STENMIN: A Software Package for Large, Sparse Unconstrained Optimization Using Tensor Methods*

Ali Bouaricha†

Argonne National Laboratory

We describe a new package for minimizing an unconstrained nonlinear function where the Hessian is large and sparse. The software allows the user to select between a tensor method and a standard method based upon a quadratic model. The tensor method models the objective function by a fourth–order model, where the third– and fourth–order terms are chosen such that the extra cost of forming and solving the model is small. The new contribution of this package consists of the incorporation of an entirely new way of minimizing the tensor model that makes it suitable for solving large, sparse optimization problems efficiently. The test results indicate that, in general, the tensor method is often more efficient and more reliable than the standard Newton method for solving large, sparse unconstrained optimization problems.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra–*sparse and very large systems*; G.1.6 [**Numerical Analysis**]: Optimization–*unconstrained optimization*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms

Additional Key Words and Phrases: tensor methods, sparse problems, large-scale optimization, rank-deficient matrices

---

*Part of this work was performed while the author was research associate at CERFACS (Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France).

†Author's address: Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 60439. bouarich@@mcs.anl.gov. This work was supported in part by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

1. Introduction

This paper describes a software package for solving the unconstrained optimization problem

$$\text{given } f : \Re^n \to \Re, \text{ find } x_* \in \Re^n \text{ such that } f(x_*) \leq f(x) \text{ for all } x \in D, \tag{1.1}$$

using tensor methods, where D is some open set containing x_*. We assume that f is at least twice continuously differentiable and $\nabla^2 f(x_c)$ is large and sparse.

Tensor methods for unconstrained optimization are general–purpose methods primarily intended to improve upon the performance of standard methods especially on problems where $\nabla^2 f(x_*)$ has a small rank deficiency, and to be at least as efficient as standard methods on problems where $\nabla^2 f(x_*)$ is nonsingular. Tensor methods for unconstrained optimization base each iteration upon the fourth–order model of the objective function $f(x)$

$$M_T(x_c + d) = f(x_c) + \nabla f(x_c) \cdot d + \frac{1}{2}\nabla^2 f(x_c) \cdot d^2 + \frac{1}{6}T_c \cdot d^3 + \frac{1}{24}V_c \cdot d^4, \tag{1.2}$$

where $d \in \Re^n$, x_c is the current iterate, $\nabla f(x_c)$ and $\nabla^2 f(x_c)$ are the first and second analytic derivatives of f at x_c, or finite difference approximations to them, and the tensor terms at x_c, $T_c \in \Re^{n \times n \times n}$ and $V_c \in \Re^{n \times n \times n \times n}$, are symmetric. (We use the notation $\nabla f(x_c) \cdot d$ for $\nabla f(x_c)^T d$, and $\nabla^2 f(x_c) \cdot d^2$ for $d^T \nabla^2 f(x_c) d$ to be consistent with the tensor notation $T_c \cdot d^3$ and $V_c \cdot d^4$. We abbreviate terms of the form dd, ddd, and $dddd$ by d^2, d^3, and d^4, respectively.)

Schnabel and Chow [10] select T_c and V_c such that the model interpolates function and gradient values from p past iterates, where p is a small number. This strategy results in T_c and V_c being low-rank tensors, which is crucial for the efficiency of the tensor method. Here, we consider only the case where the tensor model interpolates $f(x)$ and $\nabla f(x)$ at the previous iterate (i.e., $p = 1$). The reasons for this choice are that the performance of the tensor version that allows $p \geq 1$ is similar overall to that constraining p to be 1, and that the method is simpler and less expensive to implement in this case.

The above choice of T_c and V_c yields the tensor model

$$M_T(x_c + d) = f(x_c) + \nabla f(x_c) \cdot d + \frac{1}{2}\nabla^2 f(x_c) \cdot d^2 + \frac{1}{2}(b^T d)(s^T d)^2 + \frac{\gamma}{24}(s^T d)^4, \tag{1.3}$$

where $s \in \Re^n$ is the step from x_c to the previous iterate x_{-1} (i.e., $s = x_{-1} - x_c$) and $b \in \Re^n$ and $\gamma \in \Re$ are uniquely determined by the requirements $M_T(x_{-1}) = f(x_{-1})$ and $\nabla M_T(x_{-1}) = \nabla f(x_{-1})$. The whole process of forming the tensor model requires only $O(n^2)$ arithmetic operations. The storage needed for forming and storing the tensor model is only a total of $6n$.

The tensor algorithms described in [10] are QR-based algorithms involving orthogonal transformations of the variable space. These algorithms are effective for minimizing the tensor model when the Hessian is dense because they are stable numerically, especially when the Hessian is singular. They are not efficient for sparse problems, however, because they destroy the sparsity of the Hessian due to the orthogonal transformation of the variable space. To preserve the sparsity of the Hessian, we developed in [4] an entirely new way of minimizing the tensor model that employs a sparse variant of the Cholesky decomposition. This makes the new algorithms well suited for sparse problems. In this new approach, we show that the minimization of (1.3) can be reduced to the solution of a third–order polynomial in one unknown, plus the solution of three systems of linear equations that all involve the same coefficient matrix $\nabla^2 f(x_c)$. The `STENMIN` package is essentially based on this new approach.

The remainder of this paper is organized as follows. In §2 an iteration of tensor methods for large, sparse unconstrained optimization is outlined. In §3 we give an overview of the input, output, and important options provided by the software package. Finally, in §4 we describe comparative testing for an implementation based on the tensor method versus an implementation based on the Newton method, and we present summary statistics of the test results.

2. An Iteration of Tensor Methods

In this section, we present the overall algorithm for tensor methods for large, sparse unconstrained optimization. Algorithm 2.1 is a slightly modified version of the algorithm described in [4] in the way the tensor step is selected when the β equation (see algorithm below) has more than one root. In general, this new way of computing the tensor step appears to perform better than the strategy described in [4], in both function evaluations and execution times. A summary of the experimental results for this implementation is presented in §4.

Algorithm 2.1. An Iteration of Tensor Methods for Large, Sparse Unconstrained Optimization

Let x_c be the current iterate, x_+ the next iterate, d_t the tensor step, and d_n the Newton step.

1. Calculate $\nabla f(x_c)$, and decide whether to stop. If not:
2. Calculate $\nabla^2 f(x_c)$
3. Calculate b and γ in the tensor model (1.3), so that the tensor model interpolates $f(x)$ and $\nabla f(x)$ at x_{-1}
4. Find a potential minimizer d_t of the tensor model
Factorize $\nabla^2 f(x_c)$ using the MA27 package [8]
if $\nabla^2 f(x_c)$ has full rank **then**
4.1. Form the β equation ($\beta \in \Re$): $-u + (yw - uv - 1)\beta - \frac{3}{2}v\beta^2 + (\frac{1}{2}wz - \frac{\gamma}{6}w - \frac{1}{2}v^2)\beta^3$,
where $u = s^T\nabla^2 f(x_c)^{-1}\nabla f(x_c)$, $v = s^T\nabla^2 f(x_c)^{-1}b$, $w = s^T\nabla^2 f(x_c)^{-1}s$,
$y = b^T\nabla^2 f(x_c)^{-1}\nabla f(x_c)$, and $z = b^T\nabla^2 f(x_c)^{-1}b$
4.2. Calculate the tensor step:
$d_t = -\nabla^2 f(x_c)^{-1}(\nabla f(x_c) + \theta_*\beta_* s + \frac{1}{2}\beta_*^2 b + \frac{\gamma}{6}\beta_*^3 s)$,
where $\beta_* = \min(|\beta_i|)$ with β_i being the roots of the β equation, and
$\theta_* = -\dfrac{(u + \beta_* + \frac{1}{2}v\beta_*^2 + \frac{\gamma}{6}w\beta_*^3)}{w\beta_*}$
elseif $\nabla^2 f(x_c)$ is singular with $\operatorname{rank}(\nabla^2 f(x_c)) = n-1$ **then**
4.3. Form the β equation ($\beta \in \Re$): $u + (1 + \hat{\beta}v)\beta + (\frac{1}{2}v + \frac{\gamma}{2}w\hat{\beta})\beta^2 + \frac{\gamma}{6}w\beta^3$,
where $u = s^T\hat{\nabla}^2 f(x_c)^{-1}\hat{\nabla} f(x_c)$, $\hat{\nabla}^2 f(x_c) = \nabla^2 f(x_c) + ss^T$,
$\hat{\nabla} f(x_c) = \nabla f(x_c) + \nabla^2 f(x_c)\hat{d} + \hat{\theta}\hat{\beta}s + \frac{1}{2}\hat{\beta}^2 b + \frac{\gamma}{6}\hat{\beta}^3 s$, $\hat{\beta} = s^T\hat{d}$, $\hat{\theta} = b^T\hat{d}$,
$\hat{d}$ is the global step computed in the previous iteration, $v = s^T\hat{\nabla}^2 f(x_c)^{-1}b$, and
$w = s^T\hat{\nabla}^2 f(x_c)^{-1}s$.
4.4. Calculate the tensor step of the transformed tensor model (2.1) below:

$\delta = -\hat{\nabla}^2 f(x_c)^{-1}(\hat{\nabla} f(x_c) + \hat{\beta}\beta_* b + \hat{\beta}\theta_* s + \beta_*\theta_* s + (\frac{1}{2}b + \frac{\gamma}{2}\hat{\beta}s)\beta_*^2 + \frac{\gamma}{6}\beta_*^3 s)$
where $\beta_* = \min(|\beta_i|)$ with β_i being the roots of the β equation
and $\theta_* = \frac{1}{w(\hat{\beta}+\beta_*)}(yw\hat{\beta} - u - uv\hat{\beta} + (yw + zw\hat{\beta}^2 - 2v\hat{\beta} - v^2\hat{\beta}^2 - uv - 1)\beta_*$
$+(\frac{3}{2}zw\hat{\beta} - \frac{\gamma}{2}w\hat{\beta} - \frac{3}{2}v - \frac{3}{2}v^2\hat{\beta}) + \frac{1}{2}zw - \frac{\gamma}{6}w - \frac{v^2}{2})\beta_*^3)$,
where $y = b^T\hat{\nabla}^2 f(x_c)^{-1}\hat{\nabla} f(x_c)$, and $z = b^T\hat{\nabla}^2 f(x_c)^{-1}b$

4.5. Calculate the tensor step of the original model (1.3):
$d_t = \delta + \hat{d}$

else$\{rank(\nabla^2 f(x_c)) < n-1\}$

4.6. Modify the eigencomponents of $\nabla^2 f(x_c)$

4.7. Perform steps 4.1–4.2

endif

5. Test whether the tensor step is descent. If it is not compute the Newton step

if $\nabla^T f(x_c)d_t > 0$ **then**

5.1. Compute the Newton step

if $rank(\nabla^2 f(x_c)) < n-1$ **then**
$d_n = \nabla_m^2 f(x_c)^{-1}\nabla f(x_c)$, where $\nabla_m^2 f(x_c)$ is $\nabla^2 f(x_c)$ with the eigencomponents modified, (d_n is obtained for free)

else
Modify the eigencomponents of $\nabla^2 f(x_c)$
if all the eigencomponents of $\nabla^2 f(x_c)$ remain unchanged $\{\nabla^2 f(x_c)$ is already positive definite$\}$ **then**
$d_n = \nabla^2 f(x_c)^{-1}\nabla f(x_c)$, ($d_n$ is obtained for free)
else
$d_n = \nabla_m^2 f(x_c)^{-1}\nabla f(x_c)$
endif

endif

endif

6. Compute an acceptable next iterate x_+ using a line search global strategy

7. $x_c = x_+$, $f(x_c) = f(x_+)$, **go to** step 1

Algorithm 2.2. Line Search Strategy for Large, Sparse Unconstrained Optimization

Let x_c, d_t, and d_n be defined as is Algorithm 2.1.

if d_t is descent **then**
$x_+^t = x_c + d_t$
if $f(x_+^t) < f(x_c) + 10^{-4}\cdot\nabla f(x_c)d_t$ **then**
$x_+ = x_+^t$
else
Find an acceptable x_+^n in the Newton direction d_n
using the line search given by Algorithm A6.3.1 [9, p.325]
Find an acceptable x_+^t in the tensor direction d_t
using the line search given by Algorithm A6.3.1 [9, p.325]
if $f(x_+^n) < f(x_+^t)$ **then**

$x_+ = x_+^n$

else

$x_+ = x_+^t$

endif

endif

else

Find an acceptable x_+^n in the Newton direction d_n
using the line search given by Algorithm A6.3.1 [9, p.325]

$x_+ = x_+^n$

endif

In step 1, the gradient is either computed analytically or approximated by the algorithm A5.6.3 given in Dennis and Schnabel [7]. In step 2, the Hessian matrix is either calculated analytically or approximated by a graph coloring algorithm described in [6]. In step 4.3, the matrix $\hat{\nabla}^2 f(x_c)$ is factorized using the augmented system approach described in [4]. In steps 4.4 and 4.5, we first compute the tensor step δ of the transformed model (obtained by substituting $\hat{d} + \delta$ for d in (1.3), where $\hat{d}$ is the global step computed in the previous iteration)

$$\begin{aligned} M_T(x_c + d) \quad = \quad & f(x_c) \ + \ \nabla f(x_c) \cdot \hat{d} \ + \ \frac{1}{2}\nabla^2 f(x_c) \cdot \hat{d}^2 \ + \ \frac{1}{2}(b^T\hat{d})(s^T\hat{d})^2 \\ & + \ \frac{\gamma}{24}(s^T\hat{d})^4 \ + \ (\nabla f(x_c) \ + \ \nabla^2 f(x_c)\hat{d} \ + \ (b^T\hat{d})(s^T\hat{d})s \\ & + \ \frac{1}{2}(s^T\hat{d})^2 b \ + \ \frac{\gamma}{24}(s^T\hat{d})^3 s) \cdot \delta \ + \ \frac{1}{2}(\nabla^2 f(x_c) \\ & + \ (b^T\hat{d} \ + \ \frac{\gamma}{2} ss^T) \cdot \delta^2 \ + \ (s^T\hat{d})(b^T\delta)(s^T\delta) \ + \ \frac{1}{2}(b^T\delta)(s^T\delta)^2 \\ & + \ \frac{\gamma}{6}(s^T\hat{d})(s^T\delta)^3 \ + \ \frac{\gamma}{24}(s^T\delta)^4. \end{aligned} \tag{2.1}$$

Then we set the tensor step d_t of the original tensor model (1.3) to $\delta + \hat{d}$. In step 4.6, we obtain a perturbation μ such as $\nabla^2 f(x_c) + \mu I$ is safely positive definite by using the Gill, Murray, Ponceleon, and Saunders method [9]. After we compute the $\mathrm{LDL^T}$ of the Hessian matrix using the MA27 package [8], we change the block diagonal matrix D to $\mathrm{D + E}$. The modified matrix is block diagonal positive definite. This guarantees that the decomposition $\mathrm{L(D + E)L^T}$ is sufficiently positive definite. Note that the Hessian matrix is not modified if it is already positive definite. In step 5, we test whether or not the tensor step is descent. If it is not, then we compute the Newton step d_n as a by–product of the minimization of the tensor model. That is, if $rank(\nabla^2 f(x_c)) < n - 1$ or all the eigencomponents of D turn out to be positive, i.e., $\nabla^2 f(x_c)$ is positive definite, then the Newton step is obtained for free; otherwise we perform another solve after we have modified the eigencomponents of D. Thus, d_n is the modified Newton step $(\nabla^2 f(x_c) + \mu I)^{-1}\nabla f(x_c)$, where $\mu = 0$ if $\nabla^2 f(x_c)$ is safely positive definite, and $\mu > 0$ otherwise. In step 6, we compute a next iterate x_+ by performing the standard backtracking line search global strategy described in Algorithm 2.2. The line search tensor method is much simpler to implement and to understand than the two-dimensional trust region tensor method introduced in [4], and is appreciably faster. For these reasons, this software uses a line search method. The global framework for the line search method we used in conjunction with our tensor method for large, sparse unconstrained optimization is similar to the one used for systems of nonlinear equations [3, 5]. This strategy has proved successful for large, sparse systems of nonlinear equations. This approach always tries the full tensor step first. If this provides enough decrease in the objective function, then we terminate; otherwise we find acceptable next iterates in both the Newton and tensor directions and select the one with the lower function value as the next iterate. Algorithm 2.1 terminates successfully if the scaled gradient is less than GRADTL $= \epsilon^{\frac{1}{3}}$, where ϵ is the machine precision. It terminates unsuccessfully if the iteration limit of 500 is exceeded or five consecutive steps of length STEPMX $= \max\{||x_0||_2 \cdot 10^3, 10^3\}$ have been taken, where x_0 is the initial approximation of the solution. If the last global step fails to locate a point lower than the current iterate in the line search strategy or the length of the last step is less than STEPTL $= \epsilon^{\frac{2}{3}}$, the algorithm stops and reports this condition; this may indicate either success or failure.

3. Overview of the Software Package

The required input to the software is the number of variables N, the function FCN that computes $f(x)$, an initial guess X0, the number of nonzeros NZ stored in the lower or upper triangular part of the Hessian matrix, and the row and column indices IRN and ICN of these nonzeros given in any order.

Two methods of calling the package are provided. In the short version, the user supplies only the above information, and default values of all other options are used. These include the calculation of the gradient and Hessian matrix by finite differences, and the use of the tensor rather than the standard Newton method. In the other method for calling the package, the user may override any default values of the package options.

The user has the option to choose between the tensor method and the standard Newton method. If the flag METHOD is set to 0, the package will use the standard Newton method. The tensor method is used otherwise.

Upon completion, the program returns with an approximation XPLS to the minimizer x_*, the value of the objective function FPLS at XPLS, the value of the gradient GPLS(XPLS), the Hessian HESS(XPLS), and a flag specifying under which stopping condition the algorithm has terminated.

The software package is coded so that if the user inputs the typical magnitude $\mathtt{TYPX}_i$ of each component of x, the performance of the package is the equivalent to what would result from redefining the independent variable x with

$$x_{\text{scaled}} = \begin{bmatrix} 1/\mathtt{TYPX}_1 & & \\ & \ddots & \\ & & 1/\mathtt{TYPX}_n \end{bmatrix} \cdot x \qquad (3.1)$$

and then running the package without scaling. The default value of each $\mathtt{TYPX}_i$ is 1. Scaling is often important to use for problems in which the variable components are widely different in magnitudes.

The user may supply analytic routines for the gradient and/or the Hessian. If they are not supplied the package computes them by finite differences. The parameters `GRDFLG` and `HSNFLG` specify whether analytic gradient and Hessian have been provided, respectively. When the analytic gradient and/or Hessian are supplied, the user has the option of checking the supplied analytic routines against the package's finite difference routines.

The standard (default) output from this package consists of printing the input parameters and the final results. The printed input parameters are those used by the algorithm and hence include any corrections made by the program module `OPTCHK`, which examines the input specifications for illegal entries and consistency. The program will provide an error message if it terminates as a result of input errors. The printed results include a message indicating the reason for termination, an approximation `XPLS` to the solution x_*, the function value at `XPLS`, and the gradient vector `GPLS`. The package provides an additional means for the control of output via the variable `MSG`. The standard output is the input state, the final results, and the stopping conditions. The user may suppress all output or may print the intermediate iteration results in addition to the standard output.

If the user sets the variable `INFORM` to 1, then the package uses reverse communication to obtain the multiplication of the Hessian matrix at the current iterate by a given vector. If `INFORM` is set to 0, then this quantity is computed by the subroutine `STHMUV` provided by the package.

4. Test Results

We tested our tensor and Newton methods on the set of unconstrained optimization problems from the `CUTE` [2] and the `MINPACK-2` [1] collections. Most of these problems have nonsingular Hessians at the solution. We also created singular test problems as proposed in [3, 11] by modifying the nonsingular test problems from the `CUTE` collection. The dimensions of these problems range from 100 to 10000. All our computations were performed on a Sun SPARC 10 Model 40 machine using double–precision arithmetic.

A summary for the test problems whose Hessians at the solution have ranks n, $n-1$, and $n-2$ is presented in Table 4.1. The descriptions of the test problems and the detailed results are given in [4]. In Table 4.1 the 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 gradient evaluation. The "tie" column represents the number of times the tensor and Newton methods required within one gradient evaluation of each other. For each set of problems, we summarize the comparative costs of the tensor and Newton methods using average ratios of three measures: gradient evaluations, function evaluations, and execution times. The average gradient evaluation ratio (geval) is the total number of gradient evaluations required by all the tensor runs, divided by the total number of gradient evaluations required by all the Newton runs on these problems. The same measure is used for the average function evaluation (feval) and execution time (time) ratios. These average ratios include only problems where both methods converge to the same minimizer. On the other hand, the statistics for the "better," "worse," and "tie" columns also include the cases where only one of the two methods converges. Moreover,

Table 4.1: Summary of the `CUTE` and `MINPACK-2` test problems using line search

Rank	Tensor/Standard			Pbs Solved		Average Ratio–Tensor/Standard		
$\nabla^2 f(x_*)$	better	tie	worse	t/s	s/t	feval	geval	time
n	54	38	4	4	0	0.98	0.68	0.63
$n-1$	18	2	0	5	0	0.51	0.48	0.40
$n-2$	18	1	1	7	0	0.66	0.63	0.62

we excluded from all statistics problems requiring a number of gradient evaluations less or equal than three by both methods. Finally, columns "t/s" and "s/t" show the number of problems solved by the tensor method but not by the Newton method and the number of problems solved by the Newton method but not by the tensor method, respectively.

The improvement by the tensor method over the Newton method on problems with rank $n-1$ is dramatic, averaging 49% in function evaluations, 52% in gradient evaluations, and 60% in execution times. This is due in part to the rate of convergence of the tensor method being faster than that of Newton's method, which is known to be only linearly convergent with constant $\frac{2}{3}$. A typical convergence rate of the tensor method on rank $n-1$ problems is around 0.01. Whether this is a superlinear convergence remains to be proved. On problems with rank $n-2$, the improvement by the tensor method over the Newton method is also substantial, averaging 34% in function evaluations, 37% in gradient evaluations, and 38% in execution times. In the test results obtained for the nonsingular problems, the tensor method is only 2% better than the Newton method in function evaluations, but 32% and 37% better in gradient evaluations and in execution times, respectively. The tensor method requires more function evaluations than the Newton method on some nonsingular problems. This is because the full tensor step does not provide sufficient decrease in the objective function, and therefore the tensor method has to perform a line search method in both the Newton and tensor directions, which causes the number of function evaluations required by the tensor method to be inflated.

The tensor method solved a total of four nonsingular problems, five rank $n-1$ problems, and seven rank $n-2$ problems, that Newton's method failed to solve. The reverse never occurred. This clearly indicates that the tensor method is most likely to be more robust than Newton's method.

The overall results presented in this paper show that the tensor method is often more efficient and more reliable than the standard Newton method in solving large, sparse unconstrained optimization problems. Furthermore, the tensor method is likely to solve a wider range of problems. In order to firmly establish the conclusion above, additional testing is required.

Acknowledgments. I am grateful to Nick Gould for his assistance and encouragements. I also thank my CERFACS colleague Jacko Koster for reviewing this paper and Gail Pieper from the MCS division at Argonne National Laboratory for her suggestions for improvement.

References

[1] B. M. Averick, R. G. Carter, J. J. Moré, and G. L. Xue. The MINPACK-2 test problem collection. Technical Report ANL/MCS-P153-0692, Argonne National Laboratory, 1992.

[2] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Trans. Math. Software*, 21(1):123–160, 1995.

[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. Tensor methods for large, sparse unconstrained optimization. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, preprint MCS-P452-0794, 1994.

[5] A. Bouaricha and R. B. Schnabel. TENSOLVE: A software package for solving systems of nonlinear equations and nonlinear least squares problems using tensor methods. Preprint MCS-P463-0894, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[6] T. F. Coleman and J. J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Programming*, 28:243–270, 1984.

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

[8] I. S. Duff and J. K. Reid. MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report R-10533, AERE Harwell Laboratory, Harwell, UK, 1983.

[9] P. E. Gill, W. Murray, D. B. Ponceleon, and M. A. Saunders. Preconditioners for indefinite systems arising in optimization and nonlinear least squares problems. Technical Report SOL 90-8, Department of Operations Research, Stanford University, California, 1990.

[10] R. B. Schnabel and T. Chow. Tensor methods for unconstrained optimization using second derivatives. *SIAM J. Optimization*, 1:293–315, 1991.

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