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 [16] 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 [16] 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. We describe the user interface to the package in §4, which includes both a simplified (default) and a longer calling sequence. In §5 we describe the meaning of the input, input-output, and output parameters for the package. In §6 we present the default values provided by the package. A few implementation details are described in §7. In §8 we give an example of the use of the package. Finally, in §9 we describe comparative testing for an implementation based on the tensor method versus an implementation based on the Newton's 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 §9.

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 [13]
  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
    $$\theta_* = -\frac{(u + \beta_* + \frac{1}{2}v\beta_*^2 + \frac{\gamma}{6}w\beta_*^3)}{w\beta_*}$$
  elseif $\nabla^2 f(x_c)$ is singular with $\mathrm{rank}(\nabla^2 f(x_c)) = n-1$ **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 [12]. In step 2, the Hessian matrix is either calculated analytically or approximated by a graph coloring algorithm described in [9]. 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 [14]. After we compute the $\mathrm{LDL^T}$ of the Hessian matrix using the MA27 package [13], we change the block diagonal matrix D to 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. The stopping criteria of Algorithm 2.1 are described by the parameter `TERMCD` in §5.

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 $\texttt{TYPX}_i$ of each component of x, the performance of the package is the equivalent to what would result from redefining the independent variable x with

$$x_{\text{scaled}} = \begin{bmatrix} 1/\texttt{TYPX}_1 & & \\ & \cdots & \\ & & 1/\texttt{TYPX}_n \end{bmatrix} \cdot x \tag{3.1}$$

and then running the package without scaling. The default value of each $\texttt{TYPX}_i$ 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 described in §5. 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. Interfaces and Usage

Two interfaces have been provided with the package. If the user wishes to use all the defaults options provided by the package, then he (or she) should call STUMSD (STUMSS if single–precision is used). Only the required input described in §3 needs to be supplied. The other interface, STUMCD (STUMCS if single–precision is used), requires the user to supply all parameters. The user may specify selected parameters only by first invoking the subroutine STDFLT, which sets all parameters to their default values, and then overriding only the desired values. The two calling sequences are as follows:

```
C STUMSD interface: the default options provided by STENMIN are used.

      CALL STUMSD(N, X0, NZ, IRN, LIRN, ICN, LICN, FCN, TYPX, MSG, XPLS,
     *            FPLS, GPLS, HESS, WRK, LWRK, IWRK, LIWRK, TERMCD)


C STUMCD interface: the user first invokes the subroutine STDFLT to obtain the default
C options provided by STENMIN, then overrides the desired values.

      CALL STDFLT(N, TYPX, FSCALE, GRADTL, STEPTL, ILIM, STEPMX,
     *            IPR, METHOD, GRDFLG, HSNFLG, NDIGIT, INFORM, MSG)

C USER OVERRIDES SPECIFIC DEFAULT VALUES PARAMETERS, E.G.

            GRADTL   = 1.0D-6
```

```
             ILIM     = 1000
             GRDFLG   = 1
             HSNFLG   = 1

      CALL STUMCD(N, XO, NZ, IRN, LIRN, ICN, LICN, FCN, UGR, USH,
     *            TYPX, FSCALE, GRADTL, STEPTL, ILIM, STEPMX, IPR,
     *            METHOD, GRDFLG, HSNFLG, NDIGIT, MSG, XPLS, FPLS,
     *            GPLS, HESS, WRK, LWRK, IWRK, LIWRK, TERMCD, HTV,
     *            INFORM)
```

5. Parameters and Default Values

The parameters used in the calling sequences of §4 are fully described here. STUMSD uses only those parameters that are preceded by an asterisk. When it is noted that module STDFLT returns a given value, this is the default employed by interface STUMSD. The user may override the default value by utilizing STUMCD.

Following each variable name in the list below appears a one– or a two–headed arrow symbol of forms $\rightarrow$, $\leftarrow$, and $\longleftrightarrow$. These symbols signify that the variable is for input, output, and input-output, respectively.

*N$\rightarrow$: A positive integer variable specifying the number of variables in the problem. **Restriction:** N ≥ 1.

*XO$\rightarrow$: An array of length N that contains an initial estimate of the minimizer x_*.

*NZ$\rightarrow$: An integer variable that must be set by the user to the number of nonzeros stored in the lower or upper triangular part of the Hessian matrix. It is not altered by the program. **Restriction:** NZ ≥ 1.

*IRN$\rightarrow$: An integer array of length LIRN. On entry, it must hold the row index of each nonzero stored in the lower or upper triangular part of the Hessian matrix.

*LIRN$\rightarrow$: An integer variable that must be set by the user to the length of array IRN. LIRN need not be as large as LICN; normally it need not be very much greater than NZ. It is not altered by the program. **Restriction:** LIRN $\geq$ NZ.

*ICN$\rightarrow$: An integer array of length LICN. On entry, it must hold the column index of the nonzeros stored in lower or upper triangular part of the Hessian matrix.

*LICN$\rightarrow$: An integer variable that must be set by the user to the length of the Hessian array HESS and ICN. LICN should ordinarily be 2 to 4 times as large as NZ. It is not altered by the program. **Restriction:** LICN $\geq$ NZ.

*FCN$\rightarrow$: The name of a user supplied subroutine that evaluates the function f at an arbitrary vector x. The subroutine must be declared `EXTERNAL` in the user's calling program and must conform to

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

where `X` is a vector of length `N`. The subroutine must not alter the values of `X`.

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

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

where `N` is the dimension of the problem, `X` is a vector of length `N`, and `G` is the gradient at `X`. `UGR` must not alter the values of `N` and `X`. When using the interface `STUMCD`, if no analytic gradient is supplied (`GRDFLG = 0`), the user must use the dummy name `STDUGR`.

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

```
CALL USH(N, X, NZ, LICN, HESS, IRN, ICN)
```

where `N` is the dimension of the problem, `X` is a vector of length `N`, `HESS` is the Hessian matrix at `X`, `LICN` is the length of `HESS`, `NZ` is the number of nonzeros in the lower or upper triangular part of `HESS`, and `IRN` and `ICN` are the row and column indices of the nonzeros in `HESS`. `USH` must not alter the values of `N`, `X`, and `LICN`. Only the lower or upper triangular part of `HESS` should be given. When using the interface `STUMCD`, if no analytic Hessian is supplied (`HSNFLG = 0`), the user must use the dummy name `STDUSH`.

`*TYPX`→: An array of length `N` in which the typical size of the components of `X` are specified. The typical component sizes should be positive real scalars. If a negative value is specified, its absolute value will be used. When 0.0 is specified, 1.0 will be used. The program will not abort. This vector is used by the the package to determine the scaling matrix D_x. Although the package may work reasonably well in a large number of instances without scaling, it may fail when the components of x_* are of radically different magnitude and scaling is not invoked. If the sizes of the parameters are known to differ by many orders of magnitude, then the scale vector `TYPX` should definitely be used. Module `STDFLT` returns `TYPX` = (1.0,…, 1.0). For example, if it is anticipated that the range of values for the iterates x_k is

$$\begin{aligned} x_1 &\in [-10^{10}\ ,\ 10^{10}] \\ x_2 &\in [-10^{2}\ ,\ 10^{4}] \\ x_3 &\in [-6\times 10^{-6},\ 9\times 10^{-6}] \end{aligned}$$

then an appropriate choice will be `TYPX` = (1.0E+10, 1.0E+3, 7.0E-6).

`FSCALE`→: A positive real number estimating the magnitude of $f(x)$ near the minimizer x_*.

It is used in the gradient stopping condition given below. If $f(x_0)$ is much greater than $f(x_*)$, FSCALE should be approximately $f(x_*)$. If a negative value is specified for FSCALE, its absolute value is used. When 0.0 is specified, 1.0 will be used. The program will not abort.

GRADTL→: Positive scalar giving the tolerance at which the scaled gradient of $f(x)$ is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in f in each direction x_i divided by the relative change in x_i. More precisely, the test used by the program is

$$\max_i \left\{ \frac{|\nabla f(x)|_i \ \max\{|x_i|, \mathtt{TYPX}_i\}}{\max\{|f|, \mathtt{FSCALE}\}} \right\} \leq \mathtt{GRADTL}.$$

The module STDFLT returns the value $\epsilon^{1/3}$. If the user specifies a negative value, the default value is used instead.

STEPTL→: A positive scalar providing the minimum allowable relative step length. STEPTL should be at least as small as 10^{-d}, where d is the number of accurate digits the user desires in the solution x_*. The actual test used is

$$\max_i \left\{ \frac{|x_i{}^k - x_i{}^{k-1}|}{\max\{|x_i{}^k, \mathtt{TYPX}_i|\}} \right\} \leq \mathtt{STEPTL},$$

where x^k and x^{k-1} are the new and old iterates, respectively. The program may terminate prematurely if STEPTL is too large. Module STDFLT returns the value $\epsilon^{2/3}$. If the user specifies a negative value, then the default value is used instead.

ILIM→: Positive integer specifying the maximum iterations to be performed before the program is terminated. Module STDFLT returns ILIM = 500. If the user specifies ILIM ≤ 0, the default value is used instead.

STEPMX→: A positive scalar providing the maximum allowable scaled step length $||D_x(x_+ - x_c)||_2$, where $D_x = diag(\mathtt{1/TYPX}_1, \ldots, \mathtt{1/TYPX}_n)$. STEPMX is used to prevent steps that would cause the optimization problem 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 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. Module STDFLT returns the value STEPMX $= \max\{||x_0||_2 \cdot 10^3, 10^3\}$, where x_0 is the initial approximation provided by the user.

IPR→: The unit on which the routine outputs information. STDFLT returns the value 6, which is the standard FORTRAN unit for the printer.

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

- METHOD $= 0$: Use Newton's method.
- METHOD $= 1$: Use the tensor method.

Module STDFLT returns value 1. If the user specifies an illegal value, module OPTCHK will set METHOD to 1; the program will not abort.

GRDFLG$\rightarrow$: Integer flag designating whether or not analytic Hessian has been supplied by the user.

- GRDFLG $= 0$: No analytic gradient supplied.
- GRDFLG $= 1$: Analytic gradient supplied (will be checked against finite difference gradient.)
- GRDFLG $= 2$: Analytic gradient supplied (will not be checked against finite difference gradient.)

When GRDFLG $= 0$, the gradient is obtained by forward finite differences. When GRDFLG $= 1$ or 2, the name of the user supplied routine that evaluates $\nabla f(x)$ must be supplied in UGR. 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 if the relative difference between any two components is greater than 0.01. The module STDFLT returns GRDFLG $= 0$. If the user specifies an illegal value, the module OPTCHK supplies the value 0.

HSNFLG$\rightarrow$: Integer flag designating whether or not analytic Hessian has been supplied by the user.

- HSNFLG $= 0$: No analytic Hessian supplied.
- HSNFLG $= 1$: Analytic Hessian supplied (will be checked against finite difference Hessian.)
- HSNFLG $= 2$: Analytic Hessian supplied (will not be checked against finite difference Hessian.)

When HSNFLG $= 0$, the Hessian values are computed by forward finite differences based on gradient values. When HSNFLG $= 1$ or 2, the name of the user–supplied routine that evaluates $\nabla^2 f(x)$ must be supplied in USH. When HSNFLG $= 1$, the program compares the value of the user's analytic Hessian routine at x_0 with a finite difference estimate and aborts if the relative difference between any two components is greater than 0.01. The module STDFLT returns HSNFLG $= 0$. If the user specifies an illegal value, the module OPTCHK supplies the value 0.

NDIGIT$\rightarrow$: Integer estimating the number of accurate digits on the objective function $f(x)$. STDFLT returns the value -$\mathtt{LOG}_{10}(\epsilon)$, where ϵ is machine precision. If NDIGIT ≤ 0 then the default value is used instead.

*MSG$\longleftrightarrow$: An integer variable that the user may set on input to inhibit certain automatic checks or override certain default characteristics of the package. Currently, three "message" features can be used individually or in combination.

- MSG $= 0$: No output will be produced.
- MSG $= 1$: Print the input state, the final results, and the stopping conditions.
- MSG $= 2$: Print the intermediate results, that is, the input state, the values of the objective function and the scaled gradient at each iteration, and the final results including the stopping conditions and the number of function, gradient, and Hessian evaluations.

The module STDFLT returns a value of 1. On output, if the program has terminated because of erroneous input, MSG contains an error code indicating the reason:

- MSG $=$ -1 : Illegal dimension N; N ≤ 0. The program aborts.
- MSG $=$ -2 : Illegal length of LIRN or LICN; LIRN ≤ 0 or LICN ≤ 0. The program aborts.

- MSG = -3 : Illegal length of LIWRK or LWRK; LIWRK $<$ 2*LIRN+12*N+2 or LWRK $<$ 7*N. The program aborts.
- MSG = -4 : Illegal number of nonzeros NZ; NZ ≤ 0. The program aborts.
- MSG = -5 : The K-th element of IRN or the K-th element of ICN is not an integer between 1 and N; (IRN(K) $<$ 1 or IRN(K) $>$ N) or (ICN(K) $<$ 1 or ICN(K) $>$ N). The program aborts.
- MSG = -6 : The K-th diagonal element is not in the sparsity pattern. This is checked only if HSNFLG = 0 because the finite difference Hessian approximation require that diagonal elements be in the sparsity pattern. The program aborts.
- MSG = -7 : Redundant entries in sparsity pattern was encountered. When HSNFLG = 1 or HSNFLG = 2, the program aborts. When HSNFLG = 0, the program eliminates the redundant entries and continue the execution (no error message is reported in this case).
- MSG = -8 : Probable coding error in the user's analytic gradient routine . Analytic and finite difference gradient do not agree within a tolerance of 0.01. The program aborts. (This check can be overridden by setting GRDFLG = 2.)
- MSG = -9 : Probable coding error in the user's analytic Hessian routine USH. Analytic and finite difference Hessian do not agree within a tolerance of 0.01. The program aborts. (This check can be overridden by setting HSNFLG = 2.)

*XPLS$\leftarrow$: An array of length N containing the best approximation to the minimizer x_* upon return. (If the algorithm has not converged, the last iterate is returned.)

*FPLS$\leftarrow$: A scalar variable that contains the function value at the final iterate XPLS.

*GPLS$\leftarrow$: An array of length N containing the gradient value at XPLS.

HESS$\longleftrightarrow$: An array that is used to store the Hessian matrix at each iteration. It needs to be at least of dimension LICN. Only the nonzeros in the lower or upper triangular part of the Hessian matrix is stored in HESS. On entry, these nonzeros may be given in any order. On exit, HESS contains the Hessian matrix at the minimizer x_* with the nonzeros sorted by columns if HSNFLG was set to 0.

*WRK$\rightarrow$: An array of length LWRK. This is used as workspace by the package. Its length must be at least 8*N if the STUMSD interface is used and at least 7*N if the STUMCD interface is used.

*LWRK$\rightarrow$: An integer variable. It must be set by the user to the length of array WRK and is not altered by the package.

*IWRK$\rightarrow$: An integer array of length LIWRK. This is used as workspace by the package. Its length must be at least 2*LIRN+12*N+2.

*LIWRK$\rightarrow$: An integer variable. It must be set by the user to the length of array IWRK and is not altered by the package.

*TERMCD←: An integer that specifies the reason why the algorithm has terminated.

- TERMCD = 1 : The scaled gradient at the final iterate was less than GRADTL.
- TERMCD = 2 : The length of the last step was less than STEPTL.
- TERMCD = 3 : Last global step failed to locate a point lower than XPLS. It is likely that either XPLS is an approximate solution of the function or STEPTL is too large.
- TERMCD = 4 : The iteration limit has been exceeded.
- TERMCD = 5 : Five consecutive steps of length STEPMX have been taken.

HTV↔: An array of length N. It need not be set by the user on entry. If INFORM is set to 1, a re-entry must be made with HTV set to HESS times HTV (see INFORM.)

INFORM⟷: An integer variable. If it is set to 1, the user must obtain HESS times HTV and re-enter STUMCD (STUMCS if single–precision is used) with INFORM unchanged. The result of HESS times HTV must be stored in HTV. The default value of INFORM is 0, meaning that HESS times HTV is computed by the package.

6. Summary of Default Values

The following parameters are returned by the module STDFLT:

ILIM = 500
GRDFLG = 0
HSNFLG = 0
IPR = 6
GRADTL = $\epsilon^{1/3}$ (ϵ is machine precision)
STEPTL = $\epsilon^{2/3}$
METHOD = 1
NDIGIT = -LOG$_{10}$(ϵ)
STEPMX = 0.0
TYPX = (1.0,...,1.0)
FSCALE = 1.0
MSG = 1
INFORM = 0

7. Implementation Details

This software package has been coded in Fortran 77. The user has the choice between single– and double–precision versions. The user must then preprocess the package at compile time using either the **tosngl** or **todble** tools from CUTE [2], for the single– and double–precision versions, respectively. The **tosngl** program picks up the appropriate version by selecting any statement that begins with CS in the first column, where the S character means that this is a single–precision version. On the other hand, the **todble** program picks up the appropriate version by selecting any statement that begins with CD in the first column, with D meaning that this is a double–precision version. Note that a statement that begins by neither CS nor CD will be picked by both tools.

The following software are included in the package:

1. Harwell code MA27 [13], which is used for computing the $L^T DL$ factorization of the sparse Hessian matrix.
2. The Coleman and Moré graph coloring software [9, 8, 7], which is used for estimating a finite-difference approximation of a sparse Hessian matrix.
3. The subroutine DSYPRC [10, 11], which is used for modifying the negative eigencomponents obtained when factorizing an indefinite Hessian matrix using the Harwell code MA27.
4. The function DPMEPS [6], which is used for dynamically determining the machine precision.

The program was developed and tested on a Sun SPARC 10 Model 40 computer.

The machine precision is calculated by the package and used in several places including finite differences stepsizes and stopping criteria. On some computers, the returned value may be incorrect because of compiler optimizations. The user may wish to check the computer value of the machine epsilon and, if it is incorrect, replace the code in the function DPMEPS with the following statement

```
DPMEPS = correct value of machine epsilon
```

8. Example of Use

In the example code shown in Figure 8.1, we first call the routine STDFLT, which returns the default values. We then override the values of GRADTL, GRDFLG and HSNFLG. Next we call either the interface STUMSD or STUMCD for the single– and double–precision version, respectively, to solve the sparse unconstrained optimization problem coded in FCN and whose gradient and Hessian are given by UGRAD and UHESS, respectively.

```
C
C STENMIN MINIMIZES AN UNCONSTRAINED NONLINEAR FUNCTION IN N
C UNKNOWNS WHERE THE HESSIAN IS LARGE AND SPARSE, USING TENSOR
C METHODS.
C
C EXAMPLE OF USE FOR STENMIN.  THE TEST PROBLEM IS THE
C THE BROYDEN TRIDIAGONAL [15].
C
C  ALI BOUARICHA, OCTOBER 1994.
C  MCS DIVISION, ARGONNE NATIONAL LAB.
C
      INTEGER          NMAX, N, NZ, LIRN, LICN, ILIM, IPR, METHOD
      INTEGER          GRDFLG, HSNFLG, NDIGIT, MSG, LWRK, LIWRK
      INTEGER          TERMCD, INFORM, I
CD    DOUBLE PRECISION FSCALE, GRADTL, STEPTL, FPLS, STEPMX, ONE
```

```
CS    REAL              FSCALE, GRADTL, STEPTL, FPLS, STEPMX, ONE
      PARAMETER       ( NMAX = 10000, LIRN = 50000, LICN = 500000 )
      PARAMETER       ( LIWRK = 2 * LIRN + 12 * NMAX + 2 )
      PARAMETER       ( LWRK = 7 * NMAX )
      INTEGER           IRN ( LIRN  ), ICN ( LICN )
      INTEGER           IWRK( LIWRK )
CD    DOUBLE PRECISION X   ( NMAX ),  TYPX( NMAX ), XPLS( NMAX )
CD    DOUBLE PRECISION GPLS( NMAX ),  HESS( LICN ), WRK ( LWRK )
CD    DOUBLE PRECISION HTV ( NMAX )
CS    REAL              X   ( NMAX ),  TYPX( NMAX ), XPLS( NMAX )
CS    REAL              GPLS( NMAX ),  HESS( LICN ), WRK ( LWRK )
CS    REAL              HTV ( NMAX )
      EXTERNAL          FCN, UGRAD, UHESS
CD    DATA ONE / 1.0D0 /
CS    DATA ONE / 1.0E0 /

C READ DATA

      READ(5,*) N

C  COMPUTE THE STANDARD STARTING POINT.

      DO 10 I = 1, N
         X(I) = -ONE
 10   CONTINUE

      CALL STDFLT(N,TYPX,FSCALE,GRADTL,STEPTL,ILIM,STEPMX,
     *             IPR,METHOD,GRDFLG,HSNFLG,NDIGIT,INFORM,MSG)

CD    GRADTL = 1.0D-5
CS    GRADTL = 1.0E-3
      GRDFLG = 2
      HSNFLG = 2

C  CALL THE SPARSE OPTIMIZER

CD    CALL STUMCD(N,X,NZ,IRN,LIRN,ICN,LICN,FCN,UGRAD,
CS    CALL STUMCS(N,X,NZ,IRN,LIRN,ICN,LICN,FCN,UGRAD,UHESS,TYPX,
     *             FSCALE,GRADTL,STEPTL,ILIM,STEPMX,IPR,METHOD,
     *             GRDFLG,HSNFLG,NDIGIT,MSG,XPLS,FPLS,GPLS,HESS,
     *             WRK,LWRK,IWRK,LIWRK,TERMCD,HTV,INFORM)

      STOP
      END
```

```
C THE FOLLOWING IS A SUBROUTINE FOR THE BROYDEN TRIDIAGONAL
C PROBLEM

        SUBROUTINE FCN(N, X, F)
        INTEGER N
CD      DOUBLE PRECISION X(N), F
CS      REAL X(N), F

C  LOCAL VARIABLES

        INTEGER I
CD      DOUBLE PRECISION ONE, TWO, THREE
CS      REAL ONE, TWO, THREE
CD      DATA ONE, TWO, THREE / 1.0D0, 2.0D0, 3.0D0 /
CS      DATA ONE, TWO, THREE / 1.0E0, 2.0E0, 3.0E0 /

        F = ((THREE - TWO * X(1)) * X(1) - TWO * X(2) + ONE) ** 2
        DO 10 I = 2, N-1
           F = F + ((THREE - TWO * X(I)) * X(I) - X(I-1) -
     *         TWO * X(I+1) + ONE) ** 2
 10     CONTINUE
        F = F + ((THREE - TWO * X(N)) * X(N) - X(N-1) + ONE) ** 2
        RETURN
        END

C THE FOLLOWING IS A SUBROUTINE FOR THE GRADIENT OF THE BROYDEN
C TRIDIAGONAL PROBLEM

        SUBROUTINE UGRAD(N, X, G)
        INTEGER N
CD      DOUBLE PRECISION X(N), G(N)
CS      REAL X(N), G(N)

C LOCAL VARIABLES

        INTEGER I
CD      DOUBLE PRECISION RL, RM, RR, ONE, TWO, THREE, FOUR
CS      REAL RL, RM, RR, ONE, TWO, THREE, FOUR
CD      DATA ONE, TWO, THREE, FOUR/1.0D0, 2.0D0, 3.0D0, 4.0D0/
CS      DATA ONE, TWO, THREE, FOUR/ 1.0E0, 2.0E0, 3.0E0, 4.0E0/

        RL = (THREE - TWO * X(1)) * X(1) - TWO * X(2) + ONE
        RR = (THREE - TWO * X(2)) * X(2) - X(1) - TWO * X(3) + ONE
```

```
      G(1) = TWO * (RL * (THREE - FOUR * X(1)) - RR)
      DO 10 I = 2, N-1
         IF(I .NE. 2) THEN
            RL = (THREE - TWO * X(I-1)) * X(I-1) - X(I-2) -
     *              TWO * X(I) + ONE
         ENDIF
         RM = (THREE - TWO * X(I)) * X(I) - X(I-1) -
     *            TWO * X(I+1) + ONE
         IF(I .EQ. N-1) THEN
            RR = (THREE - TWO * X(N)) * X(N) - X(N-1) + ONE
         ELSE
            RR = (THREE - TWO * X(I+1)) * X(I+1) - X(I) -
     *              TWO * X(I+2) + ONE
         ENDIF
      G(I) = -TWO * (TWO * RL - RM * (THREE - FOUR * X(I)) + RR)
 10   CONTINUE
      G(N) = -TWO * (TWO * RM - RR * (THREE - FOUR * X(N)))
      RETURN
      END

C THE FOLLOWING IS A SUBROUTINE FOR THE HESSIAN OF THE BROYDEN
C TRIDIAGONAL PROBLEM

      SUBROUTINE UHESS(N,X,NZ,LICN,HESS,IRN,ICN)
      INTEGER N, NZ, LICN
      INTEGER IRN(NZ), ICN(LICN)
CD    DOUBLE PRECISION X(N), HESS(LICN)
CS    REAL X(N), HESS(LICN)

C LOCAL VARIABLES

      INTEGER I
CD    DOUBLE PRECISION RL,RM,RR,DRLIM1,DRMI
CD    DOUBLE PRECISION ONE,TWO,THREE,FOUR
CS    REAL RL,RM,RR,DRLIM1,DRMI
CS    REAL ONE,TWO,THREE,FOUR
CD    DATA ONE, TWO, THREE, FOUR/1.0D0, 2.0D0, 3.0D0, 4.0D0/
CS    DATA ONE, TWO, THREE, FOUR/1.0E0, 2.0E0, 3.0E0, 4.0E0/

      NZ = 1
      RL = (THREE - TWO * X(1)) * X(1) - TWO * X(2) + ONE
      HESS(NZ) = TWO * ((THREE - FOUR * X(1))**2 -
     *            FOUR * RL + ONE)
      IRN(NZ) = 1
```

```
      ICN(NZ) = 1
      DO 10 I = 2, N-1
         DRLIM1 = THREE - FOUR * X(I-1)
         DRMI = THREE - FOUR * X(I)
         IF(I .NE. 2) THEN
            NZ = NZ + 1
            HESS(NZ) = FOUR
            IRN(NZ) = I
            ICN(NZ) = I-2
         ENDIF
         NZ = NZ + 1
         HESS(NZ) = -TWO * (TWO * (THREE - FOUR * X(I-1)) +
     *                ONE * (THREE - FOUR * X(I)))
         IRN(NZ) = I
         ICN(NZ) = I-1
         RM = (THREE - TWO * X(I)) * X(I) - X(I-1) -
     *          TWO * X(I+1) + ONE
         NZ = NZ + 1
         HESS(NZ) = -TWO * (-FOUR - (THREE - FOUR * X(I))**2 +
     *                FOUR * RM  - ONE)
         IRN(NZ) = I
         ICN(NZ) = I
 10   CONTINUE
      RR = (THREE - TWO * X(N)) * X(N) - X(N-1) + ONE
      NZ = NZ + 1
      HESS(NZ) = FOUR
      IRN(NZ) = N
      ICN(NZ) = N-2
      NZ = NZ + 1
      HESS(NZ) = -TWO * (TWO * (THREE - FOUR * X(N-1)) +
     *              THREE - FOUR * X(N))
      IRN(NZ) = N
      ICN(NZ) = N-1
      NZ = NZ + 1
      HESS(NZ) = TWO * (FOUR + (THREE - FOUR * X(N))**2 - FOUR * RR)
      IRN(NZ) = N
      ICN(NZ) = N
      RETURN
      END
```

Figure 8.1: Code to solve the Broyden tridiagonal problem

If we use the double–precision version of the package to solve the Broyden tridiagonal problem given by `FCN`, for `N = 10000`, we obtain the following output:

```
  STDRU0      GRADIENT FLAG      = 2
  STDRU0      HESSIAN FLAG       = 2
  STDRU0      METHOD             = 1
  STDRU0      ITERATION LIMIT    =  500
  STDRU0      MACHINE EPSILON    =  0.2220446049250E-15
  STDRU0      STEP TOLERANCE     =  0.3666852862501E-10
  STDRU0      GRADIENT TOLERANCE =  0.1000000000000E-04
  STDRU0      MAXIMUM STEP SIZE  =  0.1000000000000E+06

---------------------------------------------
  STRSLT    ITERATION K   =    0
  STRSLT    FUNCTION AT X(K)
  STRSLT        0.1001100000000E+05
  STRSLT    SCALED GRADIENT AT X(K)
  STRSLT        0.3800000000000E+02
---------------------------------------------

  STCHKS    RELATIVE GRADIENT CLOSE TO ZERO
  STCHKS    CURRENT ITERATE IS PROBABLY SOLUTION

---------------------------------------------
  STRSLT    ITERATION K   =    4
  STRSLT    FUNCTION AT X(K)
  STRSLT        0.1884575867777E-13
  STRSLT    SCALED GRADIENT AT X(K)
  STRSLT        0.1113397081739E-05
---------------------------------------------

  STRSLT    NUMBER OF FUNCTION EVALUATIONS    5
  STRSLT    NUMBER OF GRADIENT EVALUATIONS    5
  STRSLT    NUMBER OF HESSIAN EVALUATIONS    4
```

In the Appendix, we give another example of use–the optimal design with composite materials problem–from the `MINPACK-2` collection [1].

9. 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, 17] 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 9.1. The descriptions of the test problems and the detailed results are given in [4]. In Table 9.1 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, 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.

Table 9.1: 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

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] W. J. Cody. MACHAR: A subroutine to dynamically determine machine parameters. *ACM Trans. Math. Softw.*, 14:303–311, 1988.

[7] T. F. Coleman, B. S. Garbow, and J. J. Moré. Fortran subroutines for estimating sparse Hessian matrices. *ACM Trans. Math. Software*, 11:378, 1985.

[8] T. F. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Hessian matrices. *ACM Trans. Math. Software*, 11:363–377, 1985.

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

[10] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. Report 89-19, Namur University, Namur, Belgium, 1989.

[11] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT. Springer Series in Computational Mathematics. Springer-Verlag, 1992.

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

[13] 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.

[14] 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.

[15] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Software*, 7(1):17–41, 1981.

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

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

A. Appendix: Another Example of Use: The Optimal Design Problem

In the example given in Figure A.1, we first call the routine STDFLT, which returns the default values. We then override the value of GRADTL and GRDFLG. Next we call either the interface STUMCS or STUMCD for the single– and double–precision version, respectively, to solve the optimal design with composite materials problem (ODC) from the MINPACK-2 collection [1]. Since in the MINPACK-2 collection both the function and the gradient of the ODC problem are coded in the same subroutine DODCFG, we split DODCFG in two subroutines: DODCF and DODCG for the function and gradient evaluations, respectively.

```
C
C STENMIN MINIMIZES AN UNCONSTRAINED NONLINEAR FUNCTION IN N
C UNKNOWNS WHERE THE HESSIAN IS LARGE AND SPARSE, USING TENSOR
C METHODS.
C
C EXAMPLE OF USE FOR STENMIN.  THE TEST PROBLEM IS THE
C OPTIMAL DESIGN WITH COMPOSITE MATERIALS PROBLEM FROM
C THE MINPACK-2 TEST PROBLEM COLLECTION.
C
C  ALI BOUARICHA, OCTOBER 1994.
C  MCS DIVISION, ARGONNE NATIONAL LAB.
C
      INTEGER          NMAX, N, NZ, LIRN, LICN, ILIM, IPR, METHOD
      INTEGER          GRDFLG, HSNFLG, NDIGIT, MSG, LWRK, LIWRK
      INTEGER          TERMCD, INFORM, I, J, K, NX, NY
CD    DOUBLE PRECISION FSCALE, GRADTL, STEPTL, FPLS, STEPMX
CD    DOUBLE PRECISION LAMBDA, HX, HY, TEMP, ONE
CS    REAL             FSCALE, GRADTL, STEPTL, FPLS, STEPMX
CS    REAL             LAMBDA, HX, HY, TEMP, ONE
      PARAMETER      ( NMAX = 10000, LIRN = 50000, LICN = 500000 )
      PARAMETER      ( LIWRK = 2 * LIRN + 12 * NMAX + 2 )
      PARAMETER      ( LWRK = 7 * NMAX )
      INTEGER          IRN ( LIRN  ), ICN ( LICN )
      INTEGER          IWRK( LIWRK )
CD    DOUBLE PRECISION X   ( NMAX ),  TYPX( NMAX ), XPLS( NMAX )
CD    DOUBLE PRECISION GPLS( NMAX ),  HESS( LICN ), WRK ( LWRK )
CD    DOUBLE PRECISION HTV ( NMAX )
CS    REAL             X   ( NMAX ),  TYPX( NMAX ), XPLS( NMAX )
CS    REAL             GPLS( NMAX ),  HESS( LICN ), WRK ( LWRK )
CS    REAL             HTV ( NMAX )
      COMMON / PARAM / NX, NY
      COMMON / OTHER / LAMBDA
      EXTERNAL         DODCF, DODCG, STDUSH
CD    INTRINSIC        DBLE, MIN
```

```
CS    INTRINSIC        FLOAT, MIN
CD    DATA ONE / 1.0D0 /
CS    DATA ONE / 1.0E0 /

C READ DATA

      READ(5,*) NX, NY, LAMBDA
      N = NX * NY

C  COMPUTE THE STANDARD STARTING POINT.

CD    HX = ONE/DBLE(NX+1)
CD    HY = ONE/DBLE(NY+1)
CS    HX = ONE/FLOAT(NX+1)
CS    HY = ONE/FLOAT(NY+1)
      DO 20 J = 1, NY
CD       TEMP = DBLE(MIN(J,NY-J+1))*HY
CS       TEMP = FLOAT(MIN(J,NY-J+1))*HY
         DO 10 I = 1, NX
            K = NX*(J-1) + I
CD          X(K) = -(MIN(DBLE(MIN(I,NX-I+1))*HX,TEMP))**2
CS          X(K) = -(MIN(FLOAT(MIN(I,NX-I+1))*HX,TEMP))**2
   10    CONTINUE
   20 CONTINUE

C  DEFINE THE SPARSITY STRUCTURE OF THE HESSIAN.

      CALL DODCSP(NX,NY,NZ,IRN,ICN)

C SET THE DEFAULT VALUES OF THE PACKAGE.

      CALL STDFLT(N,TYPX,FSCALE,GRADTL,STEPTL,ILIM,STEPMX,
     *            IPR,METHOD,GRDFLG,HSNFLG,NDIGIT,INFORM,MSG)

CD    GRADTL = 1.0D-5
CS    GRADTL = 1.0E-3
      GRDFLG = 2

C  CALL THE SPARSE OPTIMIZER.

CD    CALL STUMCD(N,X,NZ,IRN,LIRN,ICN,LICN,DODCF,DODCG,
CS    CALL STUMCS(N,X,NZ,IRN,LIRN,ICN,LICN,DODCF,DODCG,STDUSH,
     *            TYPX,FSCALE,GRADTL,STEPTL,ILIM,STEPMX,IPR,
     *            METHOD,GRDFLG,HSNFLG,NDIGIT,MSG,XPLS,FPLS,GPLS,
```

```
*            HESS,WRK,LWRK,IWRK,LIWRK,TERMCD,HTV,INFORM)

      STOP
      END
```

Figure A.1: Code to solve the optimal design with composite materials problem

If we use the double–precision version of the package to solve the ODC problem for the following input:

```
NX, NY, LAMBDA :   100 100 0.008,
```

we obtain the following output:

```
  STDRUO      GRADIENT FLAG      = 2
  STDRUO      HESSIAN FLAG       = 0
  STDRUO      METHOD             = 1
  STDRUO      ITERATION LIMIT    =  500
  STDRUO      MACHINE EPSILON    =  0.2220446049250E-15
  STDRUO      STEP TOLERANCE     =  0.3666852862501E-10
  STDRUO      GRADIENT TOLERANCE =  0.1000000000000E-04
  STDRUO      MAXIMUM STEP SIZE  =  0.6521118878154E+04

---------------------------------------------
  STRSLT    ITERATION K   =    0
  STRSLT    FUNCTION AT X(K)
  STRSLT        0.4823420295546E-01
  STRSLT    SCALED GRADIENT AT X(K)
  STRSLT        0.1931183217332E-01
---------------------------------------------

  STCHKS    RELATIVE GRADIENT CLOSE TO ZERO
  STCHKS    CURRENT ITERATE IS PROBABLY SOLUTION

---------------------------------------------
  STRSLT    ITERATION K   =   20
  STRSLT    FUNCTION AT X(K)
  STRSLT       -0.1137724408643E-01
  STRSLT    SCALED GRADIENT AT X(K)
  STRSLT        0.3938142592477E-05
---------------------------------------------

  STRSLT    NUMBER OF FUNCTION EVALUATIONS   67
  STRSLT    NUMBER OF GRADIENT EVALUATIONS   21
  STRSLT    NUMBER OF HESSIAN EVALUATIONS   20
```

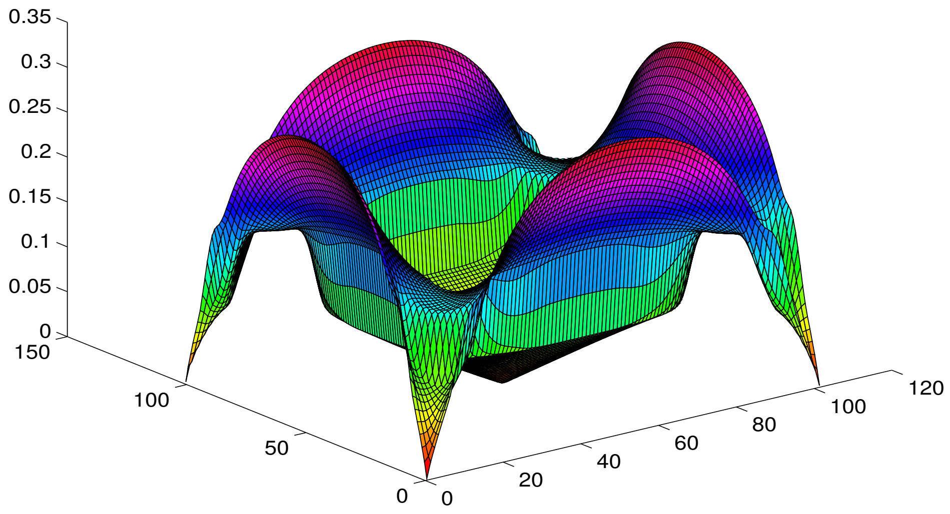

Figure A.2: Norm of $||\nabla v||$ for the stress field v in a design with composite materials

A plot of the norm $||\nabla v||$ of the gradient of the stress field v in the bounded domain $D = (0,1) \times (0,1)$ where `LAMBDA` $= 0.008$ is given in Figure A.2. Figure A.3 shows the contour plot for this surface.

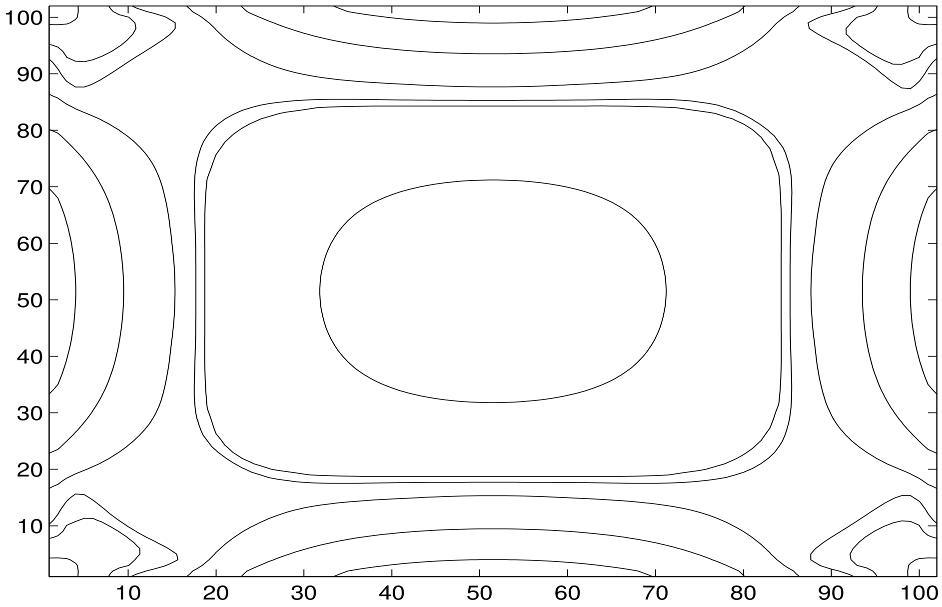

Figure A.3: Contours of $||\nabla v||$ for the stress field v in a design with composite materials