TENSOLVE: A Software Package for Solving Systems of Nonlinear Equations and Nonlinear Least Squares Problems Using Tensor Methods

Ali Bouaricha
Argonne National Laboratory
and
Robert B. Schnabel
University of Colorado at Boulder

This paper describes a modular software package for solving systems of nonlinear equations and nonlinear least squares problems, using a new class of methods called tensor methods. It is intended for small to medium–sized problems, say with up to 100 equations and unknowns, in cases where it is reasonable to calculate the Jacobian matrix or approximate it by finite differences at each iteration. The software allows the user to select between a tensor method and a standard method based upon a linear model. The tensor method approximates $F(x)$ by a quadratic model, where the second-order term is chosen so that the model is hardly more expensive to form, store, or solve than the standard linear model. Moreover, the software provides two different global strategies, a line search and a two-dimensional trust region approach. Test results indicate that, in general, tensor methods are significantly more efficient and robust than standard methods on small and medium–sized problems in iterations and function evaluations.

Categories and Subject Descriptors: G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*systems of equations*; G.1.6 [**Numerical Analysis**]: Optimization—*least squares methods*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms

Additional Key Words and Phrases: tensor methods, nonlinear equations, nonlinear least squares, rank-deficient matrices

---

Name: Ali Bouaricha
Address: Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 60439; email: bouarich@mcs.anl.gov. Research supported in part by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.
Name: Robert B. Schnabel
Address: Department of Computer Science, University of Colorado, Boulder, Colorado 80309-0430; email: bobby@cs.colorado.edu. Research supported by AFOSR Grants No. AFOSR-90-0109 and F49620-94-1-0101, ARO Grants No. DAAL03-91-G-0151 and DAAH04-94-G-0228, and NSF Grant No. CCR-9101795.

1. INTRODUCTION

This paper describes a modular software package for solving systems of nonlinear equations of the form

$$F \; : \; R^n \rightarrow R^m, \quad m \; \geq \; n, \tag{1}$$

where F is assumed to be at least once continuously differentiable, using a new class of methods called *tensor methods.* If m is equal to n, the package solves the nonlinear equations problem, $F(x) = 0$, while if m is greater than n it solves the nonlinear least squares problem, $\min_{x \in R^n} \|F(x)\|_2$

Tensor methods base each iteration on a quadratic model of the nonlinear function,

$$M(x_c \; + \; d) \; = \; F(x_c) \; + \; F'(x_c)d \; + \; \frac{1}{2} \, T_c dd, \tag{2}$$

where x_c is the current iterate, and T_c is a three-dimensional object referred to as a tensor. No second derivative information is used in forming the tensor term T_c. Instead, T_c is formed by asking the model to interpolate up to $\sqrt{n}$ past function values in a way that hardly increases the storage requirements or arithmetic cost per iteration over standard linear model based methods. The package also provides an option to use a method based on the standard linear model ((2) without the tensor term); it then performs a standard Newton method for nonlinear equations or Gauss-Newton method for nonlinear least squares. The global strategy used in either case can be either a line search strategy or a two-dimensional trust region method over the subspace spanned by the steepest descent direction and the tensor (or Newton/Gauss-Newton) step.

Required input to the package includes the dimensions m and n of the problem, where m is the number of nonlinear equations and n is the number of unknowns; a subroutine to evaluate the function $F(x)$; and an estimate x_0 of the solution x_*. The user may provide a code to calculate the Jacobian rather than having it computed by finite differences, may choose the standard method rather than the tensor method, and may specify various tolerances.

Upon completion, the program returns with an approximation x_p to the solution x_*, the value of the sum of squares of the function $F(x_p)$, the value of the gradient $G(x_p) = F'(x_p)^T F(x_p)$, and a flag specifying under which stopping condition the algorithm was terminated.

The tensor methods upon which this software package is based were originally introduced by Schnabel and Frank [11], for nonlinear equations. One main contribution of this paper is the provision and extensive testing of a software package incorporating these methods. In addition, the extension of these methods to nonlinear least squares, and the incorporation of a trust region strategy with tensor methods, are new contributions.

The remainder of the paper is organized as follows. In Section 2 we give a brief overview of tensor methods for nonlinear least squares problems (tensor methods for nonlinear equations can be regarded as a special case of these). In Section 3 we discuss the globally convergent modifications for tensor methods for systems of nonlinear equations and nonlinear least squares problems. Section 4 gives an overview of the key features and options provided by the software package. Finally, in Section 5 we summarize and discuss our experimental results using the package, with both line search and trust region strategies, on nonsingular and singular test problems.

2. BRIEF OVERVIEW OF TENSOR METHODS

Tensor methods are general-purpose methods intended especially for problems where the Jacobian matrix at the solution is singular or ill-conditioned. The idea is to base each iteration upon a model that has more information than the standard linear model but is not appreciably more expensive to form, store, or solve. Specifically, each iteration is based upon a quadratic model (2) of the nonlinear function $F(x)$. The particular choice of the tensor term $T_c \in R^{m \times n \times n}$ causes the second-order term $T_c dd$ in (2) to have a simple and useful form. The tensor term is chosen to allow the model $M(x_c + d)$ to interpolate values of the function $F(x)$ at past iterates x_{-k}; that is, the model should satisfy

$$F(x_{-k}) = F(x_c) + F'(x_c)s_k + \frac{1}{2}T_c s_k s_k, \qquad k = 1, \cdots, p, \tag{3}$$

where

$$s_k = x_{-k} - x_c, \qquad k = 1, \cdots, p.$$

The past points $x_{-1}, \cdots, x_{-p}$ are selected so that the set of directions $\{s_k\}$ from x_c to the selected points is strongly linearly independent; each direction s_k is required to make an angle of at least 45 degrees with the subspace spanned by the previously selected past directions. The procedure of finding linearly independent directions is implemented easily by using a modified Gram-Schmidt algorithm, and usually results in $p = 1$ or 2.

After selecting the linearly independent past directions s_k, the tensor term is chosen by the procedure of Schnabel and Frank [11], which generalizes in a straightforward way to nonlinear least squares. T_c is chosen to be the smallest matrix that satisfies the interpolation conditions (3); that is,

$$\underset{T_c \in R^{m \times n \times n}}{\text{minimize}} \; \|T_c\|_F \tag{4}$$

$$\text{subject to } T_c s_k s_k = 2\,(F(x_{-k}) - F(x_c) - F'(x_c)s_k),$$

where $\|T_c\|_F$, the Frobenius norm of T_c, is defined by

$$\|T_c\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} (T_c[i,j,k])^2.$$

The solution to (4) is the sum of p rank–one tensors whose horizontal faces are symmetric:

$$T_c = \sum_{k=1}^{p} a_k s_k s_k, \tag{5}$$

where a_k is the k−th column of $A \in R^{m \times p}$, A is defined by $A = ZM^{-1}$, Z is an $(m \times p)$ matrix whose columns are $Z_j = 2\,(F(x_{-j}) - F(x_c) - F'(x_c)s_j)$, and M is a $(p \times p)$ matrix defined by $M_{ij} = ({s_i}^T s_j)^2$, $1 \le i,\ j \le p$.

If we use the tensor term (5), the tensor model (2) becomes

$$M(x_c + d) = F(x_c) + F'(x_c)d + \frac{1}{2}\sum_{k=1}^{p} a_k \, \{d^T s_k\}^2. \tag{6}$$

The simple form of the quadratic term in (6) is the key to being able to efficiently form, store, and solve the tensor model. The cost of forming the tensor term in the tensor model is $O(mnp) \leq O(mn^{1.5})$ arithmetic operations, since $p \leq \sqrt{n}$, which is small in comparison with the $O(mn^2)$ cost per iteration of Gauss-Newton methods. The additional storage required is $4p$ m-vectors, which is small in comparison with the storage for the Jacobian.

Once the tensor model (6) is formed, a root of the tensor model is found. It is possible that no root exists; in this case a least squares solution of the model is found instead. Thus, in general, we solve the problem

$$\underset{d \in R^n}{\text{minimize}} \, \| M(x_c + d) \|_2. \tag{7}$$

A generalization of the process in Schnabel and Frank [11] shows that the solution to (7) can be reduced to the solution of a small number of quadratic equations, $m - n + q$ quadratic equations in p unknowns, plus the solution of $n - q$ linear equations in $n - p$ unknowns. Here q is equal to p whenever $F'(x_c)$ is nonsingular and usually when $\text{rank}(F'(x_c)) \geq n - p$; otherwise, q is greater than p. Thus the system of linear equations is square or underdetermined, and the system of quadratic equations is equally determined or overdetermined. The main steps of the algorithm are the following:

(1) An orthogonal transformation of the variable space is used to cause the m equations in n unknowns to be linear in $n-p$ variables $\hat{d}_1 \in R^{n-p}$, and quadratic only in the remaining p variables $\hat{d}_2 \in R^p$.

(2) An orthogonal transformation of the equations is used to eliminate the $n - p$ transformed linear variables from $n - q$ of the equations. The result is a system of $m - n + q$ quadratic equations in the p unknowns $\hat{d}_2$, plus a system of $n - q$ equations in all the variables that is linear in the $n - p$ unknowns $\hat{d}_1$.

(3) A nonlinear unconstrained optimization software package, `UNCMIN` [12], is used to minimize the l_2 norm of the $m-n+q$ quadratic equations in the p unknowns $\hat{d}_2$. (If $p = 1$, this procedure is done analytically instead.)

(4) The system of $n - q$ linear equations that is linear in the remaining $n - p$ unknowns is solved for $\hat{d}_1$.

The arithmetic cost per iteration of the above process is the standard $O(mn^2)$ cost of a QR factorization of an $m \times n$ matrix, plus an additional $O(mnp) \leq O(mn^{1.5})$ operations, plus the cost of using `UNCMIN` in Step 3 of the algorithm. The cost of using `UNCMIN` is expected to be $O(p^4) \leq O(n^2)$ operations, since each iteration requires $O(p^3)$ ($O(p^2q)$ when $q > p$) operations and a small multiple of p iterations generally suffice. Thus, the total cost of the above algorithm is the $O(mn^2)$ cost of the standard method plus at most an additional cost of $O(mn^{1.5})$ arithmetic operations. Note that in the case when $p = 1$ and $q \geq 1$, the one-variable minimization problem is solved very inexpensively in closed form; this turns out to be the most common case in practice.

The Newton or Gauss-Newton step is computed inexpensively (in $O(mnp)$ operations) as a by-product of the tensor step solution. Using the tensor step and the Newton or Gauss-Newton step, a line search or a two-dimensional trust region global strategy determines the next iterate, as described in the next section. The overall algorithm is summarized below.

Algorithm 1. An Iteration of the Tensor Method

Given m, n, x_c, $F(x_c)$.

Step 0 Calculate $F'(x_c)$, and decide whether to stop.
Step 1 Select the past points to use in the tensor model from among the $\sqrt{n}$ most recent points.
Step 2 Calculate the second-order term of the tensor model, T_c, so that the tensor model interpolates $F(x)$ at all the points selected in Step 2.
Step 3 Find the root of the tensor model, or its minimizer (in the l_2 norm) if it has no real root.
Step 4 **if** $m > n$ or the two-dimensional trust region is used **then**
Compute the standard step as a by-product of the tensor model solution.
Select the tensor or standard step using Algorithm 2.
Step 5 Select x_+ using either a line search or a two-dimensional trust region global strategy:

if the line search is used **then**
  if $m > n$ **then**
   perform Algorithm 4, where the search direction is the step selected in Step 4
  else $\{m = n\}$
   perform Algorithm 3
elseif the two-dimensional trust region is used **then**
  Perform Algorithm 5 using the model selected in Step 4

Step 6 Set $x_c \leftarrow x_+$, $F(x_c) \leftarrow F(x_+)$, go to Step 0.

The reader may refer to [1], [2], [6], and [11] for more details on tensor methods for nonlinear equations and nonlinear least squares problems. These papers give preliminary indications that tensor methods are more efficient and more robust computationally than standard methods, and show that tensor methods have a superior rate of convergence to Newton's method on nonlinear equations problems where $\text{rank}\{F'(x_*)\} = n - 1$.

3. GLOBALLY CONVERGENT MODIFICATIONS FOR TENSOR METHODS

This section describes the global strategies in the tensor algorithm given above. As with all algorithms for nonlinear equations and optimization, purely local tensor methods may fail to converge if the initial guess is far away from the solution. To address this problem, two types of modification are used in general, line search methods and trust region methods, and either may be best for a particular problem. For this reason, both of these global methods are included in our software package.

This section first describes the overall framework that is used in both the line search and trust region approaches for tensor methods. This framework involves a choice of whether to use the tensor step or the Newton/Gauss-Newton step as the basis for the global strategy at a given iteration. Next we briefly describe the line search that is used in the line search methods. Finally, we describe a new model/trust region approach for tensor methods that is used in the trust region methods.

3.1 Globally Convergent Framework for Nonlinear Least Squares

Our computational experience has shown that when one is far from the solution, it is important to allow the global step to be based sometimes upon the Newton/Gauss-Newton step rather than the tensor step, and we have constructed heuristics to make this choice. Our experimentation has led to two different sets of heuristics, one that is used in both the line search and trust region methods for nonlinear least squares as well as the trust region method for nonlinear equations, and a second that is used in line search methods for nonlinear equations. They differ primarily in how much they bias the choice toward the tensor step. Both are constructed so that close to the solution, the tensor step is nearly always selected. This section gives these heuristics and the overall global frameworks that are based upon them.

Algorithm 2 gives the global framework that is used for nonlinear least squares and for trust region methods for nonlinear equations. In this framework, the Gauss-Newton step is chosen whenever the tensor step is not a descent direction, when the tensor step is a minimizer of the tensor model and does not provide enough decrease in the tensor model, or when the quadratic system of $m-n+q$ equations in p unknowns cannot be solved by `UNCMIN` [12] within the iteration limit. Otherwise, the tensor step is chosen. In the definitions of d_t and M_T, the Newton step and model are used for nonlinear equations, while the Gauss-Newton step and model are used for nonlinear least squares.

Algorithm 2. Global Framework for Nonlinear Least Squares and for Trust Region Methods for Nonlinear Equations

Let x_c = current iterate,
  $J(x_c)$= approximation to $F'(x_c)$,
  $g = J(x_c)^T F(x_c)$, the gradient of $\frac{1}{2}F(x)^T F(x)$ at x_c,
  d_t= minimizer of the tensor model,
  d_n= Newton or Gauss-Newton step: $-J(x_c)^{-1}F(x_c)$
  or $-(J(x_c)^T J(x_c))^{-1} J(x_c)^T F(x_c)$ if $J(x_c)$ is sufficiently well-conditioned,
    Levenberg-Marquardt step $-(J(x_c)^T J(x_c) + \mu I)^{-1} J(x_c)^T F(x_c)$ otherwise,
    where $\mu = \sqrt{n\epsilon \| J(x_c) \|_1 \| J(x_c) \|_\infty}$, ϵ = machine precision,
  M_T= tensor model,
  M_N= Newton or Gauss-Newton model.
  if (no root or minimizer of the tensor model was found) **or**
    ((minimizer of the tensor model that is not a root was found) **and**
    ($\| M_T(x_c + d_t) \|_2 > \frac{1}{2}(\| F(x_c) \|_2 + \| M_N(x_c + d_n) \|_2$))) **or**
    ($g^T d_t > -10^{-4} \| g \|_2 \| d_t \|_2$))
  then
    $x_+ \leftarrow x_c + \lambda d_n$, $\lambda \in (0, 1]$ selected by line search, **or**

$x_+ \leftarrow x_c + \alpha d_n - \beta g$, α, β selected by trust region algorithm
else
$x_+ \leftarrow x_c + \lambda d_t$, $\lambda \in (0, 1]$ selected by line search, **or**
$x_+ \leftarrow x_c + \alpha d_t - \beta g$, α, β selected by trust region algorithm
endif

Algorithm 3 gives the global framework that is used in line search methods for nonlinear equations. Its main difference from Algorithm 2 is that it always tries the tensor step first, whether or not this step meets the descent or model decrease conditions of Algorithm 2. If $x_c + d_t$ provides enough decrease in $\|F(x)\|$, then it is used as the next iterate. If not, the strategy may tentatively compute global steps in both the Newton and the tensor directions. That is, the global step $x_+^n = x_c + \lambda d_n$ produced by a line search in the Newton direction d_n is calculated. In addition, if d_t is a descent direction, the global step $x_+^t = x_c + \lambda d_t$ produced by a line search in the tensor direction also is calculated. Finally, we select x_+^n or x_+^t depending on whichever has the lower function value. Thus, this strategy may involve one or more extra function evaluations when both line searches are performed.

Algorithm 3. Global Framework for Line Search Methods for Nonlinear Equations

Given x_c, d_t, g as defined in Algorithm 2, and $\alpha = 10^{-4}$.

slope $:= g^T d_t$
$f_c := \frac{1}{2}\|F(x_c)\|_2^2$
$x_+^t := x_c + d_t$
$f_+ := \frac{1}{2}\|F(x_+^t)\|_2^2$
if $f_+ < f_c + \alpha \cdot \min\{slope, 0\}$ **then**
  return $x_+ = x_+^t$
else
  Compute the Newton direction d_n
  Find an acceptable x_+^n in the Newton direction d_n, using Algorithm 4
  comment. Test if the tensor step is sufficiently descent
  if $g^T d_t \geq -10^{-4}||g||_2||d_t||_2$ **then**
    return $x_+ = x_+^n$
  else
    Find an acceptable x_+^t in the tensor direction d_t, using Algorithm 4
    if $||F(x_+^n)|| < ||F(x_+^t)||$ **then**
      return $x_+ = x_+^n$
    else
      return $x_+ = x_+^t$
    endif
  endif
endif

3.2 Global Framework for Line Search Methods for Nonlinear Equations

The line search used in the global frameworks outlined above is a standard quadratic backtracking line search. It starts with $\lambda = 1$ and then, if $x_c + d$ is not acceptable, reduces λ until an acceptable $x_c + \lambda d$ is found, based upon a one-dimensional quadratic model of $F(x)^T F(x)$. Let us define

$$\hat{f}(\lambda) = \frac{1}{2} \| F(x_c + \lambda d) \|_2^2,$$

the one-dimensional restriction of $f(x) = \frac{1}{2} \| F(x) \|_2^2$ to the line through x_c in the direction d. If we need to backtrack, we use the values of $\hat{f}(0)$, $\hat{f}'(0)$, and $\hat{f}(\lambda)$ to model $\hat{f}$ and then take the value of λ that minimizes this model as the next value of λ in Algorithm 4 subject to restrictions on how much λ can decrease at once (see, e.g., [4], pages 126–127 for more details). This results in the following algorithm.

Algorithm 4. Standard Quadratic Backtracking Line Search

Given x_c, d, $g = J(x_c)^T F(x_c)$, and $\alpha = 10^{-4}$.

slope $:= g^T d$
$f_c := \frac{1}{2} \| F(x_c) \|_2^2$
$\lambda := 1.0$
$x_p := x_c + \lambda d$
$f_p := \frac{1}{2} \| F(x_p) \|_2^2$
while $f_p > f_c + \alpha \cdot \lambda \cdot$ slope **do**
    $\lambda_{temp} := -\lambda^2 \cdot$ slope $/(2[f_p - f_c - \lambda \cdot$ slope$])$
    $\lambda := \max\{\lambda_{temp}, \lambda/10\}$
    $x_p := x_c + \lambda d$
    $f_p := \frac{1}{2} \| F(x_p) \|_2^2$
endwhile

3.3 Trust Region Tensor Methods for Nonlinear Equations and Nonlinear Least Squares

Two computational methods—the locally constrained optimal (or "hook") method and the dogleg method—are generally used for approximately solving the trust region problem based on the standard model

$$\begin{aligned} &\underset{d}{\text{minimize}} \ \| F(x_c) + J(x_c) d \|_2^2 \\ &\text{subject to} \ ||d||_2 \leq \delta_c, \end{aligned} \tag{8}$$

where δ_c is the current trust region radius. When δ_c is shorter than the standard step, the locally constrained optimal method [8] finds a μ_c such that $||d(\mu_c)||_2 \approx \delta_c$, where $d(\mu_c) = -(J(x_c)^T J(x_c) + \mu I)^{-1} J(x_c)^T F(x_c)$. Then it takes $x_+ = x_c + d(\mu_c)$. The dogleg method is a modification of the trust region algorithm introduced by Powell [10]. Rather than finding a point $x_+ = x_c + d(\mu_c)$ on the curve $d(\mu_c)$ such that $||x_+ - x_c|| \approx \delta_c$, it approximates this curve by a piecewise linear function in the subspace spanned by the Newton direction and the steepest descent direction $-J(x_c)^T F(x_c)$, and takes x_+ as the point on this piecewise curve for which $||x_+ - x_c|| = \delta_c$. (See, e.g., [4] for more details.)

Unfortunately, these two methods are difficult to extend to the tensor model, because certain key properties do not generalize to this model. Trust region algorithms based on (8) are well defined because there is always a unique point x_+ on the hookstep or dogleg curve such that $||d(\mu_c)|| = \delta_c$. Additionally, the value of $\|F(x_c) + J(x_c)d\|_2^2$ along these curves decreases monotonically from x_c to x_+^n, where $x_+^n = x_c + d_n$, which makes the process reasonable. These properties do not extend to the fourth–order sum of squares of the tensor model, which may not be convex. Furthermore, the analogous curve to $d(\mu_c)$ is more expensive to compute. For these reasons, we consider a different trust region approach for our tensor methods.

The approach is to solve a two-dimensional trust region problem over the subspace spanned by the steepest descent direction and the tensor (or standard) step. The main reasons that led us to adopt this approach are that it is easy to construct and is closely related to dogleg-type algorithms over the same subspace. In addition, the resultant step may be close to the optimal trust region step in practice. Byrd, Schnabel, and Shultz [3] have shown that for unconstrained optimization using a standard quadratic model, the analogous two-dimensional minimization approach produces nearly as much decrease in the quadratic model as the optimal trust region step in almost all cases.

The two-dimensional trust region approach for the tensor model computes an approximate solution to the exact trust region problem

$$\begin{gathered}\underset{d}{\text{minimize}}\, \|F(x_c) + J(x_c)d + \frac{1}{2}\sum_{k=1}^{p} a_k \{d^T s_k\}^2\|_2^2 \\ \text{subject to } ||d||_2 \le \delta_c,\end{gathered} \tag{9}$$

by performing a minimization

$$\begin{gathered}\underset{d}{\text{minimize}}\, \|F(x_c) + J(x_c)d + \frac{1}{2}\sum_{k=1}^{p} a_k \{d^T s_k\}^2\|_2^2 \\ \text{subject to } ||d||_2 \le \delta_c, \quad d \in Range\{d_t, g_s\},\end{gathered} \tag{10}$$

where d_t and g_s are the tensor step and the steepest descent direction, respectively, and δ_c is the trust region radius. This approach always produces a step that reduces the quadratic model by at least as much as a dogleg-type algorithm, which minimizes the model over a piecewise linear curve in the same subspace. When Algorithm 2 chooses the Newton or Gauss-Newton step, we instead solve the variant of (10) where d_t is replaced by d_n and the quadratic term in the model is omitted.

Before we give the complete two-dimensional trust region algorithm for tensor methods, we show how to convert the problem (10) into an unconstrained minimization problem in one variable. This transformation is the key to solving (10) efficiently. First, we form an orthonormal basis for the two-dimensional subspace by performing the projection

$$\hat{g}_s = g_s - d_t \frac{g_s^T d_t}{d_t^T d_t} \tag{11}$$

and normalizing $\hat{g}_s$ and d_t to obtain

$$\tilde{d}_t = \frac{d_t}{||d_t||_2}, \qquad \tilde{g}_s = \frac{\hat{g}_s}{||\hat{g}_s||_2}. \tag{12}$$

Since d is in the subspace spanned by $\tilde{d}_t$ and $\tilde{g}_s$, it can be written as

$$d = \alpha\tilde{d}_t + \beta\tilde{g}_s, \quad \alpha, \beta \in \Re. \tag{13}$$

If we square the l_2 norm of this expression for d and set it to δ_c^2, we obtain the following equation for β as a function of α:

$$\beta \;=\; \sqrt{\delta_c^2 \;-\; \alpha^2}.$$

Substituting this expression for β into (13) and then the resulting d into (10) yields the global minimization problem in the one variable α:

$$\begin{aligned}\text{minimize}_\alpha \, \| \quad & F(x_c) + \alpha J(x_c)\tilde{d}_t + \sqrt{\delta_c^2 - \alpha^2} J(x_c)\tilde{g}_s \\ & + \tfrac{1}{2} \textstyle\sum_{k=1}^{p} a_k (\alpha s_k^T \tilde{d}_t + \sqrt{\delta_c^2 - \alpha^2} s_k^T \tilde{g}_s)^2) \quad \|_2^2,\end{aligned} \tag{14}$$

where $-\delta_c < \alpha < \delta_c$. Problems (14) and (10) are equivalent.

We use the same procedure to convert the problem

$$\begin{aligned}&\underset{d}{\text{minimize}} \, \|F(x_c) \;+\; J(x_c)d\,\|_2^2 \\ &\text{subject to } ||d||_2 \;\le\; \delta_c, \quad d \in Range\{d_n, g\}\end{aligned} \tag{15}$$

to the equivalent global minimization problem in the one variable α:

$$\underset{\alpha}{\text{minimize}} \;\; \|F(x_c) \;+\; \alpha J(x_c)\tilde{d}_n \;+\; \sqrt{\delta_c^2 \;-\; \alpha^2}\; J(x_c)\tilde{g}_s\,\|_2^2, \tag{16}$$

where $-\delta_c < \alpha < \delta_c$.

The two-dimensional trust region method for tensor methods is given in the following algorithm.

Algorithm 5. Two-Dimensional Trust Region for Tensor Methods

Given x_c, d_n, d_t as defined in Algorithm 2.

Let $g_s = -J(x_c)^T F(x_c)$, the steepest descent direction;
δ_c the current trust region radius;
$\tilde{d}_t$ and $\tilde{g}_s$ given by (12);
$\tilde{d}_n$ obtained in an analogous way to $\tilde{d}_t$ by applying transformations (11) and (12) to d_n.

Step 0 Compute the global step
 if *tensor model* selected **then**
  Solve problem (14)
  $d = \alpha_* \tilde{d}_t + \tilde{g}_s \sqrt{\delta_c^2 - \alpha_*^2}$
  where α_* is the global minimizer of (14)
 else {*standard model* selected }
  Solve problem (16)
  $d = \alpha_* \tilde{d}_n + \tilde{g}_s \sqrt{\delta_c^2 - \alpha_*^2}$
  where α_* is the global minimizer of (16)
 endif

Step 1 Check the new iterate and update the trust region radius
 $x_+ = x_c + d$
 if *tensor model* selected **then**

$$ratio = \frac{\frac{1}{2}\|F(x_+)\,\|_2^2 - \frac{1}{2}\|F(x_c)\,\|_2^2}{\frac{1}{2}\|F(x_c) + J(x_c)d + \frac{1}{2}\sum_{k=1}^{p} a_k\{d^T s_k\}^2\,\|_2^2 - \frac{1}{2}\|F(x_c)\,\|_2^2}$$

else

$$ratio = \frac{\frac{1}{2}\|F(x_+)\|_2^2 - \frac{1}{2}\|F(x_c)\|_2^2}{\frac{1}{2}\|F(x_c) + J(x_c)d\|_2^2 - \frac{1}{2}\|F(x_c)\|_2^2}$$

endif
if $ratio \geq 10^{-4}$ **then**
  the global step d is successful
else
  decrease trust region
  go to Step 0
endif

The methods used for adjusting the trust radius during and between steps are given in Algorithm A6.4.5 [9, p. 338]. The initial trust radius can be supplied by the user; if not, it is set to the length of the initial Cauchy step. Our software solves the one-variable global optimization problem by a straightforward partitioning scheme described in [2].

4. OVERVIEW OF THE SOFTWARE PACKAGE

This section summarizes the key features of the software package.

The user has the option to solve systems of nonlinear equations or nonlinear least squares problems. In either case, the required input for the software is the number of equations `M`, the number of variables `N`, the function `FVEC` that computes $F(x)$, and an initial guess $\mathtt{X}_0$. If `M` = `N`, the problem is nonlinear equations; if `M` > `N` it is nonlinear least squares. The user does not have to set a flag differentiating between the two problems.

Two methods of calling the package are provided. In the short version, the user supplies only the above information, and default values of all other options and parameters are used. (These include the use of the tensor rather than the standard method, the use of the line search global strategy, and the calculation of the Jacobian by finite differences). In the other method for calling the package, the user may override any default values of the package options and parameters.

The package allows the user to use the tensor method or the standard Newton or Gauss-Newton method. `METHOD` = `1` specifies the tensor method and is the default value. If `METHOD` is set to 0, the package will use the standard method.

Two global strategies are implemented in the software package: a line search method, and a two-dimensional trust region method over the subspace spanned by the steepest descent direction and the tensor (or Newton/Gauss-Newton) step. The global strategy may be specified using the parameter `GLOBAL`. `GLOBAL` = `0` is the default and specifies the line search. `GLOBAL` = `1` specifies the trust region.

The user may supply an analytic routine to evaluate the Jacobian matrix. If it is not supplied, the package computes the Jacobian by finite differences. The finite difference routine is described in detail by Dennis and Schnabel [4]. The parameter `JACFLG` specifies whether an analytic Jacobian has been provided. The default value, which specifies finite differences, is `JACFLG` = `0`. When the analytic Jacobian is supplied, the user has the option of checking the supplied analytic routine against the package's finite difference routine; if `MSG` is set to 2 modulo 4, the package will not check the analytic Jacobian against the finite difference one; otherwise it will.

Scaling information for the variables and/or the functions may be supplied by the user. The software package is coded so that if the user inputs the typical magnitude $typx_i$ of each component of x and/or the typical magnitude $typf_i$ of each component of the function F, the performance of the package is equivalent to what would result from redefining the variables and functions to be $\bar{x} = D_x x$ and $\bar{F} = D_F F$, where $D_x = \mathrm{diag}(1/typx_j)$ and $D_F = \mathrm{diag}(1/typf_i)$, and running the package without scaling. The default value of each $typx_i$ and $typf_i$ is 1 (i.e., no scaling). Scaling is often important for problems in which there is great variation in the magnitudes of individual variables and/or function components.

The package includes a module TSCHKI that examines the input parameters for illegal entries and consistency. Certain illegal or inconsistent entries are reset to default values by this module, while other illegal entries cause the package to terminate.

The standard (default) output from this package consists of printing the input parameters, the final results, and the stopping condition. The printed input parameters are those used by the algorithm and hence include any corrections made by the module TSCHKI. The program will provide an error message if it terminates as a result of input errors. The printed results include a message indicating the reason for termination, an approximation x_p to the solution x_*, the value of the sum of squares of the function $F(x_p)$, and the gradient vector $G(x_p) = F'(x_p)^T F(x_p)$ of the function $\frac{1}{2}\|F(x)\|_2^2$ at x_p. The package provides an additional means for controlling the output by means of the variable MSG. The user may suppress all output or may print the intermediate iterations results in addition to the standard output.

A general flowchart of the TENSOLVE package is shown in Figure 1.

5. TEST RESULTS

We have tested the TENSOLVE software package using the algorithms described above on a variety of nonsingular and singular problems. This section summarizes and discusses the test results.

In our tests, the package terminates successfully if the relative size of $(x_+ - x_c)$ is less than $\epsilon^{\frac{1}{2}}$, or $\| F(x_+) \|_\infty$ is less than $\epsilon^{\frac{2}{3}}$. It terminates unsuccessfully if the iteration limit of 150 is exceeded. If the last global step fails to locate a point lower than x_c in the line search or trust region global strategies, or the relative size of $J(x_+)^T F(x_+)$ is less than $\epsilon^{\frac{1}{3}}$, the method stops and reports this condition; this may indicate either success or failure. All our computations were performed on a Sun SPARCstation 2 computer in the Computer Science Department at the University of Colorado at Boulder, using double–precision arithmetic.

First we tested the software package on the set of nonlinear equations and nonlinear least squares problems in Moré, Garbow, and Hillstrom [9]. These problems all have nonsingular Jacobians at the solution with the exception of Powell's singular function. Then we created singular test problems as proposed in Schnabel and Frank [11] by modifying the nonsingular test problems of Moré, Garbow, and Hillstrom to the form

$$\hat{F}(x) = F(x) - F'(x_*)A(A^T A)^{-1} A^T (x - x_*), \tag{17}$$

where $F(x)$ is the standard nonsingular test function, x_* is its root or minimizer, and $A \in R^{m \times k}$ has full column rank with $1 \le k \le n$. Note that x_* is a root or

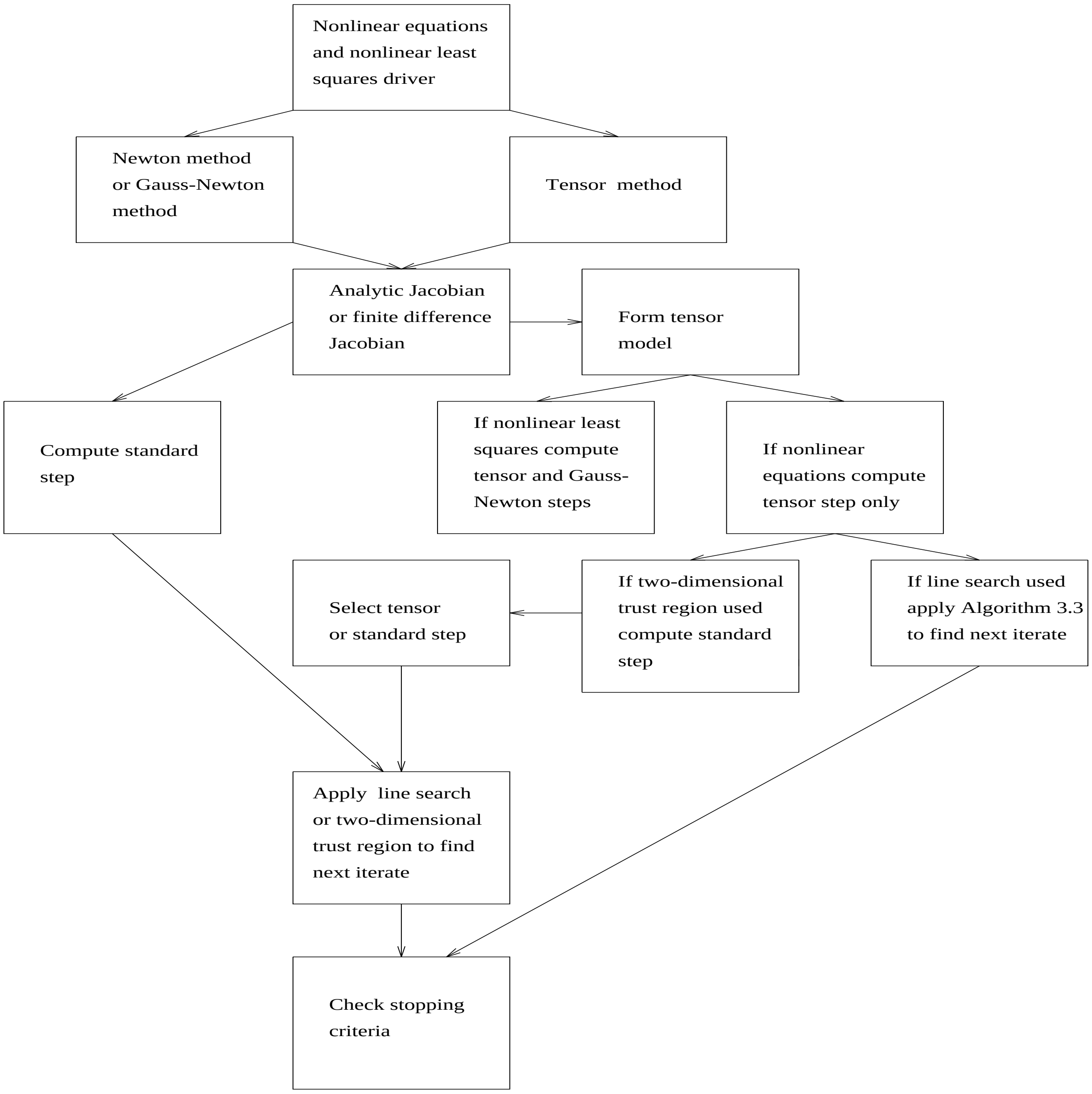

Fig. 1. Structure of the TENSOLVE package

Table 1. Summary for Nonlinear Equations Test Problems using Line Search

Rank $F'(x_*)$	Tensor			Average Ratio Tensor/Newton		Only Newton Solved	Only Tensor Solved
	Better	Worse	Tie	Itn	Feval		
n	25	2	13	0.60	0.69	1	5
$n-1$	24	0	8	0.48	0.53	0	5
$n-2$	27	1	5	0.46	0.56	0	8

critical point of the modified problem, and rank $\hat{F}'(x_*) = n - rank(A)$. We used (17) to create two sets of singular problems, with $\hat{F}'(x)$ having rank $n-1$ and $n-2$, respectively, by using

$$A \in R^{m\times 1}, \qquad A^T = (1, 1, \cdots, 1),$$

and

$$A \in R^{m\times 2}, \qquad A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & -1 & 1 & -1 & \cdots & \pm 1 \end{bmatrix}, \tag{18}$$

respectively.

We tested our tensor algorithm on 17 test functions for systems of nonlinear equations (also including 4 functions from [7] whose Jacobian at the solution x_* is singular and are designated as Griewank functions) and 11 test functions for nonlinear least squares. Some of the test problems were run at various dimensions. All of these problems were also run with the standard method. The list of test problems is given in Appendix A; the detailed test results are given in [2].

Our computational results for the test problems whose Jacobians at the solution have ranks n, $n-1$, and $n-2$ are summarized in Tables 1 to 4. In each of these tables, columns "Better" and "Worse" represent the number of times the tensor method was better and worse, respectively, than the standard method by more than one iteration. The "Tie" column represents the number of times the tensor and standard methods required within one iteration of each other. For each set of problems, we summarize the comparative costs of the tensor and standard methods using average ratios of two measures: iterations, and function evaluations. The average iteration ratio is the total number of iterations required by the tensor method over all the problems included, divided by the total number of iterations required by the standard method on the same problems. The same measure is used for the average function evaluation ratio. These average ratios include only problems that were successfully solved by both methods. We have excluded from the summary of statistics all cases where the tensor and standard methods converge to a different root, or to the same root as each other but not the singular root x_* in the case of singular problems. However, the statistics for the "Better," "Worse," and "Tie" columns include the cases where only one of the two methods converges, and exclude the cases where both methods do not converge. The total number of problems that were solved by one method but not the other are given in the last two columns of each table.

In the test results obtained for both nonsingular and singular nonlinear equations problems, the tensor method is virtually never less efficient than the standard method and usually is more efficient. The improvement by the tensor method over

Table 2. Summary for Nonlinear Equations Test Problems Using Two-Dimensional Trust Region

Rank $F'(x_*)$	Tensor			Average Ratio Tensor/Newton		Only Newton Solved	Only Tensor Solved
	Better	Worse	Tie	Itn	Feval		
n	26	3	13	0.61	0.72	1	6
$n-1$	24	1	9	0.49	0.63	0	4
$n-2$	26	1	5	0.64	0.73	0	4

Table 3. Summary for Nonlinear Least Squares Test Problems Using Line Search

Rank $F'(x_*)$	Tensor			Average Ratio Tensor/Gauss-Newton		Only Gauss-Newton Solved	Only Tensor Solved
	Better	Worse	Tie	Itn	Feval		
n	20	1	8	0.52	0.51	0	4
$n-1$	18	0	8	0.45	0.41	0	2
$n-2$	28	0	5	0.48	0.48	0	4

the standard method with the same global strategy is substantial, averaging about 49% in iterations and 41% in function evaluations when the line search is used, and about 42% in iterations and 31% in function evaluations when the trust region is used, on the problems that are successfully solved by both methods. The improvement by the tensor method over the standard method is more dramatic on problems with small rank deficiency than on nonsingular problems, but is substantial in all cases. On rank $n-1$ problems, this is due in part to the tensor methods achieving 3 step Q-order $\frac{3}{2}$ convergence, whereas Newton's method is linearly convergent with constant $\frac{1}{2}$ [6].

The tensor method is also significantly more robust than the standard Newton-based method for the nonlinear equations test set. Over all the nonlinear equations test problems, 5 rank n problems, 5 rank $n-1$ problems, and 8 rank $n-2$ problems were solved by the tensor method and not by the standard method when the line search was used, and 6 rank n problems, 4 rank $n-1$ problems, and 4 rank $n-2$ problems were solved by the tensor method and not by the standard method when the trust region was used. On the other hand, only 1 rank n problem was solved by the standard method and not by the tensor method when the line search was used, and similarly when the trust region was used.

For the entire set of nonsingular and singular nonlinear least squares problems,

Table 4. Summary for Nonlinear Least Squares Test Problems Using Two-Dimensional Trust Region

Rank $F'(x_*)$	Tensor			Average Ratio Tensor/Gauss-Newton		Only Gauss-Newton Solved	Only Tensor Solved
	Better	Worse	Tie	Itn	Feval		
n	26	1	5	0.66	0.76	0	3
$n-1$	19	2	5	0.66	0.71	0	1
$n-2$	28	1	4	0.63	0.69	0	1

Table 5. Average Ratios of the Tensor Method versus the Gauss-Newton Method on Zero Residual Problems for Line Search and Trust Region

Rank	Line Search		Trust Region	
$F'(x_*)$	Itn	Feval	Itn	Feval
n	0.43	0.44	0.43	0.56
$n-1$	0.41	0.37	0.64	0.62
$n-2$	0.48	0.48	0.51	0.57

Table 6. Average Ratios of the Tensor Method versus the Gauss-Newton Method on Nonzero Residual Problems for Line Search and Trust Region

Rank	Line Search		Trust Region	
$F'(x_*)$	Itn	Feval	Itn	Feval
n	0.64	0.64	0.78	0.88
$n-1$	0.48	0.45	0.67	0.79
$n-2$	0.49	0.48	0.68	0.76

the average improvement of the tensor method over the standard Gauss-Newton method also is substantial. Over the problems solved successfully by both methods, the improvement averages about 52% in iterations and 53% in function evaluations when the line search is used, and about 35% in iterations and 28% in function evaluations when the trust region is used.

The tensor method is also considerably more robust than the Gauss-Newton method for the nonlinear least squares test set, especially in the line search comparison. The tensor method solves several problems that the standard Gauss-Newton method does not, and the reverse never occurs. Over all the nonlinear least squares test problems, 4 rank n problems, 2 rank $n-1$ problems, and 4 rank $n-2$ problems were solved by the tensor method and not by the standard Gauss-Newton method when the line search was used, and 3 rank n problems, 1 rank $n-1$ problems, and 1 rank $n-2$ problems were solved by the tensor method and not by the standard Gauss-Newton method when the trust region was used. On the other hand, there were no problems solved by the standard Gauss-Newton method and not by the tensor method when either the line search or the trust region was used.

A closer examination of the nonlinear least squares test results shows that the improvements by the tensor method are considerably larger for zero residual problems than for nonzero residual problems. The difference is most dramatic in the nonsingular case. Tables 5 and 6 show the average iteration and function evaluation ratios of the tensor method versus the Gauss-Newton method for zero and nonzero residual problems, respectively. The performance differences may be attributable to the fact that both the standard and tensor methods are linearly convergent on nonzero residual problems, but are more quickly convergent on zero residual problems.

The comparison between the line search methods and the trust region methods is very interesting, for both the standard and tensor methods. This is summarized in Tables 7 and 8. These tables show that on the average, the two-dimensional trust region approach is often more efficient than the line search method, especially on nonsingular problems. It is important to note, however, that the line search

Table 7. Average Ratios of Iterations and Function Evaluations of Newton with Trust Region versus Newton with Line Search and Tensor with Trust Region versus Tensor with Line Search for Nonlinear Equations

Rank	Newton TR/LS		Tensor TR/LS	
$F'(x_*)$	Itn	Feval	Itn	Feval
n	0.80	0.84	0.70	0.57
$n-1$	0.78	0.89	0.96	0.93
$n-2$	0.86	0.93	0.92	0.94

Table 8. Average Ratios of Iterations and Function Evaluations of Gauss-Newton with Trust Region versus Gauss-Newton with Line Search and Tensor with Trust Region versus Tensor with Line Search for Nonlinear Least Squares Problems

Rank	Gauss-Newton TR/LS		Tensor TR/LS	
$F'(x_*)$	Itn	Feval	Itn	Feval
n	0.70	0.65	0.75	0.76
$n-1$	0.72	0.71	1.05	1.09
$n-2$	1.01	0.97	0.74	0.80

method is simpler to implement and to understand than the the two-dimensional trust region approach, and is appreciably faster in terms of CPU time on small, inexpensive problems where the complexity of the code becomes the dominant cost. It should also be noted that there is considerable variation in the comparative efficiency of the line search and trust region methods on individual problems and that either may be more efficient for a particular problem class.

Perhaps a more important consideration in the general comparison of the line search and trust region methods, however, is that the two-dimensional trust region method solves considerably more of the test problems than the line search method. The advantage in robustness is particularly large in comparing line search and trust region versions of the standard methods; it is smaller but still significant in comparing tensor methods for nonlinear least squares, and insignificant in our tests of tensor methods for nonlinear equations. Over all the nonlinear equations problems, 20 problems were solved by the trust region and not by the line search, whereas only 5 problems were solved by the line search and not by the trust region. Over all the nonlinear least squares problems, 27 problems were solved by the trust region and not by the line search, whereas only 6 problems were solved by the line search and not by the trust region. The above statistics include the test results for both the tensor and standard methods. Thus, the trust region version seems to have a considerable advantage over the line search version in its robustness, although more when using the standard method than the tensor method. We note that the smaller average improvement of the tensor method over the standard method in the trust region cases (Tables 2 and 4) than the line search cases (Tables 1 and 3) is related to the difference in problem sets that are included in these statistics, because of the differing robustness of the line search and trust region methods.

Finally, we compared our tensor method with the `NL2SOL` package [5] on the set of nonlinear least squares problems used in [5] that is listed in Appendix B. The rea-

Table 9. Comparison of Tensor Method with NL2SOL on the Nonlinear Equations and Nonlinear Least Squares Problems Listed in Table B-1

Global strategy	Tensor versus NL2SOL			Average Ratio–Tensor/NL2SOL	
	Better	Worse	Tie	Itn	Feval
Tensor w/ LS	25	8	2	0.42	0.71
Tensor w/ TR	24	9	2	0.76	0.93

son we were interested in making this comparison is that theoretically the NL2SOL method is superlinearly convergent on nonzero residual problems ([5]), whereas the tensor method of this paper, like Gauss-Newton methods, is only linearly convergent on nonzero residual problems. (This difference is related to NL2SOL using a quadratic model of $F(x)^T F(x)$ whereas the tensor and Gauss-Newton methods use models of $F(x)$.) The problems include a mixture of zero, small, and large residual problems.

Table 9 reports the comparative test results of the tensor method versus NL2SOL on this test set. The first row of Table 9 compares the tensor method using a line search with NL2SOL, whereas the second row compares the tensor method using a two-dimensional trust region with NL2SOL. (NL2SOL uses a trust region global strategy.) The table shows that on these test problems, the tensor method on the average is somewhat more efficient than NL2SOL, with an average improvement of about 58% in iterations and 29% in function evaluations when the line search is used, and about 24% in iterations and 7% in function evaluations when the trust region is used. (Note that the tensor method with line search is more efficient than the tensor method with trust region on this test set.) There is no difference in the robustness of the two packages of this test set; only 1 problem in the test set was solved by NL2SOL and not by the tensor method using either a line search or a trust region method, and only 1 problem was solved by the tensor method and not by NL2SOL. These limited results indicate that the tensor method appears to be quite competitive with NL2SOL for solving least squares problems.

ACKNOWLEDGMENTS

We would like to thank the two referees for criticism that helped us improve the paper.

REFERENCES

[1] A. Bouaricha, *A Software Package for Solving Systems of Nonlinear Equations and Nonlinear Least Squares Problems Using Tensor Methods*, M.S. thesis, Department of Computer Science, University of Colorado at Boulder, 1986.

[2] A. Bouaricha, *Solving Large Sparse Systems of Nonlinear Equations and Nonlinear Least Squares Problems Using Tensor Methods on Sequential and Parallel Computers*, Ph.D. thesis, Department of Computer Science, University of Colorado at Boulder, 1992.

[3] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, *Approximation Solution of the Trust Region Problem by Minimization over Two-Dimensional Subspaces*, Mathematical Programming, 40 (1988), 247–263.

[4] J. E. Dennis and R. B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, N.J., 1983.

[5] J. E. Dennis, D. M. Gay, and R. E. Welsch, *An Adaptive Nonlinear Least Squares Algorithm*, ACM Trans. Math. Softw., 7 (1981), 348–368.

[6] D. Feng, P. Frank, R. B. Schnabel, *An Analysis of Tensor Methods for Nonlinear Equations*, Technical Report CS-CS-729-94, Department of Computer Science, University of Colorado at Boulder, 1992.

[7] A. O. Griewank, *Analysis and Modification of Newton's Method at Singularities*, Ph.D. thesis, Australian National University, Canberra, 1980.

[8] J. J. Moré, *The Levenberg-Marquardt Algorithm: Implementation and Theory*, in *Numerical Analysis*, G. A. Watson, ed., Lecture Notes in Mathematics, vol. 630, Springer–Verlag, Berlin, 1977, 105–116.

[9] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing Unconstrained Optimization Software*, ACM Trans. Math. Softw., 7 (1981), 17–41.

[10] M. J. D. Powell, *A New Algorithm for Unconstrained Optimization*, in *Nonlinear Programming*, J. B. Rosen, O.L. Mangasarian, and K. Ritter, eds., Academic Press, New York, 1970, 33–65.

[11] R. B. Schnabel and P. D. Frank, *Tensor Methods for Nonlinear Equations*, SIAM. J. Num. Anal., 21 (1984), 815–843.

[12] R. B. Schnabel, J. E. Koontz, and B. E. Weiss, *A Modular System of Algorithms of Unconstrained Minimization*, ACM Trans. Math. Softw., 11 (1985), 419–440.

APPENDIX

Appendix A

Table A-1. List of Nonlinear Equations and Nonlinear Least Squares Test Problems Used in the Comparison of Tensor Method versus Standard Method (NS and OS are the dimension of null space and the order of singularity for Griewank's singular functions, respectively)

Problem	Dimension	
	m	n
Brown almost linear	10	10
Broyden banded	30	30
Broyden tridiagonal	30	30
Chebyquad	7	7
Discrete boundary	30	30
Discrete integral	10	10
Helical valley	3	3
Powell singular	4	4
Rosenbrock	2	2
Trigonometric	30	30
Variable dimension	10	10
Watson	31	31
Wood gradient	4	4
NS = 1 OS = 1	3	3
NS = 2 OS = 1	3	3
NS = 1 OS = 2	3	3
NS = 2 OS = 2	3	3
Wood	6	4
Variable dimension	12	10
Bard	15	3
Beale	3	2
Kowalik	11	4
Penalty1	11	10
Penalty2	10	5
Brown badly scaled	3	2
Gauss function	15	3
Brown and Dennis	10	4
Chebyquad	8	4
Chebyquad	12	4
Chebyquad	16	4

Appendix B

Table B-1. List of Nonlinear Equations and Nonlinear Least Squares Test Problems Used in the Comparison of Tensor Method versus NL2SOL

Problem	Dimension	
	m	n
Rosenbrock	2	2
Helical Valley	3	3
Powell Singular	4	4
Wood	6	4
Beale	3	2
Box three-dimensional	10	3
Freudenstein and Roth	2	2
Watson	31	6
Watson	31	9
Watson	31	12
Watson	31	20
Chebyquad	8	8
Bard	15	3
Jennrich and Sampson	10	2
Kowalik	11	4
Osborne 1	33	5
Osborne 2	65	11