Tensor Methods for Large, Sparse Nonlinear Least Squares Problems

Ali Bouaricha*
Argonne National Laboratory
and
Robert B. Schnabel†
University of Colorado

Abstract

This paper introduces tensor methods for solving large, sparse nonlinear least squares problems where the Jacobian either is analytically available or is computed by finite difference approximations. Tensor methods have been shown to have very good computational performance for small to medium-sized, dense nonlinear least squares problems. In this paper we consider the application of tensor methods to large, sparse nonlinear least squares problems where a sparse factorization of the Jacobian can be stored. This involves an entirely new way of solving the tensor model that is efficient for sparse problems. A number of interesting linear algebraic implementation issues are addressed. The test results of the tensor method applied to a set of sparse nonlinear least squares problems compared with those of the standard Gauss-Newton method reveal that the tensor method is significantly more robust and efficient than the standard Gauss-Newton method.

Key Words. tensor methods, sparse problems, large-scale optimization, rank-deficient matrices

AMS(MOS) subject classification. 65F20, 65F50, 65H10, 65K05, 65K10

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439, bouarich@mcs.anl.gov. Research supported in part 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.

†Department of Computer Science, University of Colorado, Boulder, Colorado 80309-0430, 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

In this paper we introduce tensor methods for large, sparse nonlinear least squares problems. The nonlinear least squares problem is

$$\underset{x \in \Re^n}{\text{minimize}} \, \| F(x) \|_2, \;\; F(x) : \Re^n \to \Re^m, \;\; m \geq n, \tag{1.1}$$

where it is assumed that $F(x)$ is at least once continuously differentiable and $F'(x)$ is sparse. Large sparse nonlinear least squares arise especially in data-fitting problems. Often, $F'(x_*)$ is ill conditioned or singular with a small rank deficiency. For example, this occurs if two parameters in the model that is being fit are closely correlated. Large, sparse nonlinear least squares methods usually are solved by sparse variants of the Gauss-Newton method. Tensor methods are especially intended to improve the efficiency of standard algorithms based on the Gauss-Newton method when $F'(x_*)$ is singular or ill conditioned. Tensor methods are also intended to be at least as efficient as standard methods on problems where $F'(x_*)$ is nonsingular and well conditioned, and in practice they often prove to be considerably more efficient on these problems as well.

The tensor model came initially from the research of [23] for nonlinear equations and was extended to nonlinear least squares in [7]. For both nonlinear equations and nonlinear least squares, the tensor model is 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, $F'(x_c) \in \Re^{m \times n}$ is the Jacobian matrix at x_c, and $T_c \in \Re^{m \times n \times n}$ is the tensor term at x_c. The notation $T_c dd$ is defined as follows.

Definition 1.1. Let $T \in \Re^{n \times n \times n}$. Then T is composed of n horizontal faces $H_i \in \Re^{n \times n}, i = 1, \ldots, n$, where $H_i[j,k] = T[i,j,k]$. For $v, w \in \Re^n, Tvw \in \Re^n$ with

$$Tvw[i] = v^T H_i w = \sum_{j=1}^{n} \sum_{k=1}^{n} T[i,j,k] v[j] w[k].$$

The tensor term is selected so that the model interpolates a very small number, p, of function values from previous iterations. Hence, T_c is a very low rank tensor, which is crucial to the efficiency of the tensor method. Methods for forming the tensor term and solving the tensor model for dense nonlinear least squares problems are reviewed in more detail in the next section.

Methods based on (1.2) have been implemented in `TENSOLVE` [7], a software package for solving small to medium–sized nonlinear least squares problems, say with up to 100 variables and observations. These methods have been shown to be efficient in terms of algorithmic overhead and to exhibit fast local convergence as well. In the test results obtained for both nonsingular and singular dense nonlinear least squares problems, the improvement of the tensor method over the standard Gauss-Newton method is substantial, averaging about 52% in iterations and 53% in function evaluations when the line search is used in each, and about 35% in iterations and 28% in function evaluations when the trust region is used in each, on problems solved by both methods. Furthermore, the tensor method solves several problems that the Gauss-Newton method does not, and the reverse is never the case. In [7], the tensor method has also been compared with the NL2SOL package [13] on a set of nonlinear least squares problems described in [13]. These problems include a mixture of zero, small, and large residual problems. The obtained test results showed that on these test problems, the line search tensor method is more efficient than NL2SOL, with an average improvement of about 58% in iterations and 29% in function evaluations, whereas the trust region tensor method is more efficient than NL2SOL with an average improvement of about 24% in iterations and 7% in function evaluations. The two packages were equally robust as they solved the same number of test problems.

In order to extend tensor methods to large, sparse nonlinear least squares problems, several key issues need to be considered. First, tensor methods require that the Jacobian matrix be available at each iteration. It turns out that this requirement is not a problem in the large, sparse case because derivatives usually are computed by using efficient sparse finite differences (see [9, 10, 11, 12]) or automatic differentiation (see, e.g., [3]). Second, the method for forming the tensor model must be modified to adapt to the large, sparse case. Again, the solution is simple: we need only to change the upper bound on the number of recent past points from $\sqrt{n}$ to a much smaller number, say 1 or 2. This process eliminates virtually all of the cost of the modified Gram-Schmidt procedure for selecting the past points that is used in the dense case (see Section 2). In addition, since the number of past points selected for the interpolation process is almost always equal to 1 or 2 in practice, the performance of the tensor method probably will not be affected by this modification. The final question is how to use the sparsity of the Jacobian matrix efficiently in computing the tensor step. The tensor algorithms that have been designed for dense problems are QR-based algorithms involving orthogonal transformations of both the variable and function space. QR-based algorithms are very effective for solving the tensor model when the Jacobian is dense, because they are efficient and very stable numerically, especially when the tensor model has no real root or when the Jacobian is singular or ill conditioned. However, QR-based algorithms are not efficient for sparse problems because they destroy the sparsity of the Jacobian as a result of the orthogonal transformation of the variable space.

In this paper we address this last problem by developing an entirely new way of solving the tensor model that is efficient for sparse problems. We consider a number of interesting linear algebraic implementation issues. We also show how to solve the tensor model efficiently when the Jacobian is singular and how to compute the Gauss-Newton step, which is used in the tensor algorithm, as a by-product of the tensor step calculation. We then give test results of the tensor method applied to a set of large, sparse nonlinear least squares problems. These results indicate that the tensor method is significantly more robust and efficient than the Gauss-Newton method, in terms of iterations, function evaluations, and execution time.

In general, the sparse tensor algorithms behave similarly to the dense tensor algorithms implemented in TENSOLVE [7], in terms of iterations and function evaluations. The fact that p is restricted to a much smaller value in the sparse implementation has only affected the execution time.

The remainder of this paper is organized as follows. In Section 2 we briefly review tensor methods for dense nonlinear least squares problems. In Section 3 we describe efficient algorithms for solving the tensor model in the cases when the Jacobian matrix has full rank and when it is rank deficient. Then, in Section 4, we discuss the large, sparse linear least squares problems that arise from tensor algorithms, and we present our approach for solving these least squares problems. In Section 5 we describe a method to compute the Gauss-Newton step as a by-product of the solution of the tensor model. In Section 6 we give an overall implementation of the tensor method for sparse, nonlinear least squares problems; present summary statistics of the test results; and provide a detailed analysis of these results. Finally, in Section 7, we summarize our work and discuss future research.

2 Overview of Tensor Methods for Dense Nonlinear Least Squares

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. Each iteration is based upon a quadratic model (1.2) of the nonlinear function $F(x)$. The particular choice of the tensor term $T_c \in \Re^{m\times n\times n}$ causes the second-order term $T_c dd$ in (1.2) to have a simple and useful form.

In tensor methods, the tensor term T_c is chosen to allow the model $M(x_c + d)$ to interpolate values of the function $F(x)$ at p past iterates x_{-k}. That is, the model should satisfy

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

where

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

For dense problems, 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 of finding linearly independent directions is implemented easily 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 by the procedure of Schnabel and Frank [23], which generalizes in a straightforward way to nonlinear least squares. T_c is chosen to be the smallest matrix that satisfies the interpolation conditions (2.1), that is,

$$\underset{T_c \in \Re^{m\times n\times n}}{\text{minimize}} \; \|T_c\|_F \tag{2.2}$$

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

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

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

The solution to (2.2) is the sum of p rank-one tensors whose horizontal faces are symmetric:

$$T_c = \sum_{k=1}^{p} a_k s_k s_k, \tag{2.3}$$

where a_k is the k-th column of $A \in \Re^{m \times p}$, 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 $M_{ij} = ({s_i}^T s_j)^2$, $1 \leq i, j \leq p$.

If we use the tensor term (2.3), the tensor model (1.2) becomes

$$M(x_c + d) = F(x_c) + F'(x_c)d + \frac{1}{2}\sum_{k=1}^{p} a_k\{d^T s_k\}^2. \tag{2.4}$$

The simple form of the quadratic term in (2.4) 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 for dense problems 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.

After the tensor model (2.4) is formed, the least squares solution of the model,

$$\underset{d \in \Re^n}{\text{minimize}} \, \| M(x_c + d) \|_2, \tag{2.5}$$

is computed. In [7], we showed that the solution to (2.5) can be reduced to the solution of a small number $(m - n + q)$ quadratic equations in p unknowns, plus the solution of $n - q$ linear equations in $n - p$ unknowns (see Step 2 of the algorithm below). 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 $\hat{d}_1 \in \Re^{n-p}$, and quadratic only in the remaining p variables $\hat{d}_2 \in \Re^p$.

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 [24], 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 $O(p^3)$ $(O(p^2 q)$ when $q > p)$ operations and a small multiple of p iterations generally suffices. 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 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 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 Gauss-Newton step, a line search or a two-dimensional trust region global strategy determines the next iterate. The overall algorithm is summarized below.

Algorithm 2.1 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 1.

Step 3 Find a minimizer (in the l_2 norm) of the tensor model.

Step 4 Compute the Gauss-Newton step as a by-product of the tensor model solution.

Step 5 Compute the next iterate x_+ using a line search or trust region global strategy based on the tensor or Gauss-Newton step.

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

At each iteration, the tensor method bases its step upon either the tensor step or the Gauss-Newton step. The Gauss-Newton step is chosen whenever the tensor step is not a descent direction, or the tensor step is a minimizer of the tensor model and does not provide enough decrease in the tensor model, or the quadratic system of $m - n + q$ equations in p unknowns cannot be solved by UNCMIN within the iteration limit. Otherwise the tensor step is chosen. For further details, see [7].

3 Solving the Tensor Model for Sparse Nonlinear Least Squares

The major challenge in constructing an efficient tensor method for sparse nonlinear least squares problems is finding a way to obtain the root or minimizer of the tensor model that preserves and effectively uses the sparsity of the Jacobian matrix. This section presents such a method that works both when the Jacobian is nonsingular and when it is rank-deficient. The basic approach is related to the approach given in [6] for solving the tensor model for sparse nonlinear equations, but the situation for nonlinear least squares is considerably different and more complex, as are the linear algebraic issues (considered in the following sections). For simplicity, we use the matrix notation $A\{S^T d\}^2$ to denote the quadratic term $\sum_{k=1}^{p} a_k \{d^T s_k\}^2$ of the tensor model, where $A \in \Re^{m \times p}$, with a_k being the k-th column of A, $S \in \Re^{n \times p}$, with s_k being the k-th column of S, and $(\{S^T d\}^2)_i = (s_i^T d)^2$.

3.1 Solving the Tensor Model When the Jacobian Has Full Rank

In this section we show how to reduce the solution of the tensor model problem (2.5) to a sparse matrix factorization, a very small number of backsolves using this factorization, and a very small unconstrained optimization problem. Let $J = F'(x_c)$, and $J^+ \in \Re^{n\times m}$ be the pseudo-inverse matrix of J (i.e., $J^+ = (J^TJ)^{-1}J^T$). Let Q be an orthogonal matrix that has the structure

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

where $U \in \Re^{m\times p}$, $U = J^{+T}S(S^T(J^TJ)^{-1}S)^{-\frac{1}{2}}$; $N \in \Re^{m\times(m-n)}$ is an orthonormal basis for the null space of J^T; and $Z \in \Re^{m\times(n-p)}$ is an orthonormal basis for the subspace orthogonal to the subspace spanned by the columns of ${J^+}^TS$ and N. Note that $Z^T{J^+}^TS = Z^TN = 0$. Now consider the equivalent minimization problem to (2.5),

$$\underset{d\in\Re^n}{\text{minimize}} \, \|QM(x_c+d)\|_2. \tag{3.1}$$

If we define $W = S^T(J^TJ)^{-1}S = LL^T$, $\beta = S^Td$, and

$$q(\beta) = S^TJ^+F + \beta + \frac{1}{2}S^TJ^+A\beta^2, \tag{3.2}$$

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

$$QM(x_c+d) = \begin{bmatrix} L^{-1}q(\beta) \\ N^TF + \frac{1}{2}N^TA\beta^2 \\ Z^TM(x_c+d) \end{bmatrix}. \tag{3.3}$$

The following lemma is the key to showing that (3.1) can be solved efficiently through (3.3).

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

$$d = (J^TJ)^{-1}SW^{-1}\beta + J^+Zt, \tag{3.4}$$

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

$$S^Td = S^T(J^TJ)^{-1}SW^{-1}\beta + S^TJ^+Zt = \beta,$$

and

$$Z^TM(x_c+d) = Z^TF + Z^T{J^+}^TSW^{-1}\beta + Z^TJJ^+Zt + \tfrac{1}{2}Z^TA\beta^2. \tag{3.5}$$

From the definitions of N and J^+

$$JJ^+ = I - NN^T.$$

If we substitute the above expression for JJ^+ into the expression Z^TJJ^+Z, we get

$$Z^TJJ^+Z = I.$$

Finally, since $Z^T {J^+}^T S = 0$, equation (3.5) simplifies to

$$Z^T M(x_c + d) = Z^T F + t + \frac{1}{2} Z^T A\beta^2. \tag{3.6}$$

Thus the choice

$$t = -Z^T(F + \frac{1}{2}A\beta^2) \tag{3.7}$$

in (3.4) yields a value of d for which $Z^T M(x_c + d) = 0$ and $S^T d = \beta$. □

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 (3.3) show that problem (3.1) can be reduced to the following minimization problem in p variables:

$$\underset{\beta \in \Re^p}{\text{minimize}} \left\| \begin{bmatrix} L^{-1} q(\beta) \\ n(\beta) \end{bmatrix} \right\|_2^2, \tag{3.8}$$

where $n(\beta) = N^T F + \frac{1}{2} N^T A\beta^2$. Furthermore, once the value of β that solves problem (3.8) is determined, we can obtain the solution d to (3.1) as follows. Substituting (3.7) into (3.4) yields

$$d = (J^T J)^{-1} S W^{-1} \beta - J^+ Z Z^T (F + \frac{1}{2} A\beta^2). \tag{3.9}$$

Since Q is orthogonal, we have

$$ZZ^T = I - NN^T - UU^T = I - NN^T - {J^+}^T S W^{-1} S^T J^+.$$

The substitution of the above expression for ZZ^T into equation (3.9) yields

$$\begin{aligned} d \;\; = \;\; & (J^T J)^{-1} S W^{-1} \beta - J^+(F + \tfrac{1}{2} A\beta^2) + J^+ N N^T (F + \tfrac{1}{2} A\beta^2) \\ & + J^+ {J^+}^T S W^{-1} S^T J^+ (F + \tfrac{1}{2} A\beta^2). \end{aligned} \tag{3.10}$$

Since $J^T N = 0$ and $J^+ {J^+}^T = (J^T J)^{-1}$ equation (3.10) simplifies to

$$d = (J^T J)^{-1} S W^{-1} q(\beta) - J^+(F + \frac{1}{2} A\beta^2). \tag{3.11}$$

Thus, once we know β, we simply calculate the value of $q(\beta)$ and substitute these two values into equation (3.11) to obtain the value of d.

Now we discuss how to determine the value of β that minimizes (3.8). First, we need to calculate $n(\beta)$, and in particular $N^T F$ and $N^T A$. Since the minimization problem (3.8) is equivalent to

$$\underset{\beta \in \Re^p}{\text{minimize}} \{ \| L^{-1} q(\beta) \|_2^2 + \| n(\beta) \|_2^2 \}, \tag{3.12}$$

where

$$\| n(\beta) \|_2^2 = F^T N N^T F + F^T N N^T A\beta^2 + \frac{1}{4} \beta^2 A^T N N^T A \beta^2,$$

we actually need to compute only $NN^T F$ and $NN^T A$. But $NN^T F$ is the projection of F into the orthogonal complement of the subspace spanned by the columns of J and therefore is the residual of the least squares problem $Jx = F$. Similarly, NN^TA are the projections of the columns a_j of the matrix A into the orthogonal complement of the subspace spanned by the columns of J and hence are the residuals of the least squares problems $Jx_j = a_j,\ j = 1, \ldots, p$. Hence, these quantities are available from the computation of J^+F and J^+A, by

$$NN^TF = F - JJ^+F,$$

and

$$NN^TA = A - JJ^+A,$$

respectively. Thus, if we let R_1 be the residual vector of the least squares problem $Jx = F$ and let R_2 be the matrix whose columns are the residuals of the least squares problems $Jx_j = a_j,\ j = 1, \ldots, p$, then

$$\|n(\beta)\|_2^2 = F^TR_1 + {R_1}^TA\beta^2 + \frac{1}{4}\beta^2A^TR_2\beta^2 = \|R_1 + \frac{1}{2}\beta^2R_2\|_2^2. \tag{3.13}$$

Now we can give an efficient algorithm that solves problem 3.1. (An alternative algorithm was considered in [5] but proved to be less efficient.)

Algorithm 3.1 Minimization of the Tensor Model When the Jacobian Has Full Rank

Let $J \in \Re^{m\times n}$ be sparse, $F \in \Re^m$, $S \in \Re^{n\times p}$, and $A \in \Re^{m\times p}$, $m > n$.

Step 0 Form $q(\beta)$ (i.e., equation (3.2)) as follows: Form J^+F and J^+A, by solving the sparse linear least squares problem $Jx = F$ and the sparse linear least squares problems $Jx_j = a_j,\ j = 1, \ldots, p$, respectively, using the augmented matrix approach (4.1). These linear least squares solutions will yield the values of R_1 and R_2 as well.

Step 1 Form the positive definite matrix $W = S^T(J^TJ)^{-1}S$ as follows: Form $(J^TJ)^{-1}S$, by solving the sparse system of equations $(J^TJ)x_j = s_j, j = 1, \ldots, p$, using the augmented matrix approach (4.4). Then premultiply $(J^TJ)^{-1}S$ by S^T to obtain W.

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

Step 3 Use UNCMIN [24], an unconstrained minimization software package, to solve

$$\underset{\beta\in\Re^p}{\text{minimize}}\{\|L^{-1}q(\beta)\|_2^2 + \|n(\beta)\|_2^2\}, \tag{3.14}$$

where $n(\beta) = R_1 + \frac{1}{2}\beta^2R_2$, or solve (3.14) in closed form if $p = 1$.

Step 4 Substitute the value of the solution to (3.14), β_*, and $q(\beta_*)$ into the following equation for d,

$$d = (J^TJ)^{-1}SW^{-1}q(\beta_*) - J^+F - \frac{1}{2}J^+A\beta_*^2. \tag{3.15}$$

The total cost of Algorithm 3.1 is the factorization of the sparse augmented matrix in (4.1), $2p+1$ solves using this factorization, the unconstrained minimization of a function of p variables, and some lower-order ($O(n)$) costs.

3.2 Solving the Tensor Model When the Jacobian Is Rank Deficient

An important attribute of tensor methods is that the tensor model may have a well-defined solution even if the Jacobian matrix at the current iterate is singular. A stable solution procedure is given in [7] for dense nonlinear least squares. This section gives a method that is efficient when the Jacobian is large and sparse.

If the Jacobian matrix is rank deficient, we can solve the tensor model by building upon the process just described. Specifically, we transform the tensor model given in (2.4) as follows. Let $d = \hat{d} + \delta$ and $\hat{\beta} = S^T \hat{d}$ for a fixed $\hat{d}$, where δ is the new unknown. Substituting this expression for d in the tensor model (2.4) yields the following model, which becomes a function of δ:

$$M(x_c + \delta) = F(x_c) + J(x_c)(\hat{d} + \delta) + \frac{1}{2} A\{S^T(\hat{d} + \delta)\}^2. \tag{3.16}$$

Equation (3.16) is equivalent to

$$\hat{M}(x_c + \delta) = \hat{F}(x_c) + \hat{J}(x_c)\delta + \frac{1}{2} A\{S^T \delta\}^2, \tag{3.17}$$

where $D_{\hat{\beta}} = diag(\hat{\beta})$, $\hat{F}(x_c) = F(x_c) + J(x_c)\hat{d} + \frac{1}{2}A\{S^T\hat{d}\}^2$, and $\hat{J}(x_c) = J(x_c) + AD_{\hat{\beta}}S^T$.

To solve this model in the case when $J(x_c)$ is singular, we try to find the value of δ that minimizes $\|\hat{M}(x_c + \delta)\|_2$. If $rank(J) \geq n - p$, usually $\hat{J}$ will have full column rank (a necessary and sufficient condition for $\hat{J}$ to be nonsingular is similar to that for sparse nonlinear equations [6]). Thus, if $\hat{J}$ has full rank, we can use Algorithm 3.1 to obtain the value of δ, but we use a special procedure (described in Section 4.2) to efficiently solve the linear least squares problems involving $\hat{J}^+$. We then obtain the tensor step by adding the value of δ to the fixed step $\hat{d}$. An appropriate choice of $\hat{d}$ would be the step computed in the previous iteration. We will use the singular Gauss-Newton step computed in Section 5 as the step direction for the current iteration in the case when $\hat{J}$ is rank deficient. The entire procedure is given in Algorithm 6.1.

4 Solving the Sparse Least Squares Problems

The algorithm described in the preceding section requires the solution of large, sparse linear least squares problems. The method that we use for solving these linear least squares problems (i.e., minimize$_{x \in \Re^n} \|Jx - b\|_2$) is based on the augmented system approach first proposed by Hachtel [21],

$$\begin{bmatrix} I & J \\ J^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{4.1}$$

where I is the $m \times m$ identity matrix. The first block row of equation (4.1) states that the residual vector is given by

$$r = b - Jx, \tag{4.2}$$

and the second block row requires that

$$J^T r = 0 \tag{4.3}$$

be satisfied. Substituting (4.2) into (4.3) gives the normal equations, thereby showing that x given by (4.1) solves the linear least squares problem. The system (4.1) is symmetric and indefinite and can be solved by sparse techniques.

One reason that we use (4.1) is the efficiency that this approach has been shown to have. Duff and Reid [16] compared four methods for the solution for sparse linear least squares problems: the direct formation and solution of normal equations, orthogonal reduction to upper triangular form [20, 18], LU factorization of A [22], and Hachtel's augmented matrix method (4.1). The criteria used for comparison were the number of operations for matrix decomposition, storage, and number of operations required for subsequent solutions. Duff and Reid found that in every instance the best algorithm is either use of normal equations or Hachtel's augmented matrix method. Because of ill conditioning with the use of normal equations however, they recommend the augmented matrix method.

A second reason we use (4.1) is that various research has shown the method to be quite accurate for large, sparse linear least squares. Duff and Reid [16] conducted tests on the augmented matrix method, avoiding any multiplier bigger in modulus (or norm) than a limit u and making this stability requirement override sparsity considerations. With u set to 20 they found essentially no growth in the size of the largest matrix element (and on some problems they found no such growth with u as high as 10^5). The conditioning of the augmented system (4.1) was studied by Björck [4]. He considered the slightly generalized system obtained by scaling the augmented system with a scaling factor α. A simple analysis shows that if α is chosen equal to the smallest singular value of A, the condition number of the augmented system is approximately equal to 1.6 times the condition number of A, and is about the same as the condition number of A when A is ill conditioned. Arioli et al. [2] demonstrated that conditioning can be greatly improved by a scaling similar to that of Björck; they suggested an automatic technique for selecting the best scaling factor α. Unfortunately, both Björck and Arioli et al. use a condition number estimator to calculate α, which would make a tensor iteration very expensive. We decided to use the heuristic scaling strategy discussed in [19], which sets the value of α to the maximum element of the matrix in absolute value divided by 1000.

A recent study has shown that methods based on a sparse QR decomposition can perform better than methods based on (4.1) in terms of accuracy and efficiency [5, Sec. 6.9.2]. Unfortunately, these methods cannot be extended to solve the tensor model when the Jacobian matrix is rank deficient. Such extension is, however, possible for methods based on (4.1). This extension is discussed in Section 6.

Given that the matrix of equation (4.1) is symmetric, one would normally expect that the solution scheme that took advantage of this fact would have much lower storage requirements for the factors than would an unsymmetric code. By ignoring the symmetry, however, one can bias the pivot selection so that early pivots are chosen from the last n rows (thereby exploiting the zeros in the right-hand side of (4.1) in the first triangular solve) and later pivots are chosen from the last n columns (thereby exploiting the fact that equation (4.1) need be solved for x only in the second triangular solve). By making this judicious choice of pivots and discarding the portions of the factors that are not required, Duff and Reid [16] obtained substantial savings in the number of operations for the triangular solves. Therefore Duff and Reid recommend using the unsymmetric factorization when there are multiple right-hand sides.

Using the collection of problems in [16], we computed the number of operations required for (1) one unsymmetric decomposition that gives a slight bias toward early pivots in late rows and late pivots in late columns, plus three back solves (which is the number of back solves required by the tensor method when $p = 1$ and the tensor model has no root, or when $p = 2$ and the tensor model has a root), and (2) one symmetric indefinite decomposition obtained by the method of Bunch [8] using an unstable partial pivoting scheme based on 2×2 pivots, plus three back solves. These computations showed that the unsymmetric decomposition is approximately 25% more efficient than the symmetric one. For greater values of p, the unsymmetric decomposition has an even larger advantage over the symmetric one. Based upon this experiment and the recommendation of Duff and Reid, we use Hachtel's method with an unsymmetric decomposition in our tensor method.

The remaining issue in the case when J is nonsingular is the pivoting strategy; this is discussed in Section 4.1. In the case when J is singular, based on Section 3.2 we instead wish to solve a linear least squares problem involving $\hat{J}$; this leads to additional issues that are discussed in Section 4.2.

4.1 Solving the Least Squares Problems When the Jacobian Has Full Rank

As discussed above, we wish to determine an unsymmetric row and column pivoting strategy for factoring the augmented matrix in (4.1) that reduces the costs of the backsolves needed in Algorithm 3.1. Note that Algorithm 3.1 involves both terms of the form J^+b and terms of the form $(J^TJ)^{-1}v$. Terms of the form J^+b will be computed by using (4.1). To compute terms of the form $(J^TJ)^{-1}v$, where $v \in \Re^n$, we solve the augmented system

$$\begin{bmatrix} I & J \\ J^T & 0 \end{bmatrix} \begin{bmatrix} -t \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ -v \end{bmatrix} \tag{4.4}$$

for z. From its two components

$$J^T t = v \tag{4.5}$$

and

$$Jz - t = 0, \tag{4.6}$$

(4.4) is equivalent to $(J^TJ)z = v$.

From Algorithm 3.1, it is seen that the computations of the form J^+b require both the solution x and the residual r. Thus, the column-pivoting decisions are irrelevant as far as saving computations in these backsolves. On the other hand, in solving (4.4) we need to compute z but not t. Therefore we construct the pivoting strategy to bias the last n pivots to come from the last n columns in order to save steps in the upper triangular backsolve. For the row-pivoting strategy, we can bias the selection so that the first backsolve can skip the computations involving either the n zeroes in the right-hand side of (4.1) or the m zeroes in the right-hand side of (4.4).

Since the latter choice results in a larger savings, we bias the first m pivots to come from the first m rows.

We use a sparse variant of Gaussian elimination on the augmented matrix (4.4), using the following Markowitz pivot selection strategy, which is based on a strategy in [16]. The sparsity cost of a nonzero in row i and column j is $r_i c_j$, where r_i is either the number of nonzeros in row i if $i \leq m$ or one more if $i > m$, and c_j is either the number of other nonzeros in column j if $j \leq m$ or one more if $j > m$. In general, this pivot strategy produces a factorization where the first $m_1 < m$ components of the vector $P_1 \begin{bmatrix} 0 & -v \end{bmatrix}^T$ are zeros, and z_1 is the position of the first z component in the vector ${P_2}^T \begin{bmatrix} -t & z \end{bmatrix}^T$. Therefore, we can omit the first m_1 steps in the lower triangular backsolve and disregard the first m_1 columns of this lower triangular factor, and can omit the last $z_1 - 1$ steps in the upper triangular backsolve and disregard the last $z_1 - 1$ rows of the upper triangular factor, in solving (4.4) for z.

4.2 Solving the Least Squares Problems When the Jacobian Is Rank Deficient

When the Jacobian is rank deficient, we need to solve linear least squares problems to compute $\hat{J}^+ F$ and $\hat{J}^+ A$, and linear systems of equations to compute $(\hat{J}^T \hat{J})^{-1} S$, where $\hat{J} = J + A D_{\hat{\beta}} S^T$, $J \in \Re^{m \times n}$, $A \in \Re^{m \times p}$, $D_{\hat{\beta}} \in \Re^{p \times p}$, $S \in \Re^{n \times p}$, and J is rank deficient.

To solve a least squares problem of the form

$$\underset{x \in \Re^n}{\text{minimize}} \, \| (J + A D_{\hat{\beta}} S^T) x - b \|_2, \tag{4.7}$$

where $x \in \Re^n$ and $b \in \Re^m$, we first write (4.7) as an augmented system of $(m+n)$ equations for the $(m+n)$ unknown components of r and x,

$$\begin{bmatrix} I & J + A D_{\hat{\beta}} S^T \\ J^T + S D_{\hat{\beta}} A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{4.8}$$

(As in (4.1), $r \in \Re^m$ is the residual of (4.7).) However, since $\hat{J} = J + A D_{\hat{\beta}} S^T$ is dense, forming and solving (4.8) would not be efficient. Instead, if we define

$$S^T x = y \tag{4.9}$$

and

$$D_{\hat{\beta}} A^T r = z, \tag{4.10}$$

the system (4.8) is equivalent to

$$\begin{bmatrix} I & J & AD_{\hat{\beta}} & 0 \\ J^T & 0 & 0 & S \\ D_{\hat{\beta}}A^T & 0 & 0 & -I \\ 0 & S^T & -I & 0 \end{bmatrix} \begin{bmatrix} r \\ x \\ y \\ z \end{bmatrix} \begin{bmatrix} b \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.11}$$

This system has $2p$ additional rows and columns in comparison with (4.8). Now, recall that the system (4.11) is constructed only if the augmented system (4.1) is rank deficient. That means that we have already factorized the augmented matrix in (4.1) and discovered it to be rank deficient. Therefore, we need to finish factorizing the remaining rows and columns starting at the point where (4.1) was discovered to be rank deficient, to obtain the factorization of the augmented matrix in (4.11). To detect the rank deficiency of (4.1), 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 (4.1) to be greater than a given tolerance. While it is clearly not an optimal test for detecting singularity, it appears to work well in practice. If $\hat{J}$ is singular, we calculate the Gauss-Newton step (see Section 5), which will be used as the search direction from the current iterate x_c for the line search global strategy.

Similarly to the full-rank case, we are also able to use the same augmented matrix that solves $\hat{J}^+F$ and $\hat{J}^+A$ to solve the systems of equations $(\hat{J}^T\hat{J})^{-1}S$. We proceed by solving the augmented system

$$\begin{bmatrix} I & \hat{J} \\ \hat{J}^T & 0 \end{bmatrix} \begin{bmatrix} -t_i \\ x_i \end{bmatrix} = \begin{bmatrix} 0 \\ -s_i \end{bmatrix} \tag{4.12}$$

for $x_i \in \Re^n,\ i = 1, \ldots, p$, where $t_i \in \Re^m$ and $s_i \in \Re^n =$ column i of S. If we define $S^T x_i = y$, and $D_{\hat{\beta}} A^T t_i$, the augmented system (4.12) can be rewritten as

$$\begin{bmatrix} I & J & AD_{\hat{\beta}} & 0 \\ J^T & 0 & 0 & S \\ D_{\hat{\beta}} A^T & 0 & 0 & -I \\ 0 & S^T & -I & 0 \end{bmatrix} \begin{bmatrix} -t_i \\ x_i \\ y_i \\ -z_i \end{bmatrix} \begin{bmatrix} 0 \\ -s_i \\ 0 \\ 0 \end{bmatrix} \tag{4.13}$$

for $x_i,\ i = 1, \ldots, p$. This system has the same matrix as (4.11), but a different zero pattern on the right-hand side.

As when J is nonsingular, we choose the pivoting strategy that is optimal for solving (4.13). That is, we wish to bias the first m pivots to come from the first m rows of the augmented matrix (4.13), and the last n pivots to come from the last n columns. In the best case, this strategy enables us to omit the first m steps in the lower triangular backsolve and disregard the first m columns of the lower triangular factor, as well as omit the last n steps in the upper triangular backsolve and disregard the last n rows of the upper triangular factor, when computing an expression of the form $(\hat{J}^T \hat{J})^{-1} v$.

5 Solving the Gauss-Newton Model

This section discusses a solution for the linear least squares problem

$$\underset{d \in \Re^n}{\text{minimize}} \| Jd + F \|_2 \tag{5.1}$$

along with the tensor model, where $J \in \Re^{m \times n}$ is sparse, and $F \in \Re^m$.

If J has full rank, the solution to the linear least squares problem (5.1) is already available from the computation of $J^+ F$ from Step 0 of Algorithm 3.1.

If the Jacobian matrix is rank deficient, however, we compute a solution d to (5.1) by solving the augmented system (4.1) for some solution x. Such a solution is found by setting any component of x corresponding to a zero pivot of a factorization of (4.1) to zero. Note that the augmented system (4.1) is always consistent and hence any solution to this system solves the least squares problem (5.1). Moreover, since

$$(J^T J d = -J^T F) => d^T J^T F = -d^T J^T J d \leq 0, \tag{5.2}$$

the solution d to (5.1) is a descent direction unless $Jd = 0$, which implies that $J^T F = 0$.

6 Implementation and Testing of Tensor Methods

In this section we summarize the overall implementation of tensor methods for solving sparse nonlinear least squares problems. We also describe testing of an algorithm based on this implementation compared with an algorithm that is based on the linear Gauss-Newton model but that is identical otherwise. We present analysis and summary statistics of the test results.

6.1 Implementation

The following algorithm is an implementation of an iteration of our tensor method for sparse nonlinear least squares problems. We have used only one past point ($p = 1$) in the implementation, since this again turned out to be the most efficient choice in virtually all cases in terms of overall excecution times.

Algorithm 6.1 An Iteration of the Tensor Method for Sparse Nonlinear Least Squares

Given x_c and $F(x_c)$, let M_1 and M_2 be the matrices given in (4.1) and (4.11), respectively.

Step 0 Calculate $J(x_c)$, and decide whether to stop.

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

Step 2 Form the augmented matrix M_1, and factorize it using the MA28 software package [15], employing the Markowitz pivoting strategy oriented toward efficiently computing an expression of the form $(J^T J)^{-1} v$, where $v \in \Re^n$ (see Section 4.1.)

Step 3 If M_1 has full rank, use Algorithm 3.1 to solve the tensor model $M(x_c + d) = F(x_c) + F'(x_c)d + \frac{1}{2}A\{S^T d\}^2$, to compute the tensor step d_t. Go to Step 5. Else

- **Step 3.1** Construct the augmented matrix M_2.
- **Step 3.2** Complete the factorization of M_2 as follows. Let ρ denote the rank of M_1.
  - **Step 3.2.1** Update the lower left $2p \times \rho$ rectangular submatrix, and the upper right $\rho \times 2p$ rectangular submatrix of M_2, using the multipliers stored in the L factor of the LU factorization of M_1.
  - **Step 3.2.2** Factor the lower right square $(m + n - \rho + 2p) \times (m + n - \rho + 2p)$ submatrix using the MA28 software package [15].
  - **Step 3.2.3** Update M_2 by combining the LU factorization of the submatrix in Step 3.2.2, the updated submatrices in Step 3.2.1, and the LU factorization of M_1, into one LU factorization of M_2.

Step 4 If J is rank deficient but M_2 has full rank, use Algorithm 3.1 to solve the tensor model $\hat{M}(x_c + \delta) = \hat{F}(x_c) + \hat{J}(x_c)\delta + \frac{1}{2}A\{S^T \delta\}^2$, where $\hat{F}(x_c) = F(x_c) + J(x_c)\hat{d} + \frac{1}{2}A\{S^T \hat{d}\}^2$, $\hat{J}(x_c) = J(x_c) + AD_{\hat{\beta}}S^T$, and $\hat{d}$ is the step computed in the previous iteration, to compute the step δ. Then set $d_t = \hat{d} + \delta$. (Any values of the form $\hat{J}^+ v$ or $(\hat{J}^T \hat{J})^{-1} v$ are computed using the augmented matrix approaches described in Section 4.2.) Else

- **Step 4.1** Calculate the singular Gauss-Newton step d_n as described in Section 5.
- **Step 4.2** Select the next iterate x_+ using a standard backtracking line search strategy [14], where d_n is the search direction; go to Step 6.

Step 5 Select the next iterate x_+ using the global framework described in Algorithm 6.2.

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

Note that in the sparse nonlinear least squares solver the same pivot sequence is reused in subsequent factorizations of the Jacobian matrix a long as the sparsisty stucture of the Jacobian has not been altered.

Algorithm 6.2 Global Framework for Line Search for Sparse Nonlinear Least Squares

Let x_c be the current iterate, d_n the Gauss-Newton step, d_t the tensor step, $g = J(x_c)^T F(x_c)$ the gradient at x_c, 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
  comment. Test if the tensor step is sufficiently descent
  if $g^T d_t < -10^{-4}||g||_2||d_t||_2$ **then**
    Find an acceptable x_+^t in the tensor direction d_t,
    by performing a standard backtracking line search [14]
    return $x_+ = x_+^t$
  else
    Perform a standard backtracking line search [14] on d_n to obtain x_+^n
    return $x_+ = x_+^n$
  endif
endif

6.2 Test Results

We have run the sparse tensor and Gauss-Newton codes on versions of the nonlinear least squares problems described by Al-Baali and Fletcher [1] and singular modifications of these problems. Both of these codes terminate successfully if the relative size of $(x_+ - x_c)$ is less than $macheps^{\frac{2}{3}}$, or $||F(x_+)||_\infty$ is less than $macheps^{\frac{2}{3}}$, or the relative size of $J(x_+)^T F(x_+)$ is less than $macheps^{\frac{1}{3}}$, and 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, the method stops and reports this condition; this may indicate either success or failure. We use the graph coloring algorithm of Coleman and Moré [11] to compute the sparse finite difference approximation of the Jacobian matrix.

The first type of test problem is the *Signomial* problem, defined by

$$r_i(x) = e_i + \sum_{k=1}^{l} c_{ik} \prod_{j=s}^{n,2} x_j{}^{a_{ijk}}, \quad i = 1, \ldots, m, \tag{6.1}$$

where $\prod_{j=s}^{n,2}$ means $j = s,\ s+2,\ s+4, \ldots, n$, and $s = mod(i,2)$. The parameter values are determined by using pseudo-random numbers. The a_{ijk} are uniformly distributed random integers in [0,3], and with likelihood lh ($lh = min(100-[\frac{200}{n}],\ 90)$) these values are randomly reset to zero. The parameters c_{ik} and e_i are in [-100,100] and [-10,10], respectively, the initial vector x_0 has elements in [1,2], and the index $l = 8$ is chosen. The solution of this type of problem is not known a priori, and indeed different local solutions may exist. The size of the residuals at the solution is typically large and is determined mainly by the bounds [-10,10] on e_i and can be changed by varying these bounds. We also tested a modified version of the *Signomial* problem where the residual at the solution $(1, \ldots, 1)$ is zero.

The next type of test problem is the *Exponential* problem, defined by

$$r_i(x) = -e_i + \sum_{k=1}^{l} c_{ik} exp(\sum_{j=s}^{n,10} a_{ijk} x_j), \quad i = 1, \ldots, m, \tag{6.2}$$

where $\sum_{j=s}^{n,10}$ means $j = s,\ s+10,\ s+20, \ldots, n$, and $s = mod(i,10)$. The parameters are again determined by using pseudo-random numbers; the a_{ijk} are real numbers in [-0.2,0.3], and with likelihood 0.5 these numbers are reset to zero. The parameters c_{ik} are random numbers in [-5,0], and the index $l = 5$ is chosen. The parameters e_i are determined as follows. A random vector x' with elements in [-1,0] is generated, and

$$e_i(x) = \nu_i + \sum_{k=1}^{l} c_{ik} exp(\sum_{j=s}^{n,10} a_{ijk} x'_j), \quad i = 1, \ldots, m, \tag{6.3}$$

determines e_i, where ν_i is random in [-10,10]. The initial vector x_0 is defined by

$$x_0 = x' + 0.1(x'' - x'), \tag{6.4}$$

where x'' is another random vector with elements in [-1,0]. Again the solution of this type of problem is not known a priori. The size of the residuals at the solution is typically large and is determined mainly by the bounds [-10,10] on ν_i. We also tested a modified version of the Exponential problem where the residual at the solution $(1, \ldots, 1)$ is zero.

The final type of test problem is the *Trigonometric* problem, defined by

$$\tilde{r}_i(x) = -e_i + \sum_{j=s}^{n,4} (a_{ij} sin x_j + b_{ij} cos x_j), \quad i = 1, \ldots, m, \tag{6.5}$$

where $\sum_{j=s}^{n,4}$ means $j = s,\ s+4,\ s+8, \ldots, n$ and $s = mod(i,4)$, and a_{ij}, b_{ij} are random integers in [-100,100]. We determine the coefficients e_i from the equations $\tilde{r}_i(x') = 0$, where x' has random elements in $[-\pi, \pi]$. However, the residuals $\tilde{r}_i(x')$ are zero at x'; hence, to generate a large residual problem, we define

$$r_i(x) = -d_i + \tilde{r}_i(x)^2, \tag{6.6}$$

where d_i is random in [-10,10]. Unlike (6.5), the sum of squares of these residuals is not minimized by x', and so the solution is not known in advance. The starting point x_0 is also generated randomly in $[-\pi,\ \pi]$. The size of the residuals at the solution is typically large and is determined mainly by the bounds [-10,10] on d_i. We also tested a modified version of the Trigonometric problem where the residual at the solution $(1, \ldots, 1)$ is zero.

We then created singular test problems as proposed in Schnabel and Frank [23] by modifying both the zero residual and the nonzero residual cases of the nonsingular test problems described above to the form

$$\hat{F}(x) = F(x) - F'(x_*)A(A^TA)^{-1}A^T(x - x_*), \tag{6.7}$$

where $F(x)$ is the standard nonsingular test function, x_* is its root, and $A \in \Re^{m\times k}$ has full column rank with $1 \leq k \leq n$. Note that $\hat{F}(x)$ also has a root at x_* and rank $(\hat{F}'(x_*)) = n - rank(A)$. We used (6.7) to create two sets of sparse singular problems, with $\hat{F}'(x)$ having rank $n-1$ and $n-2$, respectively, by using the matrix $A \in \Re^{m\times 1}$ and $\Re^{m\times 2}$, respectively, and whose columns are the unit vectors.

All our computations were performed on the Sun SPARC 2 computer of the Computer Science Department of the University of Colorado at Boulder, using double-precision arithmetic. Most of these test problems were run with dimensions $(m, n) = (300, 100)$, and $(m, n) = (600, 200)$. For each test problem we used several different starting guesses generated by

$$\hat{x}_0 = x_0 + const(x_0 - x_*), \tag{6.8}$$

where *const* is a real number indicating how far the initial guess is from the solution, and x_* is the solution resulting from running the problem with the initial starting guess x_0. We increased the iteration limit to 200 and sometimes to 300 when we ran large residual problems. The main reason is that the convergence rate for such problems is linear at best, and usually it takes many iterations to converge to the solution. For any given problem, the same iteration limit was used by both methods.

Tables 1 to 6 summarize the performance of the sparse tensor and Gauss-Newton methods on the test problems described above. Each table presents the test results for a nonsingular test problem and its rank $n-1$ and rank $n-2$ singular versions. Columns "Better" and "Worse" represent the number of times the tensor method was better and worse, respectively, than the Gauss-Newton 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 Gauss-Newton methods required within one iteration of each other. For each set of problems, we summarize the comparative costs of the tensor and Gauss-Newton methods using average ratios of three measures: iterations, execution times, and function evaluations. The average iteration ratio is the total number of iterations required by the tensor method, divided by the total number of iterations required by the Gauss-Newton method on these problems. The same measure is used for the average execution time and function evaluation ratios. Tables 7 to 9 present the average iteration and function evaluation ratios on the sparse nonlinear least squares problems described above. All these 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 Gauss-Newton methods converge to a different solution, or to the same solution but not the singular solution x_* if singular problems are considered. 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 following observations can be made on the basis of Tables 1 to 9. The tensor method almost always outperforms the Gauss-Newton method. On this particular set of test problems, the tensor method improvement over the Gauss-Newton method is about the same for rank n, $n-1$, and $n-2$ problems, in iterations and execution time, and more dramatic in function evaluations for rank $n-1$ problems. Overall, the average improvement of the tensor method over the Gauss-Newton method is about 31% in iterations, about 24% in execution time, and about 32% in function evaluations. We comment on the smaller improvement in execution times than in function evaluations or iterations below.

Of all the test problems, 8 nonsingular problems, 24 rank $n-1$ problems, and 23 rank $n-2$ problems were solved by the tensor method but not by the Gauss-Newton method. On the other hand, only 1 rank $n-1$ problem and no rank n or rank $n-2$ problems were solved by the Gauss-Newton method and not by the tensor method. Most problems solved by the tensor method but not by the Gauss-Newton method have large residuals at the solution; of the problems solved by the tensor method but not the Gauss-Newton method, only 1 nonsingular problem, 1 rank $n-1$ problem, and 4 rank $n-2$ problems have zero residuals at the solution.

We observe from Tables 8 and 9 that the average improvement of the tensor method over the Gauss-Newton method in execution time is about 9% less than that in iterations for zero-residual problems, and about 5% less for large residual problems. This is because a tensor iteration requires at least $2p$ more backsolves than a Gauss-Newton iteration (here, $p = 1$). Empirically, the increased cost per iteration for zero-residual problems ranges from 1% in problems with expensive function evaluation, like the Trigonometric problem, to 23% in problems with inexpensive function evaluation, like the Exponential problem. For large residual problems, the increased cost ranges from 1% in problems with expensive function evaluation to 21% in problems with inexpensive function evaluation. This accounts for the smaller improvements in execution time.

A closer examination of the sparse nonlinear least squares test results shows that the average improvements by the tensor method are slightly more for zero-residual problems than for large residual problems: about 33% in iterations, 24% in execution times, and 36% in function evaluations for zero residual problems, as opposed to 29% in iterations, 23% in execution times, and 28% in function evaluations for large residual problems. On rank $n-1$ problems this is due in part to the tensor method achieving superlinear convergence on zero residual problems, whereas the Gauss Newton method is linearly convergent at best on these problems.

The analysis in [17] shows that tensor methods for nonlinear equations have at least a 3-step order 1.5 rate of convergence on a class of problems with rank deficiency one at the solution, whereas Newton's method is linearly convergent with constant 0.5 on the same problems. These results can be extended to tensor and Gauss-Newton methods for nonlinear least squares on zero residual problems with rank deficiency one. To study these theoretical results experimentally, we examined the sequence of ratios

$$||x^k - x_*|| / ||x^{k-1} - x_*|| \tag{6.9}$$

produced by the Gauss-Newton and tensor methods on problems with $\text{rank}(F'(x_*)) = n-1$ and where the residual at the solution is very small or zero. These ratios for a typical problem are given in Table 10. In almost all cases the Gauss-Newton method exhibits local linear convergence

Table 1: Summary for the Signomial Problem with Zero Residual

		Rank	Tensor			Average Ratio–Tensor/Standard		
m	n	$F'(x_*)$	Better	Worse	Tie	Iteration	Time	Feval
300	100	n	8	0	0	0.63	0.69	0.65
		$n-1$	8	0	0	0.53	0.59	0.54
		$n-2$	8	0	0	0.58	0.63	0.59

Table 2: Summary for the Trigonometric Problem with Zero Residual

		Rank	Tensor			Average Ratio–Tensor/Standard		
m	n	$F'(x_*)$	Better	Worse	Tie	Iteration	Time	Feval
300	100	n	10	0	0	0.52	0.53	0.52
		$n-1$	9	0	0	0.60	0.61	0.60
		$n-2$	6	0	0	0.66	0.68	0.67
600	200	n	8	1	0	0.66	0.67	0.68
		$n-1$	9	0	1	0.69	0.71	0.70
		$n-2$	6	1	0	0.81	0.83	0.82

with constant near 0.5. The local convergence rate of the tensor method is faster, with a typical final ratio of 0.01. This final ratio might be even smaller if analytic Jacobians are used in combination with tighter stopping tolerances.

We also examined the sequence of ratios produced by the Gauss-Newton and tensor methods on problems with $\text{rank}(F'(x_*)) = n-1$ and where the residual at the solution is large. These ratios for a typical problem are given in Table 11. In almost all these cases the Gauss-Newton method is slowly locally q-linearly convergent. The local convergence rate of the tensor method is still linear, as expected, but usually with a smaller linear convergence constant than the Gauss-Newton method.

Table 3: Summary for the Exponential Problem with Zero Residual

		Rank	Tensor			Average Ratio–Tensor/Standard		
m	n	$F'(x_*)$	Better	Worse	Tie	Iteration	Time	Feval
300	100	n	19	0	2	0.69	0.88	0.71
		$n-1$	18	0	3	0.73	0.93	0.76
		$n-2$	19	2	0	0.74	0.92	0.76
600	200	n	19	0	2	0.77	0.95	0.78
		$n-1$	21	1	0	0.76	0.93	0.77
		$n-2$	21	0	0	0.74	0.89	0.75

Table 4: Summary for the Signomial Problem with Large Residual

		Rank	Tensor			Average Ratio–Tensor/Standard		
m	n	$F'(x_*)$	Better	Worse	Tie	Iteration	Time	Feval
300	100	n	4	0	0	0.70	0.71	0.70
		$n-1$	4	0	0	0.72	0.73	0.72
		$n-2$	4	0	0	0.72	0.73	0.72

Table 5: Summary for the Trigonometric Problem with Large Residual

		Rank	Tensor			Average Ratio–Tensor/Standard		
m	n	$F'(x_*)$	Better	Worse	Tie	Iteration	Time	Feval
300	100	n	17	1	0	0.75	0.76	0.75
		$n-1$	9	2	1	0.74	0.76	0.74
		$n-2$	6	0	0	-	-	-
600	200	n	17	1	0	0.73	0.74	0.73
		$n-1$	15	0	0	-	-	-
		$n-2$	12	0	0	0.50	0.50	0.49

We ran most of the test problems using one and two past points in the tensor method, and noticed almost no difference in iterations or function evaluations. However, there was an increase in execution time when we used two past points because of the two extra back solves required per tensor iteration. Thus, our software package for solving sparse nonlinear least squares uses only one past point. This algorithmic decision also decreases the storage requirements of the package.

Table 6: Summary for the Exponential Problem with Zero Residual

		Rank	Tensor			Average Ratio–Tensor/Standard		
m	n	$F'(x_*)$	Better	Worse	Tie	Iteration	Time	Feval
300	100	n	9	0	0	0.70	0.80	0.70
		$n-1$	9	0	0	0.71	0.82	0.72
		$n-2$	10	0	0	0.70	0.80	0.70
600	200	n	7	0	0	0.76	0.85	0.77
		$n-1$	6	1	0	0.76	0.85	0.77
		$n-2$	5	1	0	0.89	1.00	0.90

Table 7: Average Ratios of Tensor Method versus Gauss-Newton Method on All Sparse Nonlinear Least Squares

Rank	Tensor		
$F'(x_*)$	Iterations	Execution Time	Function Evaluations
n	0.69	0.75	0.70
$n-1$	0.69	0.77	0.63
$n-2$	0.70	0.77	0.71

Table 8: Average Ratios of Tensor Method versus Gauss-Newton Method on Sparse Nonlinear Least Squares with Zero Residuals

Rank	Tensor		
$F'(x_*)$	Iterations	Execution Time	Function Evaluations
n	0.65	0.74	0.66
$n-1$	0.66	0.75	0.55
$n-2$	0.70	0.79	0.71

Table 9: Average Ratio of the Tensor Method versus the Gauss-Newton Method on Sparse Nonlinear Least Squares with Large Residuals

Rank	Tensor		
$F'(x_*)$	Iterations	Execution Time	Function Evaluations
n	0.72	0.77	0.73
$n-1$	0.73	0.79	0.73
$n-2$	0.70	0.75	0.70

Table 10: Speed of Convergence on a Typical Zero Residual Problem with Rank $\hat{F}'(x_*) = n - 1$ (Signomial problem, $m = 300,\ n = 100$, as modified by (6.7), started from x_0)

Iteration (k)	Tensor Method	Standard Method
1	0.870	0.870
2	0.825	0.880
3	0.829	0.892
4	0.847	0.900
5	0.801	0.893
6	0.768	0.889
7	0.800	0.892
8	0.788	0.886
9	0.654	0.864
10	0.758	0.849
11	0.590	0.867
12	0.370	0.849
13	0.302	0.849
14	0.483	0.827
15	0.375	0.928
16	0.057	0.885
17	0.020	0.847
18	0.012	0.802
19		0.776
20		0.644
21		0.570
22		0.442
23		0.240
24		0.143
25		0.482
26		0.499
27		0.499
28		0.499
29		0.499
30		0.499
31		0.499
32		0.499
33		0.499
34		0.499
35		0.500
36		0.500

7 Summary and Future Work

We have extended tensor methods to large, sparse nonlinear least squares problems. These methods form the tensor model in the same way as in the tensor methods developed for small to medium-sized dense nonlinear least squares [5, 7], with the exception that only one past point is used in the interpolation process. The tensor step, however, is computed by an entirely new approach that preserves the sparsity of the Jacobian matrix and allows the tensor model to be solved efficiently and stably when the Jacobian matrix is singular. This new solution approach is the main contribution of this paper and is essential because the approach for small dense problems used in [5, 7] destroys the sparsity of the Jacobian matrix as a result of orthogonal transformations of the variable and function spaces.

The numerical test results show that the tensor method is much more efficient than the Gauss-Newton method on both nonsingular and singular test problems, in terms of iterations, function evaluations, and execution times. The tensor method has also proved to be significantly more robust than the Gauss-Newton method in terms of the number of problems solved. The consistency of these improvements indicates that the tensor method is preferable to the Gauss-Newton method for solving sparse nonlinear least squares problems.

We currently are implementing trust region tensor methods for large, sparse nonlinear least squares. The testing of these methods and their comparison with the line search tensor methods described in this paper will be reported in forthcoming paper. Finally, we are implementing the algorithms discussed in this paper in a software package, and we plan to make it generally available in the near future.

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

References

[1] M. Al-Baali and R. Fletcher. Variational methods for non-linear least squares. *J. Oper. Res. Soc.*, 36:405–421, 1985.

[2] M. Arioli, I. S. Duff, and P. P. M. Rijk. On the augmented system approach to sparse least-squares problems. *Numer. Math.*, 55:667–684, 1989.

[3] B. M. Averick, J. J. Moré, C. H. Bischof, A. Carle, and A. Griewank. Computing large sparse Jacobian matrices using automatic differentiation. *SIAM J. Sci. Comput.*, 15:285–294, 1994.

[4] Å. Björck. Iterative refinement of linear least squares solutions. *BIT*, 7:257–278, 1967.

[5] 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.

Table 11: Speed of Convergence on a Typical Large Residual Problem with Rank $\hat{F}'(x_*) = n-1$ (Exponential problem, $m = 300$, $n = 100$, as modified by (6.7), started from x_0)

Iteration (k)	Tensor Method	Standard Method
...	...	...
27	.6792	.7507
28	.6787	.7634
29	.6782	.7903
30	.6782	.8415
31	.6779	.7056
32	.6778	.7506
33	.6775	.7532
34	.6777	.7569
35	.6774	.7631
36	.6772	.7753
37	.6775	.7999
38	.6760	.8006
39	.6754	.7602
40	.6733	.7624
41	.6699	.7650
42	.6730	.7713
43	.6634	.7830
44	.6746	.8078
45	.6634	.8003
46	.6591	.7665
47	.6387	.7651
8	.8334	.7682
49	.7016	.7694
50	.6640	.7776
51	.6797	.7879
52		.8338
53		.7734
54		.7635
55		.7825
56		.7735
57		.7667
58		.8405
59		.8195
60		.7641
61		.7745
62		.7862
63		.7865
64		.8414
65		.7453
66		.9812
67		.7959
68		.8123

[6] A. Bouaricha and R. B. Schnabel. Tensor methods for large, sparse systems of nonlinear equations. Preprint MCS-P473-1094, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[7] A. Bouaricha and R. B. Schnabel. Algorithm 768: TENSOLVE: A software package for solving systems of nonlinear equations and nonlinear least squares problems using tensor methods. *ACM Trans. Math. Softw.*, 23:174–195, 1997.

[8] J. R. Bunch. Partial pivoting strategies for symmetric matrices. *SIAM J. Numer. Anal.*, 11:521–528, 1974.

[9] 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.

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

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

[12] A. M. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *Inst. Math. Applics.*, 13:117–120, 1974.

[13] J. E. Dennis, D. M. Gay, and R. E. Welsch. An adaptive nonlinear least squares algorithm. *ACM Trans. Math. Softw.*, 7:348–368, 1981.

[14] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, Englewood Cliffs, N.J., 1983.

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

[16] I. S. Duff and J. K. Reid. A comparison of some methods for the solution of sparse overdetermined system. *J. Inst. Math. Applics*, 17:267–280, 1975.

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

[18] W. M. Gentleman. Least squares computation by givens transformations without squares roots. *J. Inst. Math. Applics.*, 12:329–336, 1973.

[19] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix. Anal. Appl.*, 13:333–356, 1992.

[20] G. H. Golub. Numerical methods for solving linear least squares problems. *Numer. Math.*, 7:206–216, 1965.

[21] G. D. Hachtel. Extended applications of the sparse tableau approach-finite elements and least squares. In W. R. Spillers, editor, *Basic questions of design theory.* North Holland Publishing Company, Amsterdam, 1974.

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

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

[24] 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.