Software for Estimating Sparse Hessian Matrices

THOMAS F. COLEMAN
Cornell University
and
BURTON S. GARBOW and JORGE J. MORÉ
Argonne National Laboratory

The solution of a nonlinear optimization problem often requires an estimate of the Hessian matrix for a function f. In large scale problems, the Hessian matrix is usually sparse, and then estimation by differences of gradients is attractive because the number of differences can be small compared to the dimension of the problem. In this paper we describe a set of subroutines whose purpose is to estimate the Hessian matrix with the least possible number of gradient evaluations.

Categories and Subject Descriptors: E.1 [Data]: Data Structures—*graphs*; E.2 [Data]: Data Storage Representations—*linked representations*; G.1.3 [Numerical Analysis]: Numerical Linear Algebra—*sparse and very large systems*; G.1.6 [Numerical Analysis]: Optimization—*nonlinear programming*

General Terms: Algorithms

Additional Key Words and Phrases: Numerical differentiation, gradient, Hessian matrix, large sparse optimization, nonlinear problems, graph coloring

The Algorithm: Algorithm 649. FORTRAN Subroutines for Estimating Sparse Hessian Matrices. *ACM Trans. Math. Softw. 11*, 4 (Dec. 1985) 378

1. INTRODUCTION

The solution of a nonlinear optimization problem often requires an estimate of the Hessian matrix for a mapping $f: R^n \to R$. In large scale problems the Hessian matrix $\nabla^2 f(x)$ is usually sparse, and then estimation by differences of gradients is attractive because the number of differences can be small compared to the dimension of the problem. In this paper we describe a set of subroutines whose purpose is to estimate the Hessian matrix of a mapping $f: R^n \to R$ with the least possible number of gradient evaluations.

The problem of estimating a sparse Hessian matrix can be phrased in the following terms: Given a symmetric matrix A of order n, obtain vectors $d_1, d_2, \ldots, d_p$ such that $Ad_1, Ad_2, \ldots, Ad_p$ determine A uniquely. In this formulation A is associated with the Hessian matrix $\nabla^2 f(x)$ and the product Ad is associated with an estimate of $\nabla^2 f(x)d$. Typically, the estimate of $\nabla^2 f(x)d$ is obtained by the forward difference

$$\nabla^2 f(x)d \approx \nabla f(x + d) - \nabla f(x) \tag{1.1}$$

The work of the first author was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research of the U.S. Department of Energy under Contract DE-AC02-83ER13069. The work of the second and third authors was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research of the U.S. Department of Energy under Contract W-31-109-Eng-38.

Authors' addresses: T. F. Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853. B. S. Garbow and J. J. Moré, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

or the central difference

$$\nabla^2 f(x)d \approx \tfrac{1}{2}[\nabla f(x + d) - \nabla f(x - d)] \tag{1.2}$$

approximation. Thus each evaluation of Ad requires at least one gradient evaluation. Also note that since A is associated with the Hessian matrix, the sparsity structure of A should represent the sparsity structure of $\nabla^2 f(x)$ for all x of interest. In particular, since in a minimization problem the Hessian matrix is usually positive definite at the solution, it is natural to assume that A has nonzero diagonal elements.

The algorithms that we have implemented are based on the work of Powell and Toint [10], and Coleman and Moré [2]. These authors considered direct and indirect methods for determining symmetric matrices. Indirect methods usually require fewer gradient evaluations while direct methods produce more accurate approximations to the Hessian matrix. We have implemented both types of methods. For purposes of exposition it is convenient to view our algorithms as lower triangular substitution methods.

A lower triangular substitution method is specified by a permutation matrix π and a partition of the columns of a symmetric matrix A into groups $C_1, \ldots, C_p$ such that if L_π is the lower triangular part of $\pi^T A \pi$ then columns in the same group do not have a nonzero in the same row position of L_π. We now show that a partition of the columns of A with this property is consistent with the determination of A by a lower triangular substitution method. Let

$$S = \{(i, j): \pi(i) \geq \pi(j)\},$$

and given a group C define a direction $d \in R^n$ with components $\delta_j \neq 0$ if j belongs to C and $\delta_j = 0$ otherwise. Then

$$Ad = \sum_{l \in C} \delta_l a_l,$$

where $a_1, \ldots, a_n$ are the columns of A. To determine a_{ij} with $(i, j) \in S$ and $j \in C$, note that for any other column $l \in C$ we must have $(i, l) \notin S$. Thus,

$$(Ad)_i = \delta_j a_{ij} + \sum_{(i,l) \notin S, l \in C} \delta_l a_{il}.$$

This expression and the symmetry of A show that a_{ij} depends on $(Ad)_i$ and on elements of L_π in rows $l > i$. It follows that we can determine the rows of L_π in the order $n, \ldots, 1$, and thus A can be determined with p evaluations of Ad.

A lower triangular substitution method is direct if whenever $(i, j) \in S$ and $j \in C$ for a given group C, then there is no pair (i, l) with $(i, l) \notin S$ and $l \in C$. Thus in a direct method a_{ij} depends only on the difference parameter δ_j, while in an indirect method a_{ij} may also depend on other difference parameters δ_l with $l \in C$. As we shall see, this is an important difference between direct and indirect methods.

Given the sparsity pattern of a symmetric matrix A of order n, subroutine DSSM determines a permutation matrix π and a partition of the columns of A consistent with the determination of A by either a direct or indirect lower triangular substitution method. Subrouting FDHS computes an approximation to the Hessian matrix of a mapping $f: R^n \to R$ by a lower triangular substitution method. Most of the information needed by FDHS is provided by DSSM; the user only needs to provide the appropriate difference parameters and gradient

differences. Additional information on DSSM and FDHS can be found in Sections 2 and 3, respectively. An example illustrating the use of subroutines DSSM and FDHS is provided in Section 4. This example also serves as a test program for our package.

An overview of the complete package is presented in Section 5. Users interested in algorithmic details should consult Section 6 where we describe the algorithms that determine the partition of the columns of A in the direct method and the permutation matrix π in the indirect method. The interface subroutines and the subroutines that implement the algorithms of Section 6 are new; all other subroutines are from the package of Coleman, Garbow, and Moré [1] for estimating sparse Jacobian matrices.

Numerical results are presented in Section 7. We discuss the performance of DSSM on the Everstine [4] sparsity patterns, and we also compare our software with subroutines TD03A and TD03B in the Harwell subroutine library.

2. SUBROUTINE DSSM

Given the sparsity pattern of a symmetric matrix A of order n, subroutine DSSM determines a symmetric permutation of A and a partition of the columns of A consistent with the determination of A by a lower triangular substitution method.

The user specifies a definition of the sparsity pattern of A by providing the pairs (i, j) for which $a_{ij} \neq 0$. Since A is symmetric, it is only necessary to provide the indices for the nonzero elements in the lower triangular part of A:

$$(indrow(k), indcol(k)), \quad k = 1, 2, \ldots, npairs. \tag{2.1}$$

These pairs can be provided in any order. Duplicate pairs are allowed, but the subroutine eliminates them. DSSM requires that the diagonal elements be part of the sparsity pattern and replaces any pair (i, j) where $i < j$ with the pair (j, i).

On output, DSSM defines the symmetric permutation of A via the integer array *listp* by placing the (i, j) element of A in the $(listp(i), listp(j))$ position of the permuted matrix. Since the permuted matrix is $\pi^T A \pi$, it follows that column $listp(j)$ of π is the jth column of the identity matrix. The partition of the columns of A is defined via the integer array *ngrp* by setting $ngrp(j)$ to the group number of column j. In addition, the variable *mingrp* provides a lower bound on the number of groups required in a partition consistent with the determination of A by a lower triangular substitution method, and the variable *maxgrp* is the number of groups in the partition obtained by DSSM.

On output, DSSM also has transformed the specification of the sparsity pattern provided by *indrow* and *indcol* into an alternative specification more appropriate for the algorithms used by DSSM. The original data is effectively preserved because this alternative specification allows the user to recover the pairs (i, j) for which $a_{ij} \neq 0$. Details are provided at the end of this section.

The algorithm used by DSSM to determine the symmetric permutation and the partition of the columns of A depends on the integer parameter *method*. If $method = 1$, then DSSM uses a direct method; otherwise an indirect method is used.

The value of *mingrp* is independent of the choice of method. If L_π is the lower triangular part of $\pi^T A \pi$ and $\rho_{max}(\pi)$ is the maximum number of nonzeros in any

row of L_π, then DSSM sets

$$mingrp = min\{\rho_{max}(\pi): \pi \text{ a permutation}\}.$$

The value of *maxgrp*, on the other hand, is heavily dependent on the choice of method. The direct method usually requires more evaluations of Ad to determine A than the indirect method. Thus, if *method* = 1, a user should expect a larger value of *maxgrp* than for *method* $\neq$ 1. However, the direct method produces a more accurate approximation to the Hessian matrix; see, for example, the numerical results in Section 4.

Our experience on practical problems is that a direct method usually determines a partition with *maxgrp* about 50 percent higher than with an indirect method. We have also noted that the indirect method in DSSM typically requires one or two more groups than the bound specified by *mingrp*. For some problems *maxgrp* agrees with *mingrp* and then DSSM is an optimal lower triangular substitution method.

Execution times for the indirect method in DSSM are satisfactory—the number of operations required by one call is proportional to

$$\sum_{i=1}^{n} \rho_i^2(\pi), \tag{2.2}$$

where $\rho_i(\pi)$ is the number of nonzeros in the ith row of L_π. Estimates for the execution time of the direct method are not as simple. We note, however, that in all the practical problems tried, the direct method has been faster than the indirect method. Of course, the claim that (2.2) is a measure of the running time of DSSM assumes that *npairs* is not more than a constant times (2.2). This is certainly the case in any nontrivial situation since (2.2) is not less than the number of nonzero elements in the lower triangular part of A.

An impression of the overhead required by DSSM can be obtained by noting that for a quadratic mapping f, the number of operations needed to evaluate the Hessian matrix by differences is at least (2.2). Indeed, since a lower triangular substitution method requires at least $\rho_{max}(\pi)$ gradients, the number of operations needed to evaluate all the gradients is at least

$$\rho_{max}(\pi) \sum_{i=1}^{n} \rho_i(\pi). \tag{2.3}$$

Note that (2.2) is bounded above by (2.3) for any choice of permutation matrix π. If ∇f is a nonlinear mapping, then estimation of the Hessian matrix is likely to require considerably more operations than (2.3). Moreover, in a typical nonlinear problem DSSM will only be called once, whereas it will be necessary to estimate the Hessian matrix many times. These arguments support our claim that the execution times for DSSM are satisfactory.

Implementation of DSSM, so that the execution time is bounded by a constant multiple of (2.2), requires an appropriate data structure. The pairs (i, j) for which $a_{ij} \neq 0$ is a convenient data structure for the user, but DSSM requires a different data structure. The algorithms called by DSSM require both *column-oriented* and *row-oriented* definitions of the sparsity pattern of the lower triangular part of A. The arrays *indrow* and *jpntr* provide a column-oriented definition if the

row indices for the nonzero elements of the jth column are

$$indrow(k), \qquad k = jpntr(j), \ldots, jpntr(j+1) - 1,$$

while the arrays *indcol* and *ipntr* provide a row-oriented definition if the column indices for the nonzero elements of the ith row are

$$indcol(k), \qquad k = ipntr(i), \ldots, ipntr(i+1) - 1.$$

Given the pairs (2.1) for which $a_{ij} \neq 0$, subroutine DSSM generates column-oriented and row-oriented definitions of the sparsity pattern of the lower triangular part of A. The original data can be recovered because

$$(indrow(k), j), \qquad k = jpntr(j), \ldots, jpntr(j+1) - 1,$$

are the nonzero elements in column j of the lower triangular part of A; the nonzero elements in a given row can be generated in a similar manner.

3. SUBROUTINE FDHS

Given a symmetric permutation of the Hessian matrix and a partition of the columns of the Hessian matrix consistent with the determination of the Hessian matrix by a lower triangular substitution method, subroutine FDHS computes an approximation to the Hessian matrix.

Most of the information needed by FDHS is provided by DSSM. In particular, the symmetric permutation is defined by the array *listp* and the partition is defined by the array *ngrp*. The user must provide an array *eta* of difference parameters and, for each group number *numgrp*, an array *fhesd* with an approximation to $\nabla^2 f(x)d$ where the vector d is defined by setting $d(j) = eta(j)$ if $ngrp(j) = numgrp$ and $d(j) = 0.0$ otherwise. We do not discuss techniques for choosing the difference parameters, but see, for example, [3], and [5]. The approximation to $\nabla^2 f(x)d$ would usually be either (1.1) or (1.2).

The lower triangular method requires that the approximations to $\nabla^2 f(x)d$ be stored in special locations of the array *fhes*. This is done by executing

```
call fdhs(n, indrow, jpntr, indcol, ipntr, listp,
*          ngrp, maxgrp, numgrp, eta, fhesd, fhes, iwa)
```

successively with $numgrp = 1, 2, \ldots, maxgrp$. On the call with $numgrp = maxgrp$, FDHS proceeds to overwrite *fhes* with the approximation to the lower triangular part of the Hessian matrix. Storage is done with a column-oriented definition of the sparsity pattern, and thus the nonzero elements of column j in the lower triangular part of the Hessian matrix are

$$fhes(k), \qquad k = jpntr(j), \ldots, jpntr(j+1) - 1.$$

An example of the use of FDHS can be found in the next section.

4. EXAMPLE

We illustrate the use of subroutines DSSM and FDHS by considering the problem of approximating the Hessian matrix in a minimal surface problem.

The classical minimal surface problem is to find a function with minimal surface area over the unit square and with specified values at the boundary. We

```
      nnz = 0
      do 10 j = 1, n
        nnz = nnz + 1
        indcol(nnz) = j
        indrow(nnz) = j
        if (mod(j, l) .ne. 0) then
          nnz = nnz + 1
          indcol(nnz) = j
          indrow(nnz) = j + 1
          end if
        if (j + l) .le. n) then
          nnz = nnz + 1
          indcol(nnz) = j
          indrow(nnz) = j + l
          if (mod(j, l) .ne. 1) then
            nnz = nnz + 1
            indcol(nnz) = j
            indrow(nnz) = j + l - 1
            end if
          if (mod(j, l) .ne. 0) then
            nnz = nnz + 1
            indcol(nnz) = j
            indrow(nnz) = j + l + 1
            end if
          end if
   10     continue
```

Program I

discretize this problem as done by Griewank and Toint [7]. The unit square is subdivided into $m = (l + 1)^2$ equal subsquares so that the unknowns of the problem become the $n = l^2$ function values at interior corners of the subsquares. The discrete minimal surface problem is then to minimize

$$f(x) = \sum_{i,j=0}^{l} \sigma_{i,j}(x), \tag{4.1}$$

where $\sigma_{i,j}(x)$ is the surface area approximation for the i, j subsquare; if we let $\xi_s = s/(l + 1)$, then the i, j subsquare has coordinates (ξ_i, ξ_j) in the corner closest to the origin. For the interior subsquares we use the approximation

$$\sigma_{i,j}(x) = \frac{1}{m}\left(1 + \frac{m}{2}\left((x_{k+l+1} - x_k)^2 + (x_{k+l} - x_{k+1})^2\right)\right)^{1/2},$$

where x_k, with $k = l(j - 1) + i$, is the minimal area function at (ξ_i, ξ_j). A similar approximation is used for the subsquares on the boundary. Thus $\nabla^2 f(x)$ is a block tridiagonal matrix where each block is a tridiagonal matrix of order l.

The sparsity pattern of the Hessian matrix of (4.1) can be specified by Program I where it is assumed that $n = l^2$. Given the sparsity pattern, an appropriate partition can be determined with a call to DSSM:

```
      call dssm(n, nnz, indrow, indcol, method, listp, ngrp, maxgrp, mingrp,
     *          info, ipntr, jpntr, iwa, liwa).
```

As pointed out in Section 2, on output from DSSM the array pairs *indrow, jpntr*

Table I. Output from DSSM for Minimal Surface Problem

			Direct method		Indirect method	
n	*nnz*	*mingrp*	*maxgrp*	time	*maxgrp*	time
100	442	5	10	0.37	7	0.57
400	1882	5	10	1.52	7	2.53
900	4322	5	11	3.67	7	5.72
1600	7762	5	11	6.40	7	10.30
2500	12202	5	10	9.55	7	17.80

```
      call fcn(l, x, gvec)
      do 30 numgrp = 1, maxgrp
        do 10 j = 1, n
          xd(j) = x(j)
          if (ngrp(j) .eq. numgrp) xd(j) = x(j) + eta(j)
10      continue
        call fcn(l, xd, fhesd)
        do 20 i = 1, n
          fhesd(i) = fhesd(i) - gvec(i)
20      continue
        call fdhs (n, indrow, jpntr, indcol. ipntr, listp,
                   ngrp, maxgrp, numgrp, eta, fhesd, fhes, iwa)
30      continue
```

Program II

and *indcol*, *ipntr* provide, respectively, column-oriented and row-oriented definitions of the sparsity pattern for the lower triangular part of the Hessian matrix. In this section we are mainly interested in the output values of *mingrp* and *maxgrp*.

Table I shows that for this problem the *maxgrp* values for the direct method (*method* = 1) are about 50 percent higher than for the indirect method; this is typical. Also note that the indirect method requires two more groups than the bound specified by *mingrp*. This is not unusual, although for many problems *maxgrp* is closer to *mingrp*. We will discuss further the relationship between *maxgrp* and *mingrp* in Section 7.

The timing results in Table I show that for this problem the execution time (measured in seconds on a VAX 11/780) for the indirect (as well as the direct) method in DSSM is approximately a linear function of n. This is as expected because the execution time for the indirect method is proportional to (2.2), and for the minimal surface problem (2.2) is bounded by a linear function of n. The timing results further support our observation that on realistic problems the direct method executes faster than the indirect method.

Given the output from DSSM, we can readily determine an approximation to the Hessian matrix of (4.1). In addition to the output from DSSM, we need a subroutine to evaluate the gradient ∇f of f. Since $\sigma_{i,j}$ has a simple form, and since (4.1) holds, it is not difficult to write a subroutine *fcn*(l, x, *gvec*) which will evaluate ∇f at x and return $\nabla f(x)$ in the array *gvec* of length $n = l^2$. Given this information, we can call FDHS to obtain an approximation to the Hessian matrix of f. Assuming that *eta* is a vector of difference parameters suitable for the forward difference approximation (1.1), the code in Program II stores the approximation in the array *fhes*.

Table II. Output from FDHS

	Direct method		Indirect method	
n	*abserr*	*relerr*	*abserr*	*relerr*
100	$2.4e-4$	$6.5e-3$	$3.6e-4$	$1.0e-2$
400	$3.3e-4$	$2.2e-2$	$1.1e-3$	$5.1e-2$
900	$4.8e-4$	$4.8e-2$	$3.4e-3$	$1.2e-1$
1600	$6.3e-4$	$8.6e-2$	$8.3e-3$	$3.4e-1$
2500	$7.8e-4$	$1.3e-1$	$7.4e-3$	$7.7e-1$

We have already noted that indirect methods usually determine the Hessian matrix with fewer evaluations of the gradient. This is clearly an advantage. On the other hand, indirect methods produce less accurate approximations to the Hessian matrix than direct methods. We will use the minimal surface problem to illustrate this last point.

Table II shows the largest absolute and relative errors of the approximate Hessian matrix when FDHS is used to approximate the Hessian matrix of (4.1) at the point x where

$$x(l(j-1)+i) = \xi_i^2 + \xi_j^2, \qquad 1 \le i, j \le l,$$

with $\xi_s = s/(l+1)$. At this point the magnitudes of the nonzero elements of the Hessian matrix range between 2.0 and about $1/n$. The approximate Hessian matrix was determined by Program II with difference parameters

$$eta(j) = 10^{-4}. \tag{4.2}$$

All computations were done in single precision on a VAX 11/780 which has a machine precision of $1.2\ 10^{-7}$.

Table II shows that the direct method produces a more accurate approximation to the Hessian matrix. In a direct method, the accuracy of the approximate Hessian matrix is completely determined by the choice of difference parameters, and in this case the accuracy is quite reasonable. In an indirect method, the accuracy of the approximate Hessian matrix also depends on the errors from the substitution process. The results of Table II show that in this case the additional loss of accuracy is not severe.

The error analysis in [10] suggests that the loss of accuracy in an indirect method can be severe if the difference parameters vary significantly in magnitude. We illustrate this possibility with the choice of difference parameters

$$eta(j) = \frac{5\ 10^{-4}}{l}\left(\left[\frac{j-1}{l}\right] + 1\right), \tag{4.3}$$

where $[r]$ is the integer part of r. This choice of difference parameters satisfies

$$eta(j) \in [(1/l)5\ 10^{-4}, 5\ 10^{-4}] = I.$$

For the minimal surface problem it can be shown that any choice of difference parameters in the interval I is reasonable. Thus, it is not surprising that in Table III the errors for the direct method with choice (4.3) are quite similar to those obtained with choice (4.2), with the errors increasing by at most a factor of 3.

Table III. Output from FDHS

	Direct method		Indirect method	
n	*abserr*	*relerr*	*abserr*	*relerr*
100	$3.2e-4$	$1.4e-2$	$3.0e-3$	$7.8e-2$
400	$6.7e-4$	$5.6e-2$	$4.6e-2$	$2.9e+0$
900	$1.3e-3$	$1.2e-1$	$3.6e-2$	$1.2e+0$
1600	$1.2e-3$	$2.1e-1$	$3.5e-1$	$7.7e+0$
2500	$1.5e-3$	$3.3e-1$	$2.8e-1$	$2.1e+1$

The reason for choice (4.3) is to illustrate the effect of variations in the magnitude of difference parameters on the errors for the indirect method.

The results of Table III show that the use of the indirect method leads to significant increases in the magnitude of the errors. The absolute errors have grown by a factor of about l, with dramatically larger increases in the last two cases, while the relative errors show that some of the entries in the approximate Hessian matrix have no correct digits.

The importance of the loss in accuracy with an indirect method depends on the application and on the precision of the computations. In this example, all computations were done with a precision of about six decimal digits, and thus the additional loss of accuracy is severe. If all computations, including the evaluation of the gradient, had a higher precision, then the loss would not be as significant.

5. SUBROUTINES FOR ESTIMATING SPARSE HESSIAN MATRICES

In addition to the interface subroutines DSSM and FDHS, implementation of a lower triangular substitution method requires seven subroutines from the package of Coleman, Garbow, and Moré [1] for estimating sparse Jacobian matrices, and three new subroutines. In this section we present an overview of the complete package.

There are two stages in the implementation of a lower triangular substitution method. The first stage computes a suitable permutation matrix π and determines a partition of the columns of A such that columns of L_π in the same group do not have a nonzero in the same row position. The input to the first stage is the sparsity pattern of the matrix A. In the second stage, appropriate difference parameters and gradient differences must be provided, in addition to a description of the permutation matrix π and the partition of the columns of the Hessian matrix. The result of the second stage is an approximation to the Hessian matrix.

In a direct method we first obtain the partition of the columns of A and then determine the permutation matrix π, the reverse order of the indirect method. Subroutine SDPT determines the partition in the direct method, while the permutation for the indirect method is obtained with subroutines SLOG and IDOG. These three subroutines can be conveniently viewed as graph algorithms. They are discussed further in Section 6, where we also describe the choice of permutation for the direct method.

The software for determining the partition in the indirect method is based on the subroutine package in [1]. Given row-oriented and column-oriented defini-

tions of the sparsity pattern of an m by n matrix B, these subroutines produce a partition of the columns of B such that columns in the same group do not have a nonzero in the same row position. We can thus determine an appropriate partition if we obtain row-oriented and column-oriented definitions of the sparsity pattern of L_π.

The transition from (2.1) to the appropriate data structure for L_π is not difficult. Recall that the input pairs (2.1) specify the nonzero elements in the lower triangular part of A, and that the array *listp* describes the permutation matrix π by requiring that column *listp*(j) of π be column j of the identity matrix. Element (*listp*(i), *listp*(j)) of $\pi^T A \pi$ is thus a_{ij}, and hence the program segment

```
      do 10 k = 1, npairs
        i = indrow(k)
        j = indcol(k)
        indrow(k) = max(listp(i), listp(j))
        indcol(k) = min(listp(i), listp(j))
10      continue
```

generates the sparsity pattern of L_π. The appropriate data structure for L_π can now be obtained because a standard task in sparse matrix manipulation is to transform this description of the sparsity pattern of a matrix into row-oriented and column-oriented definitions of the sparsity pattern. See, for example, [1].

We complete this overview with some algorithmic details on the interface subroutine FDHS. We follow the outline presented in the introduction, and assume that rows $k+1$ through n of L_π have been determined. The kth row can then be determined in three steps.

(1) Let $i = \pi^{-1}(k)$.
(2) Find the positions of the elements in the ith row of A that have been determined.
(3) For each element a_{ij} that has not been determined, let C be the group of column j and form

$$\sigma_{ij} = \sum_{\pi(i) \le \pi(l),\, l \in C} \delta_l a_{il}.$$

Compute a_{ij} and store it in the appropriate position.

Implementation of these steps needs a bit of clarification and care. Step (1) requires the inversion of the permutation π. In step (2) we can find out if an element a_{ij} has been determined by checking that $\pi(i) \ge \pi(j)$. The position of a_{ij} in the column-oriented storage of the lower triangular part of A is stored in an auxiliary integer work array of length n. Given the information determined in step (2), the sum σ_{ij} in step (3) is formed by noting that $l \in C$ if and only if *ngrp*(l) = *ngrp*(j). Element a_{ij} is computed via

$$a_{ij} = ((Ad)_i - \sigma_{ij})/\delta_j.$$

Since this computation requires that $(Ad)_i$ be readily available, we initially process all the vectors Ad, and store $(Ad)_i$ in the position of a_{ij}.

The above description provides the major ideas involved in the implementation of FDHS. If this outline is followed then the execution time of FDHS is bounded

by (2.3). This bound is adequate because, as noted in Section 2, the number of operations needed to evaluate all the vectors Ad is at least (2.3).

6. GRAPH ALGORITHMS

The algorithms for obtaining the partition of the columns of A in a direct method, and the permutation matrix π in an indirect method, can be readily expressed as graph algorithms. The purpose of this section is to provide algorithmic details.

A graph G is an ordered pair (V, E) where V is a finite and nonempty set of *vertices* and the *edges* E are unordered pairs of distinct vertices. The vertices u and v are *adjacent* if (u, v) is an edge with *endpoints* u and v. The *degree* of a vertex v is the number $deg(v)$ of edges with v as an endpoint. Given a nonempty subset W of V, the *subgraph* $G[W]$ *induced* by W has vertex set W and edge (u, v) if $(u, v) \in E$ and $u, v \in W$.

Given a symmetric matrix A of order n, the appropriate graph for our work has vertices $1, 2, \ldots, n$ and edge (i, j) if and only if $i \neq j$ and $a_{ij} \neq 0$. In graph theory terminology, this is the *adjacency graph* of A. An ordering of the vertices of the adjacency graph of a matrix A thus corresponds to a permutation of the rows and columns of A.

In the *smallest-last* ordering vertex v_k is determined after $v_{k+1}, \ldots, v_n$ have been selected by choosing v_k so that its degree in the subgraph induced by

$$V - \{v_{k+1}, \ldots, v_n\}$$

is minimal. In the *incidence degree* ordering v_k is determined after $v_1, \ldots, v_{k-1}$ have been selected by choosing v_k so that its degree in the subgraph induced by $\{v_1, \ldots, v_k\}$ is maximal. The *incidence degree* of v_k is the degree of v_k in this subgraph.

The smallest-last and incidence degree orderings are implemented by subroutines SLOG and IDOG, respectively. These subroutines can be implemented to run in time proportional to $|V| + |E|$ provided we are given the adjacency lists for the graph; that is, arrays $npntr(\cdot)$ and $nghbr(\cdot)$ such that the vertices adjacent to the jth vertex are

$$nghbr(k), \qquad k = npntr(j), \ldots, npntr(j + 1) - 1.$$

See, for example, the description in [9] for the smallest-last ordering. In our application we are working with the adjacency graph of A and have row-oriented and column-oriented data structures for the lower triangular part of A. Thus

$$\{i: i = indrow(k), \qquad k = jpntr(j), \ldots, jpntr(j + 1) - 1\}$$

are the vertices adjacent to vertex j with $i \geq j$, and

$$\{i: i = indcol(k), \qquad k = ipntr(j), \ldots, ipntr(j + 1) - 1\}$$

are the vertices adjacent to vertex j with $i \leq j$. These are, respectively, the successor and predecessor adjacency lists. This is an uncommon data structure for a graph algorithm, but since all vertices adjacent to a given vertex can be readily determined, it is not difficult to implement the ordering algorithms to run in time proportional to $|V| + |E|$. Thus the execution time for subroutines SLOG and IDOG is proportional to the number of nonzero elements in A.

Recall that $\rho_{max}(\pi)$ is the maximum number of nonzeros in any row of L_π. Matula [8], and Powell and Toint [10], proved independently that the smallest-last ordering produces a permutation π_1 which minimizes $\rho_{max}(\pi)$ over all possible permutations. This is an important property because a lower triangular substitution method needs at least $\rho_{max}(\pi)$ groups. From this point of view, any permutation π_2 such that

$$\rho_{max}(\pi_2) = \rho_{max}(\pi_1) \tag{6.1}$$

is equivalent to π_1. We have observed that (6.1) usually holds when π_2 is the permutation produced by the incidence degree ordering of subroutine IDOG, and that in these cases the use of π_2 leads to better results. Thus DSSM only uses the permutation π_1 generated by SLOG if (6.1) fails.

Subroutine SDPT implements the direct method of Powell and Toint [10]. As pointed out by Coleman and Moré [2], this method can be viewed as a graph algorithm that determines a mapping φ which assigns to each vertex v an integer $\varphi(v) \in \{1, 2, \ldots, |V|\}$ such that if w is adjacent to distinct vertices v_1 and v_2 with $\varphi(v_1) = \varphi(v_2)$ then

$$\varphi(w) < \varphi(v_1) = \varphi(v_2).$$

The kth stage of the direct method consists of the following four steps.

(1) Let U_k be the unassigned vertices. If U_k is empty then terminate the algorithm.
(2) Sort the vertices of $G[U_k]$ in decreasing order of degree in $G[U_k]$.
(3) Build a vertex set W_k by examining the vertices in U_k in the order determined in (2), and adding a vertex v to W_k if there is not a path in $G[U_k]$ between v and some vertex in W_k of length $l \leq 2$.
(4) For each $v \in W_k$ let $\varphi(v) = k$.

If the above algorithm is applied to the adjacency graph of A, then the mapping φ defines a partition of the columns of A such that if two columns in the same group have a nonzero in row k, then column k is in a previous group. The direct method can be viewed as a lower triangular substitution method for any permutation matrix π such that the sequence $\{\varphi(\pi(i))\}$ is nondecreasing. This claim will be established by showing that columns of L_π in the same group do not have a nonzero in the same row position. If columns i and j of L_π have a nonzero in row k then $k \geq \max(i, j)$ because L_π is lower triangular. Hence, by definition of π we must have

$$\varphi(\pi(k)) \geq \max(\varphi(\pi(i)), \varphi(\pi(j))).$$

On the other hand, if columns i and j of L_π are in the same group then $\varphi(\pi(i)) = \varphi(\pi(j))$, and the algorithm which determines φ guarantees that

$$\varphi(\pi(k)) < \varphi(\pi(i)) = \varphi(\pi(j)).$$

This contradiction shows that the direct method of Powell and Toint is actually a lower triangular substitution method.

It is not difficult to implement the direct method if we are given predecessor and successor adjacency lists. Step (3) is the most expensive because it requires

Table IV. DSSM output for Everstine Problems

			Direct	Indirect
n	*nnz*	*mingrp*	*maxgrp*	*maxgrp*
59	163	4	6	4
66	193	3	6	3
72	147	3	4	3
87	314	5	10	6
162	672	5	10	6
193	1843	12	27	17
198	795	5	10	6
209	976	7	13	9
221	925	5	10	6
234	534	3	5	4
245	853	6	10	7
307	1415	6	11	7
310	1379	5	10	6
346	1786	7	15	10
361	1657	5	10	7
419	1991	7	13	8
492	1824	5	9	5
503	3265	9	21	12
512	2007	7	13	8
592	2848	6	11	8
607	2869	6	13	8
758	3376	5	10	7
869	4077	6	12	8
878	4163	5	11	7
918	4151	6	11	7
992	8868	10	23	14
1005	4813	10	19	13
1007	4791	5	11	7
1242	5834	7	13	8
2680	13853	7	13	9

inspection of vertices 2 edges away from any given vertex $v \in U_k$. The precise amount of work is difficult to predict, but has turned out to be reasonable in all the practical cases tried.

7. NUMERICAL RESULTS

Table IV shows the results of using DSSM on the 30 sparsity patterns of the Everstine [4] collection. The patterns are for symmetric matrices with orders ranging from 59 to 2680. Table IV contains the order n of the matrix, the number *nnz* of nonzeros in the lower triangular part of the matrix, the lower bound *mingrp*, and the output values of *maxgrp* for both the direct and indirect methods in DSSM. The results show that on the Everstine problems the direct method in DSSM usually determines a partition with *maxgrp* about 50 percent higher than with the indirect method, while the indirect method usually requires one or two more groups than the bound specified by *mingrp*.

It is important to note that lower triangular substitution methods may not be able to determine a matrix with *mingrp* groups. For example, the construction in Theorem 7.2 in [2] shows that there are symmetric matrices A of order n with *mingrp* = 3 but such that any lower triangular substitution method requires at least $n^{1/3}$ groups to determine A.

The sparsity patterns of the Everstine problems are irregular, and this makes it difficult to decide if a lower triangular substitution method can determine the matrices with *mingrp* groups; for regular sparsity patterns this is sometimes

possible. For example, Goldfarb and Toint [6] have shown that for the pattern of the Hessian matrix of (4.1) and for other patterns arising from finite difference approximations to partial differential equations there is an indirect method, dependent on the stencil used in the finite difference scheme, which determines the matrix with *mingrp* groups.

It is also worthwhile noting that direct methods may not be able to take advantage of symmetry. For example, Coleman and Moré [2] have shown that if A is a symmetric band matrix such that

$$a_{ij} \neq 0 \Leftrightarrow |i - j| \leq \beta$$

then a direct method requires at least $2\beta + 1$ groups (an indirect method can determine A with $\beta + 1$ groups). The direct method has not taken advantage of symmetry since it is possible to determine any (unsymmetric) band matrix in $2\beta + 1$ groups. We conjecture that direct methods cannot take advantage of symmetry in the minimal surface problem. Since a lower bound on the number of groups needed to determine a general matrix A is the maximum number of nonzeros in any row of A, our conjecture is that any direct method for the minimal surface problem requires at least 9 groups. Interestingly enough, the subroutine package in [1] determines any band matrix in at most $2\beta + 1$ groups, and determines any matrix with the sparsity pattern of the minimal surface problem in 9 groups.

Finally, we turn to a brief comparison of our software with subroutines TD03A and TD03B in the Harwell subroutine library.

Subroutines DSSM and TD03A have the same purpose, but use different algorithms to analyze the sparsity pattern. On input both subroutines require the sparsity pattern for the lower triangular part of A. DSSM requires the pairs (i, j) for the nonzero elements, while TD03A requires a column-oriented definition with the restriction that $indrow(jpntr(j)) = j$. In terms of storage requirements, DSSM needs $2nnz + 10n$ integer words while TD03A needs $3nnz + 5n$ integer words. Another difference is that TD03A does not have the option to specify a direct method; as demonstrated in Section 4, this DSSM option is worthwhile.

We have also compared the number of groups required by DSSM and TD03A to estimate a given sparsity pattern. Our numerical results for various types of sparsity patterns show that the indirect method in DSSM often requires fewer groups but never requires more groups than TD03A. In particular, on the Everstine problems DSSM obtains a partition with fewer groups on 19 problems. This represents an average improvement of 10 percent.

Subroutines FDHS and TD03B compute the approximate Hessian matrix but use different mechanisms to obtain the gradient information. FDHS uses a reverse communication interface, while with TD03B the user supplies a subroutine which evaluates the gradient. FDHS needs $2nnz + 5n$ integer words and $nnz + 2n$ working precision words of storage, while TD03B needs $3nnz + 4n$ integer words and $nnz + 5n$ working precision words.

Our initial testing revealed that the accuracies of the approximate Hessian matrices determined by FDHS (indirect method option) and TD03B are similar if the difference parameters are equal, but that FDHS provides a significantly

more accurate approximation to the Hessian matrix if the parameters are different. In particular, for the minimal surface problem with difference parameters (4.2), the errors for TD03B are just slightly larger than those in Table II; however, for choice (4.3) the errors for TD03B are at least 100 times larger than those in Table III. Alerted to this comparison, Philippe Toint discovered and corrected an error in TD03B. For the minimal surface problem with difference parameters (4.2), the errors for the corrected TD03B are now similar to those in Table II. The errors for choice (4.3) have decreased, but they are still significantly larger than those in Table III; in particular, the relative errors for $n = 1600$ and $n = 2500$ are at least 100 times larger.

Finally, let us mention the length of the source programs. The five new subroutines needed to implement lower triangular substitution methods contain 1290 lines (rounded to the nearest multiple of 10) of which 800 are comments. We also need seven routines from the subroutine package of Coleman, Garbow, and Moré [1] which contain 1030 lines of which 730 are comments. In contrast, the subroutines in the TD03A/TD03B package contain 890 lines of which 480 are comments.

REFERENCES

1. COLEMAN, T. F., GARBOW, B. S., AND MORÉ, J. J. Software for the estimation of sparse Jacobian matrices. *ACM Trans. Math. Softw. 10*, 3 (Sept. 1984), 329–345.
2. COLEMAN, T. F. AND MORÉ, J. J. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program. 28*, (Apr. 1984), 243–270.
3. CURTIS, A. R. AND REID, J. K. The choice of step lengths when using differences to approximate Jacobian matrices. *J. Inst. Math. Appl. 13*, (Feb. 1974), 121–126.
4. EVERSTINE, G. C. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront. *Int. J. Numer. Meth. Eng. 14*, (June 1980), 837–853.
5. GILL, P. E., MURRAY, W. AND WRIGHT, M. H. *Practical Optimization*. Academic Press, New York, 1981.
6. GOLDFARB, D. AND TOINT, P. L. Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations. *Math. Comp. 43*, (July 1984), 69–88.
7. GRIEWANK, A. AND TOINT, P. L. Partitioned variable metric updates for large structured optimization problems. *Numer. Math. 39*, (June 1982), 119–137.
8. MATULA, D. W. A min-max theorem for graphs with application to graph coloring. *SIAM Rev. 10*, (Oct. 1968), 481–482.
9. MATULA, D. W. AND BECK, L. L. Smallest-last ordering and clustering and graph coloring algorithms, *J.A.C.M. 30*, (July 1983), 417–427.
10. POWELL, M. J. D. AND TOINT, P. L. On the estimation of sparse Hessian matrices, *SIAM J. Numer. Anal. 16*, (Dec. 1979), 1060–1074.

Received January 1985; accepted September 1985.