Tensor Methods for Large, Sparse Systems of Nonlinear Equations

Ali Bouaricha
Robert Schnabel

CRPC-TR94587
November 1994

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Also available as *MCS-P473-1094* from Argonne National Laboratory, Mathematics and Computer Science Division.

Tensor Methods for Large Sparse Systems of Nonlinear Equations

Ali Bouaricha*
MCS Division, Argonne National Laboratory, Argonne, IL 60439, USA
and
Robert B. Schnabel†
Department of Computer Science, University of Colorado, Boulder, CO 80309-0430, USA

Abstract. This paper introduces tensor methods for solving large sparse systems of nonlinear equations. Tensor methods for nonlinear equations were developed in the context of solving small to medium-sized dense problems. They base each iteration on a quadratic model of the nonlinear equations, where the second-order term is selected so that the model requires no more derivative or function information per iteration than standard linear model-based methods, and hardly more storage or arithmetic operations per iteration. Computational experiments on small to medium-sized problems have shown tensor methods to be considerably more efficient than standard Newton-based methods, with a particularly large advantage on singular problems. This paper considers the extension of this approach to solve large sparse problems. The key issue considered is how to make efficient use of sparsity in forming and solving the tensor model problem at each iteration. Accomplishing this turns out to require an entirely new way of solving the tensor model that successfully exploits the sparsity of the Jacobian, whether the Jacobian is nonsingular or singular. We develop such an approach and, based upon it, an efficient tensor method for solving large sparse systems of nonlinear equations. Test results indicate that this tensor method is significantly more efficient and robust than an efficient sparse Newton-based method, in terms of iterations, function evaluations, and execution time.

Key words. tensor methods, nonlinear equations, sparse problems, rank-deficient matrices.

*Work supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the National Aerospace Agency under Purchase Order L25935D, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

†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

In this paper we introduce tensor methods for solving the sparse nonlinear equations problem

$$\text{given} \quad F : \Re^n \to \Re^n, \quad \text{find } x_* \in \Re^n \quad \text{such that } F(x_*) = 0, \tag{1.1}$$

where it is assumed that n is large (say, $n > 100$), $F(x)$ is a least once continuously differentiable, and the Jacobian matrix $F'(x) \in R^{n \times n}$ is sparse. Large sparse systems of nonlinear equations arise frequently in many practical applications including various network-flow problems and equations produced by finite-difference or finite-element discretizations of boundary values problems for ordinary and partial differential equations. In many situations, $F'(x_*)$ is ill conditioned or singular with a small rank deficiency. For example, this occurs in curve-tracking applications at or near bifurcation points. In such cases, tensor methods are especially intended to improve upon the efficiency of standard algorithms based on Newton's method. Tensor methods are also intended to be at least as efficient as standard methods on problems where $F'(x_*)$ is nonsingular, and in practice they often seem to be considerably more efficient on these problems as well.

Tensor methods for small to medium-sized dense systems of nonlinear equations were introduced by Schnabel and Frank [20], and a software package implementing them is described in [4]. The methods base each iteration on a quadratic model of $F(x)$ that has the form

$$M(x_c + d) = F(x_c) + F'(x_c)d + \frac{1}{2}T_c dd, \tag{1.2}$$

where x_c is the current iterate, and $T_c \in R^{n \times n \times n}$ is the tensor term at x_c. The tensor term is selected so that the model interpolates a very small number, p, of function values from previous iterations. This results in T_c being a rank p tensor, which is crucial to the efficiency of the tensor method. After the model (1.2) is formed, the problem

$$\text{find} \quad d \in R^n \quad \text{that minimizes} \quad ||M(x_c + d)||_2 \tag{1.3}$$

is solved; that is, at each iteration of tensor methods, a minimizer of the model is used if no root exists. Methods for forming the tensor term and solving the tensor model for dense systems of nonlinear equations are reviewed in more detail in the next section. The tensor method requires no more derivative or function information per iteration than Newton's method, and its storage requirement and arithmetic cost per iteration are not appreciably more than for Newton's method.

Methods based on (1.2) have been shown to have very good theoretical properties and very good computational performance on small to medium-sized dense problems. Theoretically, the methods converge at least as quickly as Newton's method on nonsingular problems and have been shown to have 3-step Q-order 1.5 convergence on problems where the Jacobian has rank $n-1$ at the solution, whereas Newton's method is linearly convergent with constant 1/2 on such problems [12]. In tests reported in [4] for both nonsingular and singular problems, the tensor method virtually never is less efficient than a standard method based upon a linear (Newton) model, and usually is more efficient. The improvement by the tensor method over the standard method is substantial, averaging about 49% in iterations and 41% in function evaluations when

a line search is used in each, and about 42% in iterations and 31% in function evaluations when the trust region is used in each, on problems solved successfully by both methods. Furthermore, the tensor method solves a considerable number of problems that the standard method does not, and the reverse virtually never is the case.

The preliminary success of tensor methods for small to medium-sized nonlinear equations makes it reasonable to consider their application to large sparse systems of nonlinear equations. In doing so, there are several key considerations. First, tensor methods require that the Jacobian matrix be available, either analytically or by finite differences, at each iteration. While this is not always the case for small problems—quasi-Newton approximations to the Jacobian sometimes being used instead—it is almost always the case in methods that are used for solving large sparse systems of nonlinear equations. The derivatives usually come from efficient sparse finite differences (see, e.g., [8]), from user-supplied analytic derivatives, or recently through automatic differentiation (see, e.g., [14, 15]). Hence, this requirement is not a problem and indeed fits this approach well. Second, the methods for forming and solving the tensor model must make efficient use of the sparsity of the Jacobian matrix and not involve any dense linear algebra using $n \times n$ matrices. The existing method for forming the tensor model adapts immediately to sparsity, as is shown in Section 2. However, the most difficult and expensive part of the tensor method is solving the quadratic model (1.2) efficiently, and the algorithms used for this so far are entirely inappropriate for large sparse problems. These algorithms make crucial use of orthogonal transformations of both the variable and function space, especially to deal efficiently and stably with cases when the Jacobian matrix is singular or the tensor model has no root. They are not applicable to sparse problems because the orthogonal transformation of the variable space would destroy the sparsity of the Jacobian.

To deal efficiently with sparsity, we develop an entirely new way of solving the tensor model. This approach is able to utilize a sparse variant of Gaussian elimination or any other sparse direct solver. It includes techniques that allow the tensor model to be solved efficiently and stably when the Jacobian matrix is singular. It also entails ways to efficiently calculate the Newton step, which is sometimes used in the tensor algorithm, as a by-product of the calculation of the tensor step.

Using these ingredients, we formulate an efficient tensor method for large sparse nonlinear equations and apply this method to a number of test problems. We compare it with an efficient Newton-based method for solving sparse nonlinear equations that is based upon the same sparse linear equations software and global strategy. Our experimental results indicate that the tensor method is significantly more robust and efficient than the standard method, in terms of iterations, function evaluations, and execution time.

The remainder of this paper is organized as follows. In Section 2 we briefly review tensor methods for dense nonlinear equations, and point out the issues involved in extending them to large sparse problems. In Section 3 we first describe an efficient algorithm for solving the tensor model when the Jacobian matrix is sparse and nonsingular. Next, we present an efficient algorithm for solving the tensor model when the Jacobian is sparse and rank deficient. In Section 4 we show how to efficiently solve the standard linear model in conjunction with these algorithms for solving the tensor model, both when the Jacobian matrix is nonsingular and when it is rank deficient. Section 5 gives a high-level description of the complete tensor method for sparse nonlinear equations, including the global strategy. In Section 6 we compare results

for this implementation with those for the same implementation based on Newton's method. Finally, Section 7 gives a brief summary and discussion of future work.

2. Brief Overview of Tensor Methods for Dense Nonlinear Equations

Tensor methods are general-purpose methods intended to improve upon the efficiency of standard algorithms based on Newton's method particularly on problems where the Jacobian matrix at the solution is singular or ill-conditioned. Each iteration is based upon a quadratic model (1.2) of the nonlinear function $F(x)$. The choice of the tensor term $T_c \in R^{n \times n \times n}$ in this model causes the second-order term $T_c dd$ in (1.2) 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 satisfies

$$F(x_{-k}) = F(x_c) + F'(x_c)s_k + \frac{1}{2}T_c s_k s_k, \qquad k = 1, \ldots, p, \tag{2.1}$$

where

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

The past points $x_{-1}, \ldots, x_{-p}$ 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 for finding linearly independent directions is implemented by using a modified Gram-Schmidt algorithm, and usually results in $p = 1$ or 2.

After the linearly independent past directions, s_k, are selected, the tensor term is chosen to be the smallest matrix that satisfies the interpolation conditions (2.1), that is,

$$\min_{T_c \in R^{n \times n \times n}} \quad || \, T_c \, ||_F \tag{2.2}$$

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

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

$$||T_c||_F{}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}(T_c[i,j,k])^2. \tag{2.3}$$

The solution to (2.3) is the sum of p rank-one tensors whose horizontal faces are symmetric,

$$T_c = \sum_{k=1}^{p} a_k s_k s_k, \tag{2.4}$$

where a_k is the k-th column of $A \in R^{n \times p}$, A defined by $A = ZM^{-1}$; Z is an $n \times p$ 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 $M(i,j) = ({s_i}^T s_j)^2$, $1 \le i, j \le p$.

Using the tensor term (2.4), we obtain the tensor model

$$M(x_c + d) = F(x_c) + F'(x_c)d + \frac{1}{2} \sum_{k=1}^{p} a_k \, (d^T s_k)^2. \tag{2.5}$$

The simple form of the quadratic term in (2.5) is the key to being able to efficiently form, store, and solve the tensor model. For dense problems, the cost of forming the tensor term in the tensor model is $O(n^2p) \leq O(n^{2.5})$ arithmetic operations, since $p \leq \sqrt{n}$. The leading term comes from the p matrix-vector products $F'(x_c)s_k$. The next most significant cost is the $O(np^2)$ operations required to calculate $A = ZM^{-1}$, and the $O(np^2)$ cost of the Gram-Schmidt orthogonalization. The additional storage required is $4p$ n-vectors.

Once the tensor model (2.5) 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, the problem

$$\text{find} \quad d \in R^n \quad \text{that minimizes} \ \| \ M(x_c \ + \ d) \ \|_2 \tag{2.6}$$

is solved. Schnabel and Frank [20] show that the solution to (2.6) can be reduced to the solution of q quadratic equations in p unknowns (i.e., a very small system of quadratics), 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 $\text{rank}(F'(x_c)) \geq n-p$, and q is greater than p otherwise. In the dense case, the main steps of the algorithm used to solve (2.6) are the following:

1. An orthogonal transformation of the variable space is used to cause the n 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 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 [21], is used to minimize the l_2 norm of the 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 $\hat{d}_1$.

An advantage of this algorithm is that it efficiently and stably solves (2.6), whether or not the tensor model has a root or the Jacobian is nonsingular.

In the dense case, the arithmetic cost per iteration of the above algorithm is the standard $O(n^3)$ cost of a matrix factorization, plus an additional $O(n^2p)$ ($\leq O(n^{2.5})$) operations for the orthogonal transformations, plus the cost of using UNCMIN [21] 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 $O(p^3)$ operations ($O(p^2q)$ when $q > p$) and a small multiple of p iterations generally suffice. Thus, the total cost of the above algorithm is the $O(n^3)$ cost of Newton's method plus at most an additional cost of $O(n^{2.5})$ arithmetic operations. The Newton step is computed inexpensively (in $O(n^2p) \leq O(n^{2.5})$ operations) as a by-product of the tensor step solution.

An iteration of the tensor method is summarized in Algorithm 2.1 below. For more details on tensor methods, including the global strategy used in Step 5 of Algorithm 2.1, see Schnabel

and Frank [20] and Bouaricha and Schnabel [4].

Algorithm 2.1. An Iteration of the Tensor Method for Dense Nonlinear Equations

Given n, current iterate x_c, $F(x_c)$

Step 0 Calculate $F'(x_c)$, and decide whether to stop. If not:

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 1.

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 Select the next iterate x_+ using either a line search global strategy or a two-dimensional trust region method.

Step 5 Set $x_c \leftarrow x_+$, $F(x_c) \leftarrow F(x_+)$; go to Step 1.

Now consider applying Algorithm 2.1 to large sparse systems of nonlinear equations. The leading costs of the tensor model formation are p Jacobian-vector products, to form $F'(x_c)s_k$; n solutions of a dense $p \times p$ systems of linear equations with the same $p \times p$ matrix M to form A; and a Gram-Schmidt orthogonalization of p n-vectors. Thus, as long as p is restricted to being less than or equal to a very small integer (rather than $p \leq \sqrt{n}$ as for dense problems), these costs are small for large sparse problems: the p Jacobian-vector products can be calculated efficiently by utilizing the sparsity of the Jacobian, and the remaining costs total a small multiple of n operations. Since dense tensor methods generally choose $p = 1$ or 2 anyhow, even when $\sqrt{n}$ is considerably larger, the restriction on the size of p is not a problem. In fact, our test software will be seen to use $p = 1$ because larger values did not improve its performance.

The procedure for solving the tensor model in the dense case, however, does not adapt to large sparse problems. The first step of this process, the orthogonal transformation of the variable space, is crucial to this approach and would destroy the sparsity of the Jacobian, making the remaining steps have an $O(n^3)$ cost even if the Jacobian had been sparse. Therefore, if tensor methods are to be applied to large sparse problems, an entirely different method for solving the tensor model is needed. This is developed in the next section.

3. Solving the Tensor Model When the Jacobian Is Sparse

As motivated in Section 2, the key challenge in developing an efficient tensor method for large sparse systems of nonlinear equations is to construct an efficient algorithm for finding a root of the tensor model (2.5) when the Jacobian matrix is large and sparse. That is,

$$\text{Find } d \in R^n \text{ such that}$$

$$M(x_c + d) = F(x_c) + F'(x_c)d + \frac{1}{2}\sum_{k=1}^{p} a_k \{d^T s_k\}^2 = 0, \tag{3.1}$$

where $F'(x_c)$ is large and sparse. We give such an algorithm in this section. We show that the solution of (3.1) can be reduced to the solution of a system of p quadratic equations in p unknowns, plus the solution of $p+1$ systems of linear equations that all involve the same matrix, $J(x_c)$, if this one is nonsingular and well conditioned. We also show that our algorithm efficiently solves the generalization of (3.1),

$$\text{find } d \in R^n \text{ that minimizes } ||M(x_c + d)||_2. \tag{3.2}$$

The basic approach used in all these cases is illustrated by the case when the Jacobian matrix is nonsingular and the tensor model has a root. In this case, premultiplying (3.1) by ${s_i}^T J^{-1}, i = 1, \ldots, p$, gives the p quadratic equations in the p unknowns $\beta_i = {s_i}^T d$,

$${s_i}^T J^{-1} F + \beta_i + \frac{1}{2}\sum_{k=1}^{p} ({s_i}^T J^{-1} a_k){\beta_k}^2 = 0, \quad i = 1, \ldots, p. \tag{3.3}$$

(Here and in the remainder of this section, we let F denote $F(x_c)$ and J denote $F'(x_c)$.) These equations can be solved for $\beta_i, i = 1, \ldots, p$, and then from (3.1) the equation

$$F + Jd + \frac{1}{2}\sum_{k=1}^{p} a_k {\beta_k}^2 = 0$$

can be solved for d. The entire process requires the solution of $p+1$ systems of linear equations in the matrix J to compute $J^{-1}F$ and $J^{-1}a_k, k = 1, \ldots, p$ (or, alternatively, $J^{-1}(F + \frac{1}{2}\sum_{k=1}^{p} a_k {\beta_k}^2)$ and $J^{-T}s_i, i = 1, \ldots, p$) and the solution of the small system of quadratics (3.3).

3.1. Solving the Sparse Tensor Model When the Jacobian Is Nonsingular

The preceding paragraph indicated how to solve (3.2) efficiently when the Jacobian matrix is nonsingular and the tensor model has a root. Now we address the more general problem of solving (3.2) efficiently whether or not the model has a root, when the Jacobian matrix is nonsingular. We do this by considering the equivalent minimization problem to (3.2),

$$\min_{d \in R^n} ||QM(x_c + d)||_2, \tag{3.4}$$

where Q is an $n \times n$ orthogonal matrix that has the structure

$$Q = \begin{bmatrix} U^T \\ Z^T \end{bmatrix},$$

with

$$U \in \Re^{n \times p};\ U = J^{-T}S[S^T(J^TJ)^{-1}S]^{-\frac{1}{2}},\ S \text{ an } (n \times p) \text{ matrix}$$
$$\text{whose columns are } s_i = 1, \ldots, p$$

$$Z \in \Re^{n\times(n-p)} \text{ is an orthonormal basis for the orthogonal complement of the subspace spanned by the columns of } J^{-T}S.$$

Note that $Z^T J^{-T} S = 0$. If we define $W = [S^T(J^TJ)^{-1}S]$, $\beta = S^T d$, and

$$q(\beta) = S^T J^{-1} F + \beta + \frac{1}{2} S^T J^{-1} A \beta^2,$$

where β^2 denotes the vector in $\Re^p$ whose i-th component is $(\beta_i)^2$, then

$$QM(x_c + d) = \begin{bmatrix} W^{-\frac{1}{2}} q(\beta) \\ Z^T M(x_c + d) \end{bmatrix}. \tag{3.5}$$

The following lemma is the key to showing that (3.4) can be solved efficiently through (3.5).

Lemma 3.1. For any $\beta \in \Re^p$, there exists a $d \in \Re^n$ such that $Z^T M(x_c + d) = 0$ and $S^T d = \beta$.
Proof. Let

$$d = (J^TJ)^{-1} S W^{-1} \beta + J^{-1} Z t, \tag{3.6}$$

where t is an arbitrary vector $\in \Re^{n-p}$. Then

$$S^T d = S^T (J^TJ)^{-1} S W^{-1} \beta + S^T J^{-1} Z t = \beta,$$

from the definitions of W and Z, and

$$\begin{aligned} Z^T M(x_c + d) &= Z^T F + Z^T J[(J^TJ)^{-1} S W^{-1} \beta + J^{-1} Z t] + \tfrac{1}{2} Z^T A \beta^2 \\ &= Z^T F + t + \tfrac{1}{2} Z^T A \beta^2. \end{aligned}$$

Thus the choice

$$t = -Z^T[F + \frac{1}{2} A \beta^2]$$

in (3.6) yields a value of d for which $Z^T M(x_c + d) = 0$ and $S^T d = \beta$ are both satisfied.□

Since for any β we are able to find a step d such that $Z^T M(x_c + d) = 0$ and $S^T d = \beta$, Lemma 3.1 and (3.5) show that the minimization problem (3.4) can be reduced to the minimization problem in p variables

$$\min_{\beta \in \Re^p} ||W^{-\frac{1}{2}} q(\beta)||_2. \tag{3.7}$$

Furthermore, once the value of β that solves (3.7) is determined, we can obtain the solution d to (3.4) efficiently as follows. From (3.5) and Lemma 3.1, d_* must satisfy

$$\begin{aligned} M(x_c + d_*) &= Q^T \begin{bmatrix} W^{-\frac{1}{2}} q(\beta) \\ 0 \end{bmatrix} \\ &= U W^{-\frac{1}{2}} q(\beta). \end{aligned}$$

From this equation and the definition of U we have

$$F + Jd_* + \frac{1}{2}A\beta^2 \;=\; J^{-T}SW^{-1}q(\beta)$$

and, hence,

$$d_* \;=\; -J^{-1}[F + \frac{1}{2}A\beta^2 - J^{-T}SW^{-1}q(\beta)]. \tag{3.8}$$

Therefore, once we know β, we simply calculate the value of $q(\beta)$ and substitute these two values into Equation (3.8) to obtain the value of d_*.

Now we can give the implementation that we use to solve (3.2).

Algorithm 3.1. Solving the Sparse Tensor Model When J Is Nonsingular

Let $J \in R^{n\times n}$ be sparse, $F \in R^n$, S, $A \in R^{n\times p}$.

Step 0 Form the $q(\beta)$ equations (3.3) by calculating $J^{-T}S$ as follows: factor J, and solve $J^T y_j = s_j, j = 1, \ldots, p$.

Step 1 Form the positive definite matrix $W \in R^{p\times p}$, where $W_{ij} = [s_i{}^T(J^TJ)^{-1}s_j]$, $1 \leq i, j \leq p$, as follows: $W_{ij} = (J^{-T}s_i)^T(J^{-T}s_j) = y_i{}^Ty_j$.

Step 2 Perform a Cholesky decomposition of W (i.e., $W = LL^T$), resulting in $L \in R^{p\times p}$, a lower triangular matrix.

Step 3 Use UNCMIN ([21]), an unconstrained minimization software package, to solve

$$\min_{\beta\in R^p} \; || \; L^{-1} \; q(\; \beta \;)||_2{}^2, \tag{3.9}$$

or solve (3.9) in closed form if $p = 1$.

Step 4 Substitute the values of β and $q(\beta)$ into

$$d \;=\; -J^{-1}(F \;+\; \frac{1}{2}A\beta^2 \;-\; J^{-T}SW^{-1}q(\beta)) \tag{3.10}$$

to obtain the tensor step d; this involves one additional solve, since the factorization of J is already calculated.

The total cost of this process is the factorization of the sparse matrix J, $p + 1$ back solves using this factorization, the unconstrained minimization of a function of p variables, and some lower-order ($O(n)$) costs.

4. Solving the Newton Model Along with the Sparse Tensor Model

As in the dense case [20, 4], the global strategy that is used in our tensor method for sparse nonlinear equations sometimes utilizes the Newton step rather than the tensor step (see Section 5). In the dense case, the Newton step can be computed inexpensively as a by-product of computing the tensor step. In this section, we show that this computation can also be done in the large sparse case.

If the Jacobian matrix J is nonsingular, then the calculation of the tensor step described above produces a sparse LU factorization of J. In this case, the Newton step is simply found by performing one additional pair of triangular solves to solve the system

$$Jd \;=\; -\,F. \tag{4.1}$$

That is, since

$$J \;=\; {P_1}^T L U {P_2}^T, \tag{4.2}$$

where $L \in R^{n\times n}$ is unit lower triangular, $U \in R^{n\times n}$ is upper triangular, and P_1 and P_2 are row and column permutation matrices, we first solve

$$Ly \;=\; c \tag{4.3}$$

for y, where $y \;=\; U{P_2}^T d$ and $c \;=\; -\,P_1 F$. Then we solve

$$Uz \;=\; y \tag{4.4}$$

for z, where $z = \; {P_2}^T d$. Finally $d \;=\; P_2\, z$. Our algorithm uses the MA28 package [11] to perform the sparse matrix factorization and triangular solves.

If the matrix J is singular, then (4.1) has either zero or an infinite number of solutions. Therefore, we would like to solve the least squares problem

$$\min_{d\in\, R^n} \quad ||Jd+F||_2. \tag{4.5}$$

The method that we use to solve the problem (4.5) is an extension of the method of Peters and Wilkinson [19] that was suggested by Bjorck and Duff [2]. This approach usually produces a better solution to (4.5) than the one obtained by using the MA28 package, which sets the last r components of the solution z in (4.4) to 0, where r is the rank deficiency of J. In particular, on problems where singular or very nearly singular Jacobians are encountered, Newton-based methods using the step produced by the Bjorck and Duff method usually require fewer iterations than those using the step produced by MA28. The remainder of this section reviews the method of Bjorck and Duff.

The first step in the method of Bjorck and Duff [2] is to compute an LU factorization of the Jacobian matrix J, using Gaussian elimination with both row and column interchanges. This is equivalent to multiplying a permutation of J from the left by the product, G, of a sequence of elementary elimination matrices, to obtain

$$GP_1JP_2 = \begin{pmatrix} U \\ 0 \end{pmatrix}, \tag{4.6}$$

where P_1, P_2 are permutation matrices, and U is an $r \times n$ upper trapezoidal matrix with $r = rank(J)$. If we apply the same transformations to the right-hand side $b = -F$, we obtain

$$GP_1 b = \begin{pmatrix} c \\ e \end{pmatrix}, \tag{4.7}$$

where $c \in R^r$ and $e \in R^{n-r}$

If we look at this in terms of an LU decomposition of J,

$$P_1 J P_2 = LU, \tag{4.8}$$

with L a unit lower trapezoidal $n \times r$ matrix, we have

$$P_1 b = Lc + \begin{pmatrix} 0 \\ e \end{pmatrix}. \tag{4.9}$$

Now if d_s is any solution of the system

$$U{P_2}^T d = c, \tag{4.10}$$

the residual norm corresponding to it is given by

$$||Jd_s - b||_2 = ||P_1(Jd_s - b)||_2 = ||Lc - Lc - \begin{pmatrix} 0 \\ e \end{pmatrix}||_2 = ||e||_2. \tag{4.11}$$

Thus, if $||e||_2 < \epsilon$ (ϵ some suitable tolerance), then d_s is a solution to (4.5) with a slightly perturbed right-hand side b, and we can immediately accept d_s as the solution to our problem at the cost of a simple forward elimination (4.7) and back substitution (4.10).

However, if $||e||_2$ is larger, we would like to solve the least squares problem using our initial decomposition (4.6) and (4.7). For an arbitrary d we have that

$$\begin{aligned} P_1(Jd - b) &= LU{P_2}^T d - Lc - \begin{pmatrix} 0 \\ e \end{pmatrix} \\ &= Lz - \begin{pmatrix} 0 \\ e \end{pmatrix}, \end{aligned} \tag{4.12}$$

where

$$U{P_2}^T d = c + z. \tag{4.13}$$

Therefore, d is a least squares solution of (4.5) if it satisfies (4.13), where z is the solution of

$$\text{minimize } ||Lz - \begin{pmatrix} 0 \\ e \end{pmatrix}||_2. \tag{4.14}$$

This least squares problem can be solved by using the $(n+r) \times (n+r)$ augmented system matrix

$$\begin{bmatrix} 0 & L^T \\ L & I \end{bmatrix} \begin{bmatrix} z \\ \rho \end{bmatrix} = \begin{bmatrix} 0 \\ e \end{bmatrix}, \tag{4.15}$$

followed by the solution of (4.13) for d. Here ρ is the residual of (4.14). We use the augmented system approach because it is an efficient method in terms of preservation of sparsity and accuracy.

Hence, if $||e||_2$ is larger than ϵ, then d is the solution to (4.5) at the cost of one forward solve (4.7), one back solve (4.10), an LU factorization of the augmented matrix (4.15) followed by one forward and one backward solve using the resulting factors, and a back solve (4.13).

An advantage of Bjorck and Duff's method is that (4.14) is used only to compute a correction to the equations (4.13). Hence, for problems with small residuals, this method should be reasonably stable, since any ill-conditioning in L will affect only the correction z. Also, L is less likely than U to be ill conditioned. Furthermore, since

$$(J^T Jd \;=\; -J^T F) \Rightarrow d^T J^T F \;=\; -d^T J^T Jd \leq 0, \tag{4.16}$$

the solution d to (4.5) is a descent direction unless $Jd \;=\; 0$, which would imply that $J^T F \;=\; 0$. Hence d is a descent direction unless we are at a root of $F(x)$ or a critical point of $||F(x)||_2{}^2$. The step produced by MA28 when J is singular does not necessarily have this property.

5. Implementation of Tensor Methods for Sparse Nonlinear Equations

This section gives a complete high-level description of an iteration of the sparse tensor method for nonlinear equations that is used in our computational tests. The description includes some more details about the sparse matrix factorization than were given in preceding sections, and a description of the global strategy. We present test results for this implementation in Section 6.

As stated previously, the sparse linear equation solutions in our implementation use the MA28 package [11], a widely used package for solving large, sparse, unsymmetric systems of linear equations. To detect near-singularity of the Jacobian, we have modified the factorization phase of MA28 to be able to detect the row and column indices of the first pivot whose absolute value is less than or equal to some given tolerance, *tol*, times the largest element in absolute value in the pivot row. This stability test detects a sufficient condition for the condition number of the Jacobian to be greater than a given tolerance. While it is clearly not an optimal test for ill-conditioning, it appears to work well in practice. Also, as mentioned previously, the implementation reported here uses only one past iterate at each iteration to form the tensor term T_c (i.e., $p = 1$). We use only one past point because our tests indicated that no further improvements were obtained by allowing a larger number of past points. In addition, using $p =$ 1 reduces the storage requirement and cost per iteration of the tensor method, allows the tensor model to be solved in closed form, and does not require an unconstrained optimization package. The entire additional cost of an iteration of the tensor method with $p = 1$, in comparison with Newton's method, is essentially one sparse matrix vector multiplication of $F'(x_c)$ times a vector to form the tensor model, one additional upper and lower triangular solve to solve the tensor model, and sometimes a second additional pair of triangular solves to calculate the Newton step. Some parts of Algorithm 5.1 are still stated in terms of arbitrary p, for generality.

The global strategy that is used in our implementation is a standard line search. In [4], both line search and two-dimensional trust region strategies were used in tensor methods for small, dense systems of nonlinear equations. In the tests in that paper, both methods appeared to be

equally robust, with the trust region method possibly having a small advantage in efficiency. We have used the line search in the sparse code, however, because of its greater simplicity and because the two-dimensional trust region method requires two additional matrix-vector multiplications involving the Jacobian matrix.

The line search strategy that we use is identical to that developed and used in [20] and [4], so we review it only very briefly here. If the full tensor step provides sufficient decrease in $||F(x)||$, it is taken. Otherwise, line searches usually are conducted in both the tensor and Newton directions, resulting in two possible next iterates, and the point with the lower function value is chosen as the next iterate. (The extra cost of this dual line search strategy, usually one function evaluation per iteration, has proven empirically to be justified by the decrease in the number of iterations required.) However, if the tensor step is not a descent direction or the Jacobian matrix is singular, the line search is based solely upon the Newton direction.

Algorithm 5.1. An Iteration of the Tensor Method for Sparse Nonlinear Equations

Given current iterate $x_c, F(x_c)$

Step 0 Calculate $J = F'(x_c)$ analytically or by finite-difference approximations [6, 7], and decide whether to stop. If not:

Step 1 Form the second-order term of the tensor model, T_c, so that the tensor model interpolates $F(x)$ at the most recent past point (i.e., $p = 1$).

Step 2 Factorize J by using the MA28 software package [11].

Step 3 If J has full rank, perform Algorithm 3.1 on the tensor model $M(x_c + d) = F(x_c) + Jd + \frac{1}{2}\sum_{k=1}^{p} a_k(d^T s_k)^2$ to compute the tensor step d_t, and go to Step 4. Else:

- **Step 3.1** Calculate the Newton step d_n from the LU factorization of J by the Bjorck and Duff [2] method to find some solution to $\min_{d\in R^n} ||Jd + F||_2$.
- **Step 3.2** Select the next iterate x_+ by using the line search algorithm 5.2 outlined below, where d_n is the search direction, and go to Step 5.

Step 4 Select the next iterate x_+ by using a line search global strategy as follows:

- **Step 4.1** If $x_c + d_t$ is acceptable, set $x_+ = x_c + d_t$ and go to Step 5. Else:
- **Step 4.2** Calculate the Newton step d_n from the LU factorization of J (or as in Step 3.1 if J is singular). Then calculate $x_+^n = x_c + \lambda d_n$ for some $\lambda > 0$, using Algorithm 5.2.
- **Step 4.3** If the tensor step is a descent direction, then calculate $x_+^t = x_c + \lambda d_t$ for some $\lambda > 0$, using Algorithm 5.2.
- **Step 4.4** If $||F(x_+^n)||_2 > ||F(x_+^t)||_2$, then $x_+ \leftarrow x_+^t$; else $x_+ \leftarrow x_+^n$.

Step 5 Set $x_c \leftarrow x_+, F(x_c) \leftarrow F(x_+)$. Go to Step 0.

Algorithm 5.2. Standard Quadratic Backtracking Line Search

Given x_c, search direction 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**

$\lambda_{temp} := -\lambda \cdot$ 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

The sparse tensor code (and the Newton code) terminates successfully if the relative size of $(x_+ - x_c)$ is less than $macheps^{\frac{2}{3}}$, or $||F(x_+)||_\infty$ is less than $macheps^{\frac{2}{3}}$; it terminates unsuccessfully if the iteration limit is exceeded. If the last global step fails to locate a point lower than x_c in the line search global strategy or if the relative size of $J(x_+)^T F(x_+)$ is less than $macheps^{\frac{1}{3}}$, the method stops and reports this condition; this situation may indicate either success or failure.

6. Test Results

This section describes the comparative testing of the sparse tensor method from Section 5 with an analogous implementation based upon a linear model (Newton's method). The Newton's method algorithm is identical to the tensor Algorithm 5.1 except that the tensor model is never formed or solved, and the next iterate x_+ is calculated from a line search based solely on the Newton search direction d_n. That is, it uses steps 0, 2, 4.2/3.1, and 5 of Algorithm 5.1. As in Algorithm 5.1, the Jacobian matrix is factored at each iteration by using the MA28 package, and if the Jacobian is singular, the search direction is calculated by the method of Bjorck and Duff. In all our experiments, we calculate the Jacobian matrix by finite-difference approximations [6, 7].

We tested these algorithms on a variety of nonsingular and singular problems. First we tested them on three sparse problems provided to us from Boeing Computer Services (BCS) and used as test problems in [13]. These problems are described as follows:

1. LTS : This problem discretizes the differential equations for the Linear Tangent Steering problem in Bryson and Ho [5]. This is the search problem formulation where the adjoint differential equations are also discretized and an optimality condition is imposed.
2. GRST : This problem discretizes the differential equations for a coast about a spherical earth. The problem is initialized on the equator in an orbit with inclination of 1.0 radians. A search problem is obtained by requiring that the vehicle be at a given latitude at the final time. There are two solutions for every desired final latitude that is less than the inclination in absolute value. There is a single solution if the final latitude is required to be greater than the inclination.
3. LGNDR : The recurrence relation for the Legendre polynomials is used to generate a sparse

system of nonlinear equations equivalent to finding the value of x at which the n-th Legendre degree polynomial is equal to 1.
We then tested our methods on a system of sparse trigonometric equations from [18] that have the form

$$\sum_{j=1}^{n}(a_{ij} sin\ x_j + b_{ij} cos\ x_j) + \sum_{j=1}^{n} c_{ij}x_j \;=\; d_j, \qquad i \;=\; 1,2,\ldots,n, \tag{6.1}$$

where the matrices $\{a_{ij}\}$ and $\{b_{ij}\}$ have the same sparsity pattern as one another, including nonzeros on the diagonal, and $\{c_{ij}\}$ has a different sparsity pattern. Each matrix is a band matrix consisting of a main diagonal and zero, one, or two superdiagonals and subdiagonals that are each distance two apart. The nonzero values are generated randomly in $[0,1]$. The solution components are generated randomly in $[0,1]$, and the right-hand side vector d is calculated from the solution. For the starting iterate we randomly perturb the components of the solution by adding or subtracting 0.1 from each.

We also ran our methods on some sparse nonlinear equations problems from Moré, Garbow, and Hillstrom [17], namely, the Broyden banded, the Broyden tridiagonal, and the variable-dimension test problems, and on the distillation column test problem from [16]. Finally, we tested our methods on the MINPACK-2 test problems collection [1], namely, the driven cavity, the flow in a channel, the incompressible elastic rod, and the swirling flow between disks problems. In the remainder of this section, we will refer to the trigonometric, the BCS, and the Moré, Garbow, and Hillstrom test problems as the TBM collection.

These problems all have nonsingular Jacobians at the solution. We created singular test problems as proposed in Schnabel and Frank [20] by modifying these nonsingular test problems to the form

$$\hat{F}(x) \;=\; F(x) - F'(x_*)A(A^TA)^{-1}A^T(x - x_*), \tag{6.2}$$

where $F(x)$ is the standard nonsingular test function, x_* is its root, and $A \in R^{n\times k}$ has full column rank with $1 \le k \le n$. Note that $\hat{F}(x)$ also has a root at x_* and $\mathrm{rank}(\hat{F}'(x_*)) \;=\; n - \mathrm{rank}(A)$. We used (6.2) to create two sets of sparse singular problems, with $\hat{F}'(x_*)$ having rank $n-1$ and $n-2$, respectively, by using the matrices $A \in R^{n\times 1}$ and $R^{n\times 2}$ whose columns are the unit vectors e_1, and $\{e_1, e_2\}$, respectively. Note that these changes do not affect the sparsity pattern of the Jacobian, except possibly for the first and second diagonal elements.

The dimensions of the test problems we ran ranged from $n \;=\; 31$ to $n \;=\; 324$, with eleven of the fourteen problems we used having dimension 300 or greater. For each test problem, we used several different starting guesses, generated by

$$\hat{x}_0 \;=\; x_0 + const(x_0 \;-\; x_*), \tag{6.3}$$

where *const* is an real number indicating how far the initial guess is from the solution, and x_* is the solution resulting from running the problem with initial guess x_0. 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.

Tables 6.1 and 6.2 summarize the performance of the sparse tensor and sparse Newton methods on the TBM and MINPACK-2 test problems collections. Each subtable in Table 6.1 presents the test results for a nonsingular test problem and for its rank $n-1$ and rank $n-2$ singular

versions, whereas each subtable in Table 6.2 presents the test results only for a nonsingular test problem. Columns "Better" and "Worse" represent the number of times the tensor method was better and worse, respectively, than Newton's method by more than one iteration, over all the starting points for the problem under consideration. The "Tie" column represents the number of times the tensor and Newton methods required within one iteration of each other. The columns labeled "Average Ratio" measure the efficiency of the tensor method against Newton's method; for example, if the test set contained two problems for which the tensor method required 3 and 5 iterations, respectively, and Newton's method 7 and 9 iterations, respectively, then the average ratio would be $\frac{3+5}{7+9} = 0.50$. The same measure is used for execution times and function evaluations. These average ratios include only problems that were successfully solved by both methods. Columns "Itns", "Time", and "Fevals" represent the number of iterations, the execution time, and the number of function evaluations, respectively. Problems that were solved by only one method are included in the "Better" and "Worse" columns, however, and the numbers of such problems are discussed below. We have excluded entirely from Tables 6.1 and 6.2 all cases where the tensor and Newton methods converge to different roots, or to the same root but not the singular root x_* for a singular problem.

Though nonlinear systems of equations may sometimes converge to a stationary point that is not a root, this convergence failure has not occurred in our test experiments. The convergence failures that were encountered in our test results are (1) iteration limit exceeded and (2) failure of the last global step to locate a point lower than the current iterate in the line search global strategy. This latter failure generally occurs when the generated tensor or Newton steps are too small that the line search fails to satisfy the sufficient decrease conditions in the objective function.

Table 6.3 presents the average iteration, execution time, and function evaluation ratios of the tensor method versus Newton's method for all of the rank n, $n-1$, and $n-2$ problems that are included in the average ratio statistics in Tables 6.1 and 6.2. The one exception is that we exclude the rank $n-2$ versions of the Legendre problem from the last line in Table 6.3 because in almost all cases, the tensor and Newton methods converge to a different root. For the three cases where the two methods converged to the same root, the tensor method was dramatically more efficient than the Newton method. These results are so different from any of the others that it seemed best to eliminate them from the summary statistics. Their inclusion would change the numbers in the last line of Table 6.3 to 0.43, 0.51, and 0.43.

On the basis of Table 6.1, the following observations can be made. The tensor method virtually never is less efficient than Newton's method and usually is more efficient in terms of iterations, function evaluations, and execution times. The improvement by the tensor method over Newton's method is substantial, averaging about 50% in iterations, 40% in execution times, and 50% in function evaluations, if all problems are considered. For all the nonsingular problems, the improvement averages 40% in iterations, 28% in execution times, and 41% in function evaluations. For problems where $F'(x_*)$ has a small rank deficiency, the improvement is greater. It averages 56% in iterations, 45% in execution times, and 46% in function evaluations for rank $n-1$ problems, and 52% in iterations, 45% in execution times, and 52% in function evaluations for rank $n-2$ problems. In the case of the rank $n-1$ problems, this advantage is due in part to the tensor method achieving 3 step Q-order $\frac{3}{2}$ convergence whereas Newton's method is linearly convergent ([12]).

The tensor method also has a substantial advantage in robustness in comparison with Newton's method on this test set. Over all the test problems, 12 nonsingular problems, 11 rank $n-1$ problems, and 12 rank $n-2$ problems were solved by the tensor and not by Newton's method. On the other hand, there were no problems that were solved by Newton's method and not by the tensor method.

A close look at the numerical performance of the tensor and Newton methods is presented in Table 6.4. This table shows the test results for the TBM collection in which the values of *const* in the equation for the starting point, 6.3, are given in the first column. Note that "Fevals" also includes the function evaluations required by the finite-difference approximations of the Jacobian matrix, and that 150 in the "Itns" column means that the iteration limit has been exceeded. Table 6.4 clearly shows that the tensor method outperforms the Newton method by a large margin on the rank $n-1$ and $n-2$ versions of the TBM test problems collection. This is a reflection of faster local convergence by the tensor method on singular problems with a small rank deficiency. The substantial gains by the tensor method over Newton's method on the rank $n-2$ TBM test problems are somewhat surprising because in theory the tensor method may not always achieve faster than linear convergence on problems where the rank of $F'(x_*)$ is $n-2$ or less [20]. Though the tensor and Newton methods are both quadratically convergent on nonsingular problems, the tensor method is clearly the more efficient on the nonsingular TBM test problems. The improvement by the tensor method over the Newton method on the nonsingular TBM test problems, however, is less dramatic.

Another important observation that can be made on the basis of Table 6.3 is that the average improvement of the tensor method over Newton's method in execution times is about 10% smaller than in iterations. This is primarily because a tensor iteration requires at least one more pair of triangular solves than a Newton iteration (two more if both the tensor and Newton directions are calculated), and one additional matrix vector multiplication. The increased cost per iteration ranges from 12% on problems with relatively expensive function evaluations, like the LTS problem, to 57% on problems with very sparse Jacobians and inexpensive function evaluations, like the Broyden tridiagonal problem. (Note that one exception is the rank n and $n-2$ versions of the trigonometric problem in Table 6.1. Here, the average execution time improvement is about 5% more than the average iteration improvement. This is because Newton's method line search requires many nonunit steps on this problem, as is clearly indicated by the large improvement in function evaluations, and because function evaluations are expensive for the trigonometric test problem.)

The MINPACK-2 test results presented in Table 6.2 show that the tensor method performs very slightly worse than its Newton counterpart on the DFIC problem, essentially the same on the DSFD problem, and considerably better on the DFDC (Reynolds = 400), DFDC (Reynolds = 1000), and the DIER problems. Note that all of these functions are relatively inexpensive with respect to the finite-difference approximation of the Jacobian matrix and its factorization, which explains why the improvement by the tensor method over Newton's method in execution times on the DFDC ((Reynolds = 400), DFDC (Reynolds = 1000), and DIER problems is still significant even though on function evaluations both methods perform about the same. Over all the runs on the MINPACK-2 collection, two problems were solved by the tensor method and not by Newton's method, whereas no problems were solved by Newton's method and not by the tensor method. Both methods failed to converge on many problems for which the starting

Table 6.1: Summary Statistics of the Test Results for the TBM Collection

LTS Problem

Dimension	Rank	Tensor			Average Ratio–Tensor/Newton		
n	$F'(x_*)$	Better	Worse	Tie	Itns	Time	Fevals
313	n	12	2	3	0.78	0.90	0.84
	$n-1$	11	0	0	0.55	0.67	0.60
	$n-2$	7	0	0	0.62	0.68	0.65
GRST Problem							
324	n	2	0	5	0.52	0.63	0.52
	$n-1$	14	0	0	0.48	0.57	0.51
	$n-2$	14	0	1	0.46	0.53	0.43
LGNDR Problem							
50	n	13	0	0	0.86	1.02	0.86
	$n-1$	7	0	1	0.45	0.87	0.51
	$n-2$	3	0	0	0.10	0.15	0.10
Trigonometric Problem							
300	n	4	1	2	0.40	0.35	0.21
	$n-1$	5	0	1	0.47	0.47	0.31
	$n-2$	8	0	0	0.42	0.38	0.26
Broyden Banded Problem							
300	n	11	0	0	0.81	0.95	0.83
	$n-1$	11	0	0	0.69	0.81	0.69
	$n-2$	11	0	0	0.66	0.77	0.64
Broyden Tridiagonal Problem							
300	n	11	0	0	0.30	0.48	0.35
	$n-1$	11	0	0	0.23	0.36	0.27
	$n-2$	11	0	0	0.31	0.47	0.50
Variable Dimension Problem							
300	n	11	0	0	0.36	0.39	0.38
	$n-1$	11	0	0	0.38	0.40	0.39
	$n-2$	10	0	0	0.34	0.36	0.35
Distillation Column Problem (31 Variables)							
31	n	5	0	10	0.93	1.16	0.95
	$n-1$	4	0	0	0.43	0.48	0.40
	$n-2$	8	0	0	0.53	0.66	0.53
Distillation Column Problem (99 Variables)							
99	n	3	0	4	0.45	0.61	0.44
	$n-1$	6	0	0	0.31	0.34	0.31
	$n-2$	5	0	0	0.50	0.60	0.49

points were chosen to be relatively far from the solution. Based on our previous experience with trust region and line search methods [4], we believe that a trust region strategy often would have helped on these cases.

We examined our test results to obtain an experimental indication of the local convergence behavior of the tensor method and Newton's method on problems where $\text{rank}(F'(x_*)) = n-1$. Specifically, we examined the sequence of ratios

$$||x^k - x_*||/||x^{k-1} - x_*|| \tag{6.4}$$

produced by the Newton and tensor methods on problems with $\text{rank}(F'(x_*)) = n-1$. The ratios for a typical problem are given in Table 6.5. In almost all cases the standard method exhibits local linear convergence with constant near 0.5, which is consistent with the theoretical analysis (see, e.g., [9, 10]). The local convergence rate of the tensor method is faster, with a typical final ratio of around 0.01. This final ratio might be smaller if analytic Jacobians were used in combination with tighter stopping tolerances. As is anticipated in [12], the convergence usually seems to be one-step superlinear, although only a three-step Q-order $\frac{3}{2}$ result can be proven.

Finally, we also tried, on most of the test problems, a variant of the tensor method that allows up to two past points (i.e., $p \leq 2$) to be used in the tensor model formation. There was almost no difference in terms of number of iterations or function evaluations. There was, however, an increase in execution time by approximately 10% to 20% when we allowed two past points. This is due in part to the extra pair of triangular solves required per tensor iteration, because when $p \leq 2$ a total of up to 3 solves may be performed.

Overall, the size and consistency of the efficiency gains indicate that the tensor method may be preferable to the linear model-based method for solving large sparse systems of nonlinear equations. The tensor method seems to obtain a surprisingly large improvement from a comparatively small amount of additional information. In particular, the tensor method using only one past point seems to be more efficient than the tensor method using more than one past point, from the viewpoints of execution time and storage.

Table 6.2: Summary Statistics of the Test Results for the MINPACK-2 Collection

Dimension	Rank	Tensor			Average Ratio–Tensor/Newton		
n	$F'(x_*)$	Better	Worse	Tie	Itns	Time	Fevals
Driven Cavity Problem (DFDC) (Reynolds = 400)							
304	n	2	0	3	0.69	0.75	0.98
Driven Cavity Problem (DFDC) (Reynolds = 1000)							
303	n	2	0	0	0.52	0.51	0.94
Flow in a Channel Problem (DFIC)							
308	n	1	0	4	1.06	1.03	1.20
Incompressible Elastic Rod Problem (DIER)							
324	n	1	0	0	0.61	0.64	0.83
Swirling Flow between Disks Problem (DSFD)							
324	n	0	0	6	1.00	0.99	1.04

Table 6.3: Average Ratios of Tensor Method versus Newton's Method for the TBM Collection

Rank	Tensor		
$F'(x_*)$	Itns	Time	Fevals
n	0.60	0.72	0.59
$n-1$	0.44	0.55	0.44
$n-2$	0.48	0.55	0.48

7. Summary and Future Work

We have developed and tested an efficient tensor method for solving large sparse systems of nonlinear equations. The method, like previous tensor methods for nonlinear equations, is based upon using a second-order model of the nonlinear equations at each iteration. The tensor model is formed in the same way as in the previous tensor method research for small, dense nonlinear equations ([20, 4]), since this approach still is efficient for large sparse problems. The solution of the tensor model, however, uses an entirely new approach. With this new approach, we are able to make the main step of the tensor model solution procedure be a (sparse) factorization of the Jacobian matrix, which can be performed efficiently. In contrast, previous approaches for solving the tensor model required orthogonal transformations to the Jacobian matrix, which would destroy its sparsity, before performing a matrix factorization. The approach also allows a minimizer of the tensor model to be found efficiently if no root exists.

In computational comparisons using an analogous code based on Newton's method, the tensor method is significantly more efficient in terms of iterations, function evaluations, and execution times. The advantages of the tensor method are greater on singular problems than on nonsingular problems, but are large in both cases, averaging about 30% to 40% for nonsingular problems and about 45% to 55% for problem with small rank deficiencies. The tensor method code also solves considerably more problems successfully than Newton's method code. The most effective tensor method uses a rank-one second-order term, in which the tensor model interpolates the function value at just the previous iterate. The additional storage and arithmetic cost per iteration needed to use this tensor model are particularly small.

We are continuing to refine and test the software corresponding to the methods described in this paper, and plan to make it generally available in the near future. We have also developed tensor methods for solving large, sparse nonlinear least squares problems. The issues involved are considerably different because of the different large sparse linear algebraic computations that are required. This work is described in [3] and in a forthcoming paper. Finally, we continue to develop variants of tensor methods for solving very large systems of nonlinear equations that are based on iterative linear solvers such as Krylov subspace methods.

References

[1] B. M. Averick, R. G. Carter, J. J. Moré, and G. L. Xue. The MINPACK-2 test problem collection. Preprint MCS-P153-0692, Mathematics and Computer Science Division, Argonne

Table 6.4: Test Results of the Tensor and Newton Methods for Some Values of *const*

const	Newton			Tensor		
	Itns	Time	Fevals	Itns	Time	Fevals
LTS Problem						
4	24	19.37	467	14	13.70	283
4	46	37.10	866	17	16.21	343
4	150	204.36	3676	40	62.33	802
GRST Problem						
10	50	24.45	831	13	7.66	224
10	68	33.68	1065	24	13.46	393
10	72	36.38	1176	24	13.97	370
LGNDR Problem						
10	74	2.45	375	69	2.78	350
10	75	2.40	381	26	1.65	166
10	150	5.35	1564	8	0.38	57
Trigonometric Problem						
1	28	10.55	425	11	3.49	80
1	18	6.54	225	13	4.27	91
1	66	24.67	939	25	9.63	283
Broyden Banded Problem						
0	22	5.76	184	18	5.54	152
0	37	9.72	321	20	6.34	168
0	44	11.63	411	23	7.06	192
Broyden Tridiagonal Problem						
0	14	1.67	60	5	0.97	24
0	27	2.99	112	7	1.23	32
0	31	3.63	151	9	1.47	40
Variable Dimension Problem						
0	24	42.48	7525	5	10.21	1806
0	44	87.31	13546	12	26.01	3913
0	44	91.25	13546	14	30.88	4515
Distillation Column Problem (31 Variables)						
1	5	0.21	72	5	0.25	72
1	19	0.96	280	7	0.39	96
1	26	0.87	357	12	0.51	157
Distillation Column Problem (99 Variables)						
0	8	1.38	136	8	1.55	140
0	20	4.11	315	11	2.49	182
0	26	3.75	436	13	2.29	211

Table 6.5: Speed of Convergence on the LTS Problem ($n = 313$), modified by (6.2) to have $\text{rank}(\hat{F}'(x_*)) = n - 1$, started from x_0. The ratios in the second and third columns are defined by (6.4)

Iteration (k)	Tensor Method	Standard Method
.	...	...
3	0.9789	0.9789
4	0.9511	0.9511
5	0.9899	0.9396
6	0.9710	0.9289
7	0.9362	0.8863
8	0.9207	0.7632
9	0.8209	0.4815
10	0.4955	0.6176
11	0.5573	0.4443
12	0.3450	0.6730
13	0.6667	0.5756
14	0.1131	0.2224
15	0.1104	0.4119
16	0.1233	0.7639
17	0.6085	0.9472
18	0.5505	0.9474
19	0.9529	0.9476
20	0.1571	0.9477
21	0.1032	0.9478
22	0.0440	0.9480
23	0.0095	0.9481
24		0.9482
25		0.9483
26		0.9484
27		0.9477
28		0.9445
29		0.9409
30		0.9367
31		0.9317
32		0.9258
33		0.9187
34		0.9100
35		0.8989
36		0.8846
37		0.8654
38		0.8390
39		0.8013
40		0.4967
41		0.4996
42		0.4998
43		0.4999
44		0.4999
45		0.4999

National Laboratory, 1992.

[2] A. Bjorck and I. S. Duff. A direct method for the solution of sparse linear least squares problems. *Linear Algebra and Its Applications*, 34:43–67, 1980.

[3] 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, Computer Science Department, University of Colorado at Boulder, 1992.

[4] A. Bouaricha and R. B. Schnabel. TENSOLVE: A software package for solving systems of nonlinear equations and nonlinear least squares problems using tensor methods. Technical Report CU-CS-735-94, Department of Computer Science, University of Colorado at Boulder, 1994.

[5] A. E. Bryson and A. E. Ho. *Applied Optimal Control,* Chap. 2. Wiley, New York, 1975.

[6] T. F. Coleman, B. S. Garbow, and J. J. Moré. Fortran subroutines for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 10:346–347, 1984.

[7] T. F. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 10:329–345, 1984.

[8] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20:187–207, 1983.

[9] D. W. Decker and C. T. Kelly. Newton's method at singular points I. *SIAM J. Numer. Anal.*, 17:66–70, 1980.

[10] D. W. Decker and C. T. Kelly. Newton's method at singular points II. *SIAM J. Numer. Anal.*, 17:465–471, 1980.

[11] I. S. Duff. MA28: A set of Fortran subroutines for for sparse unsymmetric linear equations. Technical Report R-8730, AERE Harwell Laboratory, 1977.

[12] D. Feng, P. Frank, and R. B. Schnabel. Local convergence analysis of tensor methods for nonlinear equations. *Math. Prog.*, 62:427–459, 1993.

[13] P. D. Frank and W. P. Huffman. Parallel solution of large and sparse nonlinear systems. Technical Report ECA-TR-128, Boeing Computer Services, 1989.

[14] Andreas Griewank. On automatic differentiation. In *Mathematical programming: Recent developments and applications*, pages 83–108, Amsterdam, 1989. Kluwer Academic Publishers, Amsterdam.

[15] Andreas Griewank and George F. Corliss, editors. *Automatic differentiation of algorithms: Theory, implementation, and application.* Society for Industrial and Applied Mathematics, 1991.

[16] J. J. Moré. A collection of nonlinear model problems. In E. L. Allgower and K. Georg, editors, *Computational solution of nonlinear systems of equations*, volume 26 of *Lecture Notes in Applied Mathematics*, pages 723–762. American Mathematical Society, 1990.

[17] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Software*, 7:17–41, 1981.

[18] N. Munksgaard. NS02: A Fortran subroutine for solving sparse sets of nonlinear equations by Powell's Dog-leg algorithm. Technical Report R-11047, AERE Harwell Laboratory, 1938.

[19] G. Peters and J. H. Wilkinson. The least squares problem and pseudo-inverses. *Computer J.*, 13:309–316, 1970.

[20] R. B. Schnabel and P. D. Frank. Tensor methods for nonlinear equations. *SIAM J. Numer. Anal.*, 21:815–843, 1984.

[21] R. B. Schnabel, J. E. Koontz, and B. E. Weiss. A modular system of algorithms of unconstrained minimization. *ACM Trans. Math. Softw.*, 11:419–440, 1985.