UNIVERSIDAD POLITECNICA DE VALENCIA

Jose Luis Oliver Herrero.
Universidad Politécnica de Valencia.
Dpto. Ing. Mecánica y Materiales.
P. O. Box 22012.
46022 VALENCIA.
SPAIN.
Phone (34)-6- 361 50 51, Ext. 128
Fax # (34)-6-360 31 78

19th September, 1988, Valencia,

UNIVERSITY OF WATERLOO
Department of Computer Science
Waterloo, Ontario N21 3G1
CANADA

Dear Mrs. DeAngelis:

Thank you for your kindness. We have received the reports CS-78-12 and CS-78-30.

Enclosed, I send you check number 002182/2076 payable to COMPUTER SCIENCE DEPARTMENT, UNIVERSITY OF WATERLOO, for $ CAD. 8.

Sincerely yours

[illegible]

Prof. J. L. O. Herrero.

rec'd Oct. 3/88

VENDOR NO.	VENDOR NAME
WTR1	UNIVERSITY OF WATERLOO

INFORMATION ON DEMAND, INC.
2112 BERKELEY WAY
BERKELEY, CA 94704

INVOICE NO.	AMOUNT	DISCOUNT		VOUCHER NO.	
SERVICES	4.00			007684	
CHECK#	4473	CHECK AMOUNT	$4.00		

received payment

JAN 16 1987

S.P.

PLEASE DETACH BEFORE DEPOSITING

BY ENDORSEMENT THIS CHECK WHEN PAID IS ACCEPTED IN FULL PAYMENT OF THE FOLLOWING ACCOUNT

University of Waterloo

INVOICE

Waterloo, Ontario, Canada
N2L 3G1

Faculty of Mathematics
Department of Computer Science
519/885-1211

Telex Number
069-55259

November 3, 1986.

Ms. Amy Rafferty,
"Information on Demand",
P.O. Box 1370,
Berkeley, CA 94701
U.S.A.

Dear Ms. Rafferty:

Enclosed please find our technical report CS-78-30 which you requested in your phone call of October 24th, 1986.

Please be advised that the cost of this report is $4.00 Canadian. Would you please make your cheque or money order payable to the University of Waterloo, Computer Science Dept.

Thanking you in advance.

Yours truly,

Susan DeAngelis

Susan DeAngelis (Mrs.),
Technical Report Secretary.

/sd
Encl.

PHONE CALL

Date 24 Oct. 86 Time 1:58

To Sue

WHILE YOU WERE OUT

$4.00

of for report

Phone 415-644 4500

Telephoned	✓	Please call	✓
Called to see you		Will call again	
Wants to see you		Returned your call	

MESSAGE CS-78-30 Bob Morgan User Guide Sparspak

Amy Rafferty

Mail Stop → 288 4020

"Information on Demand":

P.O. 1370

Operator Berkeley, CA 94701

C & M OFFICE SERVICES LTD.
127 Sheldon Drive
Cambridge, Ontario N1R 6T7
Telephone 621-4361 653-5675
Office Furniture ● Business Machines ● Office Stationery

Printing Requisition/Graphic Services

45615

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION: CS-78-30

DATE REQUISITIONED: Oct. 29 | DATE REQUIRED: ASAP | ACCOUNT NO.: 1 2 6 4 4 3 1 0 2

REQUISITIONER- PRINT: SUE DeANGELIS | PHONE: 2192 | SIGNING AUTHORITY: Sue DeAngelis

MAILING INFO – NAME | DEPT. | BLDG. & ROOM NO. | ☐ DELIVER [✓ PICK-UP]

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 68 | NUMBER OF COPIES: 1

TYPE OF PAPER STOCK: [✓ BOND] ☐ NCR ___ PT. ☐ COVER ☐ BRISTOL ☐ SUPPLIED ☐ ___

PAPER SIZE: [✓ 8½ x 11] ☐ 8½ x 14 ☐ 11 x 17 ☐ ___

PAPER COLOUR: [✓ WHITE] ☐ ___ | INK: [✓ BLACK] ☐ ___

PRINTING: ☐ 1 SIDE ___ PGS. [✓ 2 SIDES] ___ PGS. | NUMBERING FROM ___ TO ___

BINDING/FINISHING: [✓ COLLATING] ☐ STAPLING ☐ ___ HOLE PUNCHED ☐ PLASTIC RING

FOLDING/PADDING | CUTTING SIZE

Special Instructions

Title page + page ii on separate pages.

COPY CENTRE OPER. NO. | BLDG. | MACH. NO.

DESIGN & PASTE-UP OPER. NO. | TIME | LABOUR CODE

OPER. NO.	TIME	LABOUR CODE
		D01
		D01
		D01

TYPESETTING QUANTITY

	QUANTITY			
PAP00000				T01
PAP00000				T01
PAP00000				T01

PROOF

PRF				
PRF				
PRF				

	QUANTITY	OPER. NO.	TIME	LABOUR CODE
NEGATIVES				
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01
PMT				
PMT				C01
PMT				C01
PMT				C01
PLATES				
PLT				P01
PLT				P01
PLT				P01
STOCK				
				001
				001
				001
				001
BINDERY				
RNG				B01
RNG				B01
RNG				B01
MIS00000				B01

OUTSIDE SERVICES

$ ___ COST

TAXES – PROVINCIAL ☐ FEDERAL ☐

GRAPHIC SERV. OCT. 85 482-2

PrintingRequisition/GraphicServices

15065

1. **Please complete unshaded areas** on form as applicable.
2. **Distribute copies** as follows: White and Yellow to Graphic Services. **Retain Pink Copies for your records.**
3. On completion of order the **Yellow copy will be returned** with the printed material.
4. **Please direct enquiries, quoting requisition number and account number,** to **extension 3451.**

TITLE OR DESCRIPTION: CS-78-30

DATE REQUISITIONED: Aug. 15 | DATE REQUIRED: Aug. 16 | ACCOUNT NO.: 1,2,6,4,4,3,1,0,2

REQUISITIONER – **PRINT** | PHONE: 2192 | SIGNING AUTHORITY: Sue DeAngelis

MAILING INFO – NAME | DEPT. | BLDG. & ROOM NO. | [ ] DELIVER [✓ PICK-UP]

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 68 | NUMBER OF COPIES: 1

TYPE OF PAPER STOCK: [✓ BOND] [ ] NCR ____ PT. [ ] COVER [ ] BRISTOL [ ] SUPPLIED [ ] ____

PAPER SIZE: [✓ 8½ x 11] [ ] 8½ x 14 [ ] 11 x 17 [ ] ____

PAPER COLOUR: [✓ WHITE] [ ] ____ | INK: [✓ BLACK] [ ] ____

PRINTING: [✓ 1 SIDE] ____ PGS. [ ] 2 SIDES ____ PGS. | NUMBERING FROM ____ TO ____

BINDING/FINISHING: [✓ COLLATING] [✓ STAPLING] [ ] ____ HOLE PUNCHED [ ] PLASTIC RING

FOLDING/PADDING | CUTTING SIZE

Special Instructions

COPY CENTRE OPER. NO. | BLDG. | MACH. NO.

DESIGN & PASTE-UP

OPER. NO.	TIME	LABOUR CODE
		D01
		D01
		D01

TYPESETTING

	QUANTITY			
PAP000000				T01
PAP000000				T01
PAP000000				T01

PROOF

PRF				
PRF				
PRF				

	QUANTITY	OPER. NO.	TIME	LABOUR CODE
NEGATIVES				
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01
PMT				
PMT				C01
PMT				C01
PMT				C01
PLATES				
PLT				P01
PLT				P01
PLT				P01
STOCK				
				001
				001
				001
				001
BINDERY				
RNG				B01
RNG				B01
RNG				B01
MIS000000				B01

OUTSIDE SERVICES

$ ____ COST

TAXES – PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 85 482-2

MATH 369 33

Printing Requisition/Graphic Services

Dept. No. 92440

1. Please complete unshaded areas on form as applicable. (4 part no carbon required).
2. Distribute copies as follows: White, Canary and Pink—Printing, Arts Library or applicable Copy Centre Goldenrod—**Retain.**
3. On completion of order, pink copy will be returned with printed material. Canary copy will be costed and returned to requisitioner, **Retain as a record of your charges.**
4. Please direct enquiries, quoting requisition number, to Printing/Graphic Services, Extension 3451.

Title or Description User Guide for SPARSPAK: CS-78-30

Date July 15/80 **Date Required** A.S.A.P. Please **Account** 126-6602-41

Signature Edith Huang **Signing Authority** [illegible]

Department Computer Science **Room** MC5100B **Phone** 3402

Delivery [✓ Mail] ☐ Pick-up ☐ Via Stores ☐ Other

Reproduction Requirements [✓ Offset] ☐ Signs/Repro's ☐ Xerox

Number of Pages 68 **Number of Copies** 100

Type of Paper Stock [✓ Bond] ☐ Book ☐ Cover ☐ Bristol ☐ Supplied

Paper Size [✓ 8½ x 11] ☐ 8½ x 14 ☐ 11 x 17 10M

Paper Colour [✓ White] ☐ Other **Ink** [✓ Black]

Printing ☐ 1 Side [✓ 2 Sides] **Numbering** FIRST 2 PAGES 1 SIDE

Binding/Finishing Operations [✓ Collating] [✓ Corner Stitching] ☐ 3 Ring ☐ Tape ☐ Plastic Ring ☐ Perforating

Folding Finished Size 3 STAPLES **Cutting** Finished Size

Special Instructions
STAPLE TOP MIDDLE + BOTTOM ON LEFT HAND SIDE.
2 SIDES REST OF PAGES.

Film Qty Size **Plates** Qty Size & Type

Paper Qty Size **Plastic Rings** Qty Size

Outside Services

Sub-Total Time

Sub-Total Materials

Prov. Tax

Total

Graphic Services May 68 482-2

User Guide for SPARSPAK: Waterloo Sparse Linear Equations Package

Alan George, Joseph Liu, Esmond Ng

Research Report CS-78-30
(Revised, Jan. 1980)

Department of Computer Science

ABSTRACT

SPARSPAK USER GUIDE

This document describes the structure and use of SPARSPAK, the Waterloo Sparse Linear Equations Package, which is designed to efficiently solve large sparse systems of linear equations. Computer programs for solving sparse systems of linear equations typically involve fairly complicated data structures and storage management. In many cases the user of such programs simply wants to solve his problem, and should not have to understand how the storage management is done, or how the matrix components are actually stored. One of the attractive features of this package is that it effectively insulates the user from these considerations, while still allowing the package to be used in a variety of ways. Another important feature of the package is the provision of a variety of methods for solving sparse systems, along with convenient means by which the best method for a given problem can be selected.

CONTENTS

SECTION 1

INTRODUCTION AND BASIC STRUCTURE OF SPARSPAK

SPARSPAK offers a <u>collection</u> of <u>methods</u> for solving sparse systems of linear equations

$$A x = b ,$$

where A is an N by N nonsingular matrix, and x and b are vectors of length N . We assume the user is aware of the basic issues involved in solving sparse matrix equations, and the basic facts about solving systems of linear equations using Gaussian elimination. For a discussion on the initial design of this package, see [5].

For all the methods provided in SPARSPAK, the user and the package interact to solve the matrix problem through the following basic steps:

Step 1. The user supplies the nonzero structure of A to the package using a set of subroutines described in Section 2.2.

Step 2. The package reorders the original problem (finds a permutation P), and allocates storage for the triangular factorization of $PAP' = LU$, as described in Section 2.3 [1].

Step 3. The user supplies the numerical values for the matrix A to the package, as described in Section 2.4.

Step 4. The package computes the triangular factors L and U of PAP' , as described in Section 2.5.

Step 5. The user supplies numerical values for b , as described in Section 2.4. (This step may come before Step 4, and may be intermixed with Step 3.)

Step 6. The package computes the solution x , using L, U, P and b , as described in Section 2.5.

The different <u>methods</u> provided in SPARSPAK correspond to different algorithms for choosing P (along with appropriate storage methods), and whether or not A is symmetric. When A is symmetric, U is replaced by L' in the above description, and of course only one of L and L' is stored.

---

[1] P' stands for the transpose of the matrix P.

The user chooses a particular method by calling the appropriate subroutines in Steps 2, 3, 4 and 6. The methods are distinguished by a numerical digit i, $1 \le i \le 6$, which is the last character of the subroutine names. The subroutines used in Steps 1 and 5 apply to all the methods. The best method to use depends very much on the particular problem, and the context in which it is being solved, so we cannot provide rigid rules as to which method to use. Some guidelines and considerations regarding the choice of method are given in Section 3.

RESTRICTIONS AND ASSUMPTIONS

1. SPARSPAK assumes that the nonzero structure of A is _symmetric_. If this is not the case, the package will still work, but if A has highly unsymmetric structure, this may lead to some inefficiencies because the matrix will be treated as though its structure is that of $A + A'$. The diagonal elements of A are assumed to be nonzero.

2. SPARSPAK assumes that for any permutation matrix P, Gaussian elimination applied to PAP' _without row or column interchanges_ yields an acceptably accurate factorization LU. In other words, the package assumes that A can be symmetrically permuted without regard for numerical stability. This is true, for example, when A is symmetric and positive definite, or diagonally dominant.

SECTION 2

MODULES OF SPARSPAK AND HOW TO USE THEM

2.1 USER MAINLINE PROGRAM AND AN EXAMPLE

SPARSPAK allocates all its storage from a single one dimensional floating point array[2] which for purposes of discussion we will denote by S . In addition, the user must provide its size MAXS, which is transmitted to the package via a common block SPKUSR , (SPARSPAK USER), which has four variables:

```
        COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
```

Here MSGLVL is the message level indicator which is used to control the amount of information printed by the package. The second variable IERR is an error code, which the user can examine in his mainline program for possible errors detected by the package. Detailed discussion of the roles of MSGLVL and IERR is provided in Section 7. The variable NEQNS is the number of equations.

The following program illustrates how one might use SPARSPAK. The various subroutines referenced are described in the subsequent parts of this section. The problem that is solved is a 10 by 10 symmetric tridiagonal system $Ax = b$ where the diagonal elements of A are all 4 , the superdiagonal and subdiagonal elements are all -1 , and the entries in the right hand side vector b are all ones.

---

[2] Declared either REAL or DOUBLE PRECISION, depending on the version of SPARSPAK that is available. The examples in this manual assume a single precision version is being used.

```
      REAL       S(250), FOUR, ONE
      INTEGER    I, IERR, MAXS, MSGLVL, NEQNS
      COMMON     /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
C
         CALL  SPRSPK
         MAXS = 250
C        ------------------------------------------------
C        INPUT THE MATRIX STRUCTURE.  THE DIAGONAL IS
C        ALWAYS ASSUMED TO BE NONZERO, AND SINCE THE
C        MATRIX IS SYMMETRIC, SPARSPAK ONLY NEEDS TO
C        KNOW THAT THE SUBDIAGONAL ELEMENTS ARE NONZERO.
C        ------------------------------------------------
         CALL  IJBEGN
         DO  100  I = 2, 10
            CALL INIJ ( I, I-1, S )
  100    CONTINUE
         CALL  IJEND ( S )
C        ----------------------------------------
C        FIND THE ORDERING AND ALLOCATE STORAGE.
C        ----------------------------------------
         CALL  ORDRA1 ( S )
C        ------------------------------------------------
C        INPUT THE NUMERICAL VALUES FOR A AND B.  SINCE
C        THE MATRIX IS SYMMETRIC, ONLY THE LOWER TRIANGLE
C        AND THE DIAGONAL ARE INPUT.
C        ------------------------------------------------
         FOUR = 4.0E0
         ONE  = 1.0E0
         DO  200  I = 1, 10
            IF  ( I .GT. 1 )
     *            CALL  INAIJ1 ( I, I-1, (-ONE), S )
            CALL  INAIJ1 ( I, I, FOUR, S )
            CALL  INBI ( I, ONE, S )
  200    CONTINUE
C        ----------------
C        SOLVE THE SYSTEM.
C        ----------------
         CALL  SOLVE1 ( S )
C        ------------------------------------------
C        PRINT THE SOLUTION, FOUND IN THE FIRST TEN
C        LOCATIONS OF THE WORKING STORAGE ARRAY S.
C        ------------------------------------------
         WRITE (6, 11)  (S(I), I = 1, 10)
   11    FORMAT ( / 10H SOLUTION  / (5F12.5) )
C        ------------------------------------------
C        PRINT SOME STATISTICS GATHER BY SPARSPAK.
C        ------------------------------------------
         CALL  PSTATS
C
         STOP
      END
```

<u>NOTE</u>: If the SPARSPAK available to you is a double precision version, the REAL declaration in this example should be changed to DOUBLE PRECISION.

The module SPRSPK must be called before any part of the package is used. Its role is to initialize some system parameters (e.g. the logical unit numbers for output files), and to set default values for options (e.g. initializing the timing routine). The routine needs only to be called once in the user program, and the FORTRAN statement is simply

```
CALL SPRSPK
```

Note that the only variable in the common block SPKUSR that must be explicitly assigned a value by the user is MAXS.

It is assumed that the subroutines which comprise SPARSPAK have been compiled into a <u>library</u>, and that the user can reference them from his FORTRAN program just as he references the standard FORTRAN library subroutines, such as SIN, COS, etc. Normally, a user will use only a small fraction of the subroutines provided in SPARSPAK.

<u>WARNING</u>

The modules of SPARSPAK communicate with each other through labelled common blocks whose names are SPKUSR, SPKSYS, SPKCON, SPKMAP, SPKDTA, and SPKOPS. Thus, the user must not use labelled common blocks with these names in his program.

If these common block names cause conflicts in your program or at your computer installation, it is possible to have the package distributed with these common blocks having <u>specifically requested</u> labels. These names should be specified when the package is acquired.

2.2 <u>MODULES FOR INPUT OF THE MATRIX STRUCTURE</u>

SPARSPAK has to know the matrix structure before it can determine an appropriate ordering for the system. We now describe the group of routines which provide a variety of ways through which the user can inform the package where the nonzero entries are; that is, those subscripts (i, j) for which the (i, j)-th element of A is nonzero. Before any of these input routines is called, the user must execute an initialization routine called IJBEGN, which tells the package that the structure of a new matrix problem is about to be input:

```
CALL IJBEGN
```

a) Input of a nonzero location.

To tell the package that the (i, j)-th element of A is nonzero, the user simply executes the statement

```
CALL  INIJ ( I, J, S )
```

where I and J are the subscripts of the nonzero, and S is the working storage array declared by the user for use by the package.

In this example,

```
      .
      .
      .
      I = 4
      J = 3
      CALL  INIJ ( I, J, S )
      .
      .
      .
```

the package will record a logical nonzero in the position (4, 3) of the matrix.

b) Input of the structure of a row, or part of a row.

When the structure of a row or part of a row is available, it is more efficient to use the routine INROW. The statement to use is

```
CALL  INROW ( I, NIR, IR, S )
```

where I denotes the subscript of the row under consideration, IR is an array containing the column subscripts of some or all of the nonzeroes in the I-th row, NIR is the number of subscripts in IR, and S is the user-declared working storage.

For example, in

```
    .
    .
    .
    I = 5
    IR(1)  = 2
    IR(2)  = 7
    IR(3)  = 5
    CALL  INROW ( I, 3, IR, S )
    .
    .
    .
```

the package is informed of nonzeroes in locations (5, 2), (5, 5) and (5, 7) of the matrix. Note that the subscripts in the array IR can be in arbitrary order, and the rows can be input in any order.

c) Input of a submatrix structure.

To provide greater flexibility, the package allows the user to input the structure of a submatrix. The calling statement is

```
CALL  INIJIJ ( NIJ, II, JJ, S )
```

where NIJ is the number of input subscript pairs, and II and JJ are the arrays containing the subscripts.

The following example

```
    .
    .
    .
    II(1)  = 1
    JJ(1)  = 1
    II(2)  = 1
    JJ(2)  = 3
    II(3)  = 2
    JJ(3)  = 3
    CALL  INIJIJ ( 3, II, JJ, S )
    .
    .
    .
```

informs the package that there are nonzeroes in locations (1, 1), (1, 3) and (2, 3).

d) Input of a full submatrix structure.

The structure of an entire matrix is completely specified if all the full submatrices are given. In applications where they are readily available, the routine INCLQ is useful. Its calling sequence is

```
CALL  INCLQ ( NCLQ, CLQ, S )
```

where NCLQ is the size of the submatrix and CLQ is an array containing the subscripts of the submatrix.

Thus, to inform the package that the submatrix corresponding to subscripts 1, 3, 5 and 6 is full, we execute

```
.
.
.
CLQ(1) = 1
CLQ(2) = 3
CLQ(3) = 5
CLQ(4) = 6
CALL  INCLQ ( 4, CLQ, S )
.
.
.
```

The type of structure input routine to use depends on how the user obtains the matrix structure. Anyway, the one or ones that best suit the application can be selected; SPARSPAK allows <u>mixed use</u> of the routines in inputting a matrix structure. The package automatically removes duplications so the user does not have to worry about inputting duplicated subscript pairs.

```
.
.
.
CLQ(1) = 1
CLQ(2) = 2
CLQ(3) = 5
CALL  INCLQ ( 3, CLQ, S )
IR(1) = 2
IR(2) = 4
IR(3) = 5
CALL  INROW ( 4, 3, IR, S )
CALL  INIJ ( 1, 3, S )
.
.
.
```

The above code would input the matrix structure

```
┌             ┐
│ *  *  *   * │
│ *  *      * │
│ *     *     │
│    *     * *│
│ *  *     * *│
└             ┘
```

into the package.

When all pairs have been input, using one or a combination of the input routines, the user is required to tell SPARSPAK explicitly that structure input is complete by calling the routine IJEND. The statement to use is

```
CALL  IJEND ( S )
```

and its purpose is to transform the data from the format used during the recording phase to the standard format used by the later phases. The user does not have to concern himself with this representation or transformation.

IMPORTANT NOTE:

SPARSPAK assumes that the value of NEQNS (the number of equations) is equal to the maximum subscript supplied by the routines which transmit the (i, j) pairs to the package. Thus, it is imperative that the user supplies at least one (i, j) pair for which i or j is equal to NEQNS. The routine IJEND assigns the value of NEQNS found by the package to the corresponding variable in the common block SPKUSR .

Common Errors

The most common cause of error during matrix structure input is insufficient working storage. If we denote the number of offdiagonal nonzeroes in the matrix by OFFDA, then the minimum amount of storage necessary to successfully input the structure is given by

```
OFFDA + 2*NEQNS + 1 .
```

Of course sometimes the user does not know the value of OFFDA, and may guess too low. SPARSPAK will still accept and count the (i, j) pairs, even after running out of storage, and the user can obtain an upper bound for OFFDA by calling the module PSTATS, described in Section 7, after all pairs have been input. (The number reported may be unnecessarily large because duplicate input pairs may not now be detected, and thus may be counted twice by the package.)

For a complete list of errors which may be generated by the structure input modules, see Section 7.3.1.

2.3 MODULES FOR ORDERING AND STORAGE ALLOCATION

With an internal representation of the nonzero structure of the matrix A available, SPARSPAK is ready to reorder the matrix problem. This is initiated by calling an ordering routine, whose name always has the form ORDRxi . Here i is a numerical digit between 1 and 6 that signifies the storage method. The character x can take values A or B , which denotes one of two ordering strategies tailored for storage method i .

Executing the statement

```
CALL   ORDRA1 ( S )
```

will imply the use of storage method 1 and the first ordering algorithm for this method. See Section 3 for a discussion of the various methods provided, and some guidance on which one to use. Section 8 contains a list of ordering strategies provided by the package. The routine ORDRxi not only determines an appropriate ordering for the storage method, it sets up the data structure for the reordered matrix problem. The package is now ready for numerical input.

Common Errors

Just as in the structure input phase, the most common cause of abnormal termination of the ORDRxi module is insufficient working storage. As mentioned above, this module actually performs two functions: ordering, and storage allocation. The ordering step determines the permutation P , and the allocation step sets up the appropriate data structures to store the triangular factors L and U of the permuted matrix PAP' .

In general, the ordering and allocation subroutines require different amounts of storage. Furthermore, their storage requirements are often unpredictable, because the number of data structure pointers, and the number of nonzeroes in the factors L and U , are not known until the subroutines have been executed.

Thus, the interface module ORDRxi may terminate in several distinctly different ways:

a) There was not enough storage to execute the ordering subroutine.
b) The ordering was successfully obtained, but there was insufficient storage to even initiate execution of the data structure set-up (storage allocation) subroutine.
c) The data structure set-up subroutine was executed, and the amount of storage required for the data structure pointers etc. was determined, but there was insufficient storage for these pointers.
d) The data structure was successfully generated, but there is insufficient storage for the actual numerical values, so the next step (input of the numerical values) cannot executed.
e) ORDRxi was successfully executed, and there is sufficient storage to proceed to the next step.

If any of the above conditions occurs, the user may execute SAVE, and re-initiate the computation after adjusting his storage declarations (either up or down) and executing RESTRT[3]. If a) or b) occurs, information is supplied indicating the minimum value of MAXS needed so that c), d) or e) will occur upon re-execution. If c) occurs, the minimum value of MAXS needed for d) and e) is provided.

When c) or d) occurs, after executing SAVE, adjusting our storage declaration, then executing RESTRT, we must again call ORDRxi . However, the interface will detect that the ordering and/or storage allocation have already been performed, and will skip that part of the computation. Note that if a user is simply using SPARSPAK to select a particular method, c) may be an acceptable termination state. (See Example 6 in Section 9.)

2.4 MODULES FOR INPUTTING NUMERICAL VALUES

The modules in this group are similar to those for inputting the matrix structure. They provide a means of transmitting the actual numerical values of the matrix problem to SPARSPAK. Since the data structures for different storage methods are different, the package must have a different matrix input subroutine for each method. For the user's convenience, SPARSPAK uses the same set of

[3] See Section 4 for details on how to use SAVE and RESTRT, and Examples 4, 5 and 6 in Section 9.

subroutine names for all the methods, except for the last digit which distinguishes the method, and the parameter lists for all the methods are the same.

IMPORTANT NOTE:

The elements of A and b transmitted to SPARSPAK by these routines are either single or double precision floating point numbers, depending on the version of SPARSPAK being used. The examples in this manual assume a single precision version of the package is being used.

There are three ways of passing the numerical values to SPARSPAK. In all of them, subscripts passed to the package always refer to those of the original given problem. The user need not be concerned about the various permutations to the problem which may have occurred during the ordering step.

a) Input of a single nonzero component.

The subroutine INAIJi is provided for this purpose and its calling sequence is

```
          CALL  INAIJi ( I, J, VALUE, S )
```

where I and J are the subscripts, and VALUE is the numerical value. The subroutine INAIJi adds the quantity VALUE to the appropriate current value in storage, rather than making an assignment. This is helpful in situations (e.g. in some finite element applications) where the numerical values are obtained in an incremental fashion.

For example, the execution of

```
      .
      .
      .
      CALL  INAIJ2 ( 3, 4, 9.5, S )
      CALL  INAIJ2 ( 3, 4, -4.0, S )
      .
      .
      .
```

effectively assigns 5.5 to the (3, 4)-th component of A .

b) Input of a row of nonzeroes.

The routine INROWi can be used to input the numerical values of a row or part of a row in the matrix. Its calling sequence is similar to that of INROW, described on Section 2.2:

```
CALL  INROWi ( I, NIR, IR, VALUES, S )
```

Here the additional parameter VALUES is a floating point array containing the numerical values of the row. Again, the numerical values are added to the current values in storage.

c) Input of a submatrix.

The routine that allows the input of a submatrix is INMATi . Its parameter list corresponds to that of IN1JIJ with the additional parameter VALUES that stores the numerical quantities:

```
CALL  INMATi ( NIJ, II, JJ, VALUES, S )
```

Again, the numerical values in VALUES are added to those currently held by the package.

Mixed use of the routines INAIJi, INROWi and INMATi is permitted. Thus, the user is free to use whatever routine is most convenient.

The same convenience is provided in the input of numerical values for the right hand side vector b . SPARSPAK includes the routine INBI which inputs an entry of the right hand side vector

```
CALL  INBI ( I, VALUE, S )
```

Here I is the subscript and VALUE is the numerical value. Alternatively, the routine INBIBI can be used to input a subvector, and its calling sequence is

```
CALL  INBIBI ( NI, II, VALUES, S )
```

where NI is the number of input numerical values, and II and VALUES are vectors containing the subscripts and numerical values respectively. In both routines, incremental calculation of the numerical values is performed.

In some situations where the entire right hand side vector is available, the user can use the routine INRHS which transmits the whole vector to SPARSPAK. It has the form

```
CALL INRHS ( RHS, S )
```

where RHS is the vector containing the numerical values.

In all three routines, the numbers provided are added to those currently held by the package, and the use of the routines can be intermixed. The storage used for the right hand side by SPARSPAK is initialized to zero the first time any of them is executed.

<u>IMPORTANT NOTES</u>:

a) When the matrix A is symmetric, so that method i, with i odd, is used, SPARSPAK requires that the elements of the <u>lower</u> triangle be provided. Thus, for example, the following statement will cause an error.

```
CALL INAIJ3 ( 3, 5, 1.3, S )
```

b) The examples which we have given assume that a single precision version of SPARSPAK is being used. If the version is in double precision, the numerical values and numerical variables should be declared as double precision. For example:

```
CALL INAIJ3 ( 5, 3, 1.3D0, S )
```

2.5 <u>MODULES FOR NUMERICAL SOLUTION</u>

The numerical computation of the solution vector is initiated by the FORTRAN statement

```
CALL SOLVEi ( S )
```

where S is the working storage array for SPARSPAK. Again, the last digit i is used to distinguish between solvers for different storage methods.

Internally, the routine SOLVEi consists of both the factorization and forward/backward solution steps. If the factorization has been performed in a previous call to SOLVEi, SPARSPAK will automatically skip the factorization step, and perform the solution step directly. The solution vector is returned in the first NEQNS locations of the storage vector S . If SOLVEi is called before any right hand side values are input, only the factorization will be performed. The solution returned will be all zeroes. See Examples 3 and 4 in Section 9.

SECTION 3

SOME GUIDELINES ON SELECTING A METHOD

We mentioned in Section 1 that there are six basic methods, distinguished by a numerical digit i satisfying $1 \le i \le 6$. These six methods can be viewed as grouped into three odd-even pairs; the only distinction between method i (odd) and method i+1 is that method i assumes A is symmetric, and method i+1 assumes A is unsymmetric. Thus, we really only provide three essentially distinct methods, with each one having a symmetric and unsymmetric version. Hence, in this section we will largely confine our remarks to methods 1, 3 and 5; comparative remarks about them will also apply to their unsymmetric analogues, methods 2, 4 and 6.

The basic methods are as follows; the remarks comparing them, and the advice provided, should be regarded as at best tentative. Characteristics of sparse matrices vary a great deal.

Method	Basic Strategy and References
1, 2	The objective of these methods is to reorder A so it has a small bandwidth or profile [6]. The well-known reverse Cuthill-McKee algorithm is used. For relatively small problems, say $N \le 200$, they are probably the best overall methods to use.
3, 4	The objective of these methods is to reduce storage requirements, but the factorization time will usually be substantially higher than any of the other methods. Their storage requirements will usually be substantially less than methods (1, 2) (unless N is very large). The same remark is true about the relative solution times. Thus, these methods are often useful when storage is restricted, and/or when many problems which differ only in the right hand side must be solved (see Section 6). There are two ordering options provided: ORDRA3 and ORDRB3 (and similarly for the unsymmetric case). The A option is

specifically tailored for 'finite element problems', typical of those arising in structural analysis and the numerical solution of partial differential equations [1]. The B option is effective for less specific problems; and uses a refined quotient tree ordering described in [2].

5, 6 These methods attempt to find orderings which minimize fill-in, and they exploit all zeroes. Their ordering times are almost always greater than those above, but for moderate-to-large problems the reduced factorization times usually are more than compensatory.

Just as for methods (3, 4), there are two ordering options provided. The A option is again specifically designed for finite element problems, and uses a so-called nested dissection ordering [3]. The B option uses the minimum degree algorithm, and is suitable for all sparse problems [4].

To summarize, our tentative advice and guidelines are as follows:

1. For small problems, use method (1, 2).
2. For small to moderate size problems that have to be solved only once, use method (1, 2) if enough storage is available. If not, use method (3, 4). If the problem is quite large, method (5, 6) might be better.
3. For moderate to large problems, use either method (3, 4) or (5, 6). If many problems differing only in the right hand side must be solved, method (3, 4) is probably the best. If the problem is quite large, and many problems having the same structure, but different numerical values, must be solved, then method (5, 6) is probably the best. (See Sections 5 and 6.)

SPARSPAK has been designed so that the ORDRxi modules can be used as aids in selecting a method. The basic strategy, as illustrated in Example 6 in Section 9, is to input the matrix structure, and then run the various ORDRxi modules on it, printing the storage statistics gathered by the package (using PSTATS, described in Section 7) after each ordering module has been executed.

SECTION 4

SAVE AND RESTART FACILITIES

SPARSPAK provides two subroutines called SAVE and RESTRT which allow the user to stop the calculation at some point, save the results on an external sequential file, and then restart the calculation at exactly that point some time later. To save the results of the computation done thus far, the user executes the statement

```
CALL  SAVE ( K, S )
```

where K is the FORTRAN logical unit on which the results are to be written, along with other information needed to restart the computation. If execution is then terminated, the state of the computation can be re-established by executing the statement

```
CALL  RESTRT ( K, S )
```

Examples 4, 5 and 6 provided in Section 9 illustrate the use of SAVE and RESTRT.

Note that executing SAVE does not destroy any information; the computation can proceed just as if SAVE was not executed.

When errors occur in a module, the routines SAVE and RESTRT are useful in saving the results of previously successfully executed modules (see Section 7.3 and Example 5 in Section 9).

Another potential use of the SAVE and RESTRT modules is to make the working storage array S available to the user in the middle of a sparse matrix computation. After SAVE has been executed, the working storage array S can be used by some other computation.

Finally, the SAVE and RESTRT modules allow the user to segment the computation into several distinct phases, and thereby reduce the amount of program that must be resident in storage at any given time.

IMPORTANT NOTES:

a) In the subroutines SAVE and RESTRT, information is either written on or read from the FORTRAN logical unit K using binary format.

b) If the subroutines SAVE and RESTRT are used, then before the user executes his program, he must define a file for the FORTRAN logical unit K using the appropriate system control card or command (this depends on the environment in which the program is being executed). Furthermore, this file must be preserved by the user for later access by the RESTRT subroutine.

SECTION 5

SOLVING MANY PROBLEMS HAVING THE SAME STRUCTURE

In certain applications, many problems which have the same sparsity structure, but different numerical values, must be solved. In this case, the structure input, ordering, and data structure set-up needs only to be done once. This situation can be accommodated perfectly well by SPARSPAK. The control sequence is depicted by the following flowchart:

```
                 ┌------------┐
                 |   SPRSPK   |
                 └------------┘
                       |
                       |
                       V
             ┌--------------------┐
             |  Input Structure   |
             |      of  A         |
             └--------------------┘
                       |
                       |
                       V
               ┌----------------┐
               |  CALL ORDRxi   |
               └----------------┘
                       |
                       |
                       V
           ┌--------------------------┐
           |     Input Numerical      |
      ┌-->|   Values of  A  and  b   |
      |    └--------------------------┘
      |                |
      |                |
      |                V
      |        ┌----------------┐
      |        |  CALL SOLVEi   |
      |        └----------------┘
      |                |
      |                |
      └----------------┘
```

When the numerical input routines (INAIJi, INBI, ..., etc.) are first called after SOLVEi has been called, this is detected by SPARSPAK, and the computer storage used for A and b is initialized to zero.

Note that if such problems must be solved over an extended time period (i.e., in different runs), the user can execute SAVE after executing ORDRxi and thus avoid the input of the structure of A and the execution of ORDRxi in subsequent equation solutions.

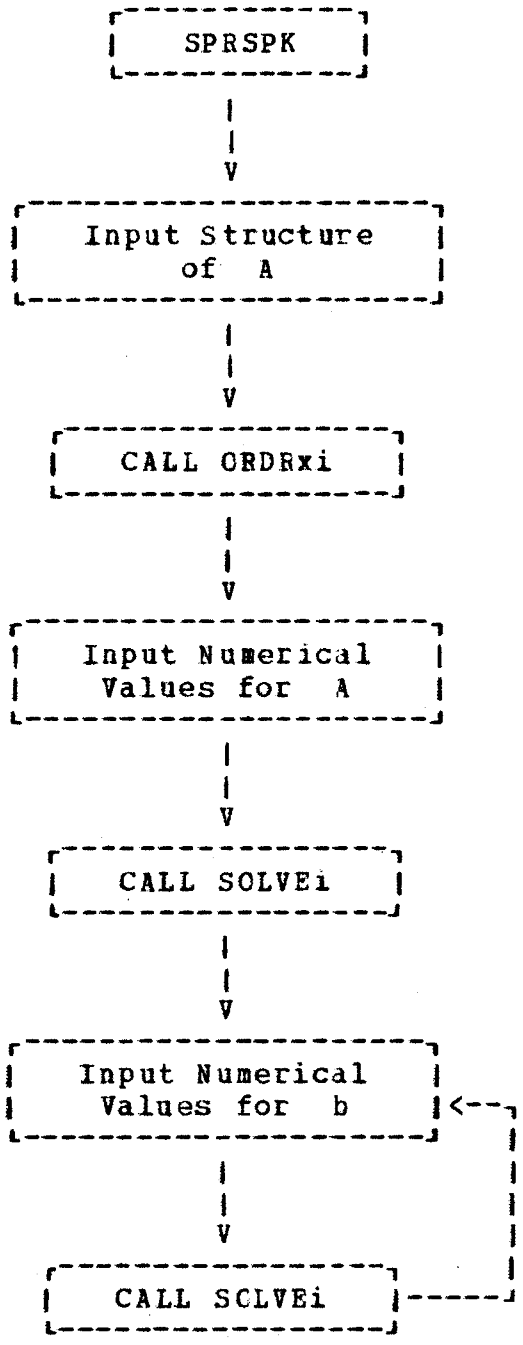
SECTION 6

SOLVING MANY PROBLEMS WHICH DIFFER ONLY IN THEIR RIGHT HAND SIDE

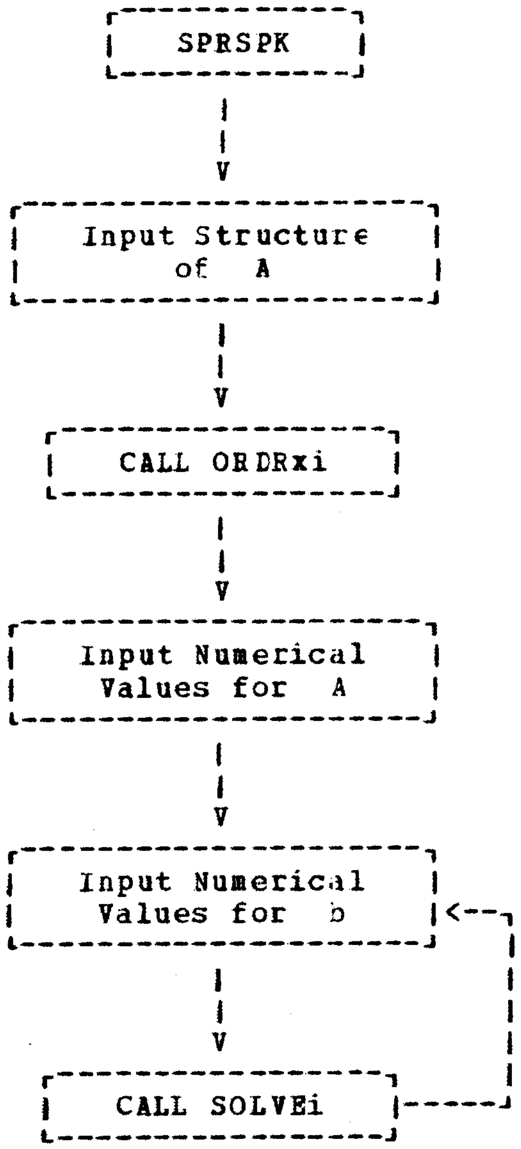
In some applications, numerous problems which differ only in their right hand sides must be solved. In this case, we only want to factor A into LU (or LL') once, and use the factors repeatedly in the calculation of x for each different b . Again, SPARSPAK can handle this situation in a straightforward manner, as illustrated by the flowcharts on the following page.

When SPARSPAK is used as indicated by flowchart (1), the package detects that no right hand side has been provided during the first execution of SOLVEi , and only the factorization is performed. In subsequent calls to SOLVEi , SPARSPAK detects that the factorization has already been performed, and that part of the SOLVEi module is bypassed. In flowchart (2), both factorization and solution are performed during the first call to SOLVEi , with only the solve part performed in subsequent executions of SOLVEi . (See Example 3 in Section 9.)

Note that SAVE can be used after SOLVEi has been executed, if the user wants to save the factorization for use in some future calculation.


SPRSPK
Input Structure of A
CALL ORDRxi
Input Numerical Values for A
CALL SOLVEi
Input Numerical Values for b
CALL SOLVEi

(1)


SPRSPK
Input Structure of A
CALL ORDRxi
Input Numerical Values for A
Input Numerical Values for b
CALL SOLVEi

(2)

SECTION 7

OUTPUT FROM SPARSPAK

As noted earlier in Section 2, the user supplies a one-dimensional floating point array S , from which all array storage is allocated. In particular, the interface allocates the first NEQNS storage locations in S for the solution vector of the linear system. After all the interface modules for a particular method have been successfully executed, the user can retrieve the solution from these NEQNS locations.

In addition to the solution x , SPARSPAK may print other information about the computation, depending upon the value of MSGLVL , whether or not errors occur, and whether or not the module PSTATS is called. This section discusses these features of SPARSPAK.

NOTE:

SPARSPAK writes output to two FORTRAN logical output units, whose numbers are given by IPRNTS and IPRNTE. The values for these variables are set in the module SPRSPK when the package is installed. Standard output requested by the user is printed on unit IPRNTS, while any error messages raised by SPARSPAK are printed on unit IPRNTE. In an interactive environment, IPRNTE is usually the user's terminal, while IPRNTS is some other output device on which the output of the (hopefully) successful run is recorded. In a batch oriented environment, IPRNTS and IPRNTE are usually the same. Note that the user and/or the computer installation must ensure that the files associated with IPRNTS and IPRNTE are available to the user's program before execution begins.

7.1 MESSAGE LEVEL (MSGLVL)

The first variable MSGLVL in the common block SPKUSR stands for 'message level', and governs the amount of information printed by the interface modules. Its default value is two, and for this value a relatively small amount of summary information is printed, indicating the initiation of each phase. When MSGLVL is set to one by the user, only fatal error messages are printed; this option could be useful if SPARSPAK is being used in the 'inner loop' of a large computation, where even summary information would generate excessive output. Increasing the value of MSGLVL (up to 4) provides increasingly detailed information about the computation. Note that the module SPRSPK sets MSGLVL to its default value; if the user wishes MSGLVL to be different from 2, he must reset it _after_ SPRSPK has been called.

In many circumstances, SPARSPAK will be imbedded in still another 'super' package which models phenomena producing sparse matrix problems. Messages printed by SPARSPAK may be useless or even confusing to the ultimate users of the super package, or the super package may wish to field the error conditions and perhaps take some corrective action which makes the error messages erroneous. Thus, _all_ printing by SPARSPAK can be inhibited by setting MSGLVL to zero.

To summarize, we have

MSGLVL	Printed Output
0	No messages
1	Fatal error messages
2	Minimal summary information
3	More detailed information
4	Detailed debugging information

7.2 STATISTICS GATHERING (PSTATS)

SPARSPAK gathers a number of statistics which the user will find useful if he is comparing various methods, or is going to solve numerous similar problems and wants to adjust his working storage to the minimum necessary. The package has a common block called SPKDTA containing variables whose values can be printed by executing the statement

```
CALL  PSTATS
```

The information printed is:

the number of equations,
the number of off-diagonal nonzeroes in the matrix,
the size of the working storage,
the time used to find the ordering,
the time used for data structure set-up,
the time used for the factorization step,
the time used for the triangular solution step,
number of operations required by the factorization step,

number of operations required by the triangular solution,
the storage used by the ordering subroutine,
the storage used by the data structure set-up subroutine,
the storage used by the SOLVEi module.

Since the module PSTATS can be called at any time, some of the above information may not be available, and will not be printed. The word 'operations' here means multiplicative operations (multiplications and divisions). Since most of the arithmetic performed in sparse matrix computation occurs in multiply-add pairs, the number of operations (as defined here) is a useful measure of the amount of arithmetic performed.

The reader is referred to the examples in Section 9 for more discussion about the output from PSTATS.

7.3 ERROR MESSAGES (IERR)

When a fatal error is detected, so that the computation cannot proceed, a positive code is assigned to IERR . The user can simply check the value of IERR to see if the execution of module has been successful. This error flag can be used in conjunction with the save/restart feature described in Section 4 to retain the results of successfully completed parts of the computation, as shown by the program fragment below.

```
          .
          .
          .
          CALL  ORDRA1 ( S )
          IF  ( IERR .EQ. 0 )   GO TO 100
          CALL  SAVE ( 3, S )
          STOP
      100 CONTINUE
          .
          .
          .
```

The variable IERR is set to the value 10*k+l , where $0 \leq l \leq 9$ distinguishes the error, and k is determined by the type of module that sets IERR positive:

k	
0	save and restart modules (SAVE, RESTRT)
1	matrix structure input modules (INIJ, INIJIJ, etc.)
2	matrix ordering and allocation modules (ORDRxi)
3	matrix numerical input (INAIJi, ..., etc.)
4	right hand side numerical input (INBI, ..., etc.)
5	factorization and solution modules (SOLVEi)

7.3.1 Save and Restart Routines

IERR	RESTRT
1	Insufficient storage to restart the computational process. The minimum value of MAXS required is printed in the error message.

7.3.2 Input of the Matrix Structure

IERR	INIJ, INIJIJ, INCLQ
11	Insufficient storage was provided in the working storage array. The (i, j) pairs input to INIJ, INIJIJ, and INCLQ will be counted and discarded. Duplicates which are detected will not be counted, but some duplicates may be missed.
12	Negative or zero subscript is found.
13	Incorrect execution sequence. Probable cause of error: routine IJBEGN was not called before (i, j) pairs input began.

IERR	IJEND
16	Insufficient storage to transform matrix structure. The minimum value of MAXS required is printed in the error message.
17	Incorrect execution sequence. IJEND was called before new matrix structure has been input.
18	NEQNS is zero.

7.3.3 Ordering and Storage Allocation Routines

IERR

21	Incorrect execution sequence. Probable cause: subroutine IJEND did not execute successfully.
22	Incompatible ordering method. User probably executed part of the ordering subroutine ORDRxi, and then executed SAVE because of insufficient storage. The execution was then restarted, using RESTRT , but ORDRxj was called with $i \neq j$.
23	Insufficient storage in working storage array to begin execution Response: execute SAVE , and call ORDRxi with MAXS at least as large as that indicated in the error message.
24	Insufficient storage in working storage array to execute the storage allocation subroutine. The ordering routine has successfully executed. Response: same as for error 23.
25	Working storage array was not large enough. The storage allocation routine was executed, but there was not enough storage to hold the data structure pointers. Response: same as for error 23.
26	Working storage array is large enough for execution of ORDRxi , and it has successfully executed. However, there is not enough storage available for the numerical values, so computation cannot proceed. Response: execute SAVE , and re-initiate computation after adjusting MAXS to at least the value specified in the error message.

7.3.4 Input of the Numerical Values

IERR	INAIJi, INROWi, INMATi
31	Incorrect execution sequence. Probable cause: unsuccessful execution of the ordering routine ORDRxi .
32	Incompatible input routine - attempt to use input routine INAIJi, INFCWi, or INMATi after using ORDRxj , where $i \neq j$. Use the routine specified in the error message.
33	Attempt to input the (i, j)-th element of matrix A for $i < j$. (This error occurs only for symmetric matrix methods; i.e., when method is odd). Methods for symmetric matrices expect elements of the lower triangle to be input.
34	Attempt to input an (i, j)-th element of matrix A where $i > N$, $j > N$, $i < 1$, or $j < 1$.
35	Attempt to input a numerical value for the (i, j)-th element of matrix A into the data structure, but the data structure has no space for it. Probable cause: the user has not called INIJ, INIJIJ, INCLQ or INROW with all the pairs (i, j) for which the (i, j)-th elements of A are nonzero. (SPARSPAK thinks A is sparser than it really is.)

IERR	INBI, INBIBI, INRHS
41	Incorrect execution sequence. Probable cause is the unsuccessful execution of ORDRxi .
42	Subscript out of range - attempt to input a numerical value for the i-th element of b where $i > N$ or $i < 1$.

7.3.5 Factorization and Solution

IERR	SOLVEi
51	Incorrect execution sequence. Probable cause is unsuccessful execution of the numerical input routines.
52	Incompatible ordering and solution routines have been called. Response: execute SAVE and restart the computation using SOLVEi where i is the value of METHOD specified in the error message.
53	Zero pivot or negative square root detected in the factorization routine. Possible causes: a) incorrect use of the numerical input routines. b) the matrix may require pivoting in order to preserve numerical stability. In this case the use of SPARSPAK to solve the problem is inappropriate. (See restrictions in Section 1.)

SECTION 8

SUMMARY LISTING OF INTERFACE ROUTINES

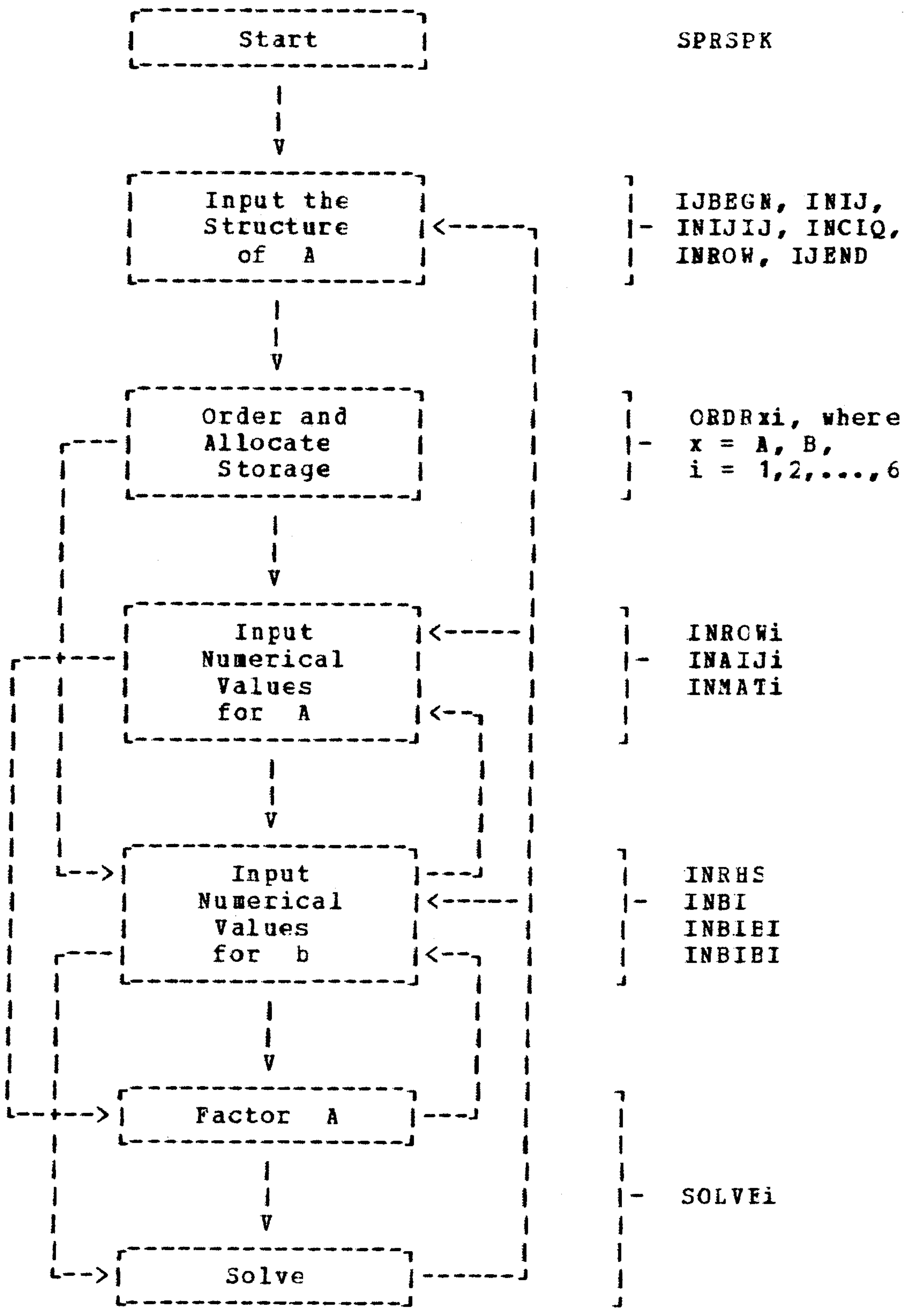
SPRSPK	Initialization of SPARSP
IJBEGN INIJ (I, J, S) INROW (I, NIR, IR, S) INIJIJ (NIJ, II, JJ, S) INCLQ (NCLQ, CLQ, S) IJEND (S)	Structure input
ORDRxi (S)	Ordering (see next page)
INAIJi (I, J, VALUE, S) INROWi (I, NIR, IR, VALUES, S) INMATi (NIJ, II, JJ, VALUES, S)	Matrix input
INBI (I, VALUE, S) INBIBI (NI, II, VALUES, S) INRHS (RHS, S)	Right hand side input
SOLVEi (S)	Factorization and/or Solution
PSTATS	Print statistics
SAVE (k, S) RESTRT (k, S)	Save and Restart the computation

Ordering Choices

ORDRxi

x	i	
A	1	Reverse Cuthill-McKee ordering [7]; symmetric A
A	2	Reverse Cuthill-McKee ordering [7]; unsymmetric A
A	3	One-way Dissection ordering [1]; symmetric A
A	4	One-way Dissection ordering [1]; unsymmetric A
B	3	Refined quotient tree ordering [2]; symmetric A
B	4	Refined quotient tree ordering [2]; unsymmetric A
A	5	Nested Dissection ordering [3]; symmetric A
A	6	Nested Dissection ordering [3]; unsymmetric A
B	5	Minimum Degree ordering [4]; symmetric A
B	6	Minimum Degree ordering [4]; unsymmetric A

Sketch of Possible Execution Paths through SPARSPAK Modules


Start
SPRSPK
Input the Structure of A
IJBEGN, INIJ, INIJIJ, INCIQ, INROW, IJEND
Order and Allocate Storage
ORDRxi, where x = A, B, i = 1,2,...,6
Input Numerical Values for A
INROWi
INAIJi
INMATi
Input Numerical Values for b
INRHS
INBI
INBIBI
INBIBI
Factor A
Solve
SOLVEi

SECTION 9

EXAMPLES

In this section, we provide several programs which illustrate how SPARSPAK can be used. These programs are derived from the one given in Section 2.1.

These examples were run using a standard single precision version of SPARSPAK under the IBM FORTRAN H extened compiler on an IBM 3031 computer. All times reported are in seconds. It should be noted that the results will be different if a different version of SPARSPAK is used.

Example 1

This is an example of the simplest use of SPARSPAK, with each of the modules of method 1 used in sequence. The problem that is solved is a 10 by 10 symmetric system Ax = b where the diagonal elements of A are all 4, and the superdiagonal and subdiagonal elements are all -1. The right hand side vector b is chosen so that the entries of the solution vector x are all ones.

In the program, the nonzero structure of A is input using IJBEGN, INIJ and IJEND . After ORDRA1 is executed, the interface modules INAIJ1 and INBI are used to transmit the numerical values of A and b to the package respectively. The module SOLVE1 is called to do the numerical solution and then PSTATS is called to print out the statistics gathered by the interface during execution. Finally, the error in the computed approximate solution is computed.

Note that the size of the working storage provided was 250, while the maximum amount used by any of the modules was 60, which was the storage requirement for the ORDRA1 and SOLVE1 module. Thus, if the user was going to solve this problem again, he could adjust his storage down to 60.

```
C          M A I N L I N E    P R O G R A M
C
      REAL          S(250), ERROR, FOUR, ONE, TWO, ZERO
      INTEGER       I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL          RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 250
          CALL  IJBEGN
          DO  100  I = 2, 10
              CALL  INIJ ( I, I-1, S )
  100     CONTINUE
          CALL  IJEND ( S )
          CALL  ORDRA1 ( S )
          ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          FOUR = 4.0E0
          DO  200  I = 1, 10
              IF ( I .GT. 1 )  CALL  INAIJ1 ( I, I-1, -ONE, S )
              CALL  INAIJ1 ( I, I, FOUR, S )
              CALL  INBI ( I, TWO, S )
  200     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 10, ONE, S )
          CALL  SOLVE1 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  300  I = 1, 10
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  300     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********        RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980


        OUTPUT UNIT FOR ERROR MESSAGE        6
        OUTPUT UNIT FOR STATISTICS           6

   IJBEGN- BEGIN STRUCTURE INPUT...

   INIJ-   INPUT OF ADJACENCY PAIRS

   IJEND-  END OF STRUCTURE INPUT

   ORDRA1- RCM ORDERING

   INAIJ1- INPUT OF MATRIX COMPONENTS

   INBI-   INPUT OF RIGHT HAND SIDE

   SOLVE1- ENVELOPE SOLVE

   PSTATS- STATISTICS
              NUMBER OF EQUATIONS                    10
              OFF-DIAGONAL NONZEROS                  18
              SIZE OF WORKING STORE (MAXS)          250
              TIME FOR ORDERING                      0.003
              STORAGE FOR ORDERING                  60.
              TIME FOR ALLOCATION                    0.0
              STORAGE FOR ALLOCATION                60.
              STORAGE FOR SOLUTION                  60.
              TIME FOR FACTORIZATION                 0.0
              TIME FOR SOLVING                       0.003
              OPERATIONS IN FACTORIZATION           18.
              OPERATIONS IN SOLVING                 38.

MAXIMUM ERROR        1.013E-06
```

Example 2

This is the same as Example 1, except that the matrix A is unsymmetric. The diagonal elements of A are all 4, the superdiagonal elements are all 1, and the subdiagonal elements are all -1. The right hand side vector b is chosen so that the entries of the solution vector x are all ones.

```
C          M A I N L I N E     P R O G R A M
C
      INTEGER       I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL          S(250), ERROR, FOUR, ONE, ZERO
      REAL          RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 250
          CALL  IJBEGN
          DO  100  I = 2, 10
              CALL  INIJ ( I, I-1, S )
  100     CONTINUE
          CALL  IJEND ( S )
          CALL  ORDRA2 ( S )
          ZERO = 0.0E0
          ONE  = 1.0E0
          FOUR = 4.0E0
          DO  200  I = 1, 10
              IF ( I .GT. 1 )  CALL  INAIJ2 ( I, I-1, -ONE, S )
              IF ( I .LT. 10 )  CALL  INAIJ2 ( I, I+1, ONE, S )
              CALL  INAIJ2 ( I, I, FOUR, S )
              CALL  INBI ( I, FOUR, S )
  200     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 10, -ONE, S )
          CALL  SOLVE2 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  300  I = 1, 10
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  300     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
*********** UNIVERSITY OF WATERLOO
*********** SPARSE MATRIX PACKAGE
***********  ( S P A R S P A K )
***********       RELEASE  2
***********     (C)  JANUARY 1979
*********** STANDARD VERSION
*********** SINGLE PRECISION
*********** LAST UPDATE JANUARY 1980


         OUTPUT UNIT FOR ERROR MESSAGE        6
         OUTPUT UNIT FOR STATISTICS           6

    IJBEGN- BEGIN STRUCTURE INPUT...

    INIJ-   INPUT OF ADJACENCY PAIRS

    IJEND-  END OF STRUCTURE INPUT

    ORDRA2- RCM ORDERING

    INAIJ2- INPUT OF MATRIX COMPONENTS

    INBI-   INPUT OF RIGHT HAND SIDE

    SOLVE2- ENVELOPE SOLVE

    PSTATS- STATISTICS
               NUMBER OF EQUATIONS                  10
               OFF-DIAGONAL NONZEROS                18
               SIZE OF WORKING STORE (MAXS)        250
               TIME FOR ORDERING                     0.003
               STORAGE FOR ORDERING                60.
               TIME FOR ALLOCATION                   0.0
               STORAGE FOR ALLOCATION              60.
               STORAGE FOR SOLUTION                69.
               TIME FOR FACTORIZATION                0.003
               TIME FOR SOLVING                      0.0
               OPERATIONS IN FACTORIZATION         18.
               OPERATIONS IN SOLVING               28.

MAXIMUM ERROR       0.0
```

Example 3

This is similar to Example 1, except that method 3 is used (with the A ordering option), and two problems differing only in their right hand sides are solved. After solving the problem whose solution vector contains all ones, a new right hand side is input which corresponds to a different problem whose solution vector contains all twos. When the module SOLVE3 is called a second time, the interface detects that the factorization has already been done, and only the triangular solution is performed.

```
C          M A I N L I N E     P R O G R A M
C
      INTEGER        I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL           S(250), ERROR, FOUR, ONE, TWO, ZERO
      REAL           RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 250
          CALL  IJBEGN
          DO  100  I = 2, 10
              CALL  INIJ ( I, I-1, S )
  100     CONTINUE
          CALL  IJEND ( S )
          CALL  ORDRA3 ( S )
          ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          FOUR = 4.0E0
          DO  200  I = 1, 10
              IF ( I .GT. 1 )  CALL  INAIJ3 ( I, I-1, -ONE, S )
              CALL  INAIJ3 ( I, I, FOUR, S )
              CALL  INBI ( I, TWO, S )
  200     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 10, ONE, S )
          CALL  SOLVE3 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  300  I = 1, 10
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  300     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
C
          DO  400  I = 1, 10
              CALL  INBI ( I, FOUR, S )
  400     CONTINUE
          CALL  INBI ( 1, TWO, S )
          CALL  INBI ( 10, TWO, S )
          CALL  SOLVE3 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  500  I = 1, 10
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-TWO ) )
  500     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********       RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980


        OUTPUT UNIT FOR ERROR MESSAGE        6
        OUTPUT UNIT FOR STATISTICS           6

    IJBEGN- BEGIN STRUCTURE INPUT...

    INIJ-   INPUT OF ADJACENCY PAIRS

    IJEND-  END OF STRUCTURE INPUT

    ORDRA3- ONE WAY DISSECTION ORDERING

    INAIJ3- INPUT OF MATRIX COMPONENTS

    INBI-   INPUT OF RIGHT HAND SIDE

    SOLVE3- IMPLICIT BLOCK SOLVE

    PSTATS- STATISTICS
               NUMBER OF EQUATIONS                     10
               OFF-DIAGONAL NONZEROS                   18
               SIZE OF WORKING STORE (MAXS)           250
               TIME FOR ORDERING                        0.007
               STORAGE FOR ORDERING                   91.
               TIME FOR ALLOCATION                      0.0
               STORAGE FOR ALLOCATION                 94.
               STORAGE FOR SOLUTION                   94.
               TIME FOR FACTORIZATION                   0.003
               TIME FOR SOLVING                         0.003
               OPERATIONS IN FACTORIZATION            18.
               OPERATIONS IN SOLVING                  38.

MAXIMUM ERROR        1.013E-06

    INBI-   INPUT OF RIGHT HAND SIDE

    SOLVE3- IMPLICIT BLOCK SOLVE
                    FACTORIZATION ALREADY DONE.

    PSTATS- STATISTICS
               NUMBER OF EQUATIONS                     10
               OFF-DIAGONAL NONZEROS                   18
               SIZE OF WORKING STORE (MAXS)           250
               TIME FOR ORDERING                        0.007
               STORAGE FOR ORDERING                   91.
```

```
                    TIME FOR ALLOCATICN                        0.0
                    STORAGE FOR ALLOCATION                    94.
                    STORAGE FCR SCLUTION                      94.
                    TIME FOR FACTCRIZATION                     0.003
                    TIME FOR SCLVING                           0.003
                    OPERATIONS IN FACTORIZATION               18.
                    OPERATICNS IN SCLVING                     38.

MAXIMUM ERROR          1.907E-06
```

Example 4

This example illustrates the use of the save/restart feature of SPARSPAK. After the factorization is computed, SAVE is executed, which writes the current state of the computation on FORTRAN logical unit 3. In a second program the module RESTRT is executed to read the information from unit 3, and the computation resumes at the point at which SAVE was invoked.

```
C           M A I N L I N E     P R O G R A M
C
      INTEGER       I, IERR, MAXS, MSGLVL, NEQNS
      REAL          S(250), ERROR, FOUR, ONE
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
C
          CALL  SPRSPK
          MAXS = 250
          CALL  IJBEGN
          DO  100  I = 2, 10
              CALL  INIJ ( I, I-1, S )
  100     CONTINUE
          CALL  IJEND ( S )
          CALL  ORDRA1 ( S )
          ONE  = 1.0E0
          FOUR = 4.0E0
          DO  200  I = 1, 10
              IF ( I .GT. 1 )   CALL  INAIJ1 ( I, I-1, -ONE, S )
             CALL  INAIJ1 ( I, I, FOUR, S )
  200     CONTINUE
          CALL  SOLVE1 ( S )
          CALL  PSTATS
          CALL  SAVE ( 3, S )
          STOP
      END
```

```
********** UNIVERSITY OF WATERLCC
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********        RELEASE  2
**********      (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980


         OUTPUT UNIT FOR ERRCR MESSAGE         6
         OUTPUT UNIT FOR STATISTICS            6

    IJBEGN- BEGIN STRUCTURE INPUT...

    INIJ-   INPUT OF ADJACENCY PAIRS

    IJEND-  END OF STRUCTURE INPUT

    CRDRA1- RCM ORDERING

    INAIJ1- INPUT OF MATRIX COMPONENTS

    SOLVE1- ENVELOPE SCLVE
                    NO RIGHT HAND SIDE PROVIDED,
                    SOLUTION WILL BE ALL ZEROS.

    PSTATS- STATISTICS
               NUMBER OF EQUATIONS                      10
               OFF-DIAGONAL NONZEROS                    18
               SIZE OF WCRKING STORE (MAXS)            25C
               TIME FCR CRDERING                        0.003
               STORAGE FOR ORDERING                    60.
               TIME FCR ALLOCATION                      0.C
               STORAGE FOR ALLCCATION                  60.
               STORAGE FCR SCLUTION                    60.
               TIME FOR FACTORIZATION                   0.0
               TIME FCR SCLVING                         0.C
               OPERATIONS IN FACTORIZATION             18.
               OPERATIONS IN SCLVING                    0.

    SAVE-   STORAGE VECTCR SAVED
```

```
C            M A I N L I N E     P R O G R A M
C
      INTEGER        I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL           S(250), ERROR, ONE, TWO, ZERO
      REAL           RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 250
          CALL  RESTRT ( 3, S )
          ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          DO  100  I = 1, 10
             CALL  INBI ( I, TWO, S )
  100     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 10, ONE, S )
          CALL  SOLVE1 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  200  I = 1, 10
             ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  200     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********        RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980


        OUTPUT UNIT FOR ERROR MESSAGE        6
        OUTPUT UNIT FOR STATISTICS           6

    RESTRT- RESTART SYSTEM

    INBI-   INPUT OF RIGHT HAND SIDE

    SOLVE1- ENVELOPE SOLVE
                    FACTORIZATION  ALREADY DONE.

    PSTATS- STATISTICS
               NUMBER OF EQUATIONS                        10
               OFF-DIAGONAL NONZEROS                      18
               SIZE OF WORKING STORE (MAXS)              250
               TIME FOR ORDERING                          0.003
               STORAGE FOR ORDERING                      60.
               TIME FOR ALLOCATION                        0.0
               STORAGE FOR ALLOCATION                    60.
               STORAGE FOR SOLUTION                      60.
               TIME FOR FACTORIZATION                     0.0
               TIME FOR SOLVING                           0.0
               OPERATIONS IN FACTORIZATION               18.
               OPERATIONS IN SOLVING                     38.

MAXIMUM ERROR        1.013E-06
```

Example 5

This example consists of four runs of essentially the same program, illustrating how the SAVE and RESTRT modules can be used to avoid repeating successfully completed computations when the execution cannot proceed further because of lack of working storage. In the first run, MAXS was too small to accommodate the structure, and a message was printed indicating that MAXS must be at least 999 in order to input the structure. A second run with MAXS = 999 was executed, and the structure was successfully input; however, the ORDRA5 module could not execute because MAXS was less than 1400 . The module SAVE was then executed and the run terminated.

The third run had MAXS = 2500 , and the ordering and storage allocation were successfully performed. However, ORDRA5 terminated with an error because it detected that too little storage was available for the numerical computation (SOLVE5), so SAVE was again executed. Finally, the last run was executed with MAXS set to 2509 (the maximum value, printed in the third run), and the solution to the problem was obtained.

NOTE:
The following examples were run using a single precision version of SPARSPAK. The working storage required will therefore be different if a different version of SPARSPAK is used.

```
C          M A I N L I N E    P R O G R A M
C
      INTEGER       I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL          S(900), ERROR, FOUR, ONE, TWO, ZERO
      REAL          RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 900
          CALL  IJBEGN
          DO  100  I = 2, 200
              CALL  INIJ ( I, I-1, S )
  100     CONTINUE
          CALL  IJEND ( S )
          IF ( IERR .EQ. 0 )   GO TO 200
              CALL  PSTATS
              STOP
  200     CALL  ORDRA5 ( S )
          IF ( IERR .EQ. 0 )   GO TO 300
              CALL  SAVE ( 3, S )
              CALL  PSTATS
              STOP
  300     ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          FOUR = 4.0E0
          DO  400  I = 1, 200
              IF ( I .GT. 1 )   CALL  INAIJ5 ( I, I-1, -ONE, S )
              CALL  INBI ( I, TWO, S )
  400     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 200, ONE, S )
          CALL  SOLVE5 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  500  I = 1, 200
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  500     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********       RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980

        OUTPUT UNIT FOR ERROR MESSAGE        6
        OUTPUT UNIT FOR STATISTICS           6

    IJBEGN- BEGIN STRUCTURE INPUT...

    INIJ-   INPUT OF ADJACENCY PAIRS

    IJEND-  END OF STRUCTURE INPUT

    IJEND -  ERROR NUMBER  16
    TOO LITTLE STORAGE,
    MAXS MUST AT LEAST BE            999

    PSTATS- STATISTICS
               NUMBER OF EQUATIONS                    200
               OFF-DIAGONAL NONZEROS                  398
               SIZE OF WORKING STORE (MAXS)           900
```

```
C          M A I N L I N E     P R O G R A M
C
      INTEGER       I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL          S(999), ERROR, FOUR, ONE, TWO, ZERO
      REAL          RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 999
          CALL  IJBEGN
          DO  100  I = 2, 200
              CALL  INIJ ( I, I-1, S )
  100     CONTINUE
          CALL  IJEND ( S )
          IF ( IERR .EQ. 0 )    GO TO 200
              CALL  PSTATS
              STOP
  200     CALL  ORDRA5 ( S )
          IF ( IERR .EQ. 0 )    GO TO 300
              CALL  SAVE ( 3, S )
              CALL  PSTATS
              STOP
  300     ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          FOUR = 4.0E0
          DO  400  I = 1, 200
              IF ( I .GT. 1 )   CALL  INAIJ5 ( I, I-1, -ONE, S )
              CALL  INBI ( I, TWO, S )
  400     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 200, ONE, S )
          CALL  SOLVE5 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  500  I = 1, 200
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  500     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********       RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980

     OUTPUT UNIT FOR ERROR MESSAGE       6
     OUTPUT UNIT FOR STATISTICS          6

  IJBEGN- BEGIN STRUCTURE INPUT...

  INIJ-   INPUT OF ADJACENCY PAIRS

  IJEND-  END OF STRUCTURE INPUT

  ORDRA5- NESTED DISSECTION ORDERING

  ORDRXI  (X=A, B.  I=1,2,3,4,5,6)
     -    ERROR NUMBER  23

  INSUFFICIENT STORAGE FOR ORDERING.
  MAXS MUST BE AT LEAST           1400

  SAVE-   STORAGE VECTOR SAVED

  PSTATS- STATISTICS
             NUMBER OF EQUATIONS                     200
             OFF-DIAGONAL NONZEROS                   398
             SIZE OF WORKING STORE (MAXS)            999
```

```
C          M A I N L I N E    P R O G R A M
C
      INTEGER       I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL          S(2500), ERRCR, FOUR, ONE, TWO, ZERO
      REAL          RATIOL, RATICS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATICL, TIME
C
          CALL  SPRSPK
          MAXS = 2500
          CALL  RESTRT ( 3, S )
          CALL  ORDRA5 ( S )
          IF ( IERR .EQ. 0 )   GC TO 100
              CALL  SAVE ( 3, S )
              CALL  PSTATS
              STOP
  100     ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          FOUR = 4.0E0
          DO  200  I = 1, 200
              IF ( I .GT. 1 )  CALL  INAIJ5 ( I, I-1, -CNE, S )
              CALL  INAIJ5 ( I, I, FOUR, S )
              CALL  INBI ( I, TWO, S )
  200     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 200, ONE, S )
          CALL  SOLVE5 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  300  I = 1, 200
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  300     CONTINUE
          WRITE (IPRNTS, 11)  ERRCR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********       RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980


      OUTPUT UNIT FOR ERROR MESSAGE        6
      OUTPUT UNIT FOR STATISTICS           6

   RESTRT- RESTART SYSTEM

   ORDRA5- NESTED DISSECTION ORDERING

   ORDRXI  (X=A, B.  I=1,2,3,4,5,6)
        -  ERROR NUMBER  26

   INSUFFICIENT STORAGE
   FOR SOLVEI (I=1,2,3,4,5,6)
   MAXS MUST BE AT LEAST           2509

   SAVE-   STORAGE VECTOR SAVED

   PSTATS- STATISTICS
              NUMBER OF EQUATIONS                    200
              OFF-DIAGONAL NONZEROS                  398
              SIZE OF WORKING STORE (MAXS)          2500
              TIME FOR ORDERING                        0.083
              STORAGE FOR ORDERING                 1400.
              TIME FOR ALLOCATION                      0.030
              STORAGE FOR ALLOCATION               2324.
              STORAGE FOR SOLUTION                 2509.
```

```
C           M A I N L I N E     P R O G R A M
C
      INTEGER      I, IERR, IPRNTE, IPRNTS, MAXS, MSGLVL, NEQNS
      REAL         S(2509), ERROR, FOUR, ONE, TWO, ZERO
      REAL         RATIOL, RATIOS, TIME
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
      COMMON  /SPKSYS/  IPRNTE, IPRNTS, RATIOS, RATIOL, TIME
C
          CALL  SPRSPK
          MAXS = 2509
          CALL  RESTRT ( 3, S )
          CALL  ORDRA5 ( S )
          IF ( IERR .EQ. 0 )   GO TO 100
              CALL  SAVE ( 3, S )
              CALL  PSTATS
              STOP
  100     ZERO = 0.0E0
          ONE  = 1.0E0
          TWO  = 2.0E0
          FOUR = 4.0E0
          DO  200  I = 1, 200
              IF ( I .GT. 1 )  CALL  INAIJ5 ( I, I-1, -ONE, S )
              CALL  INAIJ5 ( I, I, FOUR, S )
              CALL  INBI  ( I, TWO, S )
  200     CONTINUE
          CALL  INBI ( 1, ONE, S )
          CALL  INBI ( 200, ONE, S )
          CALL  SOLVE5 ( S )
          CALL  PSTATS
          ERROR = ZERO
          DO  300  I = 1, 200
              ERROR = AMAX1 ( ERROR, ABS ( S(I)-ONE ) )
  300     CONTINUE
          WRITE (IPRNTS, 11)  ERROR
   11     FORMAT ( / 15H MAXIMUM ERROR  , 1PE15.3 )
          STOP
      END
```

```
*********** UNIVERSITY OF WATERLOO
*********** SPARSE MATRIX PACKAGE
***********  ( S P A R S P A K )
***********       RELEASE  2
***********    (C) JANUARY 1979
*********** STANDARD VERSION
*********** SINGLE PRECISION
*********** LAST UPDATE JANUARY 1980

         OUTPUT UNIT FOR ERROR MESSAGE          6
         OUTPUT UNIT FOR STATISTICS             6

     RESTRT- RESTART SYSTEM

     ORDRA5- NESTED DISSECTION ORDERING

     INAIJ5- INPUT OF MATRIX COMPONENTS

     INBI-   INPUT OF RIGHT HAND SIDE

     SOLVE5- GENERAL SPARSE SOLVE

     PSTATS- STATISTICS
                NUMBER OF EQUATIONS                    200
                OFF-DIAGONAL NONZEROS                  398
                SIZE OF WORKING STORE (MAXS)          2509
                TIME FOR ORDERING                        0.083
                STORAGE FOR ORDERING                 1400.
                TIME FOR ALLOCATION                      0.030
                STORAGE FOR ALLOCATION               2324.
                STORAGE FOR SOLUTION                 2509.
                TIME FOR FACTORIZATION                   0.043
                TIME FOR SOLVING                         0.017
                OPERATIONS IN FACTORIZATION           953.
                OPERATIONS IN SOLVING                1168.
MAXIMUM ERROR        1.550E-06
```

Example 6

This is a program to illustrate how one might use SPARSPAK to choose a method. The matrix is 300 by 300, it has nonzeroes on the diagonal, the first column and the last row. The structure of the matrix is input using IJBEGN, INIJ and IJEND, and then saved on FORTRAN unit 3. The modules ORDRA1, ORDRA3 and ORDRA5 are then executed, each one followed by a call to PSTATS to obtain the storage information. Note that RESTRT is called after execution of ORDRA1 and ORDRA3, to restore the package to the state that existed immediately after the structure inputting routines were executed. Note also that SAVE could have been used after each ordering module (with different output unit numbers). After one of the methods was chosen, RESTRT (with the appropriate unit number) could be used to initiate the computation, avoiding re-executing the ordering module corresponding to the method chosen.

```
C          M A I N L I N E     P R O G R A M
C
      INTEGER        I, IERR, MAXS, MSGLVL, NEQNS
      REAL           S(7500)
      COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
C
          CALL  SPRSPK
          MAXS = 7500
          CALL  IJBEGN
          DO  100  I = 1, 300
              CALL  INIJ ( I, 1, S )
              CALL  INIJ ( 300, I, S )
  100     CONTINUE
          CALL  IJEND ( S )
          CALL  SAVE ( 3, S )
          CALL  ORDRA1 ( S )
          CALL  PSTATS
          CALL  RESTRT ( 3, S )
          CALL  ORDRA3 ( S )
          CALL  PSTATS
          CALL  RESTRT ( 3, S )
          CALL  ORDRA5 ( S )
          CALL  PSTATS
          STOP
      END
```

```
********** UNIVERSITY OF WATERLOO
********** SPARSE MATRIX PACKAGE
**********  ( S P A R S P A K )
**********       RELEASE  2
**********     (C) JANUARY 1979
********** STANDARD VERSION
********** SINGLE PRECISION
********** LAST UPDATE JANUARY 1980

      OUTPUT UNIT FOR ERROR MESSAGE          6
      OUTPUT UNIT FOR STATISTICS             6

   IJBEGN- BEGIN STRUCTURE INPUT...

   INIJ-   INPUT OF ADJACENCY PAIRS

   IJEND-  END OF STRUCTURE INPUT

   SAVE-   STORAGE VECTOR SAVED

   ORDRA1- RCM ORDERING

   PSTATS- STATISTICS
              NUMBER OF EQUATIONS                      300
              OFF-DIAGONAL NONZEROS                   1194
              SIZE OF WORKING STORE (MAXS)            7500
              TIME FOR ORDERING                          0.087
              STORAGE FOR ORDERING                    2396.
              TIME FOR ALLOCATION                        0.013
              STORAGE FOR ALLOCATION                  2396.
              STORAGE FOR SOLUTION                    2098.

   RESTRT- RESTART SYSTEM

   ORDRA3- ONE WAY DISSECTION ORDERING

   PSTATS- STATISTICS
              NUMBER OF EQUATIONS                      300
              OFF-DIAGONAL NONZEROS                   1194
              SIZE OF WORKING STORE (MAXS)            7500
              TIME FOR ORDERING                          0.137
              STORAGE FOR ORDERING                    3297.
              TIME FOR ALLOCATION                        0.060
              STORAGE FOR ALLOCATION                  3300.
              STORAGE FOR SOLUTION                    3002.

   RESTRT- RESTART SYSTEM

   ORDRA5- NESTED DISSECTION ORDERING

   PSTATS- STATISTICS
              NUMBER OF EQUATIONS                      300
              OFF-DIAGONAL NONZEROS                   1194
```

```
SIZE OF WORKING STORE (MAXS)         7500
TIME FOR ORDERING                       0.123
STORAGE FOR ORDERING                2696.
TIME FOR ALLOCATION                     0.043
STORAGE FOR ALLOCATION              3599.
STORAGE FOR SOLUTION                3301.
```

Appendix A

IMPLEMENTATION OVERVIEW

In this section, we describe briefly the use of labelled common blocks in the internal implementation of SPARSPAK and the various methods of communication between modules.

A.1 USER/MODULE COMMUNICATION

As described in previous sections of this user guide, the user supplies a one-dimensional floating point array S, from which all array storage is allocated. In particular, the interface allocates the first NEQNS storage locations in S for the solution vector of the linear system of equations. After all the interface modules for a particular method have been successfully executed, the user can retrieve the solution from these NEQNS locations.

There is one labelled common block that the user must provide, having four variables:

```
COMMON  /SPKUSR/  MSGLVL, IERR, MAXS, NEQNS
```

The variable MAXS is the declared size of the one-dimensional floating point array S and it must be set by user at the beginning of his program. For each module in the interface that allocates storage (e.g. INIJ, IJEND, ORDRxi), MAXS is used to make sure that there is enough storage to carry out the particular phase.

A.2 MODULE/MODULE COMMUNICATION

There are several labelled common blocks used for communication among modules within the interface. Two important ones are the control block and the storage map block:

```
COMMON  /SPKCON/  STAGE, MXUSED, MXREQD, NEQNS,
                  NEDGES, METHOD, {and other
                  method-related control variables}

COMMON  /SPKMAP/  PERM, INVP, RHS,
```

{and other method-related data
structure pointers}

The control block has fourteen integer variables and contains control information about the specific problem being solved. There are fifteen variables in the storage map block, which keep the locations (origins in S) of the various arrays used in the particular storage scheme. These storage schemes differ in complexity across the methods, so the same storage map block must be used in the corresponding routines ORDRxi, INAIJi, INROWi, INMATi, and SOLVEi. An example is given below.

```
             r----┐
RHS     --->|    |
            |    |
            |    |     right hand side vector
            |    |
            |----|
PERM    --->|    |
            |    |
            |    |     permutation vector
            |    |
            |----|
INVP    --->|    |
            |    |
            |    |     inverse permutation vector
            |    |
            |----|
XENV    --->|    |
            |    |
            |    |     index to envelope structure of L
            |    |
            |----|
DIAG    --->|    |
            |    |
            |    |     diagonal of the matrix factor L
            |    |
            |----|
ENV     --->|    |
            |    |
            |    |     envelope of the matrix factor L
            |    |
            |
            |
```

Storage allocation for the symmetric envelope method
(CBIEA1)

A.3 SAVE/RESTART IMPLEMENATION

The SAVE routine saves the control information in the control block, the storage pointers in the storage map block, as well as the storage vector S. In this way, the state of the computation can be re-established by executing the module RESTRT, which restores the control block and the storage map block, and the storage vector S.

The variable MXUSED in the control block is used to avoid saving irrelevant data from S. After the successful completion of each phase, MXUSED is set to the maximum number of storage locations in S used thus far. It is then only necessary to save the first MXUSED locations of S whenever the routine SAVE is called.

Some operation systems allow a program to change the space it occupies in main storage during execution. Thus, in some installations the user of SPARSPAK may be able to dynamically increase or decrease the size of the working storage S. He can determine what the value of MAXS should be by declaring the labelled common block SPKCON in his mainline program, and examining the value of MXREQD. At the end of each successfully executed phase of the computation, MXREQD is set to the minimum value of MAXS required to successfully execute the next phase of the computation.

It is often the case that when this dynamic growing of program space is provided, the effect is to increase the space allocated to the unlabelled COMMON, which is usually assigned the highest memory loactions in the user's program area. In such a circumstance, the array S in the user's program would have to be declared in blank common.

A.4 METHOD CHECKING

As we discussed in the introduction, using a particular 'method' means calling the appropriate interface routines ORDRxi, INAIJi, INROWi, INMATi, and SOLVEi, where the last character is a numerical digit denoting the method. These ordering, input, and solve modules canot be mixed since they in general involve different data structures. In order to ensure that these modules are not inadvertently mixed by the user, ORDRxi sets the variable METHOD in the control block SPKCON equal to (10*i + k), where k is an integer that distinguishes orderings A and B. This variable is checked by subsequently executed input and solve modules.

A.5 STAGE(SEQUENCE) CHECKING

Another control variable that deserves comment is STAGE. As its name implies, it is used to keep track of the current step or stage of the execution. This variable is particularly important in connection with SAVE and RESTRT modules. In restarting the system using the RESTRT routine, the variable STAGE in the control block SPKCON is restored, and it indicates the last successfully completed stage or phase before the routine SAVE was called. In this way, the execution can be restarted without repeating already successfully completed steps.

Another function of this variable is to enforce the correct execution sequence of the various interface routines. Before the actual execution of each interface routines, the variable STAGE is used to check that all previous interface modules have been successfully completed. This avoids producing erroneous results due to improper processing sequence, or accidental omission of steps.

The content of the variable STAGE is only changed after a phase has been successfully executed. When an error occurs during the execution of the phase, the variable STAGE remains unchanged. This prevents the execution of all the subsequent phases, even if they are invoked by the user. The variable STAGE is also used by the modules to determine whether some initialization is necessary in a module, or whether part of the module has already successfully executed during a previous call to it.

A.6 STORAGE ALLOCATION OF INTEGER AND FLOATING POINT ARRAYS

The ANSI FORTRAN standard specifies that the number of bits used to represent integers and floating point numebrs are the same. However, some vendors provide the user with the option of specifying 'short' integers, either explicitly in the declarations such as 'INTEGER*2', or via a parameter to the FORTRAN processor which automatically represents all integers using fewer bits than used for floating point numbers. Since a significant portion of the storage used in sparse matrix computations involves integer data for pointers, subscripts etc., it is desirable to try to exploit these 'short' integer features whenever it makes sense to do so.

SPARSPAK contains parameters RATIOS and RATIOL, set in the module SPRSPK[*], which specify the ratios of the number of bits used for floating point numbers to the number used for 'short' and 'long' integers. For example, in a double precision IBM version of the package which exploits 'short' integers, RATIOS is 4 and RATIOL is 2. Let U(x) be the smallest integer m such that $m \geq x$. The package then uses RATIOS (RATIOL) to allocate only U(p/RATIOS) (U(p/RATIOL)) elements of S for 'short' ('long') integer arrays of length p.

SPARSPAK assumes that the declaration of S that the user makes in his program is of the same type as that used for floating point computation. We also make the reasonable assumption that RATIOS $\geq$ 1 and RATIOL $\geq$ 1.

A.7 <u>STATISTICS GATHERING</u>

SPARSPAK contains a labelled common block called SPKDTA which appears below. These variables are used to provide the output described in Section 7.2.

```
  COMMON  /SPKDTA/  ORDTIM, ALOCTM, FCTIME, SLVTIM, FCTOPS
                    SLVOPS, ORDSTR, ALOSTR, SLVSTR, OVERHD
```

In order to supply timing information, SPARSPAK assumes the existence of a real function DTIME which returns the processor execution time that has elapsed since DTIME was last referenced. Thus, the DTIME function is also installation dependent.

* Thus SPRSPK is an installation dependent subroutine.

BIBLIOGRAPHY

[1] Alan George, _An automatic one-way dissection algorithm for irregular finite element problems_, Proc. 1977 Dundee Conference on Numerical Analysis, Lecture Notes in Mathematics No. 630, Springer-Verlag, 1978, pp. 78-89.

[2] Alan George and J. W-H. Liu, _Algorithms for matrix partitioning and the numerical solution of finite element systems_, SIAM J. Numer. Anal., 15 (1978), pp. 297-327.

[3] Alan George and J. W-H. Liu, _An automatic nested dissection algorithm for irregular finite element problems_, SIAM J. Numer. Anal., 15 (1978), pp. 1053-1069.

[4] Alan George and J. W-H. Liu, _A fast implementation of the minimum degree algorithm using quotient graphs_, Research Report CS-78-12, July 1978. Department of Computer Science, University of Waterloo.

[5] Alan George and J. W-H. Liu, _The design of a user interface for a sparse matrix package_, ACM Trans. on Math. Software, 5 (1979), pp. 139-162.

[6] Alan George and J. W-H. Liu, _Computer solution of large sparse symmetric positive definite systems of linear equations_, to be published by Prentice Hall, Inc.

[7] Liu, J. W-H. and A. H. Sherman, _Comparative analysis of the Cuthill-McKee and Reverse Cuthill-McKee ordering algorithms for sparse matrices_, SIAM J. Numer. Anal., 13 (1976), pp. 198-213.