TRESNEI, A MATLAB TRUST-REGION SOLVER FOR SYSTEMS OF NONLINEAR EQUALITIES AND INEQUALITIES *

BENEDETTA MORINI † AND MARGHERITA PORCELLI ‡

Abstract. The `Matlab` implementation of a trust-region Gauss-Newton method for bound-constrained nonlinear least-squares problems is presented. The solver, called `TRESNEI`, is adequate for zero and small-residual problems and handles the solution of nonlinear systems of equalities and inequalities. The structure and the usage of the solver are described and an extensive numerical comparison with functions from the `Matlab` Optimization Toolbox is carried out.

Key words. Bound-constrained nonlinear least-squares; nonlinear systems; nonlinear systems of inequalities; simple bounds; trust-region methods; algorithm design.

April 6, 2009

1. Introduction. Bound-constrained nonlinear least-squares problems model a large number of practical applications. Let consider the problem given by

$$\min_{l \le x \le u} f(x) = \frac{1}{2}\|F(x)\|_2^2, \tag{1.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differentiable function and l, $u \in \mathbb{R}^n$ are lower and upper bounds such that $l < u$. It includes as a special case solving a bound-constrained system of nonlinear equations. If a solution to the nonlinear system cannot be found, a local minimizer for the violations of the equations is computed. Also, problem (1.1) is also a suitable reformulation of systems of nonlinear equalities and inequalities ([15]) which arise in model formulation design, parameter identification problems, detection of feasible points in nonlinear programming and take the form

$$\begin{aligned} &C_E(x) = 0, \\ &C_I(x) \le 0, \\ &L \le x \le U. \end{aligned} \tag{1.2}$$

Here $C_E : \mathbb{R}^n \to \mathbb{R}^{m_E}$, $C_I : \mathbb{R}^n \to \mathbb{R}^{m_I}$, and the components of the bound-constraints L, $U \in \mathbb{R}^n$ satisfy $-\infty \le L_i \le U_i \le \infty$, $i = 1, \cdots, n$, see e.g. [8, 12, 17]. The functions C_E and C_I are supposed to be continuously differentiable. To obtain problem (1.1), the general inequalities $C_I(x) \le 0$ are transformed into equalities while the simple bounds are left unchanged. Thus any zero-residual solution of the least-squares problem solves (1.2).

In this paper we present a new `Matlab` solver for problem (1.1) which handles systems of nonlinear equalities and inequalities (1.2). Our proposal is based on the results presented by the authors in the recent papers [15, 16]. In particular, in [15] a trust-region Gauss-Newton method and a trust-region Levenberg-Marquardt method for solving (1.1) are presented. These methods handle the least-squares problem irrespective of its dimensions m and n, generate feasible iterates and rely on matrix

*Work supported by MIUR, Roma, Italia, through PRIN 2007, "Sviluppo ed analisi di modelli matematici e di metodi numerici per equazioni alle derivate parziali per le applicazioni a problemi ambientali ed industriali" and INDAM-GNCS.

†Dipartimento di Energetica "S. Stecco", Università di Firenze, via C. Lombroso 6/17, 50134 Firenze, Italia, benedetta.morini@unifi.it

‡Dipartimento di Matematica "U. Dini", Università di Firenze, viale Morgagni 67a, 50134 Firenze, Italia, margherita.porcelli@math.unifi.it

factorizations. The theoretical analysis conducted in [15, 16] showed that such methods are globally and fast locally convergent under standard assumptions. A wide experimental study showed that they are reliable in solving zero and small-residual problems and that the trust-region Gauss-Newton algorithm is more efficient and robust then the trust-region Levenberg-Marquardt procedure.

Our solver implements the trust-region Gauss-Newton algorithm proposed in [15] and offers an internal reformulation of (1.2) as problem (1.1) which preserves the smothness required by the trust-region algorithm. Hence, it is called `TRESNEI`, Trust-REgion Solver for Nonlinear Equalities and Inequalities.

If there are no general inequalities, i.e. $m_I = 0$, then `TRESNEI` solves nonsquare bound-constrained nonlinear systems and it turns out to be a nontrivial extension of the solver `STRSCNE` for square bound-constrained systems, [1]. Moreover, it overcomes a limitation in the functions for bound-constrained nonlinear least-squares problems provided by the `Matlab` Optimization Toolbox [18]; in fact, these functions cannot solve underdetermined problems, i.e. problems where the dimensions of F are such that $m < n$.

It is important to note that we may attempt to formulate (1.2) as an unconstrained nonlinear least-squares problem where the objective function includes all the inequalities and apply numerical methods for unconstrained optimization, see [4, 11]. Our motivations for not converting also the simple bounds into equalities comes from the observation that restricting the expected size of each variable serves both as check on the problem formulation and as a specification of the domain of the maps C_E and C_I; further, it can prevent function evaluations at unreasonable points during the iterations. Thus, it is advisable to adopt the formulation (1.1) and apply a procedure that enforce feasibility of the iterates with respect to these constraints.

`TRESNEI` has been intensively tested and the goals of our experiments were twofold. First, we were interested in assessing if the formulation (1.1) may offer an advantage as compared with an unconstrained least-squares formulation. Second, we were interested in comparing the computational cost and robustness of our algorithm with competing solvers. The function `lsqnonlin` from the `Matlab` Optimization Toolbox served our purpouses as will be shown in §4 and §6. The overall performance of `TRESNEI` shows that it is cost effective and robust and suggests us some lines for future research.

Finally, we note that `TRESNEI` does not require any special toolbox, so it can easily serve as a template for translations in another language. The solver is freely accessible through the web site: `http://TRESNEI.de.unifi.it`.

Throughout the paper, the subscript k will denote an iteration counter and for any function h, h_k will be the shorthand for $h(x_k)$. The ith component of a vector x will be denoted by either x_i or $(x)_i$ and for any set of indices I, $[x]_I$ will denote the subvector of x with components x_i, $i \in I$. Finally, $[l, u]$ will denote the feasible set $\{x \in \mathbb{R}^n \mid l \leq x \leq u\}$.

2. Proposed approach. The procedure implemented in `TRESNEI` consists of two phases. In the first phase, the constrained problem (1.1) is formed; then in the second phase such problem is solved. In this section we outline the proposed approach. The implementation details will be fully described in §3.

The first step of `TRESNEI` is to express the problem (1.2) as a bound-constrained least-squares problem where F is continuously differentiable and $l < u$. This is done considering a general and widespread modelling of the systems of equalities and inequalities which is also adopted in the CUTEr collection of test problems [9].

In (1.2), the components x_i of x can be either free or bounded on one side or bounded from above and from below. Moreover, it is standard to provide the problem parameters as "fixed" variables, i.e. variables with equal upper and lower bounds. In what follows, we let I_{fx}, I_{lb} and I_{ub} be the sets containing the indeces of fixed, lower and upper bounded variables respectively:

$$I_{fx} = \{i \in \{1, \ldots, n\} : L_i = U_i\}, \tag{2.1}$$

$$I_{lb} = \{i \in \{1, \ldots, n\} : i \notin I_{fix} \text{ and } L_i \neq -\infty\}, \tag{2.2}$$

$$I_{ub} = \{i \in \{1, \ldots, n\} : i \notin I_{fix} \text{ and } U_i \neq +\infty\}. \tag{2.3}$$

Obviously I_{lb} and I_{ub} may not be disjoint.

Suppose that the problem (1.2) contains no fixed variables neither nonlinear inequalities. Then, `TRESNEI` attempts to solve (1.1) where

$$\begin{aligned} &F(x) = C_E(x), \\ &m = m_E, \quad l = L, \quad u = U. \end{aligned}$$

Otherwise, the problem (1.2) is posed as a bound-constrained nonlinear least-squares including fixed variables and the general inequalities $C_I(x) \leq 0$ into the function F. In particular, the general inequalities are converted into equalities using the continuously differentiable function $[t]_+ = \max\{t, 0\}^2/2$ and the bounds on the fixed variables are dropped introducing equalities of the form

$$[x]_{I_{fx}} - [U]_{I_{fx}} = 0. \tag{2.4}$$

Thus, the function F in (1.1) takes the form

$$F(x) = \begin{pmatrix} C_E(x) \\ [x]_{I_{fx}} - [U]_{I_{fx}} \\ [C_I(x)]_+ \end{pmatrix}, \tag{2.5}$$

and the number of its components is

$$m = m_E + m_I + n_{fx},$$

where n_{fx} is the cardinality of the set I_{fx}. The remaining simple bounds are kept separate from the objective function so that the bounds l and u in (1.1) are given by

$$l_i = \begin{cases} -\infty & \text{if } i \in I_{fx} \\ L_i & \text{otherwise} \end{cases}, \qquad u_i = \begin{cases} +\infty & \text{if } i \in I_{fx} \\ U_i & \text{otherwise} \end{cases}, \tag{2.6}$$

$i = 1, \ldots, n$.

Clearly, zero-residual solutions of the above constrained least-squares problem solve (1.2). The first-order optimality conditions for (1.1) can be stated as ([3])

$$D(x)\nabla f(x) = 0, \tag{2.7}$$

where $\nabla f(x) = F'(x)^T F(x)$, F' is the Jacobian matrix of F, and D is the scaling matrix

$$D(x) = diag(|d_1(x)|, \ldots, |d_n(x)|),$$

$$d_i(x) = \begin{cases} x_i - u_i & \text{if } (\nabla f(x))_i < 0,\ u_i < \infty, \\ x_i - l_i & \text{if } (\nabla f(x))_i \geq 0,\ l_i > -\infty, \\ 1 & \text{if } (\nabla f(x))_i \geq 0,\ l_i = -\infty, \text{ or} \\ & \quad (\nabla f(x))_i < 0,\ u_i = \infty. \end{cases}$$

The method implemented in `TRESNEI` for solving (1.1) uses a Gauss-Newton model for f together with a trust-region strategy and generates a sequence $\{x_k\}$ of feasible iterates. At kth iteration, the trust-region problem is given by

$$\min_p\{m_k(p): \quad \|p\|_2 \leq \Delta_k\}, \tag{2.8}$$

where

$$m_k(p) = \frac{1}{2}\|F_k' p + F_k\|_2^2, \tag{2.9}$$

and $\Delta_k > 0$ is the trust-region radius.

Let the sequence $\{x_k\}$ be defined by $x_{k+1} = x_k + p_k$, $k \geq 0$. The first-order optimality conditions are satisfied at every limit point of $\{x_k\}$ if the step p_k satisfies the fraction of Cauchy decrease condition

$$\rho_c(p_k) = \frac{m_k(0) - m_k(p_k)}{m_k(0) - m_k(p_k^C)} \geq \beta_1, \qquad \beta_1 \in (0,1), \tag{2.10}$$

where p_k^C is the scaled Cauchy step defined as

$$p_k^C = \underset{p \in span\{-D_k \nabla f_k\}}{\operatorname{argmin}} \; m_k(p) \quad \text{subject to} \quad \|p\|_2 \leq \Delta_k, \;\; x_k + p \in [l,u], \tag{2.11}$$

see [15]. To find such a step p_k, first we compute a solution p_{tr} to the trust-region problem. In particular, since in general the global minimizer to m_k is non-unique, we evaluate the minimum norm solution p_k^N to the problem $\min_p m_k(p)$ which is given by

$$p_k^N = -{F_k'}^+ F_k, \tag{2.12}$$

where ${F_k'}^+$ denotes the Moore-Penrose pseudoinverse of matrix F_k'. The vector p_k^N solves the trust-region problem (2.8) if it is inside the trust-region, otherwise the dogleg step between p_k^N and the Cauchy direction for (2.8) is computed. Second, we form the projected step $\bar{p}_{tr}$

$$\bar{p}_{tr} = P(x_k + p_{tr}) - x_k,$$

where P is the projection map, $P(x) = \max\{l, \min\{x, u\}\}$, and compute a step p_k of the form

$$p_k = t\, p_k^C + (1-t)\bar{p}_{tr}, \qquad t \in [0,1], \tag{2.13}$$

satisfying the condition (2.10). Since the steps p_k^C and $\bar{p}_{tr}$ give rise to feasible points, the point $x_k + p_k$ is feasible too.

Finally, the predicted reduction of the quadratic model m_k and the actual reduction of the objective function f at the trial point $x_k + p_k$ are compared using the standard rule

$$\rho_f(p_k) = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} \geq \beta_2, \qquad \beta_2 \in (0,1). \tag{2.14}$$

If the trial point is accepted, then $x_{k+1} = x_k + p_k$ and a new iteration begins. Otherwise the trust-region radius is reduced.

The results of the convergence analysis for the Gauss-Newton trust-region method carried out in [15, 16] are summarized below. The global convergence properties are guaranteed by (2.10); the local fast convergence rate properties follow from using the step p_k^N.

THEOREM 2.1. *Let $\{x_k\}$ be the sequence generated by the Gauss-Newton trust-region method and L be an open, bounded and convex set containing the whole sequence $\{x_k\}$. Suppose that the Jacobian matrix F' is Lipschitz continuous in L and $\|F'\|$ is bounded above on L. Then the following statements hold:*

i) Every limit point of the sequence $\{x_k\}$ is a first-order stationary point for the problem (1.1).

ii) If x^* is a limit point of $\{x_k\}$ and $\|F(x^*)\|_2 = 0$, then all the limit points of $\{x_k\}$ solve the problem (1.2).

iii) If the sequence $\{x_k\}$ has a limit point x^* such that $\|F(x^*)\|_2 = 0$ and $F'(x^*)$ is full rank, then $\{x_k\}$ converges to x^* quadratically.

Proof. See [15, Theorems 4.1, 4.2] and [16, Theorem 3.1].

We remark that the above assumptions on function F are satisfied if C_I, C'_E, C'_I are bounded in norm in the set L and C'_E and C'_I are Lipschitz continuous in L.

3. Implementation details. The procedure implemented in `TRESNEI` is described in Algorithm 3.1. In this section we give details on its steps and discuss the user's options for the problem description and the choice of the parameters.

3.1. Problem description. `TRESNEI` offers its user the facility of requiring a minimal description of the problem (1.2) and building an internal reformulation of the problem. In particular, the functions C_E, C_I and the bounds L and U are expected. Then, the problem (1.1) is internally formed as described in §2, see Step 1.

Following the notation in (2.5), it easy to see that the solution of a bound-constrained least-squares problem is attempted providing its residual function C_E.

If the Jacobian matrices C'_E, C'_I have been provided along with C_E and C_I, the Jacobian matrix F' is formed. Otherwise, a finite difference approximation of matrix F' is evaluated.

It is important to note that the user may prefer an alternative transformation of the nonlinear inequalities to the one employed in (2.5). For example, the use of slack variables casting nonlinear inequalities into nonlinear equalities, can be accomplished providing the resulting sistem of nonlinear equations to `TRESNEI`.

3.2. Solution of the trust-region problem. The solution of the trust-region problem (2.8) is addressed in Step 3 of the algorithm. If the Jacobian matrix F' is square and nonsingular, then the step p_k^N given in (2.12) is computed by the `Matlab` backslash operator. Otherwise, the complete orthogonal decomposition of F' is applied using procedures which are sligth modifications of those given in [13]. Clearly, the use of matrix factorization sets limits on the size of problems that can be solved efficiently by `TRESNEI`.

If p_k^N does not solve the trust-region problem, then we use the classical dogleg path to approximate the trust-region solution, [17]. The Cauchy step c_k for the problem (2.8) has the form

$$c_k = -\min\left\{\frac{\|\nabla f_k\|_2^2}{\|F'_k\nabla f_k\|_2^2},\ \frac{\Delta_k}{\|\nabla f_k\|_2}\right\}\nabla f_k. \tag{3.1}$$

The procedure described above was implemented achieving economies in the calculations. Since a zero components of $[C_I(x_k)]_+$ gives rise to a zero component in F

and to a null row in F', instead of (2.9) we use the reduced model

$$\hat{m}_k(p) = \frac{1}{2}\|\hat{F}'_k\, p + \hat{F}_k\|_2^2, \tag{3.2}$$

where $\hat{F}_k$ is the vector formed by $C_E(x_k)$ and the nonzero components of $[C_I(x_k)]_+$ and $\hat{F}'_k$ is the Jacobian of $\hat{F}$ at x_k, see [11].

3.3. Computation of the trial step. In Step 4 of the algorithm, the trial step p_k is formed testing the condition (2.10); the scalar β_1 used in (2.10) is an internal parameter fixed in Step 2. To evaluate p_k^C defined in (2.11), let

$$q_k = \underset{p \in span\{-D_k \nabla f_k\}}{\operatorname{argmin}} \; m_k(p) \quad \text{subject to} \quad \|p\|_2 \leq \Delta_k$$

i.e.

$$q_k = -\omega D_k \nabla f_k, \qquad \omega = \min\left\{ \frac{\|D_k^{\frac{1}{2}} \nabla f_k\|_2^2}{\|F'_k D_k \nabla f_k\|_2^2},\ \frac{\Delta_k}{\|D_k \nabla f_k\|_2} \right\},$$

and

$$\xi = \min_{1 \leq i \leq n} \lambda_i, \quad \lambda_i = \begin{cases} \max\{\frac{l_i - (x_k)_i}{(q_k)_i}, \frac{u_i - (x_k)_i}{(q_k)_i}\} & \text{if } (q_k)_i \neq 0, \\ \infty & \text{if } (q_k)_i = 0. \end{cases}$$

Then p_k^C takes the form

$$p_k^C = \begin{cases} q_k & \text{if } \ x_k + q_k \in [l, u], \\ \xi q_k, & \text{otherwise.} \end{cases} \tag{3.3}$$

The step $\bar{p}_{tr}$ is accepted as the trial step p_k if it satisfies (2.10). Alternatively, we find the step p_k of the form (2.13) such that $\rho_c(p_k) = \rho_c(t\, p_k^C + (1-t)\bar{p}_{tr}) = \beta_1$. It is easy to verify that this is a quadratic scalar equation admitting a unique positive root t^*.

3.4. Test on the trial step and trust-region radius update. The trial step p_k is accepted in Step 5 of the algorithm if condition (2.10) is satisfied. In this case the trust-region radius is updated following a standard strategy and imposing that the initial trust-region radius Δ_{k+1} is greater than or equal to a prescribed value Δ_{m1}. Clearly, on termination of each iteration, the trust-region radius may be smaller that Δ_{m1}. On the other hand, if p_k fails to satisfy (2.14), then it is rejected and the trust-region radius Δ_k is reduced. Note that if Δ_k becomes smaller than the fixed parameter Δ_{m2}, we terminate the procedure and declare a failure.

The parameters β_2, Δ_{m1}, Δ_{m2} are set internally in Step 2.

3.5. Termination criteria and accuracy. Successful termination of `TRESNEI` means that one of the following conditions is met

$$\|F_k\|_\infty \leq \epsilon_1 \sqrt{n}, \tag{3.4}$$

$$\min\{\|D_k \nabla f_k\|_2, \|P(x_k - \nabla f_k) - x_k\|_2\} \leq \epsilon_2, \tag{3.5}$$

where ϵ_1 and ϵ_2 are prescribed tolerances. The condition (3.5) involves two optimality measures: the scaled gradient $D\nabla f$ which is a key ingredient of our method, and the projected gradient of function f. The use of both measures is due to the fact that the value $\|D\nabla f\|$ may oscillate and exhibit a large growth at some iterations. Thus, the use of the norm of the projected gradient provides a more reliable stopping condition.

Algorithm 3.1: THE TRUST-REGION GAUSS-NEWTON ALGORITHM

Input: C_E, C_I, L, U, m_E, m_I, n,
$x_0 \in [l, u]$, Δ_0, k_M, ϵ_1, ϵ_2.

1. *Problem description*
Let l, u as in (2.6) and m be the length of F in (2.5).
Compute $F(x_0)$ by (2.5) and $F'(x_0)$. Set $k = 1$.
2. *Internal parameters*
Set $\beta_1 = 1/10$, $\beta_2 = 1/4$, $\beta_3 = 3/4$.
Let ϵ_m be the machine precision, set $\Delta_{m1} = \sqrt{\epsilon_m}$, $\Delta_{m2} = \epsilon_m$.

While $k \leq k_M$ do

3. *Solve the trust-region problem*
3.1 If $m \neq n$ compute p_k^N given in (2.12) by the complete orthogonal decomposition.
If $m = n$ solve the linear system $F_k' \, p_k^N = -F_k$;
If F_k' is singular, compute p_k^N by the complete orthogonal factorization.
3.2 If $\|p_k^N\|_2 \leq \Delta_k$ set $p_{tr} = p_k^N$;
Else form c_k given in (3.1).
compute the dogleg step p_{tr} between p_k^N and c_k.
4. *Compute the trial step p_k.*
4.1 Let $\bar{p}_{tr} = P(x_k + p_{tr}) - x_k$.
4.2 Compute p_k^C in (3.3).
4.3 If $\bar{p}_{tr}$ satisfies (2.10), set $p_k = \bar{p}_{tr}$;
Else compute the positive root t^* of $\rho_c(t \, p_k^C + (1-t)\bar{p}_{tr}) - \beta_1 = 0$
set $p_k = t^* \, p_k^C + (1 - t^*)\bar{p}_{tr}$.
5. *Test on p_k and trust-region radius update*
5.1 Compute $F(x_k + p_k)$ by (2.5).
If p_k satisfies (2.14) set $x_{k+1} = x_k + p_k$.
Else set $\Delta_k = \min\{\Delta_k/4, \, \|p_k\|_2/2\}$;
If $\Delta_k > \Delta_{m2}$, go to Step 3.2;
Else exit.
5.2 If $\rho_f(p_k) \geq \beta_3$, set $\Delta_{k+1} = \max\{\Delta_k, \, \Delta_{m1}, \, 2\|p_k\|_2\}$;
Else set $\Delta_{k+1} = \max\{\Delta_k, \, \Delta_{m1}\}$;
6. *Termination test*
6.1 If (3.4) or (3.5) is satisfied, exit.
Else compute $F'(x_{k+1})$ and increment k.

4. An alternative approach. The use of the commercial `Matlab` Optimization Toolbox software for solving least-squares problems, gives rise to an approach alternative to the one used in `TRESNEI`. In particular, it yields to testing an unconstrained least-squares formulation of (1.2).

The `MATLAB` Optimization Toolbox includes the function `lsqnonlin` which consists of two implementations: the large-scale algorithm and the medium-scale algorithm. The large-scale algorithm is a subspace trust-region method while the medium-scale algorithm uses either the Levenberg-Marquardt method or the Gauss-Newton method globalized by a line-search strategy.

The applicability of `lsqnonlin` has some limitations. Bounds on the variables can be handled only by the large-scale algorithm. On the other hand, such algorithm cannot solve underdetermined problems i.e. problems where the number of elements of F is lower than the number of variables. Therefore, bound-constrained underdetermined least-squares problems cannot be solved by `lsqnonlin`.

Because of the above limitations, the only way to solve a variety of systems of equalities and inequalities without restrictions on their dimensions consists in expressing the problems as unconstrained least-squares problems. In fact, given (1.2) we apply the medium-scale algorithm to the problem

$$\min_x g(x) = \|G(x)\|_2^2, \tag{4.1}$$

where

$$G(x) = \begin{pmatrix} C_E(x) \\ [x]_{I_{fx}} - [U]_{I_{fx}} \\ [C_I(x)]_+ \\ \max\{\big[\,[L-x]_{I_{lb}}\,\big]_+, \big[\,[x-U]_{I_{ub}}\,\big]_+\} \end{pmatrix}. \tag{4.2}$$

Note that the function G differs from (2.5) as it incorporates the simple bounds in the sets I_{lb}, I_{ub}. It easy to verify that G is continuously differentiable. Similar reformulations can be found in [4, 11].

We will cosider the solution of (4.1) by the medium-scale algorithm of `lsqnonlin`. It terminates successfully either if the directional derivative along the search direction s_k and the infinity-norm of the gradient of g_k are less than prescribed tolerances, i.e.

$$\nabla g_k^T s_k \le \zeta_1 \quad \text{and} \quad \|\nabla g_k\|_\infty \le 10(\zeta_1 + \zeta_2), \tag{4.3}$$

or if the magnitude of search direction is sufficiently small, i.e.

$$\|s_k\|_\infty \le \zeta_2. \tag{4.4}$$

On the other side, a failure is declared if the line-search strategy can not sufficiently decrease the residual along the current search direction.

5. Benchmarking. The solvers `TRESNEI` and `lsqnonlin` do not test a uniform stopping criterium. Hence, it is essential benchmarking the two solvers in order to guarantee that the returned approximate solutions satisfy the same accuracy requirement.

We adopt the benchmarking process proposed in [6] for general constrained optimization problems and we fit it to the problem (1.1). It consists in computing and checking a specific test for the solvers a posteriori. Specifically, each solver is run using the default tolerances. If the approximate solution returned by the solver does not satisfy the a posteriori convergence test, then the native solver tolerances are reduced and the problem is solved again. Further tolerance reductions are made until the a posteriori convergence test is satisfied or a failure is declared.

The definition of the a posteriori convergence test is given in terms of measures for feasibility and stationarity. Such measures are defined using an error measure function $\delta[\cdot,\cdot]$ which involves a mixture of the absolute and relative error. In particular, given real numbers ξ_1 and ξ_2, $\delta[\xi_1,\xi_2]$ is defined as

$$\delta[\xi_1,\xi_2] = \min\left\{|\xi_1-\xi_2|, \frac{|\xi_1-\xi_2|}{|\xi_1|+|\xi_2|}\right\},$$

with $\delta[0,0] = 0$ and $\delta[\xi_1, \xi_2] = 1$ if either ξ_1 or ξ_2 is infinite. The function $\delta[\cdot,\cdot]$ is continuous.

The feasibility measure is given by

$$\nu_f(x) = \|v(x)\|_\infty, \tag{5.1}$$

where $v(x) \in \mathbb{R}^n$ and

$$(v(x))_i = \begin{cases} 0 & \text{if } l_i \le x_i \le u_i, \\ \min\{\delta[x_i, l_i], \delta[x_i, u_i]\} & \text{otherwise.} \end{cases}$$

Clearly, ν_f is null at feasible points while it measures the constraints violations at infeasible points. Given a small positive scalar τ, a vector x is defined to be τ-feasible if $\nu_f(x) \le \tau$, [6].

The stationarity measure ν_s can be defined as

$$\nu_s(x, \tau) = \|r(x, \tau)\|_\infty, \tag{5.2}$$

where τ is a small positive scalar, $r(x, \tau) \in \mathbb{R}^n$ and

$$(r(x,\tau))_i = \begin{cases} \min\{0, (\nabla f(x))_i\} & \text{if } \delta[x_i, l_i] \le \tau, \delta[x_i, u_i] > \tau, \\ \max\{0, (\nabla f(x))_i\} & \text{if } \delta[x_i, l_i] > \tau, \delta[x_i, u_i] \le \tau, \\ (\nabla f(x))_i & \text{if } \delta[x_i, l_i] > \tau, \delta[x_i, u_i] > \tau, \\ 0 & \text{if } \delta[x_i, l_i] \le \tau, \delta[x_i, u_i] \le \tau \text{ or} \\ & \text{if } \delta[x_i, L_i] \le \tau, i \in I_{fx}. \end{cases} \tag{5.3}$$

Note that the last assignement in (5.3) is related to the equation (2.4) and that L_i is the value assumed by the fixed variable $x_i, i \in I_{fx}$, of the problem (1.2).

The relationship between the optimality measures (5.1) and (5.2) and the first-order optimality conditions for the problem (1.1) is clarified by the following theorem.

THEOREM 5.1. *Let $\tau > 0$ be given and let x^* be a first-order stationary point of the problem (1.1). If $\{x_k\}$ is a sequence that converges to x^*, then $\{\nu_f(x_k)\}$ and $\{\nu_s(x_k, \tau)\}$ converge to zero.*

Proof. The sequence $\{\nu_f(x_k)\}$ trivially converges to zero since the sequence $\{x_k\}$ converges to a feasible point x^* of problem (1.1).

We prove that $\{\nu_s(x_k, \tau)\}$ converges to zero recalling that the first-order optimality conditions (2.7) are equivalent to the conditions

$$\begin{array}{lll} (\nabla f(x^*))_i = 0 & \text{if} & l_i < x_i^* < u_i, \\ (\nabla f(x^*))_i \le 0 & \text{if} & x_i^* = u_i, \\ (\nabla f(x^*))_i \ge 0 & \text{if} & x_i^* = l_i. \end{array} \tag{5.4}$$

Consider the ith component of x_k for k sufficiently large and without lack of generality suppose that if x_i^* is active then $x_i^* = l_i$. Since $\{x_k\}$ converges to x^*, if $x_i^* = l_i$ then $\delta[(x_k)_i, l_i] \le \tau$ for all k sufficiently large, otherwise either $\delta[(x_k)_i, l_i] \le \tau$ or $\delta[(x_k)_i, l_i] > \tau$ may hold.

If $x_i^* = l_i$ and $(\nabla f(x^*))_i > 0$, then by (5.3) and the continuity of the gradient, we get

$$(r(x_k, \tau))_i = \min\{0, (\nabla f(x_k))_i\} = 0,$$

for k sufficiently large.

If $x_i^* = l_i$ and $(\nabla f(x^*))_i = 0$ or $x_i^* > l_i$ and $\delta[(x_k)_i, l_i] \leq \tau$, then

$$(r(x_k, \tau))_i = \min\{0, (\nabla f(x_k))_i\} \leq |(\nabla f(x_k))_i|. \tag{5.5}$$

Since $\lim_{k\to\infty}(\nabla f(x_k))_i = 0$, then from (5.5) we obtain $\lim_{k\to\infty}(r(x_k, \tau))_i = 0$.

Finally if $x_i^* > l_i$ and $\delta[(x_k)_i, l_i] > \tau$, $\lim_{k\to\infty}(r(x_k, \tau))_i = 0$ easily follows from (5.3) and (5.4).

The case $x_i^* = u_i$ can be studied as above and therefore we can conclude that $\lim_{k\to\infty} \nu_s(x_k, \tau) = 0$. □

The first requirement on the solutions returned by the solvers `TRESNEI` and `lsqnonlin` is their τ-feasibility. Moreover, we assess the accuracy of the solutions by using the stationarity measure ν_s. In practice, benchmarking requires that the solutions delivered by `TRESNEI` and `lsqnonlin` satisfy

$$\nu_f(x) \leq \tau_f, \quad \nu_s(x, \tau_f) \leq \tau_s, \tag{5.6}$$

for specified tolerances τ_f and τ_s. The τ-feasibility feature is nontrivially fulfilled by `lsqnonlin` as it may return a nonzero-residual stationary point for (4.1). On the other hand, since `TRESNEI` generates a sequence of feasible iterations, enforcing (5.6) means controlling only the stationarity measure ν_s.

6. Numerical experience. In this section we discuss the numerical experiments with `TRESNEI` and `lsqnonlin`, with particular emphasis on the effects of the enforcement of the convergence test (5.6). All the tests were performed on an Intel Xeon (TM) 3.4 Ghz, 1GB RAM using `Matlab 7.6` and machine precision $\epsilon_m \approx 2 \times 10^{-16}$.

6.1. The problem set. The test examples are from the CUTEr test collection [10]. In view of the suitability of `TRESNEI` for medium-size problems, we selected 135 problems of the form (1.2) and we adjusted their dimensions to obtain variants where $n \leq 500$. Among the problems considered, there are 14 systems of nonlinear equations; the rest of the problems are constraint sets of problems from CUTEr.

In Tables (6.1)-(6.3) we report the names along with the main characteristics of the problems under consideration. In particular, n_{fr} and n_{fx} indicate the number of free and fixed variables respectively, n_b the number of variables that are bounded at least on one side and n_r the number of variables bounded on both sides ("range" variables). Moreover, the number m_E of equalities and the number m_I of general inequalities are reported.

The starting point x_0 and the analytical Jacobian matrices C'_E and C'_I are provided by CUTEr as part of each problem specification.

6.2. Algorithmic options. `TRESNEI` was run with the initial trust-region radius Δ_0 in Algorithm 3.1 set equal to 1. To obtain a feasible starting guess, the initial guess provided by CUTEr was projected onto the box $[l, u]$.

In `lsqnonlin`, for the line search algorithm we selected a safeguarded cubic polynomial method instead of the default strategy. This choice is recommended in [18] if gradients are supplied and can be calculated quite inexepensively. The initial point used is the one given by CUTEr.

For both solvers, all attempts to solve the test problems were limited to a maximum of 1000 iterations or 1000 function evaluations. The default tolerances ϵ_1, ϵ_2 in (3.4) and (3.5) and ζ_1, ζ_2 in (4.3) and (4.4) are

$$\epsilon_1 = \epsilon_2 = 10^{-6}, \qquad \zeta_1 = \zeta_2 = 10^{-6}.$$

Benchmarking of the solvers has been performed as follows. If one solver reports a failure with the default tolerances then the benchmarking process is not activated. Otherwise, the a posteriori test (5.6) is checked as suggested in [6], i.e. setting

$$\tau_f = \tau_s = 10^{-6}.$$

In case (5.6) is not satisfied, the tolerances provided to the solvers are reduced by a factor 10 and the problem is solved again. The progressive reduction of the tolerances is stopped, and a failure is declared, when they reach the value 10^{-16}. It is important to remark that if the solvers fails during the repeated runs but the test (5.6) is satisfied at the returned approximation, then we declare a successful run.

6.3. Results. Firstly, we tested `TRESNEI` and `lsqnonlin` on the problem set using the default tolerances. Secondly, we compared `TRESNEI` and `lsqnonlin` checking the a posteriori test (5.6).

Let consider the experiments conducted with the default tolerances. Both the Gauss-Newton method and the Levenberg-Marquardt method implemented in the medium-scale algorithm of `lsqnonlin` were run, see §4. `TRESNEI` solved 121 of the 135 problems, the Levenberg-Marquardt and the Gauss-Newton implementations of `lsqnonlin` solved 130 and 91 problems respectively. In fact, the Gauss-Newton implementation of `lsqnonlin` fails to handle overdetermined problems with rank-deficient Jacobian matrices. Due to this pitfall, we will refer to the Levenberg-Marquardt implementation of `lsqnonlin` in the remaining of the section.

For the successful runs, we analyzed the value of residual functions F in (2.5) and G in (4.2) returned by `TRESNEI` and `lsqnonlin` respectively; Figure 6.1 shows the values

$$\rho_T = -\log_{10} \|F(x)\|_2, \qquad \rho_l = -\log_{10} \|G(x)\|_2. \tag{6.1}$$

We note that the final residual is less than 10^{-6} in 70 problems for `TRESNEI` and in 26 problems for `lsqnonlin`. For problem `HS99EXP`, the residual $\|G\|_2$ returned is around 1.

Since the convergence tests of the solvers are not consistent, conclusions are difficult to drawn from the results obtained. It is quite evident that, `TRESNEI` returns small values of $\|F\|_2$; hence we can safely conclude that the solutions returned by the solver are accurate approximations to the solutions of problem (1.2). On the other hand, assessing the accuracy of the solutions delivered by `lsqnonlin` is more difficult. The residual function G in (4.1) includes the simple bounds and the values of $\|G\|_2$ shown in Figure 6.1 may indicate the computation of an infeasible solution of (1.2) with respect to the simple bounds.

Performing the benchmark, `TRESNEI` and `lsqnonlin` computed a solution satisfying (5.6) in 119 (88%) of the 135 problems and 117 (87%) of the tests, respectively. Figures 6.2 displays the function-evaluations count performance profile [5] for these runs. The plot shows that both solvers are very reliable and makes clear that `TRESNEI` is the most efficient for about 75% of the runs and `lsqnonlin` is within a factor 5 of `TRESNEI` for about 80% of the runs.

Some comments on the failures occurring in the benchmarking process are needed. `TRESNEI` fails to satisfy the stationarity requirement ν_s in (5.6) for 2 problems while `lsqnonlin` fails 13 times as the τ-feasibility required in (5.6) is not met. The reason why `lsqnonlin` fails to satisfy this requirement is that its iterates may not be feasible and tipically the problem (1.2) has nonisolated solutions. Thus, it may happen that

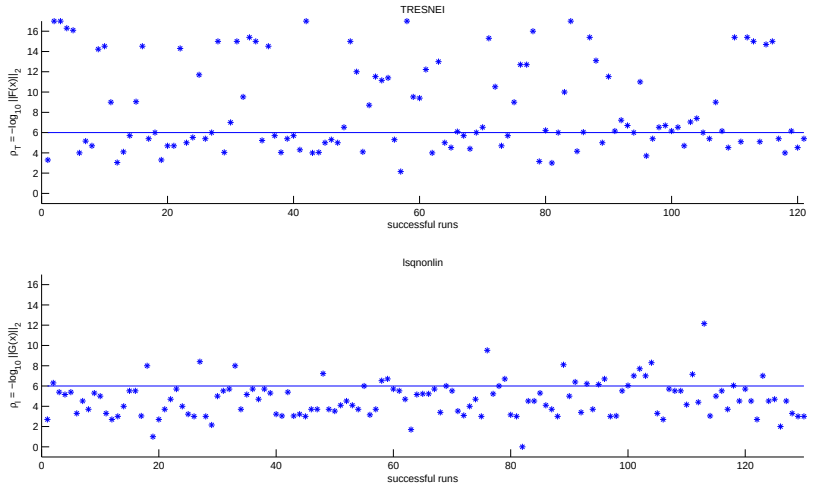

FIG. 6.1. *Plot of final residuals (6.1) for the successful runs.*

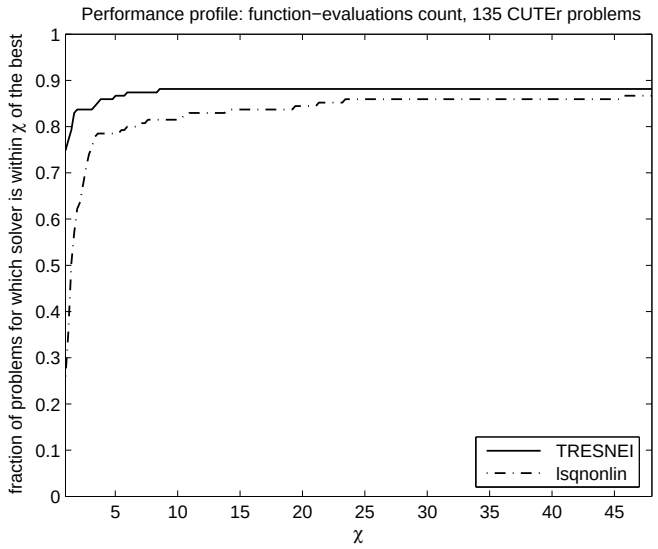

FIG. 6.2. *Performance profile, $\psi(\chi)$: Function-evaluation counts for the 135 problem under consideration.*

the sequence generated by `lsqnonlin` converges to a solution of problem (4.1) that is not feasible but close to the boundary of the box $[l, u]$. This situation can be verified numerically as the value of ν_f settles down for decreasing values of the tolerances ζ_1 and ζ_2 in (4.3) and (4.4).

The purpose of what follows is to investigate the effect of the convergence criteria (5.6) on solvers performance. In Figures 6.3-6.4, for each problem we report the bar

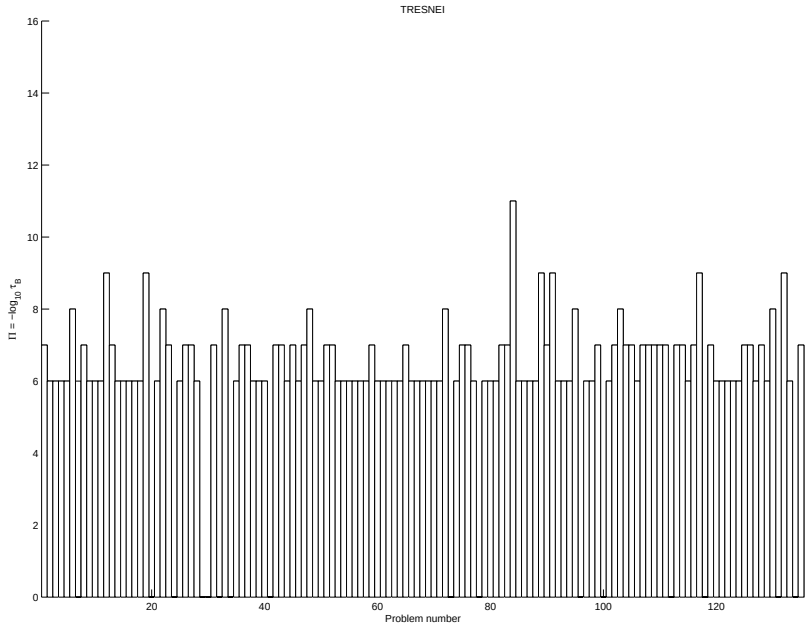

FIG. 6.3. *Graph of the performance measure (6.2) for* `TRESNEI`.

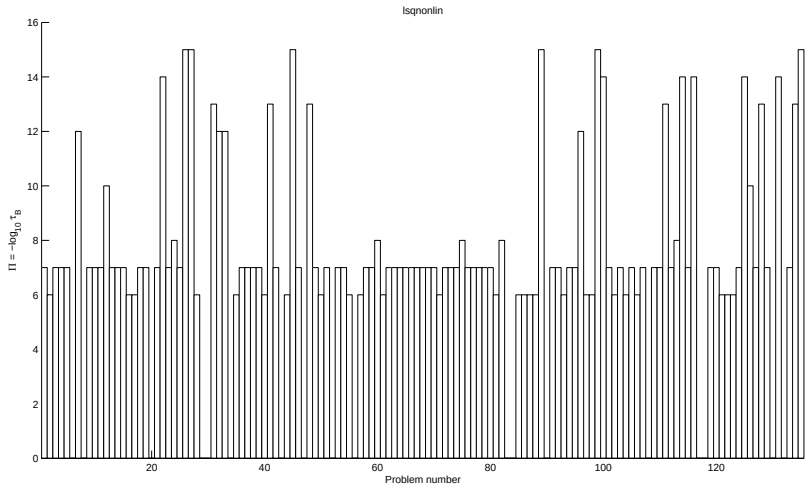

FIG. 6.4. *Graph of the performance measure (6.2) for* `lsqnonlin`.

showing the values

$$\Pi = -\log_{10}(\tau_B), \tag{6.2}$$

where τ_B is the tolerance needed by either `TRESNEI` or `lsqnonlin` to satisfy the condition (5.6). If one solver failed either with the default tolerances or in the bench-

marking, no bar is plotted. Concerning runs made with `TRESNEI`, Figure 6.3 shows that the heigth of 65 bars is equal to 6 and that for only 7 problems the bars reach values greater than or equal to 9. On the other hand, for `lsqnonlin` 26 bars have heigth equal to 6 and 24 bars are higher than 9; this indicates that lower tolerances than the default ones are often necessary to obtain accurate solutions in the sense of (5.6), see Figure 6.4.

These observations are confirmed in Figures 6.5-6.6 where for each problem and solver we plot the performance metric

$$p(x) = -\log_{10}(\max\{\nu_f(x), \nu_s(x, \tau_f)\}), \tag{6.3}$$

for the computed solution x. Clearly, the heights of the bars give the levels of accuracy reached, and returned solutions x such that $p(x) < 6$ do not satisfy the test (5.6). The white bars indicate the values $p(x)$ obtained using the default tolerances. The black bars indicate the values $p(x)$ resulting from the benchmarking process; if a black bar is not present, then (5.6) is satisfied using the default tolerances. If no bar is present, then the solver fails with the default tolerances.

Figure 6.5 shows that `TRESNEI` is able to compute highly accurate solutions and, in accordance to Figure 6.3, the convergence test (5.6) is satisfied with the default tolerances for most of the problems. Therefore, we can conclude that criteria (3.4)-(3.5) tend to agree with (5.6) in most cases. On the other hand, comparing Figure 6.5 and 6.6 it is evident that the level of accuracy of the solutions computed by `lsqnonlin` is remarkably lower than the level of accuracy reached using `TRESNEI`.

7. Conclusions. We have fully described the implementation of the solver `TRESNEI` and performed a numerical comparison to assess its reliability. Its overall performance against `lsqnonlin` encourages us to study possible improvements and extensions of the algorithm implemented. A chance of extending the applicability of `TRESNEI` comes from the use of iterative linear solvers in the trust-region solution. Further, the basic trust-region scheme can be enhanced by using a filter strategy.

REFERENCES

[1] S. Bellavia, M.Macconi, B. Morini, *STRSCNE: A scaled trust-region solver for constrained nonlinear equations*, Comput. Optim. Appl. 28, pp. 31-50, 2004.

[2] A.R. Conn, N.I.M. Gould, Ph.L. Toint, *Trust-region methods*, SMPS/SIAM Series on Optimization, 2000.

[3] T.F. Coleman, Y. Li, *An interior trust-region approach for nonlinear minimization subject to bounds*, SIAM J. Optim. 6, pp. 418-445, 1996.

[4] J.E. Dennis, M. El-Alem, K. Williamson, *A trust-region approach to nonlinear systems of equalities and inequalities*, SIAM J. Optim. 9, pp. 291-315, 1999.

[5] E.D. Dolan, J.J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming 91, 201-213, 2002.

[6] E.D. Dolan, J.J. Moré, T.S. Munson, *Optimality measures for performance profiles*, SIAM J. Optim. 16, pp. 891-909, 2006

[7] R. Fletcher, S. Leyffer, *Filter-type Algorithms for Solving Systems of Algebraic Equations and Inequalities*, High Performance Algorithms and Software for Nonlinear Optimization, G. Di Pillo and A. Murli, editors, Kluwer Academic Publishers, pp. 259-278,2003.

[8] C.A. Floudas et al., *Handbook of test problems in local and global optimization*, Kluwer Academic Publishers, Nonconvex Optimization and its Applications, 33, 1999.

[9] N. Gould, D. Orban and Ph. L. Toint, *CUTEr, a Contrained and Unconstrained Testing Environment, revisited*, ACM Trans. Math. Soft. 29(4), pp. 373-394, 2003.

[10] N.I.M. Gould, D.Orban, Ph.L. Toint, *CUTEr, a constrained and unconstrained testing environment, revisited*, ACM Transactions on Mathematical Software 29, pp. 373-394, 2003.

[11] N.I.M. Gould, Ph.L. Toint, *FILTRANE: a Fortran 95 filter-trust-region package for solving nonlinear least-squares and nonlinear feasibility problems*, ACM Trans. Math. Soft. 33, pp. 3-25, 2007.
[12] R. Fletcher, S. Leyffer *Nonlinear programming without a penalty function*, Mathematical Programming 91, pp. 239270, 2002.
[13] N.J. Higham, *The Matrix Computation Toolbox*, http://www.ma.man.ac.uk/∼higham/mctoolbox.
[14] C. Kanzow, N. Yamashita, M. Fukushima, *Levenberg-Marquardt methods for constrained nonlinear equations with strong local convergence properties*, J. Comput. Appl. Math. 172, pp. 375-397, 2004.
[15] M.Macconi, B.Morini, M.Porcelli, *Trust-region quadratic methods for nonlinear systems of mixed equalities and inequalitie*, Applied Numerical Mathematics, 59, pp. 859-876, 2009.
[16] M.Macconi, B. Morini, M. Porcelli, *A Gauss-Newton method for solving bound-constrained underdetermined nonlinear systems*, Optimization Methods and Software 24, pp. 219-235, 2009.
[17] J. Nocedal, S.J. Wright, *Numerical Optimization*, Springer Series in Operations Research, 1999.
[18] *Optimization Toolbox, Matlab 7*, The MathWorks, Natick, MA.

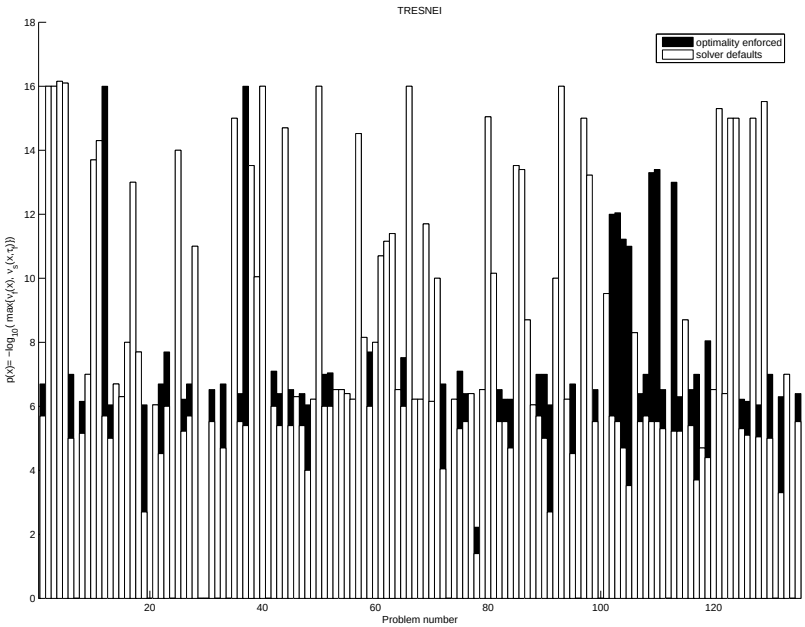

FIG. 6.5. *Graph of the performance metric (6.3) for* `TRESNEI`.

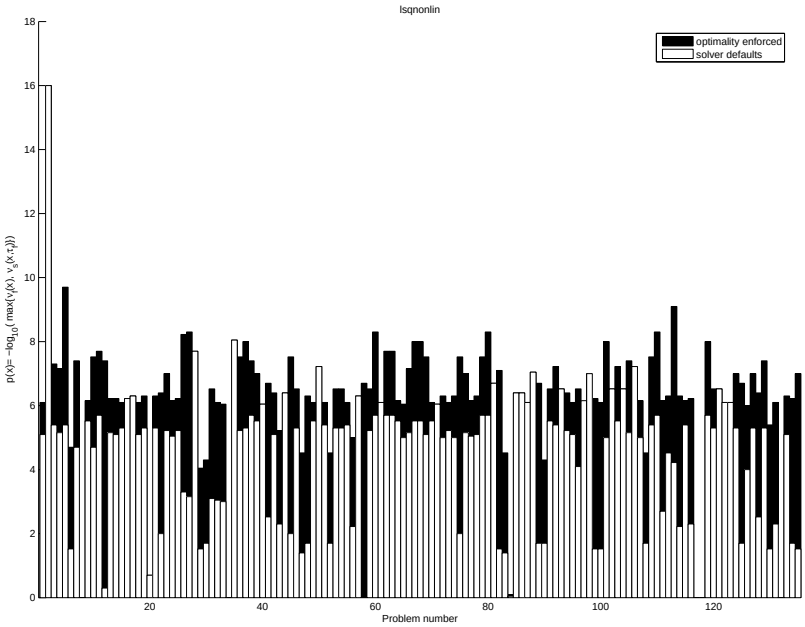

FIG. 6.6. *Graph of the performance metric (6.3) for* `lsqnonlin`.

Problem	n_{fr}	n_b	n_r	n_{fx}	m_E	m_I
AIRPORT	0	0	84	0	0	42
ALLINITC	1	1	1	1	1	0
ALJAZZAF	0	100	400	0	1	0
ALSOTAME	0	0	2	0	1	0
ANTWERP	0	3	24	0	8	2
AVGASA	0	0	8	0	0	10
AVION2	0	0	49	0	15	0
BATCH	0	2	46	0	12	61
BIGGSC4	0	0	4	0	0	13
BLOCKQP1	0	0	30	0	10	1
BLOWEYA	11	0	11	0	12	0
BT13	4	1	0	0	1	0
CAMSHAPE	0	0	100	0	0	304
CANTILVR	0	5	0	0	0	1
CHANDHEQ	0	10	0	0	10	0
CHEMRCTA	0	10	0	0	10	0
CHEMRCTB	0	10	0	0	10	0
CLNLBEAM	51	0	98	4	100	0
CONCON	10	5	0	0	11	0
CORE2	0	41	116	0	108	26
CORKSCRW	47	0	40	9	60	10
C-RELOAD	0	84	258	0	200	84
CSFI1	0	4	1	0	2	3
CSFI2	0	5	0	0	2	3
CVXQP1	0	0	100	0	50	0
DALLASM	0	0	196	0	151	0
DALLASS	0	0	46	0	31	0
DECONVC	0	51	10	0	1	0
DEGENLPA	0	0	20	0	15	0
DEMBO7	0	0	16	0	0	21
DISC2	22	0	7	0	17	6
DISCS	21	12	0	3	18	48
DNIEPER	1	0	56	4	24	0
DRUGDISE	10	30	19	4	50	0
DUAL1	0	0	85	0	1	0
EG3	1	0	100	0	1	299
EIGENA	0	0	110	0	110	0
EIGMAXA	0	0	101	0	101	0
EIGMAXB	0	0	101	0	101	0
FCCU	0	19	0	0	8	0
FEEDLOC	0	0	87	3	19	288
FLETCHER	3	1	0	0	1	3
GRIDNETA	26	16	4	14	36	0
GRIDNETC	40	20	0	0	36	0
HAGER4	10	10	0	1	10	0
HIMMELBI	0	100	0	0	0	12
HIMMELBJ	0	43	0	2	14	0
HIMMELBK	0	24	0	0	14	0
HIMMELP5	0	0	2	0	0	3
HONG	0	0	4	0	1	0
HS15	1	1	0	0	0	2
HS17	0	1	1	0	0	2
HS18	0	0	2	0	0	2
HS19	0	0	2	0	0	2
HS23	0	0	2	0	0	5
HS41	0	0	4	0	1	0

TABLE 6.1
Test problem characteristics

Problem	n_{fr}	n_b	n_r	n_{fx}	m_E	m_I
HS53	0	0	5	0	3	0
HS54	0	0	6	0	1	0
HS59	0	0	2	0	0	3
HS60	0	0	3	0	1	0
HS63	0	3	0	0	2	0
HS68	0	0	4	0	2	0
HS71	0	0	4	0	1	1
HS72	0	0	4	0	0	2
HS73	0	0	4	0	1	2
HS74	0	0	4	0	3	2
HS75	0	0	4	0	3	2
HS80	0	0	5	0	3	0
HS83	0	0	5	0	0	3
HS87	0	0	6	0	4	0
HS95	0	0	6	0	0	4
HS101	0	0	7	0	0	5
HS104	0	0	8	0	0	5
HS106	0	0	8	0	0	6
HS107	4	2	3	0	6	0
HS108	8	1	0	0	0	13
HS109	0	2	7	0	6	4
HS111	0	0	10	0	3	0
HS112	0	10	0	0	3	0
HS114	0	0	10	0	3	8
HS116	0	0	13	0	0	14
HS119	0	0	16	0	8	0
HS99EXP	21	0	7	3	21	0
HUES-MOD	0	10	0	0	2	0
HUESTIS	0	10	0	0	2	0
LEAKNET	80	70	6	0	153	0
LEWISPOL	0	0	6	0	9	0
LOTSCHD	0	12	0	0	7	0
MANNE	0	199	100	1	0	200
MCONCON	10	5	0	0	11	0
MINC44	0	11	12	4	18	0
MINPERM	0	1	4	0	5	0
MISTAKE	8	1	0	0	0	13
MRIBASIS	0	0	24	12	9	46
NET1	20	0	23	5	38	19
NCVXQP1	0	0	10	0	5	0
ODFITS	0	6	0	0	10	0
OPTCDEG3	40	39	40	3	80	0
OPTCNTRL	9	10	10	3	20	0
ORTHREGE	35	1	0	0	20	0
ORTHREGF	78	2	0	0	25	0
PFIT1	2	1	0	0	3	0
PFIT2	2	1	0	0	3	0
PFIT3	2	1	0	0	3	0
PFIT4	2	1	0	0	3	0
POLYGON	0	0	48	2	0	324
PRODPL0	0	60	0	0	20	9
QR3D	145	10	0	0	155	0
QR3DBD	117	10	0	0	155	0

TABLE 6.2
Test problem characteristics

Problem	n_{fr}	n_b	n_r	n_{fx}	m_E	m_I
READING1	0	0	5	1	2	0
READING4	0	0	50	1	0	100
READING5	0	0	100	1	100	0
READING6	50	0	51	1	50	0
READING9	100	0	101	1	100	0
RK23	11	6	0	0	11	0
ROCKET	102	101	100	4	252	0
SEMICON1	0	0	10	2	10	0
SEMICON2	0	0	10	2	10	0
SINROSNB	9	0	1	0	0	18
SOSQP1	0	0	20	0	11	0
SSNLBEAM	11	0	20	2	20	0
STCQP1	0	0	17	0	8	0
STCQP2	0	0	65	0	30	0
STEENBRA	0	432	0	0	108	0
STEERING	197	1	51	7	200	0
STNQP2	0	0	65	0	30	0
SWOPF	73	0	10	0	88	14
SYNTHES2	0	2	9	0	1	14
TRAINF	200	0	200	8	202	0
TRAINH	20	0	20	8	22	0
TRUSPYR1	3	8	0	0	3	1
TWOBARS	0	0	2	0	0	2
UBH1	54	0	33	12	60	0
WATER	0	0	31	0	10	0

TABLE 6.3
Test problem characteristics