Software For Estimating Sparse Hessian Matrices*

Thomas F. Coleman[1]
Burton S. Garbow[2]
Jorge J. Moré[2]

TR 85-660
January 1985

Department of Computer Science
Cornell University
Ithaca, New York 14853

---

* Also available as Technical Memorundum No. 43, Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL 60439.

1 Dept of Computer Science, Cornell University, Ithaca, NY 14853.

2 Argonne National Laboratory, Argonne, IL 60439.

ABSTRACT

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 Structures]: Graphs; E.2 [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

Software for Estimating Sparse Hessian Matrices

Thomas F. Coleman, † *Burton S. Garbow,* ‡ *and Jorge J. Moré* ‡

Cornell University and Argonne National Laboratory

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

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

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

---

† Department of Computer Science, Cornell University, Ithaca, New York 14853. Work 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.

‡ Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439. Work 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.

nonzero diagonal elements.

The algorithms that we have implemented are based on the work of Powell and Toint [1979], and Coleman and Moré [1984]. 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. Subroutine FDHS computes an approximation to the Hessian matrix of a mapping $f : R^n \rightarrow R$ 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 [1984] for estimating sparse Jacobian matrices.

Numerical results are presented in Section 7. We discuss the performance of DSSM on the Everstine [1979] 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:

$$(2.1) \qquad (indrow(k), indcol(k)), \quad k = 1,2,\ldots,npairs \; .$$

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 j-th 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 nonzeroes 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% 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 nonzeroes in the i-th 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 the number of operations needed to evaluate the Hessian matrix by differences is on the order of (2.2) when the mapping f is quadratic. 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 proportional to (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 j-th column are

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

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

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

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

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

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, Curtis and Reid [1974], and Gill, Murray, and Wright [1981]. 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

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

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 discretize this problem as done by Griewank and Toint [1982]. 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

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

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 4.1 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* 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*.

```
      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 4.1

Table 4.1 shows that on this problem the *maxgrp* values for the direct method (*method* $= 1$) are about 50% 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.

			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

Table 4.1. Output from DSSM for Minimal Surface Problem.

The timing results in Table 4.1 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 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 4.2 stores the approximation in the array *fhes*.

```
      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 4.2

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 4.2 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

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

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 4.2 with difference parameters

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

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

Table 4.2 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 4.2 show that in this case the additional loss of accuracy is not severe.

	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

Table 4.2. Output from FDHS.

It is important to note that the loss of accuracy in an indirect method can be severe if the difference parameters vary significantly in magnitude. Consider, for example, the choice of difference parameters

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

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

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

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 4.3 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. 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.

	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

Table 4.3. Output from FDHS.

The results of Table 4.3 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 [1984] 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 of Coleman, Garbow, and Moré [1984]. Given row-oriented and column-oriented definitions 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, Coleman, Garbow and Moré [1984].

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 k-th row can then be determined in three steps.

a) Let $i = \pi^{-1}(k)$.

b) Find the positions of the elements in the i-th row of A that have been determined.

c) For each element a_{ij} that has not been determined, let C be the group of column j and form

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

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

Implementation of these steps needs a bit of clarification and care. Step a) requires the inversion of the permutation π. In step b) we can find out if an element a_{ij} has been determined by checking that $\pi(i) \geq \pi(j)$. 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 b), the sum σ_{ij} in step c) 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 non-empty 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 non-empty 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 j-th vertex are

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

See, for example, the description of Matula and Beck [1983] 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

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

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

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

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 nonzeroes in any row of L_π. Matula [1968], and Powell and Toint [1979], 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 [1979]. As pointed out by Coleman and Moré [1984], this method can be viewed as a graph algorithm that determines a mapping ϕ which assigns to each vertex v an integer $\phi(v) \in \{1,2, \ldots, |V|\}$ such that if w is adjacent to distinct vertices v_1 and v_2 with $\phi(v_1) = \phi(v_2)$ then

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

The k-th stage of the direct method consists of the following four steps.

a) Let U_k be the un-assigned vertices. If U_k is empty then terminate the algorithm.

b) Sort the vertices of $G[U_k]$ in decreasing order of degree in $G[U_k]$.

c) Build a vertex set W_k by examining the vertices in U_k in the order determined in b), 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 $l \leq 2$.

d) For each $v \in W_k$ let $\phi(v) = k$.

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 $\{\phi(\pi(i))\}$ 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

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

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

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

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 c) is the most expensive because it requires 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 7.1 shows the results of using DSSM on the 30 sparsity patterns of the Everstine [1979] collection. The patterns are for symmetric matrices with orders ranging from 59 to 2680. Table 7.1 contains the order n of the matrix, the number nnz of nonzeroes 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% 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 of Coleman and Moré [1984] 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 [1984] 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 [1984] have shown that if A is a symmetric band matrix such that

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

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

n	nnz	mingrp	Direct maxgrp	Indirect 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

Table 7.1 DSSM output for Everstine Problems.

nonzeroes 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 of Coleman, Garbow, and Moré [1984] 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%

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.

The accuracies of the approximate Hessian matrices determined by FDHS (indirect method option) and TD03B are similar if the difference parameters are equal, but if the parameters are different then FDHS provides a significantly more accurate approximation to the Hessian matrix. In particular, for the minimal surface problem with difference parameters (4.2), the errors for TD03B are just slightly larger than those in Table 4.2. However, for choice (4.3) the errors for TD03B are at least 100 times larger than those in Table 4.3.

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 [1984] 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.

8. References.

Coleman, T. F., Garbow, B. S., and Moré, J. J. [1984]. Software for the estimation of sparse Jacobian matrices, ACM Trans. Math. Software 10, 329-345.

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

Curtis, A. R. and Reid, J. K. [1974]. The choice of step lengths when using differences to approximate Jacobian matrices, J. Inst. Math. Appl. 13, 121-126.

Everstine, G. C. [1979]. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront, Internat. J. Numer. Methods Engrg. 14, 837-853.

Gill, P. E., Murray, W. and Wright, M. H. [1981]. Practical Optimization, Academic Press.

Goldfarb, D. and Toint, P. L. [1984]. Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations, Math. Comp. 43, 69-88.

Griewank, A. and Toint, P. L. [1982]. Partitioned variable metric updates for large structured optimization problems, Numer. Math. 39, 119-137.

Matula, D. W. [1968]. A min-max theorem for graphs with application to graph coloring, SIAM Rev. 10, 481-482.

Matula, D. W. and Beck, L. L. [1983]. Smallest-last ordering and clustering and graph coloring algorithms, J. Assoc. Comput. Mach. 30, 417-427.

Powell, M. J. D. and Toint, Ph. L. [1979]. On the estimation of sparse Hessian matrices, SIAM J. Numer. Anal. 16, 1060-1074.

```
      subroutine dssm(n,npairs,indrow,indcol,method,listp,ngrp,
     *                maxgrp,mingrp,info,ipntr,jpntr,iwa,liwa)
      integer n,npairs,method,maxgrp,mingrp,info,liwa
      integer indrow(npairs),indcol(npairs),listp(n),ngrp(n),
     *        ipntr(n+1),jpntr(n+1),iwa(liwa)
c     **********
c
c     subroutine dssm
c
c     Given the sparsity pattern of a symmetric matrix A of order n,
c     this subroutine determines a symmetric permutation of A and a
c     partition of the columns of A consistent with the determination
c     of A by a lower triangular substitution method.
c
c     The sparsity pattern of the matrix A is specified by the
c     arrays indrow and indcol. On input the indices for the
c     non-zero elements in the lower triangular part of A are
c
c           (indrow(k),indcol(k)), k = 1,2,...,npairs.
c
c     The (indrow(k),indcol(k)) pairs may be specified in any order.
c     Duplicate input pairs are permitted, but the subroutine
c     eliminates them. The subroutine requires that all the diagonal
c     elements be part of the sparsity pattern and replaces any pair
c     (indrow(k),indcol(k)) where indrow(k) is less than indcol(k)
c     by the pair (indcol(k),indrow(k)).
c
c     The direct method (method = 1) first determines a partition
c     of the columns of A such that two columns in a group have a
c     non-zero element in row k only if column k is in an earlier
c     group. Using this partition, the subroutine then computes a
c     symmetric permutation of A consistent with the determination
c     of A by a lower triangular substitution method.
c
c     The indirect method first computes a symmetric permutation of A
c     which minimizes the maximum number of non-zero elements in any
c     row of L, where L is the lower triangular part of the permuted
c     matrix. The subroutine then partitions the columns of L into
c     groups such that columns of L in a group do not have a non-zero
c     in the same row position.
c
c     The subroutine statement is
c
c       subroutine dssm(n,npairs,indrow,indcol,method,listp,ngrp,
c                       maxgrp,mingrp,info,ipntr,jpntr,iwa,liwa)
c
c     where
c
c       n is a positive integer input variable set to the order of A.
c
c       npairs is a positive integer input variable set to the number
c         of (indrow,indcol) pairs used to describe the sparsity
c         pattern of A.
c
```

```
c       indrow is an integer array of length npairs. On input indrow
c         must contain the row indices of the non-zero elements in
c         the lower triangular part of A. On output indrow is
c         permuted so that the corresponding column indices are in
c         non-decreasing order. The column indices can be recovered
c         from the array jpntr.
c
c       indcol is an integer array of length npairs. On input indcol
c         must contain the column indices of the non-zero elements
c         in the lower triangular part of A. On output indcol is
c         permuted so that the corresponding row indices are in
c         non-decreasing order. The row indices can be recovered
c         from the array ipntr.
c
c       method is an integer input variable. If method = 1, the
c         direct method is used to determine the partition and
c         symmetric permutation. Otherwise, the indirect method is
c         used to determine the symmetric permutation and partition.
c
c       listp is an integer output array of length n which specifies
c         the symmetric permutation of the matrix A. Element (i,j)
c         of A is the (listp(i),listp(j)) element of the permuted
c         matrix.
c
c       ngrp is an integer output array of length n which specifies
c         the partition of the columns of A. Column j belongs to
c         group ngrp(j).
c
c       maxgrp is an integer output variable which specifies the
c         number of groups in the partition of the columns of A.
c
c       mingrp is an integer output variable which specifies a lower
c         bound for the number of groups in any partition of the
c         columns of A consistent with the determination of A by a
c         lower triangular substitution method.
c
c       info is an integer output variable set as follows. For
c         normal termination info = 1. If n or npairs is not
c         positive or liwa is less than 6*n, then info = 0. If the
c         k-th element of indrow or the k-th element of indcol is
c         not an integer between 1 and n, or if the k-th diagonal
c         element is not in the sparsity pattern, then info = -k.
c
c       ipntr is an integer output array of length n + 1 which
c         specifies the locations of the column indices in indcol.
c         The column indices for row i are
c
c               indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c         Note that ipntr(n+1)-1 is then the number of non-zero
c         elements in the lower triangular part of the matrix A.
c
c       jpntr is an integer output array of length n + 1 which
c         specifies the locations of the row indices in indrow.
```

```
c           The row indices for column j are
c
c                 indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c           Note that jpntr(n+1)-1 is then the number of non-zero
c           elements in the lower triangular part of the matrix A.
c
c         iwa is an integer work array of length liwa.
c
c         liwa is a positive integer input variable not less than 6*n.
c
c       Subprograms called
c
c         MINPACK-supplied ... degr,ido,idog,numsrt,sdpt,seq,setr,
c                              slo,slog,srtdat
c
c         FORTRAN-supplied ... max,min
c
c       Argonne National Laboratory. MINPACK Project. December 1984.
c       Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c       **********
        integer i,ir,j,jp,k,maxid,maxvd,maxclq,nnz,numgrp
c
c       Check the input data.
c
        info = 0
        if (n .lt. 1 .or. npairs .lt. 1 .or. liwa .lt. 6*n) return
        do 10 k = 1, n
           iwa(k) = 0
   10      continue
        do 20 k = 1, npairs
           info = -k
           if (indrow(k) .lt. 1 .or. indrow(k) .gt. n .or.
     *         indcol(k) .lt. 1 .or. indcol(k) .gt. n) return
           if (indrow(k) .eq. indcol(k)) iwa(indrow(k)) = 1
   20      continue
        do 30 k = 1, n
           info = -k
           if (iwa(k) .ne. 1) return
   30      continue
        info = 1
c
c       Generate the sparsity pattern for the lower
c       triangular part of A.
c
        do 40 k = 1, npairs
           i = indrow(k)
           j = indcol(k)
           indrow(k) = max(i,j)
           indcol(k) = min(i,j)
   40      continue
c
c       Sort the data structure by columns.
```

```
c
      call srtdat(n,npairs,indrow,indcol,jpntr,iwa)
c
c     Compress the data and determine the number of non-zero
c     elements in the lower triangular part of A.
c
      do 50 i = 1, n
         iwa(i) = 0
   50    continue
      nnz = 0
      do 70 j = 1, n
         k = nnz
         do 60 jp = jpntr(j), jpntr(j+1)-1
            ir = indrow(jp)
            if (iwa(ir) .ne. j) then
               nnz = nnz + 1
               indrow(nnz) = ir
               iwa(ir) = j
               end if
   60       continue
         jpntr(j) = k + 1
   70    continue
      jpntr(n+1) = nnz + 1
c
c     Extend the data structure to rows.
c
      call setr(n,n,indrow,jpntr,indcol,ipntr,iwa)
c
c     Determine the smallest-last ordering of the vertices of the
c     adjacency graph of A, and from it determine a lower bound
c     for the number of groups.
c
      call slog(n,indrow,jpntr,indcol,ipntr,iwa(1),maxclq,
     *          maxvd,iwa(n+1),iwa(2*n+1),iwa(3*n+1))
      mingrp = 1 + maxvd
c
c     Use the selected method.
c
      if (method .eq. 1) then
c
c        Direct method. Determine a partition of the columns
c        of A by the Powell-Toint method.
c
         call sdpt(n,indrow,jpntr,indcol,ipntr,ngrp,maxgrp,
     *             iwa(n+1),iwa(2*n+1))
c
c        Define a symmetric permutation of A according to the
c        ordering of the column group numbers in the partition.
c
         call numsrt(n,maxgrp,ngrp,1,iwa(1),iwa(2*n+1),iwa(n+1))
         do 80 i = 1, n
            listp(iwa(i)) = i
   80       continue
      else
```

```
c
c        Indirect method. Determine the incidence degree ordering
c        of the vertices of the adjacency graph of A and, together
c        with the smallest-last ordering, define a symmetric
c        permutation of A.
c
         call idog(n,indrow,jpntr,indcol,ipntr,listp,maxclq,
     *             maxid,iwa(n+1),iwa(2*n+1),iwa(3*n+1))
         if (maxid .gt. maxvd) then
            do 90 i = 1, n
               listp(i) = iwa(i)
   90          continue
            end if
c
c        Generate the sparsity pattern for the lower
c        triangular part L of the permuted matrix.
c
         do 110 j = 1, n
            do 100 jp = jpntr(j), jpntr(j+1)-1
               i = indrow(jp)
               indrow(jp) = max(listp(i),listp(j))
               indcol(jp) = min(listp(i),listp(j))
  100          continue
  110       continue
c
c        Sort the data structure by columns.
c
         call srtdat(n,nnz,indrow,indcol,jpntr,iwa)
c
c        Extend the data structure to rows.
c
         call setr(n,n,indrow,jpntr,indcol,ipntr,iwa)
c
c        Determine the degree sequence for the intersection
c        graph of the columns of L.
c
         call degr(n,indrow,jpntr,indcol,ipntr,iwa(5*n+1),iwa(n+1))
c
c        Color the intersection graph of the columns of L
c        with the smallest-last (SL) ordering.
c
         call slo(n,indrow,jpntr,indcol,ipntr,iwa(5*n+1),iwa(4*n+1),
     *            maxclq,iwa(1),iwa(n+1),iwa(2*n+1),iwa(3*n+1))
         call seq(n,indrow,jpntr,indcol,ipntr,iwa(4*n+1),iwa(1),
     *            maxgrp,iwa(n+1))
         do 120 j = 1, n
            ngrp(j) = iwa(listp(j))
  120       continue
c
c        Exit if the smallest-last ordering is optimal.
c
         if (maxgrp .eq. maxclq) go to 140
c
c        Color the intersection graph of the columns of L
```

```
c           with the incidence degree (ID) ordering.
c
            call ido(n,n,indrow,jpntr,indcol,ipntr,iwa(5*n+1),iwa(4*n+1),
     *               maxclq,iwa(1),iwa(n+1),iwa(2*n+1),iwa(3*n+1))
            call seq(n,indrow,jpntr,indcol,ipntr,iwa(4*n+1),iwa(1),
     *               numgrp,iwa(n+1))
c
c           Retain the better of the two orderings.
c
            if (numgrp .lt. maxgrp) then
               maxgrp = numgrp
               do 130 j = 1, n
                  ngrp(j) = iwa(listp(j))
  130             continue
               end if
  140       continue
c
c           Generate the sparsity pattern for the lower
c           triangular part of the original matrix.
c
            do 150 j = 1, n
               iwa(listp(j)) = j
  150          continue
            do 170 j = 1, n
               do 160 jp = jpntr(j), jpntr(j+1)-1
                  i = indrow(jp)
                  indrow(jp) = max(iwa(i),iwa(j))
                  indcol(jp) = min(iwa(i),iwa(j))
  160             continue
  170          continue
c
c           Sort the data structure by columns.
c
            call srtdat(n,nnz,indrow,indcol,jpntr,iwa)
c
c           Extend the data structure to rows.
c
            call setr(n,n,indrow,jpntr,indcol,ipntr,iwa)
            end if
         return
c
c        Last card of subroutine dssm.
c
         end
```

```
      subroutine idog(n,nghbrp,npntrp,nghbrs,npntrs,listp,
     *                maxclq,maxid,iwa1,iwa2,iwa3)
      integer n,maxclq,maxid
      integer nghbrp(*),npntrp(n+1),nghbrs(*),npntrs(n+1),listp(n),
     *        iwa1(0:n-1),iwa2(n),iwa3(n)
c     **********
c
c     subroutine idog
c
c     Given a loopless graph G = (V,E), this subroutine determines
c     the incidence degree ordering of the vertices of G.
c
c     The incidence degree ordering is determined recursively by
c     letting list(k), k = 1,...,n be a vertex with maximal
c     incidence to the subgraph spanned by the ordered vertices.
c     Among all the vertices of maximal incidence, a vertex of
c     maximal degree is chosen. This subroutine determines the
c     inverse of the incidence degree ordering, that is, an array
c     listp such that listp(list(k)) = k for k = 1,2,...,n.
c
c     The subroutine statement is
c
c       subroutine idog(n,nghbrp,npntrp,nghbrs,npntrs,listp,
c                       maxclq,maxid,iwa1,iwa2,iwa3)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of vertices of G.
c
c       nghbrp is an integer input array which contains the
c         predecessor adjacency lists for the graph G.
c
c       npntrp is an integer input array of length n + 1 which
c         specifies the locations of the predecessor adjacency
c         lists in nghbrp. The vertices preceding and adjacent
c         to vertex j are
c
c               nghbrp(k), k = npntrp(j),...,npntrp(j+1)-1.
c
c         Note that npntrp(n+1)-1 is then the number of vertices
c         plus edges of the graph G.
c
c       nghbrs is an integer input array which contains the
c         successor adjacency lists for the graph G.
c
c       npntrs is an integer input array of length n + 1 which
c         specifies the locations of the successor adjacency
c         lists in nghbrs. The vertices succeeding and adjacent
c         to vertex j are
c
c               nghbrs(k), k = npntrs(j),...,npntrs(j+1)-1.
c
c         Note that npntrs(n+1)-1 is then the number of vertices
```

```
c            plus edges of the graph G.
c
c         listp is an integer output array of length n which specifies
c           the inverse of the incidence degree ordering of the
c           vertices. Vertex j is in position listp(j) of this ordering.
c
c         maxclq is an integer output variable set to the size
c           of the largest clique found during the ordering.
c
c         maxid is an integer output variable set to the maximum
c           incidence degree found during the ordering.
c
c         iwa1,iwa2, and iwa3 are integer work arrays of length n.
c
c       Subprograms called
c
c         MINPACK-supplied ... numsrt
c
c         FORTRAN-supplied ... max
c
c       Argonne National Laboratory. MINPACK Project. December 1984.
c       Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c       **********
      integer i,j,k,maxinc,maxdeg,maxlst,ncomp,numdeg,numinc,numord
c
c       Initialization block.
c
      do 10 j = 1, n
         listp(j) = (npntrp(j+1) - npntrp(j) - 1) +
     *              (npntrs(j+1) - npntrs(j) - 1)
   10    continue
      maxlst = (npntrp(n+1) + npntrs(n+1))/n
c
c       Sort the degree sequence.
c
      call numsrt(n,n-1,listp,1,iwa1,iwa2,iwa3)
c
c       Create a doubly-linked list to access the incidences of the
c       vertices. The pointers for the linked list are as follows.
c
c       Each un-ordered vertex i is in a list (the incidence list)
c       of vertices with the same incidence.
c
c       iwa1(numinc) is the first vertex in the numinc list
c       unless iwa1(numinc) = 0. In this case there are
c       no vertices in the numinc list.
c
c       iwa2(i) is the vertex before i in the incidence list
c       unless iwa2(i) = 0. In this case i is the first
c       vertex in this incidence list.
c
c       iwa3(i) is the vertex after i in the incidence list
c       unless iwa3(i) = 0. In this case i is the last
```

```
c       vertex in this incidence list.
c
c       If i is an un-ordered vertex, then -listp(i) is the
c       incidence of i to the graph induced by the ordered
c       vertices. If j is an ordered vertex, then listp(j)
c       is the incidence degree order of vertex j.
c
        maxinc = 0
        do 20 j = 1, n
           i = iwa1(j-1)
           iwa1(j-1) = 0
           iwa2(i) = 0
           iwa3(i) = iwa1(0)
           if (iwa1(0) .gt. 0) iwa2(iwa1(0)) = i
           iwa1(0) = i
           listp(j) = 0
   20      continue
        maxclq = 0
        maxid = 0
        numord = 1
c
c       Beginning of iteration loop.
c
   30 continue
c
c          Choose a vertex j of maximal degree among the
c          vertices of maximal incidence maxinc.
c
   40      continue
              k = iwa1(maxinc)
              if (k .gt. 0) go to 50
              maxinc = maxinc - 1
              go to 40
   50      continue
           maxdeg = -1
           do 60 i = 1, maxlst
              numdeg = (npntrp(k+1) - npntrp(k) - 1) +
     *                 (npntrs(k+1) - npntrs(k) - 1)
              if (numdeg .gt. maxdeg) then
                 maxdeg = numdeg
                 j = k
                 end if
              k = iwa3(k)
              if (k .le. 0) go to 70
   60         continue
   70      continue
           listp(j) = numord
           maxid = max(maxid,maxinc)
c
c          Update the size of the largest clique
c          found during the ordering.
c
           if (maxinc .eq. 0) ncomp = 0
           ncomp = ncomp + 1
```

```
         if (maxinc+ 1 .eq. ncomp) maxclq = max(maxclq,ncomp)
c
c        Termination test.
c
         numord = numord + 1
         if (numord .gt. n) go to 100
c
c        Delete vertex j from the maxinc list.
c
         if (iwa2(j) .eq. 0) then
            iwa1(maxinc) = iwa3(j)
         else
            iwa3(iwa2(j)) = iwa3(j)
            end if
         if (iwa3(j) .gt. 0) iwa2(iwa3(j)) = iwa2(j)
c
c        Determine all the neighbors of vertex j which precede j
c        in the subgraph spanned by the un-ordered vertices.
c
         do 80 k = npntrp(j), npntrp(j+ 1)-1
            i = nghbrp(k)
c
c           Update the pointers to the current incidence lists.
c
            numinc = -listp(i)
            if (numinc .ge. 0) then
               listp(i) = listp(i) - 1
               maxinc = max(maxinc,-listp(i))
c
c              Delete vertex i from the numinc list.
c
               if (iwa2(i) .eq. 0) then
                  iwa1(numinc) = iwa3(i)
               else
                  iwa3(iwa2(i)) = iwa3(i)
                  end if
               if (iwa3(i) .gt. 0) iwa2(iwa3(i)) = iwa2(i)
c
c              Add vertex i to the numinc+ 1 list.
c
               iwa2(i) = 0
               iwa3(i) = iwa1(numinc+ 1)
               if (iwa1(numinc+ 1) .gt. 0) iwa2(iwa1(numinc+ 1)) = i
               iwa1(numinc+ 1) = i
               end if
   80       continue
c
c        Determine all the neighbors of vertex j which succeed j
c        in the subgraph spanned by the un-ordered vertices.
c
         do 90 k = npntrs(j), npntrs(j+ 1)-1
            i = nghbrs(k)
c
c           Update the pointers to the current incidence lists.
```

```
c
               numinc = -listp(i)
               if (numinc .ge. 0) then
                  listp(i) = listp(i) - 1
                  maxinc = max(maxinc,-listp(i))
c
c                 Delete vertex i from the numinc list.
c
                  if (iwa2(i) .eq. 0) then
                     iwa1(numinc) = iwa3(i)
                  else
                     iwa3(iwa2(i)) = iwa3(i)
                     end if
                  if (iwa3(i) .gt. 0) iwa2(iwa3(i)) = iwa2(i)
c
c                 Add vertex i to the numinc+1 list.
c
                  iwa2(i) = 0
                  iwa3(i) = iwa1(numinc+1)
                  if (iwa1(numinc+1) .gt. 0) iwa2(iwa1(numinc+1)) = i
                  iwa1(numinc+1) = i
                  end if
   90          continue
c
c        End of iteration loop.
c
         go to 30
  100 continue
      return
c
c     Last card of subroutine idog.
c
      end
```

```
      subroutine sdpt(n,nghbrp,npntrp,nghbrs,npntrs,ngrp,maxgrp,
     *                iwa1,iwa2)
      integer n,maxgrp
      integer nghbrp(*),npntrp(n+1),nghbrs(*),npntrs(n+1),ngrp(n),
     *        iwa1(0:n-1),iwa2(n)
c     **********
c
c     subroutine sdpt
c
c     Given a loopless graph G = (V,E), this subroutine determines
c     a symmetric coloring of G by the Powell-Toint direct method.
c
c     The Powell-Toint method assigns the k-th color by examining
c     the un-colored vertices U(k) in order of non-increasing degree
c     and assigning color k to vertex v if there are no paths of
c     length 1 or 2 (in the graph induced by U(k)) between v and
c     some k-colored vertex.
c
c     The subroutine statement is
c
c       subroutine sdpt(n,nghbrp,npntrp,nghbrs,npntrs,ngrp,maxgrp,
c                       iwa1,iwa2)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of vertices of G.
c
c       nghbrp is an integer input array which contains the
c         predecessor adjacency lists for the graph G.
c
c       npntrp is an integer input array of length n + 1 which
c         specifies the locations of the predecessor adjacency
c         lists in nghbrp. The vertices preceding and adjacent
c         to vertex j are
c
c               nghbrp(k), k = npntrp(j),...,npntrp(j+1)-1.
c
c         Note that npntrp(n+1)-1 is then the number of vertices
c         plus edges of the graph G.
c
c       nghbrs is an integer input array which contains the
c         successor adjacency lists for the graph G.
c
c       npntrs is an integer input array of length n + 1 which
c         specifies the locations of the successor adjacency
c         lists in nghbrs. The vertices succeeding and adjacent
c         to vertex j are
c
c               nghbrs(k), k = npntrs(j),...,npntrs(j+1)-1.
c
c         Note that npntrs(n+1)-1 is then the number of vertices
c         plus edges of the graph G.
c
```

```
c         ngrp is an integer output array of length n which specifies
c           the symmetric coloring of G. Vertex j is colored with
c           color ngrp(j).
c
c         maxgrp is an integer output variable which specifies the
c           number of colors in the symmetric coloring of G.
c
c         iwa1 and iwa2 are integer work arrays of length n.
c
c       Subprograms called
c
c         FORTRAN-supplied ... max
c
c       Argonne National Laboratory. MINPACK Project. December 1984.
c       Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c       **********
      integer j,jp,k,kp,l,maxdeg,numdeg,numv
c
c       Initialization block. Numv is the current number of un-colored
c       vertices, maxdeg is the maximum induced degree of these
c       vertices, and maxgrp is the current group number (color).
c
      numv = n
      maxdeg = 0
      do 10 j = 1, n
         ngrp(j) = (npntrp(j) - npntrp(j+1) + 1) +
     *             (npntrs(j) - npntrs(j+1) + 1)
         maxdeg = max(maxdeg,-ngrp(j))
         iwa2(j) = -j
   10    continue
      maxgrp = 0
c
c       Beginning of iteration loop.
c
   20 continue
c
c          Sort the list of un-colored vertices so that their
c          induced degrees are in non-decreasing order.
c
         do 30 numdeg = 0, maxdeg
            iwa1(numdeg) = 0
   30       continue
         do 40 l = 1, numv
            numdeg = -ngrp(-iwa2(l))
            iwa1(numdeg) = iwa1(numdeg) + 1
   40       continue
         k = 1
         do 50 numdeg = maxdeg, 0, -1
            l = iwa1(numdeg)
            iwa1(numdeg) = k
            k = k + l
   50       continue
         k = 1
```

```
   60    continue
            j = iwa2(k)
            if (j .gt. 0) then
               k = iwa1(-ngrp(j))
            else
               numdeg = -ngrp(-j)
               l = iwa1(numdeg)
               iwa2(k) = iwa2(l)
               iwa2(l) = -j
               iwa1(numdeg) = iwa1(numdeg) + 1
               end if
            if (k .le. numv) go to 60
         maxgrp = maxgrp + 1
c
c        Determine the vertices in group maxgrp.
c
         do 160 l = 1, numv
            j = iwa2(l)
c
c           Examine each vertex k preceding vertex j and all
c           the neighbors of vertex k to determine if vertex
c           j can be considered for group maxgrp.
c
            do 90 jp = npntrp(j), npntrp(j+1)-1
               k = nghbrp(jp)
               if (ngrp(k) .eq. maxgrp) go to 150
               if (ngrp(k) .le. 0) then
                  do 70 kp = npntrp(k), npntrp(k+1)-1
                     if (ngrp(nghbrp(kp)) .eq. maxgrp) go to 150
   70                continue
                  do 80 kp = npntrs(k), npntrs(k+1)-1
                     if (ngrp(nghbrs(kp)) .eq. maxgrp) go to 150
   80                continue
                  end if
   90          continue
c
c           Examine each vertex k succeeding vertex j and all
c           the neighbors of vertex k to determine if vertex
c           j can be added to group maxgrp.
c
            do 120 jp = npntrs(j), npntrs(j+1)-1
               k = nghbrs(jp)
               if (ngrp(k) .eq. maxgrp) go to 150
               if (ngrp(k) .le. 0) then
                  do 100 kp = npntrp(k), npntrp(k+1)-1
                     if (ngrp(nghbrp(kp)) .eq. maxgrp) go to 150
  100                continue
                  do 110 kp = npntrs(k), npntrs(k+1)-1
                     if (ngrp(nghbrs(kp)) .eq. maxgrp) go to 150
  110                continue
                  end if
  120          continue
c
c           Add vertex j to group maxgrp and remove vertex j
```

```
c           from the list of un-colored vertices.
c
            ngrp(j) = maxgrp
            iwa2(l) = 0
c
c           Update the degrees of the neighbors of vertex j.
c
            do 130 jp = npntrp(j), npntrp(j+1)-1
               k = nghbrp(jp)
               if (ngrp(k) .lt. 0) ngrp(k) = ngrp(k) + 1
  130          continue
            do 140 jp = npntrs(j), npntrs(j+1)-1
               k = nghbrs(jp)
               if (ngrp(k) .lt. 0) ngrp(k) = ngrp(k) + 1
  140          continue
  150       continue
  160       continue
c
c        Compress the updated list of un-colored vertices.
c        Reset numv and recompute maxdeg.
c
         k = 0
         maxdeg = 0
         do 170 l = 1, numv
            if (iwa2(l) .ne. 0) then
               k = k + 1
               iwa2(k) = -iwa2(l)
               maxdeg = max(maxdeg,-ngrp(iwa2(l)))
               end if
  170       continue
         numv = k
c
c        End of iteration loop.
c
         if (numv .gt. 0) go to 20
      return
c
c     Last card of subroutine sdpt.
c
      end
```

```
      subroutine slog(n,nghbrp,npntrp,nghbrs,npntrs,listp,
     *                maxclq,maxvd,iwa1,iwa2,iwa3)
      integer n,maxclq,maxvd
      integer nghbrp(*),npntrp(n+1),nghbrs(*),npntrs(n+1),listp(n),
     *        iwa1(0:n-1),iwa2(n),iwa3(n)
c     **********
c
c     subroutine slog
c
c     Given a loopless graph G = (V,E), this subroutine determines
c     the smallest-last ordering of the vertices of G.
c
c     The smallest-last ordering is determined recursively by
c     letting list(k), k = n,...,1 be a vertex with least degree
c     in the subgraph spanned by the un-ordered vertices.
c     This subroutine determines the inverse of the smallest-last
c     ordering, that is, an array listp such that listp(list(k)) = k
c     for k = 1,2,...,n.
c
c     The subroutine statement is
c
c       subroutine slog(n,nghbrp,npntrp,nghbrs,npntrs,listp,
c                       maxclq,maxvd,iwa1,iwa2,iwa3)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of vertices of G.
c
c       nghbrp is an integer input array which contains the
c         predecessor adjacency lists for the graph G.
c
c       npntrp is an integer input array of length n + 1 which
c         specifies the locations of the predecessor adjacency
c         lists in nghbrp. The vertices preceding and adjacent
c         to vertex j are
c
c               nghbrp(k), k = npntrp(j),...,npntrp(j+1)-1.
c
c         Note that npntrp(n+1)-1 is then the number of vertices
c         plus edges of the graph G.
c
c       nghbrs is an integer input array which contains the
c         successor adjacency lists for the graph G.
c
c       npntrs is an integer input array of length n + 1 which
c         specifies the locations of the successor adjacency
c         lists in nghbrs. The vertices succeeding and adjacent
c         to vertex j are
c
c               nghbrs(k), k = npntrs(j),...,npntrs(j+1)-1.
c
c         Note that npntrs(n+1)-1 is then the number of vertices
c         plus edges of the graph G.
```

```
c
c       listp is an integer output array of length n which specifies
c         the inverse of the smallest-last ordering of the vertices.
c         Vertex j is in position listp(j) of this ordering.
c
c       maxclq is an integer output variable set to the size
c         of the largest clique found during the ordering.
c
c       maxvd is an integer output variable set to the maximum
c         vertex degree found during the ordering.
c
c       iwa1,iwa2, and iwa3 are integer work arrays of length n.
c
c     Subprograms called
c
c       FORTRAN-supplied ... max,min
c
c     Argonne National Laboratory. MINPACK Project. December 1984.
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer i,j,k,mindeg,numdeg,numord
c
c     Initialization block.
c
      mindeg = n
      do 10 j = 1, n
         iwa1(j-1) = 0
         listp(j) = (npntrp(j) - npntrp(j+1) + 1) +
     *              (npntrs(j) - npntrs(j+1) + 1)
         mindeg = min(mindeg,-listp(j))
   10    continue
c
c     Create a doubly-linked list to access the degrees of the
c     vertices. The pointers for the linked list are as follows.
c
c     Each un-ordered vertex i is in a list (the degree list)
c     of vertices with the same degree.
c
c     iwa1(numdeg) is the first vertex in the numdeg list
c     unless iwa1(numdeg) = 0. In this case there are
c     no vertices in the numdeg list.
c
c     iwa2(i) is the vertex before i in the degree list
c     unless iwa2(i) = 0. In this case i is the first
c     vertex in this degree list.
c
c     iwa3(i) is the vertex after i in the degree list
c     unless iwa3(i) = 0. In this case i is the last
c     vertex in this degree list.
c
c     If i is an un-ordered vertex, then -listp(i) is the
c     degree of i in the graph induced by the un-ordered
c     vertices. If j is an ordered vertex, then listp(j)
```

```
c       is the smallest-last order of vertex j.
c
        do 20 j = 1, n
           numdeg = -listp(j)
           iwa2(j) = 0
           iwa3(j) = iwa1(numdeg)
           if (iwa1(numdeg) .gt. 0) iwa2(iwa1(numdeg)) = j
           iwa1(numdeg) = j
   20      continue
        maxclq = 0
        maxvd = 0
        numord = n
c
c       Beginning of iteration loop.
c
   30 continue
c
c          Choose a vertex j of minimal degree mindeg.
c
   40      continue
              j = iwa1(mindeg)
              if (j .gt. 0) go to 50
              mindeg = mindeg + 1
              go to 40
   50      continue
           listp(j) = numord
           maxvd = max(maxvd,mindeg)
c
c          Mark the size of the largest clique
c          found during the ordering.
c
           if (mindeg+1 .eq. numord .and. maxclq .eq. 0)
     *        maxclq = numord
c
c          Termination test.
c
           numord = numord - 1
           if (numord .eq. 0) go to 80
c
c          Delete vertex j from the mindeg list.
c
           iwa1(mindeg) = iwa3(j)
           if (iwa3(j) .gt. 0) iwa2(iwa3(j)) = 0
c
c          Determine all the neighbors of vertex j which precede j
c          in the subgraph spanned by the un-ordered vertices.
c
           do 60 k = npntrp(j), npntrp(j+1)-1
              i = nghbrp(k)
c
c             Update the pointers to the current degree lists.
c
              numdeg = -listp(i)
              if (numdeg .ge. 0) then
```

```
               listp(i) = listp(i) + 1
               mindeg = min(mindeg,-listp(i))
c
c              Delete vertex i from the numdeg list.
c
               if (iwa2(i) .eq. 0) then
                  iwa1(numdeg) = iwa3(i)
               else
                  iwa3(iwa2(i)) = iwa3(i)
                  end if
               if (iwa3(i) .gt. 0) iwa2(iwa3(i)) = iwa2(i)
c
c              Add vertex i to the numdeg-1 list.
c
               iwa2(i) = 0
               iwa3(i) = iwa1(numdeg-1)
               if (iwa1(numdeg-1) .gt. 0) iwa2(iwa1(numdeg-1)) = i
               iwa1(numdeg-1) = i
               end if
   60       continue
c
c        Determine all the neighbors of vertex j which succeed j
c        in the subgraph spanned by the un-ordered vertices.
c
         do 70 k = npntrs(j), npntrs(j+1)-1
            i = nghbrs(k)
c
c           Update the pointers to the current degree lists.
c
            numdeg = -listp(i)
            if (numdeg .ge. 0) then
               listp(i) = listp(i) + 1
               mindeg = min(mindeg,-listp(i))
c
c              Delete vertex i from the numdeg list.
c
               if (iwa2(i) .eq. 0) then
                  iwa1(numdeg) = iwa3(i)
               else
                  iwa3(iwa2(i)) = iwa3(i)
                  end if
               if (iwa3(i) .gt. 0) iwa2(iwa3(i)) = iwa2(i)
c
c              Add vertex i to the numdeg-1 list.
c
               iwa2(i) = 0
               iwa3(i) = iwa1(numdeg-1)
               if (iwa1(numdeg-1) .gt. 0) iwa2(iwa1(numdeg-1)) = i
               iwa1(numdeg-1) = i
               end if
   70       continue
c
c        End of iteration loop.
c
```

```
      go to 30
   80 continue
      return
c
c     Last card of subroutine slog.
c
      end
```

```
      subroutine fdhs(n,indrow,jpntr,indcol,ipntr,listp,ngrp,
     *                maxgrp,numgrp,eta,fhesd,fhes,iwa)
      integer n,maxgrp,numgrp
      integer indrow(*),jpntr(n+1),indcol(*),ipntr(n+1),
     *        listp(n),ngrp(n),iwa(n)
      real eta(n),fhesd(n),fhes(*)
c     **********
c
c     subroutine fdhs
c
c     This subroutine computes an approximation to the (symmetric)
c     Hessian matrix of a function by a substitution method.
c     The lower triangular part of the approximation is stored
c     with a column-oriented definition of the sparsity pattern.
c
c     This subroutine requires a symmetric permutation of the
c     Hessian matrix and a partition of the columns of the Hessian
c     matrix consistent with the determination of the Hessian
c     matrix by a lower triangular substitution method.
c     This information can be provided by subroutine dssm.
c
c     The symmetric permutation of the Hessian matrix is defined
c     by the array listp. This array is only used internally.
c
c     The partition of the Hessian matrix is defined by the array
c     ngrp by setting ngrp(j) to the group number of column j.
c     The user must provide an approximation to the columns of
c     the Hessian matrix in each group by specifying a difference
c     parameter vector eta and an approximation to H*d where H is
c     the Hessian matrix and the vector d is defined by the
c     following section of code.
c
c           do 10 j = 1, n
c              d(j) = 0.0
c              if (ngrp(j) .eq. numgrp) d(j) = eta(j)
c        10    continue
c
c     In the above code numgrp is a group number and eta(j) is the
c     difference parameter used to approximate column j of the
c     Hessian matrix. Suitable values for eta(j) must be provided.
c
c     As mentioned above, an approximation to H*d must be provided.
c     For example, if grad(x) is the gradient of the function at x,
c     then
c
c           grad(x+d) - grad(x)
c
c     corresponds to the forward difference approximation.
c
c     The lower triangular substitution method requires that the
c     approximations to H*d for all the groups be stored in special
c     locations of the array fhes. This is done by calling fdhs
c     successively with numgrp = 1,2,...,maxgrp. On the call with
c     numgrp = maxgrp, the subroutine then proceeds to overwrite
```

```
c       fhes with the approximation to the lower triangular part of
c       the Hessian matrix.
c
c       The subroutine statement is
c
c         subroutine fdhs(n,indrow,jpntr,indcol,ipntr,listp,ngrp,
c                         maxgrp,numgrp,eta,fhesd,fhes,iwa)
c
c       where
c
c         n is a positive integer input variable set to the order
c           of the Hessian matrix.
c
c         indrow is an integer input array which contains the row
c           indices for the non-zeroes in the lower triangular part
c           of the Hessian matrix.
c
c         jpntr is an integer input array of length n + 1 which
c           specifies the locations of the row indices in indrow.
c           The row indices for column j are
c
c                 indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c           Note that jpntr(n+1)-1 is then the number of non-zeroes
c           in the lower triangular part of the Hessian matrix.
c
c         indcol is an integer input array which contains the column
c           indices for the non-zeroes in the lower triangular part
c           of the Hessian matrix.
c
c         ipntr is an integer input array of length n + 1 which
c           specifies the locations of the column indices in indcol.
c           The column indices for row i are
c
c                 indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c           Note that ipntr(n+1)-1 is then the number of non-zeroes
c           in the lower triangular part of the Hessian matrix.
c
c         listp is an integer input array of length n which specifies
c           the symmetric permutation of the Hessian matrix. Element
c           (i,j) of the Hessian matrix is the (listp(i),listp(j))
c           element of the permuted Hessian.
c
c         ngrp is an integer input array of length n which specifies
c           the partition of the columns of the Hessian matrix.
c           Column j belongs to group ngrp(j).
c
c         maxgrp is a positive integer input variable which specifies
c           the number of groups in the partition of the columns of
c           the Hessian matrix.
c
c         numgrp is a positive integer input variable set to a group
c           number in the partition.
```

```
c
c         eta is an input array of length n which contains the
c           difference parameter vector.
c
c         fhesd is an input array of length n which contains an
c           approximation to H*d, where H is the Hessian matrix
c           and d is the difference vector for group numgrp.
c
c         fhes is an output array of length nnz, where nnz is the
c           number of non-zero elements in the lower triangular part
c           of the Hessian matrix. On output with numgrp less than
c           maxgrp, the fhesd array for group numgrp has been stored
c           in fhes. When numgrp = maxgrp the subroutine overwrites
c           fhes with an approximation to the lower triangular part
c           of the Hessian matrix. The approximation is stored in
c           fhes with a column-oriented definition of the sparsity
c           pattern. Thus the elements in column j of the lower
c           triangular part of the Hessian matrix are
c
c                 fhes(k), k = jpntr(j),...,jpntr(j+1)-1,
c
c           and the row indices for these elements are
c
c                 indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         iwa is an integer work array of length n.
c
c       Subprograms called
c
c         FORTRAN-supplied ... abs
c
c       Argonne National Laboratory. MINPACK Project. December 1984.
c       Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c       **********
        integer i,ip,irow,j,jp,k,l,numg,numl
        real sum
c
c       Store the i-th element of gradient difference fhesd
c       corresponding to group numgrp if there is a position
c       (i,j) such that ngrp(j) = numgrp and (i,j) is mapped
c       onto the lower triangular part of the permuted matrix.
c
        do 50 j = 1, n
           if (ngrp(j) .eq. numgrp) then
              numl = listp(j)
              do 30 ip = ipntr(j), ipntr(j+1)-1
                 i = indcol(ip)
                 if (listp(i) .gt. numl) then
                    do 10 jp = jpntr(i), jpntr(i+1)-1
                       if (indrow(jp) .eq. j) then
                          fhes(jp) = fhesd(i)
                          go to 20
                          end if
```

```
   10                   continue
   20                continue
                     end if
   30             continue
               do 40 jp = jpntr(j), jpntr(j+1)-1
                  i = indrow(jp)
                  if (listp(i) .ge. numl) fhes(jp) = fhesd(i)
   40             continue
               end if
   50       continue
c
c     Exit if this is not the last group.
c
      if (numgrp .lt. maxgrp) return
c
c     Mark all column indices j such that (i,j) is mapped onto
c     the lower triangular part of the permuted matrix.
c
      do 80 i = 1, n
         numl = listp(i)
         do 60 ip = ipntr(i), ipntr(i+1)-1
            j = indcol(ip)
            if (numl .ge. listp(j)) indcol(ip) = -indcol(ip)
   60       continue
         do 70 jp = jpntr(i), jpntr(i+1)-1
            j = indrow(jp)
            if (numl .gt. listp(j)) indrow(jp) = -indrow(jp)
   70       continue
   80    continue
c
c     Invert the array listp.
c
      do 90 j = 1, n
         iwa(listp(j)) = j
   90    continue
      do 100 j = 1, n
         listp(j) = iwa(j)
  100    continue
c
c     Determine the lower triangular part of the original matrix.
c
      do 220 irow = n, 1, -1
         i = listp(irow)
c
c        Find the positions of the elements in the i-th row of the
c        lower triangular part of the original matrix that have
c        already been determined.
c
         do 130 ip = ipntr(i), ipntr(i+1)-1
            j = indcol(ip)
            if (j .gt. 0) then
               do 110 jp = jpntr(j), jpntr(j+1)-1
                  if (indrow(jp) .eq. i) then
                     iwa(j) = jp
```

```
                      go to 120
                      end if
  110             continue
  120          continue
               end if
  130       continue
c
c        Determine the elements in the i-th row of the lower
c        triangular part of the original matrix which get mapped
c        onto the lower triangular part of the permuted matrix.
c
         do 180 k = ipntr(i), ipntr(i+1)-1
            j = -indcol(k)
            if (j .gt. 0) then
               indcol(k) = j
c
c              Determine the (i,j) element.
c
               numg = ngrp(j)
               sum = 0.0
               do 140 ip = ipntr(i), ipntr(i+1)-1
                  l = abs(indcol(ip))
                  if (ngrp(l) .eq. numg .and. l .ne. j)
     *               sum = sum + fhes(iwa(l))*eta(l)
  140             continue
               do 150 jp = jpntr(i), jpntr(i+1)-1
                  l = abs(indrow(jp))
                  if (ngrp(l) .eq. numg .and. l .ne. j)
     *               sum = sum + fhes(jp)*eta(l)
  150             continue
c
c              Store the (i,j) element.
c
               do 160 jp = jpntr(j), jpntr(j+1)-1
                  if (indrow(jp) .eq. i) then
                     fhes(jp) = (fhes(jp) - sum)/eta(j)
                     go to 170
                     end if
  160             continue
  170          continue
               end if
  180       continue
c
c        Determine the elements in the i-th row of the strict upper
c        triangular part of the original matrix which get mapped
c        onto the lower triangular part of the permuted matrix.
c
         do 210 k = jpntr(i), jpntr(i+1)-1
            j = -indrow(k)
            if (j .gt. 0) then
               indrow(k) = j
c
c              Determine the (i,j) element.
```

```
c
               numg = ngrp(j)
               sum = 0.0
               do 190 ip = ipntr(i), ipntr(i+ 1)-1
                  l = abs(indcol(ip))
                  if (ngrp(l) .eq. numg)
     *               sum = sum + fhes(iwa(l))*eta(l)
  190             continue
               do 200 jp = jpntr(i), jpntr(i+ 1)-1
                  l = abs(indrow(jp))
                  if (ngrp(l) .eq. numg .and. l .ne. j)
     *               sum = sum + fhes(jp)*eta(l)
  200             continue
c
c              Store the (i,j) element.
c
               fhes(k) = (fhes(k) - sum)/eta(j)
               end if
  210       continue
  220    continue
c
c     Re-invert the array listp.
c
      do 230 j = 1, n
         iwa(listp(j)) = j
  230    continue
      do 240 j = 1, n
         listp(j) = iwa(j)
  240    continue
      return
c
c     Last card of subroutine fdhs.
c
      end
```

```
      subroutine dsm(m,n,npairs,indrow,indcol,ngrp,maxgrp,mingrp,
     *               info,ipntr,jpntr,iwa,liwa)
      integer m,n,npairs,maxgrp,mingrp,info,liwa
      integer indrow(npairs),indcol(npairs),ngrp(n),
     *        ipntr(m+1),jpntr(n+1),iwa(liwa)
c     **********
c
c     subroutine dsm
c
c     Given the sparsity pattern of an m by n matrix A, this
c     subroutine determines a partition of the columns of A
c     consistent with the direct determination of A.
c
c     The sparsity pattern of the matrix A is specified by
c     the arrays indrow and indcol. On input the indices
c     for the non-zero elements of A are
c
c           indrow(k),indcol(k), k = 1,2,...,npairs.
c
c     The (indrow,indcol) pairs may be specified in any order.
c     Duplicate input pairs are permitted, but the subroutine
c     eliminates them.
c
c     The subroutine partitions the columns of A into groups
c     such that columns in the same group do not have a
c     non-zero in the same row position. A partition of the
c     columns of A with this property is consistent with the
c     direct determination of A.
c
c     The subroutine statement is
c
c       subroutine dsm(m,n,npairs,indrow,indcol,ngrp,maxgrp,mingrp,
c                      info,ipntr,jpntr,iwa,liwa)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of A.
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       npairs is a positive integer input variable set to the
c         number of (indrow,indcol) pairs used to describe the
c         sparsity pattern of A.
c
c       indrow is an integer array of length npairs. On input indrow
c         must contain the row indices of the non-zero elements of A.
c         On output indrow is permuted so that the corresponding
c         column indices are in non-decreasing order. The column
c         indices can be recovered from the array jpntr.
c
c       indcol is an integer array of length npairs. On input indcol
c         must contain the column indices of the non-zero elements of
```

```
c           A. On output indcol is permuted so that the corresponding
c           row indices are in non-decreasing order. The row indices
c           can be recovered from the array ipntr.
c
c         ngrp is an integer output array of length n which specifies
c           the partition of the columns of A. Column jcol belongs
c           to group ngrp(jcol).
c
c         maxgrp is an integer output variable which specifies the
c           number of groups in the partition of the columns of A.
c
c         mingrp is an integer output variable which specifies a lower
c           bound for the number of groups in any consistent partition
c           of the columns of A.
c
c         info is an integer output variable set as follows. For
c           normal termination info = 1. If m, n, or npairs is not
c           positive or liwa is less than max(m,6*n), then info = 0.
c           If the k-th element of indrow is not an integer between
c           1 and m or the k-th element of indcol is not an integer
c           between 1 and n, then info = -k.
c
c         ipntr is an integer output array of length m + 1 which
c           specifies the locations of the column indices in indcol.
c           The column indices for row i are
c
c                 indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c           Note that ipntr(m+1)-1 is then the number of non-zero
c           elements of the matrix A.
c
c         jpntr is an integer output array of length n + 1 which
c           specifies the locations of the row indices in indrow.
c           The row indices for column j are
c
c                 indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c           Note that jpntr(n+1)-1 is then the number of non-zero
c           elements of the matrix A.
c
c         iwa is an integer work array of length liwa.
c
c         liwa is a positive integer input variable not less than
c           max(m,6*n).
c
c       Subprograms called
c
c         MINPACK-supplied ... degr,ido,numsrt,seq,setr,slo,srtdat
c
c         FORTRAN-supplied ... max
c
c       Argonne National Laboratory. MINPACK Project. December 1984.
c       Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
```

```
c     **********
      integer i,ir,j,jp,k,maxclq,nnz,numgrp
c
c     Check the input data.
c
      info = 0
      if (m .lt. 1 .or. n .lt. 1 .or. npairs .lt. 1 .or.
     *    liwa .lt. max(m,6*n)) return
      do 10 k = 1, npairs
         info = -k
         if (indrow(k) .lt. 1 .or. indrow(k) .gt. m .or.
     *       indcol(k) .lt. 1 .or. indcol(k) .gt. n) return
   10    continue
      info = 1
c
c     Sort the data structure by columns.
c
      call srtdat(n,npairs,indrow,indcol,jpntr,iwa)
c
c     Compress the data and determine the number of
c     non-zero elements of A.
c
      do 20 i = 1, m
         iwa(i) = 0
   20    continue
      nnz = 0
      do 40 j = 1, n
         k = nnz
         do 30 jp = jpntr(j), jpntr(j+1)-1
            ir = indrow(jp)
            if (iwa(ir) .ne. j) then
               nnz = nnz + 1
               indrow(nnz) = ir
               iwa(ir) = j
               end if
   30       continue
         jpntr(j) = k + 1
   40    continue
      jpntr(n+1) = nnz + 1
c
c     Extend the data structure to rows.
c
      call setr(m,n,indrow,jpntr,indcol,ipntr,iwa)
c
c     Determine a lower bound for the number of groups.
c
      mingrp = 0
      do 50 i = 1, m
         mingrp = max(mingrp,ipntr(i+1)-ipntr(i))
   50    continue
c
c     Determine the degree sequence for the intersection
c     graph of the columns of A.
c
```

```
      call degr(n,indrow,jpntr,indcol,ipntr,iwa(5*n+ 1),iwa(n+ 1))
c
c     Color the intersection graph of the columns of A
c     with the smallest-last (SL) ordering.
c
      call slo(n,indrow,jpntr,indcol,ipntr,iwa(5*n+ 1),iwa(4*n+ 1),
     *         maxclq,iwa(1),iwa(n+ 1),iwa(2*n+ 1),iwa(3*n+ 1))
      call seq(n,indrow,jpntr,indcol,ipntr,iwa(4*n+ 1),ngrp,maxgrp,
     *         iwa(n+ 1))
      mingrp = max(mingrp,maxclq)
c
c     Exit if the smallest-last ordering is optimal.
c
      if (maxgrp .eq. mingrp) return
c
c     Color the intersection graph of the columns of A
c     with the incidence-degree (ID) ordering.
c
      call ido(m,n,indrow,jpntr,indcol,ipntr,iwa(5*n+ 1),iwa(4*n+ 1),
     *         maxclq,iwa(1),iwa(n+ 1),iwa(2*n+ 1),iwa(3*n+ 1))
      call seq(n,indrow,jpntr,indcol,ipntr,iwa(4*n+ 1),iwa(1),numgrp,
     *         iwa(n+ 1))
      mingrp = max(mingrp,maxclq)
c
c     Retain the better of the two orderings so far.
c
      if (numgrp .lt. maxgrp) then
         maxgrp = numgrp
         do 60 j = 1, n
            ngrp(j) = iwa(j)
   60       continue
c
c        Exit if the incidence-degree ordering is optimal.
c
         if (maxgrp .eq. mingrp) return
         end if
c
c     Color the intersection graph of the columns of A
c     with the largest-first (LF) ordering.
c
      call numsrt(n,n-1,iwa(5*n+ 1),-1,iwa(4*n+ 1),iwa(2*n+ 1),iwa(n+ 1))
      call seq(n,indrow,jpntr,indcol,ipntr,iwa(4*n+ 1),iwa(1),numgrp,
     *         iwa(n+ 1))
c
c     Retain the best of the three orderings and exit.
c
      if (numgrp .lt. maxgrp) then
         maxgrp = numgrp
         do 70 j = 1, n
            ngrp(j) = iwa(j)
   70       continue
         end if
      return
c
```

```
c       Last card of subroutine dsm.
c
        end
```

```
      subroutine degr(n,indrow,jpntr,indcol,ipntr,ndeg,iwa)
      integer n
      integer indrow(*),jpntr(n+1),indcol(*),ipntr(*),ndeg(n),iwa(n)
c     **********
c
c     subroutine degr
c
c     Given the sparsity pattern of an m by n matrix A,
c     this subroutine determines the degree sequence for
c     the intersection graph of the columns of A.
c
c     In graph-theory terminology, the intersection graph of
c     the columns of A is the loopless graph G with vertices
c     a(j), j = 1,2,...,n where a(j) is the j-th column of A
c     and with edge (a(i),a(j)) if and only if columns i and j
c     have a non-zero in the same row position.
c
c     Note that the value of m is not needed by degr and is
c     therefore not present in the subroutine statement.
c
c     The subroutine statement is
c
c       subroutine degr(n,indrow,jpntr,indcol,ipntr,ndeg,iwa)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       indrow is an integer input array which contains the row
c         indices for the non-zeroes in the matrix A.
c
c       jpntr is an integer input array of length n + 1 which
c         specifies the locations of the row indices in indrow.
c         The row indices for column j are
c
c               indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         Note that jpntr(n+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       indcol is an integer input array which contains the
c         column indices for the non-zeroes in the matrix A.
c
c       ipntr is an integer input array of length m + 1 which
c         specifies the locations of the column indices in indcol.
c         The column indices for row i are
c
c               indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c         Note that ipntr(m+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       ndeg is an integer output array of length n which
```

```
c          specifies the degree sequence. The degree of the
c          j-th column of A is ndeg(j).
c
c        iwa is an integer work array of length n.
c
c     Argonne National Laboratory. MINPACK Project. July 1983.
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer ic,ip,ir,jcol,jp
c
c     Initialization block.
c
      do 10 jp = 1, n
         ndeg(jp) = 0
         iwa(jp) = 0
   10    continue
c
c     Compute the degree sequence by determining the contributions
c     to the degrees from the current(jcol) column and further
c     columns which have not yet been considered.
c
      do 40 jcol = 2, n
         iwa(jcol) = n
c
c        Determine all positions (ir,jcol) which correspond
c        to non-zeroes in the matrix.
c
         do 30 jp = jpntr(jcol), jpntr(jcol+1)-1
            ir = indrow(jp)
c
c           For each row ir, determine all positions (ir,ic)
c           which correspond to non-zeroes in the matrix.
c
            do 20 ip = ipntr(ir), ipntr(ir+1)-1
               ic = indcol(ip)
c
c              Array iwa marks columns which have contributed to
c              the degree count of column jcol. Update the degree
c              counts of these columns as well as column jcol.
c
               if (iwa(ic) .lt. jcol) then
                  iwa(ic) = jcol
                  ndeg(ic) = ndeg(ic) + 1
                  ndeg(jcol) = ndeg(jcol) + 1
                  end if
   20          continue
   30       continue
   40    continue
      return
c
c     Last card of subroutine degr.
c
      end
```

```
      subroutine ido(m,n,indrow,jpntr,indcol,ipntr,ndeg,list,
     *               maxclq,iwa1,iwa2,iwa3,iwa4)
      integer m,n,maxclq
      integer indrow(*),jpntr(n+1),indcol(*),ipntr(m+1),ndeg(n),
     *        list(n),iwa1(0:n-1),iwa2(n),iwa3(n),iwa4(n)
c     **********
c
c     subroutine ido
c
c     Given the sparsity pattern of an m by n matrix A, this
c     subroutine determines an incidence-degree ordering of the
c     columns of A.
c
c     The incidence-degree ordering is defined for the loopless
c     graph G with vertices a(j), j = 1,2,...,n where a(j) is the
c     j-th column of A and with edge (a(i),a(j)) if and only if
c     columns i and j have a non-zero in the same row position.
c
c     The incidence-degree ordering is determined recursively by
c     letting list(k), k = 1,...,n be a column with maximal
c     incidence to the subgraph spanned by the ordered columns.
c     Among all the columns of maximal incidence, ido chooses a
c     column of maximal degree.
c
c     The subroutine statement is
c
c       subroutine ido(m,n,indrow,jpntr,indcol,ipntr,ndeg,list,
c                      maxclq,iwa1,iwa2,iwa3,iwa4)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of A.
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       indrow is an integer input array which contains the row
c         indices for the non-zeroes in the matrix A.
c
c       jpntr is an integer input array of length n + 1 which
c         specifies the locations of the row indices in indrow.
c         The row indices for column j are
c
c               indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         Note that jpntr(n+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       indcol is an integer input array which contains the
c         column indices for the non-zeroes in the matrix A.
c
c       ipntr is an integer input array of length m + 1 which
c         specifies the locations of the column indices in indcol.
```

```
c          The column indices for row i are
c
c                indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c          Note that ipntr(m+1)-1 is then the number of non-zero
c          elements of the matrix A.
c
c        ndeg is an integer input array of length n which specifies
c          the degree sequence. The degree of the j-th column
c          of A is ndeg(j).
c
c        list is an integer output array of length n which specifies
c          the incidence-degree ordering of the columns of A. The j-th
c          column in this order is list(j).
c
c        maxclq is an integer output variable set to the size
c          of the largest clique found during the ordering.
c
c        iwa1,iwa2,iwa3, and iwa4 are integer work arrays of length n.
c
c     Subprograms called
c
c        MINPACK-supplied ... numsrt
c
c        FORTRAN-supplied ... max
c
c     Argonne National Laboratory. MINPACK Project. August 1984.
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer ic,ip,ir,jcol,jp,
     *        maxinc,maxlst,ncomp,numinc,numlst,numord,numwgt
c
c     Sort the degree sequence.
c
      call numsrt(n,n-1,ndeg,-1,iwa4,iwa2,iwa3)
c
c     Initialization block.
c
c     Create a doubly-linked list to access the incidences of the
c     columns. The pointers for the linked list are as follows.
c
c     Each un-ordered column ic is in a list (the incidence list)
c     of columns with the same incidence.
c
c     iwa1(numinc) is the first column in the numinc list
c     unless iwa1(numinc) = 0. In this case there are
c     no columns in the numinc list.
c
c     iwa2(ic) is the column before ic in the incidence list
c     unless iwa2(ic) = 0. In this case ic is the first
c     column in this incidence list.
c
c     iwa3(ic) is the column after ic in the incidence list
```

```
c     unless iwa3(ic) = 0. In this case ic is the last
c     column in this incidence list.
c
c     If ic is an un-ordered column, then list(ic) is the
c     incidence of ic to the graph induced by the ordered
c     columns. If jcol is an ordered column, then list(jcol)
c     is the incidence-degree order of column jcol.
c
      maxinc = 0
      do 10 jp = n, 1, -1
         ic = iwa4(jp)
         iwa1(n-jp) = 0
         iwa2(ic) = 0
         iwa3(ic) = iwa1(0)
         if (iwa1(0) .gt. 0) iwa2(iwa1(0)) = ic
         iwa1(0) = ic
         iwa4(jp) = 0
         list(jp) = 0
   10    continue
c
c     Determine the maximal search length for the list
c     of columns of maximal incidence.
c
      maxlst = 0
      do 20 ir = 1, m
         maxlst = maxlst + (ipntr(ir+1) - ipntr(ir))**2
   20    continue
      maxlst = maxlst/n
      maxclq = 0
      numord = 1
c
c     Beginning of iteration loop.
c
   30 continue
c
c        Choose a column jcol of maximal degree among the
c        columns of maximal incidence maxinc.
c
   40    continue
            jp = iwa1(maxinc)
            if (jp .gt. 0) go to 50
            maxinc = maxinc - 1
            go to 40
   50    continue
         numwgt = -1
         do 60 numlst = 1, maxlst
            if (ndeg(jp) .gt. numwgt) then
               numwgt = ndeg(jp)
               jcol = jp
               end if
            jp = iwa3(jp)
            if (jp .le. 0) go to 70
   60       continue
   70    continue
```

```
         list(jcol) = numord
c
c        Update the size of the largest clique
c        found during the ordering.
c
         if (maxinc .eq. 0) ncomp = 0
         ncomp = ncomp + 1
         if (maxinc+1 .eq. ncomp) maxclq = max(maxclq,ncomp)
c
c        Termination test.
c
         numord = numord + 1
         if (numord .gt. n) go to 100
c
c        Delete column jcol from the maxinc list.
c
         if (iwa2(jcol) .eq. 0) then
            iwa1(maxinc) = iwa3(jcol)
         else
            iwa3(iwa2(jcol)) = iwa3(jcol)
            end if
         if (iwa3(jcol) .gt. 0) iwa2(iwa3(jcol)) = iwa2(jcol)
c
c        Find all columns adjacent to column jcol.
c
         iwa4(jcol) = n
c
c        Determine all positions (ir,jcol) which correspond
c        to non-zeroes in the matrix.
c
         do 90 jp = jpntr(jcol), jpntr(jcol+1)-1
            ir = indrow(jp)
c
c           For each row ir, determine all positions (ir,ic)
c           which correspond to non-zeroes in the matrix.
c
            do 80 ip = ipntr(ir), ipntr(ir+1)-1
               ic = indcol(ip)
c
c              Array iwa4 marks columns which are adjacent to
c              column jcol.
c
               if (iwa4(ic) .lt. numord) then
                  iwa4(ic) = numord
c
c                 Update the pointers to the current incidence lists.
c
                  numinc = list(ic)
                  list(ic) = list(ic) + 1
                  maxinc = max(maxinc,list(ic))
c
c                 Delete column ic from the numinc list.
c
                  if (iwa2(ic) .eq. 0) then
```

```
                        iwa1(numinc) = iwa3(ic)
                     else
                        iwa3(iwa2(ic)) = iwa3(ic)
                        end if
                     if (iwa3(ic) .gt. 0) iwa2(iwa3(ic)) = iwa2(ic)
c
c                    Add column ic to the numinc+1 list.
c
                     iwa2(ic) = 0
                     iwa3(ic) = iwa1(numinc+1)
                     if (iwa1(numinc+1) .gt. 0) iwa2(iwa1(numinc+1)) = ic
                     iwa1(numinc+1) = ic
                     end if
   80             continue
   90          continue
c
c           End of iteration loop.
c
            go to 30
  100    continue
c
c        Invert the array list.
c
         do 110 jcol = 1, n
            iwa2(list(jcol)) = jcol
  110       continue
         do 120 jp = 1, n
            list(jp) = iwa2(jp)
  120       continue
         return
c
c        Last card of subroutine ido.
c
         end
```

```
      subroutine numsrt(n,nmax,num,mode,index,last,next)
      integer n,nmax,mode
      integer num(n),index(n),last(0:nmax),next(n)
c     **********.
c
c     subroutine numsrt
c
c     Given a sequence of integers, this subroutine groups
c     together those indices with the same sequence value
c     and, optionally, sorts the sequence into either
c     ascending or descending order.
c
c     The sequence of integers is defined by the array num,
c     and it is assumed that the integers are each from the set
c     0,1,...,nmax. On output the indices k such that num(k) = l
c     for any l = 0,1,...,nmax can be obtained from the arrays
c     last and next as follows.
c
c           k = last(l)
c           while (k .ne. 0) k = next(k)
c
c     Optionally, the subroutine produces an array index so that
c     the sequence num(index(i)), i = 1,2,...,n is sorted.
c
c     The subroutine statement is
c
c       subroutine numsrt(n,nmax,num,mode,index,last,next)
c
c     where
c
c       n is a positive integer input variable.
c
c       nmax is a positive integer input variable.
c
c       num is an input array of length n which contains the
c         sequence of integers to be grouped and sorted. It
c         is assumed that the integers are each from the set
c         0,1,...,nmax.
c
c       mode is an integer input variable. The sequence num is
c         sorted in ascending order if mode is positive and in
c         descending order if mode is negative. If mode is 0,
c         no sorting is done.
c
c       index is an integer output array of length n set so
c         that the sequence
c
c               num(index(i)), i = 1,2,...,n
c
c         is sorted according to the setting of mode. If mode
c         is 0, index is not referenced.
c
c       last is an integer output array of length nmax + 1. The
c         index of num for the last occurrence of l is last(l)
```

```
c             for any l = 0,1,...,nmax unless last(l) = 0. In
c             this case l does not appear in num.
c
c           next is an integer output array of length n. If
c             num(k) = l, then the index of num for the previous
c             occurrence of l is next(k) for any l = 0,1,...,nmax
c             unless next(k) = 0. In this case there is no previous
c             occurrence of l in num.
c
c         Argonne National Laboratory. MINPACK Project. July 1983.
c         Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c         **********
          integer i,j,jinc,jl,ju,k,l
c
c         Determine the arrays next and last.
c
          do 10 i = 0, nmax
             last(i) = 0
   10        continue
          do 20 k = 1, n
             l = num(k)
             next(k) = last(l)
             last(l) = k
   20        continue
          if (mode .eq. 0) return
c
c         Store the pointers to the sorted array in index.
c
          i = 1
          if (mode .gt. 0) then
             jl = 0
             ju = nmax
             jinc = 1
          else
             jl = nmax
             ju = 0
             jinc = -1
             end if
          do 50 j = jl, ju, jinc
             k = last(j)
   30        continue
                if (k .eq. 0) go to 40
                index(i) = k
                i = i + 1
                k = next(k)
                go to 30
   40        continue
   50        continue
          return
c
c         Last card of subroutine numsrt.
c
          end
```

```
      subroutine seq(n,indrow,jpntr,indcol,ipntr,list,ngrp,maxgrp,
     *               iwa)
      integer n,maxgrp
      integer indrow(*),jpntr(n+1),indcol(*),ipntr(*),list(n),
     *        ngrp(n),iwa(n)
c     **********
c
c     subroutine seq
c
c     Given the sparsity pattern of an m by n matrix A, this
c     subroutine determines a consistent partition of the
c     columns of A by a sequential algorithm.
c
c     A consistent partition is defined in terms of the loopless
c     graph G with vertices a(j), j = 1,2,...,n where a(j) is the
c     j-th column of A and with edge (a(i),a(j)) if and only if
c     columns i and j have a non-zero in the same row position.
c
c     A partition of the columns of A into groups is consistent
c     if the columns in any group are not adjacent in the graph G.
c     In graph-theory terminology, a consistent partition of the
c     columns of A corresponds to a coloring of the graph G.
c
c     The subroutine examines the columns in the order specified
c     by the array list, and assigns the current column to the
c     group with the smallest possible number.
c
c     Note that the value of m is not needed by seq and is
c     therefore not present in the subroutine statement.
c
c     The subroutine statement is
c
c       subroutine seq(n,indrow,jpntr,indcol,ipntr,list,ngrp,maxgrp,
c                      iwa)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       indrow is an integer input array which contains the row
c         indices for the non-zeroes in the matrix A.
c
c       jpntr is an integer input array of length n + 1 which
c         specifies the locations of the row indices in indrow.
c         The row indices for column j are
c
c               indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         Note that jpntr(n+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       indcol is an integer input array which contains the
c         column indices for the non-zeroes in the matrix A.
```

```
c
c       ipntr is an integer input array of length m + 1 which
c         specifies the locations of the column indices in indcol.
c         The column indices for row i are
c
c               indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c         Note that ipntr(m+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       list is an integer input array of length n which specifies
c         the order to be used by the sequential algorithm.
c         The j-th column in this order is list(j).
c
c       ngrp is an integer output array of length n which specifies
c         the partition of the columns of A. Column jcol belongs
c         to group ngrp(jcol).
c
c       maxgrp is an integer output variable which specifies the
c         number of groups in the partition of the columns of A.
c
c       iwa is an integer work array of length n.
c
c     Argonne National Laboratory. MINPACK Project. July 1983.
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer ic,ip,ir,j,jcol,jp
c
c     Initialization block.
c
      maxgrp = 0
      do 10 jp = 1, n
         ngrp(jp) = n
         iwa(jp) = 0
   10    continue
c
c     Beginning of iteration loop.
c
      do 60 j = 1, n
         jcol = list(j)
c
c        Find all columns adjacent to column jcol.
c
c        Determine all positions (ir,jcol) which correspond
c        to non-zeroes in the matrix.
c
         do 30 jp = jpntr(jcol), jpntr(jcol+1)-1
            ir = indrow(jp)
c
c           For each row ir, determine all positions (ir,ic)
c           which correspond to non-zeroes in the matrix.
c
            do 20 ip = ipntr(ir), ipntr(ir+1)-1
```

```
                  ic = indcol(ip)
c
c                 Array iwa marks the group numbers of the
c                 columns which are adjacent to column jcol.
c
                  iwa(ngrp(ic)) = j
   20             continue
   30          continue
c
c        Assign the smallest un-marked group number to jcol.
c
         do 40 jp = 1, maxgrp
            if (iwa(jp) .ne. j) go to 50
   40       continue
         maxgrp = maxgrp + 1
   50    continue
         ngrp(jcol) = jp
   60    continue
c
c        End of iteration loop.
c
      return
c
c     Last card of subroutine seq.
c
      end
```

```
      subroutine setr(m,n,indrow,jpntr,indcol,ipntr,iwa)
      integer m,n
      integer indrow(*),jpntr(n+1),indcol(*),ipntr(m+1),iwa(m)
c     **********
c
c     subroutine setr
c
c     Given a column-oriented definition of the sparsity pattern
c     of an m by n matrix A, this subroutine determines a
c     row-oriented definition of the sparsity pattern of A.
c
c     On input the column-oriented definition is specified by
c     the arrays indrow and jpntr. On output the row-oriented
c     definition is specified by the arrays indcol and ipntr.
c
c     The subroutine statement is
c
c       subroutine setr(m,n,indrow,jpntr,indcol,ipntr,iwa)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of A.
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       indrow is an integer input array which contains the row
c         indices for the non-zeroes in the matrix A.
c
c       jpntr is an integer input array of length n + 1 which
c         specifies the locations of the row indices in indrow.
c         The row indices for column j are
c
c               indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         Note that jpntr(n+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       indcol is an integer output array which contains the
c         column indices for the non-zeroes in the matrix A.
c
c       ipntr is an integer output array of length m + 1 which
c         specifies the locations of the column indices in indcol.
c         The column indices for row i are
c
c               indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c         Note that ipntr(1) is set to 1 and that ipntr(m+1)-1 is
c         then the number of non-zero elements of the matrix A.
c
c       iwa is an integer work array of length m.
c
c     Argonne National Laboratory. MINPACK Project. July 1983.
```

```
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer ir,jcol,jp
c
c     Store in array iwa the counts of non-zeroes in the rows.
c
      do 10 ir = 1, m
         iwa(ir) = 0
   10    continue
      do 20 jp = 1, jpntr(n+1)-1
         iwa(indrow(jp)) = iwa(indrow(jp)) + 1
   20    continue
c
c     Set pointers to the start of the rows in indcol.
c
      ipntr(1) = 1
      do 30 ir = 1, m
         ipntr(ir+1) = ipntr(ir) + iwa(ir)
         iwa(ir) = ipntr(ir)
   30    continue
c
c     Fill indcol.
c
      do 50 jcol = 1, n
         do 40 jp = jpntr(jcol), jpntr(jcol+1)-1
            ir = indrow(jp)
            indcol(iwa(ir)) = jcol
            iwa(ir) = iwa(ir) + 1
   40       continue
   50    continue
      return
c
c     Last card of subroutine setr.
c
      end
```

```
      subroutine slo(n,indrow,jpntr,indcol,ipntr,ndeg,list,
     *               maxclq,iwa1,iwa2,iwa3,iwa4)
      integer n,maxclq
      integer indrow(*),jpntr(n+1),indcol(*),ipntr(*),ndeg(n),
     *        list(n),iwa1(0:n-1),iwa2(n),iwa3(n),iwa4(n)
c     **********
c
c     subroutine slo
c
c     Given the sparsity pattern of an m by n matrix A, this
c     subroutine determines the smallest-last ordering of the
c     columns of A.
c
c     The smallest-last ordering is defined for the loopless
c     graph G with vertices a(j), j = 1,2,...,n where a(j) is the
c     j-th column of A and with edge (a(i),a(j)) if and only if
c     columns i and j have a non-zero in the same row position.
c
c     The smallest-last ordering is determined recursively by
c     letting list(k), k = n,...,1 be a column with least degree
c     in the subgraph spanned by the un-ordered columns.
c
c     Note that the value of m is not needed by slo and is
c     therefore not present in the subroutine statement.
c
c     The subroutine statement is
c
c       subroutine slo(n,indrow,jpntr,indcol,ipntr,ndeg,list,
c                      maxclq,iwa1,iwa2,iwa3,iwa4)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       indrow is an integer input array which contains the row
c         indices for the non-zeroes in the matrix A.
c
c       jpntr is an integer input array of length n + 1 which
c         specifies the locations of the row indices in indrow.
c         The row indices for column j are
c
c               indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         Note that jpntr(n+1)-1 is then the number of non-zero
c         elements of the matrix A.
c
c       indcol is an integer input array which contains the
c         column indices for the non-zeroes in the matrix A.
c
c       ipntr is an integer input array of length m + 1 which
c         specifies the locations of the column indices in indcol.
c         The column indices for row i are
c
```

```
c                 indcol(k), k = ipntr(i),...,ipntr(i+1)-1.
c
c           Note that ipntr(m+1)-1 is then the number of non-zero
c           elements of the matrix A.
c
c         ndeg is an integer input array of length n which specifies
c           the degree sequence. The degree of the j-th column
c           of A is ndeg(j).
c
c         list is an integer output array of length n which specifies
c           the smallest-last ordering of the columns of A. The j-th
c           column in this order is list(j).
c
c         maxclq is an integer output variable set to the size
c           of the largest clique found during the ordering.
c
c         iwa1,iwa2,iwa3, and iwa4 are integer work arrays of length n.
c
c       Subprograms called
c
c         FORTRAN-supplied ... min
c
c       Argonne National Laboratory. MINPACK Project. August 1984.
c       Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c       **********
        integer ic,ip,ir,jcol,jp,mindeg,numdeg,numord
c
c       Initialization block.
c
        mindeg = n
        do 10 jp = 1, n
           iwa1(jp-1) = 0
           iwa4(jp) = n
           list(jp) = ndeg(jp)
           mindeg = min(mindeg,ndeg(jp))
   10      continue
c
c       Create a doubly-linked list to access the degrees of the
c       columns. The pointers for the linked list are as follows.
c
c       Each un-ordered column ic is in a list (the degree list)
c       of columns with the same degree.
c
c       iwa1(numdeg) is the first column in the numdeg list
c       unless iwa1(numdeg) = 0. In this case there are
c       no columns in the numdeg list.
c
c       iwa2(ic) is the column before ic in the degree list
c       unless iwa2(ic) = 0. In this case ic is the first
c       column in this degree list.
c
c       iwa3(ic) is the column after ic in the degree list
c       unless iwa3(ic) = 0. In this case ic is the last
```

```
c       column in this degree list.
c
c       If ic is an un-ordered column, then list(ic) is the
c       degree of ic in the graph induced by the un-ordered
c       columns. If jcol is an ordered column, then list(jcol)
c       is the smallest-last order of column jcol.
c
        do 20 jp = 1, n
           numdeg = ndeg(jp)
           iwa2(jp) = 0
           iwa3(jp) = iwa1(numdeg)
           if (iwa1(numdeg) .gt. 0) iwa2(iwa1(numdeg)) = jp
           iwa1(numdeg) = jp
   20      continue
        maxclq = 0
        numord = n
c
c       Beginning of iteration loop.
c
   30 continue
c
c          Choose a column jcol of minimal degree mindeg.
c
   40      continue
              jcol = iwa1(mindeg)
              if (jcol .gt. 0) go to 50
              mindeg = mindeg + 1
              go to 40
   50      continue
           list(jcol) = numord
c
c          Mark the size of the largest clique
c          found during the ordering.
c
           if (mindeg+1 .eq. numord .and. maxclq .eq. 0)
     *         maxclq = numord
c
c          Termination test.
c
           numord = numord - 1
           if (numord .eq. 0) go to 80
c
c          Delete column jcol from the mindeg list.
c
           iwa1(mindeg) = iwa3(jcol)
           if (iwa3(jcol) .gt. 0) iwa2(iwa3(jcol)) = 0
c
c          Find all columns adjacent to column jcol.
c
           iwa4(jcol) = 0
c
c          Determine all positions (ir,jcol) which correspond
c          to non-zeroes in the matrix.
c
```

```
         do 70 jp = jpntr(jcol), jpntr(jcol+1)-1
            ir = indrow(jp)
c
c           For each row ir, determine all positions (ir,ic)
c           which correspond to non-zeroes in the matrix.
c
            do 60 ip = ipntr(ir), ipntr(ir+1)-1
               ic = indcol(ip)
c
c              Array iwa4 marks columns which are adjacent to
c              column jcol.
c
               if (iwa4(ic) .gt. numord) then
                  iwa4(ic) = numord
c
c                 Update the pointers to the current degree lists.
c
                  numdeg = list(ic)
                  list(ic) = list(ic) - 1
                  mindeg = min(mindeg,list(ic))
c
c                 Delete column ic from the numdeg list.
c
                  if (iwa2(ic) .eq. 0) then
                     iwa1(numdeg) = iwa3(ic)
                  else
                     iwa3(iwa2(ic)) = iwa3(ic)
                     end if
                  if (iwa3(ic) .gt. 0) iwa2(iwa3(ic)) = iwa2(ic)
c
c                 Add column ic to the numdeg-1 list.
c
                  iwa2(ic) = 0
                  iwa3(ic) = iwa1(numdeg-1)
                  if (iwa1(numdeg-1) .gt. 0) iwa2(iwa1(numdeg-1)) = ic
                  iwa1(numdeg-1) = ic
                  end if
   60          continue
   70       continue
c
c        End of iteration loop.
c
         go to 30
   80 continue
c
c     Invert the array list.
c
      do 90 jcol = 1, n
         iwa2(list(jcol)) = jcol
   90    continue
      do 100 jp = 1, n
         list(jp) = iwa2(jp)
  100    continue
      return
```

```
c
c       Last card of subroutine slo.
c
        end
```

```
      subroutine srtdat(n,nnz,indrow,indcol,jpntr,iwa)
      integer n,nnz
      integer indrow(nnz),indcol(nnz),jpntr(n+1),iwa(n)
c     **********
c
c     subroutine srtdat
c
c     Given the non-zero elements of an m by n matrix A in
c     arbitrary order as specified by their row and column
c     indices, this subroutine permutes these elements so
c     that their column indices are in non-decreasing order.
c
c     On input it is assumed that the elements are specified in
c
c           indrow(k),indcol(k), k = 1,...,nnz.
c
c     On output the elements are permuted so that indcol is
c     in non-decreasing order. In addition, the array jpntr
c     is set so that the row indices for column j are
c
c           indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c     Note that the value of m is not needed by srtdat and is
c     therefore not present in the subroutine statement.
c
c     The subroutine statement is
c
c       subroutine srtdat(n,nnz,indrow,indcol,jpntr,iwa)
c
c     where
c
c       n is a positive integer input variable set to the number
c         of columns of A.
c
c       nnz is a positive integer input variable set to the number
c         of non-zero elements of A.
c
c       indrow is an integer array of length nnz. On input indrow
c         must contain the row indices of the non-zero elements of A.
c         On output indrow is permuted so that the corresponding
c         column indices of indcol are in non-decreasing order.
c
c       indcol is an integer array of length nnz. On input indcol
c         must contain the column indices of the non-zero elements
c         of A. On output indcol is permuted so that these indices
c         are in non-decreasing order.
c
c       jpntr is an integer output array of length n + 1 which
c         specifies the locations of the row indices in the output
c         indrow. The row indices for column j are
c
c               indrow(k), k = jpntr(j),...,jpntr(j+1)-1.
c
c         Note that jpntr(1) is set to 1 and that jpntr(n+1)-1
```

```
c         is then nnz.
c
c       iwa is an integer work array of length n.
c
c     Subprograms called
c
c       FORTRAN-supplied ... max
c
c     Argonne National Laboratory. MINPACK Project. July 1983.
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer i,j,k,l
c
c     Store in array iwa the counts of non-zeroes in the columns.
c
      do 10 j = 1, n
         iwa(j) = 0
   10    continue
      do 20 k = 1, nnz
         iwa(indcol(k)) = iwa(indcol(k)) + 1
   20    continue
c
c     Set pointers to the start of the columns in indrow.
c
      jpntr(1) = 1
      do 30 j = 1, n
         jpntr(j+1) = jpntr(j) + iwa(j)
         iwa(j) = jpntr(j)
   30    continue
      k = 1
c
c     Begin in-place sort.
c
   40 continue
         j = indcol(k)
         if (k .ge. jpntr(j)) then
c
c           Current element is in position. Now examine the
c           next element or the first un-sorted element in
c           the j-th group.
c
            k = max(k+1,iwa(j))
         else
c
c           Current element is not in position. Place element
c           in position and make the displaced element the
c           current element.
c
            l = iwa(j)
            iwa(j) = iwa(j) + 1
            i = indrow(k)
            indrow(k) = indrow(l)
            indcol(k) = indcol(l)
```

```
            indrow(l) = i
            indcol(l) = j
            end if
         if (k .le. nnz) go to 40
      return
c
c     Last card of subroutine srtdat.
c
      end
```

```
      subroutine fdjs(m,n,col,ind,npntr,ngrp,numgrp,eta,fjacd,fjac)
      integer m,n,numgrp
      integer ind(*),npntr(*),ngrp(n)
      real eta(n),fjacd(m),fjac(*)
      logical col
c     **********
c
c     subroutine fdjs
c
c     Given a consistent partition of the columns of an m by n
c     Jacobian matrix into groups, this subroutine computes
c     approximations to those columns in a given group.  The
c     approximations are stored into either a column-oriented
c     or a row-oriented pattern.
c
c     A partition is consistent if the columns in any group
c     do not have a non-zero in the same row position.
c
c     Approximations to the columns of the Jacobian matrix in a
c     given group can be obtained by specifying a difference
c     parameter vector eta and an approximation to J*d where J
c     is the Jacobian matrix and the vector d is defined by the
c     following section of code.
c
c           do 10 j = 1, n
c              d(j) = 0.0
c              if (ngrp(j) .eq. numgrp) d(j) = eta(j)
c        10    continue
c
c     In the above code numgrp is the given group number, ngrp(j)
c     is the group number of column j, and eta(j) is the difference
c     parameter used to approximate column j of the Jacobian matrix.
c     Suitable values for eta(j) must be provided.
c
c     As mentioned above, an approximation to J*d must be provided.
c     For example, if f(x) is the function value at x, then
c
c           f(x+d) - f(x)
c
c     corresponds to the forward difference approximation.
c
c     The subroutine statement is
c
c       subroutine fdjs(m,n,col,ind,npntr,ngrp,numgrp,eta,fjacd,fjac)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of the Jacobian matrix.
c
c       n is a positive integer input variable set to the number
c         of columns of the Jacobian matrix.
c
c       col is a logical input variable. If col is set true, then the
```

```
c          Jacobian approximations are stored into a column-oriented
c          pattern. If col is set false, then the Jacobian
c          approximations are stored into a row-oriented pattern.
c
c       ind is an integer input array which contains the row
c         indices for the non-zeroes in the Jacobian matrix
c         if col is true, and contains the column indices for
c         the non-zeroes in the Jacobian matrix if col is false.
c
c       npntr is an integer input array which specifies the
c         locations of the row indices in ind if col is true, and
c         specifies the locations of the column indices in ind if
c         col is false. If col is true, the indices for column j are
c
c               ind(k), k = npntr(j),...,npntr(j+1)-1.
c
c         If col is false, the indices for row i are
c
c               ind(k), k = npntr(i),...,npntr(i+1)-1.
c
c         Note that npntr(n+1)-1 if col is true, or npntr(m+1)-1
c         if col is false, is then the number of non-zero elements
c         of the Jacobian matrix.
c
c       ngrp is an integer input array of length n which specifies
c         the partition of the columns of the Jacobian matrix.
c         Column j belongs to group ngrp(j).
c
c       numgrp is a positive integer input variable set to a group
c         number in the partition. The columns of the Jacobian
c         matrix in this group are to be estimated on this call.
c
c       eta is an input array of length n which contains the
c         difference parameter vector.
c
c       fjacd is an input array of length m which contains an
c         approximation to J*d, where J is the Jacobian matrix
c         and d is the difference vector for group numgrp.
c
c       fjac is an output array of length nnz, where nnz is the
c         number of non-zero elements in the Jacobian matrix.
c         At each call of fdjs, fjac is updated to include the
c         non-zero elements of the Jacobian matrix for those columns
c         in group numgrp.
c
c     Argonne National Laboratory. MINPACK Project. December 1984.
c     Thomas F. Coleman, Burton S. Garbow, Jorge J. More'
c
c     **********
      integer i,ip,j,jp
c
c     Compute estimates of Jacobian matrix columns in group
c     numgrp. The array fjacd must contain an approximation
c     to J*d, where J is the Jacobian matrix and d is the
```

```
c       difference vector for group numgrp.
c
        if (col) then
c
c          Column orientation.
c
           do 20 j = 1, n
              if (ngrp(j) .eq. numgrp) then
                 do 10 jp = npntr(j), npntr(j+ 1)-1
                    i = ind(jp)
                    fjac(jp) = fjacd(i)/eta(j)
   10               continue
                 end if
   20         continue
        else
c
c          Row orientation.
c
           do 50 i = 1, m
              do 30 ip = npntr(i), npntr(i+ 1)-1
                 j = ind(ip)
                 if (ngrp(j) .eq. numgrp) then
                    fjac(ip) = fjacd(i)/eta(j)
                    go to 40
                    end if
   30            continue
   40         continue
   50         continue
           end if
        return
c
c       Last card of subroutine fdjs.
c
        end
```