LANGLEY
GRANT
IN-61-CR
148375
135P.

FINAL REPORT
Algorithms and Software for Solving Finite Element Equations on Serial and Parallel Architectures

Grant NAG-803
NASA Langley Research Center
Hampton VA 23665

Eleanor Chu[*]
Department of Computer Science
University of Tennessee
Knoxville, TN 37996-1301

Alan George[†]
Departments of Computer Science and Mathematics
University of Tennessee
Knoxville, TN 37996-1301

June 22, 1988

[*]After July 1, 1988: Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

[†]After July 1, 1988: Office of the Provost, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. Telephone: (519) 885-1211 x2809

FINAL REPORT
Algorithms and Software for Solving Finite Element Equations on Serial and Parallel Architectures

Grant NAG-803
NASA Langley Research Center
Hampton VA 23665

Eleanor Chu[*]
Department of Computer Science
University of Tennessee
Knoxville, TN 37996-1301

Alan George[†]
Departments of Computer Science and Mathematics
University of Tennessee
Knoxville, TN 37996-1301

June 22, 1988

[*]After July 1, 1988: Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

[†]After July 1, 1988: Office of the Provost, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. Telephone: (519) 885-1211 x2809

Contents

1 Introduction

This report describes our research on sparse matrix techniques for the Computational Structural Mechanics (CSM) Testbed [2] conducted for NASA grant NAG-1-803. Before providing a synopsis of the report, we give a brief overview of the work that has been completed during the 10-month tenure of the grant.

A primary objective was to compare the performance of state-of-the-art techniques for solving sparse systems with those that are currently available in the CSM testbed. Thus, one of the first tasks was to become familiar with the structure of the testbed, and to install some or all of the SPARSPAK package [7,20,21] in the testbed.

We began by installing the CSM testbed on our SUN workstations. We were the first site to do this, and it was necessary to collaborate closely with the CSM group at Langley in order to resolve some minor problems with the installation procedure.

A suite of subroutines to extract from the database the relevant structural and numerical information about the matrix equations has been written. A driver program (processor) that employs these routines along with the SPARSPAK library has been written, and we have successfully solved *all* the demonstration problems distributed with the testbed. These codes have been documented, and performance studies comparing the SPARSPAK technology to the methods currently in the testbed have been completed. In addition, some preliminary studies have been done comparing some recently developed out-of-core techniques with the performance of the testbed processor INV.

An outline of the report is as follows. Section 2 contains a brief overview of the CSM Testbed software and its usage. This is essentially background material for the uninitiated, and can be ignored by those with experience in the usage of the testbed.

Since the ultimate goal of sparse matrix research for the Testbed is to enhance the performance and capabilities of the Testbed, some knowledge of the methods currently employed is essential in the development of better techniques for the Testbed. Section 3 gives an overview of the sparse matrix techniques currently employed in the CSM Testbed. Our presentation is focused on the internal working of the SPAR matrix processors [5].

Section 4 describes an interface which we have designed and implemented as a research tool for installing and appraising new matrix processors in the CSM Testbed, along with a description of a new processor SPK which consists of a subset of SPARSPAK-A [7] and a set of subroutines which provide an interface between SPARSPAK-A and the global database of the CSM testbed. A guide for installing the processor SPK in the testbed is provided in Appendix A of this report. The installation dependent modules of this processor are listed in Appendix B with comments indicating the changes to be done at a different site. A listing of all interface subroutines is provided in Appendix C.

Finally, §5 contains results of numerical experiments we performed in solving a set of testbed demonstration problems using the processor SPK and other experimental processors. These results are compared with the performance of the SPAR matrix processors on the same set of test problems.

2 The CSM Testbed Software System and Its Usage

To facilitate our discussion throughout this report, we shall first briefly introduce the concepts and terminology employed in the Testbed. Since our discussion is conducted primarily for the readers who have not used the Testbed before, the readers who are familiar with its usage can skip this section.

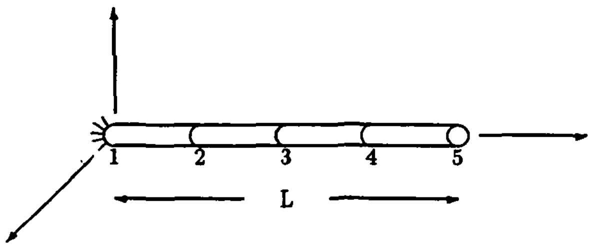
The CSM Testbed is a structural analysis system evolving from integrating the SPAR finite element code [5] and the NICE data management and command processing utilities [9,10,15]. The FORTRAN programs for SPAR (Structural Performance Analysis and Redesign) were developed in the 1970's by Lockheed Missiles and Space Company and by Engineering Information Systems, Incorporated. The SPAR system uses the finite element approach to perform stress, buckling, vibration, and thermal analysis on linear structural systems. The NICE (Network of Interactive Computational Elements) system was originally developed at Lockheed Palo Alto Research Laboratories to support engineering analyses. The major components of the NICE system include a data manager, a command language and a command interpreter. Continued effort has been made by the CSM development team at NASA Langley and at the Lockheed Palo Alto Research Laboratory to extend the analysis capability of the Testbed since the implementation of its initial version (called NICE/SPAR).

The user interface for the Testbed is described in detail in the CSM Testbed User's Guide [1]. The language, directives, interface, global-database manager and input-output manager of the CSM Testbed architecture are each documented in references [11,12,13,30,14]. For our purpose we shall simply walk through an example to quickly familiarize the readers with the general usage of the Testbed. The example we use is a Testbed demonstration problem presented in [4]. We shall refer to this example as problem "demol" throughout this report.

The operating environment Our discussion throughout this report refers to the version of the Testbed currently operational on a SUN 3/50 workstation running the UNIX[1] operating system at the University of Tennessee, Knoxville.

The problem to be solved: The tubular beam shown in Figure 1 is cantilevered at joint 1 and statically loaded at joint 5. The static solution for a transverse shear load of 1000.0 and for an axial load of 10000.0 is required.

[1]UNIX is a trademark of AT&T Bell Laboratories.



L=40

Tube, inner radius = 2.00, outer radius = 2.25

$E = 10. \times 10^6$

$\nu = 0.3$

$\rho = 0.101$

$\alpha = 0.1 \times 10^{-4}$

Figure 1: CSM Testbed Demonstration Problem - Tubular beam.

User input Edit a file to contain the script in Figure 2. The command stream demonstrates how to solve the tubular beam problem in Figure 1 using the NICE command language and the SPAR computational modules.

Comments The problem-oriented Testbed command language is called CLAMP - an acronym for Command Language for Applied Mechanics Processors. The commands with its leading keyword prefixed by an asterisk are called CLAMP directives. They are special commands used to

- directly access a global database,
- define command procedures,
- implement branching and cycling for nonsequential command processing,
- process macrosymbols in an advanced language construct,
- request other available services.

For example, the directive

```
*open 1 demo1.101 /new
```

contained in our script file will create a new library file with the library identification number (LDI) equal to "1" and file name of "demo1.l01".

The SPAR processors are each implemented as a subroutine callable by the Testbed executive module. The macroprocessor command to start the execution of a processor is [XQT. Therefore, during the execution of the Testbed, the command to run the SPAR processor named TAB is

```
[XQT TAB
```

The input (user commands and/or data) to a processor are entered after the [XQT command according to the requirements of the individual processor. The SPAR input syntax and processor requirements are described in detail in [1]. Since the CLAMP directives may be intermixed with the processor commands in the script file, it is worth noting that once the execution of a processor is initiated by [XQT, it will begin and continue accepting input until either another [XQT, a STOP or a *STOP is encountered. If a STOP occurs, execution will proceed to completion of the processor's assigned task after which the next command, which can be either a CLAMP directive or a macroprocessor command, begins execution. A *STOP terminates execution immediately. Therefore, the user command STOP in the sequence

```
[XQT SSOL
     STOP
*TOC 1
```

is necessary to ensure that processor SSOL runs to completion before the directive *TOC is processed.

The modular structure of the Testbed implies that multiple processors are typically executed to perform an analysis. These processors communicate through a common database consisting of global-access data libraries (GAL) which are operated on by the NICE data manager GAL-DBM [3]. Each GAL data library may contain multiple nominal datasets. Each dataset is made up of named records. The GAL-Processor interface facilities allow the Testbed processors to generate, store, locate, and access all of the needed information in the global database to perform a required analysis. The table of contents of an active data library may be displayed during execution of the Testbed via the CLAMP directive *TOC. In Figure 3, we display the table of contents for the data library "demo1.l01" (LDI=1) created by executing the script in Figure 2.

To execute the analysis: Note that on Unix systems the execution of the Testbed is initiated by the first command "**time nicespar << \eof**" in the script file, where "**nicespar**" is the name of the executable file and we assume that the name of the directory where "**nicespar**" resides has been inserted in the user's PATH environment variable. Note also that "\eof"is the last entry of the script. Assuming that the name of the file containing the script is "demo1.com" and that it has been made executable with the "chmod" command, the script may be run by typing

```
demo1.com
```

To print the solutions on an ordinary text file: The default output file for the Testbed is the standard output on Unix systems. The command

```
demo1.com > & demo1.log &
```

thus redirects the output to the log file. The desired static solutions are produced by processor SSOL and the actual data are contained in the dataset named STAT.DISP.1.1. To print the static solutions on the log file, the SPAR utility processor VPTR may be executed after [XQT SSOL. The command to be inserted into the script is

```
[XQT VPRT
     TRPINT STAT DISP 1 1
```

The output corresponding to this command is displayed in Figure 4. Note that each constrained component is flagged with an asterisk by the processor VPRT.

More details: We shall come back to this example from time to time to provide the details which are not needed until our discussion at a later point.

```
time nicespar << \eof                           . Start and time Testbed execution
*open 1 demo1.101 /new                          . Open data library
*set echo=off                                   . Do not echo input
 [xqt TAB                                       . Macroprocessor command to execute TAB
      START 5                                   . 5 nodes points in beam
      JOINT LOCATIONS                           . Direct TAB input
      1 0 0 0.
      2 0 0 10.
      3 0 0 20.
      4 0 0 30.
      5 0 0 40.
      MATERIAL CONSTANTS
      1 10.E+6 .3 .101 .1E-4
      BEAM ORIENTATIONS
      1 1 1 1 1.
      E21 SECTION PROPERTIES
      TUBE 1 2. 2.25
      CONSTRAINT DEFINITION 1                   . Constrain 6 components of joint 1
      ZERO 1 2 3 4 5 6                          . to be zero
      1
 [xqt ELD                                       . Define elements
      E21
      1 2                                       . Define element connectivity
      2 3
      3 4
      4 5
 [xqt E                                         . Create element datasets
 [xqt EKS                                       . Calculate element intrinsic
                                                . stiffness matrices
 [xqt RSEQ                                      . Resequence nodes
      reset METHOD=1 LJSPRT=1 LADPRT=1
 [xqt TOPO                                      . Form maps which guide the assembly
      reset PRTKMAP=1  PRTAMAP=1                . and factorization of system matrices
 [xqt K                                         . Assemble system stiffness matrix
      reset spdp=2                              . Output dataset in double precision
 [xqt INV                                       . Factor system stiffness matrix
      reset spdp=2                              . in double precision
 [xqt AUS
  ALPHA                                         . Direct AUS input
    CASE TITLES                                 . Define load titles for 2 cases
    1'TRANSVERSE LOAD
    2'AXIAL LOAD
    SYSVEC
    APPLIED FORCES
      CASE 1: I=2: J=5: 1000.                   . Dir-2 load on joint 5 of 1000.
      CASE 2: I=3: J=5: 10000.                  . Dir-3 load on joint 5 of 10000.
 [xqt SSOL                                      . Solve for static displacements
 [xqt GSF                                       . Compute stresses
 [xqt PSF                                       . Print stresses
      stop
*TOC 1                                          . Print Table of contents of library 1
\eof
```

Figure 2: A runstream for solving problem demo1.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Library  1    File: demo1.101                                              +
+ Form: GAL82   File size:     22062 words      No. of Datasets:   32        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Seq#    Date     Time     Lk Records  Processor Dataset name
  1  05:14:88 17:54:17   0        1  TAB        JDF1.BTAB.1.8
  2  05:14:88 17:54:17   0        1  TAB        JREF.BTAB.2.6
  3* 05:14:88 17:54:17   0        1  TAB        ALTR.BTAB.2.4
  4  05:14:88 17:55:17   0        3  TAB        GNTR.BTAB.6.6
  5  05:14:88 17:55:17   0        1  TAB        ALTR.BTAB.2.4
  6  05:14:88 17:55:17   0        1  TAB        JLOC.BTAB.2.5
  7  05:14:88 17:55:17   0        1  TAB        MATC.BTAB.2.2
  8  05:14:88 17:55:17   0        1  TAB        MREF.BTAB.2.7
  9  05:14:88 17:55:17   0        1  TAB        BA.BTAB.2.9
 10  05:14:88 17:56:17   0        1  TAB        CON..1
 11  05:14:88 17:56:17   0        1  TAB        QJJT.BTAB.2.19
 12  05:14:88 17:56:17   0        1  ELD        DEF.E21.1.2
 13  05:14:88 17:56:17   0        1  ELD        GD.E21.1.2
 14  05:14:88 17:56:17   0        1  ELD        GTIT.E21.1.2
 15  05:14:88 17:56:17   0        1  ELD        DIR.E21.1.2
 16  05:14:88 17:56:17   0        1  ELD        ELTS.NAME
 17  05:14:88 17:56:17   0        1  ELD        ELTS.NNOD
 18  05:14:88 17:56:17   0        1  ELD        ELTS.ISCT
 19  05:14:88 17:56:17   0        1  ELD        NS
 20  05:14:88 17:56:17   0        4  E          E21.EFIL.1.2
 21  05:14:88 17:56:17   0        1  E          DEM.DIAG
 22  05:14:88 17:56:17   0        1  RSEQ       JSEQ.BTAB.2.17
 23  05:14:88 17:56:17   0        1  TOPO       KMAP..9.3
 24  05:14:88 17:56:17   0        1  TOPO       AMAP..9.3
 25  05:14:88 17:56:17   0        1  K          K.SPAR.36
 26  05:14:88 17:56:17   0        5  INV        INV.K.1
 27  05:14:88 17:56:17   0        2  AUS        CASE.TITL.1.1
 28  05:14:88 17:56:17   0        2  AUS        APPL.FORC.1.1
 29  05:14:88 17:56:17   0        2  SSOL       STAT.DISP.1.1
 30  05:14:88 17:56:17   0        2  SSOL       STAT.REAC.1.1
 31  05:14:88 17:56:17   0        4  GSF        STRS.E21.1.1
 32  05:14:88 17:56:17   0        4  GSF        STRS.E21.1.2
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 3: Table of Contents of Library 1.

```
 ** BEGIN VPRT **    DATA SPACE=    600000 WORDS
1STATIC DISPLACEMENTS.                                          ID=   1/ 1/  1
 TRANSVERSE LOAD
0JOINT       1            2           3           4           5           6
     1  0.000e+00*  0.000e+00*  0.000e+00*  0.000e+00*  0.000e+00*  0.000e+00*
     2  0.000e+00   0.250e-01   0.000e+00  -0.463e-02   0.000e+00   0.000e+00
     3  0.000e+00   0.897e-01   0.000e+00  -0.793e-02   0.000e+00   0.000e+00
     4  0.000e+00   0.181e+00   0.000e+00  -0.992e-02   0.000e+00   0.000e+00
     5  0.000e+00   0.285e+00   0.000e+00  -0.106e-01   0.000e+00   0.000e+00
1STATIC DISPLACEMENTS.                                          ID=   1/ 1/  2
 AXIAL LOAD
0JOINT       1            2           3           4           5           6
     1  0.000e+00*  0.000e+00*  0.000e+00*  0.000e+00*  0.000e+00*  0.000e+00*
     2  0.000e+00   0.000e+00   0.300e-02   0.000e+00   0.000e+00   0.000e+00
     3  0.000e+00   0.000e+00   0.599e-02   0.000e+00   0.000e+00   0.000e+00
     4  0.000e+00   0.000e+00   0.899e-02   0.000e+00   0.000e+00   0.000e+00
     5  0.000e+00   0.000e+00   0.120e-01   0.000e+00   0.000e+00   0.000e+00
 EXIT VPRT CPUTIME=     0.5 I/O(DIR,BUF)=      0     0
```

Figure 4: The contents of dataset STAT.DISP.1.1.

3 The CSM Testbed Matrix Processors

Reference [29] contains a set of logic flowcharts developed for the key subroutines of each of the SPAR matrix processors `TOPO`, `K`, `INV`, `SSOL` and `AUS`. These charts together with the commented FORTRAN source code are very helpful in our understanding of the sparse matrix techniques currently employed in the Testbed. In this section, we shall attempt to describe the algorithms and data structures which are implemented by the processors `INV` and `SSOL`.

3.1 The Basic Algorithms

The factorization algorithm: Processor `INV` applies a specialized Gaussian elimination scheme to factor a sparse symmetric matrix K into LDL^T, where L is a unit lower triangular matrix and D is a diagonal matrix. This algorithm is numerically stable if the matrix K is also positive definite, which is the case when K is the system stiffness matrix. The basic algorithm can be easily described for a dense symmetric matrix A as follows. We assume that A is of dimension $n \times n$. Let us denote the elements of A and $M = L^T$ as a_{ij} and m_{ij}, where $1 \leq i \leq n$ and $i \leq j \leq n$, and $D = \{d_1, d_2, \dots, d_n\}$. Note that each off-diagonal a_{ij} is overwritten by m_{ij} and that each a_{ii} is overwritten by d_i^{-1} if the algorithm presented in Figure 5 is successfully executed. Algorithm I assumes that the $a_{i,j}$ elements are stored row by row.

Algorithm I The basic LDL^T factorization scheme

```
for $i \leftarrow 1, 2, \dots, n$ do
    if $a_{ii} = 0$ then
        quit
    else
        $a_{ii} \leftarrow 1/a_{ii}$
        for $k \leftarrow i + 1, \dots, n$ do
            $m \leftarrow a_{ik} * a_{ii}$
            for $j \leftarrow k, \dots, n$ do
                $a_{kj} \leftarrow a_{kj} - m * a_{ij}$
        for $k \leftarrow i + 1, \dots, n$ do
            $a_{ik} \leftarrow a_{ik} * a_{i,i}$
```

Figure 5: Computing D^{-1} and $M = L^T$ factors of A.

The following features of the algorithm above will be exploited in its sparse implementation.

1. To compute D^{-1} and the off-diagonal elements of $M = L^T$, the elements stored and accessed are those on the diagonal and in the upper triangular part of A. For example, when $n = 5$, the algorithm performs the transformation in Figure 6.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ & & a_{3,3} & a_{3,4} & a_{3,5} \\ & & & a_{4,4} & a_{4,5} \\ & & & & a_{5,5} \end{bmatrix} \longrightarrow \begin{bmatrix} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} \\ & d_2^{-1} & m_{2,3} & m_{2,4} & m_{2,5} \\ & & d_3^{-1} & m_{3,4} & m_{3,5} \\ & & & d_4^{-1} & m_{4,5} \\ & & & & d_5^{-1} \end{bmatrix}$$

Figure 6: Overwriting A by D^{-1} and $M = L^T$.

2. The a_{ij}'s which have been overwritten by the elements of D^{-1} and $M = L^T$ will not be needed in the remaining elimination stages. In particular, during the i^{th} elimination stage, the elements accessed and modified are confined to row i to row n as shown in Figure 7 for $i = 3$ and $n = 5$, where $\otimes$ represents elements which are not accessed.

$$\begin{bmatrix} \otimes & \otimes & \otimes & \otimes & \otimes \\ & \otimes & \otimes & \otimes & \otimes \\ & & a_{3,3} & a_{3,4} & a_{3,5} \\ & & & a_{4,4} & a_{4,5} \\ & & & & a_{5,5} \end{bmatrix} \longrightarrow \begin{bmatrix} \otimes & \otimes & \otimes & \otimes & \otimes \\ & \otimes & \otimes & \otimes & \otimes \\ & & d_3^{-1} & m_{3,4} & m_{3,5} \\ & & & \tilde{a}_{4,4} & \tilde{a}_{4,5} \\ & & & & \tilde{a}_{5,5} \end{bmatrix}$$

Figure 7: LDL^T factorization of A - the third stage.

Solving the triangular systems: Since Algorithm I stores the factors D^{-1} and $M = L^T$, we shall describe the solution scheme in terms of these two factors. Both of the forward and backward substitution schemes presented below access the elements of the factor M row by row.

Step 1. Forward substitution scheme (Solve $M^T y = b$).

```
for $i \leftarrow 1, \ldots, n$ do
    $y_i \leftarrow b_i$
    for $k = i + 1, \ldots, n$ do
        $b_k = b_k - m_{ik} * y_i$
```

Step 2. Backward substitution scheme (Solve $Mx = D^{-1}y$).

```
for $\rho \leftarrow 1, \ldots, n$ do
    $i \leftarrow n - \rho + 1$
    $s \leftarrow 0$
    for $j = i + 1, \ldots n$ do
        $s \leftarrow s + m_{ij} * x_j$
    $x_i \leftarrow d_i^{-1} * y_i - s$
```

3.2 The INV Implementation

In this section we shall discuss in various degrees of details the following aspects of the sparse factorization scheme implemented by the processor INV.

1. The algorithm - a block LDL^T factorization scheme.

2. Memory requirement.

3. Data structures.

4. The handling of zero constraints.

5. The handling of nonzero constraints.

6. Data archived to the global database.

A block LDL^T factorization scheme: The processor INV has tailored Algorithm I to perform an out-of-core block LDL^T factorization of large sparse matrices arising in the finite element analysis of structural mechanics problems. Before we describe the INV implementation of this scheme, let us first explain the block LDL^T algorithm by applying it to a dense symmetric matrix $\mathcal{A}$ in block form. To be specific, let us consider the 2×2 block matrix in Figure 8, where $A_{1,1} = \{a_{kj}^{(i)}\}$, $A_{1,2} = \{a_{kj}^{(ii)}\}$, $A_{2,2} = \{a_{kj}^{(iv)}\}$, with $a_{kj}^{(i)} = a_{jk}^{(i)}$, $a_{kj}^{(iv)} = a_{jk}^{(iv)}$, and $1 \le k, j \le 3$.

$$\mathcal{A} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{1,2}^T & A_{2,2} \end{pmatrix} = \left(\begin{array}{ccc|ccc} a_{1,1}^{(i)} & a_{1,2}^{(i)} & a_{1,3}^{(i)} & a_{1,1}^{(ii)} & a_{1,2}^{(ii)} & a_{1,3}^{(ii)} \\ a_{2,1}^{(i)} & a_{2,2}^{(i)} & a_{2,3}^{(i)} & a_{2,1}^{(ii)} & a_{2,2}^{(ii)} & a_{2,3}^{(ii)} \\ a_{3,1}^{(i)} & a_{3,2}^{(i)} & a_{3,3}^{(i)} & a_{3,1}^{(ii)} & a_{3,2}^{(ii)} & a_{3,3}^{(ii)} \\ \hline a_{1,1}^{(ii)} & a_{2,1}^{(ii)} & a_{3,1}^{(ii)} & a_{1,1}^{(iv)} & a_{1,2}^{(iv)} & a_{1,3}^{(iv)} \\ a_{1,2}^{(ii)} & a_{2,2}^{(ii)} & a_{3,2}^{(ii)} & a_{2,1}^{(iv)} & a_{2,2}^{(iv)} & a_{2,3}^{(iv)} \\ a_{1,3}^{(i)} & a_{2,3}^{(i)} & a_{3,3}^{(i)} & a_{3,1}^{(iv)} & a_{3,2}^{(iv)} & a_{3,3}^{(iv)} \end{array}\right)$$

Figure 8: Partitioning symmetric $\mathcal{A}$ into four 3×3 blocks.

The block LDL^T scheme works in the following manner.

Step 1. Apply Algorithm I to matrix $A_{1,1}$ to perform the following transformation.

$$\begin{pmatrix} a_{1,1}^{(i)} & a_{1,2}^{(i)} & a_{1,3}^{(i)} \\ & a_{2,2}^{(i)} & a_{2,3}^{(i)} \\ & & a_{3,3}^{(i)} \end{pmatrix} \longrightarrow \begin{pmatrix} 1/d_1^{(i)} & m_{1,2}^{(i)} & m_{1,3}^{(i)} \\ & 1/d_2^{(i)} & m_{2,3}^{(i)} \\ & & 1/d_3^{(i)} \end{pmatrix}$$

In other words, at the end of step 1, we have in fact zeroed out the nonzeros in the lower triangular part of $A_{1,1}$ and stored the multipliers $\ell_{ij} = m_{ji}$ in the upper triangular part of $A_{1,1}$.

Step 2. Apply the multiplers $m_{kj}^{(i)}$ to the $A_{1,2}$ block as if LU ($U = DL^T$) decomposition were applied to reduce $(A_{1,1}, A_{1,2})$ to an upper trapezoidal matrix. That is, the $A_{1,2}$ block is overwritten by the resulting $\{u_{kj}^{(ii)}\}$ of the following transformation.

$$\begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & -m_{2,3}^{(i)} & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ -m_{1,2}^{(i)} & 1 & \\ -m_{1,3}^{(i)} & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{1,1}^{(ii)} & a_{1,2}^{(ii)} & a_{1,3}^{(ii)} \\ a_{2,1}^{(ii)} & a_{2,2}^{(ii)} & a_{2,3}^{(ii)} \\ a_{3,1}^{(ii)} & a_{3,2}^{(ii)} & a_{3,3}^{(ii)} \end{pmatrix} = \begin{pmatrix} u_{1,1}^{(ii)} & u_{1,2}^{(ii)} & u_{1,3}^{(ii)} \\ u_{2,1}^{(ii)} & u_{2,2}^{(ii)} & u_{2,3}^{(ii)} \\ u_{3,1}^{(ii)} & u_{3,2}^{(ii)} & u_{3,3}^{(ii)} \end{pmatrix}$$

Step 3. Zero out the block $A_{1,2}^T$ implicitly by applying the multipliers directory to block $A_{2,2}$. The multipliers can be computed on the fly from

$$\begin{pmatrix} m_{1,1}^{(ii)} & m_{2,1}^{(ii)} & m_{3,1}^{(ii)} \\ m_{1,2}^{(ii)} & m_{2,2}^{(ii)} & m_{3,2}^{(ii)} \\ m_{1,3}^{(ii)} & m_{2,3}^{(ii)} & m_{3,3}^{(ii)} \end{pmatrix} = \begin{pmatrix} u_{1,1}^{(ii)} & u_{2,1}^{(ii)} & u_{3,1}^{(ii)} \\ u_{1,2}^{(ii)} & u_{2,2}^{(ii)} & u_{3,2}^{(ii)} \\ u_{1,3}^{(ii)} & u_{2,3}^{(ii)} & u_{3,3}^{(ii)} \end{pmatrix} \begin{pmatrix} 1/d_1^{(i)} & & \\ & 1/d_2^{(i)} & \\ & & 1/d_3^{(i)} \end{pmatrix}$$

The $A_{2,2}$ block is then updated to be $\tilde{A}_{2,2} = \{\tilde{a}_{kj}^{(iv)}\}$ which is obtained by the following computation.

$$\begin{pmatrix} 1 & & & & & \\ 0 & 1 & & & & \\ 0 & 0 & 1 & & & \\ -m_{1,1}^{(ii)} & -m_{2,1}^{(ii)} & -m_{3,1}^{(ii)} & 1 & & \\ -m_{1,2}^{(ii)} & -m_{2,2}^{(ii)} & -m_{3,2}^{(ii)} & 0 & 1 & \\ -m_{1,3}^{(ii)} & -m_{2,3}^{(ii)} & -m_{3,3}^{(ii)} & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_{1,1}^{(ii)} & u_{1,2}^{(ii)} & u_{1,3}^{(ii)} \\ u_{2,1}^{(ii)} & u_{2,2}^{(ii)} & u_{2,3}^{(ii)} \\ u_{3,1}^{(ii)} & u_{3,2}^{(ii)} & u_{3,3}^{(ii)} \\ a_{1,1}^{(iv)} & a_{1,2}^{(iv)} & a_{1,3}^{(iv)} \\ a_{2,1}^{(iv)} & a_{2,2}^{(iv)} & a_{2,3}^{(iv)} \\ a_{3,1}^{(iv)} & a_{3,2}^{(iv)} & a_{3,3}^{(iv)} \end{pmatrix} = \begin{pmatrix} u_{1,1}^{(ii)} & u_{1,2}^{(ii)} & u_{1,3}^{(ii)} \\ u_{2,1}^{(ii)} & u_{2,2}^{(ii)} & u_{2,3}^{(ii)} \\ u_{3,1}^{(ii)} & u_{3,2}^{(ii)} & u_{3,3}^{(ii)} \\ \tilde{a}_{1,1}^{(iv)} & \tilde{a}_{1,2}^{(iv)} & \tilde{a}_{1,3}^{(iv)} \\ \tilde{a}_{2,1}^{(iv)} & \tilde{a}_{2,2}^{(iv)} & \tilde{a}_{2,3}^{(iv)} \\ \tilde{a}_{3,1}^{(iv)} & \tilde{a}_{3,2}^{(iv)} & \tilde{a}_{3,3}^{(iv)} \end{pmatrix}$$

Since the $A_{2,2}$ diagonal block is symmetric, only the upper triangular part of $A_{2,2}$ is updated in the actual computation.

Step 4. $u_{k,j}^{(ii)} \leftarrow u_{k,j}^{(ii)}/d_k^{(i)}$, for $\forall k, j$.

Note that the transformations accomplished by the above four steps can be expressed with respect to the block upper triangular part of the given matrix as follows.

$$\left(\begin{array}{ccc|ccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ \hline & & & a_{4,4} & a_{4,5} & a_{4,6} \\ & & & a_{5,4} & a_{5,5} & a_{5,6} \\ & & & a_{6,4} & a_{6,5} & a_{6,6} \end{array}\right) \longrightarrow \left(\begin{array}{ccc|ccc} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ a_{2,1} & d_2^{-1} & m_{2,3} & m_{2,4} & m_{2,5} & m_{2,6} \\ a_{3,1} & a_{3,2} & d_3^{-1} & m_{3,4} & m_{3,5} & m_{3,6} \\ \hline & & & \tilde{a}_{4,4} & \tilde{a}_{4,5} & \tilde{a}_{4,6} \\ & & & a_{5,4} & \tilde{a}_{5,5} & \tilde{a}_{5,6} \\ & & & a_{6,4} & a_{6,5} & \tilde{a}_{6,6} \end{array}\right).$$

The final step: For this particular example, the factorization is completed after transforming

$$\begin{pmatrix} \tilde{a}_{4,4} & \tilde{a}_{4,5} & \tilde{a}_{4,6} \\ & \tilde{a}_{5,5} & \tilde{a}_{5,6} \\ & & \tilde{a}_{6,6} \end{pmatrix} \longrightarrow \begin{pmatrix} d_4^{-1} & m_{4,5} & m_{4,6} \\ & d_5^{-1} & m_{5,6} \\ & & d_6^{-1} \end{pmatrix}$$

by Algorithm I.

The output matrix: The coefficient matrices of the resulting triangular systems, namely $M^T y = b$ and $Mx = D^{-1}y$, are available from the output matrix given by

$$\begin{pmatrix} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ & d_2^{-1} & m_{2,3} & m_{2,4} & m_{2,5} & m_{2,6} \\ & & d_3^{-1} & m_{3,4} & m_{3,5} & m_{3,6} \\ & & & d_4^{-1} & m_{4,5} & m_{4,6} \\ & & & & d_5^{-1} & m_{5,6} \\ & & & & & d_6^{-1} \end{pmatrix}$$

The following observations may be made on the block LDL^T factorization scheme described above.

1. The elements in the lower triangular part of the diagonal $A_{i,i}$ blocks are not accessed during the process of computing the D^{-1} and $L^T = M$ factors.
2. The block of rows which have been updated to contain the d_k^{-1}'s and m_{kj}'s of the factors are no longer needed in the future stages of elimination.
3. Observe that the updating of $A_{2,2}$ block in Step 3 can be reformulated as follows.

$$\begin{pmatrix} s_{1,1}^{(iv)} & s_{1,2}^{(iv)} & s_{1,3}^{(iv)} \\ s_{2,1}^{(iv)} & s_{2,2}^{(iv)} & s_{2,3}^{(iv)} \\ s_{3,1}^{(iv)} & s_{3,2}^{(iv)} & s_{3,3}^{(iv)} \end{pmatrix} = \begin{pmatrix} -m_{1,1}^{(ii)} & -m_{2,1}^{(ii)} & -m_{3,1}^{(ii)} \\ -m_{1,2}^{(ii)} & -m_{2,2}^{(ii)} & -m_{3,2}^{(ii)} \\ -m_{1,3}^{(ii)} & -m_{2,3}^{(ii)} & -m_{3,3}^{(ii)} \end{pmatrix} \begin{pmatrix} u_{1,1}^{(ii)} & u_{1,2}^{(ii)} & u_{1,3}^{(ii)} \\ u_{2,1}^{(ii)} & u_{2,2}^{(ii)} & u_{2,3}^{(ii)} \\ u_{3,1}^{(ii)} & u_{3,2}^{(ii)} & u_{3,3}^{(ii)} \end{pmatrix}$$

and

$$\begin{pmatrix} \tilde{a}^{(iv)}_{1,1} & \tilde{a}^{(iv)}_{1,2} & \tilde{a}^{(iv)}_{1,3} \\ & \tilde{a}^{(iv)}_{2,2} & \tilde{a}^{(iv)}_{2,3} \\ & & \tilde{a}^{(iv)}_{3,3} \end{pmatrix} = \begin{pmatrix} a^{(iv)}_{1,1} & a^{(iv)}_{1,2} & a^{(iv)}_{1,3} \\ & a^{(iv)}_{2,2} & a^{(iv)}_{2,3} \\ & & a^{(iv)}_{3,3} \end{pmatrix} + \begin{pmatrix} s^{(iv)}_{1,1} & s^{(iv)}_{1,2} & s^{(iv)}_{1,3} \\ & s^{(iv)}_{2,2} & s^{(iv)}_{2,3} \\ & & s^{(iv)}_{3,3} \end{pmatrix}.$$

Therefore, if the elements of $A_{2,2}$ are not available in core at the time the first block row $(A_{1,1}, A_{1,2})$ is being processed, the modifications can be accumulated in the $\{s^{(iv)}_{k,j}\}$'s which are later added to the respective elements of $A_{2,2}$ when they are read into memory.

Although it appears straightforward to generalize the block LDL^T scheme to a symmetric sparse block matrix such as the example given in Figure 9, where each $K_{i,j}$ is a dense square matrix of some uniform dimension, an efficient implementation of the sparse block LDL^T scheme requires sophisticated data structures.

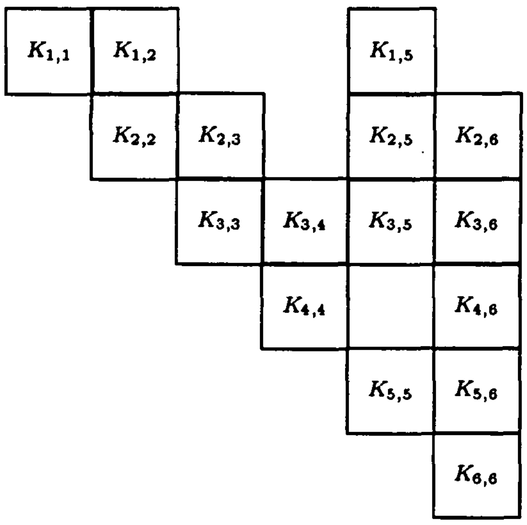


Figure 9: Upper triangular block structure of a symmetric sparse matrix $\mathcal{K}$.

Memory requirement: Suppose that the matrix $\mathcal{K}$ in Figure 9 is stored out-of-core and the rows of $\mathcal{K}$ are to be read into memory one block row (i.e., *JDF* rows if *JDF* is the dimension of each submatrix) at a time. In order to factor the first block row $(K_{1,1}, K_{1,2}, K_{1,5})$ and store the modifications to be applied to the blocks $K_{2,2}$, $K_{2,5}$ and $K_{5,5}$ later, we need memory space to store the blocks in Figure 10 as well as the indexing overhead incurred by the data structures employed. In order to proceed

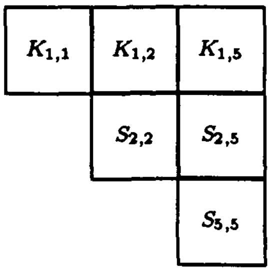


Figure 10: The storage needed for processing $(K_{1,1}, K_{1,2}, K_{1,5})$

with the factorization of the second block row $(K_{2,2}, K_{2,3}, K_{2,5}, K_{2,6})$, there must be enough working space to accommodate the blocks in Figure 11. To minimize the memory requirement, processor INV actually re-uses the space occupied by blocks $K_{1,1}$, $K_{1,2}$ and $K_{1,5}$ to accommodate the blocks needed for the current elimination stage, assuming that the factors of $(K_{1,1}, K_{1,2}, K_{1,5})$ have been archived to the database. The block submatrices needed to remain in-core for each of the next four stages are

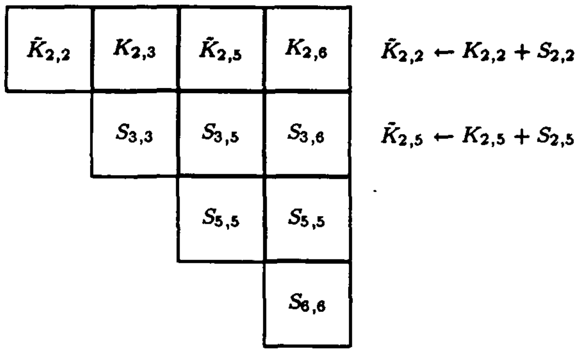


Figure 11: The storage needed for processing $(K_{2,2}, K_{2,3}, K_{2,5}, K_{2,6})$

depicted in Figure 12. Observe that although $K_{4,5}$ block is null in $\mathcal{K}$, it is to be filled in the third elimination stage. Therefore, the space to accommodate $S_{4,5}$ block must be allocated. Fortunately, the fill-in locations can be determined prior to the numerical factorization phase. With the fill-in information available, the maximum

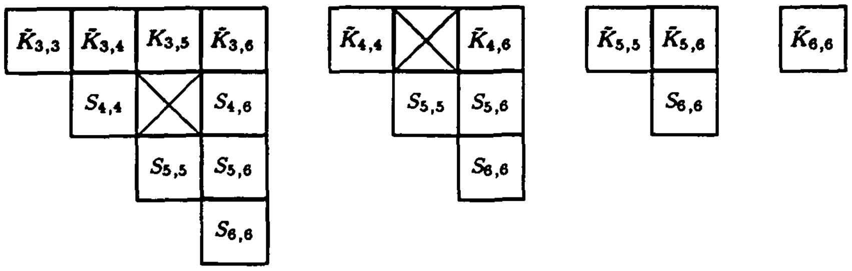


Figure 12: The storage needed to process $K_{3,*}$, $K_{4,*}$, $K_{5,*}$ and $K_{6,*}$.

number of submatrices ever needed to be in-core can also be determined. As far as the indexing overhead is concerned, a simple and effective strategy is to store one pointer for each submatrix assuming that the elements within each submatrix are stored in consecutive locations. Using this indexing strategy, the number of pointers required to be in-core for each particular elimination stage is equal to the number of submatrices to be present.

Data structures: The data structures employed by the Testbed matrix processors can again be more easily explained using our block 2×2 example given in Figure 8.

Data structure of the input coefficient matrix: Processor INV assumes that the block upper triangular part of the coefficient matrix

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ & A_{2,2} \end{pmatrix}$$

is stored out-of-core in a block-row-oriented manner. That is, the data of the blocks are stored in a one dimensional array following the block sequence as depicted in Figure 13.

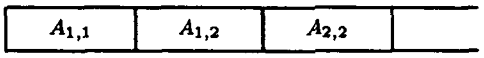


Figure 13: The block sequence of input matrix.

Within each $A_{i,j}$ block, the elements are stored column by column. For example, the elements of the $A_{1,1}$ block are stored following the sequence in Figure 14.

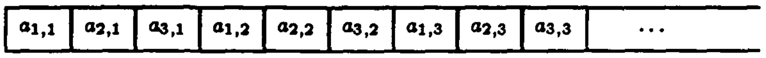


Figure 14: The element sequence of block $A_{1,1}$.

Data structure of the working array S: Except for the first block row of the input matrix, the data retrieved from the buffer for each block row are necessarily updated by adding the modifications $\{s_{i,j}\}$ accumulated in a working array S. Therefore, it is not surprising that the elements within each $JDF \times JDF$ ($JDF = 3$ in our example) $S_{i,j}$ block are stored in the same manner as the input $A_{i,j}$ block. For our example, the diagonal block $A_{2,2}$ must be updated by $S_{2,2}$ before it can be factored. Suppose that the respective addresses of these two blocks in the buffer and working array are given by the pointers *KMAP(IX)* and *AMAP(JX)* as depicted in Figure 15.

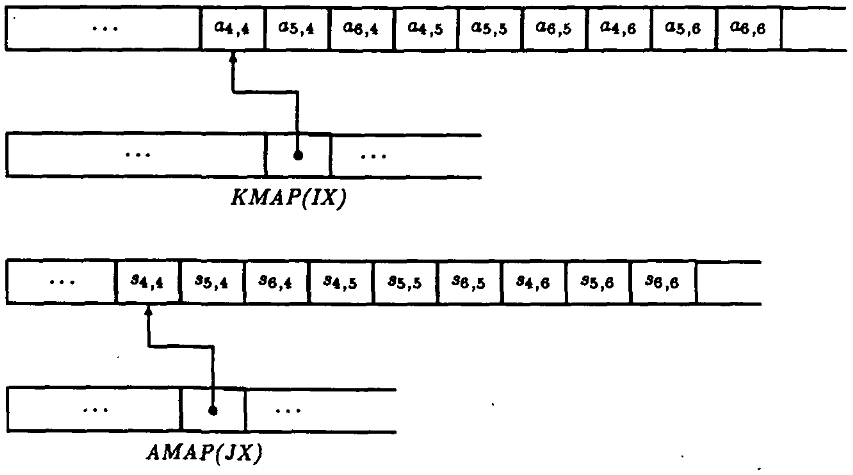


Figure 15: Indexing the buffer and working arrays.

The integration of $A_{2,2}$ into the working array is accomplished by the following segment of FORTRAN statements.

```
C           ------------------------
C           GET POINTER TO THE BUFFER
C           ------------------------
            LKSUB = KMAP(IX)
C           -----------------------------------------
C           GET POINTER TO WORKSPACE STORAGE FROM AMAP
C           -----------------------------------------
            K = AMAP(JX)
            DO 100 J = 1, JDF
               DO 200 I = 1, JDF
                  S(I,J,K) = S(I,J,K) + A(LKSUB)
                  LKSUB = LKSUB + 1
      200      CONTINUE
      100   CONTINUE
```

For a general sparse matrix, because the data stored in the working array S is dynamically changed by accommodating new data in the space occupied by data which have been written out to the database, the $S_{i,j}$ blocks corresponding to the consecutive $A_{i,j}$ blocks in the buffer array may not be neighbors in the working storage. To integrate $NSUBS$ ($NSUBS \geq 1$) $A_{i,j}$'s into S, the starting address of each $S_{i,j}$ must be computed from $AMAP$ each time, resulting in the revised code segment.

```
C           ------------------------
C           GET POINTER TO THE BUFFER
C           ------------------------
            LKSUB = KMAP(IX)
            DO 300 ISUB = 1, NSUBS
C               -----------------------------------------
C               GET POINTER TO WORKSPACE STORAGE FROM AMAP
C               JGAP IS KNOWN FROM THE DATA FORMAT OF AMAP
C               -----------------------------------------
                JX = JX + JGAP
                K = AMAP(JX)
                DO 100 J = 1, JDF
                   DO 200 I = 1, JDF
                      S(I,J,K) = S(I,J,K) + A(LKSUB)
                      LKSUB = LKSUB + 1
      200          CONTINUE
      100       CONTINUE
      300   CONTINUE
```

Conversion of the input data structure: Note that the data structure described above for the input buffer and working array is in fact the output format of the processor which assembles the system stiffness matrix from the finite element model. Since the block LDL^T factorization scheme and the following forward/backward substitution algorithms are row-oriented, the properly updated $JDF \times JDF$ submatrices of the current block row are copied from S into another one-dimensional array B, where the data are stored row by row with respect to the global matrix. For example, assuming that the dimensions of S and B are declared as *S(JDF,JDF,*)* and *B(JDF,CONRNG,*)*, the following FORTRAN statements retrieve the first row of $(A_{1,1}, A_{1,2})$, i.e. $\{a_{1,1}, a_{1,2}, \cdots, a_{1,6}\}$, from S and store them in the consecutive locations in B.

```
C             ------------------------------
C             K INDEXES THE CURRENT ROW IN B
C             ------------------------------
              K = 1
C             ------------------------------
C             M INDEXES THE CURRENT ROW IN S
C             ------------------------------
              M = 1
C             -----------------------------------------
C             OBTAIN THE NUMBER OF BLOCKS IN CURRENT ROW
C             -----------------------------------------
              CONRNG = 2
              DO 100 J= 1, CONRNG
C                --------------------------------------
C                ASSUME THAT THE LOCATION OF THE CURRENT
C                BLOCK IN S CAN BE OBTAINED FROM SUBMAP(J)
C                --------------------------------------
                 LKSUB = SUBMAP(J)
                 DO 200 I = 1, JDF
                    B(I,J,K) = S(M,I,LKSUB)
 200             CONTINUE
 100          CONTINUE
```

Since the modifications computed from B are to be accumulated into S for updating the input matrix in the future stages of the elimination process, the dimensioning of B as *B(JDF,CONRNG,*)* in parallel with the dimensioning of S is desirable. The conversion of index from S to B, or vice versa, for each element can thus be easily expressed in FORTRAN as demonstrated in the above code segment. However, there are other times the code would be much cleaner by viewing B as a two dimensional array declared as *B(JDFCON,*)*, where $JDFCON = JDF \times CON$. The technique which the processor INV uses to index the same array in either way is to declare two formal parameters, namely *B(JDF,CONRNG,*)* and *BB(JDFCON,*)* in the subroutine which does the factorization, whereas the actual parameters corresponding to B and BB in the calling sequence are *identical*. With this trick, B and BB in the subroutine refer to the same actual parameter and the programmer can work with either B or BB according to his need to access the data in a particular pattern.

Handling zero constraints: Processor INV handles zero constraints by ignoring the corresponding rows in the process of transferring data from S to B. That is, if the unknown $x_i = 0$, then row i will not be copied to B. For example, if it is known that $x_2 = 0$, then only row 1 and row 3 in $(A_{1,1}, A_{1,2})$ would be copied to array B. The actual transformation of $(A_{1,1}, A_{1,2})$ is carried out in B as shown below.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \end{pmatrix} \longrightarrow \begin{pmatrix} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ a_{3,1} & a_{3,2} & d_3^{-1} & m_{3,4} & m_{3,5} & m_{3,6} \end{pmatrix}$$

Consequently, row 3 in S becomes row 2 in B, i.e. it is possible that K<M in our sample code segment.

Handling nonzero constraints: Processor INV handles nonzero constraints by ignoring the corresponding rows in the factorization process. For example, if it is known that $x_3 = u_3 \neq 0$ in addition to $x_2 = 0$, the transformation of $(A_{1,1}, A_{1,2})$ by processor INV will not affect row 3, i.e.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \end{pmatrix} \longrightarrow \begin{pmatrix} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \end{pmatrix}$$

Elements archived: Write out to database those elements of BB which are needed for the subsequent use by processor SSOL in effecting the forward/backward substitution process. For example, assuming $x_2 = 0$, and $x_3 = u_3 \neq 0$, the output elements resulting from factoring the $(A_{1,1}, A_{1,2})$ block are given by

$$\begin{pmatrix} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ & & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \end{pmatrix}$$

3.3 The SSOL Implementation

Input Data: Processor SSOL retrieves from the database the factors archived by processor INV. For our example of the block 2×2 matrix, assuming that the constraints are $x_2 = 0$ and $x_3 = u_3 \neq 0$, the data given below are stored in a row-oriented manner in the database.

$$\begin{pmatrix} d_1^{-1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ & & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ & & & d_4^{-1} & m_{4,5} & m_{4,6} \\ & & & & d_5^{-1} & m_{5,6} \\ & & & & & d_6^{-1} \end{pmatrix}$$

In addition to the factors, the right-hand-side vector $\bar{f}$ and the nonzero-constraint vector $\bar{u}$ are also available in the database.

Handling constraints: In essence, processor SSOL simply adapts the forward/backward substitution schemes we presented for Algorithm I to solve the following triangular systems, which are to be implicitly formed from the data retrieved.

$$\begin{pmatrix} 1 & & & & \\ m_{1,4} & a_{3,4} & 1 & & \\ m_{1,5} & a_{3,5} & m_{4,5} & 1 & \\ m_{1,6} & a_{3,6} & m_{4,6} & m_{5,6} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ u_3 \\ y_4 \\ y_5 \\ y_6 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix}$$

and

$$\begin{pmatrix} 1 & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ & & & 1 & m_{4,5} & m_{4,6} \\ & & & & 1 & m_{5,6} \\ & & & & & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ 0 \\ u_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} d_1^{-1} & & & \\ & d_4^{-1} & & \\ & & d_5^{-1} & \\ & & & d_6^{-1} \end{pmatrix} \begin{pmatrix} y_1 \\ y_4 \\ y_5 \\ y_6 \end{pmatrix}$$

In particular, the SSOL implementation takes advantage of the following observations.

1. The equations corresponding to zero constraints can be ignored in the forward substitution phases.
2. The coefficients of the equations corresponding to nonzero constraints are needed to adjust the right-hand-side vector in the forward substitution phase.
3. If the solution vectors contain the constraints, the equations corresponding to constraints (either zero or nonzero) can be skipped in the backward substitution phase.

Output Solutions: The computed $x_{i,j}$'s are written out to the global database.

3.4 Other Relevant SPAR Processors

In order to briefly introduce the functions of other relevant SPAR processors, and give the readers some idea how they may be used to perform an analysis, we found that the following information available in The CSM Testbed User's Manual [1] useful. Given below is a list of processors together with comments on their individual functions. In addition, the ordering of the processors in the list serves as a template for performing the linear static analysis, which is one of the simplest types of analysis which can be performed with the Testbed.

1. Processor TAB. Define joint locations, constraints, reference frames, and possibly material and section properties. Material and section properties may be defined using either processor TAB or processors AUS and LAU (Steps 2 and 3).

2. Processor AUS. Build tables of material and section properties if the facilities in processor TAB were not used.

3. Processor LAU. Form constitutive matrix if material and section properties were not input in processor TAB.

4. Processor ELD. Define elements. Element definitions include element connectivity, element material reference frame number, element material and section type numbers.

5. Processor E. Initialize element datasets; create the dataset which will contain all important element information (e.g., intrinsic coordinates, element-to-global transformations, intrinsic stiffness matrices).

6. Processor EKS. Calculate element intrinsic stiffness matrices.

7. Processor RSEQ. Resequence nodes for minimum total execution time.

8. Processor TOPO. Form maps which guide the assembly and factorization of system matrices.

9. Processor K. Assemble global (system) stiffness matrix.

10. Processor INV. Factor system stiffness matrix.

11. Processor AUS and EQNF. Create applied nodal loading. If element loading is applied, Processor EQNF must be executed to calculate equivalent nodal loading.

12. Processor SSOL. Solve for static displacements.

13. Processor GSF. Calculate element stress resultants.

14. Post-process using any of the following processors: VPRT, PSF, PLTA, PLTB, PLOT, CONT, T2PT.

4 Developing New Matrix Factorization Processors

4.1 General Considerations

We have described in detail in §3 the internal working of processors INV and SSOL. The former performs the out-of-core LDL^T factorization of a sparse matrix in block form, and the latter solves the resulting triangular systems by forward and backward substitution schemes. The following considerations have prompted us to investigate alternative sparse factorization schemes.

1. The techniques employed by INV are particularly tailored to the large sparse linear systems arising in the structural models. The models considered are composed of finite elements connected at specified joints. Each joint can have three translational and three rotational components of deflection, totaling a maximum of six degrees of freedom per joint. The system stiffness matrix is stored and operated on as an array of $JDF \times JDF$ submatrices, where $3 \leq JDF \leq 6$ is the maximum number of degrees of freedom per joint in the model of a particular problem. However, in general the joints need not have the same number of degrees of freedom. This storage scheme thus necessitates storing dummy data – an identically zero row for each missing degree of freedom at each joint. Although the factorization scheme only operates on the non-null submatrices and some operations on the dummy rows are skipped by the processor INV, it does not fully exploit the sparsity within each submatrix. While this strategy is understandably very efficient if uniform degrees of freedom per joint prevail, it may not best suit the models with drastically varied degrees of freedom, which is not uncommon in finite element modeling applied to disciplines other than mechanical structural analysis.

2. As described in §3, the data structures employed incur the index overhead of one pointer per submatrix for all submatrices occurring in each elimination stages. Therefore, the index overhead is proportional to the number of submatrices instead of the size of them. Consequently, while the primary storage for the system stiffness matrix and the factors is reduced for models with fewer degrees of freedom, the secondary storage for their indices may remain the same and could become a significant part of the total storage. Furthermore, unlike the working storage which is determined by the maximum number of submatrices which ever occur during the entire factorization process, the addresses of the submatrices are repeatedly stored for each elimination stage.

3. The system stiffness matrix, the factors and their respective indexing information are each stored in separate data sets in the global database. The data sets are read into core or written out to the database one record at a time. The choice of record length determines the number of disk read/write operations and the required buffer

space. While the maximum record length of a data set is restricted by the available buffers, the minimum record length must be long enough to contain all of the items which are needed to completely process one entire row of submatrices. Therefore, the processor `INV` can perform in-core factorization if each record of each data set contains all information needed to complete the entire factorization process. In that case, the in-core storage is required to accommodate at least one copy of the system stiffness matrix, one copy of the factors along with the indexing information needed for all elimination stages, and a working array of the same size as needed in the out-of-core case. Since some other out-of-core sparse factorization schemes currently available perform in-place factorization and are readily adapted to performing in-core factorization, it appears worthwhile to compare their performance in both in-core and out-of-core cases.

4. When applying the out-of-core block LDL^T scheme as implemented by the processor `INV` to a dense matrix, its advantage of reducing memory requirement disappears because the working array for the first elimination stage must contain the entire upper triangular part of the stiffness system matrix.

5. The possible ill-conditioning of the system stiffness matrix is not detected by the current Testbed software.

4.2 The Design of an Interface

It is apparent from our earlier discussions that the format of the data sets is directly connected to the factorization scheme currently employed in the Testbed. It is thus likely that the particular arrangement of data items in the data sets may not be compatible with the data-accessing pattern of the other factorization algorithms to be considered. In order to evaluate the performance of alternative sparse factorization schemes in the Testbed without redesigning the database at a time when the scheme of choice is not certain yet, we have devised a set of subroutines which serve as an interface between the global database of the Testbed and SPARSPAK-A [7]. Although some components of the interface are specific for SPARSPAK-A, we hope that its overall design and the availability of some utility modules will prove to be useful in adapting the interface to work with other sparse matrix solvers. A few words about the capabilities of SPARSPAK-A are in order.

4.2.1 SPARSPAK-A: Waterloo sparse linear equations package

In this section we briefly review the important features of SPARSPAK-A, which is a package of Fortran programs designed to efficiently solve large sparse systems of linear equations by direct methods. The structure and use of the package are described in the SPARSPAK-A User's Guide [7]. The collection of algorithms implemented by SPARSPAK-A and their storage schemes are discussed in reference [19]. Although we shall consider only symmetric positive definite problems here, the actual package handles both symmetric and unsymmetric problems subject to the condition that the matrix structure is symmetric and that row and/or column interchanges are not required to maintain numerical stability. To solve a sparse symmetric positive definite linear system

$$Ax = b ,$$

the user and SPARSPAK-A interact through the following steps:

Step 1. The user supplies the nonzero structure of A to the package using a set of subroutines described in Section 2.2 of reference [7].

Step 2. The package finds a "good" ordering (permutation P) for A, and allocates storage for the triangular factorization of $PAP^T = LL^T$, as described in Section 2.3 of reference [7].

Step 3. The user supplies the numerical values for the matrix A to the package, as described in Section 2.4 of reference [7].

Step 4. The package factors PAP^T into LL^T, as described in Section 2.5 of reference [7].

Step 5. The user supplies numerical values for b, as described in Section 2.4 of reference [7]. (This step may come before Step 4, and may be intermixed with Step 3.)

Step 6. The package computes the solution by solving $Ly = Pb$ and $L^Tz = y$, and then setting $x = P^Tz$, as described in Section 2.5 of reference [7].

Step 7. The user may call a subroutine to obtain an estimate of the relative error in x as well as the inverse of the condition number of A if so desired. The subroutine is described in Section 2.6 of reference [7].

The names of the subroutines available for reordering a symmetric matrix in Step 2, together with the algorithms they implemented, are listed in Table 1. Corresponding to each ordering choice in Step 2, a different set of subroutines are provided for Steps 3, 4, 6 and 7. The subroutines used in Steps 1 and 5 are, however, independent of the ordering methods.

In the context of comparing the performance of the SPARSPAK-A factorization algorithm with that of the Testbed processor **INV**, we should note the following. Firstly, the coefficient matrix A will have been ordered differently because the ordering algorithm in the Testbed is applied to the joints in the finite element model before the system stiffness matrix is assembled, whereas SPARSPAK-A reorders the coefficient matrix itself. Since associated with each joint in the finite element model is a dense $JDF \times JDF$ submatrix,

SPARSPAK-A Subroutine	Ordering algorithm
ORDRA1	Reverse Cuthill-McKee ordering [28]
ORDRA3	One-way Dissection ordering [16]
ORDRB3	Refined quotient tree ordering [17]
ORDRA5	Nested Dissection ordering [18]
ORDRB5	Minimum Degree ordering [24]

Table 1: SPARSPAK-A ordering choices.

the resequencing of the joints relocates the *submatrices* (as a whole) in the system stiffness matrix. On the other hand, since the ordering algorithms in SPARSPAK-A are applied to the structure of the *assembled* system stiffness matrix, the zeros within each submatrices (due to constrained variables or dummy rows) may be exploited and the resulting matrix may not be in block form.

Secondly, the Cholesky factorization scheme and the upper/lower triangular system solvers implemented by SPARSPAK-A do not handle constraints or dummy rows (rows of zeros). It is therefore necessary to adjust both the system stiffness matrix and the right hand side before the nonzero structure and the numerical values are input to SPARSPAK-A. In the current version of Testbed, while the constraint information is available in a designated data set, the dummy rows can be detected only by reading the assembled system stiffness matrix. The implication is that the system stiffness matrix has to be examined twice – once for determining its "adjusted" nonzero structure (needed in Step 1), and once for retrieving its numerical coefficients (needed in Step 3). We consider the way we handle the dummy rows as an interim measure until the dataset format of the generalized element processor is available. It is expected that the generalized element processor will neither assume uniform degrees of freedom nor store dummy data. Complete details on adjusting the nonzero structure and the numerical values for input to SPARSPAK-A are given later in this section.

Thirdly, SPARSPAK-A employs a particular version of the Cholesky factorization algorithm. Since this version of the algorithm computes the Cholesky factor one column at a time and the part of the matrix remaining to be factored is not accessed during the scheme, it is commonly referred to as the "Column-Cholesky" algorithm. Depending on how the modifications to each designated column are accumulated, the Column-Cholesky algorithm can be described in two different forms. Given in Figure 16 is the commonly known scalar-product form. These formulae can be derived directly by equating the elements of A to the corresponding elements of the product LL^T.

```
for $j \leftarrow 1, 2, \ldots, n$ do
    $\ell_{j,j} \leftarrow \sqrt{a_{j,j} - \sum_{k=1}^{j-1} \ell_{j,k}^2}$
    for $i \leftarrow j+1, j+2, \ldots, n$
        $\ell_{i,j} \leftarrow \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k}\ell_{j,k}\right) / \ell_{j,j}$
```

Figure 16: The scalar-product Column-Cholesky Factorization Algorithm.

The vector-sum Column-Cholesky algorithm described in Figure 17 is an alternative formulation which avoids explicitly forming the individual inner products.

```
for $j = 1, 2, \ldots, n$ do
    for $k = 1, 2, \ldots, j-1$ do
        $\begin{pmatrix} a_{jj} \\ \vdots \\ a_{nj} \end{pmatrix} \leftarrow \begin{pmatrix} a_{jj} \\ \vdots \\ a_{nj} \end{pmatrix} - l_{jk} \begin{pmatrix} l_{jk} \\ \vdots \\ l_{nk} \end{pmatrix}$
    $l_{jj} \leftarrow \sqrt{a_{jj}}$
    for $k = j+1, j+2, \ldots, n$ do
        $l_{kj} \leftarrow a_{kj}/a_{jj}$
```

Figure 17: The vector-sum Column-Cholesky Factorization Algorithm.

SPARSPAK-A applies the vector-sum Column-Cholesky algorithm to factor a general sparse matrix. The readers are referred to [19] for a comprehensive description of various storage schemes which result in efficient implementations of the algorithm.

For $n = 5$ and $j = 3$, the in-place Column-Cholesky factorization scheme thus transforms $a_{i,3}$ to $\ell_{i,3}$ for $3 \leq i \leq 5$ as depicted in Figure 18. Note that the elements actually involved in computing the third column of L, denoted as L_{*3}, in the above example are shown in Figure 19. They are the coefficients of the third column of A and those of the computed L with their row indices greater than or equal to 3. Liu [22] makes the observation that if A is read into memory one column at a time and each column of L is written out to the auxiliary storage as soon as it is computed, the in-core working space can be economized by keeping only those $\ell_{i,j}$'s which are needed for the current stage of elimination. Suppose the computed $\ell_{i,j}$'s are saved in a linear array sequentially, we use the above example to demonstrate the necessary data reorganization when the size of this working array is *LNZSZE*= 9. As shown in Figure 20, the $\ell_{i,j}$ elements are relocated (by overwriting elements which are not

$$\begin{pmatrix} \ell_{1,1} & & & & \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & a_{3,3} & & \\ \ell_{4,1} & \ell_{4,2} & a_{4,3} & a_{4,4} & \\ \ell_{5,1} & \ell_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} \end{pmatrix} \longrightarrow \begin{pmatrix} \ell_{1,1} & & & & \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & \ell_{3,3} & & \\ \ell_{4,1} & \ell_{4,2} & \ell_{4,3} & a_{4,4} & \\ \ell_{5,1} & \ell_{5,2} & \ell_{5,3} & a_{5,4} & a_{5,5} \end{pmatrix}$$

Figure 18: Computing the third column of the Cholesky factor L.

$$\begin{pmatrix} & & & & \\ & & & & \\ \ell_{3,1} & \ell_{3,2} & a_{3,3} & & \\ \ell_{4,1} & \ell_{4,2} & a_{4,3} & & \\ \ell_{5,1} & \ell_{5,2} & a_{5,3} & & \end{pmatrix} \longrightarrow \begin{pmatrix} & & & & \\ & & & & \\ \ell_{3,1} & \ell_{3,2} & \ell_{3,3} & & \\ \ell_{4,1} & \ell_{4,2} & \ell_{4,3} & & \\ \ell_{5,1} & \ell_{5,2} & \ell_{5,3} & & \end{pmatrix}$$

Figure 19: The $\ell_{i,j}$'s accessed and the $a_{i,j}$'s modified in computing L_{*3}

accessed any more) to make room for the newly computed $\ell_{i,j}$'s. For this example, data reorganization is necessary only before computing the third column and the fourth column of L. Clearly, the larger the size of the working array the fewer number of times the data reorganization needs to be done.

In [22], Liu applies the idea above to large sparse matrices in his development of an adaptive general sparse out-of-core Cholesky factorization scheme. One of the advantages the algorithm features is that the frequency of data structure reorganization is *adaptive* to the available working space. Liu's implementation of the out-of-core Cholesky scheme is compatible with SPARSPAK-A and is intended to be used in Step 3. We have incorporated this set of subroutines into an experimental processor in the Testbed and we shall report its performance on a set of CSM Testbed demonstration problems in §5.

4.2.2 The Design of the Processor SPK

Currently the entire interface together with the driver and a subset of SPARSPAK-A modules are installed as a single processor SPK which can be invoked by the macroprocessor command [XQT SPK during the execution of the Testbed. The *choice* provided by this particular subset of SPARSPAK-A modules is the "Minimum Degree ordering [24]". Following the guideline contained in §6.2.1 of reference [2] for coding new processors, the main program of the processor SPK is implemented as a subroutine (named "*SPK*") called by the Testbed executive module "*NICESPAR*". Referring to the control diagram given in Figure 21, observe that the subroutine *SPK* calls another module "*SPKA*" which serves as

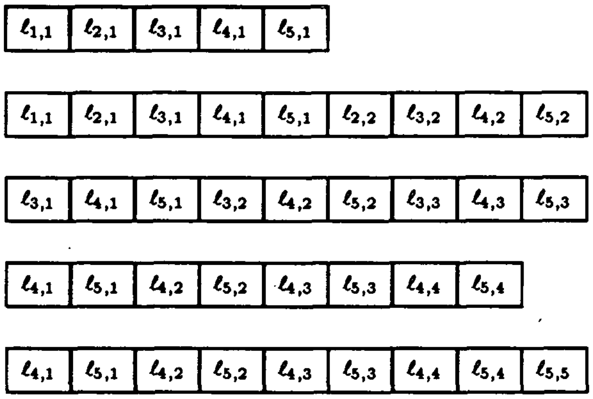

$\ell_{1,1}$ $\ell_{2,1}$ $\ell_{3,1}$ $\ell_{4,1}$ $\ell_{5,1}$
$\ell_{1,1}$ $\ell_{2,1}$ $\ell_{3,1}$ $\ell_{4,1}$ $\ell_{5,1}$ $\ell_{2,2}$ $\ell_{3,2}$ $\ell_{4,2}$ $\ell_{5,2}$
$\ell_{3,1}$ $\ell_{4,1}$ $\ell_{5,1}$ $\ell_{3,2}$ $\ell_{4,2}$ $\ell_{5,2}$ $\ell_{3,3}$ $\ell_{4,3}$ $\ell_{5,3}$
$\ell_{4,1}$ $\ell_{5,1}$ $\ell_{4,2}$ $\ell_{5,2}$ $\ell_{4,3}$ $\ell_{5,3}$ $\ell_{4,4}$ $\ell_{5,4}$
$\ell_{4,1}$ $\ell_{5,1}$ $\ell_{4,2}$ $\ell_{5,2}$ $\ell_{4,3}$ $\ell_{5,3}$ $\ell_{4,4}$ $\ell_{5,4}$ $\ell_{5,5}$

Figure 20: The organization of $\ell_{i,j}$'s in the working array.

the driver of SPARSPAK-A modules. In short, the subroutine *SPKA* allocates memory, sets up the problem by calling CSM-Interface modules, and solves the problem by calling SPARSPAK-A computational modules. The role the CSM-interface modules play is to retrieve the assembled linear system to be solved from the global database and input the problem in an appropriate form to SPARSPAK-A. The design of the processor at this level is thus generic and may be used with other sparse matrix packages.

The CSM-interface consists of twenty-two modules. For easy reference, we list the subroutine or function name of each module and its formal parameters (if there is any) in Table 2 together with those of the two driver subroutines *SPK* and *SPKA*. All of these modules are written in the FORTRAN 77 language and a complete listing of programs is provided in Appendix C of this report. We shall discuss some implementation issues in section §4.2.3 and describe how these modules interface with the Testbed global database and SPARSPAK-A in §4.2.4 and §4.2.5. The usage of the interface is described in section §4.3.

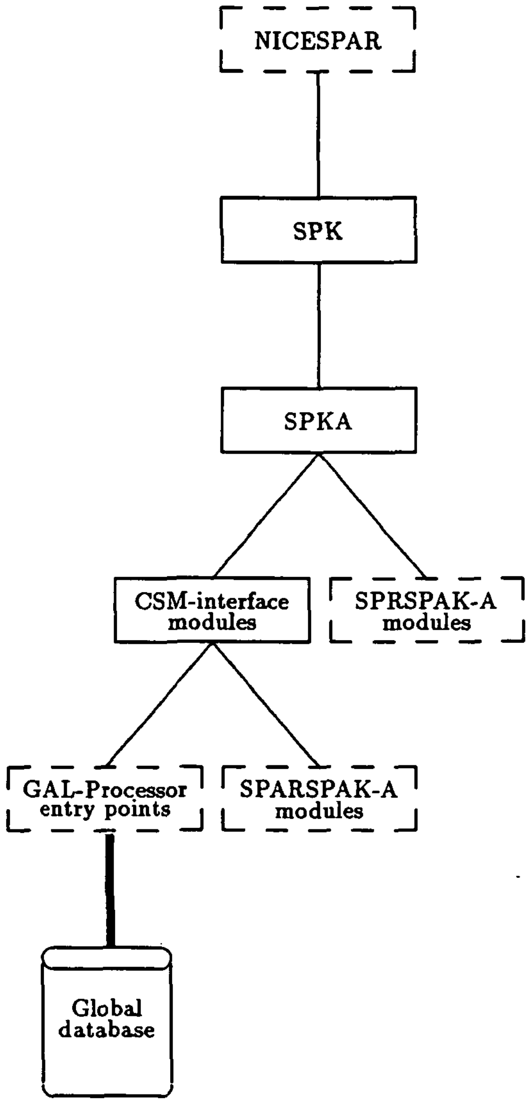


Figure 21: The control diagram of the new processor SPK.

DRIVERS
SUBROUTINE SPK
SUBROUTINE SPKA (A, MXSTOR) DOUBLE PRECISION A(1) INTEGER MXSTOR
CSM-INTERFACE INITIALIZATION MODULES
SUBROUTINE SPKCSM
REAL FUNCTION CTIME (IDUMMY) INTEGER IDUMMY
CSM-INTERFACE PROBLEM INPUT MODULES
SUBROUTINE GETJDF (IBUF) INTEGER*4 IBUF(1)
SUBROUTINE GETDOF (DOF, IBUF) INTEGER*4 DOF(1), IBUF(1)
SUBROUTINE GTZERO (DOF, FBUF, MASK) DOUBLE PRECISION FBUF(1) INTEGER*4 MASK(1), DOF(1)
SUBROUTINE GTCOND (DOF, IBUF, KC, MASK, CSIZE) INTEGER*4 DOF(1), IBUF(1), KC(1), MASK(1), CSIZE
SUBROUTINE GTMOTI (FBUF, MASK, FCON, CSIZE) INTEGER*4 MASK(1), CSIZE DOUBLE PRECISION FBUF(1), FCON(1)
SUBROUTINE GETIJ (DOF, IBUF, ICLQ, MASK, S) INTEGER*4 DOF(1), IBUF(1), ICLQ(1), MASK(1), S(1)
SUBROUTINE GTFORC (FBUF, MASK, S) INTEGER*4 MASK(1) DOUBLE PRECISION FBUF(1), S(1)
SUBROUTINE GTNUM5 (DOF, FBUF, MASK, FCON, S) INTEGER*4 DOF(1), MASK(1) DOUBLE PRECISION FBUF(1), FCON(1), S(1)
CSM-INTERFACE UTILITY MODULES
INTEGER FUNCTION SPACE (IDUMMY) INTEGER*4 IDUMMY
SUBROUTINE LIBOPN
SUBROUTINE QKINFO (DSNAME) CHARACTER*51 DSNAME
SUBROUTINE GTRECI (RECNUM, IBUF, LEN) INTEGER*4 RECNUM, IBUF(1), LEN
SUBROUTINE GTRECF (RECNUM, FBUF, LEN) INTEGER*4 RECNUM, LEN DOUBLE PRECISION FBUF(1)
CSM-INTERFACE ERROR HANDLING MODULES
SUBROUTINE EMSG
SUBROUTINE EMSG0
SUBROUTINE EMSG1
SUBROUTINE EMSG2
SUBROUTINE DEMSG0
CSM-INTERFACE STATISTICS MODULES
SUBROUTINE GETSOL (FBUF, SOL, RATIO) DOUBLE PRECISION FBUF(1), SOL(1), RATIO
SUBROUTINE STATCS

Table 2: The SPK driver and interface modules.

4.2.3 Implementation Issues

The two implementation issues we shall discuss in this section are "memory allocation" and "module/module communication".

Memory allocation Firstly, we note that the maximum working array storage available to the processor SPK is determined by the blank common dimension identically declared in the Testbed executive *NICESPAR* and the subroutine *SPK*, namely

COMMON A(KSZZZ)

Consequently, if the number of words provided by the blank common is insufficient for the processor SPK to solve a particular problem in-core, the dimension of the blank common must be increased, and the testbed and the subroutine *SPK* must both be recompiled.

We supply blank common of dimension *KSZZZ* (words) to the subroutine *SPKA* as a floating-point array of dimension *MXSTOR*. To accomplish this, we have the subroutine *SPK* execute the following statement:

CALL SPKA (A, MXSTOR)

where the value of *MXSTOR* is either *KSZZZ* or *KSZZZ/2* depending on whether *A* is declared as a single-precision or double-precision array in the subroutine *SPKA*.

All integer and floating-point arrays required by the CSM-Interface modules and SPARSPAK-A are then allocated by the subroutine *SPKA* from the one dimensional floating-point array *A(MXSTOR)*. Note that in order to interact with SPARSPAK-A, the user is required to pass a working array *S* to the package and the location of *S* is the only parameter appearing in all of the SPARSPAK-A interface modules. In our case, the array *S* must be allocated from the working array *A(MXSTOR)*. We have thus divided *A(MXSTOR)* into two segments. The top segment accommodates arrays to be passed to the CSM-interface modules and the entire bottom segment is passed to SPARSPAK-A. If we let the variable *MXUSED* denote the size of the top segment, the parameter to be passed to SPARSPAK-A is *A(SPK)*, where *SPK* = *MXUSED*+1.

A labelled common block *CSMMAP* is designated to keep the locations (origins in *A*) of the various arrays. The variables in *COMMON /CSMMAP/* and the relative locations they represent are depicted in Figure 22. The type and size of the working arrays are tabulated in Table 3. Note that the buffer space for reading integer and floating-point records has been overlapped.

Type	Formal parameter	Actual parameter	Size	Comments
INTEGER*4	*DOF*	*A(DOF)*	*NUMJNT*+1	*NUMJNT*≡ total # of joints
	MASK	*A(MASK)*	*NEQNS*	*NEQNS* ≡ total # of equations
	KC	*A(KC)*	*MAXDOF*+1	*MAXDOF* ≡ 6
	ICLQ	*A(ICLQ)*	*MAXDOF*	
	IBUF	*A(BUF)*	*BUFMAX*	maximum buffer length
DOUBLE PRECISION	*FBUF*	*A(BUF)*	*BUFMAX*	maximum buffer length
	FCON	*A(FCON)*	*CSIZE*	total # of nonzero constraints
	SPK	*A(SPK)*	*MAXSTOR−SPK*+1	the bottom segment of *A*

Table 3: The type and size of the SPK working arrays.

Module/module communication The following labelled common blocks have been used to organize the communication between the SPK modules and the CSM Testbed modules, between the SPK modules and the SPARSPAK-A modules, and among the modules within the interface.

1. *COMMON /IANDO/ IIN, IOUTX*. The two integer variables contain user input and output unit numbers assigned by the Testbed subroutine *INTRO* when the new processor begins execution.

   The */IANDO/* common appears in the SPK initialization subroutine *SPKCSM* and the SPARSPAK-A initialization subroutine *SPRSPK*.

2. *COMMON /SPAUSR/ MSGLVA, IERRA, MAXSA, NVARS*. The */SPAUSR/* common allows user and/or processor SPK to communicate with SPARSPAK-A or vice versa. The meaning of the four integer variables are explained in §4.3.2 and §4.3.3.

   The */SPAUSR/* common appears in the SPK subroutine *SPKA* which serves as the driver of SPARSPAK-A.

3. The following common blocks are for communication among the SPK modules.

   COMMON /CSMSYS/ (6 variables)
   COMMON /CSMSPK/ (6 variables)
   COMMON /CSMUSR/ (11 variables)
   COMMON /CSMMAP/ (7 variables)
   COMMON /CSMCON/ (4 variables)
   COMMON /CSMDTA/ (8 variables)
   COMMON /PRBLEM/ (3 variables)

   The collection of related variables into a labelled common block avoids passing long parameter lists in the use of the subroutines and yet makes the coupling between modules easy to identify. Comments on the variables contained in these labelled commons are made at appropriate places throughout sections §4.2.4, §4.2.5 and §4.3.

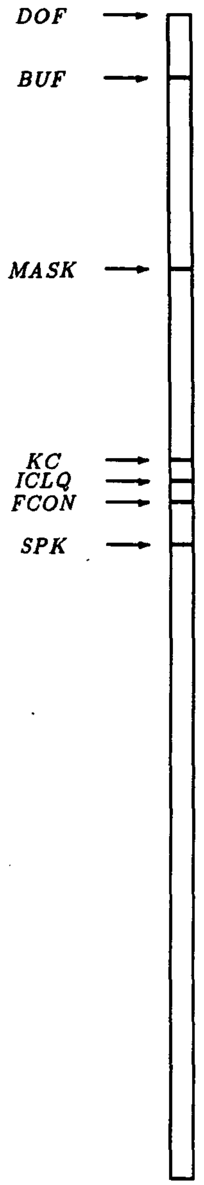


Figure 22: Storage allocation of the SPK working arrays.

4.2.4 Interfacing with the Global Database

There are eight modules in the interface which retrieve data from the global database and process them. The names of these subroutines are "*GETJDF*", "*GETIJ*", "*GTZERO*", "*GTCOND*", "*GTFORC*", "*GTMOTI*", "*GTNUM5*" and "*GETSOL*". We shall use "*G*xxxxx" to represent an arbitrary one of them. All of these modules retrieve data sets from the Testbed via two utility modules which are either "*QKINFO* and *GTRECI*" (for retrieving integer records) or "*QKINFO* and *GTRECF*" (for retrieving records containing floating-point numbers). Figure 23 depicts the coupling of the interface modules with the GAL-processors. The readers are referred to [3] for a complete description of the calling sequence and the operation of each GAL-processor employed.

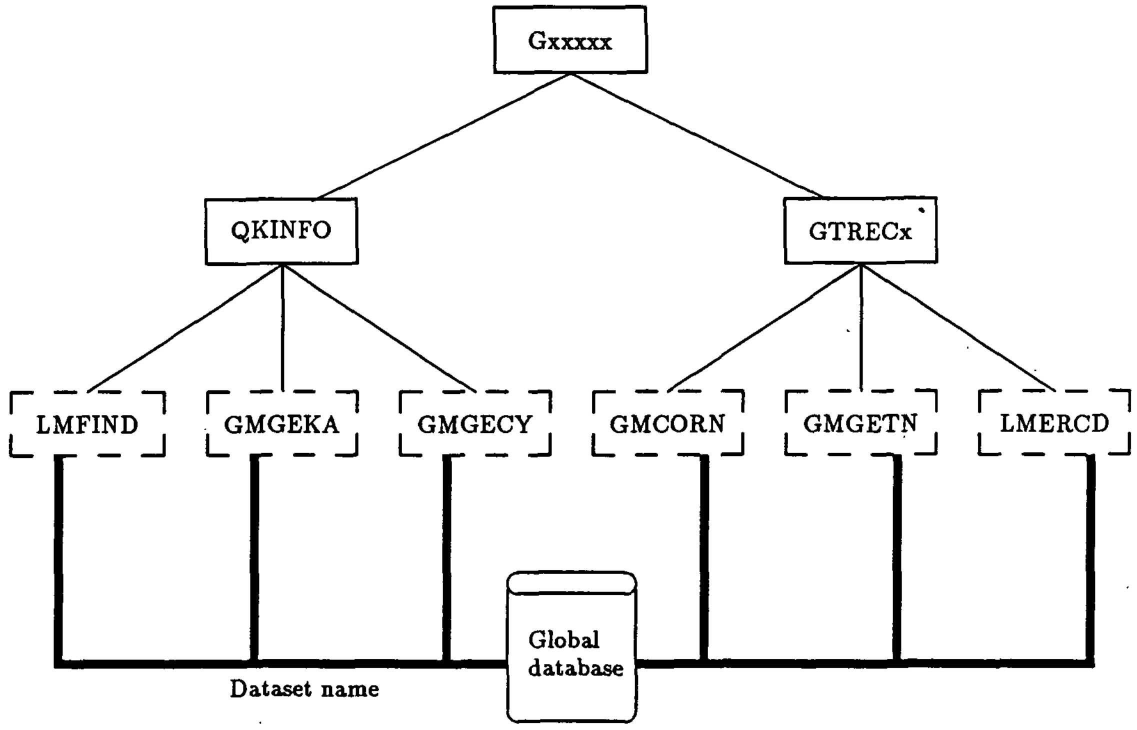


Figure 23: The coupling of CSM-interface modules and GAL-processors.

For each designated data set, the labelled common */CSMSPK/* is used to

1. provide the input arguments *LDI* and *TRACE* to the GAL-processors. (The meaning of *LDI* and *TRACE* is given in Table 4.)

2. store the dataset attributes the interface module *QKINFO* acquires from the GAL-processors *LMFIND*, *GMEGKA* and *GMGECY*.
3. communicate the dataset attributes to the interface modules *G*xxxx, and the GAL-processors *GMCORN* and *GMGETN* via the interface module *GTRECI* or *GTRECF*.

The */CSMSPK/* common thus appears in *QKINFO*, *GTRECI*, *GTRECI* and each *G*xxxx module. The variables contained in */CSMSPK/* and their meaning are given in Table 4.

COMMON /CSMSPK/	
variable	meaning
IDSN	Dataset sequence number.
LDI	Logical Device Index of library device.
NLEN	The record length.
NREC	The number of records in the data set.
RTYPE	The data type.
TRACE	A positive integer used as identifying label in error traceback prints.

Table 4: The variables in *COMMON /CSMSPK/*.

Since the actual data contained in each data set is unique, each subroutine *G*xxxxx must be specifically coded to interpret the data retrieved. The data sets to be accessed by the eight interface modules are listed in Table 5. For each data set, given in Table 5 are also the name of its source processor and the name of the dedicated interface module. The last column of Table 5 indicates the appropriate utility module which should be called to retrieve the type of data provided by the specified data set.

Source Processor	Dataset	*G*xxxxx	*GTREC*x
TAB	*JDF1.BTAB.1.8*	*GETJDF*	*GTRECI*
K	*K.SPAR.jdf2.0*	*GTZERO*	*GTRECF*
TAB	*CON.0.ncon.0*	*GTCOND*	*GTRECF*
TOPO	*KMAP.0.nsubs.ksize*	*GETIJ*	*GTRECI*
AUS	*APPL.FORC.iset.1*	*GTFORC*	*GTRECF*
AUS	*APPL.MOTI.iset.1*	*GTMOTI*	*GTRECF*
K	*K.SPAR.jdf2.0*	*GTNUM5*	*GTRECF*
SSOL	*STAT.DISP.iset.ncon*	*GETSOL*	*GTRECF*

Table 5: Datasets accessed by *G*xxxxx and *GTREC*x.

The data retrieved from each data set and how they are handled by the interface routines are described below. The readers are referred to reference [2] for a description of the format of each data set.

JDF1.BTAB.1.8 provides the total number of joints and the maximum number of active (unconstrained) degrees of freedom a joint may have in the model.

The subroutine *GETJDF* retrieves the data and stores them in the variables *NUMJNT* and *MAXDOF* in the labelled common

/PRBLEM/ MAXDOF, NEQNS, NUMJNT

In an attempt to be flexible in handling the more general case in the future, the subroutine *GETDOF* stores the active degrees of freedom for each individual joint in an accumulated form in an integer array *DOF* so that the number of degrees for joint #*I* can be computed from *DOF(I+1)−DOF(I)*, where *DOF(1)=1*, and that *DOF(NUMJNT+1)−DOF(1)* gives the total number of equations of the assembled system. The latter value is also stored in the variable *NEQNS* in the */PRBLEM/* common. Since the current version of the CSM Testbed assumes uniform degrees of freedom per joint in storing the system stiffness matrix, *DOF(I+1)−DOF(I)=MAXDOF* for $1 \leq I \leq NUMJNT$.

K.SPAR.jdf2.0 provides the assembled global stiffness matrix stored as an array of $JDF \times JDF$ submatrices, where *JDF* is the maximum degrees of freedom in the model and its value is available from the the variable *MAXDOF* in the */PRBLEM/* common block. Note that the integer *jdf*2 in the name of this data set is the square of the value of *JDF*.

The subroutine *GTZERO* retrieves the system stiffness matrix and detects dummy rows by examining its diagonal elements. For each zero diagonal coefficient detected, a zero is entered into the integer array *MASK* at the location *MASK(I)*, where *I* is the equation number of the dummy row. The convention we have adopted is that *MASK(J)*= −1 if the J^{th} equation is neither constrained nor a dummy row, *MASK(J)*= 0 if it corresponds to a dummy row or a zero constraint, *MASK(J)*= 1 if it corresponds to a nonzero constraint.

CON.0.ncon.0 provides constraint information for each joint degree of freedom. The information available indicates for each joint which component is free, which component is constrained to be zero and which component has a non-zero constraint. Such information is encoded so that one integer is stored for each joint in the model. The current encoding mechanism assumes that the maximum number of degrees of freedom a joint may have is "six". The constraints corresponding to the six degrees of freedom are encoded into the right most six bits of a seven-bit integer. The subroutine *DECODE* accepts an integer as input and returns the status of each of the *MAXDOF* degrees of freedom in the leading *MAXDOF* locations of a working array of length seven.

The subroutine *GTCOND* retrieves the encoded data from *CON.0.ncon.0*, calls *DECODE* to obtain the constraint status for each joint in the model, and sets the corresponding entries in the integer array *MASK* to be "0" or "1" as explained above. An integer output parameter *CSIZE* records the total number of nonzero constraints whose numerical values are expected to be available in the data set *APPL.MOTI.iset.1*.

Therefore, after both subroutines *GTZERO* and *GTCOND* are executed, all constraint information is available for other SPK modules in the integer array *MASK*. Note that we have treated the dummy rows as if they correspond to zero constraints.

KMAP.0.nsubs.ksize provides the block nonzero structure of the system stiffness matrix. Note that the value of *nsubs* in the name of the data set represents the total number of submatrices in the system stiffness matrix for the model, and that the integer *ksize* is the maximum number of joints active at any time during the assembly of the system matrix.

The subroutine *GETIJ* accesses *KMAP.0.nsubs.ksize* and the integer array *MASK* to obtain the matrix structure for input to SPARSPAK-A. We explain how the constraints are handled in section §4.2.5.

APPL.FORC.iset.1 provides applied forces and moments on each joint in each active direction. The integer *iset* in the dataset name identifies a unique load case.

The subroutine *GTFORC* retrieves the data but inputs a retrieved numerical value as a component of the right hand side vector to SPARSPAK-A only if it does not correspond to a variable constrained to be zero (i.e., *MASK(I)*$\neq 0$ if I is the equation number).

Since the right hand side is initialized to be identically zero in SPARSPAK-A, and the modifications to the right hand side caused by nonzero constraints are to be "added" to the appropriate components by subroutine *GTNUM5*, the input of right hand side to SPARSPAK-A is not completed before the subroutine *GTNUM5* is executed.

APPL.MOTI.iset.1 provides applied motions on each joint in each active direction. As mentioned earlier, the integer *ncon* in the name of this data set identifies a particular constraint case, and numerical values for the nonzero constraints detected by the subroutine *GTCOND* are expected from this data set.

The subroutine *GTMOTI* retrieves the available applied motions and stores them in a floating-point array *FCON(I)*, where $1 \leq I \leq$*CSIZE*, and *CSIZE* is the total number of nonzero constraints determined in the subroutine *GTCOND*. Therefore, when *CSIZE*$= 0$, the subroutine *GTMOTI* will return without attempting to access the data set. However, when *CSIZE*> 0, it is a fatal error if the data set is missing or less than *CSIZE* values are available.

The data set *K.SPAR.jdf2.0* can now be accessed the second time by a different subroutine *GTNUM5* for the input of numerical values to SPARSPAK-A. The two arrays *MASK* and *FCON* are passed to the subroutine *GTNUM5* so that it can appropriately handle the constraints as explained in section §4.2.5.

STAT.DISP.iset.ncon provides the computed static displacements for each joint in each active direction. Unique solution is obtained by specifying the load set and constraint case in the name of the data set.

The subroutine *GETSOL* retrieves the Testbed solution from this data set and verifies the correctness of the SPARSPAK-A solution by computing its relative error with respect to the Testbed solution. More details in this aspect are provided in section §5 on numerical experiments.

4.2.5 Interfacing with SPARSPAK-A

The processor SPK may interact with SPARSPAK-A via the interface modules given in Table 6, which correspond to our choice of the minimum degree ordering (*subroutine ORDRB5*) for the new processor.

Initialization of SPARSPAK-A	*SPRSPK*
Structure input	*IJBEGIN* *INIJ (I, J, S)* *INROW (I, NIR, IR, S)* *INIJIJ (NIJ, II, JJ, S)* *INCLQ (NCLQ, CLQ, S)* *IJEND(S)*
Ordering	*ORDRB5 (S)*
Matrix input	*INAIJ5 (I, J, VALUE, S)* *INROW5 (I, NIR, IR, VALUES, S)* *INMAT5 (NIJ, II, JJ, VALUES, S)*
Right hand side input	*INBI(I, VALUE, S)* *INBIBI (NI, II, VALUES, S)* *INRHS (RHS, S)*
Factorization and/or Solution	*SOLVE5 (S)*
Relative error estimation	*EREST5 (RELERR, S)*
Print statistics	*STATSA*
Save and Restart the computation	*SAVEA (K, S)* *RSTRTA (K, S)*

Table 6: SPARSPAK-A interface modules - a subset.

The coupling of the SPK modules and SPARSPAK-A is depicted in Figure 24. The modules which interact with SPARSPAK-A are "*SPKA*", "*GETIJ*", "*GTFORC*" and "*GTNUM5*". The module *SPKA* serves as the driver program of SPARSPAK-A. The module *GETIJ* inputs the nonzero structure of the system stiffness matrix to SPARSPAK-A. The modules *GTFORC* and *GTNUM5* are involved in inputting nonzero coefficients and the right hand side to SPARSPAK-A. The particular SPARSPAK-A subroutines to be called by each of these interface modules are explicitly given inside the dotted boxes.

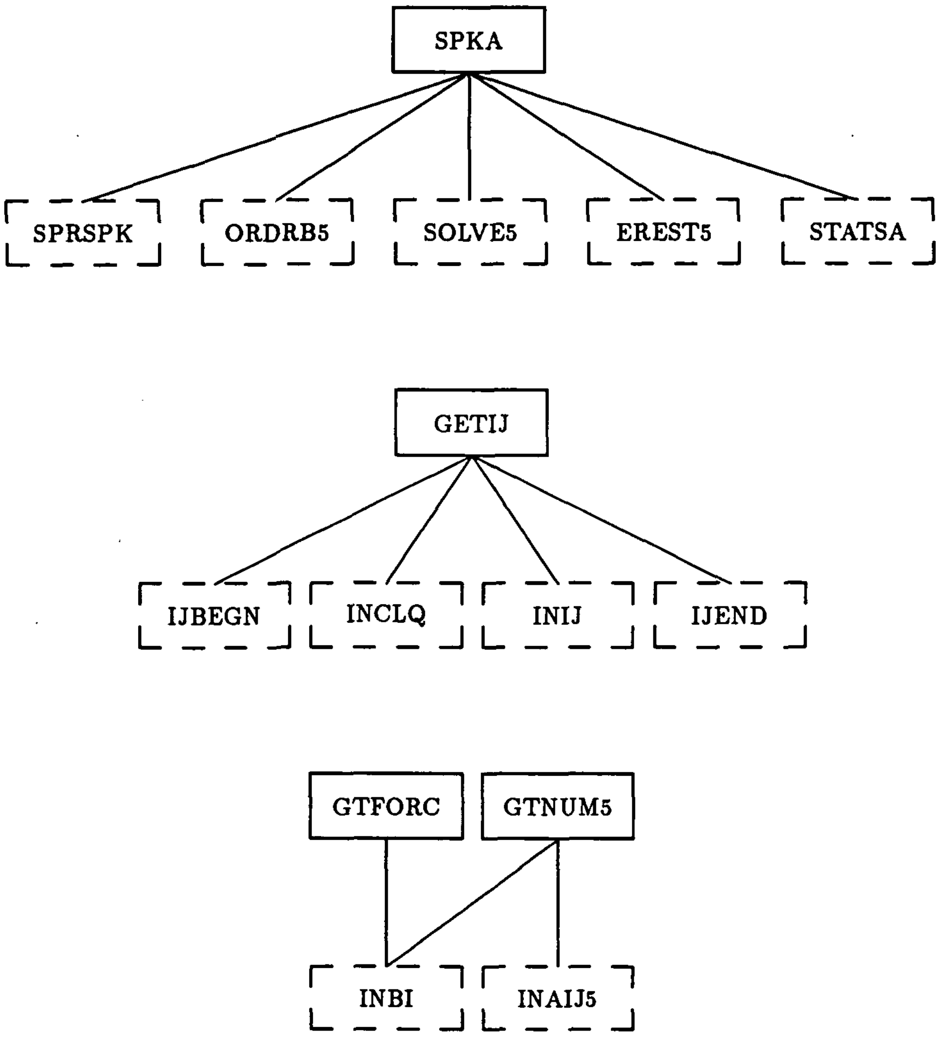


Figure 24: The coupling of the processor SPK and SPARSPAK-A.

Since SPARSPAK-A modules do not handle constraints, the retrieved system stiffness matrix and the right hand side must be adjusted before they can be input to SPARSPAK-A. The necessary modifications to the structure and the numerical values are detailed below.

Input the structure of the system stiffness matrix to SPARSPAK-A - In this section we describe how the subroutine *GETIJ* inputs the the structure of the system stiffness matrix to SPARSPAK-A. The data set *KMAP..nsubs.ksize* contains the system topology map. From this map we can retrieve the following information for each joint.

JNT - The number of the current joint.

CONRNG - The number of submatrices including the diagonal in the upper triangle for the current joint.

CONECT(CONRNG-1) - A list of joints connected to the current joint.

Let us consider the following finite-element model which is given as an example in [29].

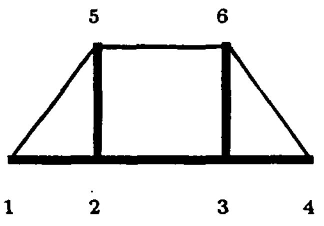


Figure 25: A model.

Element #	type	Connected Nodes
1	BEAM	1,2
2	"	2,3
3	"	3,4
4	"	2,5
5	"	3,6
6	PLATE	1,2,5
7	"	2,3,6,5
8	"	3,4,6

Table 7: A model.

For this model, the information expected to be available in *KMAP..nsubs.ksize* is listed in Table 8.

JNT	CONRNG	CONECT(CONRNG-1)
1	3	2, 5
2	4	3, 5, 6
3	4	4, 5, 6
4	2	6
5	2	6
6	1	

Table 8: From data set *KMAP..nsubs.ksize*.

Given in Figure 26 is the upper triangular block structure of the system matrix (including the diagonal blocks) described by Table 8.

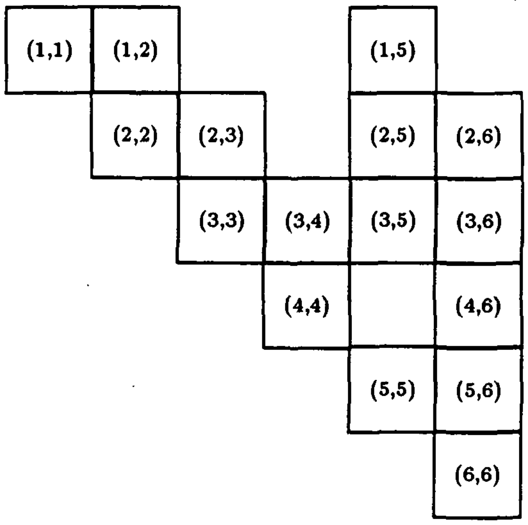


Figure 26: Upper triangular block structure of the system matrix for the model problem.

If each joint has three degrees of freedom in the model, each block is a 3×3 submatrix and the system stiffness matrix $\mathcal{K}$ has the nonzero entries as given in Figure 27.

$$\left(\begin{array}{cccccccccccccccccc}
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & \otimes & \otimes & & & \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & \otimes & \otimes & & & \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & \otimes & \otimes & & & \\
 & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
 & & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
 & & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
 & & & & & & & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes \\
 & & & & & & & & & & & & & & & \otimes & \otimes & \otimes \\
 & & & & & & & & & & & & & & & \otimes & \otimes & \otimes \\
 & & & & & & & & & & & & & & & \otimes & \otimes & \otimes
\end{array}\right)$$

Figure 27: Nonzero entries in the upper triangle of $\mathcal{K}$ (including diagonal submatrices.)

If every degree of freedom is *active* (unconstrained) on each joint, then the structure input to SPARSPAK-A is as specified in Figure 27. It should be pointed out that because SPARSPAK-A anticipates only "symmetric" nonzero structure, the structure input routine always records a logical nonzero in both (i, j) and (j, i) positions regardless of which index pair being actually entered. Furthermore, the package automatically removes duplications so that it does not matter if both (i, j) and (j, i) pairs are entered.

In order to demonstrate how we handle the constrained degrees of freedom, let us assume that the second degree of freedom on joint #5 is constrained. In this case, the corresponding columns and rows of data in $\mathcal{K}$ except for the diagonal elements will be treated as zero entries. The nonzero positions SPARSPAK-A is informed of consist of the remaining nonzeros as given in Figure 28.

$$\begin{pmatrix}
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & & \otimes & & & \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & & \otimes & & & \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & & \otimes & & & \\
& & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & \otimes & & & & \\
& & & & & & & & & & & & \otimes & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & & & \otimes & \otimes & \otimes
\end{pmatrix}$$

Figure 28: Remaining nonzero entries in the upper triangle of $\mathcal{K}$.

As seen from Figure 28, the equations corresponding to the constrained degree of freedom is the fourteenth equation. We have thus ignored the nonzero entries in locations $(i, 14)$ and $(14, i)$ for all i's except for the diagonal entries. Accordingly, the numerical coefficients corresponding to these ignored locations must not be input to SPARSPAK-A and the right hand side must be appropriately adjusted to reflect the change of the system matrix. We next explain the internal working of our numerical input module.

Input the numerical values to SPARSPAK-A - The subroutine which inputs the numerical values to SPARSPAK-A and modifies the right hand side according to each constrained degree of freedom is *GTNUMi*, where $i = 1, 3$, and 5 distinguishes the SPARSPAK input modules *INAIJi* called for each ordering.

To see how the right hand side should be modified, we refer to Figure 29 for the same example, where we label each ignored coefficient $a_{i,j}$ explicitly, and indicate that the coefficient for the diagonal entry $a_{14,14}$ is set to 1.

Let the nonzero constraint corresponding to the second degree of freedom on joint #5 be c_{14}. Our change to the system matrix and right hand side should reflect the following.

1. The fourteenth equation is replaced by

$$x_{14} = c_{14} .$$

$$\left(\begin{array}{cccccccccccccccccc}
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & a_{1,14} & \otimes & & & \\
& \otimes & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & a_{2,14} & \otimes & & & \\
& & \otimes & \otimes & \otimes & \otimes & & & & & & & \otimes & a_{3,14} & \otimes & & & \\
& & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & a_{4,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & \otimes & \otimes & \otimes & \otimes & \otimes & & & & \otimes & a_{5,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & & \otimes & \otimes & \otimes & \otimes & & & & \otimes & a_{6,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & a_{7,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & a_{8,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & \otimes & \otimes & \otimes & \otimes & \otimes & a_{9,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & & \otimes & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & \otimes & \otimes & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & & \otimes & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & \otimes & a_{13,14} & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & 1 & a_{14,15} & a_{14,16} & a_{14,17} & a_{14,18} \\
& & & & & & & & & & & & & & \otimes & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & & & \otimes & \otimes & \otimes \\
& & & & & & & & & & & & & & & & \otimes & \otimes \\
& & & & & & & & & & & & & & & & & \otimes
\end{array}\right)$$

Figure 29: Nonzero entries in the upper triangle of $\mathcal{K}$.

2. Modify the right hand side to be

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \end{pmatrix} \longleftarrow \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \end{pmatrix} - \begin{pmatrix} a_{1,14} \\ a_{2,14} \\ a_{3,14} \\ a_{4,14} \\ a_{5,14} \\ a_{6,14} \\ a_{7,14} \\ a_{8,14} \\ a_{9,14} \\ 0 \\ 0 \\ 0 \\ a_{13,14} \\ 0 \\ a_{14,15} \\ a_{14,16} \\ a_{14,17} \\ a_{14,18} \end{pmatrix} c_{14} \quad .$$

Thus, the right hand side elements b_i, $i = 1,2,3,4,5,6,7,8,9,13$ are modified to be

$$b_i \leftarrow b_i - a_{i,14} \times c_{14}$$

and the right hand side element b_j, $j = 15, 16, 17, 18$ must be modified to be

$$b_j \leftarrow b_j - a_{14,j} \times c_{14} .$$

To summarize, for each $a_{i,i}$ coefficient retrieved from the data set *K.SPAR..**, subroutine *GTNUMi* checks whether the corresponding degree of freedom is constrained. If that is the case, the value of 1.0 will be input to SPARSPAK-A as $a_{i,i}$ and the constraint value is input to SPARSPAK-A as b_i.

For each off-diagonal element $a_{i,j}$, *GTNUMi* checks the following four possible cases.

1. If both x_i and x_j are constrained, no action needs to be taken.
2. If x_i is active and x_j is constrained to be c_j then

$$b_i \leftarrow b_i - a_{i,j} \times c_j .$$

3. If x_i is constrained to be c_i and x_j is active then modify

$$b_j \leftarrow b_j - a_{i,j} \times c_i .$$

4. If neither x_i nor x_j is constrained, input the retrieved $a_{i,j}$ value to SPARSPAK-A and specify the location to be (j, i). (SPARSPAK-A requires the numerical value to be input for the lower triangular part only.)

4.3 The Usage of the Interface

4.3.1 The Execution Path

The usage of the interface in solving a testbed problem is reflected by the execution path of the subroutine *SPKA* as sketched in Table 9. The execution sequence is enforced by checking and updating the value of the variable *STAGE* in the common block */CSMCON/*. The values of *STAGE* for the successful completion of each corresponding step are listed in the last column of Table 9.

Execution path		SPARSPAK-A subroutine	Interface subroutine	Dataset dependency	*/CSMCON/ STAGE*
Step 1.1	Start	*SPRSPK*	*SPKCSM*		0
1.2			*LIBOPN*		10
1.3			*CTIME*		
Step 2.1	Problem input		*GETJDF*	*JDF1.BTAB.1.8*	20
2.2			*GETDOF*		30
2.3			*GTZERO*	*K.SPAR.jdf2.0*	40
2.4			*GTCOND*	*CON.0.ncon.0*	50
2.5			*GTMOTI*	*APPL.MOTI.iset.1*	60
Step 3	Structure input	*IJBEGIN* *INCLQ* *INIJ* *IJEND*	*GETIJ*	*KMAP.0.nsubs.ksize*	70
Step 4	Order and allocate storage	*ORDRB5*			
Step 5	Input numerical values for b	*INBI*	*GTFORC*	*APPL.FORC.iset.1*	80
Step 6	Input numerical values for A and b	*INAIJ5* *INBI*	*GTNUM5*	*K.SPAR.jdf2.0*	90
Step 7	Factor A and solve for solution x	*SOLVE5*			
Step 9 (optional)	Relative error estimation	*EREST5*			
Step 10 (optional)	Compare x with CSM Testbed solution		*GETSOL*	*STAT.DISP.iset.ncon*	
Step 11 (optional)	Collect statistics	*STATSA*	*STATCS*		

Table 9: The execution path of the subroutine *SPKA*

4.3.2 User Input to the Processor SPK

In our current implementation of the processor SPK, the user-processor communication is accomplished via an external text file. The input requirement and format are reflected by the following code segment of the subroutine *SPKA*

```
C
      SUBROUTINE SPKA ( A, MXSTOR )
C
                 ...
                 ...
                 ...
C
         INDADA = 41
C        --------------------
C        SET MSGLVL AS DESIRED
C        --------------------
         READ ( INDATA, 12 ) MSGLVL
C        --------------------
C        SET MSGLVA AS DESIRED
C        --------------------
         READ ( INDATA, 12 ) MSGLVA
C        ------------------------
C        SET MAXIMUM BUFFER LENGTH
C        ------------------------
         READ ( INDATA, 12 ) BUFMAX
  12     FORMAT( I4 )
C        --------------------------------------------------
C        INPUT NAME OF LIBRARY AND DATASETS FOR GIVEN PROBLEM
C        --------------------------------------------------
         READ ( INDATA, 22 ) LIBNAM
  22     FORMAT( A40 )
         READ ( INDATA, 32 ) JDFSET
         READ ( INDATA, 32 ) KMAP
         READ ( INDATA, 32 ) KSPAR
         READ ( INDATA, 32 ) CON
         READ ( INDATA, 32 ) APPLF
         READ ( INDATA, 32 ) APPLM
         READ ( INDATA, 32 ) STATD
  32     FORMAT( A51 )
                 ....
                 ....
                 ....

         RETURN
C
         END
```

The following comments are in order.

1. As shown in the above code segment, we have designated the logical unit number 41 to be used for the input data file. This choice is made under the restriction that logical unit numbers 1 through 40 should not be used for files other than libraries to avoid possible conflicts with CLIP and GAL [2].

2. The variable *MSGLVA* stands for "message level of SPARSPAK-A". The user may govern the amount of output from SPARSPAK-A by setting *MSGLVA* to the values Table 10.

MSGLVA	amount of output
0	no information is provided.
1	only warnings and errors are printed.
2	warnings, errors and summary are printed.
3	warnings, errors, summary and some statistics are printed.
4	detailed information for debugging purposes.

Table 10: The valid input values of *MSGLVA.*

3. The variable *MSGLVL* allows user to control the amount of output from the interface modules. Given in Table 11 are the input values acceptable for *MSGLVL.*

MSGLVL	amount of output
0,1	no information is provided.
2	warnings, errors and summary are printed.
3	detailed information for debugging purposes.

Table 11: The valid input values of *MSGLVL.*

4. The value of *BUFMAX* should be set to the maximum record length of any data set the processor SPK ever needs to retrieve.

5. The variables initialized by user input are collected into the two labelled common */SPAUSR/* and */CSMUSR/*.

6. An example – To solve the linear system of the test problem demo1 using SPARSPAK-A, edit a file named "fort.41" to contain the following data:

```
2
2
2240
/usr.MC68020/nlal/echu/ns/DEMO/demo1.101
JDF1.BTAB.1.8
KMAP..9.3
K.SPAR.36
CON..1
APPL.FORC.1.1
APPL.MOTI.1.1
STAT.DISP.1.1
```

Note that the path name of the library file "demo1.l01" is installation dependent. The data set names listed above can be identified from the table of contents of the library `demo1.l01` given in Figure 3. Note that the data sets *APPL.FORC.iset.1* and *APPL.MOTI.iset.1* may not both exist, and it is indeed the case for the problem demo1 – one cannot find the name *APPL.MOTI.1.1* listed in the table of contents of its data library. However, as noted above, we have required the user to input both names in order to maintain a uniform format for user input. In this case, the variable *APPLM* is simply a dummy variable, because the subroutine *GTMOTI* will not attempt to access this data set as explained in §4.2.4.

4.3.3 Output from the Processor SPK

1. Output from SPARSPAK-A: The readers are referred to section §7 of the SPARSPAK-A User's Guide [7] for a complete description of the statistics and error messages output.
2. Output from the interface modules:
   (a) Statistics gathering (*STATCS*) – The information contained in Table 12 may be printed by the following statement.

   CALL STATCS

MSGLVL	Information	Variable	Common block
0,1,2,3	Total CSM-time required Maximum CSM-storage required	*CSMTIM* *CSMSTR*	*/CSMDTA/*
2,3	Size of storage array	*MAXCSM*	*/CSMUSR/*
2,3	Number of joints Max degree of freedom per joint Number of equations	*NUMJNT* *MAXDOF* *NEQNS*	*/PRBLEM/*
3	Addresses of arrays	*DOF* *BUF* *MASK* *KC* *ICLQ* *FCON* *SPK*	*/CSMMAP/*

Table 12: Information printed by the subroutine *STATCS*.

   (b) Error messages (*IERR*) – When fatal error is detected, so that the computation cannot proceed, a positive code is assigned to the variable *IERR* in the common block */CSMUSR/*. The names of the modules in which the error occurs, the

numerical error codes, and the corresponding error messages as given in Table 13 may be printed by setting the variable *MSGLVL* to be "2" or a higher number.

MODULE	*IERR*	Error message
SPACE	1001	Insufficient storage. The last stage completed and the required storage are printed
LIBOPN	1011	Cannot open dataset library
	1012	The maximum logical device index = 30. The *LDI* returned exceeds this value.
GETJDF	1013	Incorrect execution sequence.
	1014	Dataset does not have all expected items.
GETDOF	1019	Incorrect execution sequence.
GTZERO	1021	Incorrect execution sequence.
GTCOND	1022	Incorrect execution sequence.
GETIJ	1023	Incorrect execution sequence.
GTFORC	1024	Incorrect execution sequence.
GTMOTI	1025	Incorrect execution sequence.
	1026	Unexpected nonzero constraint value.
	1027	Zero entry for a nonzero constraint occurs.
GTNUM5	1028	Incorrect execution sequence.
QKINFO	2001	*LMFIND*: cannot find dataset.
	2002	*GMGEKA*: record does not exist.
	2003	*GMGECY*: record group key undefined.
	2004	*GMGECY*: segmented record group noted.
	2009	Insufficient buffer space. The required value for the input variable *BUFMAX* is printed
GTRECI	2005	record type mismatch ···
	2006	*GMGETN*: error detected by *LMERCD* ···
GTRECF	2007	record type mismatch ···
	2008	*GMGETN*: error detected by *LMERCD* ···

Table 13: Error messages of the processor SPK.

4.3.4 An Example - Solving the Testbed problem demo1

Input data:

```
2
2
2240
/usr.MC68020/nla1/echu/ns/DEMO/demo1.101
JDF1.BTAB.1.8
KMAP..9.3
K.SPAR.36
CON..1
APPL.FORC.1.1
APPL.MOTI.1.1
STAT.DISP.1.1
```

The following output is produced by the macroprocessor command [xqt SPK:

```
 ** BEGIN SPK **    DATA SPACE=   600000 WORDS
1
     *********** UNIVERSITY OF WATERLOO
     *********** SPARSE MATRIX PACKAGE
     ***********  ( S P A R S P A K )
     ***********      RELEASE   3
     ***********  (C) JANUARY 1984
     *********** ANSI FORTRAN
     *********** DOUBLE PRECISION
     *********** LAST UPDATE JANUARY 1984


          OUTPUT UNIT FOR ERROR MESSAGES           6
          OUTPUT UNIT FOR STATISTICS               6

     LIBOPN- OPEN /usr.MC68020/nla1/echu/ns/DEMO/demo1.101
 <DM> OPEN,  Ldi:  2, File: /usr.MC68020/nla1/echu/ns/DEMO/demo1.101 ,
Attr: rold, Block I/O

     DATASETS TO BE ACCESSED:

          JDF1.BTAB.1.8
          KMAP..9.3
          K.SPAR.36
          CON..1
          APPL.FORC.1.1
          APPL.MOTI.1.1
          STAT.DISP.1.1

     GETJDF - GET NUMBER OF JOINTS AND ...

     GETDOF - GET DEGREES OF FREEDOM ...

     GTZERO - DETECT DUMMY ROWS ...

     GTCOND - GET CONSTRAINTED VARIABLES...

     GTMOTI - GET NONZERO CONSTRAINTS...
```

```
GETIJ - INPUT NONZERO STRUCTURES...

IJBEGN - BEGIN STRUCTURE INPUT ...

INIJ   - INPUT OF ADJACENCY PAIRS ...

IJEND  - END OF STRUCTURE INPUT ...

ORDRB5 - MINIMUM DEGREE ORDERING ...

GTFORC - INPUT RIGHT HAND SIDE...

INBI   - INPUT OF RIGHT HAND SIDE ...

GTNUM5 - GET NONZERO NUMERIC...

INAIJ5 - INPUT OF MATRIX COMPONENTS ...

SOLVE5 - GENERAL SPARSE SOLVE ...

EREST5 - ERROR ESTIMATOR ...

GETSOL - COMPARE WITH TESTBED SOLN ...

     MAX. REL ERR COMPARED TO STAT.DISP.1.1

     IS  0.4824782e-07   IN COMPONENT   26
     CSM SOL =  0.285208672285080+00  WE HAVE  0.285208686045780+00

STATCS - SYSTEM-CSM STATISTICS ...

     SIZE OF STORAGE ARRAY (MAXCSM)          300000

     NUMBER OF JOINTS                             5
     MAX DEGREE OF FREEDOME PER JOINT             6
     NUMBER OF EQUATIONS                         30
     TOTAL CSM-TIME REQUIRED                      3.740
     MAXIMUM CSM-STORAGE REQUIRED              2271.

STATSA - SYSTEM-A STATISTICS ...

     SIZE OF STORAGE ARRAY (MAXSA)           297729
     NUMBER OF EQUATIONS                         30
     NUMBER OF OFF-DIAGONAL NONZEROS            336
     TIME FOR ORDERING                            0.020
     STORAGE FOR ORDERING                       442.
     TIME FOR ALLOCATION                          0.000
     STORAGE FOR ALLOCATION                     308.
     STORAGE FOR SOLUTION                       367.
     TIME FOR FACTORIZATION                       0.040
     TIME FOR SOLUTION                            0.020
     OPERATIONS IN FACTORIZATION                956.
     OPERATIONS IN SOLUTION                     396.
     TIME FOR ESTIMATING RELATIVE ERROR           0.040
     OPERATIONS IN ESTIMATING REL ERROR        1330.
     STORAGE FOR ESTIMATING REL ERROR           397.
     ESTIMATE OF RELATIVE ERROR                 2.088e-08
```

```
         TOTAL TIME REQUIRED                              0.120
         MAXIMUM STORAGE REQUIRED                       442.
EXIT SPK CPUTIME=     4.2 I/O(DIR,BUF)=       0         0
```

5 Numerical Experiments

In this section we report experimental results of several matrix factorization processors we have installed in the CSM Testbed.

5.1 The Specifications of the Test Problems

For all processors the tests are performed on the NICE/SPAR demonstration problems listed in Table 14. The finite element model of CSM focus problem 1 has been refined to generate larger problems focus1, focus2, focus3 and focus4. The five different meshes we have used are given in Table 15.

NICE/SPAR demonstration problems	
p648	CSM focus problem 1 — Buckling of a blade-stiffened panel with a discontinuous stiffener
focus1	p648 with finer mesh I
focus2	p648 with finer mesh II
focus3	p648 with finer mesh III
focus4	p648 with finer mesh IV
demo1	Beam problem
demo2	Vibration of a circular membrane
demo3	Circular plate problems
demo4	Rectangular plate problems
demo6	Cylindrical shell problems
demo7	Buckling of a cylindrical shell due to torsional loading
demo9	Beam problems
demo10	Saturn 5 Launcher Umbilical Tower (LUT)
demo12	Hyperbolic paraboloid static solution
demo13	Composite toroidal shell

Table 14: NICE/SPAR demonstration problems.

Each problem is completely specified by the data sets in Table 16 except that the load set *APPL.FORC.iset.1* and the applied displacement dataset *APPL.MOTI.iset.1* may not both exist. For example, there is no applied force vector for the panel focus problem and there is no applied displacements for the static analysis of the mast problem. The value of *ncon* selects one of possibly more than one constraint cases and the value of *iset* specifies a particular load case of applied force and moments, which is also the load case of the applied motions if there exist nonzero constraints. Corresponding to each pair of (*ncon*, *iset*) there is a unique solution which may be retrieved from the data set *STAT.DISP.iset.ncon* to

User-specified meshes for CSM focus problem 1						
	NRINGS	NSPOKES	NELX	NELE	NELBS	NELS
p648	2	8	3	1	1	1
mesh I	4	16	3	1	1	1
mesh II	2	8	6	2	2	2
mesh III	2	8	12	2	2	2
mesh IV	4	16	6	2	2	2

Table 15: User-specified meshes for CSM focus problem 1.

verify the correctness of an experimental processor. The full names of the data sets can be found in the table of contents of the data library which can be looked up during or after the execution of a particular analysis in the Testbed. As shown in the example given in Table 16, a "0" component in the dataset name can be represented by a null entry. A sample content list of the data library `demo1.l01` was given in §2 of this report, which was produced by the CLAMP directive `*TOC` during the execution of problem demo1. For each test problem, the path name of its data library and the names of the data sets in Table 16 consist of the user input to an experimental processor. Note that the use of * as a component of the dataset name implies a generic wild-card match, hence it should not be used unless the data set with its name matching the remaining components is unique in the data library.

The accessed CSM testbed data sets	
Name	An example $(ncon, iset) = (3,6)$
JDF1.BTAB.1.8	*JDF1.BTAB.**
KMAP.0.nsubs.ksize	*KMAP..**
K.SPAR.jdf2.0	*K.SPAR.**
CON.0.ncon.0	*CON..3*
APPL.FORC.iset.1	*APPL.FORC.6.1*
APPL.MOTI.iset.1	*APPL.MOTI.6.1*
STAT.DISP.iset.ncon	*STAT.DISP.6.3*

Table 16: Data sets accessed by CSM-SPARSPAK interface modules.

The system $Ax = b$ presented to each experimental processor is the upper triangular part of the system stiffness matrix retrieved from the data set *K.SPAR.jdf2.0* subject to the changes necessitated by the way we handle constraints and dummy rows. The modified system has the following characteristics.

1. The coefficient matrix and the right hand side are modified so that each equation corresponding to a constrained variable x_i can be replaced by

$$x_i = c_i \ ,$$

where $c_i \geq 0$ is the specified constraint.

2. The identically zero rows are detected before problem input and the corresponding variables are treated as being constrained to zero.

3. The dimension of the modified coefficient matrix is equal to the product of the number of joints and the degree of freedom per joint in the model. The number of equations of each demonstration problem is given in Table 17 under the column heading "neqns".

In Table 17 we summarize the characteristics of the linear systems retrieved for each demonstration problem. The entries in the column labeled "# nonzeros in *K.SPAR*" are computed from $nsubs \times jdf2$, where we recall that *nsubs* is the total number of submatrices in the block upper triangular part (including the diagonal blocks) of the system stiffness matrix and that $jdf2 = JDF \times JDF$ represents the number of elements in each submatrix. Therefore, the nonzero count here includes the coefficients in the lower triangular part of the diagonal blocks and the coefficients in the dummy rows as well as the rows corresponding to the constrained variables. The actual off-diagonal nonzero elements input to an experimental processor are listed in the last column under the heading of "# off-diag nonz in *A*".

5.2 The Numerical Properties of the Test Problems

5.2.1 The Conditioning of the System Stiffness Matrix

In Table 18 we list the estimated condition number of the system stiffness matrix for each test problem. The condition numbers are provided by SPARSPAK-A and their computation is described in reference [6]. The order of magnitude of the condition numbers indicates that the single-precision solution of these problems my not have significant digits in some components. By comparing the single-precision static displacement solutions obtained from the Testbed processors INV and SSOL for the same problem using different joint orderings, our numerical experiments confirm that the loss of all significant digits can indeed occur in small components of the solution.

5.2.2 The Accuracy of the Computed Solutions

The condition number estimates we presented in Table 18 indicate that in order to have significant digits in all components of the solution to be stored in the data set *STAT.DISP.iset.ncon*,

Characteristics of the linear systems $Ax = b$								
problem	# joints	d.o.f	neqns	# dummy rows	# zero constraints	# nonzero constraints	# nonzeros in *K.SPAR*	# off-diag nonz in *A*
p648	108	6	648	78	98	10	17064	9706
focus1	192	6	1152	154	116	12	31320	18458
focus2	276	6	1656	228	167	17	45792	26824
focus3	480	6	2880	396	167	17	81216	50560
focus4	388	6	2328	332	185	19	65088	38656
demo1	5	6	30	0	6	0	324	168
demo2	101	3	303	101	203	0	4077	342
demo3	101	5	505	0	80	0	11325	7830
demo4	54	5	270	0	55	0	5675	3546
demo6	121	6	726	0	97	0	19476	14151
demo7	132	6	792	0	36	0	22464	18576
demo9	11	6	66	0	6	0	756	474
demo10	372	6	2232	0	24	0	47376	39072
demo12	36	6	216	0	18	0	5256	3978
demo13	337	6	2022	0	96	0	59364	49743

Table 17: Characteristics of NICE/SPAR demonstration systems.

Condition number of the system stiffness matrix	
problem	SPARSPAK-A estimate of condition number
p648	2.2×10^7
focus1	3.7×10^7
focus2	2.2×10^7
focus3	2.0×10^7
focus4	2.6×10^7
demo1	5.8×10^7
demo2	2.2×10^7
demo3	1.7×10^7
demo4	1.8×10^7
demo6	2.0×10^7
demo7	3.2×10^7
demo9	4.8×10^6
demo10	5.0×10^{10}
demo12	5.6×10^9
demo13	1.4×10^7

Table 18: Numerical properties of NICE/SPAR demonstration problems.

the system stiffness matrix should be stored in double-precision and processors INV and SSOL should perform the numerical computation in double-precision. The following information from [1] tells us how to ensure that the computations by each processor are performed with the desired precision.

1. Processor K stores the system stiffness matrix in double precision if the user input parameter SPDP is reset to 2 as shown in a sample script given later in this paragraph.
2. Processor INV calculates the triangular matrix using double precision if the input system stiffness matrix dataset is stored in double precision. However, the factors output by processor INV will be stored in the precision determined by resetting the user-controlled argument SPDP: 1 (default) = single precision, 2 = double precision.
3. Processor SSOL computes the displacement solution vector in double-precision if the factored matrix is stored in double-precision.

Therefore, each reset SPDP in the following script excerpt ensures that the output data set is in double precision, which in turn ensures that the computation by the next processor is performed in double precision.

```
      ...
      ...
[xqt K
      reset SPDP=2
[xqt INV
      reset SPDP=2
[xqt SSOL
      ...
      ...
```

For each demonstration problem, the solution provided by an experimental processor is not expected to be identical to the Testbed solutions due to potentially different amounts of round-off error caused by the following factors.

1. The coefficient matrix of the linear system to be solved by an experimental processor is ordered differently. That is, processors INV and SSOL solve (in double precision)

$$\left(PAP^T\right) P\tilde{x} = Pf \ ,$$

   whereas our experimental processor solves (in double precision)

$$\left(\tilde{P}A\tilde{P}^T\right) \tilde{P}\hat{x} = \tilde{P}f \ .$$

   Since the permutation matrix P is induced by resequencing the joints in the model, it is not the same as the permutation matrix $\tilde{P}$ chosen by SPARSPAK-A for the coefficient matrix in general.

2. Even for the same ordering of A, the factorization algorithms implemented by different processors employ different computation sequence.
3. The system stiffness matrix is ill-conditioned.

However, with the condition number estimates available for each system stiffness matrix, we can estimate the relative error in our solution $\hat{x}$ by the algorithm described in [6] and implemented in SPARSPAK-A. On the other hand, by assuming that the Testbed solution $\tilde{x}$ is the correct solution we can compute the relative error in $\hat{x}$ by

$$\max_{\forall i} \frac{|\tilde{x}_i - \hat{x}_i|}{|\tilde{x}_i|} .$$

We can now verify the correctness of our experimental processors if the relative error computed above is very close to the relative error estimated by SPARSPAK-A with respect to the true (but unknown) solution. We have listed these two quantities in Table 19 for all test problems and we see that they are essentially of the same magnitude or sufficiently close for all problems.

problem	$\max \frac{\lvert\tilde{x}_i - \hat{x}_i\rvert}{\lvert\tilde{x}_i\rvert}$	SPARSPAK-A estimate of the relative error in $\hat{x}$
p648	5.9×10^{-8}	6.9×10^{-9}
focus1	6.4×10^{-8}	2.7×10^{-8}
focus2	5.9×10^{-8}	4.6×10^{-8}
focus3	5.8×10^{-8}	4.9×10^{-8}
focus4	7.3×10^{-8}	3.8×10^{-8}
demo1	4.8×10^{-8}	1.4×10^{-8}
demo2	5.0×10^{-8}	5.4×10^{-9}
demo3	4.7×10^{-7}	2.9×10^{-8}
demo4	5.0×10^{-8}	6.4×10^{-9}
demo6	1.6×10^{-6}	1.2×10^{-8}
demo7	6.2×10^{-8}	1.9×10^{-8}
demo9	2.7×10^{-8}	1.7×10^{-9}
demo10	5.6×10^{-6}	1.6×10^{-5}
demo12	5.7×10^{-8}	4.4×10^{-6}
demo13	5.8×10^{-7}	6.9×10^{-8}

Table 19: Comparing NICE/SPAR solutions $\tilde{x}$ with SPARSPAK-A solutions $\hat{x}$.

5.3 The Experimental Factorization Processors

In this section we briefly describe the three sparse matrix factorization processors we have installed in the CSM Testbed. The three processors employ different methods in solving a sparse symmetric positive definite system

$$Ax = b\,.$$

1. Processor SPK: The method employed by the processor SPK is the direct solver provided by SPARSPAK-A corresponding to the minimum degree ordering algorithm in [26].
2. Processor EXP1: The factorization method employed by the experimental processor EXP1 is the multifrontal method implemented by Liu as described in [23].
3. Processor EXP2: The factorization method employed by the experimental processor EXP2 is the adaptive sparse out-of-core Cholesky scheme recently developed by Liu [22].

Since the factorization methods employed by the processors EXP1 and EXP2 use the same storage scheme as that used by the minimum degree ordering in SPARSPAK-A and they were intended to be used in conjunction with SPARSPAK-A [22,23], the same interface modules for inputing the problem to SPARSPAK-A can be used.

5.4 Numerical Results

We first compare the factorization time of the three experimental processors with that of the processor INV. Since the joint ordering can affect the execution time of processor INV significantly, we have attempted to report the timing results for all available joint elimination sequences. The ordering algorithms currently available in the CSM Testbed are listed in Table 20.

acronym	ordering algorithm
ND	Nested dissection (fill minimizer) [19]
MDG	Minimum degree (fill minimizer) [19,24]
RCM	Reverse Cuthill-Mckee (profile minimizer) [19]
GPS	Gibbs-Poole-Stockmeyer (bandwidth minimizer) [8]
SEQ	Sequential joint elimination sequence (i.e., no reordering of joints)

Table 20: The joint ordering methods employed in the CSM Testbed.

Since the ordering algorithms used by processors EXP1 and EXP2 are the topological orderings of the elimination tree induced by the minimum degree ordering [22,25], we have thus used "MDG*" to represent any one of them. One consequence of the choice of ordering algorithms by the experimental processors is that the amount of fill-in in the Cholesky factor is the same for the three of them. From the factorization times reported in Table 21 we see that the in-core factorization time of processors SPK and EXP1 are significantly smaller than the INV times in most cases as one would expect in view of the I/O conducted by the latter. Except for problem demo7, the saving in execution time ranges from 30% to 58% compared to the fastest INV time. As we have pointed out earlier, the reordering of

Factorization times (in seconds)								
	NICE/SPAR (INV)					SPK	EXP1	EXP2
problem	SEQ	ND	MDG	RCM	GPS	MDG*	MDG*	MDG*
focus1	466	151	106	196	239	44	43	61
focus2		236	145	445		76	76	102
focus3		770	441			313		378
focus		482	288	940		148	146	188
demo6	61	53	53	82	82	33	32	45
demo7	96	124	101	113	112	93	92	113
demo10	60	304	62	203	166	41	40	72
demo13	406					283		337

Table 21: NICE/SPAR (INV) and SPARSPAK-A factorization times.

the joint sequence in the model produces a different permutation matrix from that induced by applying the same ordering algorithm to the coefficient matrix itself. In Table 22 we compare the quality of the minimum degree algorithm when applying to each case, where we give the nonzero counts in the system stiffness matrix A and the computed factors. Due to the different storage schemes employed by the processor INV and SPARSPAK-A, the fill-in is not measured in exactly the same manner as Table 22 indicates.

	Processor INV			SPK, EXP1, EXP2		
problem	ordering	# nonzeros in *K.SPAR*	# nonzeros in *INV.K*	ordering	# off-diag nonz in A	# off-diag nonz in L
focus1	MDG	31320	71040	MDG*	18458	47487
focus2	MDG	45792	99324	MDG*	26824	72519
focus3	MDG	81216	221526	MDG*	50560	184042
focus4	MDG	65088	165480	MDG*	38656	120682
demo7	MDG	22464	62172	MDG*	18576	62829
demo10	MDG	47376	79992	MDG*	39072	71076

Table 22: Comparing the fill-in of different processors.

The performance of processors SPK and EXP1 are essentially the same in terms of execution time. In terms of storage, the in-core multifrontal Cholesky factorization scheme of processor EXP1 requires additional working storage compared with the in-place Cholesky method provided by processor SPK. However, it should be pointed out that the multifrontal method lends itself readily to out-of-core implementation [27], in which case the amount of in-core storage required to perform the entire factorization turns out to be precisely the same as the required working storage for the in-core version. The readers are referred to [27] for various strategies in minimizing the working storage. In [25] the behaviour of the multifrontal method in a paging environment is studied.

In order to compare the out-of-core performance of processor EXP2 with that of processor INV, we should note the following.

1. The number of in-core data reorganizations of the adaptive sparse out-of-core Cholesky scheme [22] is dynamically adjusted to the available memory. In particular, if the declared working space is sufficiently large for the given problem, the entire factorization process will be carried out in-core without reorganizing the data structures. In order to provide a meaningful comparison of the performance of processor EXP2 in execution time as well as storage requirement with that of processor INV, we have run the processor EXP2 with the *minimum* amount of in-core storage that will allow EXP2 to execute. This number can be determined in advance of the actual numerical factorization.
2. The processor EXP2 does I/O using ordinary text files. In particular, the sparse coefficient matrix is saved in a text file and read into memory one column at a time, and the computed Cholesky factor is written to a text file one column at a time. In the current implementation, auxiliary storage is not used to reduce the in-core overhead storage, although it is possible to do so as suggested in [22].
3. We have explained in detail how the processor INV carries out the out-of-core block LDL^T factorization process in § 3.2 of this report. The I/O traffic involved amounts to retrieving the system stiffness matrix from the data set *K.SPAR.** as well as the indexing information from the data set *AMAP..ic2.isize*, and outputting the computed factors to the data set *INV.K.ncon.0*. Because the data are read from or written to the database one record at a time, the number of disk I/O is determined by the record length of each data set. The default record length of these three data sets are listed below in Table 23.

Database interface of processor INV			
Source processor	Reset argument	Dataset name	Default record length
K	LREC	*K.SPAR.**	2240 words
TOPO	LRAMAP	*AMAP..ic2.isize*	1792 words
INV	LRA	*INV.K.ncon.0*	3584 words

Table 23: Database interface of processor INV.

Recall that one record has to accommodate at least the amount of data needed to process one block row of the coefficient matrix. Hence the default record length may not be big enough for larger or denser problems and they can again be changed by resetting the designated argument when executing the source processor of each respective data set. In particular, if necessary, processor TOPO will automatically increase the *AMAP* record length twice up to a maximum size of 2.25×LRAMAP words. The number of records contained in each data set are given under the column heading "Records" in the table of contents of the data library created for each particular analysis.

In summary, the volume of I/O involving each individual data set is roughly the product of the number of records and the record length (strictly speaking, the last record may contain fewer items than being permitted by the specified record length), whereas the number of disk read/write operations is determined by the number of records.

4. The in-core storage required by the processor INV must accommodate one record of each data set in Table 23 in addition to accommodating the maximum number of submatrices involved during the factorization process. Therefore, as suggested in [1], the memory requirement for processor INV may be estimated by the following formula.

$$\text{number of words} = J + L_3 + m\left(L_1 + L_2 + n^2 I_s\right) ,$$

where

J = the number of joints in the structure.

L_1 = record length of input dataset *K.SPAR.jdf2*.

L_2 = record length of *INV.K.ncon* dataset.

L_3 = record length of *AMAP..ic2.isize* dataset.

m = 1 for single precision; 2 for double precision.

n = maximum number of degrees of freedom per joint (default 3, 4, 5, or 6).

I_s = the maximum number of submatrices in use during any one stage of the factorization process. Its value can be obtained from the processor TOPO output parameter SIZE INDEX or from the value of *isize* from *AMAP..ic2.isize*.

It was suggested in [1] that this formula may be used to estimate the amount of space in blank common required by processor INV. If the number of words required is larger than the dimension of blank common, the blank common dimension must be increased and the testbed must be recompiled.

In Table 24, we compare the factorization time and the memory requirement of processor INV with that of the experimental processor EXP2. For each problem, we give the number of nonzero elements in the Cholesky factor computed by SPARSPAK-A (recall that the amount of fill-in is the same for all three experimental processors) under the column heading

"NOFNZ". The ratio of core requirement to the size of the computed Cholesky factor is computed for each problem and displayed for both processors. Note that the quantity of $n^2 I_s$ we use in measuring the memory requirement of processor INV is an underestimate as explained above. We use "LNZSZE" to indicate the maximum number of nonzeros which have to be present in-core for the adaptive sparse Cholesky factorization process to be successfully executed. The results in Table 24 indicate that the processor EXP2 can be quite competitive in both time and space.

		INV		EXP2	
problem	NOFNZ	$\frac{n^2 I_s}{NOFNZ}$	Time (sec)	$\frac{LNZSZE}{NOFNZ}$	Time (sec)
focus1	47487	61%	107	35%	61
focus2	72519	47%	147	31%	102
focus3	184042	36%	449	32%	378
focus4	120682	46%	288	31%	188
demo6	28302	62%	53	44%	45
demo7	62829	46%	108	45%	113
demo10	71076	39%	62	5%	72
demo13	180315	6%	406	25%	337

Table 24: Comparing two out-of-core factorization processors.

Comparing the factorization algorithm of processor EXP2 (adaptive out-of-core Cholesky) with that of processor SPK, we see that the difference in their execution time can be accounted for in the following three aspects.

1. The time spent in data structure reorganization.
2. The time for reading in the coefficient matrix A column by column.
3. The time for writing out the computed Cholesky factor L column by column.

The timing results reported in Table 24 are those with the minimum amount of memory and maximum number of data structure reorganizations. Since the frequency of data structure reorganizations can be reduced by providing more memory, there is a potential tradeoff between time and space. However, the timing results in Table 25 indicate that the time spent in this regard is too small to justify the more significant increase in storage. We can thus conclude that the I/O time can be considered to be the sole factor in determining the speed of processor EXP2.

Since the multifrontal Cholesky method is also a good candidate for out-of-core implementation, and we pointed out earlier that the "working storage" required in its in-core version is precisely what is needed to be in-core in its out-of-core version, it makes sense to

		EXP2			EXP2		
problem	NOFNZ	LNZSZE	# REORGZ	Time	LNZSZE	# REORGZ	Time
focus1	47487	28872	2	59 sec	16823	29	61 sec
focus2	72519	34020	3	99 sec	22647	31	102 sec
demo6	28302	17676	3	44 sec	12394	34	45 sec
demo10	71076	27900	2	69 sec	3330	169	72 sec

Table 25: Data structure reorganization and factorization time.

evaluate its out-of-core potential by comparing its minimum working storage requirement with the memory requirement of processor EXP2. The results we present in Table 26 indicate that the two are quite comparable as far as the test problems are concerned.

		Multifrontal (EXP1)	Column-Cholesky (EXP2)
problem	NOFNZ	LNZSZE/NOFNZ	LNZSZE/NOFNZ
focus1	47487	34%	35%
focus2	72519	31%	31%
focus3	184042	36%	32%
focus4	120682	28%	31%
demo6	28302	44%	44%
demo7	62829	56%	45%
demo10	71076	6%	5%
demo13	180315	21%	25%

Table 26: Comparing processor EXP1 with EXP2

For completeness, we provide in Table 27 the timing results of three other processors which are also essential in solving the linear system arising from a testbed problem, namely TOPO, K and SSOL.

Finally, we provide in Table 28 the total time in executing the processor SPK in the Testbed and indicate separately the time attributed to the numerical factorization phase and the triangular solution phase. The SPK time thus includes the time for retrieving data from the global database and setting up the problem for SPARSPAK-A solver.

In summary, our preliminary findings indicate that there are alternative sparse matrix techniques which are suitable for more general applications and appear to be also competitive in execution time and storage usage compared to the techniques currently employed in the CSM Testbed.

problem	TOPO	K	INV	SSOL	Total
focus1	5	23	107	13	148 sec
focus2	6	34	147	17	204 sec
focus3	16	61	449	33	559 sec
focus4	10	49	288	24	371 sec
demo6	3	11	53	9	76 sec
demo7	4	13	108	12	137 sec
demo10	5	18	62	16	101 sec
demo13	11	47	406	48	512 sec

Table 27: Timing results of TOPO, K, INV, SSOL.

problem	fact	soln	SPK
focus1	44 sec	2 sec	65 sec
focus2	76 sec	4 sec	107 sec
focus3	313 sec	9 sec	376 sec
focus4	148 sec	6 sec	194 sec
demo6	33 sec	2 sec	48 sec
demo7	93 sec	3 sec	113 sec
demo10	41 sec	4 sec	73 sec
demo13	283 sec	9 sec	331 sec

Table 28: Execution time of the processor SPK.

A Installing the Processor SPK

The processor SPK consists of a subset of SPARSPAK-A [7] modules and a set of subroutines which provide an interface between SPARSPAK-A and the global database of the CSM testbed. All of the subroutines are provided as a single directory SPARSE on a UNIX tar tape. The Fortran source for the package is distributed among a number of subdirectories. There are "make" files provided, so that the person installing the package needs only to execute a few commands to compile the package and create the run-time library.

It is advisable to read "§4 Developing New Matrix Factorization Processors" of this report before beginning installation of the package. Since the SPARSE package is used in conjunction with the CSM testbed, we assume in the sequel that the NICE/SPAR processors have been properly installed in the directory `/usr/ns/nice` and `/usr/ns/spar`, and that the SPARSE package is to be installed in the directory `/usr/ns/sparse`. The hierarchy of the directory `/usr/ns` and the files relevant to the installation and use of the SPARSE package are depicted in Figure 30.

The steps to install the SPARSE package are as follows.

1. *Create a directory for* SPARSE:

   ```
   cd /usr/ns
   mkdir sparse
   cd sparse
   ```

2. *Copy the files from tape to disk:* Put the tape in the tape drive and tar the files to the new disk directory:

   tar xvf /dev/*device*

   where *device* should be the appropriate name of the tape drive on your machine. Do an "ls" to make sure that three directories (`install`, `csm-intrface` and `spk-subset`) have been copied from the tape.

3. *Edit the installation-dependent subroutines:* The package has installation-dependent subroutines *SPK, CTIME, SPKCSM*, *DTIME* and *SPRSPK* which provide timing information to the package and set some installation-dependent parameters. In appendix §B, we provide a set of examples for these subroutines. The sample programs are written for a SUN/3 workstation running the UNIX operating system at the University of Tennessee Knoxville. Comments in these listings indicate changes which may be necessary. The subroutine *SPK* is contained in the directory `csm-intrface/driver`, the subroutines *CTIME* and *SPKCSM* are contained in the directory `csm-intrface/system`, and *DTIME* and *SPRSPK* are contained in the directory `spk-subset/system`. Samples of subroutines required by *CTIME, DTIME* and *SPRSPK* can be found in the directory `spk-subset/local`; these subroutines are appropriate for machines running Berkeley 4.2 or 4.3 UNIX and their derivatives such as SUN 05.

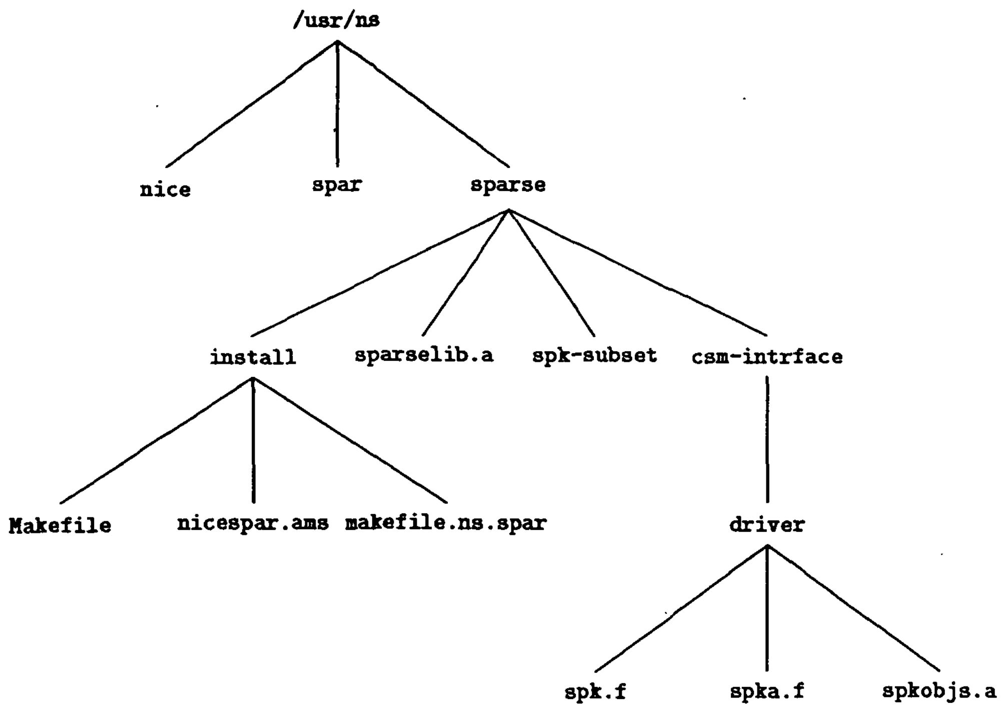


Figure 30: The file system of the directory /usr/ns.

4. *Edit the make file* `/usr/ns/sparse/install/Makefile`: Compilation of the package is performed using a collection of UNIX make files. The most important make file is called `Makefile` found in the directory `install`; it will invoke the other make files. The distributed make files assume that the package is running on a SUN workstation. There are comments in `Makefile` to help you make the appropriate changes to it for your installation. There is no need to change the make files in any other `sparse` directories.

5. *Create and install the compiled library:* After making the required changes to `Makefile`, you are ready to create and install the compiled library. Execute the following commands.

   ```
   cd /usr/ns/sparse/install
   make install
   ```

   A compiled library `sparselib.a` will be created in the directory `sparse`.

6. *Install a new processor in the testbed:* Since the SPARSE package is installed as a processor `SPK` in the testbed and a CSM processor is a subroutine called by the NICE/SPAR main program, it is necessary to compile the `SPK` driver routines in the directory `/usr/ns/sparse/csm-intrface/driver` and edit the main program master file `nicespar.ams` in the directory `/usr/ns/spar`. The object code of the `SPK` driver routines `spk.f` and `spka.f` is contained in a separate library called `spkobjs.a` in the `driver` directory so that it may be updated independent of `sparselib.a`. In addition, the makefile in the directory `/usr/ns/spar` must be edited so that the two libraries can be linked to the executable when it is created. A copy of the properly edited `nicespar.ams` and a copy of the edited makefile can be found in the directory `install`. The former has the file name `nicespar.ams` and the latter has the file name `makefile.ns.spar`. With these two files available, the following commands may be executed to install the new processor `SPK` in the testbed. Note that you must have `write` permission in the directory `spar` to do this.

   ```
   cd /usr/ns/sparse/install
   make spk
   cd ../../spar
   mv makefile  makefile.old
   mv nicespar.ams   nicespar.ams.old
   cp ../sparse/install/makefile.ns.spar   makefile
   cp ../sparse/install/nicespar.ams       nicespar.ams
   make
   ```

7. *When the file* `korcoma.inc` *is changed:* Since the include file `korcoma.inc` in the

directory **spar** declares the size of the in-core storage available for every SPAR processor, the driver source code **spk.f** of processor SPK must contain the line

```
include '/usr/ns/spar/korcoma.inc'
```

and it must be recompiled each time the declared size is changed. Since the dependence of **spk.o** on **korcoma.inc** is specified in the appropriate make file, the following commands will not only detect whether the declaration file **korcoma.inc** has been modified since **spk.o** was last created but also recompile **spk.f** and update the library **spkobjs.a** if that is the case. Finally the executable in the directory **spar** is recreated to link to the modified **spkobjs.a** after the "make" command in the last line is executed.

```
cd /usr/ns/sparse/install
make spk
cd ../../spar
make
```

8. *Recover space used by intermediate files:* If the system on which you are running is short of disk space, a substantial amount of space used during the installation of SPARSE can be recovered by deleting the ".o" files and other intermediate files generated during the creation of the library. To do this, execute the following commands.

```
cd /usr/ns/sparse/install
make clean
```

If for some reason you must later re-create some or all of the library **sparselib.a**, these intermediate files will have to be regenerated, at considerable cost in computer time. Thus, it is advisable to execute "make clean" only if you really need the space.

B Installation-dependent Subroutines

```
C*****************************************************************************
C*****************************************************************************
C                     SPK ..... A NEW CSM PROCESSOR
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - THIS IS THE DRIVER FOR INSTALLING INTO NICE/SPAR
C       OUR INTERFACE MODULES AS A SINGLE PROCESSOR WHICH
C       SOLVES CSM TESTBED PROBLEMS USING SPARSPAK-A MODULES.
C
C    THE NEW PROCESSOR IS CODED AND INSTALLED INTO NICE/SPAR DIRECTLY
C    FOLLOWING THE GUIDELINES GIVEN IN NASA TECHNICAL MEMORANDOM
C    89096, NAMELY
C
C    (a) THE NAME OF THE PROCESSOR SHOULD BE NO LONGER THAN FOUR
C        CHARACTERS.
C    (b) THE PROCESSOR SHOULD BE WRITTEN AS A FORTRAN 77 SUBROUTINE
C        WHOSE NAME IS THE PROCESSOR NAME.
C    (c) THE SUBROUTINE SHOULD HAVE NO ARGUMENTS.
C    (d) THE PROCESSOR SHOULD BEGIN EXECUTION WITH A CALL TO THE
C        LIBRARY SUBROUTINE "INTRO" WITH THE PROCESSOR NAME
C        AS THE ONLY ARGUMENT.  THE GIVEN NAME IS USED BY THE
C        "GAL" DATA MANAGER AS THE CREATING PROCESSOR FOR
C        NEW DATASETS INSERTED IN "GAL" LIBRARIES; IT ALSO
C        APPEARS IN THE INTERACTIVE PROMPT STRING IF THE
C        "SPAR READER" ROUTINE IS USED FOR INPUT COMMAND
C        PROCESSING.
C    (e) THE LABELED COMMON BLOCK /IANDO/ WITH 2 INTEGER VARIABLES
C        CONTAINING USER INPUT AND OUTPUT UNIT NUMBERS SHOULD BE
C        INCLUDED IN APPROPRIATE MODULES.  THE UNIT NUMBERS ARE
C        ASSIGNED IN THE SUBROUTINE "INTRO".
C    (f) CALL LIBRARY SUBROUTINE "FIN" TO CLOSE "GAL" LIBRARIES.
C
C           ************************************************************
C                   W   A   R   N   I   N   G
C           ************************************************************
C           THE PATH NAME OF THE INCLUDE FILE "korcoma.inc"
C           IS INSTALLATION DEPENDENT.
C
C
C*****************************************************************************
C
      SUBROUTINE SPK
C
C        ----------------------------------------------
C        INCLUDE DECLARATION CONTAINING BLANK COMMON VARIABLES AND
C        DIMENSIONS:
C        PARAMETER (KSZZZ= 200000)
C        COMMON KORE, KEVEN, KORT, A(KSZZZ)
C        ----------------------------------------------
C
      include '/usr.MC68020/nla1/echu/ns/spar/korcoma.inc'
C
      INTEGER    MXSTORE
C
C        ----------------------------------------------
C        IDENTIFY PROCESSOR TO CSM ARCHITECTURE
C        ----------------------------------------------
      CALL INTRO ( 'SPK' )
C        ----------------------------------------------
C        WORKING STORAGE A IS DECLARED AS KSZZZ WORDS WHICH IS
C        EQUIVALENT TO HALF THAT MANY DOUBLE-PRECISION FLOATING
C        POINT NUMBERS.
C        ----------------------------------------------
      MXSTOR = KSZZZ/2
      CALL SPKA ( A, MXSTOR )
      CALL FIN ( 0, 0 )
      CALL EXIT
      END
```

```
C***************************************************************************
C***************************************************************************
C              CTIME ..... ELAPSED PROCESSOR TIME
C***************************************************************************
C***************************************************************************
C
C    PURPOSE - CTIME RETURNS THE ELAPSED PROCESSOR TIME SINCE
C       IT WAS LAST CALLED.  IT USES THE COMMON VARIABLE TIME
C       TO REMEMBER THE TIME WHEN CTIME WAS LAST CALLED.
C
C    ***********************************************************************
C              W   A   R   N   I   N   G
C    ***********************************************************************
C       THIS IS AN INSTALLATION DEPENDENT ROUTINE. IT
C       SHOULD BE SET UP BY THE INSTALLER OF THE PACKAGE.
C       IN THIS EXAMPLE, ROUTINE GTIMER IS THE TIMER ROUTINE
C       THAT RETURNS THE CURRENT PROCESSOR TIME ON A SUN/3
C       WORKSTATION RUNNING THE UNIX OPERATING SYSTEM AT THE
C       UNIVERSITY OF TENNESSEE KNOXVILLE.
C    ***********************************************************************
C
C    INPUT PARAMETER -
C       IDUMMY - A DUMMY INTEGER VARIABLE.
C
C    PROGRAM SUBROUTINE -
C       GTIMER.
C
C***************************************************************************
C
      REAL FUNCTION  CTIME ( IDUMMY )
C
C***************************************************************************
C
        INTEGER   IDUMMY, IPRNTE, IPRNTS, MAXINT
        REAL      RATIOL, RATIOS, TIME  , X
C
C***************************************************************************
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL,
     1                 TIME
C
C***************************************************************************
C
        CALL  GTIMER ( X )
        CTIME = X - TIME
        TIME = X
        RETURN
C
      END
```

```
C***********************************************************************
C***********************************************************************
C         SPKCSM ..... INITIALIZE PARAMETERS
C***********************************************************************
C***********************************************************************
C
C    PURPOSE - TO SET SYSTEM PARAMETERS AND ASSIGN DEFAULT
C      VALUES TO SOME USER PARAMETERS.  IT IS A MACHINE
C      DEPENDENT ROUTINE.  THIS ROUTINE HAS TO BE CALLED
C      BEFORE ANY OTHER PACKAGE MODULE.
C
C    PARAMETERS INITIALIZED -
C      IPRNTE - THE OUTPUT UNIT NUMBER FOR ERROR MESSAGES.
C      IPRNTS - THE OUTPUT UNIT NUMBER FOR STATISTICS.
C      RATIOL - THE RATIO OF THE NUMBER OF BITS IN A FLOATING
C            POINT VARIABLE TO THAT IN A LONG INTEGER
C            VARIABLE.  FOR EXAMPLE, IF FLOATING POINT
C            NUMBERS OCCUPY TWICE AS MANY BITS AS LONG
C            INTEGERS, RATIOL SHOULD BE SET TO 2.
C      RATIOS - THE RATIO OF THE NUMBER OF BITS IN A FLOATING
C            POINT VARIABLE TO THAT IN A SHORT INTEGER
C            VARIABLE.
C      MAXINT - THE LARGEST POSITIVE INTEGER THAT CAN BE
C            STORED IN A SHORT INTEGER VARIABLE.
C      TIME   - VARIABLE USED BY THE TIMER ROUTINE CTIME.
C              SEE REMARK
C      STAGE  - STARTING STAGE OF SYSTEM-CSM.
C
C     REMARK - THIS INTERFACE PACKAGE ASSUMES THE EXISTENCE OF
C          A REAL TIME FUNCTION CTIME WHICH RETURNS THE ELAPSED
C          PROCESSOR TIME SINCE IT WAS LAST CALLED.  WITH THE
C          COMMON VARIABLE TIME, THE INSTALLER OF THE PACKAGE
C          SHOULD BE ABLE TO WRITE SUCH A FUNCTION, USING THE
C          INSTALLATION TIMER.
C
C
C***********************************************************************
C
      SUBROUTINE SPKCSM
C
C***********************************************************************
C
        CHARACTER*40 LIBNAM
        CHARACTER*51 CDUMMY(7)
        INTEGER*4 IIN, IOUTX
        INTEGER*4 IPRNTE, IPRNTS, MAXINT
        INTEGER*4 BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4 MSGLVL, IERR  , MAXCSM
        INTEGER*4 IIN, IOUTX
        REAL      RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM, CDUMMY(7)
        COMMON /IANDO/  IIN, IOUTX
C
C     *******************************************************************
C          W   A   R   N   I   N   G
C     *******************************************************************
C
C      THE FOLLOWING 4 LINES OF CODE ARE INSTALLATION
C      DEPENDENT.  THEY MAY HAVE TO BE MODIFIED BY THE
C      PERSON INSTALLING THIS PACKAGE.
C
C      OUR CURRENT ENVIRONMENT -
C        - RATIOL AND RATIOS ARE BOTH 2.
C        - MAXINT = 2**15 - 1 = 32767
C
C      -----------------
C      INSTALLATION DEPENDENT PARAMETERS
C      -----------------
        TIME = 0.0
C
        RATIOL = 2.0
        RATIOS = 2.0
C
        MAXINT = 32767
```

```
C
C     ------------------------------------
C     IPRNTE AND IPRNTS ARE BOTH SET TO THE WRITER UNIT
C     NUMBER ASSIGNED TO IOUTX WHEN THE NEW PROCESSOR
C     IS IDENTIFIED TO THE CSM-ARCHITECTURE.
C     ------------------------------------
      IPRNTE = IOUTX
      IPRNTS = IOUTX
C
C     ------------------------------------
C     INITIALIZING THE EXECUTION STAGE FOR THE INTERFACE ...
C     ------------------------------------
      STAGE  = 0
C
      RETURN
C
    END
```

```
C— SPARSPAK-A (ANSI FORTRAN) RELEASE III — NAME = DTIME
C (C) UNIVERSITY OF WATERLOO   JANUARY 1984
C*****************************************************************************
C*****************************************************************************
C              DTIME ..... DELTA TIME
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - DTIME RETURNS THE ELAPSED PROCESSOR TIME SINCE
C       IT WAS LAST CALLED.  IT USES THE COMMON VARIABLE TIME
C       TO REMEMBER THE TIME WHEN DTIME WAS LAST CALLED.
C
C     **********************************************************************
C            W   A   R   N   I   N   G
C     **********************************************************************
C       THIS IS AN INSTALLATION DEPENDENT ROUTINE. IT
C       SHOULD BE SET UP BY THE INSTALLER OF THE PACKAGE.
C       IN THIS EXAMPLE, ROUTINE GTIMER IS THE TIMER ROUTINE
C       THAT RETURNS THE CURRENT PROCESSOR TIME ON A SUN/3
C       WORKSTATION RUNNING THE UNIX OPERATING SYSTEM AT THE
C       UNIVERSITY OF TENNESSEE KNOXVILLE.
C     **********************************************************************
C
C     INPUT PARAMETER -
C       IDUMMY - A DUMMY INTEGER VARIABLE.
C
C     PROGRAM SUBROUTINE -
C       GTIMER.
C
C*****************************************************************************
C
      REAL FUNCTION  DTIME ( IDUMMY )
C
C*****************************************************************************
C
        INTEGER    IDUMMY, IPRNTE, IPRNTS, MAXINT
        REAL       MCHEPS, RATIOL, RATIOS, TIME  , X
C
C*****************************************************************************
C
        COMMON /SPKSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL,
     1                  MCHEPS, TIME
C
C*****************************************************************************
C
        CALL  GTIMER ( X )
        DTIME = X - TIME
        TIME = X
        RETURN
C
      END
```

```
C— SPARSPAK-A (ANSI FORTRAN) RELEASE III — NAME = SPRSPK
C (C) UNIVERSITY OF WATERLOO   JANUARY 1984
C***************************************************************************
C***************************************************************************
C            SPRSPK ..... START SPARSPAK-A
C***************************************************************************
C***************************************************************************
C
C   PURPOSE - TO SET SYSTEM PARAMETERS AND ASSIGN DEFAULT
C      VALUES TO SOME USER PARAMETERS.  IT IS A MACHINE
C      DEPENDENT ROUTINE.  THIS ROUTINE HAS TO BE CALLED
C      BEFORE ANY OTHER PACKAGE MODULE.
C
C   PARAMETERS INITIALIZED -
C      IPRNTE - THE OUTPUT UNIT NUMBER FOR ERROR MESSAGES.
C      IPRNTS - THE OUTPUT UNIT NUMBER FOR STATISTICS.
C      RATIOL - THE RATIO OF THE NUMBER OF BITS IN A FLOATING
C             POINT VARIABLE TO THAT IN A LONG INTEGER
C             VARIABLE.  FOR EXAMPLE, IF FLOATING POINT
C             NUMBERS OCCUPY TWICE AS MANY BITS AS LONG
C             INTEGERS, RATIOL SHOULD BE SET TO 2.
C      RATIOS - THE RATIO OF THE NUMBER OF BITS IN A FLOATING
C             POINT VARIABLE TO THAT IN A SHORT INTEGER
C             VARIABLE.
C      MAXINT - THE LARGEST POSITIVE INTEGER THAT CAN BE
C             STORED IN A SHORT INTEGER VARIABLE.
C      MCHEPS - THE MACHINE EPSILON (UNIT ROUNDOFF ERROR).
C      TIME   - VARIABLE USED BY THE TIMER ROUTINE DTIME.
C             SEE REMARK.
C      STAGEA - STAGE VARIABLE FOR SYSTEM-A.
C
C   REMARK - THIS PACKAGE ASSUMES THE EXISTENCE OF A REAL TIME
C      FUNCTION DTIME WHICH RETURNS THE ELAPSED PROCESSOR TIME
C      SINCE IT WAS LAST CALLED.  WITH THE COMMON VARIABLE
C      TIME, THE INSTALLER OF THE PACKAGE SHOULD BE ABLE TO
C      WRITE SUCH A FUNCTION, USING THE INSTALLATION TIMER.
C
C   PROGRAM SUBROUTINES -
C      ALLOW , STIMER.
C
C***************************************************************************
C
      SUBROUTINE  SPRSPK
C
C***************************************************************************
C
        INTEGER    ICPADA, ICPADB, IERRA , IERRB , IPRNTE,
     1             IPRNTS, MAXINT, MAXSA , MAXSB , MCOLS ,
     1             MDCONS, MDEQNS, MSCONS, MSEQNS, MSGLVA,
     1             MSGLVB, NVARS , STAGEA, STAGEB
        INTEGER    IIN, IOUTX
        REAL       MCHEPS, RATIOL, RATIOS, TIME
        DOUBLE PRECISION  EPS   , EPS1
C
C***************************************************************************
C
        COMMON /SPKSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL,
     1                 MCHEPS, TIME
        COMMON /SPAUSR/ MSGLVA, IERRA , MAXSA , NVARS
        COMMON /SPACON/ STAGEA, ICPADA(49)
        COMMON /SPBUSR/ MSGLVB, IERRB , MAXSB , MCOLS , MSEQNS,
     1                 MDEQNS, MSCONS, MDCONS
        COMMON /SPBCON/ STAGEB, ICPADB(49)
C
        COMMON /IANDO/  IIN, IOUTX
C
```

```
C*****************************************************************************
C
C      ***********************************************************************
C             W  A  R  N  I  N  G
C      ***********************************************************************
C      THE FOLLOWING 6 LINES OF CODE ARE INSTALLATION
C      DEPENDENT.  THEY MAY HAVE TO BE MODIFIED BY THE
C      PERSON INSTALLING THIS PACKAGE.
C
C      ON A SUN/3 WORKSTATION AT THE UNIVERSITY OF TENNESEE KNOXVILLE:
C        - STIMER IS THE ROUTINE TO START THE TIMER,
C        - ALLOW IS THE ROUTINE TO ALLOW FOR A NUMBER OF
C          ARITHMETIC UNDERFLOWS BEFORE SYSTEM ABORTS.
C        - RATIOL AND RATIOS ARE 2 AND 4 RESPECTIVELY.
C        - MAXINT = 2**15 - 1 = 32767
C
      TIME = 0.0
      CALL  STIMER
      CALL  ALLOW ( 1234567 )
C
      RATIOL = 2.0
      RATIOS = 2.0
C
      MAXINT = 32767
C
C      ------------------------------
C      IPRNTE AND IPRNTS ARE BOTH SET TO THE WRITER UNIT
C      NUMBER ASSIGNED TO IOUTX WHEN THE NEW PROCESSOR
C      IS IDENTIFIED TO THE CSM-ARCHITECTURE.
C      ------------------------------
      IPRNTE = IOUTX
      IPRNTS = IOUTX
C
C      ------------------
C      COMPUTE THE MACHINE EPSILON.
C      ------------------
      EPS = 1.0D0
 100   CONTINUE
         EPS = EPS/2.0D0
         EPS1 = 1.0D0 + EPS
         IF  ( EPS1 .GT. 1.0D0 )  GO TO 100
      MCHEPS = EPS*2.0D0
C
      WRITE (IPRNTS,11)
  11   FORMAT ( 1H1
   1         /5X, 40H********** UNIVERSITY OF WATERLOO
   1         /5X, 40H********** SPARSE MATRIX PACKAGE
   1         /5X, 40H********** ( S P A R S P A K )
   1         /5X, 40H**********     RELEASE  3
   1         /5X, 40H********** (C) JANUARY 1984             )
C
      WRITE (IPRNTS,22)
  22   FORMAT ( 5X, 40H********** ANSI FORTRAN                  )
C
      WRITE (IPRNTS,33)
  33   FORMAT ( 5X, 40H********** DOUBLE PRECISION              )
C
      WRITE (IPRNTS,44)
  44   FORMAT ( 5X, 40H********** LAST UPDATE JANUARY 1984      )
C
      WRITE (IPRNTS,55)  IPRNTE, IPRNTS
  55   FORMAT (//10X, 35HOUTPUT UNIT FOR ERROR MESSAGES      , I7
   1          /10X, 35HOUTPUT UNIT FOR STATISTICS          , I7 )
C
C      ---------------------
C      INITIALIZING USER VARIABLES FOR SYSTEM-A ...
C      ---------------------
      STAGEA = 0
C
      RETURN
C
    END
```

C Listing of Programs

```
C*****************************************************************************
C*****************************************************************************
C                    SPK ..... A NEW CSM PROCESSOR
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - THIS IS THE DRIVER FOR INSTALLING INTO NICE/SPAR
C       OUR INTERFACE MODULES AS A SINGLE PROCESSOR WHICH
C       SOLVES CSM TESTBED PROBLEMS USING SPARSPAK-A MODULES.
C
C     THE NEW PROCESSOR IS CODED AND INSTALLED INTO NICE/SPAR DIRECTLY
C     FOLLOWING THE GUIDELINES GIVEN IN NASA TECHNICAL MEMORANDOM
C     89096, NAMELY
C
C     (a) THE NAME OF THE PROCESSOR SHOULD BE NO LONGER THAN FOUR
C        CHARACTERS.
C     (b) THE PROCESSOR SHOULD BE WRITTEN AS A FORTRAN 77 SUBROUTINE
C        WHOSE NAME IS THE PROCESSOR NAME.
C     (c) THE SUBROUTINE SHOULD HAVE NO ARGUMENTS.
C     (d) THE PROCESSOR SHOULD BEGIN EXECUTION WITH A CALL TO THE
C        LIBRARY SUBROUTINE "INTRO" WITH THE PROCESSOR NAME
C        AS THE ONLY ARGUMENT.  THE GIVEN NAME IS USED BY THE
C        "GAL" DATA MANAGER AS THE CREATING PROCESSOR FOR
C        NEW DATASETS INSERTED IN "GAL" LIBRARIES; IT ALSO
C        APPEARS IN THE INTERACTIVE PROMPT STRING IF THE
C        "SPAR READER" ROUTINE IS USED FOR INPUT COMMAND
C        PROCESSING.
C     (e) THE LABELED COMMON BLOCK /IANDO/ WITH 2 INTEGER VARIABLES
C        CONTAINING USER INPUT AND OUTPUT UNIT NUMBERS SHOULD BE
C        INCLUDED IN APPROPRIATE MODULES.  THE UNIT NUMBERS ARE
C        ASSIGNED IN THE SUBROUTINE "INTRO".
C     (f) CALL LIBRARY SUBROUTINE "FIN" TO CLOSE "GAL" LIBRARIES.
C
C          ************************************************************
C                W    A    R    N    I    N    G
C          ************************************************************
C          THE PATH NAME OF THE INCLUDE FILE "korcoma.inc"
C          IS INSTALLATION DEPENDENT.
C
C
C*****************************************************************************
C
      SUBROUTINE SPK
C
C       ----------------------------------------
C       INCLUDE DECLARATION CONTAINING BLANK COMMON VARIABLES AND
C       DIMENSIONS:
C       PARAMETER (KSZZZ= 200000)
C       COMMON KORE, KEVEN, KORT, A(KSZZZ)
C       ----------------------------------------
C
      include '/usr.MC68020/nla1/echu/ns/spar/korcoma.inc'
C
      INTEGER   MXSTORE
C
C       ----------------------------------------
C       IDENTIFY PROCESSOR TO CSM ARCHITECTURE
C       ----------------------------------------
      CALL INTRO ( 'SPK' )
C       ----------------------------------------
C       WORKING STORAGE A IS DECLARED AS KSZZZ WORDS WHICH IS
C       EQUIVALENT TO HALF THAT MANY DOUBLE-PRECISION FLOATING
C       POINT NUMBERS.
C       ----------------------------------------
      MXSTOR = KSZZZ/2
      CALL SPKA ( A, MXSTOR )
      CALL FIN ( 0, 0 )
      CALL EXIT
      END
```

```
C***************************************************************************
C***************************************************************************
C         SPKA ..... A DRIVER FOR INTERFACE MODULES AND SPARSPAK-A
C***************************************************************************
C***************************************************************************
C
C     PURPOSE - THIS IS THE DRIVER CALLING INTERFACE MODULES TO
C        SOLVE CSM TESTBED PROBLEMS USING SPARSPAK-A MODULES.
C
C     INPUT PARAMETERS -
C        A - AN ARRAY OF MXSTOR DOUBLE-PRECISION FLOATING POINT
C           NUMBERS.
C        MXSTOR - SIZE OF ARRAY A IN DOUBLE-PRECISION FLOATING-POINT
C              NUMBERS.
C
C     USER INPUT -
C        MSGLVL - MESSAGE LEVEL FOR INTERFACE MODULES.
C        MSGLVA - MESSAGE LEVEL FOR SPARSPAK-A MODULES.
C        BUFMAX - MAXIMUM BUFFER LENGTH ANTICIPATED.
C        LIBNAM - NAME OF THE DATA LIBRARY.
C        JDFSET - NAME OF DATASET JDF1.BTAB.1.8
C        KMAP -   NAME OF DATASET KMAP.0.nsubs.ksize
C        KSPAR -  NAME OF DATASET K.SPAR.jdf2.0
C        CON -    NAME OF DATASET CON.0.ncon.0
C        APPLF -  NAME OF DATASET APPL.FORC.iset.1
C        APPLM -  NAME OF DATASET APPL.MOTI.iset.1
C        STATD -  NAME OF DATASET STAT.DISP.iset.ncon
C
C     INTERFACE MODULES -
C        SPKCSM, LIBOPN, CTIME, SPACE , GETJDF, GETDOF, GTZERO, GTCOND,
C        GTMOTI, GETIJ , GTFORC, GTNUM5, STATCS, GETSOL.
C
C     SPARSPAK-A INTERFACE MODULES -
C        SPRSPK, ORDRB5, SOLVE5, EREST5, STATSA.
C
C     LOGICAL READER UNIT NUMBER FOR USER INPUT - 41
C
C***************************************************************************
C
      SUBROUTINE SPKA ( A, MXSTOR )
C
      DOUBLE PRECISION   A(1)
      INTEGER            MXSTOR
C
C-------------------------------------------------
C
        CHARACTER*40   LIBNAM
        CHARACTER*51   JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        INTEGER*4      IPRNTE, IPRNTS, MAXINT
        INTEGER*4      MSGLVL , IERR  , MAXCSM
        INTEGER*4      DOF, BUF, MASK, KC, ICLQ, FCON, SPK
        INTEGER*4      BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4      MAXDOF, NEQNS, NUMJNT
        INTEGER*4      MSGLVA, IERRA , MAXSA , NVARS
        REAL           GZTIME, GCTIME, GIJTIM, GFTIME, GMTIME,GNTIME,
     1                 CSMTIM, CSMSTR
        REAL           RATIOS, RATIOL, TIME
C
        INTEGER*4      SPACE
        REAL           CTIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM,
     1                 JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        COMMON /CSMMAP/ DOF, BUF, MASK, KC, ICLQ, FCON, SPK
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /CSMDTA/ GZTIME, GCTIME, GIJTIM, GFTIME, GMTIME,GNTIME,
     1                 CSMTIM, CSMSTR
        COMMON /PRBLEM/ MAXDOF,  NEQNS , NUMJNT
C
        COMMON /SPAUSR/ MSGLVA, IERRA , MAXSA , NVARS
C
C***************************************************************************
C
        INTEGER*4    JLONG, NLONG, CSIZE
        INTEGER*4    IDUMMY, INDATA
        REAL         RN, RNJNT, ROFFS, ROFFL, DUMMY
      DOUBLE PRECISION  RELERR, RELRES
C
C      ---------------------------
C      INITIALIZE SPARSPAK-A AND SYSTEM TIMER
C      ---------------------------
      CALL SPRSPK
C      ---------------------------------
C      INITIALIZE THE CSM-SPARSPAK INTERFACE PACKAGE
C      ---------------------------------
      CALL SPKCSM
C      ---------
C      SET MSGLVL AS DESIRED
C      ---------
      INDATA = 41
      READ ( INDATA, 12 ) MSGLVL
 12    FORMAT ( I4 )
C      ---------
C      SET MSGLVA AS DESIRED
C      ---------
      READ ( INDATA, 12 ) MSGLVA
C      ----------
C      SET MAXIMUM BUFFER LENGTH
C      ----------
      READ ( INDATA, 12 ) BUFMAX
C      ------------------------------------
C      INPUT NAME OF LIBRARY AND DATASETS FOR GIVEN PROBLEM
C      ------------------------------------
      READ ( INDATA, 22 ) LIBNAM
 22    FORMAT( A40 )
      READ ( INDATA, 32 ) JDFSET
      READ ( INDATA, 32 ) KMAP
      READ ( INDATA, 32 ) KSPAR
      READ ( INDATA, 32 ) CON
      READ ( INDATA, 32 ) APPLF
      READ ( INDATA, 32 ) APPLM
      READ ( INDATA, 32 ) STATD
 32    FORMAT( A51 )
C      --------
C      OPEN THE LIBRARY
C      --------
      CALL LIBOPN
C      ---------
C      INITIALIZE THE TIMER
C      ---------
      DUMMY = CTIME(0)
      MXREQD = BUFMAX
C      ---------
C      SIZE OF STORAGE ARRAY
C      ---------
      MAXCSM = MXSTOR
C      -----------
C      CHECK MAXCSM AGAINST MXREQD
C      -----------
      IF ( SPACE ( IDUMMY ) .NE. 0 ) GO TO 9999
C      ---------------------------------
C      RETRIEVE TOTAL NUMBER OF JOINTS AND STORE IN NJMJNT
C      ---------------------------------
      CALL GETJDF ( A )
C      --------------------
C      COMPUTE FURTHER STORAGE REQUIREMENT
C      --------------------
      ROFFS = RATIOS - 0.01
      ROFFL = RATIOL - 0.01
      RNJNT = NUMJNT + 1
      JLONG = IFIX((RNJNT+ROFFL)/RATIOL)
      MXREQD = JLONG + BUFMAX
      IF ( SPACE ( IDUMMY ) .NE. 0 ) GO TO 9999
C      --------
C      COMPUTE ADDRESSES
C      --------
      DOF = 1
      BUF = DOF + JLONG
C      ---------------------
C      RETRIEVE DEGREES OF FREEDOM PER JOINT,
C      AND INITIALIZE MAXDOF AND NEQNS
C      ---------------------
      CALL GETDOF ( A(DOF), A(BUF) )
```

```
C      ------------
C      ADJUST BUFFER SPACE
C      ------------
       MXREQD = MXREQD - BUFMAX
       BUFMAX = MAX0 ( BUFMAX, NEQNS )
       MXREQD = MXREQD + BUFMAX
       IF ( SPACE ( IDUMMY ) .NE. 0 ) GO TO 9999
       MXUSED = MXREQD
C      ----------------------
C      COMPUTE FURTHER STORAGE REQUIREMENT
C      ----------------------
       RN = NEQNS
       NLONG = IFIX ((RN+ROFFL)/RATIOL)
       MXREQD = MXUSED + NLONG
       IF ( SPACE ( IDUMMY ) .NE. 0 ) GO TO 9999
C      ----------
C      COMPUTE ADDRESSES
C      ----------
       MASK = BUF + BUFMAX
C      ----------
C      DETECT DUMMY ROWS
C      ----------
       CALL GTZERO (A(DOF), A(BUF), A(MASK) )
       MXUSED = MXREQD
C      ----------------------
C      COMPUTE FURTHER STORAGE REQUIREMENT
C      ----------------------
       MXREQD = MXUSED + 7
       IF ( SPACE ( IDUMMY ) .NE. 0 ) GO TO 9999
C      ----------
C      COMPUTE ADDRESSES
C      ----------
       KC = MASK + NLONG
C      ---------------
C      DETECT CONSTRAINED VARIABLES
C      ---------------
       CALL GTCOND (A(DOF), A(BUF), A(KC), A(MASK), CSIZE)
       MXUSED = MXREQD
C      ----------------------
C      COMPUTE FURTHER STORAGE REQUIREMENT
C      ----------------------
       MXREQD = MXUSED + MAXDOF + CSIZE
       IF ( SPACE ( IDUMMY ) .NE. 0 ) GO TO 9999
C      -------------
C      TOTAL STORAGE TO BE USED
C      -------------
       MXUSED = MXREQD
C      ----------
C      COMPUTE ADDRESSES
C      ----------
       ICLQ = KC + 7
       FCON = ICLQ + MAXDOF
C      --------------
C      GATHER NONZERO CONSTRAINTS
C      --------------
       CALL GTMOTI ( A(BUF), A(MASK), A(FCON), CSIZE )
C      --------------
C      INTERFACE WITH SPARSPAK-A
C      --------------
       SPK = MXUSED + 1
       MAXSA = MAXCSM - MXUSED
C      ----------------------
C      INPUT NONZERO STRUCTURE TO SPARSPAK-A
C      ----------------------
       CALL GETIJ(A(DOF), A(BUF), A(ICLQ), A(MASK), A(SPK))
C      ---------------
C      DETERMINE SYMMETRIC ORDERING
C      ---------------
       CALL ORDRB5 ( A(SPK) )
C      ------------
C      INPUT RIGHT HAND SIDE
C      ------------
       CALL GTFORC( A(BUF),A(MASK), A(SPK) )
C      ------------------------------------
C      INPUT MATRIX COEFFICIENTS AND RIGHT HAND SIDE MODIFICATIONS
C      ------------------------------------
       CALL GTNUM5(A(DOF), A(BUF), A(MASK), A(FCON), A(SPK))
```

```
C     ----------------------------------------
C      PERFORM NUMERICAL FACTORIZATION AND SOLUTION
C     ----------------------------------------
      CALL SOLVES ( A(SPK) )
      CSMTIM = CTIME(0)
      CALL ERESTS ( RELERR, A(SPK))
C     ----------------------------------
C      COMPARE WITH KNOWN NICESPAR SOLUTION
C     ----------------------------------
      CALL GETSOL (A(BUF), A(SPK), RELRES )
      CALL STATCS
      CALL STATSA
C
 9999  CONTINUE
      RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C          SPKCSM ..... INITIALIZE PARAMETERS
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - TO SET SYSTEM PARAMETERS AND ASSIGN DEFAULT
C       VALUES TO SOME USER PARAMETERS. IT IS A MACHINE
C       DEPENDENT ROUTINE. THIS ROUTINE HAS TO BE CALLED
C       BEFORE ANY OTHER PACKAGE MODULE.
C
C    PARAMETERS INITIALIZED -
C       IPRNTE - THE OUTPUT UNIT NUMBER FOR ERROR MESSAGES.
C       IPRNTS - THE OUTPUT UNIT NUMBER FOR STATISTICS.
C       RATIOL - THE RATIO OF THE NUMBER OF BITS IN A FLOATING
C              POINT VARIABLE TO THAT IN A LONG INTEGER
C              VARIABLE.  FOR EXAMPLE, IF FLOATING POINT
C              NUMBERS OCCUPY TWICE AS MANY BITS AS LONG
C              INTEGERS, RATIOL SHOULD BE SET TO 2.
C       RATIOS - THE RATIO OF THE NUMBER OF BITS IN A FLOATING
C              POINT VARIABLE TO THAT IN A SHORT INTEGER
C              VARIABLE.
C       MAXINT - THE LARGEST POSITIVE INTEGER THAT CAN BE
C              STORED IN A SHORT INTEGER VARIABLE.
C       TIME   - VARIABLE USED BY THE TIMER ROUTINE CTIME.
C                SEE REMARK
C       STAGE  - STARTING STAGE OF SYSTEM-CSM.
C
C     REMARK - THIS INTERFACE PACKAGE ASSUMES THE EXISTENCE OF
C           A REAL TIME FUNCTION CTIME WHICH RETURNS THE ELAPSED
C           PROCESSOR TIME SINCE IT WAS LAST CALLED.  WITH THE
C           COMMON VARIABLE TIME, THE INSTALLER OF THE PACKAGE
C           SHOULD BE ABLE TO WRITE SUCH A FUNCTION, USING THE
C           INSTALLATION TIMER.
C
C
C*****************************************************************************
C
      SUBROUTINE SPKCSM
C
C*****************************************************************************
C
        CHARACTER*40 LIBNAM
        CHARACTER*51 CDUMMY(7)
        INTEGER*4 IIN, IOUTX
        INTEGER*4 IPRNTE, IPRNTS, MAXINT
        INTEGER*4 BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4 MSGLVL, IERR  , MAXCSM
        INTEGER*4 IIN, IOUTX
        REAL      RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM, CDUMMY(7)
        COMMON /IANDO/  IIN, IOUTX
C
C     ***********************************************************************
C          W    A    R    N    I    N    G
C     ***********************************************************************
C
C       THE FOLLOWING 4 LINES OF CODE ARE INSTALLATION
C       DEPENDENT. THEY MAY HAVE TO BE MODIFIED BY THE
C       PERSON INSTALLING THIS PACKAGE.
C
C       OUR CURRENT ENVIRONMENT -
C          - RATIOL AND RATIOS ARE BOTH 2.
C          - MAXINT = 2**15 - 1 = 32767
C
C       ---------------------
C       INSTALLATION DEPENDENT PARAMETERS
C       ---------------------
        TIME = 0.0
C
        RATIOL = 2.0
        RATIOS = 2.0
C
        MAXINT = 32767
```

```
C
C     ________________________________
C     IPRNTE AND IPRNTS ARE BOTH SET TO THE WRITER UNIT
C     NUMBER ASSIGNED TO IOUTX WHEN THE NEW PROCESSOR
C     IS IDENTIFIED TO THE CSM-ARCHITECTURE.
C     ________________________________
      IPRNTE = IOUTX
      IPRNTS = IOUTX
C
C     ____________________________________
C     INITIALIZING THE EXECUTION STAGE FOR THE INTERFACE ...
C     ____________________________________
      STAGE  = 0
C
      RETURN
C
    END
```

```
C***************************************************************************
C***************************************************************************
C               CTIME ..... ELAPSED PROCESSOR TIME
C***************************************************************************
C***************************************************************************
C
C    PURPOSE - CTIME RETURNS THE ELAPSED PROCESSOR TIME SINCE
C      IT WAS LAST CALLED.  IT USES THE COMMON VARIABLE TIME
C      TO REMEMBER THE TIME WHEN CTIME WAS LAST CALLED.
C
C    ***********************************************************************
C                        W A R N I N G
C    ***********************************************************************
C      THIS IS AN INSTALLATION DEPENDENT ROUTINE. IT
C      SHOULD BE SET UP BY THE INSTALLER OF THE PACKAGE.
C      IN THIS EXAMPLE, ROUTINE GTIMER IS THE TIMER ROUTINE
C      THAT RETURNS THE CURRENT PROCESSOR TIME ON A SUN/3
C      WORKSTATION RUNNING THE UNIX OPERATING SYSTEM AT THE
C      UNIVERSITY OF TENNESSEE KNOXVILLE.
C    ***********************************************************************
C
C    INPUT PARAMETER -
C      IDUMMY - A DUMMY INTEGER VARIABLE.
C
C    PROGRAM SUBROUTINE -
C      GTIMER.
C
C***************************************************************************
C
      REAL FUNCTION  CTIME ( IDUMMY )
C
C***************************************************************************
C
        INTEGER   IDUMMY, IPRNTE, IPRNTS, MAXINT
        REAL      RATIOL, RATIOS, TIME  , X
C
C***************************************************************************
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL,
     1              TIME
C
C***************************************************************************
C
        CALL  GTIMER ( X )
        CTIME = X - TIME
        TIME = X
        RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C               GETJDF ..... GET NUMBER OF JOINTS ...
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - THIS ROUTINE  RETRIEVES THE TOTAL NUMBER OF JOINTS
C       FOR THE PROBLEM TO BE SOLVED.
C
C     PARAMETERS INITIALIZED  -
C       NUMJNT - THE TOTAL NUMBER OF JOINTS.
C
C     ERROR CODES -
C       0    - ERROR CODES
C       1013 - INCORRECT EXECUTION SEQUENCE
C       1014 - THE NUMBER OF ITEMS AVAILABLE FROM THE RETRIEVED
C              DATASET IS LESS THAN TWO.  SEE REMARK.
C
C     ______________________________
C     REMARK -
C       THE CURRENT VERSION OF TESTBED DATABASE ASSUMES THAT
C       ALL JOINTS HAVE THE MAXIMUM DEGREES OF FREEDOM, THE
C       NUMBER OF JOINTS AND THE MAXIMUM DEGREES PER JOINT IS
C       FROM THE FIRST TWO ITEMS RETRIEVED.  IN CASE OF
C       VARIABLE DEGREES OF FREEDOM PER JOINT, DUMMY DATA IS
C       STORED.
C     ______________________________
C
C     PROGRAM SUBROUTINES -
C       QKINFO, GETRECI, EMSG
C
C     CSM TESTBED DATASETS ACCESSED -
C       JDF1.BTAB.*
C
C*****************************************************************************
C
      SUBROUTINE GETJDF ( IBUF )
C
        INTEGER*4      IBUF(1)
C
C
C*****************************************************************************
C
        CHARACTER*40  LIBNAM
        CHARACTER*51  JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        INTEGER*4     IPRNTE, IPRNTS, MAXINT
        INTEGER*4     MSGLVL, IERR, MAXCSM
        INTEGER*4     BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4     MAXDOF, NEQNS , NUMJNT
        REAL          RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1              JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /PRBLEM/ MAXDOF, NEQNS , NUMJNT
C
C*****************************************************************************
C
        INTEGER*4  LEN
C
        IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11     FORMAT ( /5X, 'GETJDF - GET NUMBER OF JOINTS AND ... ' )
C
        IF (( STAGE .LT. 10 ) .OR. ( IERR .NE. 0 )) GO TO 100
C
C       ______________________
C       EACH DATASET IS IDENTIFIED BY A STRING OF
C       'MAINKEY.EXTENSION.CYCLE1.CYCLE2.CYCLE3'
C       MAXIMUM NUMBER OF CHARACTERS CONTAINED IS 51
C       ______________________
        CALL QKINFO ( JDFSET )
        IF ( IERR .NE. 0 ) RETURN
C
        STAGE = 15
C       ______________________
C       GET THE FIRST TWO ITEMS OF THE FIRST RECORD
```

```
C       ————————————————————————
        LEN = 2
        CALL GTRECI ( 1, IBUF, LEN )
        IF ( IERR .NE. 0 ) RETURN
        IF ( LEN .LT. 2 ) GO TO 200
C
        NUMJNT = IBUF(1)
C       ——————————————————————————————
C        READ IN MAX UNCONSTRAINED DEGREES OF FREEDOM OF THE MODEL
C       ——————————————————————————————
        MAXDOF = IBUF(2)
        STAGE = 20
        RETURN
C
C       ————————
C        ERROR HANDLING
C       ————————
 100     CONTINUE
        IERR = 1013
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
 200     IERR = 1014
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C            GETDOF ..... GET DEGREES OF FREEDOM
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - THIS ROUTINE RETRIEVES THE DEGREE OF FREEDOM
C      FOR EACH INDIVIDUAL JOINT FROM THE DATABASE.
C
C    PARAMETERS INITIALIZED -
C      IDOF  - IDOF(K) STORES THE STARTING EQUATION NUMBER FOR
C              JOINT K.  THE DEGREES OF FREEDOM FOR JOINT K IS
C              GIVEN BY IDOF(K+1) - IDOF(K).  THE TOTAL NUMBER
C              OF EQUATIONS IS EQUAL TO IDOF(NUMJNT+1) - 1.
C      MAXDOF - THE MAXIMUM DEGREE OF FREEDOM RETRIEVED FOR AN
C              INDIVIDUAL JOINT.
C      NEQNS  - THE NUMBER OF EQUATIONS EQUALS THE TOTAL DEGREES
C              OF FREEDOM.
C
C    CSM TESTBED DATASET ACCESSED -
C      CURRENTLY NONE.
C
C    ************************************************************************
C            W   A   R   N   I   N   G
C    ************************************************************************
C              THIS SUBROUTINE MUST BE MODIFIED FOR PROBLEMS WITH
C              VARIABLE DEGREES OF FREEDOM PER NODE
C
C*****************************************************************************
C
      SUBROUTINE GETDOF ( IDOF, IBUF )
C
        INTEGER*4  IDOF(1), IBUF(1)
C
C*****************************************************************************
C
        CHARACTER*40  LIBNAM
        CHARACTER*51  JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        INTEGER*4     IPRNTE, IPRNTS, MAXINT
        INTEGER*4     MSGLVL, IERR, MAXCSM
        INTEGER*4     BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4     MAXDOF, NEQNS , NUMJNT
        REAL          RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1                JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /PRBLEM/ MAXDOF, NEQNS , NUMJNT
C
C*****************************************************************************
C
        INTEGER*4  DEGREE, I
C
        IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11     FORMAT( /5X, 'GETDOF - GET DEGREES OF FREEDOM ... ' )
C
        IF (( STAGE .LT. 20 ) .OR. ( IERR .NE. 0 )) GO TO 500
C
C        ------------------------
C        THE FOLLOWING LINES OF CODE IS TEMPORARY
C        FOR THE-FIXED DEGREE PROBLEMS
C        ------------------------
        DEGREE = MAXDOF
        IDOF(1) = 1
        DO 100 I = 2, NUMJNT+1
          IDOF(I) = IDOF(I-1)+DEGREE
          IF ( MAXDOF .LT. DEGREE ) MAXDOF = DEGREE
 100    CONTINUE
        NEQNS = IDOF(NUMJNT+1) - 1
        STAGE = 30
        RETURN
C
 500    CONTINUE
        IERR = 1019
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
```

```
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C                GTZERO .... DETECT DUMMY ROWS
C*****************************************************************************
C*****************************************************************************
C
C   PURPOSE - THIS ROUTINE IDENTIFIES DUMMY ROWS (ALL ZEROS) IN
C     THE DATA MATRIX.
C
C   INPUT PARAMETERS -
C     DOF - AN INTEGER ARRAY OF SIZE EQUAL TO THE TOTAL NUMBER OF
C         JOINTS PLUS ONE.
C         IDOF(K) STORES THE STARTING EQUATION NUMBER FOR
C         JOINT K.  THE DEGREES OF FREEDOM FOR JOINT K IS
C         GIVEN BY IDOF(K+1) - IDOF(K).  THE TOTAL NUMBER
C         OF EQUATIONS IS EQUAL TO IDOF(NUMJNT+1) - 1.
C
C   OUTPUT PARAMETERS -
C     MASK - THE LINEAR ARRAY MASK STORES A 0 FOR EACH
C          ZERO DIAGONAL ELEMENT ENCOUNTERED AND A -1
C          FOR EACH NONZERO DIAGONAL ELEMENT.
C
C   WORKING PARAMETERS -
C     FBUF - A BUFFER OF MAXIMUM RECORD SIZE FOR RETRIEVING
C          REAL OR DOUBLE PRECISION DATA FORM THE TESTBED.
C
C   ERROR CODES -
C     1021 - INCORRECT EXECUTION SEQUENCE.
C
C   SUBPROGRAM MODULES -
C     QKINFO, GTRECF, EMSG
C
C   CSM TESTBED DATASETS ACCESSED -
C     K.SPAR.*
C
C   ----------------------------------
C   REMARK - THIS ROUTINE IS NEEDED FOR THE CURRENT RELEASE OF
C     TESTBED DATABASE BECAUSE THE CONSTRAINT DATASET DOES NOT
C     INCLUDE ZERO ROWS.  IN ADDITION, NOTE THAT CURRENTLY
C     THE TESTBED STORES MAXDOF EQUATIONS PER JOINT.  THEREFORE,
C     DUMMY ROWS MUST BE INSERTED FOR THE JOINTS WITH DEGREES
C     LESS THAN MAXDOF.
C   ----------------------------------
C
C*****************************************************************************
C
      SUBROUTINE GTZERO ( DOF, FBUF, MASK )
C
      DOUBLE PRECISION  FBUF(1)
      INTEGER*4   MASK(1), DOF(1)
C
C*****************************************************************************
C
      CHARACTER*40   LIBNAM
      CHARACTER*51   JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      CHARACTER*4    RTYPE
      INTEGER*4      IPRNTE, IPRNTS, MAXINT
      INTEGER*4      IDSN  , LDI   , NLEN  , NREC  , TRACE
      INTEGER*4      BUFMAX, MXUSED, MXREQD, STAGE
      INTEGER*4      MSGLVL, IERR, MAXCSM
      INTEGER*4      MAXDOF , NEQNS , NUMJNT
      REAL        RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1           TRACE
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1           JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      COMMON /PRBLEM/ MAXDOF , NEQNS , NUMJNT
C
C*********************************************************************
C
      INTEGER*4  CONRNG, I, II, IROW, IS, ITEMS, JGRPS, JOINT, LEN
      INTEGER*4  CJNT, NROWS, NCOLS, ISIZE, KOUNT, OVERHD, NZEROS
      DOUBLE PRECISION  COEF
C
```

```
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11    FORMAT( /5X, 'GTZERO - DETECT DUMMY ROWS ... ' )
C
      IF ( ( STAGE .LT. 30 ) .OR. ( IERR .NE. 0 ) ) GO TO 500
C
C      ----------------
C      INITIALIZE MASK TO BE -1
C      ----------------
      DO 100 I = 1, NEQNS
        MASK(I) = -1
 100   CONTINUE
C      --------------------------
C      EACH DATASET IS IDENTIFIED BY A STRING OF
C      'MAINKEY.EXTENSION.CYCLE1.CYCLE2.CYCLE3'
C      MAXIMUM NUMBER OF CHARACTERS CONTAINED IS 51
C      --------------------------
      CALL QKINFO ( KSPAR )
      IF ( IERR .NE. 0 ) RETURN
C
      OVERHD = 0
      KOUNT = 0
      NZEROS = 0
      TRACE = TRACE + 10
      DO 200 I = 1, NREC
        LEN = NLEN
        CALL GTRECF ( I, FBUF, LEN )
        IF ( IERR .NE. 0 ) RETURN
C        ---------------------------
C        DETERMINE NUMBER OF JOINT GROUPS IN CURRENT RECORD
C        ---------------------------
        JGRPS = FBUF(1)
        ITEMS = 1
        OVERHD = OVERHD + 1
        DO 300 II = 1, JGRPS
          CONRNG = FBUF(ITEMS+1)
          JOINT = FBUF(ITEMS+2)
          NROWS = DOF(JOINT+1) - DOF(JOINT)
C          ---------------
C          COMPUTE THE SIZE OF DATA ITEMS. IN TOTAL
C          CONRNG SUBMATRICES INCLUDING DIAGONAL SUBMATICES
C          ---------------
          ISIZE = 0
          DO 350 IS = 1, CONRNG
            CJNT  = FBUF(ITEMS+1+IS)
            NCOLS = DOF(CJNT+1) - DOF(CJNT)
            ISIZE = ISIZE + NROWS*NCOLS
 350       CONTINUE
          ITEMS = ITEMS + 1 + CONRNG
          OVERHD = OVERHD + 1 + CONRNG
C          -----------------------
C          ACCESS THE DIAGONAL ELEMENTS ON THE DIAGONAL MATRIX
C          -----------------------
          IROW = DOF(JOINT) - 1
          NCOLS = NROWS
          DO 400 IS = 1, NCOLS
            COEF = FBUF(ITEMS+(IS-1)*NROWS+IS)
C            -----------
C            A DUMMY ROW IS DETECTED
C            -----------
            IF ( COEF .EQ. 0.0D0 ) THEN
              MASK ( IROW + IS ) = 0
              KOUNT = KOUNT + 1
            ENDIF
 400       CONTINUE
          ITEMS = ITEMS + ISIZE
          NZEROS = NZEROS + ISIZE
 300     CONTINUE
 200   CONTINUE
      STAGE = 40
C      -----------
C      PRINT DEBUGGIN DATA ...
C      -----------
      IF ( MSGLVL .GE. 3 ) WRITE ( IPRNTS, 22 ) KOUNT,
     1                 OVERHD, NZEROS
 22   FORMAT ( 15X, 'NUMBER OF DUMMY ROWS:  ' , I8
     1         /15X, 'K.SPAR.* INDEX OVERHEAD:', I8
     1         /15X, 'K.SPAR.* NONZEROS :     ', I8 )
```

C-2

```
      RETURN
 500   CONTINUE
C     ------------
C      ERROR HANDLING ...
C     ------------
      IERR = 1021
      IF ( MSGLVL .GE. 2 ) CALL EMSG
      RETURN
C
    END
```

```
C****************************************************************************
C****************************************************************************
C             GTCOND .... RETRIEVE CONSTRAINT INFO
C****************************************************************************
C****************************************************************************
C
C    PURPOSE - THIS ROUTINE RETRIEVES THE CONSTRAINED COMPONENTS
C      OF EACH JOINT AND TREATS THE DUMMY ROWS AS CONSTRAINED
C      TO BE ZERO.
C
C    INPUT PARAMETERS -
C      DOF - AN INTEGER ARRAY OF SIZE EQUAL TO THE TOTAL NUMBER OF
C           JOINTS PLUS ONE.
C           IDOF(K) STORES THE STARTING EQUATION NUMBER FOR
C           JOINT K.  THE DEGREES OF FREEDOM FOR JOINT K IS
C           GIVEN BY IDOF(K+1) - IDOF(K).  THE TOTAL NUMBER
C           OF EQUATIONS IS EQUAL TO IDOF(NUMJNT+1) - 1.
C      MASK - RECORD OF DUMMY ROWS.
C
C    OUTPUT PARAMETERS -
C      MASK - RECORD OF CONSTRAINED VARIABLES IN ADDITION TO
C           DUMMY ONES.
C      CSIZE - TOTAL NUMBER OF NONZERO CONSTRAINTS.
C
C    WORKING PARAMETERS -
C      IBUF - A BUFFER OF MAXIMUM RECORD SIZE FOR RETRIEVING
C           INTEGER DATA FORM THE TESTBED.
C      KC   - AN TEMPORARY INTEGER ARRAY OF SIZE (MAXDOF+1)
C           NEEDED IN DECODING THE CONSTRAINT DATA.
C
C    ERROR CODES -
C      1022 - INCORRECT EXECUTION SEQUENCE.
C
C    SUBPROGRAM MODULES -
C      QKINFO, GTRECI, DECODE, EMSG
C
C    CSM TESTBED DATASETS ACCESSED -
C      CON..* OR CON..i (IF MULTIPLES EXISTS IN DATA LIBRARY)
C
C    ______________________________
C    REMARKS -
C      IT IS ASSUMED THAT THE CONSTRAINED DATA IS STORED
C      IN THE DATASET IN THE ORDER OF JOINT NUMBERS.
C    ______________________________
C
C****************************************************************************
C
C
      SUBROUTINE GTCOND ( DOF, IBUF, KC, MASK, CSIZE )
C
      INTEGER*4  DOF(1), IBUF(1), KC(1), MASK(1), CSIZE
C
C****************************************************************************
C
      CHARACTER*40   LIBNAM
      CHARACTER*51   JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      CHARACTER*4    RTYPE
      INTEGER*4      IPRNTE, IPRNTS, MAXINT
      INTEGER*4      IDSN  , LDI  , NLEN  , NREC  , TRACE
      INTEGER*4      BUFMAX, MXUSED, MXREQD, STAGE
      INTEGER*4      MSGLVL, IERR, MAXCSM
      INTEGER*4      MAXDOF , NEQNS , NUMJNT
      REAL           RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMSPK/ IDSN  , LDI  , NLEN  , NREC  , RTYPE ,
     1               TRACE
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1               JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      COMMON /PRBLEM/ MAXDOF , NEQNS , NUMJNT
C
C****************************************************************************
C
      INTEGER*4  I, II, IROW, JOINT, K, LEN, DEGREE, ZKOUNT,FKOUNT,
     1          ZDUMMY
C
```

```
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11    FORMAT( /5X, 'GTCOND - GET CONSTRAINTED VARIABLES... ' )
C
      IF ( ( STAGE .LT. 40 ) .OR. ( IERR .NE. 0 ) ) GO TO 500
C
C      -------------------------------
C      EACH DATASET IS IDENTIFIED BY A STRING OF
C      'MAINKEY.EXTENSION.CYCLE1.CYCLE2.CYCLE3'
C      MAXIMUM NUMBER OF CHARACTERS CONTAINED IS 51
C      -------------------------------
      CALL QKINFO ( CON )
      IF ( IERR .NE. 0 ) RETURN
      TRACE = TRACE + 10
C      ----------------
C      KOUNTING NONZERO CONSTRAINTS
C      ----------------
      CSIZE = 0
C      ---------------
C      KOUNTING ZERO CONSTRAINTS
C      ---------------
      ZKOUNT = 0
      JOINT = 1
      DO 100 I = 1, NREC
        LEN = NLEN
        CALL GTRECI ( I, IBUF, LEN )
        IF ( IERR .NE. 0 ) RETURN
        DO 200 II = 1, LEN
          IF ( JOINT .GT. NUMJNT ) GO TO 200
C          ------------------
C          CONSTRAINTS ARE ENCODED INTO 7 BITS
C          WHICH ARE DECODED INTO AN INTEGER
C          ARRAY KC OF SIZE 7 !
C          ------------------
          CALL DECODE ( IBUF(II), KC )
          DEGREE = DOF(JOINT+1) - DOF(JOINT)
          IROW = DOF(JOINT) - 1
          DO 300 K = 1, DEGREE
            IF ( KC(K) .EQ. 1 ) THEN
C              ----------
C              ZERO CONSTRAINTS
C              ----------
              MASK(IROW+K) = 0
              ZKOUNT = ZKOUNT + 1
            ELSE IF ( KC(K) .EQ. 2 ) THEN
C              ------------
C              NONZERO CONSTRAINTS
C              ------------
              MASK(IROW+K) = 1
              CSIZE = CSIZE + 1
            ENDIF
 300       CONTINUE
          JOINT = JOINT + 1
 200     CONTINUE
 100   CONTINUE
C      -------------------------------
C      KOUNTING UNCONSTRAINED DEGREES OF FREEDOM AND
C      THE NET ZERO CONSTRAINTS INCLUDING DUMMY ROWS
C      -------------------------------
      FKOUNT = 0
      ZDUMMY = 0
      DO 400 I = 1, NEQNS
        IF ( MASK(I) .EQ. -1 ) FKOUNT = FKOUNT + 1
        IF ( MASK(I) .EQ. 0  ) ZDUMMY = ZDUMMY + 1
 400   CONTINUE
      STAGE = 50
C      --------------
C      PRINT DEBUGGING DATA ...
C      --------------
      IF ( MSGLVL .GE. 3 ) WRITE (IPRNTS, 22) ZKOUNT, CSIZE,
     1 FKOUNT, ZDUMMY
 22   FORMAT( 15X, 26H  ZERO CONSTRAINTS ARE   , I8
     1       /15X, 26HNONZERO CONSTRAINTS ARE   , I8
     1       /15X, 26HFREE VARIABLES ARE        , I8
     1       /15X, 26HDUMMY ROWS + 0 CONSTRAINTS, I8 )
      RETURN
C
 500  CONTINUE
```

```
C         ___________
C         ERROR HANDLING
C         ___________
         IERR = 1022
         IF ( MSGLVL .GE. 2 ) CALL EMSG
         RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C                GTMOTI ....GET NONZERO CONSTRAINTS
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - TO RETRIEVE NUMERIC FOR NONZERO CONSTRAINTS.
C
C     INPUT PARAMETERS
C       MASK - CONSTRAINT INFORMATION FOR EACH VARIABLE.
C
C     OUTPUT PARAMETERS
C       MASK - THE LOCATIONS CORRESPONDING TO NONZEROR CONSTRAINTS
C            CONTAIN A POINTER TO THE NUMERIC VALUE IN FCON.
C       FCON - AN ARRAY OF CSIZE FLOATING-POINT CONSTRAINTS.
C
C
C     WORKING PARAMETERS
C       FBUF - A REAL OR DOUBLE PRECISION BUFFER OF SIZE BUFMAX.
C            THE ACTUAL TYPE IS AS DECLARED.
C
C     ERROR CODES -
C       1025 - INCORRECT EXECUTION SEQUENCE.
C       1026 - UNEXPECTED NONZERO CONSTRAINT VALUE.
C       1027 - ZERO ENTRY FOR A NONZERO CONSTRAINT OCCURS.
C
C     SUBROUTINE PROGRAMS -
C       QKINFO, GTRECF, EMSG.
C
C     CSM TESTBED DATASETS ACCESSES -
C       APPL.MOTI.i.j.
C
C     ------------------------------------
C     REMARKS -
C       IT IS ASSUMED THAT THE CONSTRAINT VALUES ARE STORED
C       IN SEQUENCE FROM 1 TO NEQNS.
C     ------------------------------------
C
C*****************************************************************************
C
      SUBROUTINE GTMOTI ( FBUF, MASK, FCON, CSIZE )
C
      INTEGER*4   MASK(1), CSIZE
      DOUBLE PRECISION  FBUF(1), FCON(1)
C
C*****************************************************************************
C
      CHARACTER*40   LIBNAM
      CHARACTER*51   JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      CHARACTER*4    RTYPE
      INTEGER*4      IPRNTE, IPRNTS, MAXINT
      INTEGER*4      IDSN  , LDI   , NLEN  , NREC  , TRACE
      INTEGER*4      BUFMAX, MXUSED, MXREQD, STAGE
      INTEGER*4      MSGLVL, IERR, MAXCSM
      INTEGER*4      MAXDOF , NEQNS , NUMJNT
      REAL           RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1                TRACE
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1                JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      COMMON /PRBLEM/ MAXDOF , NEQNS , NUMJNT
C
C*****************************************************************************
C
      INTEGER*4  CSIZE, NITEMS, KPTR, LEN, I, J
C
      IF (  MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11    FORMAT( /5X, 'GTMOTI - GET NONZERO CONSTRAINTS... ' )
C
      IF (( STAGE .LT. 50 ) .OR. ( IERR .NE. 0 ) ) GO TO 1000
C
      IF ( CSIZE .EQ. 0 ) THEN
C       --------------------
C         NONZERO CONSTRAINTS ARE NOT EXPECTED
```

```
C         ---------------------
          IF ( MSGLVL .GE. 3 ) WRITE ( IPRNTS, 21 )
 21        FORMAT( /10X, 'APPLIED DISPLACEMENTS ARE NOT EXPECTED.')
          STAGE = 60
          RETURN
        ENDIF
C
C        ----------------------
C        RETRIEVE NEQNS ITEMS FORM 'APPL.MOTI.* '
C        ----------------------
        CALL QKINFO ( APPLM )
        IF ( IERR .NE. 0 ) RETURN
        TRACE = TRACE + 10
        NITEMS = 0
        KPTR = 0
        DO 100 I = 1, NREC
          LEN = MIN0 ( NEQNS - NITEMS, NLEN )
          IF ( LEN .GT. 0 ) THEN
            CALL GTRECF ( I, FBUF, LEN )
            IF ( IERR .NE. 0 ) RETURN
            DO 200 J = 1, LEN
              NITEMS = NITEMS + 1
C              ----------------------------
C              CHECK ERROR DUE TO INCONSISTENT CONSTRAINT VALUES
C              ----------------------------
              IF (( MASK(NITEMS) .NE. 1 ) .AND.
     1            ( FBUF(J) .NE. 0.0D0 )) GO TO 1100
              IF (( MASK(NITEMS) .EQ. 1 ) .AND.
     1            ( FBUF(J) .EQ. 0.0D0 )) GO TO 1200
              IF ( MASK(NITEMS) .EQ. 1 ) THEN
C                -----------------
C                ENTER NUMERIC FOR NONZERO CONSTRAINT
C                -----------------
                KPTR = KPTR + 1
                FCON(KPTR) = FBUF(J)
C                -----------------
C                STORE THE ADDRESS POINTER IN MASK
C                -----------------
                MASK(NITEMS) = KPTR
              ENDIF
 200        CONTINUE
          ENDIF
 100    CONTINUE
        STAGE = 60
        RETURN
C
C        --------
C        ERROR HANDLING
C        --------
1000     CONTINUE
        IERR = 1025
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
1100     CONTINUE
        IERR = 1026
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
1200     CONTINUE
        IERR = 1027
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
      END
```

```
C***********************************************************************
C***********************************************************************
C            GETIJ .... INPUT NONZERO STRUCTURES
C***********************************************************************
C***********************************************************************
C
C     PURPOSE - TO RETRIEVE NONZERO STRUCTURES FROM DATASET KMAP..*
C        AND INPUT THE SAME TO SPARSPAK-A.
C
C     INPUT PARAMETERS
C        DOF  - AN INTEGER ARRAY OF SIZE EQUAL TO THE TOTAL NUMBER OF
C               JOINTS PLUS ONE.
C               IDOF(K) STORES THE STARTING EQUATION NUMBER FOR
C               JOINT K.  THE DEGREES OF FREEDOM FOR JOINT K IS
C               GIVEN BY IDOF(K+1) - IDOF(K).  THE TOTAL NUMBER
C               OF EQUATIONS IS EQUAL TO IDOF(NUMJNT+1) - 1.
C        MASK - CONSTRAINT INFORMATION FOR EACH VARIABLE.
C
C     OUTPUT PARAMETERS
C        S - NONZERO STRUCTURES SET UP BY SPARSPAK-A.
C
C     WORKING PARAMETERS
C        IBUF - AN INTEGER BUFFER OF SIZE BUFMAX.
C        ICLQ - A TEMPORARY ARRAY OF SIZE MAXDOF.
C
C     ERROR CODES -
C        1023 - INCORRECT EXECUTION SEQUENCE.
C
C     SUBROUTINE PROGRAMS -
C        QKINFO, GTRECI, EMSG
C
C     SPRSPAK-A SUBROUTINES -
C        IJBEGN, INCLQ, INIJ, IJEND.
C
C     CSM TESTBED DATASETS ACCESSES -
C        KMAP..*
C
C***********************************************************************
C
      SUBROUTINE GETIJ ( DOF, IBUF, ICLQ, MASK,  S )
C
      INTEGER*4  DOF(1), IBUF(1), ICLQ(1), MASK(1), S(1)
C
C***********************************************************************
C
      CHARACTER*40   LIBNAM
      CHARACTER*51   JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      CHARACTER*4    RTYPE
      INTEGER*4      IPRNTE, IPRNTS, MAXINT
      INTEGER*4      IDSN  , LDI  , NLEN  , NREC  , TRACE
      INTEGER*4      BUFMAX, MXUSED, MXREQD, STAGE
      INTEGER*4      MSGLVL, IERR, MAXCSM
      INTEGER*4      MAXDOF , NEQNS , NUMJNT
      REAL           RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMSPK/ IDSN  , LDI  , NLEN  , NREC  , RTYPE ,
     1                TRACE
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1                JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      COMMON /PRBLEM/ MAXDOF , NEQNS , NUMJNT
C
C***********************************************************************
C
      INTEGER*4  CONRNG, I, II, ICOL, IROW, ITEMS, J, JGRPS, JOINT,
     1           K, NCLQ, LEN, IX, JX, LRNG, NODES, JJ, NROWS,
     1           NCOLS
C
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
   11 FORMAT( /5X, 'GETIJ - INPUT NONZERO STRUCTURES... ' )
C
      IF (( STAGE .LT. 60 ) .OR. ( IERR .NE. 0 ) ) GO TO 1000
C
      CALL IJBEGN
C     ---------------------------
C      INIJ INSURES NONZERO FOR ALL DIAGONAL ELEMENTS
```

```
C      IF POSITION (NEQNS, NEQNS) IS ENTERED
C      ---------------------------------
      CALL INIJ ( NEQNS, NEQNS , S )
C      ---------------------------------
C      ACCESS EACH RECORD IN DATA SET 'KMAP..* '
C      ---------------------------------
      CALL QKINFO ( KMAP )
      IF ( IERR .NE. 0 ) RETURN
      TRACE = TRACE + 10
      DO 100 I = 1, NREC
        LEN = NLEN
        CALL GTRECI ( I, IBUF, LEN )
        IF ( IERR .NE. 0 ) RETURN
C        ---------------------------------
C        DETERMINE NUMBER OF JOINT GROUPS IN CURRENT RECORD
C        ---------------------------------
        JGRPS = IBUF(1)
        ITEMS = 1
        DO 200 II = 1, JGRPS
C          ---------------------------------
C          GET THE CURRENT JOINT AND COMPUTE THE ROW NUMBER
C          ---------------------------------
          JOINT = IBUF(ITEMS+1)
C          ---------------------------------
C          NUMBER OF DEGREES FOR CURRENT JOINT
C          ---------------------------------
          NROWS = DOF(JOINT+1) - DOF(JOINT)
C          ---------------------------------
C          COMPUTE THE THE ROW NUMBER BY IROW + K,
C          WHERE IROW IS GREATER THAN OR EQUAL TO 0
C          ---------------------------------
          IROW = DOF(JOINT) - 1
          NCLQ = 0
          DO 300  K = 1, NROWS
            IF ( MASK ( IROW + K ) .EQ. -1 ) THEN
C              ---------------------------------
C              THIS ROW IS NOT CONSTRAINED
C              ---------------------------------
              NCLQ = NCLQ + 1
              ICLQ(NCLQ) = IROW + K
            ENDIF
300       CONTINUE
C          ---------------------------------
C          INPUT DIAGONAL BLOCK TO SPARSPAK
C          ---------------------------------
          IF ( NCLQ .GT. 0 )  CALL INCLQ( NCLQ, ICLQ, S )
C          ---------------------------------
C          SKIP UNRELATED ITEMS IN CURRENT JOINT GROUP
C          ---------------------------------
          LRNG = IBUF(ITEMS+2)
          ITEMS = ITEMS + 2
          DO 220 JJ = 1, LRNG
            NODES = IBUF(ITEMS + 1)
            ITEMS = ITEMS + 6 + (NODES*(NODES+1))/2
220       CONTINUE
C          ---------------------------------
C          NUMBER OF SUBMATRICES FOR THE CURRENT JOINT
C          ---------------------------------
          CONRNG = IBUF(ITEMS+1)
          ITEMS = ITEMS + 1
C          ---------------------------------
C          ENTER NONZERO IN THE CONNECTED SUBMATRIX
C          IN ADDITION TO THE DIAGONAL SUBMATRIX
C          ---------------------------------
          DO 400 J = 1, CONRNG-1
            JOINT = IBUF(ITEMS + J)
C            ---------------------------------
C            DEGREE OF FREEDOM OF THE CONNECTED JOINT
C            ---------------------------------
            NCOLS = DOF(JOINT+1)- DOF(JOINT)
C            ---------------------------------
C            COMPUTE STARTING COLUMN NUMBER
C            ---------------------------------
            ICOL = DOF(JOINT) - 1
C            ---------------------------------
C            COMPUTE NONZERO POSITION COLUMN BY COLUMN
C            ---------------------------------
```

```
              DO 500 JX = 1, NCOLS
                DO 550 IX = 1, NROWS
                  IF ( ( MASK(ICOL+JX) .EQ. -1 ) .AND.
     1                 ( MASK(IROW+IX) .EQ. -1 ) ) THEN
C                      ------------------------
C                      THE CORRESPONDING VARIABLES
C                      ARE NOT CONSTRAINED
C                      ------------------------
                       CALL INIJ ( IROW+IX, ICOL+JX, S )
                  ENDIF
 550            CONTINUE
 500          CONTINUE
 400        CONTINUE
           ITEMS = ITEMS + 2*CONRNG - 1
C           ------------------------
C           END OF CURRENT JOINT GROUP
C           ------------------------
 200      CONTINUE
C         ---------------
C         END OF CURRENT RECORD
C         ---------------
 100    CONTINUE
C
      CALL IJEND ( S )
      STAGE = 70
      RETURN
C
1000    CONTINUE
C       ----------
C       ERROR HANDLING
C       ----------
      IERR = 1023
      IF ( MSGLVL .GE. 2 ) CALL EMSG
      RETURN
C
    END
```

```
C*****************************************************************************
C*****************************************************************************
C              GTFORC .... INPUT RIGHT HAND SIDE
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - TO RETIEVE RIGHT HAND SIDE FROM DATASET APPL.FORC.i.j
C      AND INPUT THOSE COMPONENTS  CORRESPONDING TO UNCONSTRAINED
C      VARIABLES TO SPARSPAK-A.
C
C    INPUT PARAMETERS
C      MASK - CONSTRAINT INFORMATION FOR EACH VARIABLE.
C      S    - INPUT TO SPARSPAK-A ROUTINES.
C
C    OUTPUT PARAMETER
C      S -  SPARSPAK-A OUPUT.
C
C    WORKING PARAMETERS
C      FBUF - A REAL OR DOUBLE PRECISION BUFFER OF SIZE BUFMAX.
C           THE ACTUAL TYPE IS AS DECLARED.
C
C    ERROR CODES -
C      1024 - INCORRECT EXECUTION SEQUENCE.
C
C    SUBROUTINE PROGRAMS -
C      QKINFO, GTRECF, EMSG.
C
C    SPRSPAK-A SUBROUTINES -
C      INBI.
C
C    CSM TESTBED DATASETS ACCESSES -
C      APPL.FORC.i.j.
C
C    ______________________________
C    REMARKS -
C      IT IS ASSUMED THAT THE ROWS CORRESPONDING TO DUMMY AND
C      CONSTRAINED VARAIBLES ARE INCLUDED IN THE DATA MATRIX.
C    ______________________________
C
C*****************************************************************************
C
      SUBROUTINE GTFORC ( FBUF, MASK,  S )
C
      INTEGER*4 MASK(1)
      DOUBLE PRECISION  FBUF(1), S(1)
C
C*****************************************************************************
C
      CHARACTER*40    LIBNAM
      CHARACTER*51    JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      CHARACTER*4     RTYPE
      INTEGER*4       IPRNTE, IPRNTS, MAXINT
      INTEGER*4       IDSN  , LDI   , NLEN  , NREC  , TRACE
      INTEGER*4       BUFMAX, MXUSED, MXREQD, STAGE
      INTEGER*4       MSGLVL, IERR, MAXCSM
      INTEGER*4       MAXDOF , NEQNS , NUMJNT
      REAL          RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1         TRACE
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1           JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      COMMON /PRBLEM/ MAXDOF , NEQNS , NUMJNT
C
C*****************************************************************************
C
      INTEGER*4 I, J, IROWS, LEN
C
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11   FORMAT( /5X, 'GTFORC - INPUT RIGHT HAND SIDE... ' )
C
      IF (( STAGE .LT. 70 ) .OR. ( IERR .NE. 0 ) ) GO TO 1000
C
C     __________________________
C      RETRIEVE RIGHT HAND SIDE FORM 'APPL.FORC.* '
```

```
C       -----------------------------
        CALL QKINFO ( APPLF )
C       -----------------------------
C        NOTE APPLY.FORC.. DOES NOT NECESSARILY EXIST
C       -----------------------------
        IF ( IERR .NE. 0 ) GO TO 900
        TRACE = TRACE + 10
        IROWS = 0
        DO 100 I = 1, NREC
          LEN = MIN0 ( NEQNS - IROWS, NLEN )
          IF ( LEN .GT. 0 ) THEN
C            ----------
C            READ NEXT RECORD
C            ----------
             CALL GTRECF ( I, FBUF, LEN )
             IF ( IERR .NE. 0 ) RETURN
C            -----------------------
C            RETRIEVE EACH ITEM IN CURRENT RECORD
C            -----------------------
             DO 200 J = 1, LEN
               IROWS = IROWS + 1
               IF ( MASK ( IROWS ) .EQ. -1 ) THEN
C                -----------------
C                THE VARIABLE IS NOT CONSTRAINED
C                -----------------
                 CALL INBI ( IROWS, FBUF(J), S )
               ENDIF
 200         CONTINUE
          ENDIF
 100    CONTINUE
        STAGE = 80
        RETURN
C
 900    CONTINUE
C       ---------------
C       RIGHTHAND SIDE DOES NOT EXIST
C       ---------------
        IF ( MSGLVL .GE. 3 ) WRITE ( IPRNTS, 21 )
 21     FORMAT( /10X, 'THERE IS NO APPLIED FORCE VECTOR')
        IERR = 0
        STAGE = 80
        RETURN
C
1000    CONTINUE
C       -------
C       ERROR HANDLING
C       -------
        IERR = 1024
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
      END
```

```
C*****************************************************************************
C=============================================================================
C           GTNUM5 ... INPUT NONZERO NUMERICS
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - TO RETRIEVE AND INPUT NUMERICAL NONZEROS OF THE
C       SYSTEM MATRIX. IN ADDITION, RIGHT HAND SIDE IS APPROP-
C       RIATELY ADJUSTED USING CONSTRAINTS AVAILABLE.
C
C     INPUT PARAMETERS
C       DOF  - AN INTEGER ARRAY OF SIZE EQUAL TO THE TOTAL NUMBER OF
C              JOINTS PLUS ONE.
C              IDOF(K) STORES THE STARTING EQUATION NUMBER FOR
C              JOINT K.  THE DEGREES OF FREEDOM FOR JOINT K IS
C              GIVEN BY IDOF(K+1) - IDOF(K).  THE TOTAL NUMBER
C              OF EQUATIONS IS EQUAL TO IDOF(NUMJNT+1) - 1.
C       MASK - THE LOCATIONS CORRESPONDING TO NONZEROR CONSTRAINTS
C              CONTAIN A POINTER TO THE NUMERIC VALUE IN FCON.
C              THE OTHER LOCATIONS INDICATE FREE OR CONSTRAINED
C              TO ZERO VARIABLES.
C       FCON - AN ARRAY OF CSIZE FLOATING-POINT CONSTRAINTS.
C       S    - STORAGE ARRAY FOR SPARSPAK-A.
C
C     WORKING PARAMETERS
C       FBUF - A REAL OR DOUBLE PRECISION BUFFER OF SIZE BUFMAX.
C              THE ACTUAL TYPE IS AS DECLARED.
C
C     ERROR CODES -
C       1028 - INCORRECT EXECUTION SEQUENCE.
C
C     SUBROUTINE PROGRAMS -
C       QKINFO, GTRECF, EMSG.
C
C     SPARSPAK-A ROUTINES -
C       INAIJ5, INBI.
C
C     CSM TESTBED DATASETS ACCESSES -
C       K.SPAR.*.
C
C     ____________________________________
C     REMARKS -
C       IT IS ASSUMED THAT THE VARIABLES ARE ORDERED IN THE
C       GIVEN ORDER OF THE JOINTS AND DEGREES.
C     ____________________________________
C
C*****************************************************************************
C
      SUBROUTINE GTNUM5 ( DOF, FBUF, MASK, FCON, S )
C
        INTEGER*4        DOF(1), MASK(1)
        DOUBLE PRECISION  FBUF(1), FCON(1), S(1)
C
C*****************************************************************************
C
        CHARACTER*40   LIBNAM
        CHARACTER*51   JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        CHARACTER*4    RTYPE
        INTEGER*4      IPRNTE, IPRNTS, MAXINT
        INTEGER*4      IDSN  , LDI   , NLEN  , NREC  , TRACE
        INTEGER*4      BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4      MSGLVL, IERR, MAXCSM
        INTEGER*4      MAXDOF , NEQNS , NUMJNT
        REAL         RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1          TRACE
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM,
     1          JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        COMMON /PRBLEM/ MAXDOF , NEQNS , NUMJNT
C
C*****************************************************************************
C
        INTEGER*4 CONRNG, I, II, ICOL, IROW, ISTRT, ITEMS,
     1         JGRPS, JOINT, M, MTXKNT, MYI, MYJ, NCOL,
     1         NROW, LEN  , NCOLS, NROWS
      DOUBLE PRECISION  COEF, BIX, BJX
C
      IF (  MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 )
 11    FORMAT( /5X, 'GTNUM5 - GET NONZERO NUMERIC... ' )
C
      IF (( STAGE .LT. 80 ) .OR. ( IERR .NE. 0 ) ) GO TO 1000
C
C      ------------------------
C      ACCESS EACH RECORD IN DATA SET 'K.SPAR.* '
C      ------------------------
      CALL QKINFO ( KSPAR )
      IF ( IERR .NE. 0 ) RETURN
      TRACE = TRACE + 10
      DO 100 I = 1, NREC
        LEN = NLEN
        CALL GTRECF ( I, FBUF, LEN )
        IF ( IERR .NE. 0 ) RETURN
C        ---------------------------
C        DETERMINE NUMBER OF JOINT GROUPS IN CURRENT RECORD
C        ---------------------------
        JGRPS = FBUF(1)
        ITEMS = 1
        DO 200 II = 1, JGRPS
C          -------------
C          GET NUMBER OF SUBMATRICES
C          -------------
          CONRNG = FBUF(ITEMS+1)
C          -----------
C          GET THE CURRENT JOINT
C          -----------
          JOINT = FBUF(ITEMS+2)
          IROW = DOF(JOINT) - 1
          NROWS = DOF(JOINT+1) - DOF(JOINT)
          ISTRT = ITEMS + 1 + CONRNG
C          ----------------------
C          RETRIEVE UPPER TRIANGULAR PART OF DIAGONAL SUBMATRIX
C          ----------------------
          NCOLS = NROWS
          DO 400 NCOL = 1, NCOLS
            MYJ = IROW + NCOL
            DO 500 NROW = 1, NROWS
              ISTRT = ISTRT + 1
              IF ( NROW .GT. NCOL ) GO TO 500
              COEF = FBUF(ISTRT)
              MYI = IROW + NROW
C              ----------------
C              RETRIEVE THE NONZERO CONSTRAINTS
C              ----------------
              IF ( MASK(MYI) .GT. 0 ) BIX = FCON(MASK(MYI))
              IF ( MASK(MYJ) .GT. 0 ) BJX = FCON(MASK(MYJ))
              IF ( MYI .EQ. MYJ ) THEN
                IF ( MASK(MYI) .NE. -1 ) THEN
C                  ----------------
C                  CHANGE DIAGONAL ELEMENT TO BE 1.0D0
C                  FOR CONSTRAINED ROW
C                  ----------------
                 COEF = 1.0D0
C                  ----------------
C                  ENTER NONZERO CONSTRAINT VALUE AS RHS
C                  ----------------
                 IF ( MASK(MYI) .GT. 0 )
     1             CALL INBI ( MYI, BIX, S )
                ENDIF
C                ---------------
C                INPUT DIAGONAL ELEMENT COEF
C                ---------------
                CALL INAIJ5 ( MYI, MYI, COEF, S )
              ELSE IF ((MASK(MYJ) .GT. 0 ) .AND.
     1             (MASK(MYI) .EQ. -1 )) THEN
                CALL INBI ( MYI, -COEF*BJX, S )
              ELSE IF ((MASK(MYI) .GT. 0 ) .AND.
     1             (MASK(MYJ) .EQ. -1 )) THEN
                CALL INBI ( MYJ, -COEF*BIX, S )
              ELSE IF ((MASK(MYI) .EQ. -1) .AND.
     1             (MASK(MYJ) .EQ. -1)) THEN
C                ---------------------
```

```
C                 INPUT COEF IN LOWER TRIANGULAR MATRIX
C                 ------------------------------------
                  CALL INAIJ5 ( MYJ, MYI, COEF, S )
                ENDIF
 500          CONTINUE
C             -----------------------
C             NEXT COLUMN IN DIAGONAL SUBMATRIX
C             -----------------------
 400        CONTINUE
C           ---------------------------
C           RETRIEVE OFF-DIAGONAL SUBMATRICES IN THE UPPER
C           TRIANGULAR PART OF THE SYSTEM STIFFNESS MATRIX
C           ---------------------------
          MTXKNT = CONRNG - 1
          IF ( MTXKNT .EQ. 0 ) GO TO 199
          ITEMS = ITEMS + 2
          DO 600 M = 1, MTXKNT
            JOINT = FBUF(ITEMS + M)
            ICOL = DOF(JOINT) - 1
            NCOLS = DOF(JOINT+1) - DOF(JOINT)
            DO 800 NCOL = 1, NCOLS
              MYJ = ICOL + NCOL
              DO 900 NROW = 1, NROWS
                ISTRT = ISTRT + 1
                COEF = FBUF(ISTRT)
                MYI = IROW + NROW
C                 -------------------
C                 RETRIEVE NONZERO CONSTRAINTS
C                 -------------------
                IF ( MASK(MYI) .GT. 0) BIX = FCON(MASK(MYI))
                IF ( MASK(MYJ) .GT. 0) BJX = FCON(MASK(MYJ))
C                 ---------------------
C                 INPUT COEF OR MODIFY RIGHT HAND SIDE
C                 ---------------------
                IF (( MASK(MYI) .EQ. -1 ) .AND.
     1              ( MASK(MYJ) .EQ. -1 )) THEN
C                   ------------------------
C                   ENTER COEF WITH SYMMETRIC POSITION
C                   IN LOWER TRIANGULAR TO SPARSPAK-A
C                   ------------------------
                  IF ( MYI .LT. MYJ )
     1              CALL INAIJ5 ( MYJ, MYI, COEF, S )
                  IF ( MYI .GT. MYJ )
     1              CALL INAIJ5 ( MYI, MYJ, COEF, S )
                ELSE IF ( (MASK(MYI) .GT. 0 ) .AND.
     1                  (MASK(MYJ) .EQ. -1) ) THEN
                  CALL INBI ( MYJ, -COEF*BIX, S )
                ELSE IF ( (MASK(MYJ) .GT. 0 ) .AND.
     1                  (MASK(MYI) .EQ. -1) ) THEN
                  CALL INBI ( MYI, -COEF*BJX, S )
                ENDIF
 900          CONTINUE
C             ----------
C             NEXT COLUMN
C             ----------
 800        CONTINUE
C           -------
C           NEXT SUBMATRIX
C           -------
 600      CONTINUE
C         ---------------------------
C         PROCESS THE NEXT JOINT GROUP IN THE CURRENT RECORD
C         ---------------------------
 199      ITEMS = ISTRT
 200      CONTINUE
C         ------
C         NEXT RECORD
C         ------
 100    CONTINUE
        STAGE = 90
        RETURN
C
1000    CONTINUE
C       ------
C       ERROR HANDLING
C       ------
        IERR = 1028
```

```
      IF ( MSGLVL .GE. 2 ) CALL EMSG
      RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C              SPACE .... CHECK AVAILABLE STORAGE
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - CHECK STORAGE REQUIRED AGAINST STORAGE AVAILABLE.
C
C    SUBROUTINE PROGRAMS -
C      EMSG.
C
C*****************************************************************************
C
      INTEGER FUNCTION SPACE ( IDUMMY )
C
        INTEGER*4     IDUMMY
C
        CHARACTER*40  LIBNAM
        CHARACTER*51  CDUMMY(7)
        INTEGER*4     MSGLVL , IERR  , MAXCSM
        INTEGER*4     BUFMAX, MXUSED, MXREQD, STAGE
C
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR , MAXCSM, CDUMMY(7)
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
C
C*****************************************************************************
C
        IF ( MXREQD .LE. MAXCSM ) THEN
          SPACE = 0
          RETURN
        ELSE
          SPACE = 1
          GO TO 100
        ENDIF
C
C         ------
C         ERROR HANDLING
C         ------
 100      CONTINUE
        IERR = 1001
        IF ( MSGLVL .GE. 2 ) CALL EMSG
        RETURN
C
      END
```

```
C*************************************************************************
C*************************************************************************
C                LIBOPN .... OPEN DATA LIBRARY ...
C*************************************************************************
C*************************************************************************
C
C     PURPOSE - THIS ROUTINE OPENS AN EXISTING LIBRARY RESIDENT
C       ON A DISKFILE OR MAIN STORAGE, AND CONNECTS IT TO A
C       LOGICAL DEVICE INDEX (LDI).  THE NAME OF THE LIBRARY
C       IS SPECIFIED BY PARAMETER LIBNAM.
C
C     PARAMETERS INITIALIZED -
C       LDI - LOGICAL DEVICE INDEX ASSIGNED TO THE EXTERNAL
C             DEVICE SPECIFIED BY LIBNAM.
C
C     ERROR CODES -
C       0    - NO ERROR.
C       1011 - UNSUCCESSFUL OPEN.
C       1012 - THE LOGICAL DEVICE NUMBER EXCEEDS THE MAXIMUM VALUE
C              OF 30.
C
C     GAL-PROCESSOR ENTRY POINTS -
C       LMOPEN, EMSG.
C
C*************************************************************************
C
      SUBROUTINE LIBOPN
C
C*************************************************************************
C
      CHARACTER*40  LIBNAM
      CHARACTER*51  JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      CHARACTER*4   RTYPE
      INTEGER*4     IPRNTE, IPRNTS, MAXINT
      INTEGER*4     MSGLVL, IERR  , MAXCSM
      INTEGER*4     IDSN  , LDI   , NLEN  , NREC  , TRACE
      INTEGER*4     ICPAD , STAGE
      REAL          RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM,
     1                JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
      COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1                TRACE
      COMMON /CSMCON/ ICPAD(3), STAGE
C
      INTEGER*4     LMOPEN
C
C*************************************************************************
C
      CHARACTER*10  LIBKEY
      INTEGER*4     LIMIT
C
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 11 ) LIBNAM
   11 FORMAT ( /5X, 'LIBOPN- OPEN ', A40 )
C
      IERR  = 0
C      ------------------------------
C      LIBKEY IS A STRING OF FORM 'MAINKEY/QUALIFIER'
C      MAXIMUM NUMBER OF CHARACTERS IS 10
C      ------------------------------
      LIBKEY = 'ROLD '
      LIMIT  = 0
      TRACE = 1000
      LDI = LMOPEN ( LIBKEY, 0, LIBNAM, LIMIT, TRACE )
C      ------------------------------
C      LDI RANGES FROM 1 THROUGH 30 FOR SUCCESSUL OPEN
C      ------------------------------
      IF (( LDI .LT. 1 ) .OR. ( LDI .GT. 30 )) GO TO 100
      STAGE = 10
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 21 ) JDFSET, KMAP,
     1                     KSPAR, CON, APPLF, APPLM, STATD
   21 FORMAT(/5X, 35HDATASETS TO BE ACCESSED:
     1      /5X, 35H
     1      /10X, A51,
     1      /10X, A51,
     1      /10X, A51,
     1        /10X, A51,
     1        /10X, A51,
     1        /10X, A51,
     1        /10X, A51 )
      RETURN
C
 100   CONTINUE
C      --------
C      ERROR HANDLING
C      --------
      IF ( LDI .LE. 0 ) IERR = 1011
      IF ( LDI .GT. 30) IERR = 1012
      IF ( MSGLVL .GE. 2 ) CALL EMSG
      RETURN
C
    END
```

```
C*****************************************************************************
C*****************************************************************************
C              QKINFO ... ANQUIRE DATASET ATTRIBUTES
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - ACQUIRE THE ATTRIBUTES OF A NAMED DATA SET.
C
C     INPUT PARAMETER -
C       DSNAME - NAME OF THE DATASET.
C
C     PARAMETERS UPDATED -
C       IDNS -  UNIQUE SEQUENCE NUMBER OF NAMED DATASET.
C       NLEN -  LOGICAL LENGTH (ITEMS) OF A RECORD.
C       RTYPE - RECORD TYPE.
C       NREC  - TOTAL NUMBER OF RECORDS IN THE DATASET.
C
C     ERROR CODES -
C       0    - NO ERROR.
C       2001 - DATASET DOES NOT EXIST.
C       2002 - NO RECORD EXISTS IN DATASET.
C       2003 - RECORD GROUP KEY IS UNDEFINED.
C       2004 - SEGMENTED RECORD GROUP NOTED.
C       2009 - RECORD LENGTH GREATER THAN BUFFER LENGTH
C
C     GAL-PROCESSOR ENTRY POINTS -
C       LMFIND, GMGEKA, GMGECY, EMSG.
C
C*****************************************************************************
C
      SUBROUTINE QKINFO ( DSNAME )
C
        CHARACTER*51  DSNAME
C
C*****************************************************************************
C
        CHARACTER*40  LIBNAM
        CHARACTER*51  CDUMMY(7)
        CHARACTER*4   RTYPE
        INTEGER*4     MSGLVL, IERR, MAXCSM
        INTEGER*4     IDSN  , LDI   , NLEN  , NREC  , TRACE
        INTEGER*4     BUFMAX, MXUSED, MXREQD, STAGE
C
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR, MAXCSM, CDUMMY(7)
        COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1              TRACE
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
C
        INTEGER*4      LMFIND
C
C*****************************************************************************
C
        CHARACTER*1  OP
        CHARACTER*12 RKEY
C
        INTEGER*4    IHI   , ILO   , MDIM
C
C       ---------------------------
C       OBTAIN THE SEQUENCE NUMBER OF DATASET DSNAME
C       MAXIMUM LENGTH OF DSNAME IS 51 CHARACTERS
C       ---------------------------
        TRACE = TRACE + 10
        IDSN = LMFIND ( LDI, DSNAME, TRACE )
        IF ( IDSN .EQ. 0 ) GO TO 100
C
C       ---------------------------
C       OP IS PRESENTLY A DUMMY ARGUMENT FOR BOTH
C       GMGEKA AND GMGECY.
C       ---------------------------
        OP = ' '
C       ---------------------------
C       RKEY CONTAINS THE RECORD KEY LIFTJUSTIFIED.
C       MAXIMUM LENGTH IS 12 CHARACTERS.
C       ---------------------------
        RKEY = 'DATA '
        TRACE = TRACE + 10
C       ----------------------------------------
```

```
C      RETRIEVE ATTRIBUTES RTYPE AND NLEN FOR RECORDS OF GIVEN KEY
C      ------------------------------------------
      CALL GMGEKA ( OP, LDI, IDSN, RKEY, RTYPE, NLEN, MDIM, TRACE )
      IF ( NLEN .EQ. 0 ) GO TO 200
      IF ( NLEN .GT. BUFMAX ) GO TO 500
C
C      -------------------------------
C      NUMBER OF RECORDS FOUND WITH GIVEN KEY
C      -------------------------------
      TRACE = TRACE + 10
      CALL GMGECY ( OP, LDI, IDSN, RKEY, NREC, ILO, IHI, TRACE )
      IF ( NREC .EQ. 0 ) GO TO 300
C
C      ----------------------------------------
C      NREC = IHI-ILO+1 FOR AN UNSEGMENTED RECORD GROUP
C      ----------------------------------------
      IF ( NREC .NE. (IHI-ILO+1) ) GO TO 400
      RETURN
C
C      --------
C      ERROR HANDLING
C      --------
100    CONTINUE
      IERR = 2001
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
200    CONTINUE
      IERR = 2002
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
300    CONTINUE
      IERR = 2003
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
400    CONTINUE
      IERR = 2004
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
500    CONTINUE
      IERR = 2009
      BUFMAX = NLEN
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
    END
```

```
C*****************************************************************************
C*****************************************************************************
C             GTRECI ... READ A RECORD FROM A DATASET
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - THIS ROUTINE READS A RECORD FROM A NAMED DATASET.
C       THE DATASET MUST BE OF TYPE INTEGER.
C
C     INPUT PARAMETERS -
C       RECNUM - RECORD CYCLE OF AN INDIVIDUAL RECORD.
C
C     OUTPUT PARAMETERS-
C       LEN  - THE NUMBER OF ITEMS CONTAINED IN THE RECORD.
C
C     WORKING PARAMETERS -
C       IBUF - A BUFFER OF MAXIMUM RECORD SIZE FOR READIN DATASETS
C            OF TYPE INTEGER.
C
C     ERROR CODES -
C       0    - NO ERROR.
C       2005 - RECORD TYPE IN THE DATASET IS NOT INTEGER.
C       2006 - ERROR IN GMGETN DETECTED BY LMERCD.
C
C     GAL-PROCESSOR ENTRY POINTS -
C       GMCORN, GMGETN, LMERCD, EMSG.
C
C*****************************************************************************
C
      SUBROUTINE GTRECI ( RECNUM, IBUF, LEN )
C
        INTEGER*4       RECNUM, IBUF(1), LEN
C
C*****************************************************************************
C
        CHARACTER*40  DUMMY1
        CHARACTER*51  CDUMMY(7)
        CHARACTER*4   RTYPE
        INTEGER*4       IDSN  , LDI   , NLEN  , NREC  , TRACE
        INTEGER*4       MSGLVL, IERR, DUMMY2
C
        COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1              TRACE
        COMMON /CSMUSR/ DUMMY1, MSGLVL, IERR, DUMMY2, CDUMMY(7)
C
        INTEGER*4       LMERCD
C
C*****************************************************************************
C
        CHARACTER*4  BUFTYP
        CHARACTER*12 OP, RKEY
        CHARACTER*24 RNAME
        INTEGER*4     IERROR, IGAP  , IHI   , ILO   , IOFF  , MDIM
C
C         ------------
C         DETECT TYPE MISMATCH
C         ------------
        IF ( RTYPE .NE. 'I ' ) GO TO 500
C         ----------------------------------------------
C         CONSTRUCT NAME 'RKEY.RECNUM:RECNUM' FOR AN INDIVIDUAL RECORD
C         MAXIMUM LENGTH IS 24 CHARACTERS: 12 FOR RKEY, 5 FOR EACH
C         RECNUM REPRESENTING HIGH AND LOW CYCLES.
C         ----------------------------------------------
        RKEY = 'DATA '
        ILO = RECNUM
        IHI = RECNUM
        CALL GMCORN ( RNAME, RKEY, ILO, IHI )
C
C         ------------------------------
C         OP ARGUMENT FOR GMGETx: 'MAINKEY/QUALIFIER'
C         MAXIMUM LENGTH IS 11: 4 FOR KEY AND 6 FOR QUALIFIER
C         ------------------------------
        OP = 'READ/LENGTH '
        BUFTYP = 'I '
        IGAP = 0
        IOFF = 0
        CALL GMGETN ( OP, LDI, IDSN, RNAME, BUFTYP, IBUF, LEN, MDIM,
     1            IGAP, IOFF, TRACE )
C       ---------------------------------
C       TEST ERROR CONDITION AFTER AN ERROR-SENSITIVE REFERENCE
C       TO THE I/O MANAGER
C       ---------------------------------
      IERROR = LMERCD ( IERROR )
      IF ( IERROR .NE. 0 ) GO TO 600
      RETURN
C
 500   CONTINUE
      IERR = 2005
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
 600   CONTINUE
      IERR = 2006
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C            GTRECF ... READ A RECORD OF TYPE REAL ...
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - THIS ROUTINE READS A RECORD FROM A NAMED DATASET.
C      THE DATASET MUST BE OF TYPE REAL OR DOUBLE PRECISION.
C
C    INPUT PARAMETERS -
C      RECNUM - RECORD CYCLE OF AN INDIVIDUAL RECORD.
C
C    OUTPUT PARAMETERS-
C      LEN  - THE NUMBER OF ITEMS CONTAINED IN THE RECORD.
C
C    WORKING PARAMETERS -
C      FBUF - A BUFFER OF MAXIMUM RECORD SIZE FOR READIN DATASETS
C           OF TYPE REAL OR DOUBLE PRECISION.  THE ACTUAL TYPE
C           IS AS DECLARED.
C
C    ERROR CODES -
C      0    - NO ERROR.
C      2007 - RECORD TYPE IN THE DATASET IS NOT REAL.
C      2008 - ERROR IN GMGETN DETECTED BY LMERCD.
C
C    GAL-PROCESSOR ENTRY POINTS -
C      GMCORN, GMGETN, LMERCD, EMSG.
C
C*****************************************************************************
C
      SUBROUTINE GTRECF ( RECNUM, FBUF, LEN )
C
        INTEGER*4     RECNUM, LEN
        DOUBLE PRECISION  FBUF(1)
C
C*****************************************************************************
C
        CHARACTER*40  DUMMY1
        CHARACTER*51  CDUMMY(7)
        CHARACTER*4   RTYPE
        INTEGER*4     IDSN  , LDI   , NLEN  , NREC  , TRACE
        INTEGER*4     MSGLVL, IERR, DUMMY2
C
        COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE,
     1            TRACE
        COMMON /CSMUSR/ DUMMY1 , MSGLVL, IERR, DUMMY2 , CDUMMY(7)
C
        INTEGER*4   LMERCD
C
C*****************************************************************************
C
        CHARACTER*4  BUFTYP
        CHARACTER*12 OP, RKEY
        CHARACTER*24 RNAME
        INTEGER*4    IERROR, IGAP  , IHI   , ILO   , IOFF  , MDIM
C
C        ------------
C        DETECT TYPE MISMATCH
C        ------------
        IF ((RTYPE .NE. 'D ') .AND. (RTYPE .NE. 'S ')) GO TO 500
C        ----------------------------------------
C        CONSTRUCT NAME 'RKEY.RECNUM:RECNUM' FOR AN INDIVIDUAL RECORD
C        MAXIMUM LENGTH IS 24 CHARACTERS: 12 FOR RKEY, 5 FOR EACH
C        RECNUM REPRESENTING HIGH AND LOW CYCLES.
C        ----------------------------------------
        RKEY = 'DATA '
        ILO = RECNUM
        IHI = RECNUM
        CALL GMCORN ( RNAME, RKEY, ILO, IHI )
C
C        ------------------------------
C        OP ARGUMENT FOR GMGETx: 'MAINKEY/QUALIFIER'
C        MAXIMUM LENGTH IS 11: 4 FOR KEY AND 6 FOR QUALIFIER
C        ------------------------------
        OP = 'READ/LENGTH '
        BUFTYP = 'D '
        IGAP = 0
      IOFF = 0
      CALL GMGETN ( OP, LDI, IDSN, RNAME, BUFTYP, FBUF, LEN, MDIM,
     1           IGAP, IOFF, TRACE )
C      ------------------
C      TEST ERROR CONDITION AFTER AN ERROR-SENSITIVE REFERENCE
C      TO THE I/O MANAGER
C      ------------------
      IERROR = LMERCD ( IERROR )
      IF ( IERROR .NE. 0 ) GO TO 600
      RETURN
C
 500   CONTINUE
      IERR = 2007
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
 600   CONTINUE
      IERR = 2008
      IF ( MSGLVL .GE. 3 ) CALL EMSG
      RETURN
C
     END
```

```
C*****************************************************************************
C*****************************************************************************
C            EMSG ... ERROR MESSAGE HANDLINE ROUTINE
C*****************************************************************************
C*****************************************************************************
C
C   PURPOSE - THIS ROUTINE IS USED TO HANDLE ERROR MESSAGES IN
C     SYSTEM-CSM WHICH INTERFACES SPARSPAK-A WITH CSM TESTBED
C     DATABASE.
C
C   PROGRAM SUBROUTINES -
C     EMSG0, EMSG1, DEMSG0
C
C*****************************************************************************
C
      SUBROUTINE EMSG
C
C*****************************************************************************
C
      CHARACTER*40 LIBNAM
      CHARACTER*51 CDUMMY(7)
      INTEGER*4   IPRNTE, IPRNTS, MAXINT
      INTEGER*4   MSGLVL, IERR  , MAXCSM
      REAL        RATIOS, RATIOL, TIME
C
C*****************************************************************************
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM, CDUMMY(7)
C
      INTEGER*4   LEVEL
C
      WRITE ( IPRNTE, 11 )
 11   FORMAT (/5X, 'EMSG - SYSTEM-CSM ERROR ... '  )
C
C     -------------------------
C     DETERMINE THE TYPE OF MODULE THAT CALLED EMSG,
C     AND CALL THE APPROPRIATE ERROR ROUTINE TO PRINT
C     THE ERROR MESSAGE
C     -------------------------
C
      IF ( IERR .GT. 2000 ) GO TO 1000
C
      LEVEL = (IERR - 1000)/10 + 1
      GO TO ( 100, 200, 300 ) , LEVEL
C
 100  CONTINUE
C     ------------
C     IERR RANGES FROM 1001 TO 1009
C     ------------
      CALL EMSG0
      RETURN
C
 200  CONTINUE
C     ------------
C     IERR RANGES FROM 1011 TO 1019
C     ------------
      CALL EMSG1
      RETURN
C
 300  CONTINUE
C     ------------
C     IERR RANGES FROM 1021 TO 1029
C     ------------
      CALL EMSG2
      RETURN
C
1000  CONTINUE
      LEVEL = (IERR - 2000)/10 + 1
      GO TO ( 1100, 1200 ) , LEVEL
C
1100  CONTINUE
C     ------------
C     IERR RANGES FROM 2001 TO 2009
C     ------------
      CALL DEMSG0
      RETURN
```

```
C
1200   CONTINUE
     RETURN
C
   END
```

```
C******************************************************************************
C******************************************************************************
C              EMSG0 ..... ERROR MESSAGES FOR ...
C******************************************************************************
C******************************************************************************
C
C     PURPOSE - THIS ROUTINE IS AN ERROR MESSAGE PRINTING
C       ROUTINE FOR THE MODULE SPACE.
C
C******************************************************************************
C
      SUBROUTINE EMSG0
C
C******************************************************************************
C
      CHARACTER*40  LIBNAM
      CHARACTER*51  CDUMMY(7)
      INTEGER*4     IPRNTE, IPRNTS, MAXINT
      INTEGER*4     MSGLVL , IERR  , MAXCSM
      INTEGER*4     BUFMAX, MXUSED, MXREQD, STAGE
      REAL          RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL,
     1           TIME
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR ,MAXCSM,CDUMMY(7)
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
C
C******************************************************************************
C
      INTEGER*4 IERROR
C
      IERROR = IERR - 1000
      GO TO ( 100, 200 ) , IERROR
C
 100   CONTINUE
      WRITE ( IPRNTE, 11 ) IERR, STAGE, MXREQD
 11    FORMAT (/10X, 35HSPACE - ERROR NUMBER            , I7
     1        /10X, 35HINSUFFICIENT STORAGE .            ,
     1        /10X, 35HTHE LAST STAGE COMPLETED IS       , I7
     1        /10X, 35HTO CONTINUE MAXCSM IS AT LEAST    , I7 )
      RETURN
C
 200   CONTINUE
      RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C              EMSG1 ..... ERROR MESSAGES FOR ...
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - THIS ROUTINE IS AN ERROR MESSAGE PRINTING
C        ROUTINE FOR MODULES: LIBOPN, GETJDF, GETDOF
C
C*****************************************************************************
C
      SUBROUTINE EMSG1
C
C*****************************************************************************
C
      CHARACTER*40  LIBNAM
      CHARACTER*51  CDUMMY(7)
      INTEGER*4     IPRNTE, IPRNTS, MAXINT
      INTEGER*4     MSGLVL , IERR  , MAXCSM
      REAL          RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM, CDUMMY(7)
C
C*****************************************************************************
C
      INTEGER*4 IERROR
C
        IF ( IERR .GT. 1012 ) GO TO 250
C
C         ----------------
C         ERROR FOR SUBROUTINE LIBOPN
C         ----------------
        IERROR = IERR - 1010
        GO TO ( 100, 200 ) , IERROR
C
 100      CONTINUE
C         ------
C         IERR = 1011
C         ------
        WRITE ( IPRNTE, 11 ) IERR
  11      FORMAT (/10X, 35HLIBOPN - ERROR NUMBER              , I7
   1           /10X, 35HCANNOT OPEN DATASET LIBRARY.         )
        RETURN
C
 200      CONTINUE
C         ------
C         IERR = 1012
C         ------
        WRITE ( IPRNTE, 22 ) IERR
  22      FORMAT (/10X, 35HLIBOPN - ERROR NUMBER              , I7
   1           /10X, 35HMAX LOGICAL DEVICE INDEX  = 30    ,
   1           /10X, 35HLDI RETURNED EXCEEDS THIS VALUE.     )
        RETURN
C
 250      CONTINUE
        IF ( IERR .GT. 1014 ) GO TO 450
C
C         ----------------
C         ERROR FOR SUBROUTINE GETJDF
C         ----------------
        IERROR = IERR - 1012
        GO TO ( 300, 400 ) , IERROR
C
 300      CONTINUE
C         ------
C         IERR = 1013
C         ------
        WRITE ( IPRNTE, 33 ) IERR
  33      FORMAT (/10X, 35HGETJDF - ERROR NUMBER              , I7
   1           /10X, 35HINCORRECT EXECUTION SEQUENCE.        )
        RETURN
C
 400      CONTINUE
C         ------
C         IERR = 1014
C         ------
```

```
       WRITE ( IPRNTE, 44 ) IERR
 44     FORMAT ( /10X, 35HGETJDF - ERROR NUMBER             , I7
  1            /10X, 35HDATASET DOES NOT HAVE ALL DATA.     )
       RETURN
C
 450    CONTINUE
       IF ( IERR .EQ. 1019 ) GO TO 900
       RETURN
C
 900    CONTINUE
C       ———————
C       IERR = 1019
C       ———————
       WRITE ( IPRNTE, 99 ) IERR
 99     FORMAT ( /10X, 35HGETDOF - ERROR NUMBER             , I7
  1            /10X, 35HINCORRECT EXECUTION SEQUENCE.       )
       RETURN
C
     END
```

```
C*****************************************************************************
C*****************************************************************************
C                 EMSG2 ..... ERROR MESSAGES FOR ...
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - THIS ROUTINE IS AN ERROR MESSAGE PRINTING
C        ROUTINE FOR MODULES: GTZERO, GTCOND, GETIJ, FTFORC,
C                     GTMOTI, GTNUM5.
C
C*****************************************************************************
C
      SUBROUTINE EMSG2
C
C*****************************************************************************
C
      CHARACTER*40  LIBNAM
      CHARACTER*51  CDUMMY(7)
      INTEGER*4     IPRNTE, IPRNTS, MAXINT
      INTEGER*4     MSGLVL , IERR  , MAXCSM
      REAL          RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL,
     1          TIME
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR ,MAXCSM, CDUMMY(7)
C
C*****************************************************************************
C
      INTEGER*4 IERROR
C
      IERROR = IERR - 1020
      GO TO ( 100, 200, 300, 400, 500, 600, 700, 800), IERROR
C
 100   CONTINUE
C      -------
C      IERR = 1021
C      -------
      WRITE ( IPRNTE, 11 ) IERR
 11    FORMAT (/10X, 35HGTZERO - ERROR NUMBER              , I7
     1        /10X, 'INCORRECT EXECUTION SEQUENCE '         )
      RETURN
C
 200   CONTINUE
C      -------
C      IERR = 1022
C      -------
      WRITE ( IPRNTE, 22 ) IERR
 22    FORMAT (/10X, 35HGTCOND - ERROR NUMBER              , I7
     1        /10X, 'INCORRECT EXECUTION SEQUENCE '         )
      RETURN
C
 300   CONTINUE
C      -------
C      IERR = 1023
C      -------
      WRITE ( IPRNTE, 33 ) IERR
 33    FORMAT (/10X, 35HGETIJ - ERROR NUMBER              , I7
     1        /10X, 'INCORRECT EXECUTION SEQUENCE '         )
      RETURN
C
 400   CONTINUE
C      -------
C      IERR = 1024
C      -------
      WRITE ( IPRNTE, 44 ) IERR
 44    FORMAT (/10X, 35HGTFORC - ERROR NUMBER              , I7
     1        /10X, 'INCORRECT EXECUTION SEQUENCE '         )
      RETURN
C
 500   CONTINUE
C      -------
C      IERR = 1025
C      -------
      WRITE ( IPRNTE, 55 ) IERR
 55    FORMAT (/10X, 35HGTMOTI - ERROR NUMBER              , I7
     1        /10X, 'INCORRECT EXECUTION SEQUENCE '         )
      RETURN
```

```
C
600     CONTINUE
C       --------
C       IERR = 1026
C       --------
       WRITE ( IPRNTE, 66 ) IERR
 66     FORMAT (/10X, 35HGTMOTI - ERROR NUMBER            , I7
   1          /10X, 'UNEXPECTED NONZERO CONSTRAINT VALUE'    )
       RETURN
C
700     CONTINUE
C       --------
C       IERR = 1027
C       --------
       WRITE ( IPRNTE, 77 ) IERR
 77     FORMAT (/10X, 35HGTMOTI - ERROR NUMBER            , I7
   1          /10X, 'ZERO ENTRY FOR A NONZERO CONSTRAINT OCCURS')
       RETURN
C
800     CONTINUE
C       --------
C       IERR = 1028
C       --------
       WRITE ( IPRNTE, 88  ) IERR
 88     FORMAT (/10X, 35HGTNUMi - ERROR NUMBER            , I7
   1          /10X, 'INCORRECT EXECUTION SEQUENCE '       )
       RETURN
C
     END
```

```
C*****************************************************************************
C*****************************************************************************
C          DEMSG0 ..... ERROR MESSAGES FOR DATASET ACCESSES
C*****************************************************************************
C*****************************************************************************
C
C     PURPOSE - THIS ROUTINE IS AN ERROR MESSAGE PRINTING
C       FOR MODULES ACCESSING DATASETS.
C
C*****************************************************************************
C
      SUBROUTINE DEMSG0
C
C*****************************************************************************
C
      CHARACTER*40  LIBNAM
      CHARACTER*51  CDUMMY(7)
      INTEGER*4     IPRNTE, IPRNTS, MAXINT
      INTEGER*4     MSGLVL , IERR  , MAXCSM
      INTEGER*4     BUFMAX, MXUSED, MXREQD, STAGE
      REAL          RATIOS, RATIOL, TIME
C
      COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
      COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM, CDUMMY(7)
      COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
C
C*****************************************************************************
C
      INTEGER*4  IERROR
C
        IF ( IERR .GT. 2004 ) GO TO 450
C
C       -----------------
C       ERROR FROM SUBROUTINE QKINFO
C       -----------------
        IERROR = IERR - 2000
        GO TO ( 100, 200, 300, 400 ) , IERROR
C
 100    CONTINUE
C       --------
C       IERR = 2001
C       --------
        WRITE ( IPRNTE, 11 ) IERR
  11    FORMAT (/10X, 35HQKINFO - ERROR NUMBER               , I7
   1          /10X, 35HLMFIND: CANNOT FIND DATASET.          )
        RETURN
C
 200    CONTINUE
C       --------
C       IERR = 2002
C       --------
        WRITE ( IPRNTE, 22 ) IERR
  22    FORMAT (/10X, 35HQKINFO - ERROR NUMBER               , I7
   1          /10X, 35HGMGEKA: RECORD DOES NOT EXIST.        )
        RETURN
C
 300    CONTINUE
C       --------
C       IERR = 2003
C       --------
        WRITE ( IPRNTE, 33 ) IERR
  33    FORMAT (/10X, 35HQKINFO - ERROR NUMBER               ,I7
   1          /10X, 35HGMGECY: RECORD GROUP KEY UNDEFINED.   )
        RETURN
C
 400    CONTINUE
C       --------
C       IERR = 2004
C       --------
        WRITE ( IPRNTE, 44 ) IERR
  44    FORMAT(/10X, 35HQKINFO - ERROR NUMBER               ,I7
   1         /10X, 35HGMGECY: SEGMENTED RECORD GROUP NOTED.   )
        RETURN
C
 450    CONTINUE
C       -------------------------
C       ERROR FROM SUBROUTINE GETRECI OR GTRECF
```

```
C       ----------------------------
        IERROR = IERR - 2004
        GO TO ( 500, 600, 700, 800, 900 ) IERROR
C
500      CONTINUE
C        --------
C        IERR = 2005
C        --------
        WRITE ( IPRNTE, 55 ) IERR
 55      FORMAT(/10X, 35HGETRECI - ERROR NUMBER                    ,I7
   1           /10X, 35HRECORD TYPE MISMATCH ...                 )
        RETURN
C
600      CONTINUE
C        --------
C        IERR = 2006
C        --------
        WRITE ( IPRNTE, 66 ) IERR
 66      FORMAT(/10X, 35HGETRECI - ERROR NUMBER                    ,I7
   1           /10X, 35HGMGETN: ERROR DETECTED BY LMERCD...       )
        RETURN
C
700      CONTINUE
C        --------
C        IERR = 2007
C        --------
        WRITE ( IPRNTE, 77 ) IERR
 77      FORMAT(/10X, 35HGETRECF - ERROR NUMBER                    ,I7
   1           /10X, 35HRECORD TYPE MISMATCH ...                 )
        RETURN
C
800      CONTINUE
C        --------
C        IERR = 2008
C        --------
        WRITE ( IPRNTE, 88 ) IERR
 88      FORMAT(/10X, 35HGETRECF - ERROR NUMBER                    ,I7
   1           /10X, 35HGMGETN: ERROR DETECTED BY LMERCD...       )
        RETURN
C
900      CONTINUE
C        --------
C        IERR = 2009
C        --------
        WRITE ( IPRNTE, 99 ) IERR, BUFMAX
 99      FORMAT(/10X, 35HQKINFO - ERROR NUMBER                     ,I7
   1           /10X, 35HBUFMAX MUST BE AT LEAST            ,I7)
        RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C              GETSOL ..... RETRIEVE TESTBED SOLUTION ...
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - RETRIEVE THE TESTBED SOLUTION.  ASSUMING THAT THE TESTBED
C      SOLUTION IS CORRECT, THE MAXIMUM RELATIVE ERROR IS THEN  COMPUTED
C      FOR EACH COMPOMENT IN THE SOLUTION VECTOR RETURNED BY SPARSPAK-A
C      SOLVER "SOLVE5".
C
C    INPUT PARAMETERS -
C      SOL  - THE LEADING NEQNS LOCATIONS OF THIS VECTOR CONTAIN
C           THE SOLUTION RETURNED BY SPARSPAK-A LINEAR SYSTEM
C           SOLVER.
C
C    WORKING PARAMETER -
C      FBUF - A REAL OR DOUBLE PRECISION BUFFER OF SIZE BUFMAX.
C           THE ACTUAL TYPE IS AS DECLARED.
C
C    OUTPUT PARAMETERS -
C      RATIO - THE MAXIMUM RELATIVE ERROR ENCOUNTERED.
C
C*****************************************************************************
C
      SUBROUTINE GETSOL ( FBUF, SOL, RATIO )
C
        DOUBLE PRECISION FBUF(1), SOL(1), RATIO
C
C*****************************************************************************
C
        CHARACTER*40    LIBNAM
        CHARACTER*51    JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        CHARACTER*4     RTYPE
        INTEGER*4       IPRNTE, IPRNTS, MAXINT
        INTEGER*4       IDSN  , LDI   , NLEN  , NREC  , TRACE
        INTEGER*4       MSGLVL, IERR  , MAXCSM
        INTEGER*4       MAXDOF   , NEQNS , NUMJNT
        REAL            RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMSPK/ IDSN  , LDI   , NLEN  , NREC  , RTYPE ,
     1              TRACE
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM,
     1              JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        COMMON /PRBLEM/ MAXDOF   , NEQNS , NUMJNT
C
C*****************************************************************************
C
        INTEGER*4 I, II, LEN, NITEMS, INDEX, MAXIND
        DOUBLE PRECISION  DELTAX, CSM, WEHAVE, CSMMAX
C
        WRITE ( IPRNTS, 11 )
  11    FORMAT (/5X, 40HGETSOL - COMPARE WITH TESTBED SOLN ...   )
C
        IF ( IERR .NE. 0 ) GO TO 300
C
C       ---------------------------
C       ACCESS RECORDS IN DATA SET 'STAT.DISP.* '
C       TO RETRIEVE NEQNS SOLUTIONS
C       ---------------------------
        CALL QKINFO ( STATD )
        IF ( IERR .NE. 0 ) GO TO 999
        TRACE = TRACE + 10
        RATIO = 0.0D0
        NITEMS = 0
        CSMMAX = 0.0D0
        DO 100 I = 1, NREC
          LEN = MIN0 ( NEQNS - NITEMS, NLEN )
          IF ( LEN .GT. 0 ) THEN
C             ---------
C             READ NEXT RECORD
C             ---------
              CALL GTRECF ( I, FBUF, LEN )
              IF ( IERR .NE. 0 ) RETURN
C             ---------------------
C             COMPUTE THE MAXIMUM RELATIVE ERROR
```

```
C          FBUP CONTAINS THE DATABASE SOLUTION
C          ---------------------------
          DO 200 II = 1, LEN
            NITEMS = NITEMS + 1
C             ---------------------------
C             GET THE COMPONENT WITH MAXIMUM MAGNITUDE
C             ---------------------------
            IF ( DABS (FBUF(II)) .GT. CSMMAX ) THEN
              CSMMAX = DABS (FBUF(II))
              MAXIND = NITEMS
            ENDIF
            DELTAX = DABS ( FBUF(II) - SOL(NITEMS) )
            IF ( FBUF(II) .NE. 0.0D0 )
     1         DELTAX = DELTAX/DABS(FBUF(II))
            IF ( DELTAX .GT. RATIO ) THEN
              RATIO = DELTAX
              INDEX = NITEMS
C               ---------------------------
C               SAVE THE PAIR WHICH CAUSES MAX REL ERR
C               ---------------------------
              CSM = FBUF(II)
              WEHAVE = SOL(INDEX)  .
            ENDIF
 200       CONTINUE
        ENDIF
 100    CONTINUE
C       -----------
C       SUMMARY .....
C       -----------
      IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 21 ) STATD, RATIO,
     1                 INDEX, CSM, WEHAVE
  21  FORMAT( /10X, 'MAX. REL ERR COMPARED TO ', A51,
     1     /10X, 'IS ', E14.7, '   IN COMPONENT', I5,
     1     /10X, 'CSM SOL = ', E21.14, '  WE HAVE ', E21.14 )
       RETURN
C
 300    CONTINUE
C       ---------------------------
C       ERROR HANDLING .... (NOT INCLUDED IN EMSG)
C       ---------------------------
       IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 31 )
  31    FORMAT (/10X, 35HGETSOL-INCORRECT EXECUTION SEQUENCE )
       RETURN
C
 999    CONTINUE
C       ---------------------------
C       ERROR HANDLING .... (NOT INCLUDED IN EMSG)
C       ---------------------------
       IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 91 ) STATD
  91    FORMAT( /10X, 'CANNOT FIND ', A51 )
       RETURN
C
      END
```

```
C*****************************************************************************
C*****************************************************************************
C              STATCS ......PRINT STATISTICS
C*****************************************************************************
C*****************************************************************************
C
C    PURPOSE - THIS ROUTINE PRINTS TIME AND STORAGE REQUIREMENTS OF
C      THE CURRENT RUN.
C
C*****************************************************************************
C
      SUBROUTINE STATCS
C
        CHARACTER*40    LIBNAM
        CHARACTER*51    JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        INTEGER*4       IPRNTE, IPRNTS, MAXINT
        INTEGER*4       MSGLVL , IERR  , MAXCSM
        INTEGER*4       DOF, BUF, MASK, KC, ICLQ, FCON, SPK
        INTEGER*4       BUFMAX, MXUSED, MXREQD, STAGE
        INTEGER*4       MAXDOF,  NEQNS , NUMJNT
        REAL              GZTIME, GCTIME, GIJTIM, GFTIME, GMTIME,GNTIME,
     1                  CSMTIM, CSMSTR
        REAL              RATIOS, RATIOL, TIME
C
        COMMON /CSMSYS/ IPRNTE, IPRNTS, MAXINT, RATIOS, RATIOL, TIME
        COMMON /CSMUSR/ LIBNAM, MSGLVL, IERR  , MAXCSM,
     1                  JDFSET, KMAP, KSPAR, CON, APPLF, APPLM, STATD
        COMMON /CSMMAP/ DOF, BUF, MASK, KC, ICLQ, FCON, SPK
        COMMON /CSMCON/ BUFMAX, MXUSED, MXREQD, STAGE
        COMMON /CSMDTA/ GZTIME, GCTIME, GIJTIM, GFTIME, GMTIME,GNTIME,
     1                  CSMTIM, CSMSTR
        COMMON /PRBLEM/ MAXDOF,  NEQNS , NUMJNT
C
C*****************************************************************************
C
        WRITE ( IPRNTS, 11 )
 11     FORMAT (/5X, 40HSTATCS - SYSTEM-CSM STATISTICS ...        )
C
        IF ( STAGE .GE. 20 ) GO TO 100
           WRITE (IPRNTS,22)
 22        FORMAT (/10X, 35HNO STATISTICS AVAILABLE.              )
           RETURN
C
 100     CONTINUE
        IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 33 ) MAXCSM
 33     FORMAT (/10X, 35HSIZE OF STORAGE ARRAY (MAXCSM)     , I10 )
        IF ( MSGLVL .GE. 2 ) WRITE ( IPRNTS, 44 ) NUMJNT,MAXDOF,NEQNS
 44     FORMAT (/10X, 35HNUMBER OF JOINTS                   , I10
     1          /10X, 35HMAX DEGREE OF FREEDOME PER JOINT   , I10
     1          /10X, 35HNUMBER OF EQUATIONS                , I10    )
        IF ( MSGLVL .GE. 3 ) THEN
          WRITE ( IPRNTS, 45 )
          WRITE ( IPRNTS, 46 ) DOF,BUF,MASK,KC,ICLQ,FCON,SPK
 45     FORMAT (/10X, 35HADDRESSES OF ARRARYS                 )
 46     FORMAT (/10X, 10HDOF       , I10
     1          /10X, 10HBUF       , I10
     1          /10X, 10HMASK      , I10
     1          /10X, 10HKC        , I10
     1          /10X, 10HICLQ      , I10
     1          /10X, 10HFCON      , I10
     1          /10X, 10HSPK       , I10    )
        ENDIF
c
        CSMSTR = MXREQD
        WRITE (IPRNTS, 133)  CSMTIM, CSMSTR
 133     FORMAT ( 10X, 35HTOTAL CSM-TIME REQUIRED              , F13.3
     1          /10X, 35HMAXIMUM CSM-STORAGE REQUIRED         , F10.0 )
        RETURN
C
      END
```

References

[1] *The Computational Structural Mechanics Testbed User's Manual.* January 1988.

[2] *Introduction to the Computational Structural Mechanics Testbed.* NASA Langley Research Center, September 1987. NASA TM 89096.

[3] *The Nominal-Record Global-Database Manager GAL-DBM.* Lockheed Palo Alto Research Laboratory, September 1986. LMSC-D766995.

[4] *SPAR Structural Analysis System Reference Manual Vol 3 - Demonstration Problems.* Engineering Information Systems. Inc., December 1978. NASA CR 158970-3.

[5] *SPAR Structural Analysis System Reference Manual (vols. 1 - 4).* Engineering Information Systems. Inc., December 1978. NASA CR 158970-1.

[6] E. C. H. Chu and J. A. George. A note on estimating the error in Gaussian elimination without pivoting. *ACM SIGNUM Newsletter*, 20:2–7, 1985.

[7] E. C. H. Chu, J. A. George, J. W-H. Liu, and E. G-Y. Ng. *User's guide for SPARSPAK-A: Waterloo sparse linear equations package.* Technical Report CS-84-36, University of Waterloo, Waterloo, Ontario, 1984.

[8] G. C. Everstine. *The Bandit computer program for the reduction of matrix bandwidth for NASTRAN.* Technical Report 3872, NSRDC, March 1972.

[9] C. A. Felippa. Architecture of a distributed analysis network for computational mechanics. *Computers and Structures*, 13:405–413, 1981.

[10] C. A. Felippa. *A Command Language for Applied Mechanics Processors, vols. 1-3.* November 1983. LMSC-D 78511.

[11] C. A. Felippa. *The Computational Structural Mechanics Testbed Architecture: Volume 1 - The Language.* October 1987. NASA CR-XXXX.

[12] C. A. Felippa. *The Computational Structural Mechanics Testbed Architecture: Volume 2 - Directives.* October 1987. NASA CR-XXXX.

[13] C. A. Felippa. *The Computational Structural Mechanics Testbed Architecture: Volume 3 - The Interface.* October 1987. NASA CR-XXXX.

[14] C. A. Felippa. *The Computational Structural Mechanics Testbed Architecture: Volume 5 - The Input-Output Manager DMGASP.* October 1987. NASA CR-XXXX.

[15] C. A. Felippa. *The Global Database Manager EZ-GAL.* November 1982. LMSC-D 766995.

[16] J. A. George. An automatic one-way dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 17:740–751, 1980.

[17] J. A. George and J. W-H. Liu. Algorithms for matrix partitioning and the numerical solution of finite element systems. *SIAM J. Numer. Anal.*, 15:297–327, 1978.

[18] J. A. George and J. W-H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15:1053–1069, 1978.

[19] J. A. George and J. W-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[20] J. A. George and J. W-H. Liu. The design of a user interface for a sparse matrix package. *ACM Trans. on Math. Software*, 5:134–162, 1979.

[21] J. A. George and E. G-Y. Ng. *User's guide for SPARSPAK-B: Waterloo sparse constrained linear least squares package.* Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1984.

[22] J. W-H. Liu. An adaptive general sparse out-of-core Cholesky factorization scheme. *SIAM J. Sci. Stat. Comput.*, 8:585–599, 1987.

[23] J. W-H. Liu. *A Collection of Routines for an Implementation of the Multifrontal Method.* Technical Report CS-87-10, Dept of Computer Science, York University, 1986.

[24] J. W-H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. on Math. Software*, 11:141–153, 1985.

[25] J. W-H. Liu. *The Multifrontal Method and Paging in Sparse Cholesky Factorization.* Technical Report CS-87-09, Dept of Computer Science, York University, 1987.

[26] J. W-H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. on Math. Software*, 11:141–153, 1985.

[27] J. W-H. Liu. On the storage requirement in the out-of-core multi-frontal method for sparse factorization. *ACM Trans. on Math. Software*, 12, 1986.

[28] J. W-H. Liu and A. H. Sherman. Comparative analysis of the Cuthill-McKee and reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numer. Anal.*, 13:198–213, 1976.

[29] M. E. Regelbrugge and M. A. Wright. *The CSM Testbed Matrix Processors - Internal Logic and Dataflow Descriptions.* Lockheed Palo Alto Research Laboratories, January 1988. Preliminary Draft.

[30] M. A. Wright, M. E. Regelbrugge, and C. A. Felippa. *The Computational Structural Mechanics Testbed Architecture: Volume 4 - The Global-Database Manager GAL-DBM.* October 1987. NASA CR-XXXX.