in an adaptive subspace S

$$\begin{aligned} &\min\ \widetilde{\mathcal{Q}}_{k-1}(p) := \frac{1}{2}\|E_{k-1} + J_{:S}^{k-1}p\|^2 := f_{k-1} + p^T g_S^{k-1} + \frac{1}{2}(J_{:S}^{k-1}p)^T J_{:S}^{k-1}p \\ &\text{s.t}\ \ p \in \mathbb{R}^{m_{\text{sn}}} \ \text{ and } \ \|p\| \leq \Delta_{k-1}^{\text{nm}} \end{aligned} \tag{14}$$

with $f_{k-1} := f(x_{k-1})$, $E_{k-1} := E(x_{k-1})$, $J_{:S}^{k-1} := J_{:S}(x_{k-1})$, and $g_S^{k-1} := (J_{:S}^{k-1})^T E_{k-1}$.

2.3 A subspace dogleg algorithm

We define the **Cauchy step** by

$$p^{\text{c}} := -t^* g_S, \quad t^* := \text{argmin}\{\widetilde{\mathcal{Q}}(-t g_S) \mid t \geq 0,\ \|t g_S\| \leq \Delta^{\text{nm}}\}. \tag{15}$$

The goal is to solve the trust region subproblem (14) such that

$$\|p\| \leq \Delta^{\rm nm} \quad \text{and} \quad \widetilde{\mathcal{Q}}(p) \leq \widetilde{\mathcal{Q}}(p^{\rm c}) \tag{16}$$

hold. After the subspace Gauss-Newton direction is computed by (7), if it is outside a trust region, a **subspace dogleg algorithm**, called **subDogleg**, is used resulting in an estimated step enforcing (16).

The model function $\widetilde{\mathcal{Q}}$ is reduced by (15) if $dq^{\rm sn} := \widetilde{\mathcal{Q}}(0) - \widetilde{\mathcal{Q}}(p^{\rm sn}) > 0$. **subDogleg** first identifies whether $dq^{\rm sn} > 0$ or not. Then we have one of the following cases:

Case 1. If $dq^{\rm sn} > 0$, the **scaled steepest descent step**

$$p^{\rm sd} := -\frac{g_S^T g_S}{(J_S g_S)^T (J_S g_S)} g_S \tag{17}$$

is computed. If it is outside the trust region, an estimated solution of (3) is either the Cauchy step computed by (15) or the **dogleg step**

$$p^{\rm dg} := p^{\rm sd} + t(p^{\rm sn} - p^{\rm sd}); \tag{18}$$

both of (17) and (18) are on the trust region boundary. Here t is found by solving the equation $\|p^{\rm sd} + t(p^{\rm sn} - p^{\rm sd})\| = \Delta^{\rm nm}$. If the condition $dp := (p^{\rm sd})^T(p^{\rm sn} - p^{\rm sd}) \leq 0$ holds, a positive root is computed by

$$t := \frac{-dp + \sqrt{dp^2 + \|p^{\rm sn} - p^{\rm sd}\|((\Delta^{\rm nm})^2 - \|p^{\rm sd}\|^2)}}{\|p^{\rm sn} - p^{\rm sd}\|^2} \in (0, 1). \tag{19}$$

Otherwise, t is computed by

$$t := \frac{(\Delta^{\rm nm})^2 - \|p^{\rm sd}\|^2}{dp + \sqrt{dp^2 + \|p^{\rm sn} - p^{\rm sd}\|((\Delta^{\rm nm})^2 - \|p^{\rm sd}\|^2)}} \in (0, 1); \tag{20}$$

e.g., see [31].

Case 2. If $dq^{\rm sn} \leq 0$, the model function $\widetilde{\mathcal{Q}}$ is convex since the matrix $(J_S g_S)^T (J_S g_S)$ is symmetric and positive semidefinite. An estimated solution of (3) is either $p^{\rm sd}$ computed by (17) if it is inside the trust region or the Cauchy step $p := \Delta^{\rm nm}(p^{\rm sd}/\|p^{\rm sd}\|)$ according to $p^{\rm sd}$, otherwise.

2.4 Broyden-like technique

Before a successful iteration is found by a trust region algorithm, the trust region subproblems may be solved many times with high computational cost. Instead, our idea is to use a new algorithm based on the previous best points in the hope of finding a point with good function value.

Whenever **LMLS** cannot decrease the function value, a new Broyden-like technique, called **BroydenLike**, is used in the hope of getting a decrease in the function value. Let

$x_1, \cdots, x_{m_{\rm rs}}$ be the $m_{\rm rs}$ best point stored in X. Then a point in the affine space spanned by such points has the following form

$$x_z := Xz, \quad z \in \mathbb{R}^{m_{\rm rs}}, \quad e^T z = 1, \tag{21}$$

where $e \in \mathbb{R}^{m_{\rm rs}}$ is a vector all of whose components are one. Given $B := \begin{pmatrix} X \\ e \end{pmatrix}$, the linear approximation $E(x_z) \approx Bz$ is used to replace (1) by the surrogate problem

$$\begin{array}{ll} \min & \dfrac{1}{2}\|Bz\|_2^2 \\ \text{s.t.} & e^T z = 1. \end{array} \tag{22}$$

This is a quality constrained convex quadratic problem in $m_{\rm rs}$ variables; hence can be solved in closed form. Then a QR factorization is made in the form $B = QR$, where Q is an orthogonal matrix and R is a square upper triangular matrix. By setting $Z := R^{-1}$, we make the substitution $z := Zy$, define $a^T := e^T Z$, and obtain the m-dimensional minimal norm linear feasibility problem

$$\begin{array}{ll} \min & \dfrac{1}{2}\|y\|_2^2 \\ \text{s.t.} & a^T y = 1 \end{array} \tag{23}$$

whose solution is $y = a/\|a\|_2$. Hence, a new trial point for the next algorithm is

$$x_{\rm trial} := x_z := XR^{-1}a/\|a\|_2. \tag{24}$$

BroydenLike tries to find a point with better function value when no decrease in f is found along $\pm p$. It takes X, F, x, E, B, f, δ_f, and $f^{\rm nm}$ as input and uses the following tuning parameter: $\underline{\gamma} \in (0,1)$ (tiny factor for adjusting δ_f), $\overline{\gamma} > 1$ (parameter for expanding δ_f), γ_s (tiny parameter for the finite difference step size), $m_{\rm rs}$ (memory for affine space), and $0 < \gamma_r < 1$ (tiny parameter for adjusting the scaled random directions). It returns a new δ_f and $f^{\rm nm}$ (and f, X, F, E, S, and J_S if a decrease in f is found) as output.

2.1 Algorithm. (**BroydenLike**, Broyden-like method)

(S0) Construct a QR factorization for the matrix B and get the square upper triangular matrix R.

(S1) **A new trial point.** Compute the new trial point $x_{\rm trial}$ by (24) and $E_{\rm trial} := E(x_{\rm trial})$. Then set $f_{\rm trial} := 0.5\|E_{\rm trial}\|^2$. If $f_{\rm trial} \le f^{\rm nm}$, then

(1) set $x := x_{\rm trial}$, $E := E_{\rm trial}$, and $f := f_{\rm trial}$;

(2) save x in X and f in F by **updateXF**;

(3) compute S and $J_S := J_{:S}(x)$ by **SRFD**;

Then update δ_f by (9) and compute $f^{\rm nm}$ by (10).

Since the Jacobian matrix may be singular or indefinite, a new point may move toward a maximizer. To remedy this disadvantage, **BroydenLike** does not lead to accept such a point with largest function value; like a maximizer.

2.5 A limited memory trust region algorithm

We describe all steps of a new **limited memory algorithm**, called **LMLS** using the new subspace direction (7), the new non-monotone technique (10), the new adaptive radius strategy (11), and **BroydenLike**.

In each iteration, an estimated solution of the trust region subproblem (14) is found. Whenever the condition $\rho^{\rm nm} \geq \gamma_t$ holds, the iteration is called to be **successful** while updating both the non-monotone term (10) and adaptive radius formula (11), and estimating the Jacobian matrix in an adaptive subspace by **SRFD**. Otherwise, the iteration is called to be unsuccessful. In this case, **BroydenLike** is performed in the hope of finding a decrease in the function value. If a decrease in the function value is found, the iteration becomes successful; otherwise, it remains unsuccessful while reducing the radius and updating the non-monotone term (10) until a decrease in the function value is found and the iteration becomes successful.

LMLS solves a unconstrained nonlinear black box least squares problems. This algorithm takes the initial point x, maximal time in seconds (`secmax`), and maximal number of function evaluations (`nfmax`). It uses the following tuning parameters: $m_{\rm sn}$ (subspace size), $\gamma_t \in (0,1)$ (parameter for trust region), $0 < \sigma_1 < 1 < \sigma_2$ (parameters for updating λ), $\gamma_{\rm init}$ (parameter for updating the initial δ_f), $\underline{\gamma} \in (0,1)$ (tiny factor for adjusting δ_f), $\overline{\gamma} > 1$ (parameter for expanding δ_f), $\underline{\lambda}$ (lower bound for λ), $\overline{\lambda}$ (upper bound for λ), γ_s (tiny parameter for adjusting finite difference step sizes), and $0 < \gamma_r < 1$ (tiny parameter for adjusting random coordinate directions). It returns a solution of a nonlinear least squares problem as output.

2.2 Algorithm. (**LMLS**, limited memory method for nonlinear least squares problems)

(S0) **Initialization:**

(1) Choose $\lambda \in (\underline{\lambda}, \overline{\lambda})$.

(2) Compute $E := E(x)$ and set $f := 0.5\|E\|^2$. Then set $X := x$ and $F := f$.

(3) Compute the initial subspace S and $J_{:S}(x)$ by **SRFD**.

(4) Compute $g_S := J_{:S}^T E$ and δ_f by (9).

(S1) Compute the subspace Gauss-Newton direction $p := p^{\rm sn}$ by (7).

(S2) If $\|p^{\rm sn}\| > \Delta^{\rm nm}$, **obtain an approximated solution** p **of** (14) by calling **subDogleg**, which is a variant of **dogleg.m**, available at

`http://www.math.ubc.ca/~loew/m604/mfiles.htm`

but with the difference that

(1) t is recomputed by (20) when t computed by (19) is not positive,

(2) it can be handled for problems in high dimensions,

(3) the concavity of $\widetilde{\mathcal{Q}}$ is ignored,

(4) an adaptive subspace is used.

(S3) If $g^T p \geq 0$, which may happen, e.g., due to rounding errors, we modify the search direction according to $\texttt{ind} := \{i \mid g_i p_i > 0\}$ and $p_{\texttt{ind}} := -p_{\texttt{ind}}$ so that the **descent condition** $g^T p < 0$ holds.

(S4) **A new trial point:**

(1) Compute $x_{\text{trial}} := x + p$, $E_{\text{trial}} := E(x_{\text{trial}})$, and $f_{\text{trial}} := 0.5\|E_{\text{trial}}\|^2$.

(2) If either `secmax` or `nfmax` is exceeded, **LMLS** ends.

(3) Otherwise, compute ρ^{nm} by (13); if $\rho^{\text{nm}} < \gamma_t$, go to (S5); otherwise, go to (S6).

(S5) **Unsuccessful iteration:**

(1) Call **BroydenLike**. If a decrease in f is found go to (S6).

(2) Update δ_f by (9) and f^{nm} by (10).

(3) Reduce λ to (12) and the radius Δ^{nm} to (11). Then go to (S2).

(S6) **Successful iteration:**

(1) Set $x := x_{\text{trial}}$, $f := f_{\text{trial}}$, and $E := E_{\text{trial}}$.

(2) Compute the new subspace S and $J_{:S}(x)$ by **SRFD**. Then compute $g_S := J_{:S}^T E$.

(3) Update δ_f by (9) and f^{nm} by (10).

(4) Expand λ by (12) and Δ^{nm} by (11).

(5) Store x in X and f in F by **updateXF**.

Then go to (S1).

LMLS was implemented in Matlab; the source code is available at

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

3 Numerical results

We updated the test environment constructed by Kimiaei & Neumaier [26] to use test problems suggested by Lukšan et al [28]. **LMLS** is compared with unconstrained least squares and unconstrained optimization solvers, for some of which we had to choose options different from the default to make them competitive in the first subsection.

3.1 Codes compared

Least squares solvers:

- **CoDoSol** is a solver for constrained nonlinear systems of equations, obtained from

  `http://codosol.de.unifi.it`.

  It combines Newton method and a trust-region method, see [7]. The following option used

  $$\texttt{parms} = [\texttt{maxit}, \texttt{maxnf}, \texttt{tr}, \texttt{delta}, \texttt{scaling}, \texttt{outflag}] = [\texttt{inf}, \texttt{nfmax}, 1, -1, 0, 2].$$

  Note that $\texttt{delta} = -1$ means that $\Delta_0 = 1$. According to our numerical results, **CoDoSol** was not sensitive for the initial radius; hence, the default used.

- **STRSCNE** is a solver for constrained nonlinear systems of equations, obtained from

  `http://codosol.de.unifi.it`.

  It combines Newton method and a trust-region procedure, see [6]. The option $\texttt{parms} = [\texttt{maxit}, \texttt{maxnf}, \texttt{delta}, \texttt{outflag}] = [\texttt{inf}, \texttt{nfmax}, -1, 0]$ used. Note that $\texttt{delta} = -1$ means that $\Delta_0 = 1$. According to our numerical results, **STRSCNE** was not sensitive for the initial radius; hence, the default used for Δ_0.

- **NLEQ1** is a damped affine invariant Newton method for nonlinear systems, obtained from

  `http://elib.zib.de/pub/elib/codelib/nleq1_m/nleq1.m`

  and suggested by Deuflhard [10] and written by Nowak & Weimann [34]. The default tuning parameters used; only we selected $\texttt{iopt.jacgen} = 2$, $\texttt{iopt.qrank1} = 1$ and $\texttt{wk.fmin} = \texttt{1e} - \texttt{50}$.

- **NLEQ2** is the same as **NLEQ1**; only we selected $\texttt{iopt.qrank1} = 0$.

- **MINLBFGS** is a L-BFGS with line search algorithm, obtained from

  `https://www.tensorlab.net/`

  and written by Sorber et al. [37]. The default parameters used, except for m. **MINLBFGS1** and **MINLBFGS2** denote **MINLBFGS** with $m = \min(n, 30)$ and $m = \min(n, 100)$, respectively.

- **MINLBFGSDL** is a L-BFGS with dogleg trust region algorithm with the option set $\texttt{MaxIter} = \texttt{nfmax}$, $\texttt{TolFun} = \texttt{1e} - \texttt{50}$, $\texttt{TolX} = \texttt{1e} - \texttt{50}$. The default parameters used for `PlaneSearch` and M. **MINLBFGSDL1**, **MINLBFGSDL2**, **MINLBFGSDL3**, and **MINLBFGSDL4** denoted **MINLBFGSDL** with

  (1) $\texttt{PlaneSearch} = \texttt{false}$ and $M = \min(30, n)$,

  (2) $\texttt{PlaneSearch} = \texttt{false}$ and $M = \min(100, n)$,

  (3) $\texttt{PlaneSearch} = \texttt{true}$ and $M = \min(30, n)$,

  (4) $\texttt{PlaneSearch} = \texttt{true}$ and $M = \min(100, n)$.

  According to our results, **MINLBFGSDL1** was the best.

- **MINFNCG** is a nonlinear conjugate gradient solver, obtained from

  `https://www.tensorlab.net/`

  and written by Sorber et al. [37]. It used the following option set

  $$\texttt{MaxIter} = \texttt{nfmax};\ \ \texttt{TolFun} = \texttt{1e} - \texttt{50};\ \ \texttt{TolX} = \texttt{1e} - \texttt{50}.$$

  The other tuning parameter was $\texttt{Beta} \in \{\texttt{HS}, \texttt{HSm}, \texttt{PR}, \texttt{FR}, \texttt{PRm}, \texttt{SD}\}$. **MINFNCG1**, **MINFNCG2**, **MINFNCG3**, **MINFNCG4**, **MINFNCG5**, and **MINFNCG6** denoted **MINFNCG** with $\texttt{Beta} = \texttt{HS}$, $\texttt{Beta} = \texttt{HSm}$, $\texttt{Beta} = \texttt{PR}$, $\texttt{Beta} = \texttt{FR}$, $\texttt{Beta} = \texttt{PRm}$, and $\texttt{Beta} = \texttt{SD}$, respectively.

- **NLSQERR** is a global unconstrained Gauss-Newton method with error oriented convergence criterion and adaptive trust region strategy [10], obtained from

  `http://elib.zib.de/pub/elib/codelib/NewtonLib/index.html`

  The following options used

  $$\begin{aligned}&\texttt{iniscalx} = 0;\ \ \texttt{rescalx} = 0;\ \ \texttt{xthrsh} = \texttt{ones}(n, 1);\\ &\texttt{xtol} = \texttt{1.e} - \texttt{50};\ \ \texttt{ftol} = \texttt{1.e} - \texttt{50};\ \ \texttt{kmax} = \texttt{nfmax};\\ &\texttt{printmon} = 2;\ \ \texttt{printsol} = 1;\ \ \texttt{fid} = 1;\ \ \texttt{numdif} = 1;\\ &\texttt{lambda0} = \texttt{eps};\ \ \texttt{lambdamin} = \texttt{1e} - \texttt{50};\ \ \texttt{ftol} = \texttt{1.e} - \texttt{50}.\end{aligned}$$

- **NMPNTR**, nonmonotone projected Newton trust region method, is a bound constrained solver [22]. **NMPNTR1**, **NMPNTR2**, **NMPNTR3**, and **NMPNTR4** denoted **NMPNTR** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. According to our results, **NMPNTR2** was the best.

- **NATRN** is a nonmonotone trust region algorithm [4] using the full finite difference approximation. The subproblem solved in the same way as **LMLS**. **NATRN1**, **NATRN2**, **NATRN3**, and **NATRN4** denoted **NATRN** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. According to our results, **NATRN1** had the best performance.

- **NATRLS** is a nonmonotne line search and trust region algorithm [5] using the full finite difference approximation. The subproblem solved in the same way as **LMLS**. **NATRLS1**, **NATRLS2**, **NATRLS3**, and **NATRLS4** denoted **NATRLS** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. According to our results, **NATRLS1** had the best performance.

- **LSQNONLIN1**, obtained from the Matlab Optimization Toolbox at,

  `https://de.mathworks.com/help/optim/ug/lsqnonlin.html`

  is nonlinear least-squares solver with the following options:

  `options = optimoptions(@lsqnonlin,'Algorithm', 'levenberg-marquardt', 'FiniteDifferenceType','forward', 'MaxIter', Inf,'MaxFunEvals', nfmax, 'TolX', 0,'SpecifyObjectiveGradient','false').`

- **LSQNONLIN2**, obtained from the Matlab Optimization Toolbox at,

  `https://de.mathworks.com/help/optim/ug/lsqnonlin.html`

  is nonlinear least-squares solver with the following options:

  `options = optimoptions(@lsqnonlin,'Algorithm', 'trust-region reflective', 'FiniteDifferenceType','forward','MaxIter', Inf,'MaxFunEvals', nfmax, 'TolX', 0,'SpecifyObjectiveGradient','false')`.

- **DOGLEG** is Powell's dogleg method for least squares problems, which is the best algorithm from the toolbox of `immoptibox.zip` [31], available at

  `http://www2.imm.dtu.dk/projects/immoptibox/`

  The following option used

  $$\texttt{opts} = [\Delta_0, \texttt{tolg}, \texttt{tolx}, \texttt{tolr}, \texttt{maxeval}] = [\Delta_0, \texttt{1e}-\texttt{50}, \texttt{1e}-\texttt{50}, \texttt{1e}-\texttt{50}, \texttt{nfmax}].$$

  DOGLEG1, **DOGLEG2**, **DOGLEG3**, and **DOGLEG4** denoted **DOGLEG** with $\Delta_0 = 1$, $\Delta_0 = 10$, $\Delta_0 = 100$, and $\Delta_0 = 500$, respectively. The best version was **DOGLEG1**.

Unconstrained solvers:

- **FMINUNC**, obtained from the Matlab Optimization Toolbox at

  `https://ch.mathworks.com/help/optim/ug/fminunc.html`,

  is a standard quasi-Newton algorithm. We used **FMINUNC** with the options set

  `opts = optimoptions(@fminunc),'Algorithm','quasi-newton', 'Display', 'Iter', 'MaxIter',Inf,'MaxFunEvals', nfmax, 'TolX', 0,'TolFun',0,'ObjectiveLimit',-1e-50)`.

- **FMINUNC1** is **FMINUNC** with the limited memory quasi Newton approximation by Liu & Nocedal [27]. It has been added to **FMINUNC** by the present authors. The option set for it is the same as **FMINUNC**; only the memory $m = 10$ added to the option set.

3.2 Default for tuning parameters of LMLS

The tuning parameters for our new method (**LMLS**) are chosen as

$$\begin{array}{l} m_{\text{nm}} = 10;\ \gamma_r = 10^{-30};\ \gamma_p = 5\varepsilon_m;\ \gamma_s = \sqrt{\varepsilon_m};\ \gamma_t = 0.1;\ \sigma_1 = 0.5; \\ \sigma_2 = 1;\ \gamma_{\text{init}} = 10^{-8};\ \gamma_m = 10^{-30};\ \underline{\lambda} = 10^{-4};\ \Delta_0 = 10;\ \Delta_{\min} = 10^{-6}; \\ \lambda_0 = 1;\ \overline{\lambda} = 10^5; \end{array}$$

The remaining tuning parameter m_{sn} is varied in the experiment: **LMLS1**, **LMLS2**, **LMLS3**, and **LMLS4** denote **LMLS** with $m_{\text{sn}} = 3$, $m_{\text{sn}} = \min(10, n)$, $m_{\text{sn}} = \min(30, n)$,

and $m_{\rm sn} = \min(100, n)$, respectively.

3.3 Test problems used, initial point and stopping tests

Test problems suggested by LUKŠAN et al. [28] are classified in Table 2 according to whether they are **overdetermined** ($r > n$) or not ($r = n$). A shifted point for these problems is done like [25] as $\chi_i := (-1)^{i-1}2/(2+i)$ for all $i = 1, \ldots, n$. This means that the initial point is chosen by $x_i^0 = \chi_i$ for all $i = 1, \cdots, n$ and the initial function value is computed by $f^0 := f(x^0)$ while the other function values are computed by $f^\ell := f(x^\ell + \chi)$ for all $\ell \geq 1$.

Table 2: A classification of test problems

Dimensions (n)	3	5	10	16	30	50	100	300	500	1000	5000	10000
Total number of least squares problems ($r \geq n$)	49	69	76	83	83	83	83	83	83	83	83	83
Number of square problems ($r = n$)	43	53	55	62	62	62	62	62	62	62	62	62
Number of least squares problems with $r > n$	6	16	21	21	21	21	21	21	21	21	21	21

We denote `nf` and `msec` as the number of function evaluations and time in milliseconds, respectively. `nfmax` and `secmax` are the upper bounds for them, chosen as

$$\texttt{nfmax} \in \begin{cases} \{10n, 50n, 100n, 500n\} & \text{if } 1 \leq n \leq 100, \\ \{10n, 50n, 100n\} & \text{if } 101 \leq n \leq 1000 \\ \{10n, 100n\} & \text{if } 1001 \leq n \leq 10000 \end{cases}$$

and

$$\texttt{secmax} := \begin{cases} 300 & \text{if } 1 \leq n \leq 100, \\ 800 & \text{if } 101 \leq n \leq 10000. \end{cases}$$

Let $f_{\rm init}$ be the function value of the starting point (common to all solvers) and $f_{\rm best}$ be the best point known to us. Then, if the target accuracy satisfies

$$q_f := (f - f_{\rm best})/(f_{\rm init} - f_{\rm best}) \leq \begin{cases} 10^{-8} & \text{if } 1 \leq n \leq 100, \\ 10^{-3} & \text{if } 101 \leq n \leq 10000, \end{cases}$$

then the problem is solved. Otherwise, the problem is unsolved; either `nfmax` or `secmax` is exceeded, or the solver fails.

3.4 Summarizing tables and plots

For a given collection S of solvers, the strength of a solver $so \in S$ – relative to an ideal solver that matches on each problem the best solver – is measured, for any given cost measure c_s by the number, q_{so} defined by

$$q_{so} := \begin{cases} \dfrac{\min\limits_{s \in S} c_s}{c_{so}}, & \text{if } so \text{ solved the problem,} \\ 0, & \text{otherwise,} \end{cases}$$

called the **efficiency** of the solver *so* with respect to this cost measure; two cost measures `nf` and `msec` are used. Efficiencies are given as percentages, rounded (towards zero) to integers. Larger efficiencies imply a better average behaviour while a zero efficiency indicates failure.

#100 denotes the total number of test problems in which the solver needed the least number `nf` of function evaluations, and !100 the total number of test problems where the solver was the only one needing this many function evaluations.

We denote the time in seconds without the setup time for the objective function by `sec`. In tables, a sign

- n indicates that `nf` $\geq$ `nfmax` was reached.
- t indicates that `sec` $\geq$ `secmax` was reached.
- f indicates that the algorithm failed for other reasons.

The mean of total time

$$T_{\text{mean}} := \frac{\sum \text{ solved}}{\#\text{ solved}} \quad \text{(in msec)}$$

is computed regardless of the time for unsolved problems. It can be a good measure when solvers have approximately the same number of solved problems.

We use two different performance plots:

• The first performance plot is for `nf`/(`best nf`) and `msec`/(`best msec`); the percentage of problems solved within a factor τ of the best solvers; see [11].

• The second performance plot is for `nf`/(`best nf`) and `msec`/(`best msec`) as well; but it is the percentage of problems solved within the number of function evaluations and time in milliseconds; discussed in [29] by Moré & Wild.

All efficiency tables and performance plots are given in Appendix A.

3.5 Small scale: $n \in [1, 100]$

A comparison among **LMLS1**, **LMLS2**, **LMLS3**, **LMLS4**, and solvers using quasi Newton is shown in Subfigures (a) and (b) of Figure 1, so that

• **LMLS4** using the full estimated Jacobian matrix is the best in terms of the number of solved problems and the `nf` efficiency;

• **LMLS3**, **LMLS2**, and **LMLS1** are more efficient than solvers using quasi Newton approximation (**FMINUNC**, **FMINUNC1**, **MINFLBFGS1**, and **MINFLBFGSDL1**) in terms of the `nf` efficiency;

• **FMINUNC** and **MINFLBFGSDL1** are comparable with **LMLS3** – only for very large budget – in terms of the number of solved problems but **LMLS3**, **LMLS2**, and **LMLS1**

are more efficient than **FMINUNC** and **MINFLBFGSDL1** in terms of the `nf` efficiency not only for very large budget but also for small up to large budgets.

To determine whether our new non-monotone and adaptive radius strategies are effective or not, we compare **LMLS4** with solvers using other non-monotone and adaptive radius strategies, shown in Subfigures (c) and (d) of Figure 1. All solvers use the full Jacobian matrix and the trust region subproblems are solved by the same algorithm. As can be seen, **LMLS4** is much more efficient and robust than **NATRLS1**, **NMPGTR2**, and **NATRN1** in terms of the number of solved problems and the `nf` efficiency.

We compare **LMLS4** with four famous solvers **LSQNONLIN1**, **CoDoSol1**, **NLEQ1**, and **DOGLEG1** shown in Subfigures (e) and (f) of Figure 1. It is seen that **LMLS4** and **CoDoSol1** are the two best solvers in terms of the `nf` efficiency while **LMLS4** and **DOGLEG1** are the two best solvers in terms of the number of solved problems.

Another comparison is among **LMLS3**, **LMLS2**, and **LMLS1** using the Jacobian matrix in adaptive subspace and **LSQNONLIN1** and **NLEQ1** using the full Jacobian matrix. We conclude from Subfigures (g) and (h) of Figure 1 that

• **LMLS3** is the best in terms of the number of solved problems and the `nf` efficiency;

• **LMLS2** is the second best solver in terms of the number of solved problems and the `nf` efficiency for medium, large, and very large budgets;

• **LMLS1** with lowest subspace size is more efficient than **LSQNONLIN1** in terms of the number of solved problems and the `nf` efficiency; even it is more efficient than **NLEQ1** for very large budget in terms of the `nf` efficiency.

As a result, **LMLS** is competitive for small scale problems in comparison with the state-of-the-art solvers.

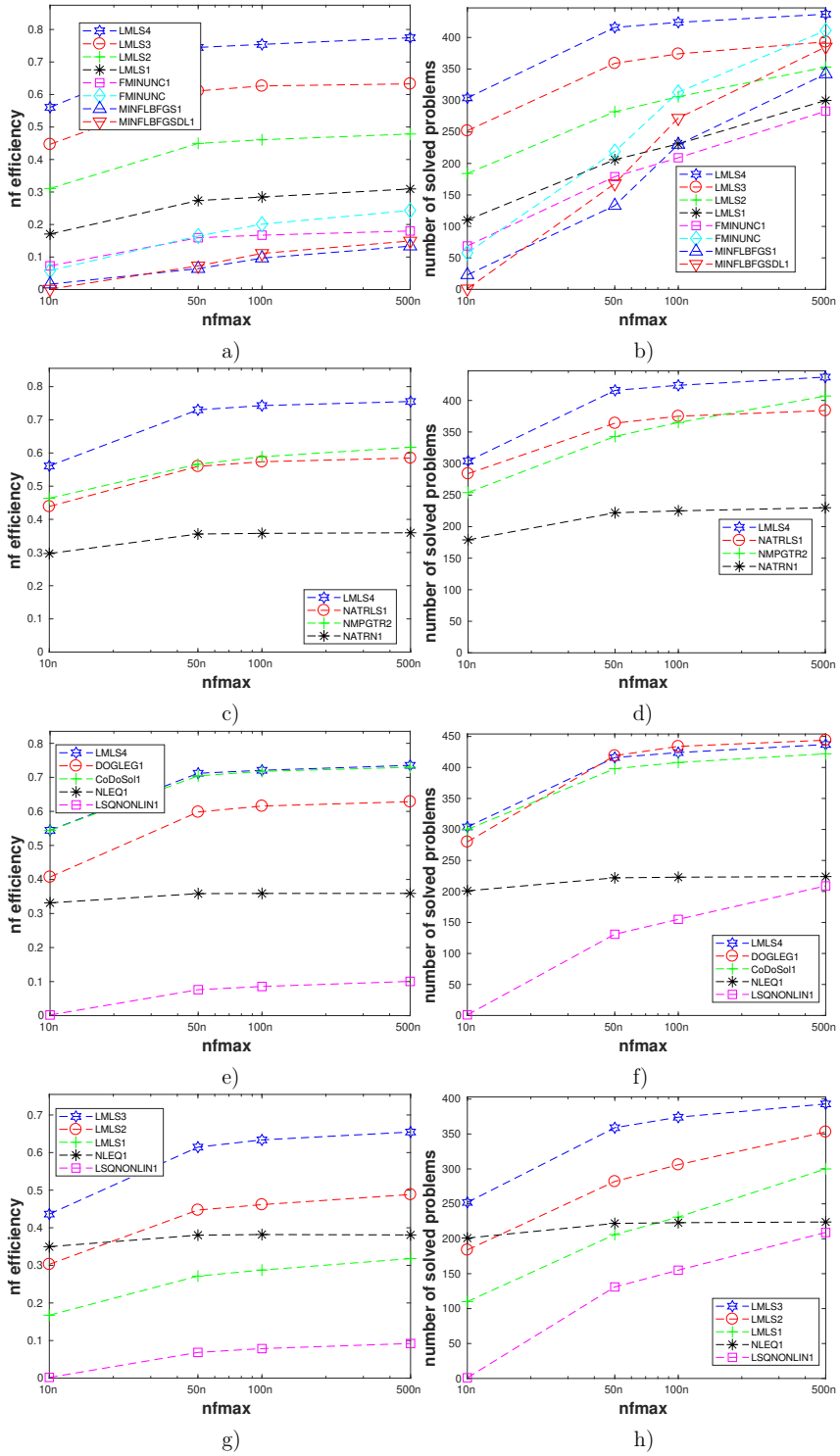


Figure 1: Performance plots for small scale problems. (a) – (b): A comparison of limited memory solvers, (c) – (d): A comparison among **LMLS** in a full subspace and solvers using other non-monotone and adaptive radius techniques, (e) – (f): A comparison among **LMLS** in a full subspace and other famous solvers, (g) – (h): A comparison among low-dimensional **LMLS1**, **LMLS2**, **LMLS3** and **NLEQ1** and **LSQNONLIN1** using full estimated Jacobian.

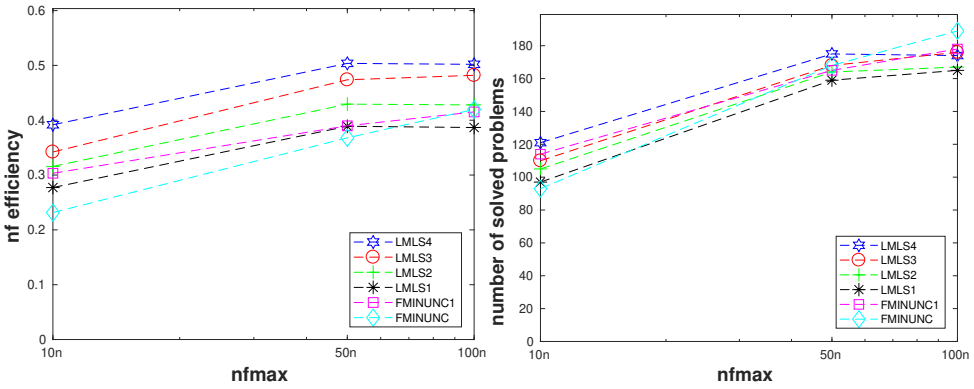
3.6 Medium scale: $n \in [101, 1000]$

In this subsection, we compare **LMLS1**, **LMLS2**, **LMLS3**, **LMLS4** using the estimated Jacobian matrices in an adaptive subspace with **FMINUNC** using standard BFGS approximations and **FMINUNC1** using limited memory BFGS ones.

From Subfigures (a) and (b) of Figure 1, we conclude that

• **LMLS4**, **LMLS3**, and **LMLS2** are the three best solvers in terms of the `nf` efficiency, respectively;

• **LMLS4** is the best solver in terms of the number of solved problems; only **FMINUNC** is the best for large budget.



a) b)

Figure 2: (a) – (b): Performance plots for medium scale problems

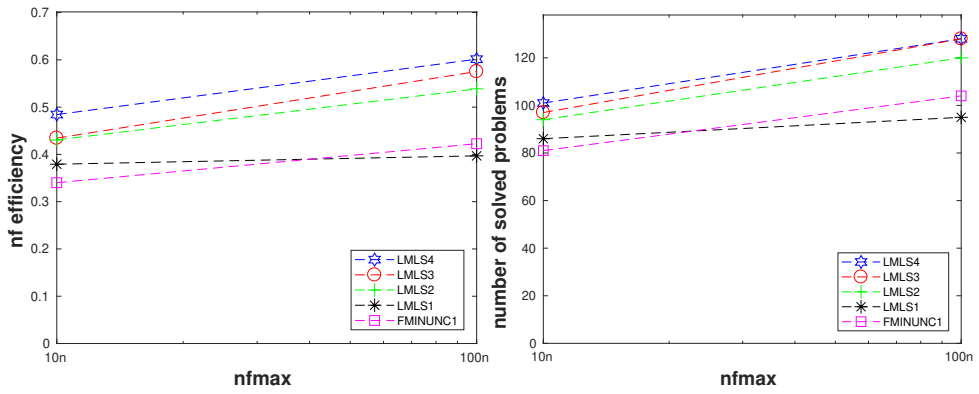
3.7 Large scale: $n \in [1001, 10000]$

In this subsection, we compare **LMLS1**, **LMLS2**, **LMLS3**, **LMLS4** using the estimated Jacobian matrices in an adaptive subspace with **FMINUNC1** using limited memory BFGS approximations.

In terms of the `nf` efficiency and the number of solved problems, Subfigures (a) and (b) of Figure 3 result in the fact that

• **LMLS4**, **LMLS3**, and **LMLS2** are the three best solvers, respectively;

• **LMLS1** with lowest subspace size is more efficient than **FMINUNC1** for small budget while **FMINUNC1** is more efficient than **LMLS1** for large budget.



a) b)

Figure 3: (a) – (b): Performance plots for medium scale problems

Acknowledgement 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.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest

Human and animal rights This study does not contain any studies with human participants or animals performed by any of the authors.

A Appendix: tables and plots

A.1 Small scale: $n \in [1, 100]$

Table 3: Results for small scale and small budget

stopping test:	$q_f \leq$ 1e-08,	sec $\leq$ 300,	nf $\leq$ 10*n							
367 of 526 problems without bounds solved									mean efficiency in %	
dim∈[1,100]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmtr4**	304	204	15	52	222	0	0	53	46
CoDoSol1	**codo1**	300	197	30	56	219	0	7	52	40
NATRLS1	**natrs1**	284	63	8	67	242	0	0	40	30
DOGLEG1	**dogleg1**	280	70	11	98	246	0	0	38	22
NMPGTR2	**nmpg2**	254	172	8	55	272	0	0	43	31
LMLS3	**lmtr3**	252	156	4	53	274	0	0	42	35
NLEQ1	**nleq1**	201	42	30	77	145	0	180	33	16
LMLS2	**lmtr2**	184	91	7	75	342	0	0	29	22
NATRN1	**natrn1**	179	54	11	89	347	0	0	26	18
LMLS1	**lmtr1**	110	47	20	112	416	0	0	16	10
STRSCNE1	**strs1**	70	69	0	28	107	0	349	13	7
FMINUNC1	**func1**	69	2	0	123	455	0	2	6	4
FMINUNC	**func**	58	2	0	139	468	0	0	5	4
MINFLBFGS1	**lbfgs1**	23	0	0	227	449	0	54	1	0
LSQNONLIN1	**lsqn1**	1	1	1	50	525	0	0	0	0
MINFLBFGSDL1	**minlbfgs1**	1	0	0	30	525	0	0	0	0
MINFNCG1	**MINFNCG1**	0	0	0	–	514	0	12	0	0

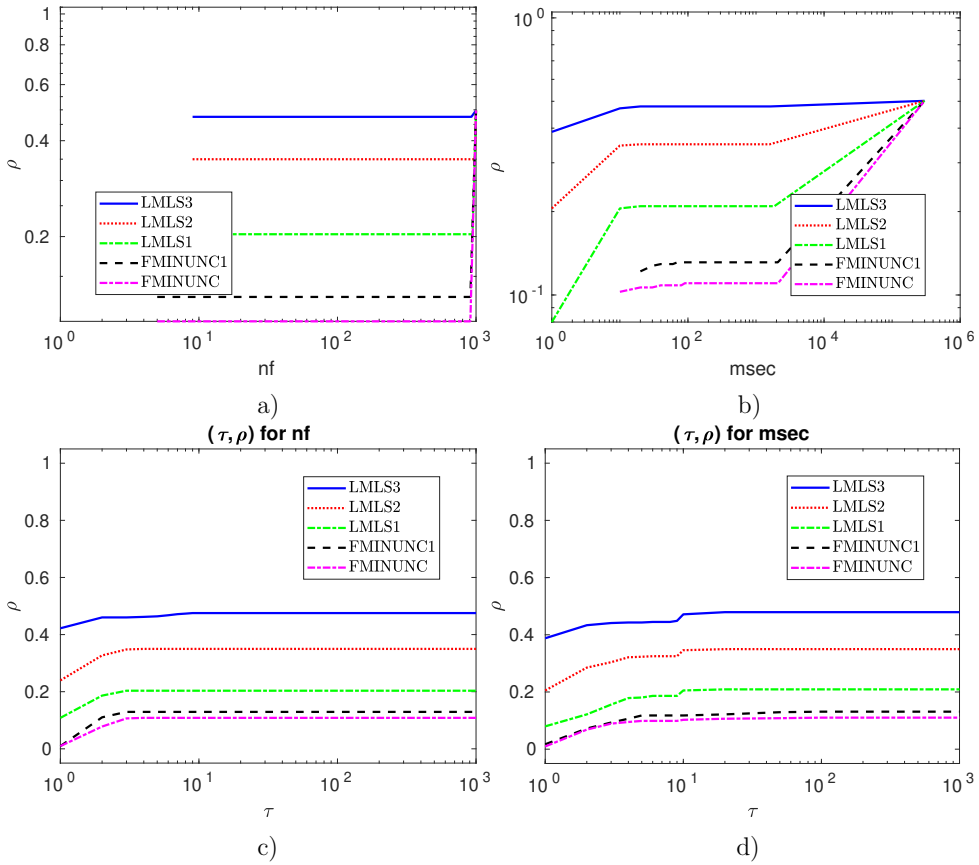


Figure 4: (a) and (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 4: Results for small scale and medium budget

stopping test: $q_f \leq$ 1e-08, sec $\leq$ 300, nf $\leq$ 50*n										
474 of 526 problems without bounds solved										mean efficiency in %
dim∈[1,100]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
DOGLEG1	**dogleg1**	419	80	17	148	107	0	0	55	30
LMLS4	**lmtr4**	416	228	17	108	110	0	0	66	59
CoDoSol1	**codo1**	398	226	52	90	107	0	21	65	54
NATRLS1	**natrs1**	364	79	21	81	162	0	0	51	39
LMLS3	**lmtr3**	359	177	8	124	167	0	0	55	46
NMPGTR2	**nmpg2**	343	191	22	79	183	0	0	53	40
LMLS2	**lmtr2**	282	111	16	197	244	0	0	40	29
NATRN1	**natrn1**	222	63	17	117	304	0	0	31	22
NLEQ1	**nleq1**	222	42	30	77	122	0	182	35	18
FMINUNC	**func**	219	4	3	167	292	0	15	13	11
LMLS1	**lmtr1**	206	49	21	253	320	0	0	24	15
FMINUNC1	**func1**	179	8	7	177	345	0	2	13	10
MINFLBFGSDL1	**minlbfgs1**	168	0	0	286	358	0	0	5	5
MINFLBFGS1	**lbfgs1**	133	0	0	200	339	0	54	5	4
LSQNONLIN1	**lsqn1**	131	1	1	314	395	0	0	5	4
STRSCNE1	**strs1**	72	69	0	28	1	0	453	13	8
MINFNCG1	**MINFNCG1**	47	0	0	698	458	0	21	1	1

Table 5: Results for small scale and large budget

stopping test: $q_f \leq$ 1e-08, sec $\leq$ 300, nf $\leq$ 100*n										
493 of 526 problems without bounds solved										mean efficiency in %
dim∈[1,100]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
DOGLEG1	**dogleg1**	434	82	19	167	92	0	0	56	31
LMLS4	**lmtr4**	424	229	17	113	102	0	0	67	59
CoDoSol1	**codo1**	408	227	53	103	88	0	30	66	54
NATRLS1	**natrs1**	375	87	27	104	151	0	0	53	39
LMLS3	**lmtr3**	374	177	9	150	152	0	0	56	48
NMPGTR2	**nmpg2**	365	196	25	103	160	0	1	54	40
FMINUNC	**func**	313	6	4	351	170	0	43	16	14
LMLS2	**lmtr2**	306	106	12	257	220	0	0	41	31
MINFLBFGSDL1	**minlbfgs1**	272	2	2	453	244	0	10	9	8
LMLS1	**lmtr1**	231	53	24	437	295	0	0	25	17
MINFLBFGS1	**lbfgs1**	230	0	0	349	232	0	64	8	6
NATRN1	**natrn1**	225	65	19	121	301	0	0	32	22
NLEQ1	**nleq1**	223	43	30	79	121	0	182	35	17
FMINUNC1	**func1**	209	8	6	327	297	0	20	14	12
LSQNONLIN1	**lsqn1**	155	1	1	374	371	0	0	6	4
MINFNCG1	**MINFNCG1**	136	0	0	524	365	0	25	2	2
STRSCNE1	**strs1**	72	69	0	28	1	0	453	13	7

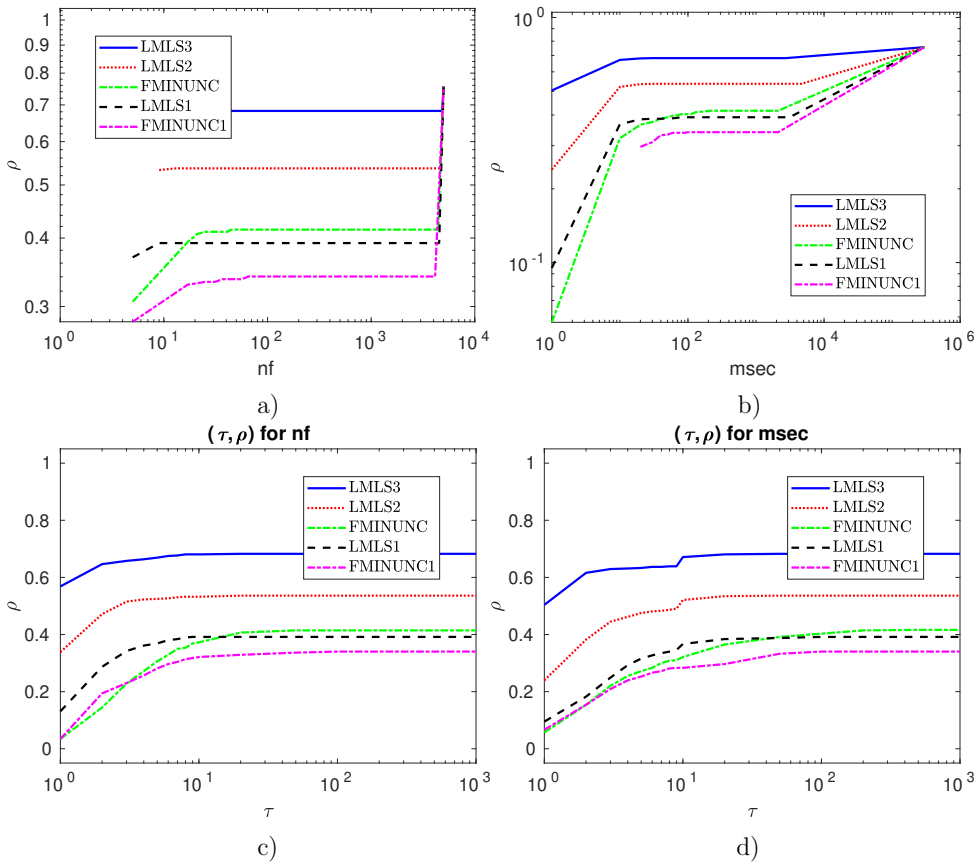


Figure 5: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

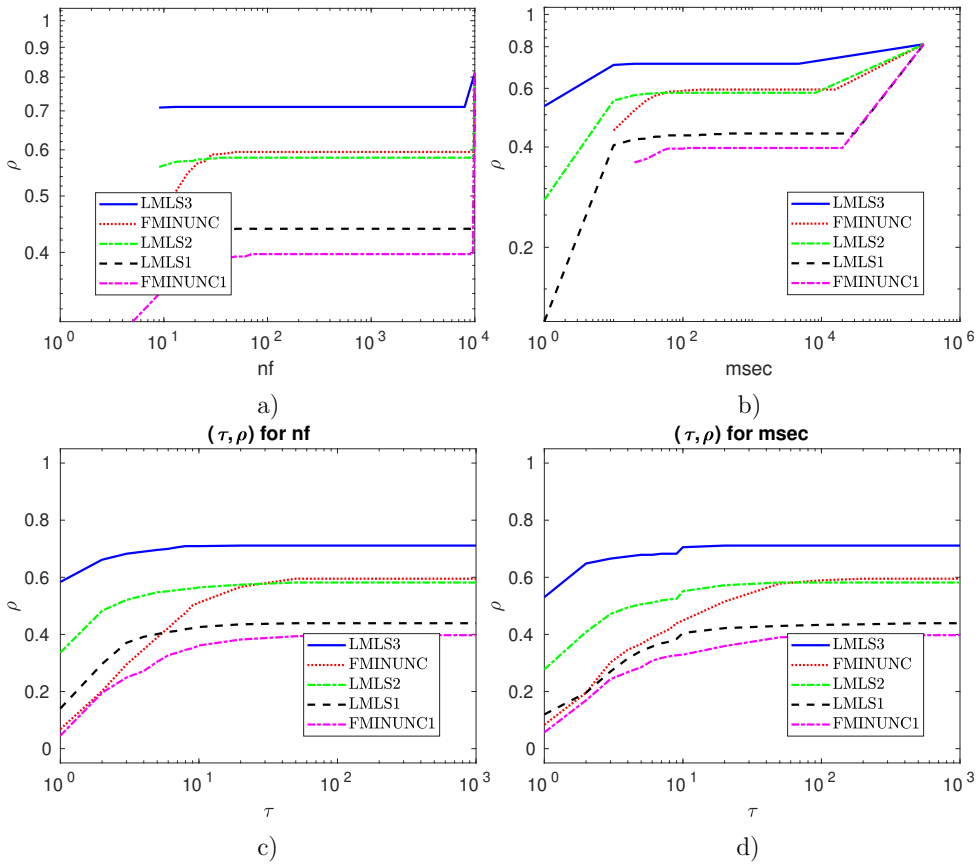


Figure 6: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

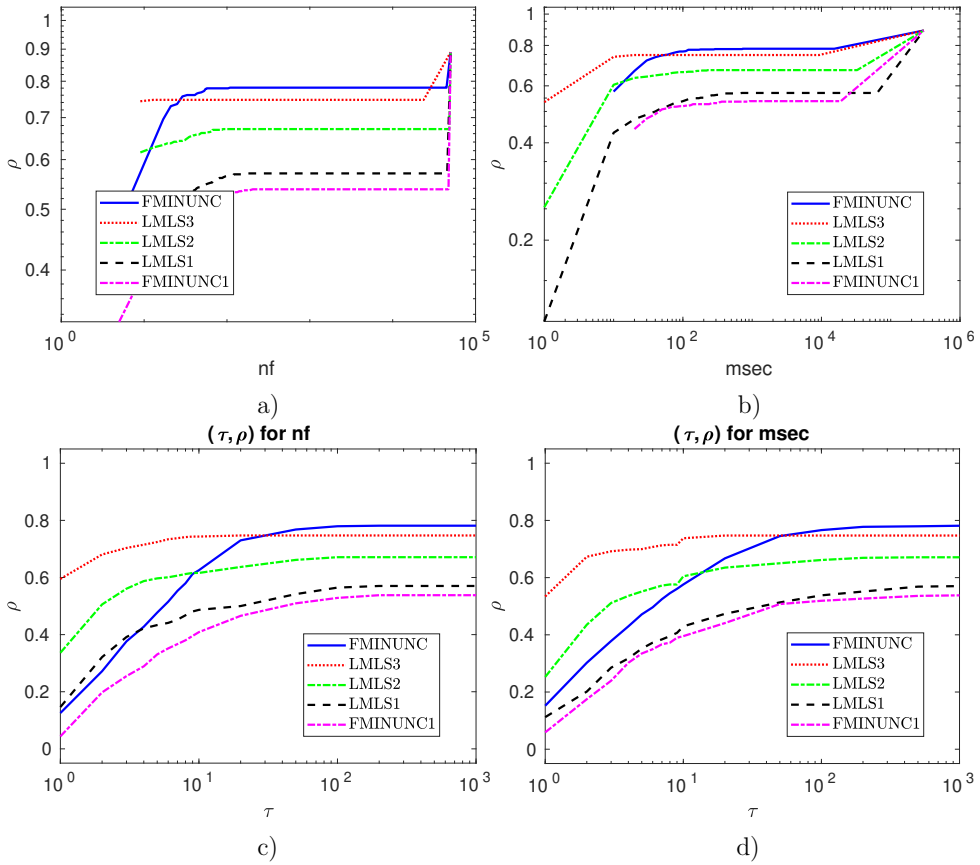


Figure 7: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 6: Results for small scale and very large budget

stopping test:		$q_f \leq$ 1e-08,		sec $\leq$ 300,		nf $\leq$ 500*n				
509 of 526 problems without bounds solved										mean efficiency in %
dim∈[1,100]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
DOGLEG1	**dogleg1**	444	79	18	220	82	0	0	57	31
LMLS4	**lmtr4**	437	229	18	159	89	0	0	68	58
CoDoSol1	**codo1**	422	226	54	172	61	0	43	67	55
FMINUNC	**func**	411	8	7	761	20	0	95	19	17
NMPGTR2	**nmpg2**	407	194	25	378	110	1	8	56	42
LMLS3	**lmtr3**	393	178	9	343	133	0	0	56	47
MINFLBFGSDL1	**minlbfgs1**	385	6	6	723	69	0	72	11	12
NATRLS1	**natrs1**	384	89	31	143	142	0	0	54	41
LMLS2	**lmtr2**	353	107	14	998	173	0	0	42	31
MINFLBFGS1	**lbfgs1**	342	4	4	973	56	0	128	10	9
LMLS1	**lmtr1**	300	53	25	1302	226	0	0	27	17
MINFNCG1	**MINFNCG1**	283	0	0	990	194	0	49	3	4
FMINUNC1	**func1**	283	8	7	534	205	0	38	15	13
NATRN1	**natrn1**	230	63	17	178	296	0	0	32	22
NLEQ1	**nleq1**	224	42	30	82	120	0	182	35	18
LSQNONLIN1	**lsqn1**	209	1	1	690	316	1	0	6	5
STRSCNE1	**strs1**	72	69	0	29	1	0	453	13	6

A.2 Medium scale: $n \in [101, 1000]$

Table 7: Results for medium scale and small budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 10*n				
154 of 249 problems without bounds solved										mean efficiency in %
dim∈[101,1000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmls4**	119	49	48	13738	129	1	0	37	37
FMINUNC1	**func1**	114	40	14	11271	133	2	0	29	35
LMLS3	**lmtr3**	110	30	28	15401	138	1	0	34	28
LMLS2	**lmtr2**	105	22	21	15006	143	1	0	31	20
LMLS1	**lmtr1**	97	15	15	18207	151	1	0	27	21
FMINUNC	**func**	93	26	0	13702	154	2	0	22	24

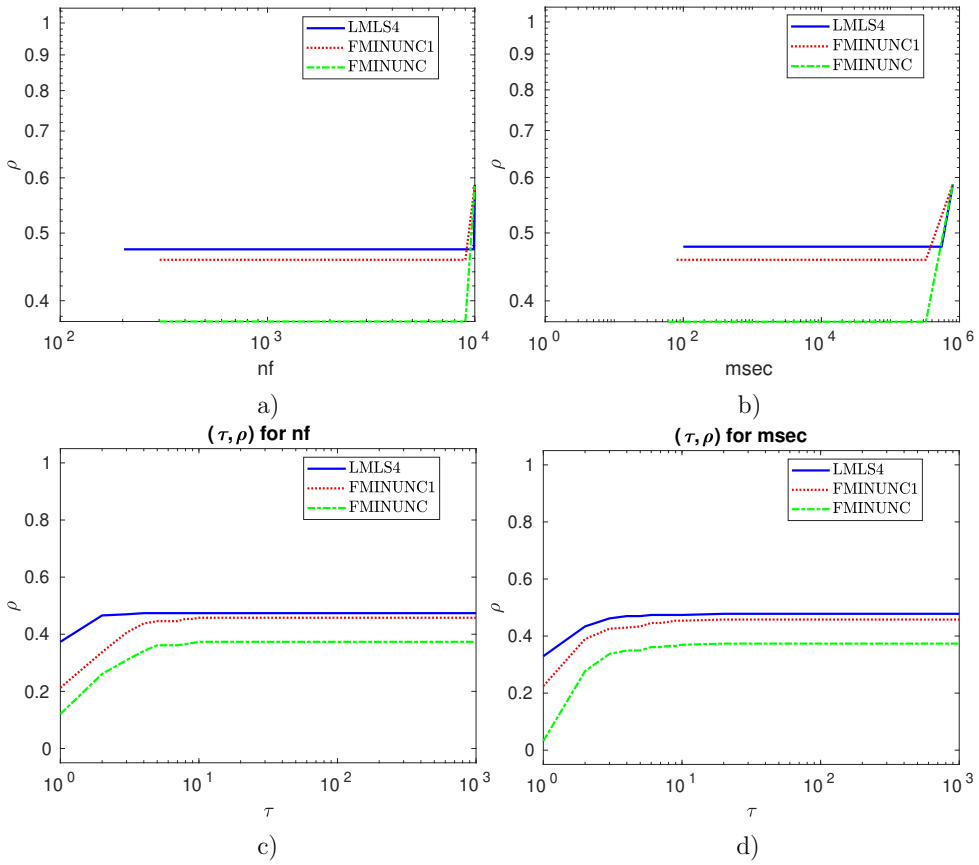


Figure 8: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 8: Results for medium scale and budget

stopping test:		$q_f \leq$ `0.001`,		`sec` $\leq$ `800`,		`nf` $\leq$ `50*n`				
188 of 249 problems without bounds solved										mean efficiency in %
dim∈[101,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmtr4**	169	58	56	11582	78	2	0	50	46
LMLS3	**lmtr3**	168	48	47	12980	79	2	0	47	37
FMINUNC	**func**	168	32	5	11064	78	3	0	36	42
FMINUNC1	**func1**	165	43	16	10503	81	3	0	39	50
LMLS2	**lmtr2**	164	16	15	15614	84	1	0	43	25
LMLS1	**lmtr1**	159	20	20	15399	88	2	0	38	28

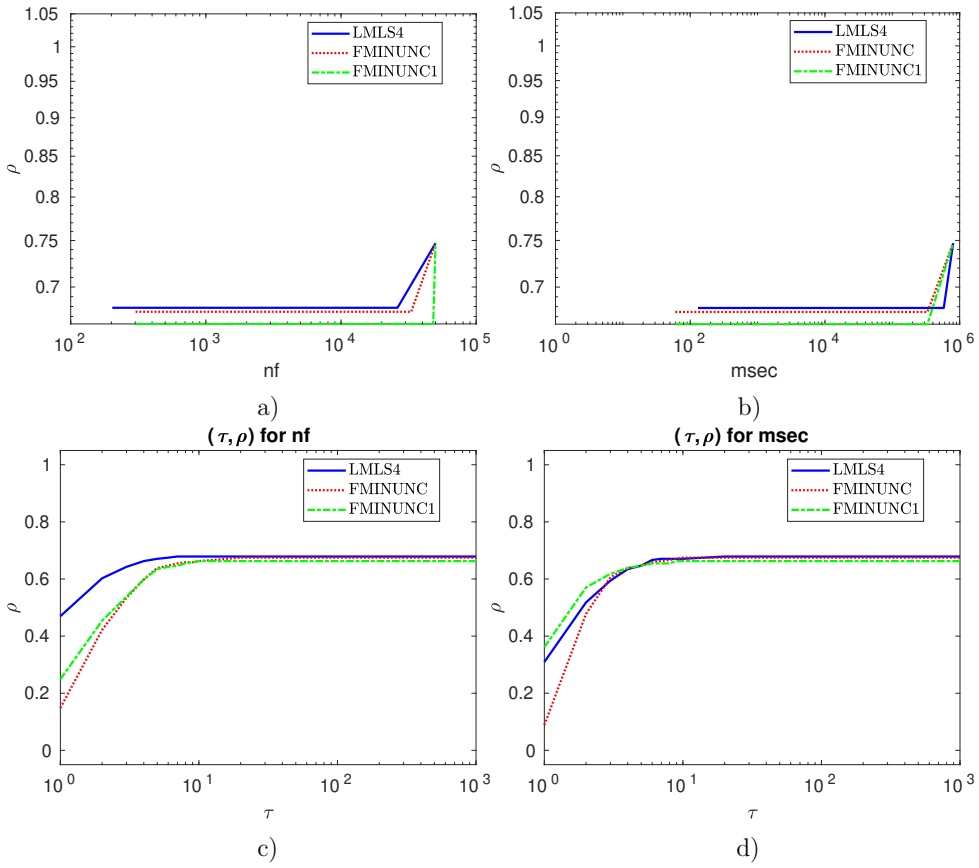


Figure 9: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 9: Results for medium scale and large budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 100*n				
198 of 249 problems without bounds solved										mean efficiency in %
dim∈[101,1000]						# of anomalies				for cost measure
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
FMINUNC	**func**	189	44	18	14603	55	3	2	41	48
FMINUNC1	**func1**	178	46	20	12133	66	3	2	41	52
LMLS3	**lmtr3**	176	39	37	16832	71	2	0	48	36
LMLS4	**lmtr4**	174	53	51	18938	73	2	0	50	47
LMLS2	**lmtr2**	167	24	24	18182	81	1	0	42	24
LMLS1	**lmtr1**	165	20	20	28711	84	0	0	38	26

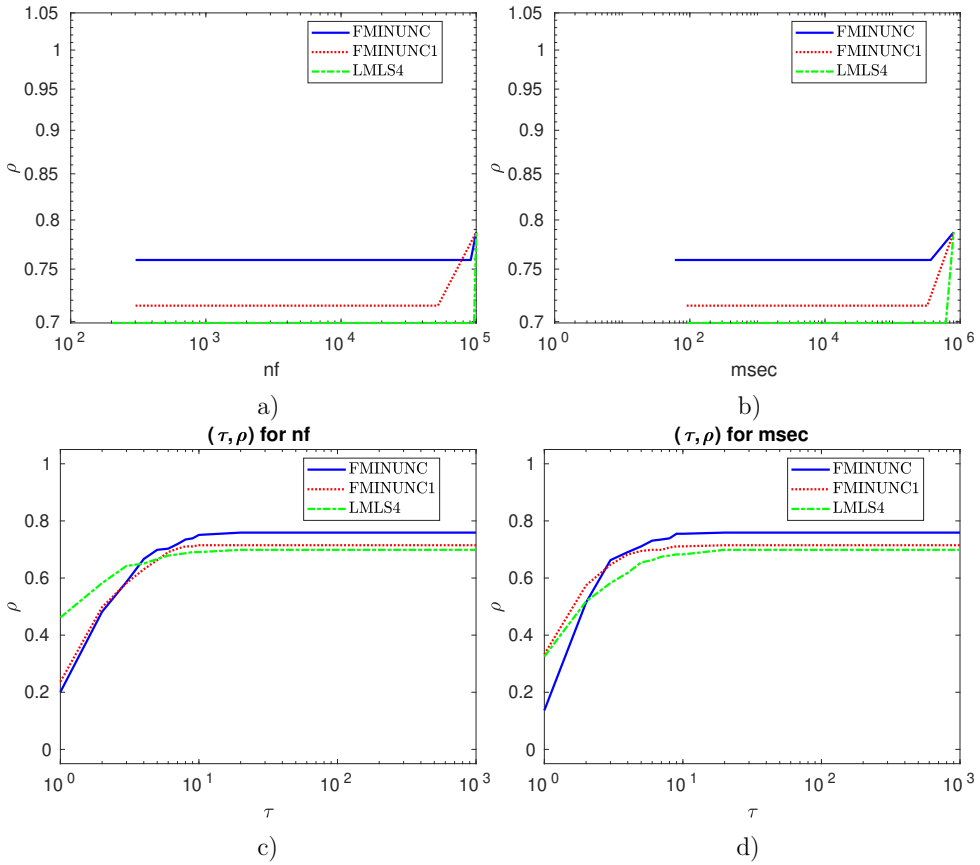


Figure 10: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

A.3 Large scale: $n \in [1001, 10000]$

Table 10: Results for large scale and small budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 10*n				
132 of 166 problems without bounds solved									mean efficiency in %	
dim∈[1001,10000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS4	**lmtr4**	101	47	44	265873	47	17	1	48	45
LMLS3	**lmtr3**	97	22	19	242590	57	10	2	43	42
LMLS2	**lmtr2**	94	11	10	262404	60	8	4	43	41
LMLS1	**lmtr1**	86	22	21	302069	60	14	6	37	32
FMINUNC1	**func1**	81	36	35	197750	60	24	1	34	37

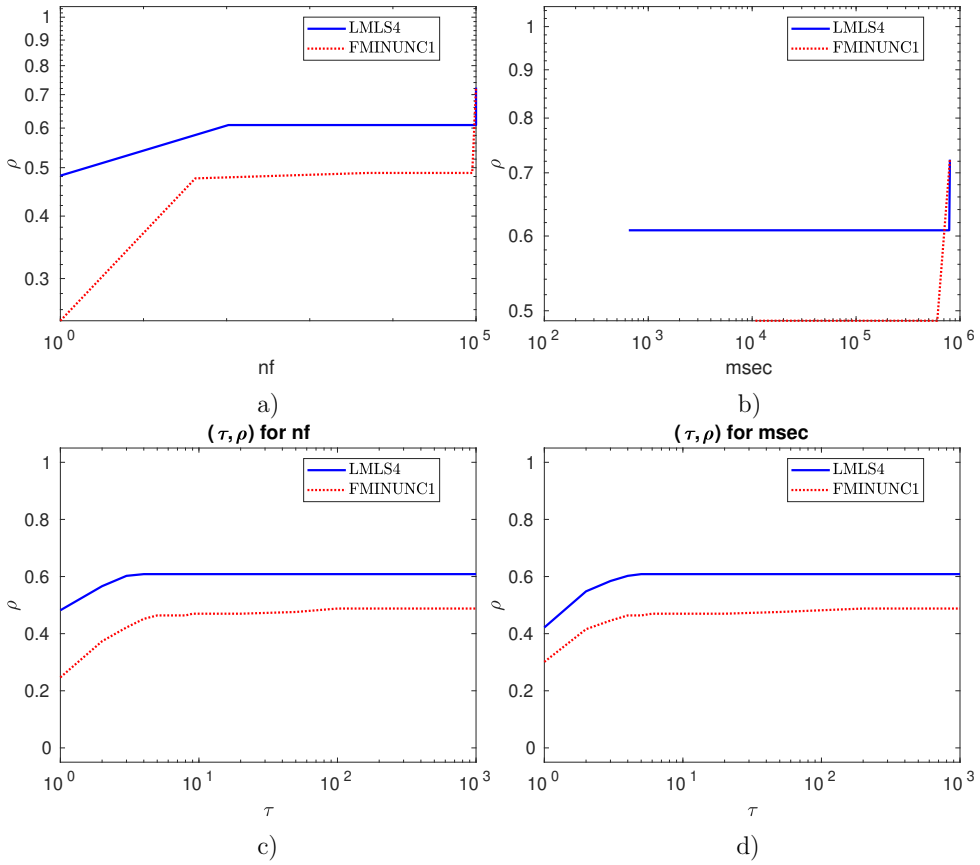


Figure 11: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 11: Results for large scale and budget

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 800,		nf $\leq$ 100*n				
147 of 166 problems without bounds solved									mean efficiency in %	
dim∈[1001,10000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
LMLS3	**lmtr3**	128	32	29	313348	0	38	0	57	55
LMLS4	**lmtr4**	128	49	47	325880	0	38	0	60	55
LMLS2	**lmtr2**	120	17	15	306190	0	46	0	53	49
FMINUNC1	**func1**	104	37	36	236336	0	61	1	42	47
LMLS1	**lmtr1**	95	18	17	332592	0	71	0	39	33

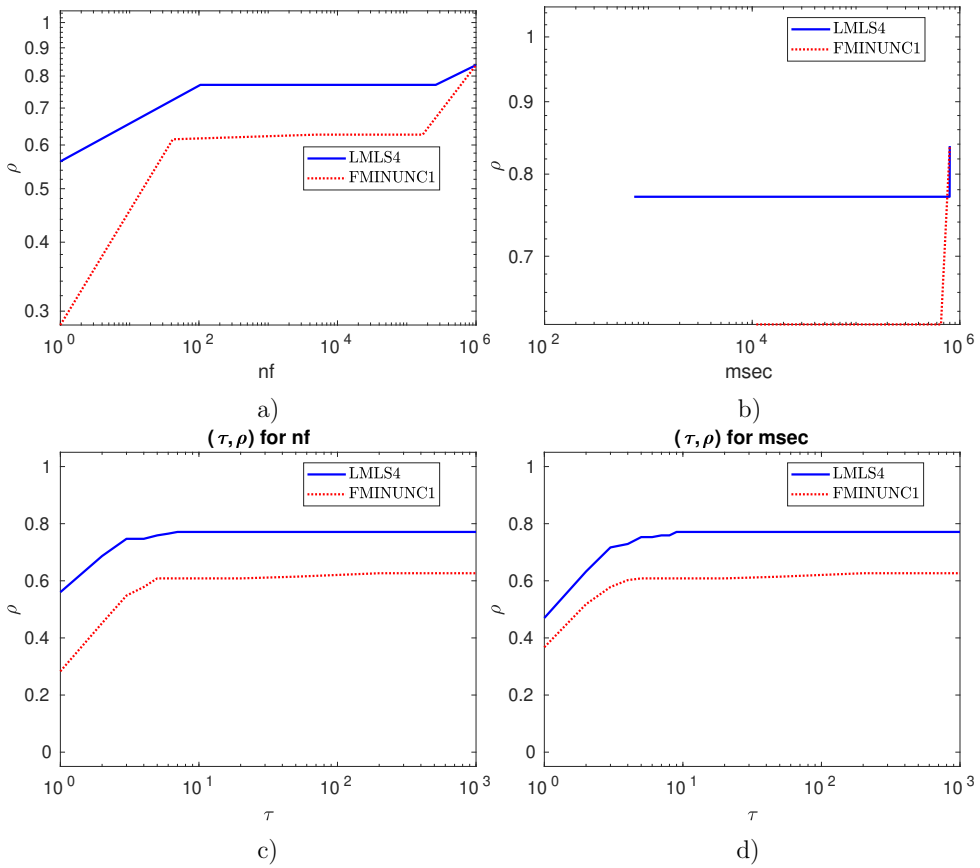


Figure 12: (a) – (b): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored. (c) – (d): Performance plots for `nf`/(`best nf`) and `msec`/(`best msec`), respectively. ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

References

[1] M. Ahookhosh and K. Amini. An efficient nonmonotone trust-region method for unconstrained optimization. *Numer. Algorithms* **59** (September 2011), 523–540.

[2] M. Ahookhosh, K. Amini, and M. Kimiaei. A globally convergent trust-region method for large-scale symmetric nonlinear systems. *Number. Func. Anal. Opt.* **36** (May 2015), 830–855.

[3] M. Ahookhosh, H. Esmaeili, and M. Kimiaei. An effective trust-region-based approach for symmetric nonlinear systems. *Int. J. Comput. Math.* **90** (March 2013), 671–690.

[4] K. Amini, H. Esmaeili, and M. Kimiaei. A nonmonotone trust-region-approach with nonmonotone adaptive radius for solving nonlinear systems. *IJNAO* **6** (February 2016).

[5] K. Amini, M. A. K. Shiker, and M. Kimiaei. A line search trust-region algorithm with nonmonotone adaptive radius for a system of nonlinear equations. *4OR-Q J. Oper. Res.* **14** (January 2016), 133–152.

[6] S. Bellavia, M. Macconi, and B. Morini. STRSCNE: A scaled trust-region solver for constrained nonlinear equations. *Comput. Optim. Appl.* **28** (April 2004), 31–50.

[7] S. Bellavia, M. Macconi, and S. Pieraccini. Constrained dogleg methods for nonlinear systems with simple bounds. *Comput. Optim. Appl.* **53** (March 2012), 771–794.

[8] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics (January 2000).

[9] N. Y. Deng, Y. Xiao, and F. J. Zhou. Nonmonotonic trust region algorithm. *J. Optim. Theory Appl.* **76** (February 1993), 259–285.

[10] P. Deuflhard. *Newton Methods for Nonlinear Problems*. Springer Berlin Heidelberg (2011).

[11] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.* **91** (January 2002), 201–213.

[12] H. Esmaeili and M. Kimiaei. An efficient adaptive trust-region method for systems of nonlinear equations. *Int. J. Comput. Math.* **92** (April 2014), 151–166.

[13] H. Esmaeili and M. Kimiaei. A new adaptive trust-region method for system of nonlinear equations. *Appl. Math. Model.* **38** (June 2014), 3003–3015.

[14] H. Esmaeili and M. Kimiaei. A trust-region method with improved adaptive radius for systems of nonlinear equations. *Math. Meth. Oper. Res.* **83** (November 2015), 109–125.

[15] J. Fan. Convergence rate of the trust region method for nonlinear equations under local error bound condition. *Comput. Optim. Appl.* **34** (March 2006), 215–227.

[16] J. Fan and J. Pan. An improved trust region algorithm for nonlinear equations. *Comput. Optim. Appl.* **48** (February 2009), 59–70.

[17] J. Fan and J. Pan. A modified trust region algorithm for nonlinear equations with new updating rule of trust region radius. *Int. J. Comput. Math.* **87** (October 2010), 3186–3195.

[18] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for newton's method. *SIAM J. Numer. Anal.* **23** (August 1986), 707–716.

[19] L. Grippo and M. Sciandrone. Nonmonotone derivative-free methods for nonlinear equations. *Comput. Optim. Appl.* **37** (March 2007), 297–328.

[20] Jr. J. E. Dennis and J. J. Moré. Quasi-newton methods, motivation and theory. *SIAM Rev.* **19** (January 1977), 46–89.

[21] Jr. J. E. Dennis and H. F. Walker. Convergence theorems for least-change secant update methods. *SIAM J. Numer. Anal.* **18** (December 1981), 949–987.

[22] M. Kimiaei. A new class of nonmonotone adaptive trust-region methods for nonlinear equations with box constraints. *Calcolo* **54** (October 2016), 769–812.

[23] M. Kimiaei. Nonmonotone self-adaptive levenberg–marquardt approach for solving systems of nonlinear equations. *Number. Func. Anal. Opt.* **39** (July 2017), 47–66.

[24] M. Kimiaei. Line search in noisy unconstrained black box optimization. Technical report, University of Vienna (2020).

[25] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. Technical report, University of Vienna (2020).

[26] M. Kimiaei and A. Neumaier. Testing and tuning optimization algorithm. Technical report, University of Vienna (2020).

[27] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.* **45** (August 1989), 503–528.

[28] L. Lukšan, C. Matonoha, and J. Vlček. Problems for nonlinear least squares and nonlinear equations. Technical Report V-1259, ICS CAS (2018).

[29] Jorge J. Moré and Stefan M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.* **20** (January 2009), 172–191.

[30] L. Nazareth. A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms. *SIAM J. Numer. Anal.* **16** (October 1979), 794–800.

[31] H.B. Nielsen. immoptibox – a matlab toolbox for optimization and data fitting (2012). version 2.2.

[32] J. Nocedal. Theory of algorithms for unconstrained optimization. *Acta Numer.* **1** (January 1992), 199–242.

[33] J. Nocedal and S. J. Wright, eds. *Numerical Optimization*. Springer-Verlag (1999).

[34] U. Nowak and L. Weimann. A family of newton codes for systems of highly nonlinear equations. Technical Report TR 91–10, Zuse Institute Berlin (ZIB) (1990).

[35] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics (January 2000).

[36] R.B. Schnabel. Sequential and parallel methods for unconstrained optimization. In *In Mathematical Programming, Recent Developments and Applications* (M. Iri and eds. K. Tanabe, eds.), pp. 227–261. Kluwer Academic Publishers (1989).

[37] L. Sorber, M. V. Barel, and L. D. Lathauwer. Unconstrained optimization of real functions in complex variables. *SIAM J. Optim.* **22** (January 2012), 879–898.

[38] Z. Yu and D. Pu. A new nonmonotone line search technique for unconstrained optimization. *J. Comput. Appl. Math.* **219** (September 2008), 134–144.

[39] Y. Yuan. Recent advances in numerical methods for nonlinear equations and nonlinear least squares. *Numer. Algebra, Control. Optim.* **1** (2011), 15–34.