Computational Optimization and Applications, 28, 31–50, 2004

STRSCNE: A Scaled Trust-Region Solver for Constrained Nonlinear Equations*

STEFANIA BELLAVIA stefania.bellavia@unifi.it
MARIA MACCONI macconi@de.unifi.it
BENEDETTA MORINI benedetta.morini@unifi.it
Dipartimento di Energetica, University of Florence, via C. Lombroso 6/17, 50134 Florence, Italy

Received December 18, 2002; Revised May 15, 2003

Abstract. In this paper a Matlab solver for constrained nonlinear equations is presented. The code, called STRSCNE, is based on the affine scaling trust-region method STRN, recently proposed by the authors. The approach taken in implementing the key steps of the method is discussed. The structure and the usage of STRSCNE are described and its features and capabilities are illustrated by numerical experiments. The results of a comparison with high quality codes for nonlinear optimization are shown.

Keywords: constrained equations, global convergence, trust-region methods, performance profile

1. Introduction

We consider the problem of the numerical solution of bound-constrained nonlinear systems. It is standard to express these problems as

$$F(x) = 0, \quad x \in \Omega, \tag{1.1}$$

where $F(x) = (F_1(x), \ldots, F_n(x))^T$ and

$$\Omega = \{x \in \mathbb{R}^n \text{ s.t. } l \leq x \leq u\}. \tag{1.2}$$

The vectors $l \in (\mathbb{R} \cup -\infty)^n$ and $u \in (\mathbb{R} \cup \infty)^n$ are specified lower and upper bounds on the variables such that Ω has a nonempty interior. We assume that F is continuously differentiable in an open set $X \subseteq \mathbb{R}^n$ containing the n-dimensional box Ω.

Problems consisting of a nonlinear system $F(x) = 0$, nonlinear inequality constraints and, possibly, variable bounds can be stated as (1.1) by introducing bounded slack variables. It is important to note that the formulation (1.1) allows to remove discontinuities when the mapping F is not defined in the whole space $\mathbb{R}^n$.

*Work supported by MIUR, Rome, Italy, through "Cofinanziamenti Programmi di Ricerca Scientifica di Interesse Nazionale" and Gruppo Nazionale per il Calcolo Scientifico.

Frequently, systems of the form (1.1) result from different areas of scientific and engineering computations. For example, they arise when $F(x) = 0$ models a real life problem but not all the solutions of the model have physical meaning or when solutions which are expected to lie in a specific area have to be sought. In these cases, suitable bounds (1.2) can be fruitfully imposed. Significative nonlinear systems with naturally positive solutions can be found in the study of concentration of a chemical species, dimension of mechanical linkages, etc. [10, 21]. Also, there is a frequent need to deal with bound-constrained nonlinear systems in chemical process modeling and in the steady-state simulation [4, 5, 24, 25].

It is worth noting that a possible approach to (1.1) consists in reformulating it as a bound-constrained nonlinear least-squares problem:

$$\min_{x\in\Omega} f(x) = \min_{x\in\Omega} \frac{1}{2}\|F(x)\|_2^2. \tag{1.3}$$

Nonlinear least-squares problems have been a fruitful area of study [7, 12, 18, 23] and many reliable and efficient software packages designed for these problems can be employed to solve the reformulated problem (1.3). Major numerical software libraries such as NAG [22] and the `Matlab` Optimization Toolbox [20] contain robust solvers for bound-constrained nonlinear least-squares problems.

Nevertheless, well known important differences between nonlinear systems and optimization induce to study adequate algorithms for solving (1.1) in its original form. The recent activity in this area is documented, among the others, in [15, 16, 19, 26] where effective algorithms for (1.1) are given. These methods fit in the wide class of globally convergent Newton-like methods. In particular, [15, 16, 19] are based on the linesearch approach while a trust-region approach is employed in [26]. All these methods are supported by theoretical convergence analysis and illustrative computational tests show their good numerical performance. Nevertheless, we are not aware of public domain software based on the algorithms given in [15, 16, 19, 26].

Recently, the authors generalized the trust-region strategy for unconstrained systems of nonlinear equations to bound-constrained systems and proposed in [1] a new reliable method for the numerical solution of (1.1). This method, named STRN (Scaled Trust-Region Newton) method, handles the bounds in a general and reliable way and generates feasible iterates only. Thus, it avoids infeasible solutions and can deal with problems where F is not defined outside Ω. Theoretical results ensure that the method exhibits global and locally fast convergence properties. More precisely, in [1] it is shown that the convergence of the STRN method to a solution of (1.1) does not depend in a critical way on the choice of the initial guess and the ultimate rate of convergence to a solution within Ω is quadratic.

The STRN method is especially well suited for small and medium size problems and has been successfully used in the solution of various type of representative constrained systems. In particular, a large number of real-life problems was used to test its efficiency, robustness and reliability [1, 2]. In [1] a comparison with the methods proposed in [15] and [19] was carried out, too. The obtained results indicate that our method compares very well with such procedures.

The purpose of this paper is to give a well developed iterative algorithm based on the STRN method and describe its implementation in a `Matlab` solver called STRSCNE (Scaled

Trust-Region Solver for Constrained Nonlinear Equations). This solver is freely accessible through the web site: `http://ciro.de.unifi.it/STRSCNE`. With STRSCNE we intend to provide a theoretically well-founded solver that could be a valid tool for the numerical solution of (1.1).

The simplest usage of STRSCNE requires only a minimal description of the given problem and a specification of the level of accuracy required. A finite difference approximation to the Jacobian of F is provided by the code, freeing the user from computing the derivatives of F. However, if the Jacobian of F is available in analytic form, the user can provide the code to compute it. Several different output levels may be requested by the user. The convergence history of the algorithm and a variety of diagnostic information allow the user to be safeguarded against unsatisfactory approximations of the required solution.

Features and capabilities of STRSCNE have been tested by extensive numerical experiments on a number of representative real life problems. Here, we summarize the results obtained in the numerical solution of a set of bound-constrained nonlinear systems modeling physical phenomena. A comparison with high-quality software for constrained nonlinear least-squares problem have been performed, too. To this end, the well-known codes LSQNONLIN [20], E04UNF [22] and our code are tested on the same set of problems. The efficiency, robustness and reliability of the considered solvers are compared by using criteria which are independent on the programming language used. Test results are presented and analyzed. They indicate that STRSCNE turns out to be a competitive solver for bound constrained nonlinear systems.

The organization of the paper is as follows. In Section 2 we explain the key steps of our method and discuss their implementations. Section 3 presents the feature and the usage of the STRSCNE code. In Section 4 we give a brief account of the codes used in the comparative study and in Section 5 we show and discuss the results of the numerical experimentation.

1.1. Some notations

The subscript k is used as index for a sequence and when clear from the context, the argument of a mapping is omitted. Thus, for any function F, the notation F_k is used to denote $F(x_k)$ and the i-th component of x_k is denoted by x_{k_i}. The Euclidean norm of $x \in \mathbb{R}^n$ is denoted by $\|x\|$ and $\nabla f(x)$ is the gradient of the merit function $f = \frac{1}{2}\|F(x)\|^2$, i.e. $\nabla f(x) = F'(x)^T F(x)$, where $F'(x)$ is the Jacobian of F. Further, $v(x)$ denotes the vector function with components $v_i(x)$, $i = 1, \ldots, n$ given by

$$\begin{array}{lll} v_i(x) = x_i - u_i & \text{if } (\nabla f(x))_i < 0 & \text{and } u_i < \infty \\ v_i(x) = x_i - l_i & \text{if } (\nabla f(x))_i \geq 0 & \text{and } l_i > -\infty \\ v_i(x) = -1 & \text{if } (\nabla f(x))_i < 0 & \text{and } u_i = \infty \\ v_i(x) = 1 & \text{if } (\nabla f(x))_i \geq 0 & \text{and } l_i = -\infty \end{array} \tag{1.4}$$

and $D(x)$ is the diagonal scaling matrix such that

$$D(x) = diag(|v_1(x)|^{-1/2}, |v_2(x)|^{-1/2} \ldots, |v_n(x)|^{-1/2}). \tag{1.5}$$

Note that although $D(x)$ may be undefined on the boundary of Ω, $D(x)^{-1}$ can be extended continuously to it. We will denote this extension as a convention by $D(x)^{-1}$ for all $x \in \Omega$. We remark that if $\Omega \equiv \mathbb{R}^n$, $D(x)$ is the identity matrix. Further, we write $int(\Omega)$ for the interior of Ω. Finally, the symbol ϵ_m is used to denote the machine epsilon provided by the `Matlab` function `eps`.

2. The algorithm and its implementation

The algorithm implemented in the STRSCNE code is based on the STRN method given in [1]. The method generates a sequence of iterates $\{x_k\}$ belonging to the interior of Ω and employs the merit function f. At each iteration, the following main steps must be performed. First, a search direction is found by solving a suitable elliptical trust-region subproblem. Then, a step along this direction is attempted and a trial point is obtained. An acceptance mechanism is used to decide if the trial point should be retained as the next iterate and a standard trust-region updating strategy completes the iteration step.

The convergence behaviour of the STRN method was investigated in [1]. The main convergence results can be summarized by the following theorem.

Theorem 2.1. *Let $\{x_k\}$ be the sequence generated by the STRN method, $r > 0$ and $L = \bigcup_{k=0}^{\infty} \{x \in \Omega \mid \|x - x_k\| \leq r\}$. Assume that F' is Lipschitz continuous in L and $\|F'(x)\|$ is bounded above on L. Then,*

- *if $\{x_k\}$ is bounded, then all its limit points are stationary points for the problem* (1.3);
- *if $\{x_k\}$ is bounded and there exists an isolated limit point x^* such that $F'(x^*)$ is invertible and $F(x^*) = 0$, then*
  (a) *$\|F_k\| \to 0$ and $x_k \to x^*$;*
  (b) *if the limit point $x^* \in int(\Omega)$, then $x_k \to x^*$ q-quadratically.*

In the sequel we will give a detailed description of the key steps of the method and we will sketch the implemented algorithm.

2.1. Computing a trial step

Let $x_k \in int(\Omega)$ be the current iterate. In order to determine x_{k+1}, a trial step p_k is computed by solving the following elliptical trust-region subproblem

$$\min_p \{m_k(p) : \|D_k p\| \leq \Delta_k\}. \tag{2.1}$$

Here Δ_k is the trust-region size, $D_k = D(x_k)$ is the diagonal scaling matrix given in (1.5) and $m_k(p)$ is the quadratic model for the merit function f:

$$\begin{aligned} m_k(p) &= \frac{1}{2}\|F_k' p + F_k\|^2 = \frac{1}{2}\|F_k\|^2 + F_k^T F_k' p + \frac{1}{2} p^T F_k'^T F_k' p \\ &= f_k + \nabla f_k^T p + \frac{1}{2} p^T F_k'^T F_k' p. \end{aligned}$$

We remark that the Newton step p_k^N given by

$$F_k' p_k^N = -F_k, \tag{2.2}$$

solves (2.1) if Δ_k is large enough that $\|D_k p_k^N\| \leq \Delta_k$ is verified.

On the other hand, if $\|D_k p_k^N\| > \Delta_k$, we compute an approximate solution of (2.1) proceeding in the following way. First, we rescale the variable p so that the trust-region is spherical in the scaled variable. In fact, by defining $\tilde{p} = D_k p$ and substituting it in (2.1), we obtain

$$\begin{cases} \min\limits_{\tilde{p}} \tilde{m}_k(\tilde{p}) = f_k + \nabla f_k^T D_k^{-1} \tilde{p} + \dfrac{1}{2} \tilde{p}^T \left(D_k^{-1} F_k'^T F_k' D_k^{-1} \right) \tilde{p} \\ \text{subject to} \quad \|\tilde{p}\| \leq \Delta_k. \end{cases} \tag{2.3}$$

From [7, Lemma 6.4.1], this problem is solved by

$$\tilde{p}_k(\mu) = -\left(D_k^{-1} F_k'^T F_k' D_k^{-1} + \mu I \right)^{-1} D_k^{-1} \nabla f_k, \tag{2.4}$$

for the unique $\mu \geq 0$ such that $\|\tilde{p}_k(\mu)\| = \Delta_k$, unless $\|\tilde{p}_k(0)\| \leq \Delta_k$ in which case the solution of (2.3) is $\tilde{p}_k(0)$. Note that $\tilde{p}_k(0) = D_k p_k^N$ and denote $\tilde{p}_k(0)$ as $\tilde{p}_k^N$.

If $\|\tilde{p}_k^N\| = \|D_k p_k^N\| > \Delta_k$, we compute an approximate solution of (2.3) by using the dogleg method ([23]). Following this strategy, the curved trajectory $\tilde{p}_k(\mu)$, $\mu > 0$ is approximated with a path consisting of two segments. The first segment runs from the origin to the unconstrained minimizer $\tilde{p}_k^u$ of $\tilde{m}_k(\tilde{p})$ along the steepest descent direction $D_k^{-1} \nabla f_k$:

$$\tilde{p}_k^u = -\frac{\left\| D_k^{-1} \nabla f_k \right\|^2}{\left\| F_k' D_k^{-2} \nabla f_k \right\|^2} D_k^{-1} \nabla f_k, \tag{2.5}$$

while the second one connects $\tilde{p}_k^u$ to $\tilde{p}_k^N$. The dogleg method approximates the solution $\tilde{p}_k$ to (2.3) by computing the minimizer of the model $\tilde{m}_k$ along this path. Namely,

$$\tilde{p}_k = \begin{cases} \Delta_k D_k^{-1} \nabla f_k \big/ \left\| D_k^{-1} \nabla f_k \right\| & \text{if } \left\| \tilde{p}_k^u \right\| \geq \Delta_k, \\ \tilde{p}_k^u + (1 - \mu)\left(\tilde{p}_k^N - \tilde{p}_k^u \right) & \text{otherwise,} \end{cases}$$

where μ is the positive solution of the following scalar quadratic equation

$$\left\| \tilde{p}_k^u + (1 - \mu)\left(\tilde{p}_k^N - \tilde{p}_k^u \right) \right\|^2 = \Delta_k^2. \tag{2.6}$$

Finally, in order to come back in the original space and to compute an approximate solution p_k of (2.1), we simply put $p_k = D_k^{-1} \tilde{p}_k$. Note that for small values of Δ_k, the direction of p_k is the scaled steepest descent direction d_k given by

$$d_k = -D_k^{-2} \nabla f_k. \tag{2.7}$$

The above discussion is summarized as follows.

Procedure 1. Let $p_k^N, \nabla f_k, D_k, \Delta_k$ be given.
1. If $\|D_k p_k^N\| \leq \Delta_k$ then
   set $p_k = p_k^N$ and stop.
2. Compute $\tilde{p}_k^u$ by (2.5).
3. If $\|\tilde{p}_k^u\| \geq \Delta_k$ then
   set $\tilde{p}_k = \Delta_k D_k^{-1}\nabla f_k / \|D_k^{-1}\nabla f_k\|$
   else
   set $\tilde{p}_k^N = D_k p_k^N$
   compute μ solving (2.6) and set $\tilde{p}_k = \tilde{p}_k^u + (1-\mu)(\tilde{p}_k^N - \tilde{p}_k^u)$.
4. Put $p_k = D_k^{-1}\tilde{p}_k$.

2.2. *Accepting the step and updating the trust-region*

Once we have computed a trial step p_k, we look for the next iterate x_{k+1} moving from x_k along p_k. To ensure that x_{k+1} stays within Ω, we first compute the stepsize $\lambda(p_k)$ along p_k to the boundary, i.e.

$$\lambda(p_k) = \begin{cases} \infty & \text{if } \Omega = \mathbb{R}^n \\ \min\limits_{1\leq i\leq n} \Lambda_i & \text{if } \Omega \subset \mathbb{R}^n \end{cases}$$

where

$$\Lambda_i = \begin{cases} \max\left\{ \dfrac{l_i - x_{k_i}}{p_{k_i}}, \dfrac{u_i - x_{k_i}}{p_{k_i}} \right\} & \text{if } p_{k_i} \neq 0 \\ \infty & \text{if } p_{k_i} = 0 \end{cases}.$$

If $\lambda(p_k) > 1$, then $x_k + p_k$ is within Ω; otherwise, a step-back along p_k is necessary to stay within Ω. Namely, we consider the temptative iterate

$$x_{k+1} = x_k + \alpha(p_k), \tag{2.8}$$

where the trial step $\alpha(p_k)$ is defined by

$$\alpha(p_k) = \begin{cases} p_k & \text{if } \lambda(p_k) > 1 \\ \max\{\theta, 1 - \|p_k\|\}\lambda(p_k) p_k & \text{otherwise} \end{cases} \tag{2.9}$$

and $\theta \in (0, 1)$ is a fixed constant independent of k.

It is standard in a trust-region framework, to seek for a new iterate that lies within the trust-region and gives a sufficient reduction in the model. The sufficient reduction is quantified in terms of the Cauchy point p_k^c, i.e. the minimizer of m_k along the scaled steepest descent direction d_k given by (2.7), subject to satisfy the trust-region bound, i.e.

$$p_k^c = \tau_k d_k = -\tau_k D_k^{-2}\nabla f_k, \tag{2.10}$$

where $\tau_k = \text{argmin}_{\tau>0}\{m_k(\tau d_k) : \|\tau D_k d_k\| \leq \Delta_k\}$ (see [23]). It can be easily derived ([23, p. 70]) that τ_k has the following form

$$\tau_k = \min\left\{ \frac{\left\| D_k^{-1}\nabla f_k \right\|^2}{\left\| F_k' D_k^{-2}\nabla f_k \right\|^2}, \frac{\Delta_k}{\left\| D_k^{-1}\nabla f_k \right\|} \right\}. \tag{2.11}$$

Then, we test if the step $\alpha(p_k)$ satisfies the following condition:

$$\rho_k^c(p_k) = \frac{m_k(0) - m_k(\alpha(p_k))}{m_k(0) - m_k\left(\alpha\left(p_k^c\right)\right)} \geq \beta_1, \tag{2.12}$$

where $\beta_1 \in (0, 1)$ is a given constant.

Specifically, condition (2.12) guarantees that the step $\alpha(p_k)$ gives a sufficient reduction in the quadratic model m_k with respect to the Cauchy point p_k^c. We remark that the solution of the trust-region subproblem (2.1) satisfies (2.12) but if a step-back is performed, i.e. $\alpha(p_k) \neq p_k$, (2.12) is not guaranteed to hold.

To complete our rule for accepting the step $\alpha(p_k)$, we impose a good agreement between the model function m_k and the objective function f testing the following additional condition:

$$\rho_k^f(p_k) = \frac{f(x_k) - f(x_k + \alpha(p_k))}{m_k(0) - m_k(\alpha(p_k))} \geq \beta_2, \tag{2.13}$$

where $\beta_2 \in (0, 1]$ is a given constant.

Conditions (2.12) and (2.13) are forced as follows. If (2.12) does not hold, we leave the current p_k and set $p_k = p_k^c$. Then, we proceed as in the classical trust-region strategies. Namely, if condition (2.13) holds, $x_k + \alpha(p_k)$ is the next iterate. Otherwise, the trial step $\alpha(p_k)$ is rejected, the trust-region size is decreased by setting

$$\Delta_k = \min\{\alpha_1\Delta_k, \alpha_2\|D_k\alpha(p_k)\|\}, \tag{2.14}$$

for some α_1, α_2 such that $0 < \alpha_1 \leq \alpha_2 < 1$ and a new trial step is computed by using Procedure 1.

The above trial-step acceptance mechanism does not break down, i.e. an acceptable $\alpha(p_k)$ is determined within a finite number of reductions of the trust-region size (see [1, Lemma 3.4]).

Once the step $\alpha(p_k)$ is accepted, the trust-region radius is updated according to standard rules. It is well known that if the trust-region radius is too small compared with the agreement between the model and the merit function, the method misses an opportunity to take a substantial improvement in the estimate of the solution. Therefore, at the end of each iteration, we test the following condition

$$\rho_k^f(p_k) = \frac{f(x_k) - f(x_k + \alpha(p_k))}{m_k(0) - m_k(\alpha(p_k))} \geq \beta_3, \tag{2.15}$$

where $\beta_3 \in (0, 1]$ is a given constant such that $\beta_2 < \beta_3 < 1$.

If (2.15) holds, we allow possible increasing in the radius of the trust-region and we set

$$\Delta_{k+1} = \max\{\Delta_k, 2\|D_k\alpha(p_k)\|\},$$

otherwise the trust-region radius is kept the same.

2.3. The implemented algorithm

In the previous subsections we provided a detailed description of the major points to be addressed in the implementation of our method and now we give the algorithm that is the core of the STRSCNE code.

Algorithm: The Scaled Trust-Region Solver

• Inizialization:
Given $x_0 \in int(\Omega)$, $\Delta_0 > 0, \theta \in (0, 1), 0 < \alpha_1 \leq \alpha_2 < 1$,
$\beta_1, \in (0, 1], 0 < \beta_2 < \beta_3 < 1$.
• For $k = 0, 1, \ldots$ do:

1. Compute F_k.
2. Check for convergence.
3. Compute the matrix D_k by using (1.5).
4. Compute the matrix F'_k.
5. Compute p_k^N by solving the linear system (2.2).
6. Repeat
   - 6.1. Compute an approximate solution p_k of (2.1) by using Procedure 1.
   - 6.2. Compute τ_k by (2.11) and the Cauchy point p_k^c from (2.10).
   - 6.3. Compute $\alpha(p_k)$ and $\alpha(p_k^c)$ by using (2.9).
   - 6.4. Compute $\rho_k^c(p_k)$ from (2.12).
   - 6.5. If $\rho_k^c(p_k) < \beta 1$ then set $p_k = p_k^c$.
   - 6.6. Set $\Delta_k^* = \Delta_k$ and decrease Δ_k using (2.14).
   - 6.7. Compute $\rho_k^f(p_k)$ from (2.13).

   Until $\rho_k^f(p_k) \geq \beta_2$
7. Set $x_{k+1} = x_k + \alpha(p_k)$, $\Delta_k = \Delta_k^*$.
8. If $\rho_k^f(p_k) \geq \beta_3$ then
   set $\Delta_{k+1} = \max\{\Delta_k, 2\|D_k\alpha(p_k)\|\}$
   else
   set $\Delta_{k+1} = \Delta_k$.

In our implementation convergence is declared when the following condition is met:

$$\|F_{k+1}\| \leq \texttt{atol} + \texttt{rtol}\|F_k\|, \tag{2.16}$$

where `atol` and `rtol` are user supplied tolerances.

On the other hand, failure is declared if one of the following situations occurs:

Failure (1): a maximum number of iterations `maxit` are performed.
Failure (2): a maximum number of F-evaluations `maxnf` are performed.
Failure (3): the trust-region size is reduced below $\sqrt{\epsilon}_m$.
Failure (4): the relative change in the function value satisfies

$$\|F_{k+1} - F_k\| \leq 100\epsilon_m \|F_k\|.$$

Failure (5): the norm of the scaled gradient of the merit function becomes very small, i.e.

$$\left\| D_k^{-1} \nabla f_k \right\| < 100\epsilon_m.$$

Failure (6): the scaling matrix D_k cannot be computed because an overflow would be generated.

We remark that Failure 4 may indicate that the method did not manage to escape from a local minimizer of the merit function which is not a solution of (1.1), Failure 5 indicates that the sequence is approaching a minimum of f in Ω and Failure 6 occurs when the sequence is approaching a bound.

It should be stressed that we have striven to make STRSCNE as simple as possible. The simplest use of STRSCNE requires a minimal description of the problem: a user-supplied function that evaluates F and the vectors l and u that specify the lower and upper bounds. Further, the user must supply the initial guess x_0, the stopping tolerances atol and rtol, the maximum number of allowed iterations maxit and F-evaluations `maxnf`. Regarding the initial trust-region radius, the user can supply a value for Δ_0 or can adopt one of the following choices:

$$\Delta_0 = 1, \quad \text{or} \quad \Delta_0 = \left\| D_0^{-1} \nabla f_0 \right\|.$$

The Jacobian matrix F' can be either evaluated analytically by a user-supplied function or approximated using finite-differences formula provided by the code. More precisely, in the latter case, the Jacobian matrix F'_k is approximated as follows:

$$[F'_k]_j \sim \frac{1}{h_j}(F(x_k + h_j e_j) - F_k), \quad j = 1, \ldots, n,$$

where $[F'_k]_j$ denotes the j-th column of F'_k, e_j is the j-th vector of the canonic basis and

$$h_j = \begin{cases} \sqrt{\epsilon_m} & \text{if } x_{k_j} = 0 \\ \sqrt{\epsilon_m}\, sign(x_{k_j}) \max\left\{ \left| x_{k_j} \right|, \|x_k\|_1 / n \right\} & \text{otherwise .} \end{cases}$$

If the point $x_k + h_j e_j$ is not feasible, the backward approximation

$$[F'_k]_j \sim \frac{1}{h_j}(F(x_k - h_j e_j) - F_k),$$

is used.

We recall that the code performs one Jacobian evaluation at each iteration. Hence, if the matrix F_k' is approximated by finite differences, n F-evaluations are required at each iteration. At this regard, we remark that these functions evaluations are not taken into account in the limit number `maxnf` of F-evaluations.

The algebraic linear systems are solved by using the left matrix divide arithmetic operator provided by `Matlab`. This way, at each iteration, the specific algorithm employed to solve the linear system exploits the structure of the coefficient matrix F_k'.

The STRSCNE code returns a termination flag that indicates success or failure of the procedure. No matter how it stops, it returns the current iterate and an array containing the following information:

- the number of performed iterations;
- the number of performed F-evaluations;
- the 2-norm of the current value of $F(x)$;
- the 2-norm of the current value of the scaled gradient $D^{-1}(x)\nabla f(x)$;
- the total number of reductions of the trust-region radius.

Moreover, the convergence history of the algorithm is produced and stored in a matrix. For each iteration such matrix contains the 2-norm of the nonlinear residual F_k, the number of trust-region radius reductions, the step back taken to stay feasible, and the ratio of two successive nonlinear residuals.

Finally, the user can request further diagnostic information which includes the gradient of f, the rank and the singular values of the Jacobian matrix at the final iterate. These two latter quantities are computed using the `Matlab` functions `rank` and `svd` and can be useful in case of Failures (4) and (5) to establish if the algorithm got stuck onto a minimum of the merit function f. Further, such information can be used to investigate whether a computed solution within Ω approximates an isolated zero of F. In practice, it may happen that $\|F\|$ is small at an iterate x_k that is not an approximation of a zero of F, but simply an approximation of a minimum $\tilde{x}$ of f. In this situation $\tilde{x}$ can be wrongly considered as a solution of (1.1); on the other hand, if $\tilde{x} \in int(\Omega)$ and $F(\tilde{x}) \neq 0$, then $F'(\tilde{x})$ must be singular.

We close this section indicating the fixed choices of the constants involved. After extensive computational experiments we decided to set: $\theta = 0.99995, \alpha_1 = 0.25, \alpha_2 = 0.5, \beta_1 = 0.1, \beta_2 = 0.25, \beta_3 = 0.75$.

2.4. Convergence to a solution on the boundary of Ω

Let us consider the case in which problem (1.1) has a solution x^* that lies on the boundary of the feasible set. Theorem 2.1 states that if x^* is a limit point of the sequence $\{x_k\}$ generated by the STRN method and if $F'(x^*)$ is non singular, then $\{x_k\}$ converges to x^*. Further, when x^* lies in the interior of Ω the local convergence rate is quadratic. If no assumptions on the position of x^* are made, local convergence properties of the STRN method are summarized by the following result.

Theorem 2.2. *Let $\{x_k\}$ be the sequence generated by the STRN method, and ζ_k be such that $\alpha(p_k^N) = \zeta_k p_k^N$. Assume that there exists a solution x^* of* (1.1) *such that $F'(x^*)$ is nonsingular. If the sequence $\{x_k\}$ generated by the STRN method converges to x^* and eventually p_k^N solves* (2.1) *and satisfies* (2.12) *and* (2.13), *then*
(a) *If ζ_k is bounded away from zero, $\|F(x_k)\| \to 0$ q-linearly.*
(b) *If $\zeta_k \to 1$, then $x_k \to x^*$ superlinearly.*
(c) *If eventually $\zeta_k = 1$, then $x_k \to x^*$ quadratically.*

The proof of this theorem can be found in [1]. From this result, it follows that local quadratic convergence to a solution on the boundary of the box is not guaranteed. This must be ascribed to the fact that eventually the Newton step p_k^N is not ensured to solve the trust-region subproblem (2.1). Moreover, even if p_k^N solves (2.1) it could be necessary to truncate it in order to maintain strict feasibility, [1, 13]. This will prohibit fast local convergence unless eventually the scaling produces a step $\alpha(p_k^N) = \zeta_k p_k^N$ such that ζ_k goes to 1. In this case, the convergence rate of $\{x_k\}$ is superlinear.

3. Overview of the codes used for the comparison

In order to assess the validity of STRSCNE, it was compared with high-quality codes for the reformulated constrained nonlinear least-square problem (1.3). Specifically it was compared with LSQNONLIN [20] and E04UNF [22]. Since these solvers adopt complementary approaches to bound-constrained nonlinear least-square problems, the obtained experimental results can be useful to assess whether our direct approach to (1.1) is competitive with different approaches to the reformulated least square problems.

In this section we give a brief account of the codes employed in our comparative study. For more details, we address the reader to the cited references.

The `Matlab` Optimization Toolbox ([20]) covers the problem of solving (1.3) by means of the function LSQNONLIN. This code provides a large-scale algorithm and a medium-scale algorithm and the former one is the default choice. Since the medium-scale algorithm handles only the case $\Omega = \mathbb{R}^n$, it will not be considered in the sequel.

The large-scale procedure LSQNONLIN exploits an elliptical subspace trust-region strategy based on the interior Newton method proposed in [3]. It produces only points belonging to the interior of Ω. Letting $x_k \in int(\Omega)$, the iterative process searches for the next iterate by computing an approximate solution to the trust-region subproblem

$$\min_s \{\psi_k(s) : \|D_k s\| \leq \Delta_k\}. \tag{3.1}$$

Here Δ_k is the trust-region size and $\psi_k(s)$ is the quadratic function

$$\psi_k(s) = \nabla f_k^T s + \frac{1}{2} s^T \left(F_k'^T F_k' + D_k \, diag(\nabla f_k) J_k^v D_k\right) s,$$

where J_k^v is the Jacobian matrix at x_k of the function $|v(x)|$ given in (1.4). It is easy to see that the unconstrained minimizer of ψ_k, i.e. the full Newton step s_k^N,

satisfies

$$\left(D_k^{-2} F_k'^T F_k' + \ diag(\nabla f_k) J_k^v\right) s_k^N = -D_k^{-2} \nabla f_k. \tag{3.2}$$

Now, taking into account the special form of f and recalling that the solutions of problem (1.1) solve the unconstrained nonlinear system

$$D^{-2}(x) \nabla f(x) = 0, \tag{3.3}$$

s_k^N can be viewed as the step generated by a Newton-like method for this system. Specifically, solving (3.3) by a Newton-like method where the Hessian matrix of f is approximated by $F_k'^T F_k'$, gives rise to the linear system (3.2). Hence, we can conclude that LSQNONLIN is based on a method that has some similarities with our approach but it is designed for problem (3.3) instead of (1.1).

At each iteration, LSQNONLIN performs the following main steps. First, the Projected Coniugate Gradient method is used to solve the linear system (3.2) and an approximation to the Newton step s_k^N is computed. Then, instead of searching for the solution of the full trust-region subproblem (3.1), a solution to (3.1) restricted to a two-dimensional subspace is determined. The adopted two-dimensional subspace is spanned by the scaled gradient $D_k^{-2} \nabla f_k$ and the computed Newton step s_k^N. After computing the trust-region solution, this vector is reflected by using the strategy proposed in [6].

The trust region solution, its reflection and the scaled gradient, possibly truncated as in (2.9) to maintain strict feasibility, are exploited to define the trial step s_k. Specifically, the trial step is chosen as the step that gives the lower value of f. Finally, if the trial step gives a reduction on f with respect to the current value f_k according to (2.13), the step is accepted otherwise it is rejected and the trust-region is shrunk.

LSQNONLIN terminates successfully either if the norm of the current step is less than a prescribed tolerance, i.e.

$$\|x_{k+1} - x_k\| \leq \epsilon_1, \tag{3.4}$$

or if the relative change in the function value satisfies

$$\|F_{k+1} - F_k\| \leq \epsilon_2 \|F_k\|, \tag{3.5}$$

or if the sequence $\{x_k\}$ is judged to have converged to a point that satisfies the first-order optimality condition, i.e.

$$\left\| D_k^{-2} \nabla f_k \right\|_\infty \leq \epsilon_2,$$

and no negative curvature is detected.

EO4UNF is a FORTRAN77 code provided by the NAG library [22]. It is designed to minimize a smooth sum of squares of functions subject to constraints which may include simple bounds, linear constraints and smooth nonlinear constraints. Namely, E04UNF solves the

following problem:

$$\min_{x\in\mathbb{R}^n} \phi(x) = \frac{1}{2}\sum_{i=1}^{m}(y_i - F_i(x))^2, \quad \text{subject to} \quad \hat{l} \le \begin{Bmatrix} x \\ Ax \\ c(x) \end{Bmatrix} \le \hat{u}, \tag{3.6}$$

where y_i are constants and $F_i : \mathbb{R}^n \to \mathbb{R}$ are nonlinear functions, $i = 1, \ldots, m$. Concerning the constraints, $A \in \mathbb{R}^{p\times n}$ is a constant matrix, $c : \mathbb{R}^n \to \mathbb{R}^q$ is the nonlinear constraint function, $\hat{l} \in (\mathbb{R} \cup -\infty)^{n+p+q}$, $\hat{u} \in (\mathbb{R} \cup \infty)^{n+p+q}$ are specified lower and upper bounds. If $m = n$ and all the scalars y_i are null, we have $\phi \equiv f$ in (1.3). If, in addition, the matrix A and the vector c are empty, (3.6) reduces to (1.3). In the sequel we will focus on this particular case.

E04UNF is not designed for large and sparse problems since it treats all matrices as dense. The code employs a sequential quadratic programming (SQP) method. The basic structure of E04UNF involves major and minor iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that is intended to converge to a point that satisfies the first-order conditions for optimality of (3.6). At the k-th iteration, with $x_k \in \Omega$ at hand the new iterate x_{k+1} is determined as $x_{k+1} = x_k + \alpha p_k$ where the steplength α is a non-negative scalar and the search direction p_k is the solution of the quadratic programming problem:

$$\min_p \nabla f(x_k)^T p + \frac{1}{2} p^T H_k p, \quad \text{subject to} \quad \hat{l} - x_k \le p \le \hat{u} - x_k. \tag{3.7}$$

The matrix H_k is the positive-definite approximation to the Hessian of f at x_k computed by the BFGS (Broyden-Fletcher-Godfart-Shanno) quasi-Newton approximation [12]. Since solving a quadratic programming problem is itself an iterative procedure, the minor iterations are the iterations performed to solve (3.7). In E04UNF the method used for solving (3.7) is an active set type procedure where a given initial working set is iteratively updated and used to determine a descent direction for the quadratic problem (3.7). The obtained final working set suggests an initial working set for the next quadratic problems. In practice, this usually allows the subproblems to become optimal in only one minor iteration as a solution is approached (see [11, 22]). After computing the search direction p_k, the major iteration proceeds by determining a steplength α that produces a sufficient decrease in the merit function ϕ.

E04UNF terminates successfully if the sequence $\{x_k\}$ is judged to have converged and x_k is considered to satisfy the first-order optimality conditions for a minimum. Formally these conditions are expressed by

$$\|x_{k+1} - x_k\| \le \sqrt{\epsilon_3}(1 + \|x_k\|),$$

and

$$\|\nabla_{P,FR} f_k\| \le \sqrt{\epsilon_3}(1 + \max(1 + \|f_k\|^2, \|\nabla_{FR} f_k\|)),$$

where ϵ_3 is a user-defined tolerance, $\nabla_{FR} f(x)$ is the gradient of f with respect to the free variables, i.e. to the components of x that are within the bounds, and $\nabla_{P,FR} f(x)$ is the gradient of f projected on the feasible set with respect to the free variables.

4. Numerical results

This section summarizes the results of the numerical experimentation we carried out in order to verify capabilities and features of STRSCNE. Numerical experiments were performed on an HP9000 C200 workstation with epsilon machine $\epsilon_m \sim 10^{-16}$.

We performed tests using 48 problems. In particular, 45 problems come from the extensive library NLE [24] which is accessible through the web site: `www.polymath-software.com/library`.

The NLE library provides a unifying source of problems for testing the performance of numerical software. It contains over 70 real-life problems whose dimension ranges from a single equation to 14 equations. All the problems arise from mathematical models related to physical phenomena such as equations of state, chemical and phase equilibrium, pipeline flow, flowsheeting, steady state material and energy balance on a reactor, reaction rate equations and free energy minimization. For our numerical experiments we focused on the solution of the 45 problems having dimension $n \geq 2$. Note that all these problems have solutions belonging to the interior of Ω. In [24] each problem is assigned with an identification name. Such name is associated with the dimension of the problem and with a progressive number. In the sequel, we will refer to these problems using these names.

In order to show the performance of the code on problems with solutions on the boundary of Ω, we solved several complementarity problems reformulated as systems of smooth bound-constrained nonlinear equations, [27]. Here we report the results obtained on two well-known complementarity problems, i.e. the Kojima-Shindo problem [8] whose dimension is $n = 4$ and the Problem 118 of the Hock-Schittkowski collection [14] whose dimension is $n = 133$. We denote them as KS and HS118 problem, respectively. We remark that the given results are representative of the performed experiments on complementarity problems.

Finally, to show the numerical behavior of STRSCNE on larger problems, we applied the code to a problem of dimension $n = 451$ denoted `MNBVP` (Mildly-Nonlinear BVP) and given in [19, Problem 7].

The set of problems for testing the numerical performance of our code contains 48 problems whose dimension ranges from $n = 2$ to $n = 451$. The collected problems provided us with various type of representative constrained systems. In fact, several nonlinear mappings have discontinuities in the domain Ω, show poor-scaling or ill-conditioning. Furthermore, the set of tests includes systems with solution on the boundary of the feasible set, systems with both free and constrained variable components, systems with only lower (upper) bounds, systems with variable components bounded both from above and below. We point out that in the description of the problems given in [24], the specified constraints are of physical nature. Then, the physical validity of the computed solutions is guaranteed by a code that generates feasible iterates. Regarding problems `Twoeq2`, `Twoeq3`, `Twoeq6` and `Twoeq8` we added suitable bounds in order to avoid discontinuities of the function defining the nonlinear system.

We remark that the NLE Library is designed to be a useful tool to access the reliability and the robustness of nonlinear equations software. Then, known solutions are included in the library for all problems and each problem is provided with several good and poor initial guesses. Since our method requires strictly feasible starting points, we did not consider the infeasible starting guesses. Further, initial guesses on the boundary of Ω were forced to lie in Ω perturbing the components on the bounds by a quantity of the order of ϵ_m. Regarding problems KS and HS118 we used the starting guesses $x_0 = 10^{\gamma}(1, \ldots, 1)^T$ with $\gamma = 0, 1, 2$. Finally, problem MNBVP was solved starting from $x_0 = l + 0.25\gamma(u - l)$ with $\gamma = 1, 2.5, 3$. Summarizing, each problem gives rise to several tests and the total number of performed tests is 161.

The results reported here are obtained running STRSCNE with the stopping tolerances atol $= 10^{-8}$ and rtol $= 0$, and setting maxit $= 1000$ and maxnf $= 1000$. We did not supply the analytical Jacobian F'. Therefore, derivatives were estimated by the finite-differences approximations provided by the code.

Regarding the initial trust-region radius, we ran the code using both $\Delta_0 = 1$ and $\Delta_0 = \|D_0^{-1}\nabla f_0\|$.

The results obtained with $\Delta_0 = 1$ appear in Table 1 where the problems are numbered in progressive order from 1 to 48. For each problem we list the number NT of performed tests and the number NS of tests successfully solved. Further, for successful runs we report the average number AIT of performed iterations and the average number AFE of performed F-evaluations. We remark that AFE does not include F-evaluations required to approximate the Jacobian matrix F'.

We make the following observations based on Table 1. On a total of 161 tests, the solver successfully ended 127 times. Eight problems were solved starting from only one of the given initial guesses and 30 problems were solved starting from all the used starting points. STRSCNE failed in solving problems Sixeq1 and Sixeq4a which are classified as high difficulty problems in the library. The first problem is seriously badly-scaled while in the second one the function F has discontinuities within the feasible region.

Most of the problems were cheaply solved. At this regard, we remark that AFE is less than or equal to 30 for 39 problems and exceeds 100 only four times. Note that AFE is almost the same as AIT in most cases, i.e. the majority of the problems were solved with a few reductions of the trust-region radius. In particular, we have AFE $=$ AIT $+ k$, with $1 \leq k \leq 41$ and $k \geq 10$ only in four cases.

Finally, we observe that the solver performs quite well also on problems KS and HS118. This is remarkable since the solutions of these problems lie on the boundary of Ω and fast convergence of the STRN method is guaranted only when the limit point of the generated sequence belongs to the strict interior of Ω. A closer look to the numerical behaviour of the code reveals that the truncated Newton step is eventually taken. In practice, case b) of Theorem 2.2 occured and local superlinear convergence was numerically detected for all the runs except for HS118 with the initial guess $x_0 = 10^2(1, \ldots, 1)^T$. In this run, case a) of Theorem 2.2 was detected.

The behaviour of STRSCNE was slightly affected by the choice of the initial trust-region radius. Choosing $\Delta_0 = \|D_0^{-1}\nabla f_0\|$, STRSCNE solved 128 tests. The tests where NS and/or AFE were subject to major modifications are presented in Table 2.

Table 1. Performance of the STRSCNE code with $\Delta_0 = 1$.

Pb	NT	NS	AIT	AFE	Pb	NT	NS	AIT	AFE
1. Twoeq1	2	2	330	345	25. Sixeq2a	2	1	4	5
2. Twoeq2	4	2	8	9	26. Sixeq2b	2	1	4	5
3. Twoeq3	5	3	9	10	27. Sixeq2c	2	1	4	5
4. Twoeq4a	3	3	4	5	28. Sixeq3	4	2	10	12
5. Twoeq4b	3	3	4	5	29. Sixeq4a	3	0		
6. Twoeq5a	4	4	5	7	30. Sixeq4b	4	3	344	385
7. Twoeq5b	4	4	6	8	31. Seveneq1	3	3	21	30
8. Twoeq6	4	4	9	12	32. Seveneq2a	3	1	4	5
9. Twoeq7	4	4	7	9	33. Seveneq2b	3	3	71	78
10. Twoeq8	4	3	5	6	34. Seveneq3a	4	4	9	10
11. Twoeq9	4	4	308	344	35. Seveneq3b	3	2	9	10
12. Twoeq10	3	3	6	7	36. Seveneq4	4	4	14	15
13. Threeq1	4	4	22	28	37. Nineq1	4	3	23	27
14. Threeq2	4	1	5	6	38. Teneq1a	2	2	14	15
15. Threeq3	4	4	10	11	39. Teneq1b	3	3	36	42
16. Threeq4a	4	1	5	6	40. Teneq2a	3	3	10	11
17. Threeq4b	4	4	7	9	41. Teneq2b	3	3	14	15
18. Threeq5	4	1	20	28	42. Teneq3	2	2	5	6
19. Threeq6	4	4	104	145	43. 11eq1	3	3	15	16
20. Threeq7	5	4	17	20	44. 13eq1	2	2	16	22
21. Threeq8	1	1	6	7	45. 14eq1	2	2	12	13
22. Foureq1	5	5	9	12	46. KS	3	3	14	15
23. Fiveeq1	4	2	10	11	47. HS118	3	3	55	56
24. Sixeq1	4	0			48. MNBVP	3	3	19	20

Table 2. Significative results of the STRSCNE code with $\Delta_0 = \|D_0^{-1}\nabla f_0\|$.

Pb	NT	NS	AIT	AFE	Pb	NT	NS	AIT	AFE
3. Twoeq3	5	4	9	10	33. Seveneq2b	3	1	4	5
15. Threeq3	4	4	5	6	36. Seveneq4	4	4	4	5
18. Threeq5	4	3	6	8	37. Nineq1	4	4	65	71
19. Threeq6	4	4	72	110	43. 11eq1	3	3	7	8
20. Threeq7	5	5	47	54	44. 13eq1	2	2	7	8
22. Foureq1	5	4	5	6	45. 14eq1	2	2	140	174
28. Sixeq3	4	3	7	9	46. KS	3	1	38	42
30. Sixeq4b	4	3	7	7	47. HS118	3	3	14	15
31. Seveneq1	3	3	48	56					

In what follows we compare the code under study with E04UNF and LSQNONLIN giving an account of their computational effort and robustness. In this analysis, runs of STRSCNE associated to the choice $\Delta_0 = 1$ will be considered. We remark that, since we do not compare the performance of the codes using runtime or storage requirements, the performed numerical comparison is not dependent on the programming language used.

We ran E04UNF and LSQNONLIN without providing the analytical Jacobian of F, therefore the codes approximated the Jacobian F' by using finite differences. Except for the derivative approximations, all the presented results were obtained running the codes with their default settings and stopping tolerances. For sake of comparison, a limit of 1000 F-evaluations were imposed for each test without taking into account F-evaluations due to derivative approximations.[1] Codes were considered to fail if 1000 F-evaluations were not enough to satisfy the stopping criteria or if the computed solution is a minimum of f that is not a zero of F.

Figure 1 displays the robustness of the compared codes. For each code we plot the number NS of tests successfully solved as a function of the problem number. The dashed

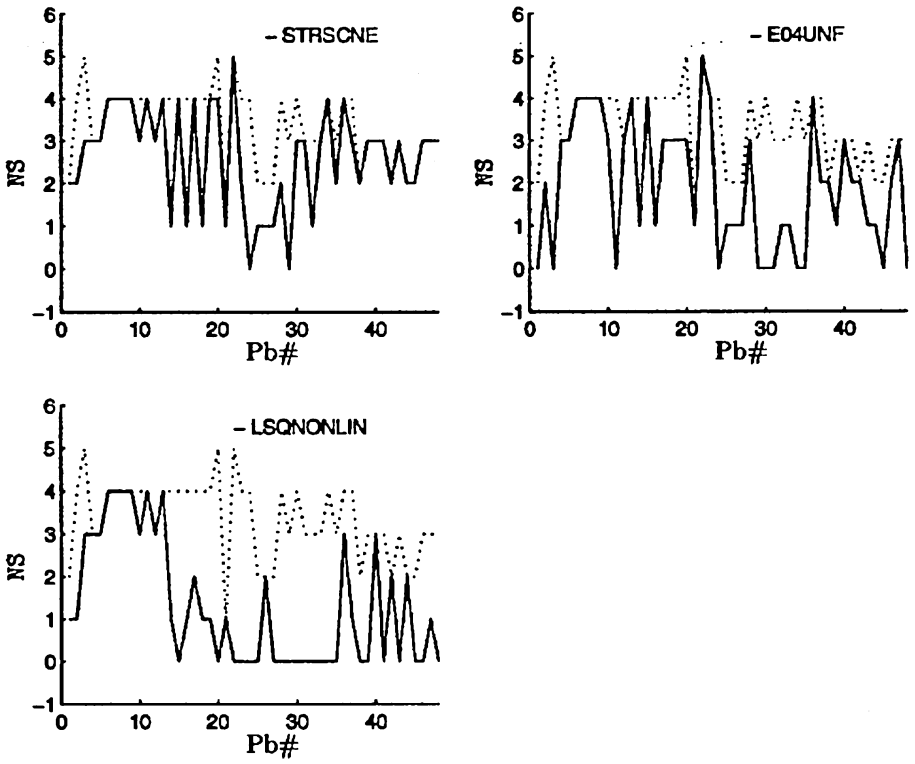

Figure 1. Performance, in terms of number of successfully solved tests, of all of the compared solvers.

line represents the behaviour of an ideal solver that successfully solves all the tests while the solid line shows the behaviour of the tested codes. It should be stressed that all the codes failed in solving problems `Sixeq1` and `Sixeq4a` starting from all the given initial guesses. This confirms the fact that they are highly difficult problems. On the other hand, 42 tests were successfully solved by all the codes and six problems were solved by all the codes for all the supplied initial guesses.

To visualize the overall behaviour of the codes, we exploit the performance profile proposed by Dolan and Moré [9]. In this approach, the profile of each code was measured considering the ratio of its computational effort versus the best computational effort of all of the codes. These profiles were introduced as an unifying tool for evaluating and comparing the performance of optimization software. Here we use the number of performed function evaluations as a measure of the computational effort.

Specifically, for each test t and solver s, let $FE_{t,s}$ denote the number of F-evaluations required to solve test t by solver s. Also, let FE_t^* be the lowest number of F-evaluations required by the three codes to solve test t. Then, the ratio

$$r_{t,s} = \frac{FE_{t,s}}{FE_t^*},$$

measures the performance on test t by solver s with the best performance by any solver on such test. Clearly, $r_{t,s} \geq 1$ and $r_{t,s} = 1$ means that the solver s was the most convenient in solving test t. Finally, for each code s the performance profile is defined as

$$\pi_s(\tau) = \frac{\text{no. of tests s.t. } r_{t,s} < \tau}{\text{total no. of tests}}, \quad \tau \geq 0.$$

Using this approach, there is no need to discard solver failures from the data. In fact, if a code does not solve test t, $r_{t,s}$ is assigned a large number, say r_m. In our experiments we have $r_{t,s} > 100$ only in one case and setting $r_m = 200$ we capture the overall performance of all the solvers. Figure 2 shows the performance profiles of the three codes in the intervals [0,10] and [0,200]. From the values of π_s (1), it is clear that STRSCNE had the most wins, i.e. it solved about 67% of the tests with the greatest efficiency. If we focus our attention on the ability of completing a run successfully we have again that STRSCNE stands out. In particular, the performance profile of STRSCNE readily flattens and our code has the ability to solve over 78% of the tests within a factor 5 of the best solver.

Summarizing our comparative study, we used a large set of problems that occur in applications. The obtained results suggest that STRSCNE constitutes a powerful approach for solving small bound-constrained nonlinear systems. It outperforms both in terms of robustness and efficiency well known codes for solving problem (1.1) reformulated as a bound-constrained least-square problem. Anyway, we are aware that our study cannot be considered conclusive since the software packages used in our testing are regularly updated and the performance of a particular solver may vary sensitively if non default options are used.

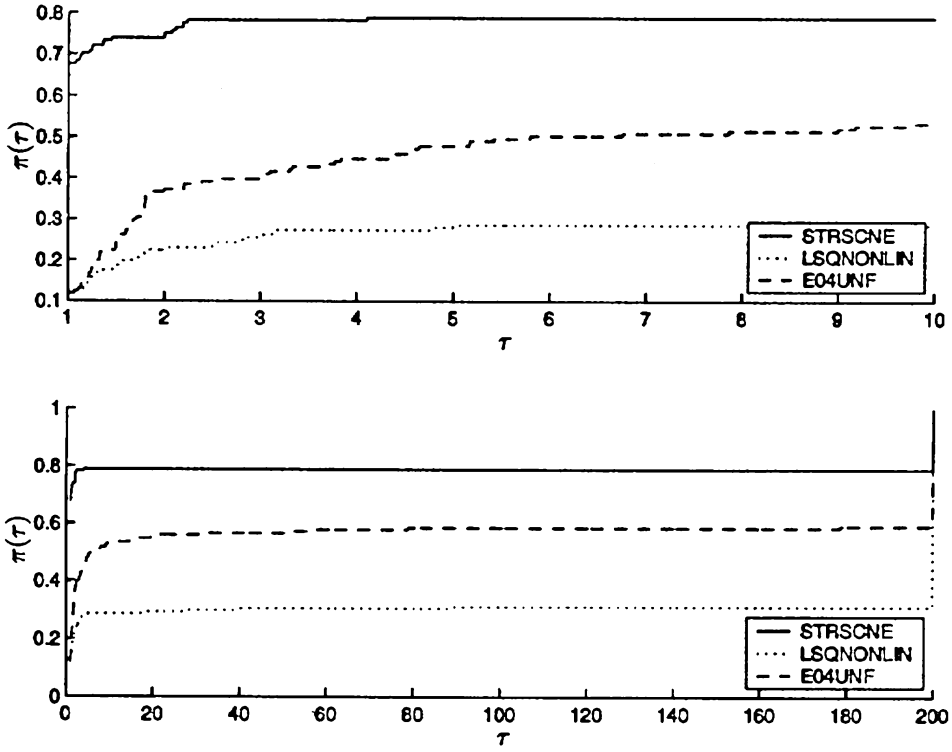

Figure 2. Performance profile.

Note

1. LSQNONLIN provides on exit the total number of performed F-evaluations, including those due to the Jacobian approximations. In the presented results we adjusted for this by discounting the extra evaluations done for approximating the Jacobian.

References

1. S. Bellavia, M. Macconi, and B. Morini, "An affine scaling trust-region method approach to bound-constrained nonlinear systems," Applied Numerical Mathematics, vol. 44, pp. 257–280, 2003.
2. S. Bellavia, M. Macconi, and B. Morini, "Numerical solution of bound-constrained nonlinear systems in chemical engineering," in Recent Advances in Computational Science and Engineering, H.P. Lee and K. Kumar (Eds.), Imperial College Press, 2002.
3. M.A. Branch, T.F. Coleman, and Y. Li, "A subspace, interior, and conjugate gradient method for large-scale bound minimization problems," SIAM J. Optim., vol. 6, pp. 418–445, 1996.
4. L.G. Bullard and L.T. Biegler, "Iterative linear programming strategies for constrained simulation," Comp. & Chem. Eng., vol. 15, pp. 239–254, 1991.
5. S.A. Burns and K.M. Mueller, "Solving systems of non-linear equations with both free and positive variables," Int. J. for Numerical Methods in Engineering, vol. 46, pp. 1987–1996, 1999.

6. T.F. Coleman and Y. Li, "On the convergence of interior-reflective newton methods for nonlinear minimization subject to bounds," Math. Programming, vol. 67, pp. 189–224, 1994.
7. J.E. Dennis and R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice Hall: Englewood Cliffs, NJ, 1983.
8. S.P. Dirkse and M.C. Ferris, "MCPLIB: A collection of nonlinear mixed complementary problems," Technical Report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994.
9. E.D. Dolan and J.J. Moré, "Benchmarking optimization software with performance profiles," Mathematical Programming, vol. 91, pp. 201–213, 2002.
10. C.A. Floudas et al., Handbook of Test Problems in Local and Global Optimization, Kluwer Academic Publishers, Nonconvex Optimization and its Applications, vol. 33, 1999.
11. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "Users's guide for NPSOL (Version 5.0): A Fortran package for nonlinear programming," Report SOL 86-2, Department of Operations Research, Standford University, 1986.
12. P.E. Gill, W. Murray, and M.H. Wright, Practical Optimization, Academic Press, 1981.
13. M. Heinkenschloss, M. Ulbrich, and S. Ulbrich, "Superlinear and quadratic convergence of affine-scaling interior-point Newton methods for problems with simple bounds without strict complementarity assumption," Math. Program., vol. Ser. A 86, pp. 615–635, 1999.
14. W. Hock and K. Schittkowski, "Test examples for nonlinear programming codes," Lecture Notes in Economics and Mathematical Systems, vol. 187, 1981.
15. C. Kanzow, "An active set-type Newton method for constrained nonlinear systems," in Complementarity: Applications, Algorithms and Extensions, M.C. Ferris, O.L. Mangasarian, and J.S. Pang (Eds.), Kluwer Academic Publishers, 2001, pp. 179–200.
16. C. Kanzow, "Strictly feasible equation-based methods for mixed complementarity problems," Numer. Math., vol. 89, pp. 135–160, 2001.
17. C. Kanzow, N. Yamashita, and M. Fukushima, "Levenberg-Marquardt methods for constrained non-linear equations with strong local convergence properties," Preprint 244, Institute of Applied Mathematics and Statistics, University of Wurzburg, 2002.
18. C.T. Kelley, "Iterative Methods for optimization," Frontiers in Applied Mathematics, SIAM, 1999.
19. D.N. Kozakevich, J.M. Martinez, and S.A. Santos, "Solving nonlinear systems of equations with simple bounds," Comput. Appl. Math., vol. 16, pp. 215–235, 1997.
20. MATLAB 6.1, The MathWorks, Natick, MA.
21. K. Meintjes and A.P. Morgan, "Chemical equilibrium systems as numerical tests problems," ACM Trans. Math. Soft., vol. 16, pp. 143–151, 1990.
22. NAG Fortran Library Manual, Mark 18, NAG Ltd., 1997.
23. J. Nocedal and S.J. Wright, Numerical Optimization, Springer Series in Operations Research, 1999.
24. M. Shacham, N. Brauner, and M. Cutlib, "A web-based library for testing performance of numerical software for solving nonlinear algebraic equations," Comp & Chem. Eng., vol. 26, pp. 547–554, 2002.
25. M. Shacham, "Numerical solution of constrained nonlinear algebraic equations," Int. J. for Numerical Methods in Engineering, vol. 23, pp. 1455–1481, 1986.
26. M. Ulbrich, "Nonmonotone trust-region methods for bound-constrained semismooth equations with applications to nonlinear mixed complementarity problems," SIAM J. Optim., vol. 11, pp. 889–917, 2000.
27. T. Wang, R.D.C. Monteiro, and K.J.S. Pang, "A potential reduction Newton method for constrained equations," SIAM J. Optim., vol. 9, pp. 729–754, 1999.