Algorithm 739: A Software Package for Unconstrained Optimization Using Tensor Methods

T. CHOW, E. ESKOW, and R. SCHNABEL
University of Colorado at Boulder

This paper describes a software package for finding the unconstrained minimizer of a nonlinear function of n variables. The package is intended for problems where n is not too large—say, $n < 100$—so that the cost of storing one $n \times n$ matrix, and factoring it at each iteration, is acceptable. The software allows the user to choose between a recently developed "tensor method" for unconstrained optimization and an analogous standard method based on a quadratic model. The tensor method bases each iteration on a specially constructed fourth-order model of the objective function not significantly more expensive to form, store, or solve than the standard quadratic model. In our experience, the tensor method requires significantly fewer iterations and function evaluations to solve most unconstrained optimization problems than standard methods based on quadratic models, and also solves a somewhat wider range of problems. For these reasons, it may be a useful addition to numerical software libraries.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization—*Gradient methods*; G.4 [**Mathematics of Computing**]: Mathematical Software—*efficiency*; *reliability and robustness*

General Terms: Algorithms

Additional Key Words and Phrases: Higher-order model, tensor method, unconstrained optimization

1. INTRODUCTION

This paper describes a software package that implements the tensor method for unconstrained optimization introduced in Schnabel and Chow [1989]. The package solves the unconstrained minimization problem

$$\min_{x \in R^n} f(x) : R^n \to R$$

by utilizing either analytic or finite-difference gradients and Hessian matrices at each iteration. The software allows the user to choose between a tensor method and an analogous standard method based on a quadratic model. It is

Authors' address: Department of Computer Science, Campus Box 430, University of Colorado at Boulder, Boulder, CO 80309-0430

intended for problems where the number of variables n is not too large—say, $n \leq 100$—so that the cost of storing one $n \times n$ matrix and factoring it at each iteration is acceptable. It can be applied to problems where $n = 1$, but software designed specifically for one-variable problems is likely to be preferable.

The tensor method employed in the software bases each iteration on a specially constructed fourth-order model of the objective function. This model interpolates the function value and gradient from the previous iterate, as well as the current function value, gradient, and Hessian matrix. The model is constructed so that the storage it requires, and the arithmetic operations per iteration required to form and solve it, is not significantly higher than for a standard second-derivative method based on a quadratic model. In our experience, the tensor method requires significantly fewer iterations and function evaluations to solve most unconstrained optimization problems than the standard method based on a quadratic model provided by the software, and also solves a somewhat wider range of problems.

The next section gives a very brief overview of the tensor method; for further information, see Schnabel and Chow [1989]. Section 3 gives an overview of the input, output, and important options provided by the software. Section 4 describes the user interface to the package, which includes both a simplified (default) and a longer calling sequence. Section 5 describes in detail all the possible input and output parameters for the package and their effects. Section 6 mentions a few implementation details. And Section 7 summarizes our experience in using this package, which can be found in more detail in Chow [1989].

2. BRIEF OVERVIEW OF THE TENSOR METHOD

Each iteration of the tensor method is based on a fourth-order model of the objective function $f(x)$ around the current iterate x_c that has the form

$$m(x_c + d) = f(x_c) + \nabla f(x_c)^T d + \tfrac{1}{2} d^T \nabla^2 f(x_c) d + \tfrac{1}{6} T_c d^3 + \tfrac{1}{24} V_c d^4. \quad (2.1)$$

The tensor terms T_c and V_c are three- and four-dimensional objects, respectively, chosen so that the model interpolates function and gradient information from previous iterates. Schnabel and Chow [1989] allowed the model to interpolate function and gradient values from more than one past iterate, but their tests showed that there was no advantage to this method in comparison to a tensor method that interpolates only the function value and gradient from the most recent past iterate. Additionally, the tensor method is significantly simpler when only information from one past iterate is used. For these reasons, this software package uses the version of the tensor method that only interpolates information from the most recent previous iterate.

Schnabel and Chow [1989] derive a procedure for determining the smallest T_c and V_c, in the Frobenius norm, that allows (2.1) to interpolate previous function and gradient values. In the case where only one past iterate is used,

they show that these T_c and V_c are rank-two and rank-one tensors, respectively. In particular, the tensor model has the form

$$m(x_c + d) = f(x_c) + \nabla f(x_c)^T d + \frac{1}{2} d^T \nabla^2 f(x_c) d + \frac{1}{6}(s_c^T d)^2 (b_c^T d) + \frac{\alpha}{24}(s_c^T d)^4, \tag{2.2}$$

where s_c is the step from x_c to the previous iterate x_p, and $b_c \in R^n$ and $\alpha \in R$ are uniquely determined by the requirements $m(x_p) = f(x_p)$ and $\nabla m(x_p) = \nabla f(x_p)$. The cost of finding b_c and α is n^2 (for the calculation $s_c^T \nabla^2 f(x_c) s_c$) + $O(n)$ multiplications and additions, which is very small in comparison to the basic $O(n^3)$ per iteration cost of the standard method. The additional storage required is just a few n-vectors for b_c, s_c, and some intermediate quantities, which is very small compared to the $n^2/2$ storage for $\nabla^2 f(x_c)$.

Near a minimizer, the step taken by the tensor method is to the minimizer of (2.2). Schnabel and Chow [1989] show that the problem of minimizing the fourth-order model (2.2) can be reduced to the problem of minimizing a fourth-order polynomial in one variable, plus the minimization of a quadratic function in $n - 1$ variables. The cost of this process is only $4n^2 + O(n)$ multiplications and additions more than the basic $O(n^3)$ of minimizing a quadratic model.

As in all nonlinear optimization methods, it is also necessary to incorporate a procedure that allows the method to converge from starting points that are far from the solution. Chow [1989] implemented both a line search method and a trust region method for the tensor method. In his tests, the tensor method with a trust region strategy was about 15% more efficient (in terms of iterations or function and derivative evaluations) than the tensor method with a line search strategy, while the difference in robustness between the two methods was indistinguishable. The line search tensor method was still about 25–35% more efficient than either a standard line search or trust region method based on the quadratic model. Additionally, the line search tensor method is much simpler to implement and to understand than the trust region tensor method, and is appreciably faster on small, inexpensive problems where the complexity of the code becomes the dominant cost. For these reasons, this software uses a line search method.

In the line search tensor method, if the tensor model has a minimizer and is in a descent direction from x_c, then we search along the direction to the minimizer, using the backtracking line search algorithm A6.3.1 from Dennis and Schnabel [1983]. This line search starts with step length one, and if the candidate point x_+ does not satisfy the condition

$$f(x_+) \leq f(x_c) + 10^{-4} \nabla f(x_c)^T (x_+ - x_c), \tag{2.3}$$

it reduces this step length by some factor between 0.1 and 0.5, and continues to do this until the first time that the candidate point satisfies (2.3). If we cannot use the tensor model because it does not have a minimizer in a descent direction from x_c, or we are at the first iteration and there is no

previous iterate, we base the step on the standard quadratic model. We use the same line search, and calculate the search direction using a variation on the modified Cholesky factorization of Schnabel and Eskow [1990]. The search direction is $-(\nabla^2 f(x_c) + ||D||I)^{-1}\nabla f(x_c)$, where D is the diagonal matrix added to the Hessian in the modified Cholesky decomposition, which results in $D = 0$ if $\nabla^2 f(x_c)$ is safely positive definite.

When we can use the tensor model to find a potential next iterate, it turns out that we can find the search direction based on the quadratic model at little additional cost. For this reason, we always also calculate the potential next iterate based on the quadratic model, and then choose the point with the lower function value as our next iterate. This strategy generally only costs one or two additional function evaluations (and no extra derivative evaluations) per iteration, and has been found to improve appreciably the efficiency of the algorithm.

A high-level description of the tensor method is contained in Figure 1.

3. OVERVIEW OF THE SOFTWARE PACKAGE

The required input to the software is the number of variables n (N), a subroutine to evaluate the function $f(x)$ (FCN), an initial approximation x_0 to the solution $x*$ (X), and the row dimension of the matrix in the user's program that will contain the Hessian matrix (NR). The user may also provide subroutines to evaluate $\nabla f(x)$ and $\nabla^2 f(x)$, but these subroutines are optional. If they are not provided, the gradient and Hessian are approximated by finite differences. If subroutines to calculate analytic derivatives are provided, they are checked at the start of the algorithm against a finite-difference approximation to detect possible coding errors, unless the user chooses to disable this option (see GRDFLG, HESFLG, Section 5).

Upon completion, the software produces its final iterate x_f (XPLS), the value of the function $f(x_f)$ (FPLS), the gradient $g(x_f)$ (GPLS), the Hessian $H(x_f)$ (H), and a flag specifying under which stopping condition the algorithm was terminated (MSG). The stopping criteria used in this software are the same as in the UNCMIN package of Schnabel et al. [1985]. Informally, they are: (1) $\nabla f(x_+) \approx 0$; (2) $x_+ \approx x_c$; (3) the package could not satisfy (2.3) at the last iteration; (4) the iteration limit was exceeded; and (5) divergence is suspected. If any of these conditions is satisfied the algorithm terminates. In our experience, when the code stops due to $\nabla f(x_+) \approx 0$, it is almost always near a local minimizer. When it stops because $x_+ \approx x_c$ it is usually near a solution; this tolerance should be set quite small, however, since optimization algorithms sometimes take very small steps while still far from the solution. When the algorithm stops because the last iteration could not satisfy (2.3), it is often near a solution and cannot achieve further accuracy due to finite-precision arithmetic; this can be assessed by checking the size of the gradient. The divergence test is meant to detect functions that are unbounded below; a very large maximum step size is imposed in the line search, and if five consecutive steps of this size are taken, divergence is suspected. A more

Given current iterate x_c, $f(x_c)$, previous iterate x_p, $f(x_p)$, $\nabla f(x_p)$:

1. Calculate $\nabla f(x_c)$ and decide whether to stop. If not:
2. Calculate $\nabla^2 f(x_c)$.
3. Set $s_c = x_c - x_p$, and calculate b_c and α in the tensor model (2.2) so the tensor model interpolates $f(x_p)$ and $\nabla f(x_p)$.
4. Calculate the the minimizer of the tensor model; if it has one, and the minimizer is in a descent direction d_T from x_c, then use the line search to calculate a potential acceptable next iterate $x_{+T} = x_c + \lambda_T d_T$.
5. Calculate the search direction d_{+N} based upon the quadratic model, as described above ($d_{+N} = -(\nabla^2 f(x_c) + \| D \| I)^{-1} \nabla f(x_c)$, $D \geq 0$), and use the line search to calculate a potential acceptable next iterate $x_{+N} = x_c + \lambda_N d_N$.
6. If (the tensor line search was conducted) and ($f(x_{+T}) \leq f(x_{+N})$)
   then set $x_+ = x_{+T}$
   else set $x_+ = x_{+N}$.
7. Set $x_p = x_c$, $x_c = x_+$.

Fig. 1. An iteration of the tensor method algorithm.

precise definition of these criteria and recommended values for the applicable tolerances (GRADTL, STEPTL, ITNLIM, STEPMX) are given in Section 5.

The user has a choice between the tensor method and a standard quadratic model-based method. In our experience, the tensor method requires usually fewer iterations and function and derivative evaluations than the standard method, but this may vary depending on the problem. The choice is based on the input parameter METHOD; the default choice is the tensor method. As previously mentioned, for both the tensor and standard methods, a line search strategy is used.

The software can perform scaling of the variable space. If the user inputs a typical magnitude $typx_i$ of each component of x_i, then the performance of the package is equivalent to what would result from redefining the independent variable x in the user's function with

$$x_{\text{scaled}} = \begin{bmatrix} 1/typx_1 & & \\ & \ddots & \\ & & 1/typx_n \end{bmatrix} \cdot x,$$

and then running the package without scaling. In our experience, scaling is often beneficial when different components of x are expected to have widely differing magnitudes, i.e., differing by several orders of magnitude, and may sometimes be necessary in order for the software to solve such problems successfully. The default is no scaling, i.e., each $typx_i = 1$. Scaling is controlled by the parameter TYPX.

The user can have the software print out information at each iteration, print out only the initial iterate and the final result, or not print out anything. When MSG = 0, the software will not print out anything; this is often useful when the software is imbedded in other software. When MSG = 1 (the default), the software will print out the initial iterate x_0, $f(x_0)$, and $\nabla f(x_0)$; the final iterate x_f, $f(x_f)$, and $\nabla f(x_f)$; the reason the algorithm was

terminated; and the number of iterations taken. When MSG = 2, the software prints out this information and, in addition to this, the values of x_c, $f(x_c)$, and $\nabla f(x_c)$ at each iteration.

By using the parameter NDIGIT, the user can input an estimate of the number of accurate digits in the objective function $f(x)$. It is important to provide this information whenever the number of accurate digits in $f(x)$ is expected to be significantly fewer than the full double precision used by this package. This may occur, for example, when $f(x)$ is itself the result of an iterative procedure which returns an answer where only some number of the leading digits are accurate, or when the routine for $f(x)$ is written in single precision. It is particularly important to supply this information when the function evaluation does not have (nearly) full double-precision accuracy and finite-difference derivatives, either gradients or Hessians, are being used.

4. CALLING THE SOFTWARE PACKAGE

There are two ways to call the package. If the user wishes to override the default values of any input parameters, or to supply routines to evaluate the gradient or Hessian, then the following sequence is used:

```
CALL DEFAULT (N, TYPX, FSCALE, GRADTL, STEPTL, ITNLIM, STEPMX, IPR,
     METHOD, GRDFLG, HESFLG, NDIGIT, MSG)
{code to override specific default parameters goes here}
CALL TENSOR (NR, N, X, FCN, GRD, HSN, TYPX, FSCALE, GRADTL, STEPTL,
     ITNLIM, STEPMX, IPR, METHOD, GRDFLG, HESFLG, NDIGIT, MSG, XPLS,
     FPLS, GPLS, H, ITNNO, WRK, IWRK)
```

The routine DFAULT sets all input parameters to their default values, so that the user only needs to specify those values that are desired to have different values than the defaults. For example, if the user wishes to use all the default values except the iteration limit (setting it instead to 300), and wishes to supply analytic gradients, then the calling sequence would be

```
CALL DFAULT (N, TYPX, FSCALE, GRADTL, STEPTL, ITNLIM, STEPMX, IPR,
     METHOD, GRDFLG, HESFLG, NDIGIT, MSG)
ITNLIM = 300
GRDFLG = 1
CALL TENSOR (NR, N, X, FCN, GRD, HSN, TYPX, FSCALE, GRADTL, STEPTL,
     ITNLIM, STEPMX, IPR, METHOD, GRDFLG, HESFLG, NDIGIT, MSG, XPLS,
     FPLS, GPLS, H, ITNNO, WRK, IWRK)
```

The name of the routine for evaluating analytic gradients would be given where the parameter GRD is shown, and this routine would be supplied by the user. Additionally, the values of NR (the row dimension of user's matrix that will contain the Hessian), N, and X would be supplied by the user, and the user would provide a routine to evaluate the objective function $f(x)$ and supply its name in FCN. WRK and IWRK are an NR $\times$ 8 double-precision array and an NR element integer vector, respectively, used as work arrays by the package.

If the user wishes to use all the default values of the parameters and evaluate derivatives by finite differences, then there is a simpler way to call the package. It is

```
CALL TENSRD (NR, N, X, FCN, MSG, XPLS, FPLS, GPLS, H, ITNNO, WRK, IWRK)
```

(TENSRD stands for Tensor Default). TENSRD simply calls DFAULT followed by TENSOR. The user must still supply values of NR, N, and X; the routine to evaluate $f(x)$ and its name in FCN; and the work arrays WRK and IWRK.

5. PARAMETERS AND DEFAULT VALUES

In this section we describe the parameters for the software package. In the parameter list, the symbol $\rightarrow$, $\leftarrow$, or $\leftarrow\rightarrow$ follows each parameter. These symbols specify that the parameter is an input, output, or input-output parameter, respectively. Most of the input parameters do not have to be supplied by the user (see Section 4 and below); if they are not specified, the code gives them the default value specified below. The calling sequence for TENSOR is

```
TENSOR (NR, N, X, FCN, GRD, HSN, TYPX, FSCALE, GRADTL, STEPTL, ITNLIM,
        STEPMX, IPR, METHOD, GRDFLG, HESFLG, NDIGIT, MSG, XPLS, FPLS,
        GPLS, H, ITNNO, WRK, IWRK).
```

The parameters NR, N, ITNLIM, IPR, METHOD, GRDFLG, HESFLG, NDIGIT, MSG, and ITNO are scalar integers; FSCALE, GRADTL, STEPTL, STEPMX, and FPLS are double-precision scalars; FCN, GRD, and HSN are external subroutines; IWRK is an n-dimensional integer vector; X, TYPX, XPLS, and GPLS are n-dimensional double-precision vectors; and the remaining parameter declarations are the double-precision arrays WRK (NR, N) and H (NR, N).

NR $\rightarrow$. A positive integer specifying the row dimension of the matrices H and WRK in the user's calling program. (H is used to store the Hessian matrix; WRK is used for workspace.) NR must be greater than or equal to N. This provision allows the user to solve several problems with different values of N while using the same user storage. If NR $<$ N, the software will print a warning message and stop.

N $\rightarrow$. A positive integer specifying the number of variables in the objective function. The value of N must be less than or equal to the value of NR. If N $\leq$ 0, the program will abort. If N = 1, the program will print a warning message, unless MSG is set to 0.

X $\rightarrow$. An N-vector containing the initial approximation to the solution $x*$.

FCN $\rightarrow$. The name of a user-supplied subroutine that returns in F the value of the objective function $f(x)$ at the current point X. FCN must be declared EXTERNAL in the user's calling program and must conform to the usage

```
CALL FCN (N, X, F),
```

where N is the dimension of the problem; X is the current point; and F is the function value at the current point. FCN must not alter the values of N and X.

GRD → . The name of a user-supplied subroutine that returns in G the value of the gradient $\nabla f(x)$ at the current point X. GRD must be declared EXTERNAL in the user's calling program and must conform to the usage

```
CALL GRD (N, X, G),
```

where N is the dimension of the problem; X is the current point; and G is the gradient at the current point. GRD must not alter the values of N and X.

HSN → . The name of a user-supplied subroutine that returns in H the value of the Hessian $\nabla^2 f(x)$ at the current point X. HSN must be declared EXTERNAL in the user's calling program and must conform to the usage

```
CALL HSN (NR, N, X, H),
```

where NR is the row dimension of H in the users program; N is the dimension of the problem; X is the current point; and H is the Hessian at the current point. HSN must not alter the values of NR, N, or X. HSN should fill only the lower triangular part and diagonal of H, i.e., $H(i, j)$, for $i \geq j$.

TYPX → . An N-vector containing the scaling vector. The default value is TYPX = $(1, 1, \ldots, 1)$. If the user supplies values for TYPX, then TYPX[I] should be the absolute value of the estimated magnitude of x_i at the solution and/or during the solution process. For example, if it is anticipated that the range of values for the iterates will be

$$x_1 \in [-10^{10}, 10^{10}]$$

$$x_2 \in [-10^2, 10^4]$$

$$x_3 \in [-6 \times 10^{-6}, 9 \times 10^{-6}],$$

then an appropriate choice would be TYPX = $[10^{10}, 10^3, 7 \times 10^{-6}]$. In cases like this, where the magnitudes of the components of X differ substantially, it may be necessary to supply scaling information to make the software successful and efficient. If a negative value is specified for TYPX[I], its absolute value is used; while if 0 is specified, 1 is used.

FSCALE → . A positive real number estimating the magnitude of $f(x)$ near the solution $x*$. FSCALE is used in the gradient stopping condition given below. The default value is FSCALE = 1. It may be helpful to specify FSCALE when the units of $f(x)$ make it always many orders of magnitude different from 1. If $f(x_0)$ is much greater than $f(x*)$, FSCALE should approximate $f(x*)$, not $f(x_0)$. If a negative value is specified for FSCALE, its absolute value is used; while if 0 is specified, 1 is used.

GRADTL → . A positive real number giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm.

The scaled gradient is a measure of the relative change in $f(x)$ in the direction x_i. The gradient stopping test used in the software is

$$\max_i \left\{ \frac{|\nabla f(x)_i| \cdot \max\{|x_i|, \text{TYPX[I]}\}}{\max\{|f|, \text{FSCALE}\}} \right\} \leq \text{GRADTL}.$$

DFAULT returns the value GRADTL = macheps$^{1/3}$. (*macheps* is described in Section 6.) If a negative value is specified for GRADTL, the default is used.

STEPTL → . A positive real number containing the tolerance at which the scaled step length is considered close enough to zero to terminate the algorithm. The test used in the software is

$$\max_i \left\{ \frac{|(x_+)_i - (x_c)_i|}{\max\{|(x_+)_i|, \text{TYPX[I]}\}} \right\} \leq \text{STEPTL},$$

where x_+ and x_c are the new and old iteratives, respectively. If the value of STEPTL is too large, the software may terminate prematurely. DFAULT returns the value macheps$^{2/3}$. If a negative value is specified for STEPTL, the default is used.

ITNLIM → . A positive integer specifying the maximum number of iterations to be performed before the program is terminated. DFAULT returns the value 100. If a nonpositive value is specified for ITNLIM, the default is used.

STEPMX → . A positive real number containing the maximum scaled step size allowed in each iteration. DFAULT returns the value

$$\text{STEPMX} = \max\{|\,|x_0|\,|_2 \cdot 10^3, 10^3\},$$

where x_0 is the initial approximation provided by the user. STEPMX is used to prevent steps that would cause the optimization function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. STEPMX should be chosen small enough to prevent the first two of these occurrences but should be larger than any anticipated "reasonable" step. The algorithm will halt and provide a diagnostic if it attempts to exceed STEPMX on five successive iterations. If a nonpositive value is specified for STEPMX, the default is used.

IPR → . A positive integer containing the number of the output unit. DFAULT returns the value 6, which is the standard Fortran output unit.

METHOD → . An integer flag designating which method to use.

METHOD = 0: Use the standard (quadratic model based) method.
METHOD = 1: Use the tensor method.

DFAULT returns the value 1. If a value other than 0 or 1 is specified for METHOD, 1 is used.

GRDFLG → . An integer flag specifying whether a routine to calculate the analytic gradient is provided by the user.

GRDFLG = 0: No analytic gradient supplied by user.

GRDFLG = 1: Analytic gradient supplied by user (will be checked against finite-difference gradient).

GRDFLG = 2: Analytic gradient supplied by user (will *not* be checked against finite-difference gradient).

When GRDFLG = 0, the gradient values are computed by finite differences. When GRDFLG = 1 or 2, the name of the user-supplied routine that evaluates $\nabla f(x)$ must be supplied in GRD. When GRDFLG = 1, the program compares the value of the user's analytic gradient routine at x_0 with a finite-difference estimate, and aborts the program if the relative difference between any two components is greater than 0.01. DFAULT returns the value 0. If a value other than 0, 1, or 2 is specified for GRDFLG, 0 is used.

HESFLG → . An integer flag specifying whether a routine to calculate the analytic Hessian is provided by the user.

HESFLG = 0: No analytic Hessian supplied by user.

HESFLG = 1: Analytic Hessian supplied by user (will be checked against finite-difference Hessian).

HESFLG = 2: Analytic Hessian supplied by user (will *not* be checked against finite-difference Hessian).

When HESFLG = 0, the Hessian values are computed by finite differences. When HESFLG = 1 or 2, the name of the user-supplied routine that evaluates $\nabla^2 f(x)$ must be supplied in HSN. When HESFLG = 1, the program compares the value of the user's analytic Hessian routine at x_0 with a finite-difference estimate, and aborts the program if the relative difference between any two components is greater than 0.01. DFAULT returns the value 0. If a value other than 0, 1, or 2 is specified for HESFLG, 0 is used.

NDIGIT → . An integer which estimates the number of accurate digits in the objective function $f(x)$. DFAULT returns the value $-\log_{10}(macheps)$. If a nonpositive value is specified, the default is used.

MSG ← → . An integer which upon entering the software package specifies the type of printed output to be produced, and which upon exiting the software package specifies the termination condition. The meaning of MSG upon input is:

0: No printed output will be produced.

1: Print out the values of x, $f(x)$, $\nabla f(x)$ at the initial and final iterates, the total number of iterations that were taken, and the reason the algorithm was terminated.

2: Print out the values of x, $f(x)$, $\nabla f(x)$ at every iteration, the total number of iterations that were taken, and the reason the algorithm was terminated.

3: Print the number of function evaluations accumulated, the relative gradient maximum and relative stepsize maximum at each iteration, as well as the information printed for MSG = 2.

DFAULT returns the value 1. If a value other than 0, 1, 2, or 3 is specified for MSG, 1 is used. The meaning of MSG upon exiting the software package is:

−1: The norm of the gradient at the final iterate was less than GRADTL.
−2: The length of the last step was less than STEPTL.
−3: The last iteration failed to locate a lower point.
−4: The iteration limit has been exceeded.
−5: Five consecutive steps of length STEPMX have been taken.
−20: Nonpositive value of N or NR < N was input; program aborted.
−21: Possible coding error in analytic gradient, because analytic gradient at x_0 is not close enough to finite-difference approximation; program aborted. (This check can be overridden by setting GRDFLG = 2.)
−22: Possible coding error in analytic Hessian, because analytic Hessian at x_0 is not close enough to finite-difference approximation; program aborted. (This check can be overridden by setting HESFLG = 2.)

XPLS ← . An N-vector containing the final iterate, which is the best approximation found to a minimizer of $f(x)$.

FPLS ← . A scalar variable that contains the function value at the final iterate XPLS.

GPLS ← . An N-vector containing the gradient value at the final iterate XPLS.

H ← . An array that is used to store the Hessian matrix $\nabla^2 f(x)$ at each iteration. The user needs to declare this array to have dimension NR × NC, where NR is a parameter described at the start of this section and obeys NR ≥ N, and NC is any integer obeying NC ≥ N. The Hessian matrix is stored in the first N rows and columns of H. Upon exiting the software, H contains the Hessian value at the final iterate XPLS.

ITNNO ← . A positive integer containing the total number of iterations that were taken.

WRK → . An NR × 8 double-precision array used as workspace by the software package.

IWRK → . An integer array of length at least N.

6. IMPLEMENTATION DETAILS

The software package is coded in Fortran 77 using double precision. The user must declare all floating-point variables to be double precision.

The software calculates the value of the machine epsilon, defined to be the smallest positive real number *macheps* for which $(1 + macheps) > 1$ in double precision on that computer, in the subroutine MCHEPS. On some computers, the returned value may be incorrect due to compiler optimizations. The user may wish to check the computer value of *macheps* and, if it is incorrect, replace the code in the subroutine MCHEPS with the statement

EPS = correct value of machine epsilon.

Several components of the software package are taken from the UNCMIN unconstrained minimization package of Schnabel et al. [1985]. These routines are: FORSLV and BAKSLV (forward and backward triangular solve); FSTOFD and SNDOFD (first- and second-order finite-difference derivatives); GRDCHK and HESCHK (compare the finite-difference gradient and Hessian to analytic gradient and Hessian, respectively); LNSRCH (line search); OPTSTP (check stopping condition); and most of OPTCHK (check input parameters).

7. SUMMARY OF TEST RESULTS

We have tested this software on the set of unconstrained optimization problems in Moré et al. [1981]. All of these problems except the Powell singular problem have $\nabla^2 f(x*)$ nonsingular. We created two sets of singular test problems $\nabla^2 f(x*)$ having rank $n - 1$ and $n - 2$, respectively, by modifying the nonsingular test problems of Moré et al. [1981] as described in Schnabel and Chow [1989]. The dimensions of these problems range from 2 to 30.

The two methods compared in the tests are the standard method, which uses a quadratic-based model and is accessed in the software by setting the METHOD parameter to 0, and the tensor method, which is used when METHOD is set to 1. The test results for the problems solved successfully by both methods are summarized in Table I. The second and third columns are computed using the total of all the iterations, or all the function evaluations, required by each method to solve all these problems. This is a reflection of the cost of the entire test set. Table I shows that, on the average, the tensor method required about 28 to 34% fewer iterations, and about 22 to 36% fewer function evaluations, to solve these problems. The number of iterations would be an accurate indication of the time required by the code to solve a problem where the number of variables n is not too small and where function evaluation is not too expensive, since in this case the cost per iteration of each method is nearly identical and is dominated by the factorization of an $n \times n$ symmetric matrix at each iteration. The number of function evaluations, which includes finite-difference derivatives, would be an accurate indication of the time required by the code to solve a problem where function evaluation is expensive, as it is on many practical problems. Table I shows also that the efficiency of the tensor method was rarely worse than the standard method, and usually better, on these test problems.

Additionally, Table II shows that the tensor method solved 18 problems that the standard method did not solve, while the standard method solved 3

Table I. Comparison of Tensor and Standard Methods

Test Set	Itns, Tensor / Itns, Standard	Fn Evalns, Tensor / Fn Evalns, Standard	Tensor Better	Tensor Worse	Tie
Nonsingular	0.714	0.779	32	4	10
Singular, rank $n-1$	0.658	0.638	39	2	10
Singular, rank $n-2$	0.674	0.685	31	5	17

Table II. Number of Test Problems Solved by One Method Only

Test Set	Solved by Tensor Method Only	Solved by Standard Method Only
Nonsingular	6	1
Singular, rank $n-1$	5	0
Singular, rank $n-2$	7	2

problems that the tensor method did not solve, with the tensor method having a greater advantage on the singular problems than on the nonsingular problems.

These results indicate that the tensor method is likely to be more efficient than the standard method in solving nonsingular and singular unconstrained optimization problems, and that it may solve a wider range of problems. They also indicate that, for any particular problem, it may be advantageous to have both methods available.

REFERENCES

CHOW, T. 1989. Derivative and secant tensor methods for unconstrained optimization. Ph.D. thesis, Dept. of Computer Science, Univ. of Colorado—Boulder.

DENNIS, J. AND SCHNABEL, R 1983 *Numerical Methods for Nonlinear Equations and Unconstrained Optimization.* Prentice-Hall, Englewood Cliffs, N.J

MORÉ, J., GARBOW, B., AND HILLSTROM, K. 1981. Testing unconstrained optimization software. *ACM Trans. Math. Softw. 7*, 17–41.

SCHNABEL, R. AND CHOW, T 1989 Tensor methods for unconstrained optimization using second derivatives. *SIAM J. Optim. 1*, 293–315.

SCHNABEL, R. AND ESKOW, E. 1990. A new modified Cholesky factorization. *SIAM J. Sci. Stat. Comput 11*, 1136–1158.

SCHNABEL, R., KOONTZ, J., AND WEISS, B. 1985 A modular system of algorithms of unconstrained minimization. *ACM Trans. Math. Softw. 11*, 419–440.

Received May 1991; revised September 1992 and November 1993; accepted December 1993