Accelerated derivative-free spectral residual method for nonlinear systems of equations*

E. G. Birgin†  J. L. Gardenghi‡  D. S. Marcondes§  J. M. Martínez¶

April 26, 2021

Abstract

Spectral residual methods are powerful tools for solving nonlinear systems of equations without derivatives. In a recent paper, it was shown that an acceleration technique based on the Sequential Secant Method can greatly improve its efficiency and robustness. In the present work, an R implementation of the method is presented. Numerical experiments with a widely used test bed compares the presented approach with its plain (i.e. non-accelerated) version that makes part of the R package BB. Additional numerical experiments compare the proposed method with NITSOL, a state-of-the-art solver for nonlinear systems. The comparison shows that the acceleration process greatly improves the robustness of its counterpart included in the existent R package. As a by-product, an interface is provided between R and the consolidated CUTEst collection, which contains over a thousand nonlinear programming problems of all types and represents a standard for evaluating the performance of optimization methods.

Key words: nonlinear systems, derivative-free, sequential residual methods, sequential secant approach, acceleration, numerical experiments.

1 Introduction

Solving nonlinear systems of equations is an ubiquitous problem that appears in a wide range of applied fields such as Physics, Chemistry, Engineering, and Statistics, just to name a few. Moreover, many times, equations are computed using black-box codes and derivatives are not available. Thus, derivative-free solution methods are in order.

Given $F : \mathbb{R}^n \to \mathbb{R}^n$, we consider the problem of finding $x \in \mathbb{R}^n$ such that

$$F(x) = 0,$$

(1)

without making use of derivatives. Observing that (1) is equivalent to $x = x - \sigma F(x)$, for any $\sigma > 0$, Sequential Residual Methods (namely SANE and DF-SANE) based on the iteration $x^{k+1} = x^k - \sigma_k F(x^k)$, where

$$\sigma_k = \frac{||s^{k-1}||^2}{(y^{k-1})^T s^{k-1}}, \quad s^{k-1} = x^k - x^{k-1}, \quad \text{and} \quad y^{k-1} = F(x^k) - F(x^{k-1})$$

were introduced in [13] and [13]. These methods were inspired by the Barzilai-Borwein step of minimization methods; see [2, 17, 18]. Although very popular, in part due to its simplicity, these methods may suffer from slow

---

*This work was supported by FAPESP (grants 2013/07375-0, 2016/01860-1, and 2018/24293-0) and CNPq (grants 302538/2019-4 and 302682/2019-8).
†Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. e-mail: egbirgin@ime.usp.br
‡Faculty UnB Gama, University of Brasília, Área Especial de Indústria Projeção A, Setor Leste, Gama, 72444-240, Brasília, DF. e-mail: john.gardenghi@unb.br.
§Department of Applied Mathematics, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. e-mail: diaulas@ime.usp.br.
¶Department of Applied Mathematics, Institute of Mathematics, Statistics, and Scientific Computing (IMECC), State University of Campinas, 13083-859 Campinas SP, Brazil. e-mail: martinez@ime.unicamp.br.
convergence. On the other hand, their simple and fast iterations made them an adequate choice to provide a global convergent framework to the Sequential Secant approach \cite{[11][22]}. This choice was explored in \cite{[3]}, where the Accelerated DF-SANE method was introduced. Numerical experiments in \cite{[3]} shown that Accelerated DF-SANE compares favorably to the classical truncated Newton approach implemented in the package NITSOL \cite{[16]}, when applied to large-scale problems coming from the discretization of partial differential equations.

In the present work, an R \cite{[20]} implementation of Accelerated DF-SANE is introduced. Numerical experiments in \cite{[3]} are complemented with numerical experiments using the widely-used testing environment for optimization CUTEst \cite{[11]}. Problems in the CUTEst collection are given in SIF (Standard Input Format; see \cite{[9]} Chapters 2 and 7) and a decoder named SifDec translates the problem into Fortran routines. Therefore, in order to be able to use the CUTEst collection, an interface with the R language is required. Such interface is introduced in the present work; and the authors hope that its dissemination in the R community could help in testing and assessing the performance of optimization methods developed in R. Classical sets of problems, like the ones introduced in \cite{[15]} and \cite{[12][19]}, are included in the CUTEst collection. In addition to the comparison with NITSOL, a comparison with the DF-SANE method implemented within the BB package \cite{[21]} implemented in R is also provided.

The rest of this work is organized as follows. The Accelerated DF-SANE method and its convergence theoretical results are condensed in Section \ref{sec:2}. The R implementation of the method and its usage are described in Section \ref{sec:3}. Numerical results are reported in Section \ref{sec:4}. Conclusions are given in the last section.

## 2 Accelerated DF-SANE

In this section, the Accelerated DF-SANE method introduced in \cite{[3]} and its theoretical convergence results are summarized. Roughly speaking, Accelerated DF-SANE performs a nonmonotone line search along the direction of the residue. As a result of a double backtracking, at each iteration $k$, a trial point $x_{trial}$ is first computed. Before deciding whether this trial point will be the next iterate $x^{k+1}$ or not (as it would be the case in the plain DF-SANE in which acceleration is not performed), an accelerated point $x_{accel}^{k+1}$ is computed. Following sequential secant ideas, $x_{accel}^{k+1}$ is given by $x_{accel}^{k+1} = x^k - S_k Y_k^\dagger F(x^k)$, where $p > 1$ is a given parameter, $k = \max\{0, k-p+1\}$,

$$s^j = x^{j+1} - x^j \text{ for } j = k, \ldots, 1,$$
$$y^j = F(x^{j+1}) - F(x^j) \text{ for } j = k, \ldots, 1,$$
$$s^k = x_{trial}^{k+1} - x^k,$$
$$y^k = F(x_{trial}^{k+1}) - F(x^k),$$
$$S_k = (s^k, \ldots, s^{k-1}, s^k),$$
$$Y_k = (y^k, \ldots, y^{k-1}, y^k),$$

and $Y_k^\dagger$ is the Moore-Penrose pseudoinverse of $Y_k$. Then, if $\|F(x_{accel}^{k+1})\|_2^2 < \|F(x_{trial}^{k+1})\|_2^2$, the methods defines $x_{accel}^{k+1} = x_{accel}^{k+1}$, while $x_{accel}^{k+1} = x_{trial}^{k+1}$ in the other case. In practice, $x_{accel}^{k+1}$ is computed by first finding the minimum norm least-squares solution $\tilde{\nu}$ of the linear system $Y_k \tilde{\nu} = F(x_{trial}^{k+1})$ and then defining $x_{accel}^{k+1} = x_{trial}^{k+1} - S_k \tilde{\nu}$. The minimum-norm least-squares solution $\tilde{\nu}$ is computed with a complete orthogonalization of $Y_k$. The key point is that matrix $Y_k$ corresponds to remove one column and add one column to matrix $Y_{k-1}$, keeping the cost of each iteration low; see \cite{[3]} \S 5.4 for details. The whole Accelerated DF-SANE method is given in the algorithm that follows.

### Algorithm 2.1: Accelerated DF-SANE

**Input.** Let $\gamma \in (0, 1)$, $0 < \sigma_{\min} < \sigma_{\max} < \infty$, $0 < \tau_{\min} < \tau_{\max} < 1$, positive integers $M$ and $p$, a sequence $\{\eta_k\}$ such that $\eta_k > 0$ for all $k \in \mathbb{N}$ and $\lim_{k \to \infty} \eta_k = 0$, and $x_0 \in \mathbb{R}^n$ be given. Set $k \leftarrow 0$.

**Step 1.** If $F(x^k) = 0$, then terminate the execution of the algorithm.

**Step 2.** Choose $\sigma_k$ such that $|\sigma_k| \in [\sigma_{\min}, \sigma_{\max}]$ and $v^k \in \mathbb{R}^n$ such that $\|v^k\| = \|F(x^k)\|$. Compute

$$\bar{f}_k = \max\{f(x^k), \ldots, f(x_{\max\{0,k-M+1\}}^k)\}. \tag{2}$$
**Step 2.1.** Set $\alpha_+ \leftarrow 1$ and $\alpha_- \leftarrow 1$.

**Step 2.2.** Set $d \leftarrow -\sigma_k v^k$ and $\alpha \leftarrow \alpha_+$. Consider

$$f(x^k + ad) \leq \tilde{f}_k + \eta_k - \gamma \alpha^2 f(x^k).$$

(3)

If (3) holds, then define $d^k = d$ and $\alpha_k = \alpha$ and go to **Step 3**

**Step 2.3.** Set $d \leftarrow \sigma_k v^k$ and $\alpha \leftarrow \alpha_-$. If (3) holds, then define $d^k = d$ and $\alpha_k = \alpha$ and go to **Step 3**

**Step 2.4.** Choose $\alpha_{+\text{new}} \in (\tau_{\min \alpha_+}, \tau_{\max \alpha_+}]$ and $\alpha_{-\text{new}} \in (\tau_{\min \alpha_-}, \tau_{\max \alpha_-}]$, set $\alpha_+ \leftarrow \alpha_{+\text{new}}$, $\alpha_- \leftarrow \alpha_{-\text{new}}$, and go to **Step 2.2**.

**Step 3.** Define $x_{\text{trial}}^{k+1} = x^k + \alpha_k d^k$. 

**Step 4.** Define $x_{\text{accel}}^{k+1} = x^k - S_k Y_k^\dagger F(x^k)$, where $k = \max\{0, k - p + 1\}$,

$$s^j = x^{j+1} - x^j \text{ for } j = k, \ldots, k - 1,$$

$$y^j = F(x^{j+1}) - F(x^j) \text{ for } j = k, \ldots, k - 1,$$

$$s^k = x_{\text{trial}}^{k+1} - x^k,$$

$$y^k = F(x_{\text{trial}}^{k+1}) - F(x^k),$$

$$S_k = (s_k, \ldots, s_1, s^k),$$

$$Y_k = (y_k, \ldots, y_1, y^k),$$

and $Y_k^\dagger$ is the Moore-Penrose pseudoinverse of $Y_k$.

**Step 5.** Choose $x^{k+1} \in \{x_{\text{trial}}^{k+1}, x_{\text{accel}}^{k+1}\}$ such that

$$\|F(x^{k+1})\| = \min \{\|F(x_{\text{trial}}^{k+1})\|, \|F(x_{\text{accel}}^{k+1})\|\}.$$ 

**Step 6.** Set $k \leftarrow k + 1$, and go to **Step 1**.

In practice, at **Step 1** given $\varepsilon > 0$, the stopping criterion $\|F(x^k)\| = 0$ is replaced with

$$\|F(x^k)\|_2 \leq \varepsilon.$$ 

(4)

(Criterion $\|F(x^k)\| = 0$ in the algorithm is necessary so we can state theoretical asymptotic properties of an infinite sequence generated by the algorithm.) At **Step 2** the spectral choice for $\sigma_k$ (see [2, 17, 18, 4, 5, 6, 7]) corresponds to

$$\sigma_{k \text{spg}} = \frac{(x^k - x^{k-1})^T (x^k - x^{k-1})}{(x^k - x^{k-1})^T F(x^k) - F(x^{k-1})}.$$ 

Following [13], if $|\sigma_{k \text{spg}}| \in [\sigma_{\min}, \min\{1, \sigma_{\max}\}]$, then we take $\sigma_k = \sigma_{k \text{spg}}$; otherwise, we take $\sigma_k = \max\{\sigma_{\min}, \min\{\|x^k\|_2/\|v^k\|_2, \sigma_{\max}\}\}$.  

Still at **Step 2** the residual choice for the search direction corresponds to $v_k = F(x^k)$. At **Step 2.4**, we compute $\alpha_{\text{new}}$ as the minimizer of the univariate quadratic $q(\alpha)$ that interpolates $q(0) = f(x^k)$, $q(\alpha_+) = f(x^k - \alpha_+ \sigma_k F(x^k))$, and $q'(0) = -\sigma_k F(x^k)^T \nabla f(x^k) = -\sigma_k F(x^k)^T J(x^k) F(x^k)$. Following [12], since we consider $J(x^k)$ unavailable, we consider $J(x^k) = I$. Thus,

$$\alpha_{+\text{new}} = \max \left\{\tau_{\min \alpha_+}, \min \left\{\frac{\alpha^2_+ f(x^k)}{f(x^k - \alpha_+ \sigma_k F(x^k)) + (2\alpha_+ - 1)f(x^k)}, \tau_{\max \alpha_+}\right\}\right\}.$$ 

Analogously,

$$\alpha_{-\text{new}} = \max \left\{\tau_{\min \alpha_-}, \min \left\{\frac{\alpha^2_- f(x^k)}{f(x^k + \alpha_- \sigma_k F(x^k)) + (2\alpha_- - 1)f(x^k)}, \tau_{\max \alpha_-}\right\}\right\}.$$ 

Theoretical results of Algorithm 2.1 are given in [3] §3 and §4. Briefly, limit points of sequences generated by the algorithm are solutions of the nonlinear system or the gradient of the corresponding sum of squares is null. Moreover, under suitable assumptions, the convergence to solutions is superlinear.
3 Usage of the R implementation

We implemented Algorithm 2.1 in R language as a subroutine named `dfsaneacc`. Codes are freely available at [https://github.com/johngardenghi/dfsaneacc](https://github.com/johngardenghi/dfsaneacc) and at the Journal web page accompanying the present work. In this section, we describe how to use `dfsaneacc` to solve a nonlinear system coded in R and how to solve a nonlinear system from the CUTEst collection.

The calling sequence of `dfsaneacc` is given by

```
R> dfsaneacc(x, evalr, nhlim, epsf, maxit, iprint, ...)
```

where

- `x`: is an \( n \)-dimensional array containing the initial guess.
- `evalr`: is the subroutine that computes \( F \) at a point \( x \). This subroutine must have the calling sequence
  ```r
  evalr <- function(x, ...) {}
  ```
  where \( ... \) represents the additional arguments of `dfsaneacc`. The subroutine must return \( F \) evaluated at \( x \).
- `nhlim`: corresponds to \( p+1 \), where \( p \geq 1 \) is the integer that says how many previous iterates must be considered in the Sequential Secant acceleration at Step 4. The “default” value is \( p = 5 \), so \( nhlim=6 \); but having a problem at hand, it is recommendable to try different values.
- `epsf`: corresponds to the stopping tolerance \( \varepsilon \) in (4).
- `maxit`: represents the maximum number of iterations. Its default value is `maxit=+\infty`.
- `iprint`: determines the level of the details in the output of the routine – `iprint=-1` means no output, `iprint=0` means basic information at every iteration, `iprint=1` adds additional information related to the backtracking strategy [Step 2], and `iprint=2` adds information related to the computation of the acceleration step [Step 4]. Its default value is `iprint=-1`.

As an example, consider the Exponential Function 2 from [14, p.596] given by \( F(x) = (F_1(x), \ldots, F_n(x))^T \), where

\[
F_1(x) = e^{x_1} - 1
\]
\[
F_i(x) = \frac{i}{10} (e^{x_1} + x_{i-1} - 1) \quad \text{for } i = 2, \ldots, n,
\]

with the initial guess \( x^0 = (\frac{1}{n^2}, \ldots, \frac{1}{n^2})^T \). The first step is to code it in R as follows:

```r
R> expfun2 <- function(x) {
+   n <- length(x)
+   f <- rep(NA, n)
+   f[1] <- exp(x[1]) - 1.0
+   f[2:n] <- (2:n)/10.0 * (exp(x[2:n]) + x[1:n-1] - 1)
+   f
+ }
```

Then, we set the dimension \( n \) and the initial point \( x^0 \) and call `dfsaneacc` as follows:

```r
R> n <- 3
R> x0 <- rep(1/n^2, n)
R> ret <- dfsaneacc(x=x0, evalr=expfun2, nhlim=6, epsf=1.0e-6*sqrt(n),
+                   iprint=0)
```

obtaining the result below:
Iter: 0 f = 0.02060606
Iter: 1 f = 0.001215612
Iter: 2 f = 4.68925e-05
Iter: 3 f = 4.654419e-08
Iter: 4 f = 1.135198e-11
Iter: 5 f = 9.154603e-16
success!

$x$
[ ,1 ]
[ 1, ] -3.582692e-11
[ 2, ] -7.222425e-08
[ 3, ] -1.638214e-08

$res$
[ 1 ] -3.582690e-11 -1.445201e-08 -2.658192e-08

$normF$
[ 1 ] 9.154603e-16

$iter$
[ 1 ] 5

$fcnt$
[ 1 ] 11

$istop$
[ 1 ] 0

where

$x$: is the approximation to a solution $x^*$.

$res$: corresponds to $F(x^*)$.

$normF$: corresponds to $f(x^*) = \|F(x^*)\|_2^2$.

$iter$: is the number of iterations.

$fcnt$: is the number of calls to evalr, i.e. the number of functional evaluations.

$istop$: is the exit code, where $istop=0$ means that $x^*$ satisfies (4), i.e. $\|F(x^*)\|_2 \leq \varepsilon$, and $istop=1$ means that the maximum allowed number of iterations was reached.

In the rest of this section, we show how to solve a nonlinear system from the CUTEst collection. CUTEst can be downloaded from [https://github.com/ralna/CUTEst](https://github.com/ralna/CUTEst) It is assumed that CUTEst is installed, in particular SifDec, and that there is a folder with all problems in SIF format.

The first step is to choose a problem and run SifDec that, based on the problem’s SIF file, generates a Fortran routine to evaluate, in this case, function $F$. It should be mentioned that problems in the CUTEst collection are general nonlinear optimization problems of the form

Minimize $\Phi(x)$ subject to $h(x) = 0$, $\ell_g \leq g(x) \leq u_g$, $\ell \leq x \leq u$,

(5)

where $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function, $h : \mathbb{R}^n \rightarrow \mathbb{R}^{m_E}$ represents $m_E$ equality constraints, $g : \mathbb{R}^n \rightarrow \mathbb{R}^{m_I}$ represents $m_I$ two-side inequality constraints, $\ell_g, u_g \in \mathbb{R}^{m_I}$, and $\ell, u \in \mathbb{R}^n$ represent bounds on the variables. (Some components of $\ell_g$ and $\ell$ can be $-\infty$ as well as some components of $u_g$ and $u$ can be equal to $+\infty$.) Thus, a nonlinear system of equations corresponds to a problem of the form (5) with constant or null objective function, equality constraints only, and $n = m_E$; and, in the context of the present work, we define $F(x) \equiv h(x)$. Once
the Fortran codes have been generated, a dynamic library must be built and loaded in R. The wrapper (written
in R) uses this library to call, using the .Call tool, a C subroutine from an existent C interface of CUTEst,
that calls the generated Fortran subroutine. In fact, CUTEst is mainly coded in Fortran and calling a Fortran
subroutine using the tool .Fortran would be the natural choice. However, numerical experiments shown that
the combination of .Call with the existent C interface of CUTEst is faster.

The wrapper consists in five routines named cutest_init, cutest_end, cutest_getn, cutest_getx0, and
cutest_evalr. Routine cutest_init receives as parameter the name of a problem and executes all initialization
tasks described in the previous paragraph. Routine cutest_end has no parameters and it cleans the environment
by freeing the memory allocated in the call to cutest_init. The other three routines are self-explanatory. So,
for example, a problem named BOOTH can be solved simply by typing:

```r
R> cutest_init('BOOTH')
R> n <- cutest_getn()
R> x0 <- cutest_getx0()
R> ret <- dfsaneacc(x=x0, evalr=cutest_evalr, nhlim=6, epsf=1.0e-6*sqrt(n),
                +           iprint=0)
R> cutest_end()
```

The output follows:

```
Iter:  0  f = 74
Iter:  1  f = 3.544615
Iter:  2  f = 9.860761e-31
success!
$x$
   [,1]
  [1,]  1
  [2,]  3
$res
  [1] -8.881784e-16 -4.440892e-16
$normF
  [1] 9.860761e-31
$iter
  [1] 2
$fcnt
  [1] 7
$istop
  [1] 0
```

There are environment variables that must be set to indicate where CUTEst was installed, which is the
folder that contains the SIF files of the problems, and which Fortran compiler and compiling options must be
used. A README file with detailed instructions accompanies the distribution of Accelerated DF-SANE and
the CUTEst interface with R.

4 Numerical experiments

In this section, we show the performance of Algorithm 2.1 by putting it in perspective in relation to the DF-
SANE algorithm of the BB package [21] and the well-known NITSOL method [16]. For that, we consider
all 70 nonlinear systems of the CUTEst collection [11] with their default dimensions and their default initial
points.
In this work, we implemented Algorithm 2.1 in R; while a Fortran implementation, available at https://www.ime.usp.br/~egbirgin/sources/accelerated-df-sane/ was given in [3]. The state-of-the-art solver NITSOL is available in Fortran at https://users.wpi.edu/~walker/NITSOL/. A Fortran version of DF-SANE is available under request to the authors of [13]; while an R implementation of DF-SANE is available as part of the BB package [21]. Problems of the CUTEst collection are written in SIF (Standard Input Format); and a tool named SifDec (SIF Decoder) generates Fortran routines to evaluate the objective function, in addition to constraints and their derivatives when desired. So, an interface between R and CUTEst was implemented in order to test DF-SANE and Accelerated DF-SANE (both in R) with the problems of the CUTEst collection. Fortran codes were compiled with the GFortran compiler of GCC (version 9.3.0). R codes were run in version 4.0.2. Tests were conducted on a computer with an Intel Core i7 7500 processor and 12 GB of RAM memory, running Linux (Ubuntu 20.10).

Regarding the DF-SANE method [13] that is available as part of the BB package [21], a few considerations are in order. First of all, in the numerical experiments, we considered function dfsane from package BB version 2019.10-1. In the BB package, there is a routine named BBSolve that is a wrapper for dfsane. BBSolve calls dfsane repeatedly with different algorithm parameters aiming to find a solution to the problem at hand. Since this strategy can be used in connection with any method, aiming a fair comparison, in the present work we report the results obtained with a single run of dfsane with its default parameters. This means that the strategies described in [21] §2.4 are not being considered. On the other hand, dfsane improves the original DF-SANE method introduced in [13] in several ways; see [21] §2.3. Among the improvements, there is one that is particularly relevant in the context of the present work: when the plain DF-SANE method fails by lack of progress, dfsane launches an alternative method – it runs L-BFGS-B for the minimization of $f(x) = \|F(x)\|_2^2$. L-BFGS-B [8] is a limited-memory quasi-Newton method for bound-constrained minimization. In some way, it could be said that this modification aims to mitigate the slow convergence of DF-SANE. In contrast to the approach presented in the present paper, this device is triggered only once slow convergence has been detected; while in the present work, acceleration is done at every iteration. Anyway, it is worth noticing that, by comparing the method being introduced in the present work with dfsane from the BB package, a comparison is being done with an improved version of the original DF-SANE introduced in [13].

From now on, we refer to the DF-SANE of the BB package simply as DF-SANE; while we refer to Algorithm 2.1 as “Accelerated DF-SANE”. NITSOL includes three main iterative solvers for linear systems: GMRES, BiCGSTAB, and TFQMR. Numerical experiments showed that, on the considered set of problems, using GMRES presents the best performance among the three options. So, from now on, we refer to NITSOL as “NITSOL (GMRES)”. All default parameters of DF-SANE and NITSOL (GMRES) were considered. For the Accelerated DF-SANE, following [3], we considered $\gamma = 10^{-4}, \tau_{\min} = 0.1, \tau_{\max} = 0.5, M = 10, \sigma_{\min} = \sqrt{\epsilon}, \sigma_{\max} = 1/\sqrt{\tau}, \eta_k = 2^{-k}\min\{\frac{1}{2}\|F(x^0)\|_2, \sqrt{\|F(x^0)\|}\}\}$, where $\epsilon \approx 10^{-16}$ is the machine precision, and $p = 5$. To promote a fair comparison, in all three methods, the common stopping criterion (4) with $\varepsilon = 10^{-6}\sqrt{\eta}$, was considered. In addition, each method has its own alternative stopping criteria, mainly related to lack of progress; and a CPU time limit of 3 minutes per method/problem was also imposed in the numerical experiments.

Table 1 shows the result of DF-SANE and Accelerated DF-SANE (recall that both methods are coded in R). In the table, the first two columns show the problem name and the number of variables and equations. Then, for each method, the table reports the value of $\|F(x)\|_2$ at the final iterate (column $\|F(x_*)\|_2$), the number of iterations (column #iter), the number of functional evaluations (column #feval), and the CPU time in seconds (column time). In column $\|F(x_*)\|_2$, figures in red are the ones that do not satisfy (4). It is worth noticing that in all cases in which the final iterate of DF-SANE does not satisfy (4), DF-SANE stops by “lack of progress” (flag equal to 5). When the same happens with Accelerated DF-SANE, since no stopping criterion due to lack of progress was implemented, it stops by reaching the CPU time limit. The table shows that Accelerated DF-SANE satisfied the stopping criterion (4) related to success in 44 out of the 70 considered problems; while DF-SANE did the same in 32 problems. Moreover, there were 30 problems that were solved by both methods, 14 problems that were solved by Accelerated DF-SANE only, and 2 problems that were solved by DF-SANE only. These figures show that the acceleration step improves the robustness of DF-SANE.

| Problem | $n$ | Accelerated DF-SANE | DF-SANE |
|---------|----|---------------------|---------|
|         |    | $\|F(x_*)\|_2$ | # iter | # feval | time | $\|F(x_*)\|_2$ | # iter | # feval | time |
| BOOTH   | 2  | 9.9E−16             | 2      | 7       | 0.005065 | 2.4E−07 | 7       | 8       | 0.004709 |
| CLUSTER | 2  | 8.3E−07             | 23     | 108     | 0.007488 | 2.4E−07 | 40      | 42      | 0.005575 |
| Method      | Description          | h   | N   | E    | R    | 180  | Notes       |
|-------------|----------------------|-----|-----|------|------|------|-------------|
| YATP2SQ     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| DENSCHNCFn |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| DENSCHNFN  |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| FREURONE    |                      | 15  | 283 | 1.9  | 165  | 0.005079 |
| GOTTFFR     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HIMMBLA     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HIMMELBC    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HIMMELBD    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HSS         |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HYPCIR      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| POWELLBS    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| POWELLSQ    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| PRICE3N     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| PRICE4N     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| RSNBRNE     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| SINVALNE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| WAYSEA1NE   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| MAHCMNE     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| DENschDNE   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| DENSCHNENE  |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HATFLDF     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HATFLDFNL   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HELIXNE     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| Himmelbe    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| RECIPF      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| RANGW1L3    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| POWERLESE   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| POWERSUMNE  |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HEART6      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HEART8      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| COOLHNS     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| MOREBNE     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| OSCIpanE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| TRIGONIE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| INTEQF      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HATFLDG     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HYDCAR6     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| METHAN8     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| HYDCAR20    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| LUvS2AN12   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| Mancinone   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| QINGNE      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| ARGTRIG     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| BROWNALN    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| CHANDHEU    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| 10FOLDTR    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| KSS         |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| MSQRTA      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| MSQRTB      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| EIGENAU     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| EIGENB      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| EIGENC      |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| NONMSQRTNE |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| BROYDN3D    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| BROYDNBD    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| BRYBNDNE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| NONDIANE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| SBRYBNDNE   |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| TQUARTICNE  |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| OSCJORNE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| CYCLIC3     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| YATP1CNE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| YATP1NE     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| YATP2CNE    |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
| YATP2SQ     |                      | 123 | 283 | 1.9  | 165  | 0.005079 |
Table 1: Detailed results of the application of Accelerated DF-SANE and DF-SANE to the 70 considered problems from the CUTEst collection.

Figure 1 compares the methods’ efficiencies using performance profiles [10]. In a performance profile, for \( i \in M = \{\text{Accelerated DF-SANE, DF-SANE}\} \),

\[
\Gamma_i(\tau) = \frac{\# \{ j \in \{1, \ldots, n_P\} \mid t_{ij} \leq \tau \min_{m \in M} \{t_{mj}\} \}}{n_P},
\]

where \( \#S \) denotes the cardinality of set \( S \), \( n_P = 70 \) is the number of problems being considered, and \( t_{ij} \) is a measure of the performance of method \( i \) when applied to problem \( j \). If method \( i \) was not able to solve problem \( j \), then we set \( t_{ij} = +\infty \). With these definitions, \( \Gamma_i(1) \) is the fraction of problems in which method \( i \) was the fastest method to find a solution; while \( \Gamma_i(\tau) \) for \( \tau \) sufficiently large is the fraction of problems that method \( i \) was able to solve, independently of the required effort. Another possibility, once the robustness of the methods being compared has been established, is to restrict the set of problems in a performance profile to the set of problems that were solved by both methods (\( n_P = 30 \) in this case); so \( t_{ij} < +\infty \) for all \( i \) and \( j \). With these definitions, the performance profile does not reflect the robustness of the methods any more (\( \Gamma_i(\tau) = 1 \) for a sufficiently large \( \tau \) for all \( i \in M \)) and it is focused on the methods’ efficiency. (\( \Gamma_i(1) \) still represents the fraction of problems in which method \( i \) was the fastest method to find a solution.) This was the choice in Figure 1 in which the number of functional evaluations and the CPU time were used as performance measures. Both graphics show the methods have very similar efficiencies. It is worth noticing that CPU times smaller than 0.01 seconds are considered as being 0.01 and that approximately 90% of the CPU times, associated with the problems that both methods solve, are smaller than 0.1 seconds.

![Figure 1: Performance profiles of Accelerated DF-SANE and DF-SANE considering the 30 problems from the CUTEst collection in which both methods found a solution.](image)

In a second experiment, in order to put our method in perspective relatively to a method that represents the state of the art in solving nonlinear systems, we compared Accelerated DF-SANE with NITSOL (GMRES). Since NITSOL (GMRES) is coded in Fortran, we considered the Fortran version of Accelerated DF-SANE in this comparison. Of course, we considered NITSOL (GMRES) without Jacobians. Table 2 and Figure 2 show the results. As in Table 1 in column \( \|F(x_*)\|_2 \), figures in red are the ones that do not satisfy [4]. In all cases the final iterate of NITSOL (GMRES) does not satisfy [4]. NITSOL (GMRES) stops by “too small step in a line search” (flag equal to 6).

Figures in Table 2 show that both Accelerated DF-SANE and NITSOL (GMRES) solve 45 problems. There are 41 problems that were solved by both methods, 4 problems that were solved by Accelerated DF-SANE only, and 4 problems that were solved by NITSOL (GMRES) only. So, both methods appear to be equally robust.

As well as Figure 1, Figure 2 focuses on efficiency and, thus, it considers only the 41 problems in which both, Accelerated DF-SANE and NITSOL (GMRES), found a solution. Figure 2(a) considers number of
functional evaluations as performance metric; while Figure 2(b) considers CPU time. Figure 2(a) shows that NITSOL (GMRES) used less functional evaluations in 63% of the problems; while Accelerated DF-SANE used less functional evaluations in 39% of the problems. (The sum of the percentages is slightly larger than 100% because ties are counted twice.) The fact that the two curves reach 0.9 before τ = 10 means that in 90% of the problems the number of function evaluations is of the same order. The Accelerated DF-SANE curve reaches the value 1 for τ > 1000 due to only 3 problems. In the problems RECIPE, HEART8 and HATFLD, Accelerated DF-SANE consumes approximately 14, 33 and 1790 times more function evaluations than NITSOL (GMRES). On the other hand, the curve of NITSOL (GMRES) reaches the value 1 between τ = 10 and τ = 100 because in the problem WAYSEA1NE NITSOL (GMRES) uses 41 times more function evaluations than Accelerated DF-SANE.

The performance profile of the Figure 2(b) that considers CPU time as performance measure, shows a similar scenario, contaminated by the fact of having a large proportion of small problems. The figure says that NITSOL (GMRES) is the fastest method in 95% of the problems; while Accelerated DF-SANE is the fastest method in 85% of the problems, i.e., there are a lot of ties. (As it can be observed in Table 2 approximately 90% of the CPU times associated with problems that are solved by both methods are smaller than 0.1 seconds; and CPU times smaller than 0.01 seconds are considered ties.) The curve of NITSOL (GMRES) reaches 1 before τ = 2 because in no problem does NITSOL (GMRES) uses more than twice the time of Accelerated DF-SANE. Accelerated DF-SANE also did not use more than twice the time of NITSOL in 37 out of the 41 problems. On the remaining 4 problems, Accelerated DF-SANE uses a little more than twice as much time on CHANDHEU and OSCIGRNE (which is why the curve passes 0.95 before τ = 3) and on problems HEART8 and HATFLD it uses 21 and 23 times as much time.

Summing up, we conclude that, while both methods are equally robust, NITSOL (GMRES) is slightly more efficient that Accelerated DF-SANE in the considered set of problems. On the other hand, it is worth noticing that numerical experiments in [3] showed that Accelerated DF-SANE outperforms NITSOL (GMRES) to a large extent on an important class of large-scale problems coming from the discretization of partial differential equations. Of course, the opposite situation can also occur, which justifies the availability of both methods.

| Problem   | n  | Accelerated DF-SANE | NITSOL (GMRES) |
|-----------|----|----------------------|----------------|
|           |    | ∥F(x*)∥ # iter # feval time                  ∥F(x*)∥ # iter # feval time |
| BOOTH     | 2  | 9.9E-16 2 7 0.000014 4.6E-09 3 8 0.000039 |
| CLUSTER   | 2  | 8.3E-07 23 108 0.000048 1.2E-09 9 25 0.000046 |
| CUBENE    | 2  | 4.0E-13 9 20 0.000022 2.1E-10 38 108 0.000076 |
| DENSCHNCN | 2  | 2.3E-11 10 23 0.000029 6.7E-07 6 15 0.000043 |
| DENSCHNFE | 2  | 2.7E-07 7 23 0.000019 1.6E-13 5 16 0.000044 |
| FREURONE  | 2  | 1.5E-08 16 55 0.000025 7.0E+00 16 112 0.000058 |
| GOTTFIR   | 2  | 1.3E-07 29 67 0.000031 3.6E-09 70 296 0.000133 |
| HIMMELBA  | 2  | 0.0E+00 2 7 0.000013 2.5E-08 3 8 0.000042 |
| HIMMELBC  | 2  | 8.4E-08 5 13 0.000018 1.1E-06 6 14 0.000041 |
| HIMMELBD  | 2  | 2.4E+00 11577102 439522728 180.000000 2.4E+00 48 246 0.000164 |
| HSS       | 2  | 4.4E-08 5 13 0.000018 2.4E-11 11 24 0.000045 |
| HYPFCIR   | 2  | 8.7E-10 6 14 0.000017 5.2E-07 5 13 0.000041 |
| POWELLS   | 2  | 1.4E-06 54229896 1259707609 152.775132 1.9E-06 231 692 0.000206 |
| POWELLSQ  | 2  | 1.4E-00 13690098 34211713 180.000000 1.3E+00 37498 309809 0.044447 |
| PRICE3NE  | 2  | 3.9E-10 7 19 0.000020 4.4E-10 7 20 0.000046 |
| PRICE4NE  | 2  | 1.3E-10 10 27 0.000030 3.0E-09 10 27 0.000048 |
| RSNBRNE   | 2  | 2.2E-16 56 204 0.000054 1.4E-06 55 161 0.000075 |
| SINVALVE  | 2  | 4.9E-15 16 77 0.000040 1.9E-14 6 19 0.000042 |
| WAYSEA1NE | 2  | 1.3E-10 12 36 0.000023 3.4E-08 331 1485 0.000291 |
| WAYSEA2NE | 2  | 8.4E-07 481 2179 0.000041 1.3E-09 766 3751 0.000677 |
| DENSCHNDE | 3  | 2.3E-07 26 62 0.000043 1.5E-06 22 71 0.000065 |
| DENSCHNEN | 3  | 9.6E-11 6 16 0.000032 1.5E-09 7 19 0.000046 |
| HATFLDF   | 3  | 1.4E-08 26 78 0.000049 9.6E-07 71 233 0.000117 |
| HATFLDNE  | 3  | 7.9E-03 11587628 252488903 180.000000 7.8E-03 372 2843 0.000672 |
| HELIXNE   | 3  | 2.8E-09 13 35 0.000045 5.0E-01 0 14 0.000040 |
| HIMMELBE  | 3  | 9.7E-16 9 21 0.000023 7.3E-09 2 9 0.000043 |
| RECIPE    | 3  | 6.2E-07 72 403 0.000116 1.4E-06 10 28 0.000048 |
| ZANGWIL3  | 3  | 1.4E-14 3 11 0.000015 5.2E-07 3 10 0.000045 |
| POWELLSE  | 4  | 7.3E-07 24 70 0.000061 1.5E-06 13 61 0.000064 |
| POWERSUMNE| 4  | 1.2E-02 8695243 130639373 180.000000 1.6E-06 1417 7084 0.004665 |
| HEART6    | 6  | 1.5E-06 124382 1818751 0.561869 2.7E-01 3854 28811 0.013372 |
Table 2: Detailed results of the application of Accelerated DF-SANE (in Fortran) and NITSOL (GMRES) to the 70 considered problems from the CUTEst collection.

Table 2: Detailed results of the application of Accelerated DF-SANE (in Fortran) and NITSOL (GMRES) to the 70 considered problems from the CUTEst collection.

| Problem       | Function evaluations | Time (log2 scale) |
|---------------|----------------------|-------------------|
| HEARTS        | 8                    | 2.8E−06           |
| COOLHANS      | 9                    | 1.5E−06           |
| MOREBVNE      | 10                   | 1.6E−06           |
| OSCIPANE      | 10                   | 1.0E+00           |
| TRIGONINE     | 10                   | 1.9E−06           |
| INTEQNE       | 12                   | 9.2E−07           |
| HATFLDG       | 25                   | 5.0E−06           |
| HYDCAR6       | 29                   | 5.0E−03           |
| METHAN8       | 31                   | 1.2E−04           |
| METHANL8      | 31                   | 4.4E−03           |
| HYDCAR20      | 99                   | 3.9E−02           |
| LUKSAN21      | 100                  | 9.8E−06           |
| MANCINONE     | 100                  | 5.9E−07           |
| QINGNE        | 100                  | 4.8E−06           |
| ARGTRIG       | 200                  | 1.2E−05           |
| BROWNVALE     | 200                  | 1.0E−05           |
| CHANDHEU      | 500                  | 1.4E−05           |
| 10FOLDTR      | 1000                 | 2.2E−07           |
| KSS           | 1000                 | 9.3E−09           |
| MSQRTA        | 1024                 | 4.7E+01           |
| MSQRTB        | 1024                 | 4.6E+01           |
| EIGENAU       | 2550                 | 9.6E+00           |
| EIGENB        | 2550                 | 9.6E+00           |
| EIGENC        | 2652                 | 9.2E+01           |
| NONMSQRNE     | 4900                 | 2.4E+02           |
| BROYDNID      | 5000                 | 5.3E−05           |
| BROYDNBD      | 5000                 | 2.4E+00           |
| BRYBNDNE      | 5000                 | 2.4E+00           |
| NONDIANE      | 5000                 | 1.0E+00           |
| SBRYBNDNE     | 5000                 | 2.5E+02           |
| SROSENBNE     | 5000                 | 2.5E−09           |
| SSBRYSBNDNE   | 5000                 | 1.7E+02           |
| TQUARTICNE    | 5000                 | 8.3E−01           |
| OSCIGRNE      | 100000               | 1.8E−04           |
| CYCIC3        | 100000               | 2.6E+02           |
| YATP1CNE      | 123200               | 2.6E−07           |
| YATP2CNE      | 123200               | 3.1E+04           |
| YATP2SQ       | 123200               | 4.1E+04           |

Figure 2: Performance profiles of Accelerated DF-SANE (in Fortran) and NITSOL (GMRES) considering the 41 problems from the CUTEst collection in which both methods found a solution.
A side note comparing the R and Fortran implementations of Accelerated DF-SANE is in order. Comparing Tables 1 and 2, it can be seen that they deliver slightly different results in a few problems and deliver identical results in 40 problems out of the 44 problems in which none of the versions stops by reaching the CPU time limit. If we consider these 40 problems, in which both versions performed an identical number of iterations and functional evaluations, the Fortran version uses, in average, around 10% of the CPU time required by the R version of the method.

5 Conclusions

In [3], where it was shown that an acceleration scheme based on the Sequential Secant Method could improve the performance of the derivative-free spectral residual method [13], numerical experiments with very large problems coming from the discretization of partial differential equations were presented. In the considered family of problems, Accelerated DF-SANE outperformed DF-SANE and NITSOL (GMRES) by a large extent.

In the present work, an R implementation of the method proposed in [3] was introduced. In addition, numerical experiments considering all nonlinear systems of equations from the well-known CUTEst collection were presented. Default dimensions of the problems were considered; and the collection includes small-, medium-, and large-scale problems. Results shown that the proposed method is much more robust than the DF-SANE method included in the R package BB [21]; while it is as robust and almost as efficient as the state-of-the-art classical NITSOL (GMRES) method (coded in Fortran). Therefore, the proposed method appears as a useful and robust alternative for solving nonlinear systems of equations without derivatives to the users of R language.

As a byproduct, an interface to test derivative-free nonlinear systems solvers developed in R with the widely-used test problems from the CUTEst collection [11] was also provided.

References

[1] J. G. P. Barnes. An algorithm for solving nonlinear equations based on the secant method. *The Computer Journal*, 8(1):66–72, 1965.

[2] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[3] E. G. Birgin and J. M. Martínez. Secant acceleration of sequential residual methods for solving large-scale nonlinear systems of equations. Technical Report arXiv:2012.13251v1, arXiv, 2021.

[4] E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[5] E. G. Birgin, J. M. Martínez, and M. Raydan. Algorithm 813: Spg – software for convex-constrained optimization. *ACM Transactions on Mathematical Software*, 27(3):340–349, 2001.

[6] E. G. Birgin, J. M. Martínez, and M. Raydan. Spectral projected gradient methods. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3652–3659. Springer US, Boston, MA, 2009.

[7] E. G. Birgin, J. M. Martínez, and M. Raydan. Spectral projected gradient methods: Review and perspectives. *Journal of Statistical Software*, 60(3), 2014.

[8] R. H. Byrd, P. Lu, and J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[9] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Lancelot – A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer, Berlin, Heidelberg, 1992.

[10] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
[11] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[12] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*, volume 187 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, Heidelberg, 1981.

[13] W. La Cruz, J. M. Martínez, and M. Raydan. Spectral residual method without gradient information for solving large-scale nonlinear systems of equations. *Mathematics of Computation*, 75:1429–1448, 2006.

[14] W. La Cruz and M. Raydan. Nonmonotone spectral methods for large-scale nonlinear systems. *Optimization Methods and Software*, 18(5):583–599, 2003.

[15] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[16] M. Pernice and H. F. Walker. NITSOL: a newton iterative solver for nonlinear systems. *SIAM Journal on Scientific Computing*, 19(1):302–318, 1998.

[17] M. Raydan. On the barzilai and borwein choice of steplength for the gradient method. *IMA Journal of Numerical Analysis*, 13(3):321–326, 1993.

[18] M. Raydan. The barzilai and borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal on Optimization*, 7(1):26–33, 1997.

[19] K. Schittkowski. *More Test Examples for Nonlinear Programming Codes*, volume 282 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, Heidelberg, 1987.

[20] The R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009.

[21] R. Varadhan and P. Gilbert. BB: An R package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. *Journal of Statistical Software*, 32(4):1–26, 2009.

[22] P. Wolfe. The secant method for simultaneous nonlinear equations. *Communications of ACM*, 2(12):12–13, 1959.