

PM 4
P5

AN INVESTIGATION INTO DIRECT SPARSE MATRIX
SOLUTION SCHEMES IN THE FINITE ELEMENT METHOD

M.W. EASTMAN.

B.Sc.Eng. (Civil Engineering)

Cape Town

July 1987

Submitted to the University of Cape Town

in partial fulfillment for the degree of

Master of Science in Engineering

The University of Cape Town has been given
the right to reproduce this thesis in whole
or in part. Copyright is held by the author.

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

DECLARATION

I, Michael Wayne Eastman submit this thesis in partial fulfillment of the requirements for the degree of Master of Science in Engineering. I affirm that this is my original work and that it has not been submitted in this or similar form for a degree at any University.

SIGNED

Signed by candidate

DATE

..... 3 AUGUST 1987

ABSTRACT

The application of the finite element method invariably involves the solution of large systems of sparse linear algebraic equations. The solution of these systems often represents a significant or even dominant component of the total solution time. Various sparse matrix techniques and strategies have been developed to reduce the time and cost of solving these equations. These techniques exploit both the zero-nonzero structure of the matrix problem and the manner in which the actual numerical components of the problem are computed. This thesis describes some of the direct methods, including the banded, skyline or profile, wavefront and hypermatrix schemes. The relative merits of each of these schemes are also indicated with respect to the number of arithmetical operations, data structure organization, secondary storage requirements and implementation strategy.

The second section of this thesis discusses the implementation of an equation solution package for application in the finite element method. Initially a partitioning scheme for a wavefront solver was investigated but due to problems encountered and the increasing complexity of the code, it was decided to use an alternative method. A Cholesky decomposition method with a hypermatrix data storage scheme was then investigated and developed. The equation solution method was developed using a virtual paging scheme as implemented by the DAS package, and a module of general hypermatrix management routines. Finally, the package was implemented and tested in the NEW NOSTRUM development at the University of Cape Town. Suggestions for further developments are briefly discussed.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Mr. J. Vos of the Applied Mechanics Research Unit, University of Cape Town, for his guidance and constructive criticism.

Thanks is also expressed to Professor J.B. Martin, Director of the Applied Mechanics Research Unit, and my teachers and peers in the Research Unit who in a cheerful manner gave more than their due. Thanks to Miss N. Ihle for the typing of this thesis.

I wish to record my appreciation to the Applied Mechanics Research Unit and the Council for Scientific and Industrial Research for their financial support.

Finally, I wish to thank my family and friends for their encouragement.

TABLE OF CONTENTS

Title Page	i
Declaration	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
Preface	ix

CHAPTER ONE: INTRODUCTION

1.1 Characteristics of the system of equations in the finite element method	1-2
1.1.1 Sparsity	1-2
1.1.2 Bandedness	1-3
1.1.3 Symmetry	1-4
1.1.4 Positive-Definiteness	1-4
1.1.5 Pivoting	1-5
1.2 Requirements of a linear equation solver for structural problems	1-5
1.2.1 Solution guaranteed for problems of any practicable size	1-5
1.2.2 Accuracy of solution sufficient for engineering purposes	1-5
1.2.3 Efficient operation	1-6
1.2.4 Modular data structure	1-6
1.2.5 Modular program system	1-7
1.2.6 Portability	1-7
1.2.7 Simple interface to package	1-7
1.3 Solving a system of linear equations in the finite element method	1-7
1.3.1 Assembly of the global stiffness matrix	1-8
1.3.2 Boundary conditions and reactions	1-8
1.4 Scope and limitations of this work	1-8

CHAPTER TWO: CLASSIFICATION OF SOLUTION SCHEMES

2.1 Direct solution schemes	2-1
2.1.1 Triangularisation (e.g. Gaussian elimination)	2-1
2.1.2 Factorisation (e.g. Crout, Cholesky decomposition)	2-2
2.1.3 Remarks	2-3
2.2 Iterative solution schemes	2-4
2.2.1 When iterative methods should be considered	2-5
2.3 Data Structures for direct methods	2-6
2.3.1 Banded storage schemes	2-6
2.3.2 Skyline storage schemes	2-8
2.3.3 Frontal storage schemes	2-11
2.3.4 Hypermatrix storage schemes	2-13

CHAPTER THREE: DATA HANDLING SCHEMES FOR THE DIRECT SOLUTION METHODS

3.1 Virtual memory management	3-1
3.1.1 A data pool and paging scheme	3-1
3.1.2 Hierarchical data structures	3-3
3.2 Partitioning the data	3-4
3.3 Partitioning schemes	3-6
3.3.1 Individual row (or column) partitioning	3-6
3.3.2 Partitioning by rows	3-6
3.3.3 Partitioning by columns	3-7
3.3.4 Restrictions of partitioning methods	3-8
3.4 Hypermatrix schemes	3-9
3.4.1 Restrictions on problem size	3-10
3.4.2 Hypermatrix concepts	3-11
3.4.3 Organisation of hypermatrix calculations	3-14
3.4.4 Advantages of hypermatrix schemes for sparse matrices ..	3-15
3.4.5 Hypermatrix and hypertable management routines	3-16

CHAPTER FOUR: A PARTITIONED FRONTAL SOLUTION SCHEME

4.1 The frontal solution scheme	4-1
4.1.1 General description of the frontal solution scheme	4-1
4.1.2 The frontal solution algorithm	4-3
4.1.3 Discussion of the frontal solution technique	4-5
4.2 Solution with a partitioned frontal solver	4-6
4.2.1 The PREFRONT routine	4-8
4.2.2 The FRONTAL routine	4-8
4.2.3 The RESOLUTION routine	4-12
4.2.4 Selection of front partition and equation buffer length	4-13
4.2.5 Treatment of suppressed and prescribed displacements ..	4-14
4.2.6 Frontal partitioning data structure	4-14
4.3 Difficulties encountered with the implementation	4-15
4.4 Implementation of a frontal solver with a hypermatrix scheme	4-19
4.5 Discussion and conclusions	4-19

CHAPTER FIVE: THE CHOLESKY HYPERMATRIX EQUATION SOLUTION SCHEME

5.1 The Cholesky solution procedure for symmetric matrices	5-2
5.2 Cholesky algorithm for hypermatrices	5-4
5.3 Assembly of the global stiffness matrix	5-9
5.3.1 Concept of static condensation	5-9
5.3.2 Implementation of the assembly of the global stiffness matrix	5-12
5.3.3 Numerical example of the assembly of the global stiffness matrix	5-13
5.4 The equation solution routines	5-18
5.4.1 Equation solution routines description	5-18
5.4.2 Structure chart for equation solution routines	5-20
5.4.3 The CHESS suite: A sample problem	5-23
5.5 Discussion	5-29

CHAPTER SIX: SUMMARY AND CONCLUSIONS

6.1 Software development strategy	6-1
6.1.1 Software design	6-1
6.1.2 Software implementation	6-3
6.2 Summary	6-4
6.3 Suggestions for further improvements	6-6
6.3.1 Unsymmetric matrices	6-6
6.3.2 Further means of improving efficiency	6-6
6.3.3 Renumbering algorithms	6-8
6.3.4 Concurrent processing	6-9

CHAPTER SEVEN: REFERENCES

APPENDIX A: COURSEWORK PASSED	A-1
APPENDIX B: EQUATION SOLUTION ROUTINES DESCRIPTION	B-1
APPENDIX C: EQUATION SOLUTION ROUTINES SOURCE LISTING	C-1
APPENDIX D: A SAMPLE PROBLEM	D-1
APPENDIX E: HYPERMATRIX ROUTINES DESCRIPTION	E-1

PREFACE

This thesis discusses the investigation of current schemes for the solution of large systems of sparse matrices in the finite element method and the subsequent development and implementation of an equation solution program suite. The program suite, CHESS - Cholesky Hypermatrix Equation Solution Scheme, consists of several program modules and uses the Cholesky decomposition algorithm which accesses a hypermatrix data management package, called DAS - Dynamic Allocation System.

Chapter 1 describes the systems of linear equations arising in the finite element method and the requirements for the solution of these equations. At the end of the chapter the scope of this work in solving such a system of equations is outlined.

Chapter 2 classifies the solution methods which may be used in the solving of the system of equations in the finite element method. The two main mathematical algorithms, triangularisation and factorization, and the data structures generally used for the implementation of direct solution methods are described.

Chapter 3 considers the need to partition the global stiffness matrix and right-hand side vector(s) due to the limited primary memory available for the solution of large problems. The concepts of data management and various methods of data partitioning are presented. Hypermatrix schemes and their suitability to the data handling problem are discussed.

Chapter 4 discusses the investigation of the frontal solution technique. The frontal solution method is presented briefly as a preview to the investigation of a published partitioned frontal solution scheme. The difficulties encountered with the implementation of the partitioned frontal scheme are outlined as well as the reasons for adopting the hypermatrix data management scheme for the final equation solution package.

Chapter 5 describes the development of the Cholesky Hypermatrix Equation Solution Scheme. The Cholesky procedure and its implementation with a hypermatrix data management scheme is discussed, as well as the method of assembling the global stiffness matrix. The routines of the equation solution package are described and a sample program for the use of the package is presented.

Chapter 6 discusses the software development strategy employed, as well as problems encountered and experience gained during this project. Finally, the summary and conclusions are presented and several suggestions for possible future work are made.

Chapter 7 contains a number of literature references about equation solution techniques, software design and the development of interface routines for equation solution packages.

Appendix A presents the coursework completed in the partial fulfillment for the Masters Degree in Engineering.

Appendix B describes in detail the FORTRAN routines in the Cholesky Hypermatrix Equation Solution Scheme package.

Appendix C contains a FORTRAN listing of the source code for the routines in the Cholesky Hypermatrix Equation Solution Scheme.

Appendix D presents a sample problem and compares the results with the finite element package ABAQUS.

Appendix E describes in detail the FORTRAN routines in the hypermatrix module which access the DAS package.

CHAPTER ONE

INTRODUCTION

This chapter describes the system of linear equations arising in the use of the finite element method and the requirements for the solution of these equations. At the end of the chapter the scope of this work in solving such a system of equations is outlined.

The finite element method is now firmly established as a tool of wide applicability, and has been used with remarkable success in a variety of mathematical and engineering problems. The use of the method typically involves solving one or more large sparse systems of linear algebraic equations using direct or iterative methods [1]. Latest developments in the finite element method have resulted in this set of equations increasing in complexity due to the types of elements used, constraint equations, boundary conditions and complex model topology.

The system of equations in the finite element method have the general form

$$Kx = F \quad (1.1)$$

where K is the structural stiffness matrix, F the global load vector and x the unknown displacements. The solution of this system of equations often represents a large percentage of the total computational effort in applying the method, approximately 60%, and therefore the method of solving these equations is critical to an

efficient solution. Whilst there may be several ways in which the operations of input/output and generation of stiffness, stress matrices and applied load vectors may be performed, the optimal approach will only produce a marginal saving in both computer storage and solution costs. However the equation solution scheme adopted can affect computational time and storage requirements considerably.

Various sparse matrix techniques and strategies have been developed to reduce the cost of solving these equations. These techniques exploit both the zero-nonzero structure of the matrix problem and the manner in which the actual numerical components of the problem are typically computed.

1.1 Characteristics of the System of Equations in the Finite Element Method

In this section the assumptions made about the special features of the coefficient matrix K of the system of equations, for the purpose of this work, are outlined. These assumptions generally hold for structural analysis problems, and allow one to obtain efficiency in computation.

1.1.1 Sparsity

The set of equations is sparsely populated, see figure 1.1, meaning that only nonzero parts of the matrix need be stored. The sparseness arises from the connectivity of the elements.

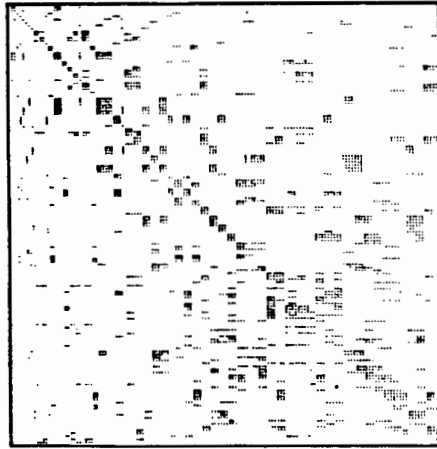


Figure 1.1: Large Sparse Matrix. Each dot indicates a nonzero element.

1.1.2 Bandedness

The coefficient matrix may be assumed to be banded, see figure 1.2, due to the organisation and numbering of nodes within the finite element problem. The bandedness implies that there are large sections of the global stiffness matrix which have zero matrix coefficients. These large regions may be exploited in the data structure used to store the global stiffness matrix.

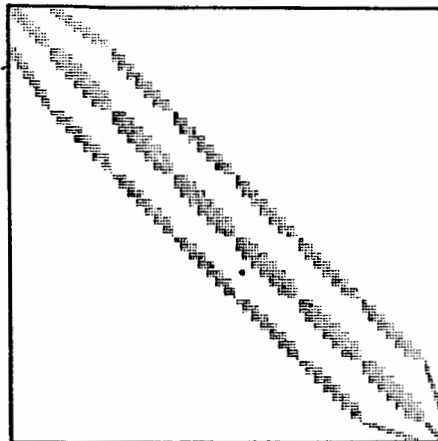


Figure 1.2: Banded Symmetric Matrix. Each dot indicates a nonzero element.

1.1.3 Symmetry

The set of equations is symmetric, see figure 1.2. This property is important because for such matrices K always has a triangular factorization $U^T U$, where U is upper triangular, see figure 1.3.

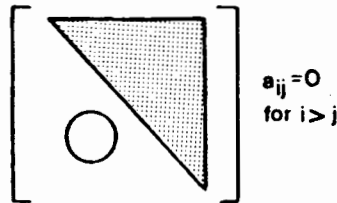


Figure 1.3: Upper triangular matrix

This allows us to process only those terms on and above the leading diagonal of the global stiffness matrix. During the triangularisation (eg. Gauss elimination) or factorization (eg. Cholesky decomposition) process, the matrix remains symmetric throughout the elimination procedure. In this way all terms below the diagonal may be omitted and this produces a large saving in computer storage with only half the global stiffness matrix required. While it is true that some finite element applications lead to unsymmetric problems, it is fair to say that the majority of finite element applications lead to symmetric positive-definite systems. Appropriate changes to the matrix schemes for unsymmetric problems should be made, e.g. the Crout algorithm must replace the Cholesky algorithm.

1.1.4 Positive-Definiteness

Given a real quadratic form $C = X^T K X$, K is said to be positive-definite if $C > 0$ for all $(X_1, \dots, X_n) \neq (0, \dots, 0)$. The structural stiffness

matrix is assumed to be positive-definite due to the physical properties of the engineering problems that are to be analysed.

1.1.5 Pivoting

The process of interchanging rows so that one has a 'large' diagonal element is called pivoting. The use of large pivots for Gaussian elimination is necessary to obtain reliable results. In the finite element method there is no need for pivoting during processing to find a large pivot, due to the physical properties of engineering problems. Thus the diagonal pivots may be used as they occur and symmetry is retained within the matrix.

1.2 Requirements of a Linear Equation Solver for Structural Problems

1.2.1 Solution Guaranteed for Problems of any Practicable Size

Commonly, problems need to be solved having as many as 1 000 to 20 000 degrees of freedom and a quite general sparsity pattern of the structural stiffness matrix. Current packages should be able to solve such problems without resorting to wasteful emergency procedures such as re-cycling of data or artificial division into substructures.

1.2.2 Accuracy of Solution Sufficient for Engineering Purposes

On computers where only 6 to 8 decimal places are available for single-precision floating point numbers, the provision of reliable bounds on numerical error is essential. A complete double precision version of the program should be available, from the calculation of the element stiffness matrices through the equation solution to handle cases when the accuracy of a single precision solution is insufficient.

As the omission of necessary supports or the input of faulty structural data can lead to the formation of a singular structural stiffness matrix, simple tests should be provided to detect such situations during the solution process.

1.2.3 Efficient Operation

In large problems the population density of the structural stiffness matrix may be less than 5 per cent [12]. In an efficient program, this sparsity should be utilized in the following complementary ways [12]:-

1. In the data structure, by storing as few zero elements as possible.
2. In the calculations, by avoiding most of the arithmetical operations on the stored zero elements.

The inner loops of the main calculations or I/O routines should remain sufficiently simple so that they can be efficiently written in assembly language to save computation time [12].

Most large computer installations are presently operating in multi-programming mode. It is therefore neither economical nor advisable that the program plus working data should occupy the entire primary memory of the computer. The program itself should be organised such that it occupies as little primary memory as possible [12].

1.2.4 Modular Data Structure

Great benefits result from arranging the data on secondary storage and in primary memory as records of fixed size, commonly called pages. Simple commands should be provided to transfer these pages between secondary storage and the primary memory.

1.2.5 Modular Program System

For example, a resolution capability should be able to be performed separately from the triangularization of the stiffness matrix, thus permitting cheaper solutions later with new loading conditions.

1.2.6 Portability

The program package should be primarily written in a simple standardized language. FORTRAN 77 seems to be the current language which is most generally applicable for practical engineering purposes. Direct access data transfer, operations on partial words and the inner loops of important matrix operations may be written in assembly language but equivalent FORTRAN routines should be provided for portability.

1.2.7 Simple Interface to Package

The package should have a module of well defined interface routines which may be called by the user application program. This layer of software should provide storage management services, insulate the user from the complicated data structures used by lower level subroutines, and provide a convenient means of communication between the user and the package.

1.3 Solving a System of Linear Equations in the Finite Element Method

Due to the nature of the finite element method there are several details which increase the complexity of the system of equations and their solution.

1.3.1 Assembly of the Global Stiffness Matrix

Due to the size of some of the engineering problems, sometimes 1 000 to 20 000 degrees of freedom, it is not possible to load the full matrix of coefficients into memory. Therefore various methods of assembling only nonzero, or as few zero elements as possible, have been developed.

The methods include implicit assembly such as the frontal solution scheme [2,6] or explicit sparse matrix schemes [12, 13, 18, 25, 35].

1.3.2 Boundary Conditions and Reactions

Coupled with the assembly problem we have the problem of prescribed or suppressed boundary conditions and the problem of retrieving the reactions at these particular nodes. The assembly of the matrix is affected because generally columns need to be zeroed and the right hand side adjusted before or during the solution process and the original equations for these degrees of freedom need to be retained for calculating the reactions after the solution process.

1.4 Scope and Limitations of this Work

The development of the NOSTRUM research code at the University of Cape Town was originally based on the program PLANET [5], which had an incore frontal solver. Since then the solver has been changed to an out of core skyline solver, but there are still severe restrictions on the use of the program due to the equation solution scheme.

This work served as an investigation into equation solution techniques and the implementation of a generally applicable equation solution

package for the finite element method. Initially the intention was to enhance a partitioned wavefront solution scheme with more modular data structures. To this end:-

- > An investigation of the frontal solution scheme described by Hinton and Owen [2] was conducted and the program PLANE (for plane stress/strain problems) was implemented on the UNIVAC 1100. Test examples were run and results compared with those published.
- > Next the partitioned wavefront solution algorithm published by Beer and Haas [6] was implemented on the UNIVAC. Several attempts were made to run examples which had been tested with the PLANE program, but intermediate results were incorrect and the program would not run to completion.
- > To obtain more of a background to the Beer and Haas code and their use of variables, the original frontal solution program of Irons [4] was implemented and tested. The examples compared favourably with those in the paper.
- > Several further attempts at debugging and executing the code of Beer and Haas were made, without success. Letters requesting further documentation and an original copy of the source code and some examples were sent, but no reply was received. At this stage it was decided to drop the partitioned frontal solution scheme.
- > In order to gain some experience with the use of hypermatrix routines, it was decided to implement a hypermatrix scheme in the

frontal solution of the program PLANE. Hypermatrix routines which used the DAS package [17] were implemented and tested independently and then implemented into the frontal solution scheme in the PLANE program. Due to the expected inefficiency of this scheme and its increasingly complex data structures it was decided to implement a more simple and well structured solution method, the Cholesky decomposition procedure with a hypermatrix data management scheme.

- > Having already implemented the hypermatrix routines, development began on the Cholesky Hypermatrix Equation Solution Scheme. Initially an incore Cholesky procedure was developed and tested. Once the incore procedure had been verified, work continued on the implementation of the 2 level Cholesky hypermatrix scheme.
- > The technique of static condensation to assemble the global stiffness matrix into various hypermatrices, depending on the global degree of freedom constraints, was then implemented along with general hypermatrix multiplication routines.
- > Following this a module of interface routines were developed which allowed the easy implementation of the package with an application program.
- > Finally, the package was transferred onto the Applied Mechanics Research Unit's MicroVax II and embedded and tested in the NEW NOSTRUM and the results compared with equivalent ABAQUS runs.

CHAPTER TWO

CLASSIFICATION OF SOLUTION SCHEMES

The numerical solution of systems of equations may be performed with either direct or iterative methods [1]. This chapter presents a classification of the equation solution methods by describing the mathematical algorithms and the data structures generally used for the implementation of several direct solution schemes.

2.1 Direct Solution Schemes

Generally a solution method (mathematical algorithm) is referred to by the name of an originator to whom the method is commonly attributed. However, instead of this, two main solution strategies may be defined [33], the triangularization methods (e.g. Gaussian elimination) and factorization methods (e.g. Crout, Cholesky decomposition). Although it can be shown that the factorization and triangularization methods are closely related in a mathematical sense [1], the computational steps and sequence of data retrieval, in a computer program, may be different and could play an important part in any comparison. It can be shown [33] that the greatest difference relates to the I/O operations and data handling schemes rather than the arithmetical effort required.

2.1.1 Triangularization method (Eg. Gaussian Elimination)

Consider the equation

$$K x = F \quad (2.1)$$

where K is a square positive-definite global stiffness matrix, F is a right-hand side load vector and x is an unknown vector(s). A series of row transformations are performed, resulting in an upper triangular coefficient matrix. This process may be represented mathematically as

$$(LK) x = (LF) \quad (2.2)$$

where L is a lower triangular matrix. Once this form is achieved, the solution proceeds through a simple back-substitution operation.

2.1.2 Factorization Method (Eg. Crout, Cholesky Decomposition)

In this method, the stiffness matrix is factorized (decomposed) into a number of matrices. The general case is given by

$$K = LDU \quad (2.3)$$

where L is a unit lower triangular matrix, D a nonsingular diagonal matrix and U is the transpose of L . Using equation (2.3), eq. (2.1) now becomes

$$LDUx = F \quad (2.4)$$

The solution vector is obtained by the usual three stage process

$$Lz = x \quad (\text{forward substitution}) \quad (2.5)$$

$$y = D^{-1} z \quad (\text{scaling}) \quad (2.6)$$

$$Ux = y \quad (\text{back substitution}) \quad (2.7)$$

In our case where K is symmetric positive-definite we have

$$U = L^T \quad (2.8)$$

and it is only necessary to compute and store one of the matrices L and U . In the Cholesky decomposition, the square root of the diagonal matrix D is obtained and multiplied by U so that

$$K = U^{*T} U^* \quad (2.9)$$

where

$$U^* = D^{\frac{1}{2}} U \quad (2.10)$$

2.1.3 Remarks

For both the triangularization and factorization (decomposition) procedures, the following holds:

- > It is possible to solve for several right hand sides simultaneously;
- > No additional storage is required for the L , D and U matrices. They can occupy the same storage which K occupied originally.
- > By saving the triangular matrices it is possible to solve for new sets of right hand sides without further triangularization or factorization.
- > Both methods may be combined with any of the data storage schemes, to be described in Section (2.3). In particular, both methods, and their derivatives, can be generalized to be hyper-row or hyper-matrix schemes.
- > Both methods require the same computational effort.

Kamel [33] therefore concludes that the relative superiority of one scheme over the other can only be determined, if at all, on the basis of their demand on computer resources - mainly core space and I/O operations. The choice therefore becomes one of the data handling strategy that is to be implemented.

2.2 Iterative Solution Schemes

Iterative methods are potentially useful in the solution of large systems of equations, although few programs currently utilize such methods. Iterative methods have the advantage of low storage requirements and possible fast convergence in certain cases, but their rate of convergence may be unpredictable [33]. The low storage requirements of the iterative solutions may become more important with large three dimensional solid problems, where storage is of prime importance, particularly if the current trend of increasingly faster central processors, coupled with relatively stagnant secondary storage technology, continues. The new technology of array processors, super computers and transputers allows concurrent processing and parallelism, but the programming for these types of machines is extremely complex. Since this work is primarily concerned with direct methods, the following iterative methods are merely mentioned as the most prominent approaches today:-

- 1) Gauss-Seidel Method
- 2) Block Relaxation Methods
- 3) Conjugate Gradient Method and
- 4) Dynamic Relaxation Method.

2.2.1 When Iterative Methods should be Considered

Guidelines as to when iterative methods should be considered instead of direct methods, have been published [34].

Iterative methods are used primarily when K is large and sparse. If K is sparse (say it has p nonzero elements per row), then one iteration requires about pn operations for a matrix of order n . Thus I iterations require about pIn operations and, for comparison with Gauss elimination for full matrices, iteration is cheaper if the number I of iterations to achieve satisfactory accuracy satisfies

$$I < \frac{n^2}{3p} \quad (2.11)$$

If K is a band matrix of width β (small compared to n), then the work for Gauss elimination is about $\beta^2 n$, while I iterations require work of about $2\beta nI$. Thus, for iteration to pay off by reducing arithmetic, the number I of iterations to achieve satisfactory accuracy for a band matrix must satisfy

$$I < \frac{\beta}{2} \quad (2.12)$$

For matrices derived from partial differential equations, the bandwidth is usually about $\sqrt{n}$, but the band is itself sparse so that p is 4 or 6 or so. In this situation, the work for Gauss elimination is n^2 while the work for I iterations is perhaps $6In$. Thus, iteration methods should be considered if they require less than $n/6$ iterations for satisfactory accuracy. However, the overriding consideration might be the memory required, which is about n^2 for Gauss elimination and pn for

an iteration method. For these problems one may have n in the range of 100 to 1 000 000 with 10 000 a not unusual case.

2.3 Data Structures for Direct Methods

This section briefly describes the data structures that the various direct solution methods use in storing the sparse global stiffness matrix. The major concern in a solution scheme is the method used to take advantage of the sparsity of the global stiffness matrix. Basically there are two ways of handling the sparsity, see figure 2.1.

1. In the data structure, by excluding large areas of zero matrix coefficients.
2. In the computation, by excluding all arithmetic operations on zeros as far as possible.

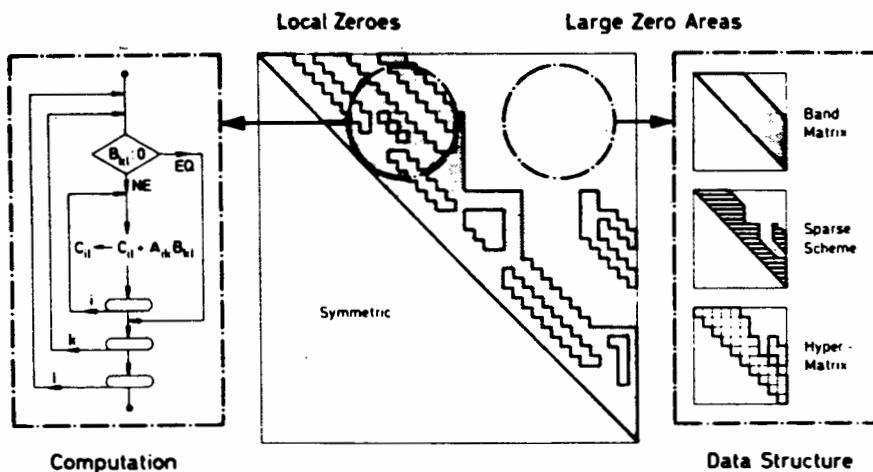


Figure 2.1: Basic methods of handling sparse matrices

2.3.1 Banded Storage Schemes

The most common and well known strategy is to order equations so that the stiffness matrix K has a small bandwidth β , where

$$\beta(K) = \max_{K_{ij} \neq 0} |i-j|. \quad (2.13)$$

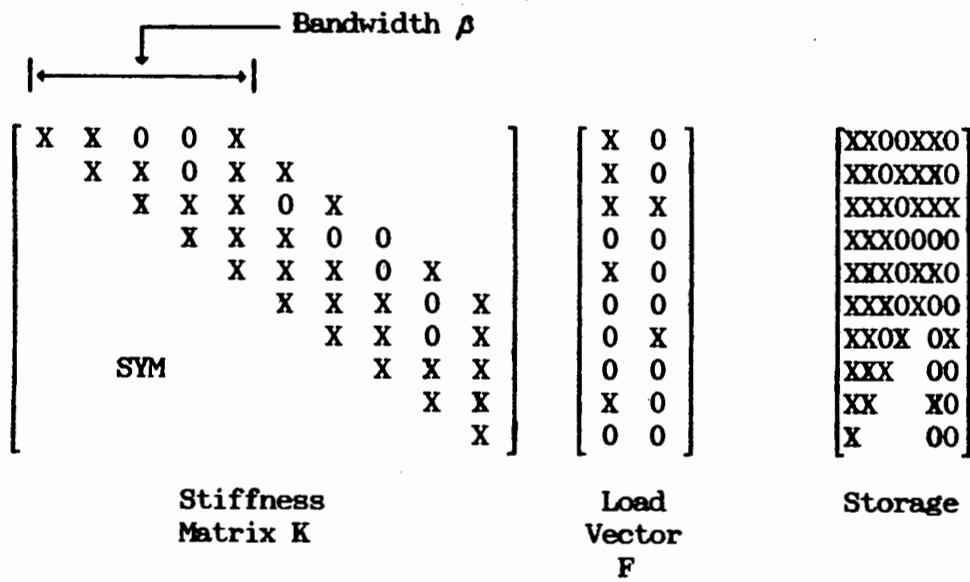


Figure 2.2: Banded Stiffness Matrix and Storage

Implicit in this practice is the assumption that K , for purposes of storage and/or computation, is subsequently to be regarded as having a potentially dense band. Although orderings which minimize bandwidth are often far from optimal in the least-fill or least-arithmetic sense, they often represent a reasonable compromise between optimality in the above complexity senses and convenient coding and storage management. Minimum or near minimum fill (or arithmetic) orderings characteristically lead to triangular factors whose nonzero components are scattered throughout the matrix, and a relatively sophisticated data structure and program are required to exploit the zeros. On the other hand, codes for solving banded systems, both in-core and out-of-core versions, are widely available [25, 35].

Reliable algorithms for producing orderings having a small bandwidth are available [24], while algorithms for finding near minimum fill or arithmetic orderings are both less reliable and less widely known.

Orderings which yield small bands are particularly attractive if secondary storage is to be used. However, the cost of the re-ordering process may exceed the resulting savings [12].

For in-core linear equation solvers, the amount of storage required for U is usually no more than $(\beta+1)n$ storage locations. The usual estimate for the number of multiplicative operations required to factor an n by n band matrix with bandwidth β is $\theta_{\beta} = \beta^2 n/2$ [10]. However, linear equation solvers seldom completely ignore zeros within the band. A check for a zero "multiplier" is inexpensive and routinely performed. The estimate θ_{β} is often a severe over-estimate for the amount of arithmetic actually performed.

2.3.2 Skyline Storage Scheme

Except for finite element applications involving quite regular meshes and/or simple elements (having no edge or interior nodes), there will generally be considerable variation in the individual column height within the banded matrix and it is natural to exploit this variation in both storage and computation.

In the skyline method the storage arrangement of matrix K is the so-called 'skyline', 'profile', 'envelope', or 'variable bandwidth' method [11,26]. The symmetric matrix K is stored in a linear array as a string of 'active' columns of the upper triangle (or, equivalently, rows of the lower triangle). The active portion of each column is bounded by the diagonal and the furthest nonzero element. The skyline storage scheme offers a reasonable compromise between organizational simplicity and computational efficiency.

The storage arrangement of the coefficient matrix K is best illustrated by a simple example. Consider the (6x6) symmetric matrix in figure 2.3.

$$K = \begin{bmatrix} k_{11} & & & & & \\ & k_{22} & & & & \\ & & 0 & & & \\ & & & k_{33} & & \\ & & & & k_{34} & \\ & & & & & k_{44} \\ & & & & & & k_{55} \\ & & & & & & & k_{16} \\ & & & & & & & & 0 \\ & & & & & & & & & 0 \\ & & & & & & & & & & k_{46} \\ & & & & & & & & & & & k_{56} \\ & & & & & & & & & & & & k_{66} \end{bmatrix}$$

SYM

Figure 2.3: Skyline of symmetric matrix

in which entries outside the skyline template are not shown. The matrix in figure 2.3 is stored as a 15-word string K :

$K: k_{11}, k_{22}, k_{13}, 0, k_{33}, k_{24}, k_{34}, k_{44}, k_{55}, k_{16},$
 $0, 0, k_{46}, k_{56}, k_{66}$

This array is complemented by a $(n+1)$ -word array $LOCA$ containing diagonal location pointers:

$LOCA: 0, 1, 2, 5, 8, 9, 15$

For a general matrix, $LOCA(I+1)$ points to k_{ii} in array K for $i=1, \dots, n$, whereas $LOCA(1)$ contains zero.

This storage scheme is attractive for several reasons. It is straightforward to verify that any fill suffered by K is confined to its profile. Thus, K can be factored "in place", with U replacing K for the Cholesky algorithm. In many contexts the storage scheme is

highly appropriate in the sense that U has few zeros within its profile. In particular, many of the algorithms for producing small profile and/or band orderings [24] yield orderings for which the profile of U has no zeros, unless cancellation occurs. An incore linear equations package which exploits this storage scheme, and which provides many of the facilities needed in finite element applications, is contained in [13], while an out-of-core skyline solution scheme is described in [21].

Let ω_i be defined by

$$\omega_i = |\{k | k > i \text{ and } f_k \leq i\}| \quad (2.14)$$

The number ω_i is simply the number of columns in the profile which intersect row i , and is often referred to as the number of "active" columns at the i -th stage of the elimination. This is a natural designation since ω_i is the number of equations which have been transformed by the i -th step or are about to be transformed at the i -th step.

It can be shown [10] that the number of multiplications and divisions required to factor A into $U^T U$ if we exploit zeros outside the profile is

$$\theta_p = \sum_{i=1}^{n-1} \omega_i (\omega_i + 3) / 2 \quad (2.15)$$

It can also be shown that θ_p is always smaller than θ_β [10]. Thus, by exploiting zeros which are within the band but also outside the

profile, the arithmetic may be reduced considerably. This also illustrates why band solver codes, even though they store the matrix as though the band is full, routinely check for zero multipliers; by doing so, they only perform Θ_p operations.

2.3.3 Frontal Storage Scheme

When secondary storage must be used, an alternative to the explicit assembly of the global stiffness matrix, (eg. band and skyline storage schemes), is the implicit assembly by the frontal solution schemes [2,4].

These schemes have two novel aspects. Firstly, as far as the actual factorization is concerned, only the active equations are in main storage. At the beginning of the i -th step of the factorization, the only rows and columns of the matrix which are in main storage are those which correspond to equations which have been modified by previous steps in the factorization. The second feature is that the assembly phase and the factorization are carried out in tandem. The element stiffness matrices are added to K one by one, with variables eliminated as soon as all elements containing that variable have been assembled. Since the elements are assembled in some specified order, and the variables eliminated as soon as possible, a particular ordering is imposed.

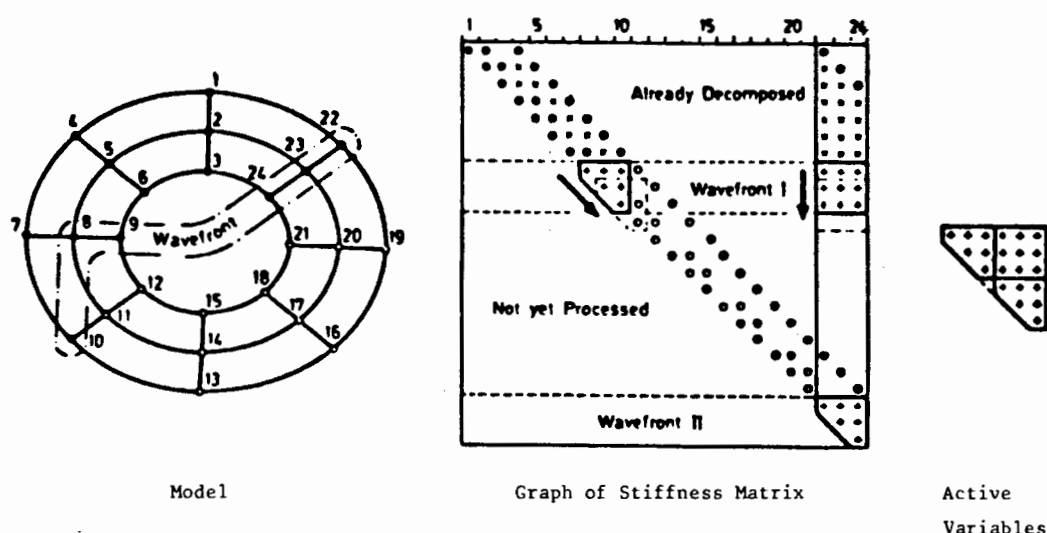


Figure 2.4: Simple Wavefront Example

At the i -th step of the factorization, ω_i is the current "frontwidth", and at most $\omega_i(\omega_i+1)/2$ storage locations are required to store enough of the matrix to proceed with step i of the factorization. Since the main reason for employing such schemes is to reduce main storage demands, when using this scheme one wants an ordering which minimizes $\bar{\omega}$, where $\bar{\omega} = \max_i \omega_i$. This contrasts with the ordering requirement if an in-core solver which exploits the profile is used, since storage requirements are minimized if the ordering minimizes $\sum_{i=1}^n \omega_i$. Fortunately, ordering algorithms designed to approximately minimize one quantity also seem to minimize the other, at least for finite element problems. Several element reordering algorithms are available in the literature [27].

2.3.4 Hypermatrix Storage Schemes

The main motivation for these schemes is the efficient use of storage and convenient data management. The basic idea is to partition K into a set of compatible s by s blocks or pages called submatrices, where s is chosen so that the amount of primary memory available is some convenient multiple of s^2 . Since K is sparse, some of the s by s pages will be all zeros. An (n/s) by (n/s) pointer matrix is held in primary memory, with each pointer component either pointing to the current location of the corresponding page, if the page contains nonzeros, or else is zero.

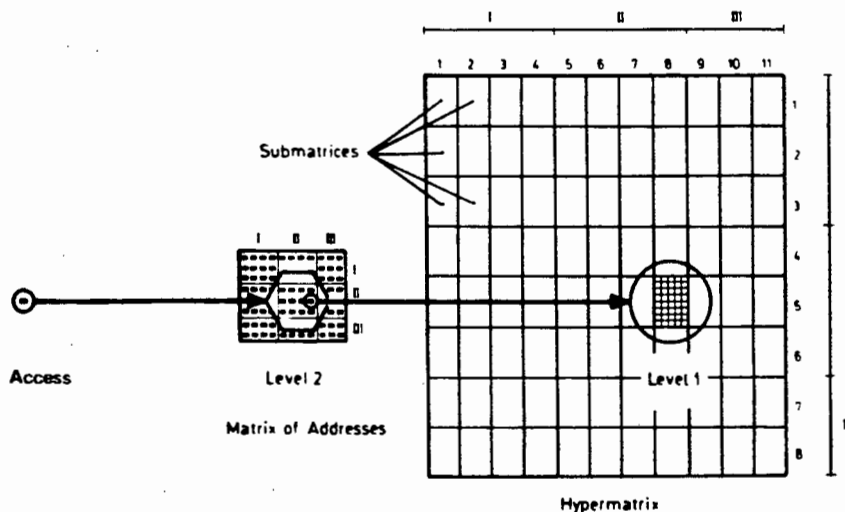


Figure 2.5: Two-level hypermatrix storage

This storage management scheme obviously entails a certain amount of overhead, but it is a viable alternative to the other out-of-core solution schemes such as band or frontal methods. The book-keeping and all other logic in ASKA [12] for a medium sized problem only accounted for 8% of the total computation time for a UNIVAC 1108, and only 25% on a CDC 6600. One advantage is that the actual matrix operations involve simple data structures as only square or rectangular arrays are involved, and thus the data management allows for a well structured solution scheme to be developed.

CHAPTER THREE

DATA HANDLING SCHEMES FOR DIRECT SOLUTION METHODS

In this chapter the need to partition the global stiffness matrix and right hand side vector(s) due to limited available memory space is considered. Large data entities may be partitioned into several smaller ones which can be stored and retrieved on secondary storage. The main considerations of these schemes are to reduce the memory requirements and minimize I/O operations during the execution of the solution algorithm. The basic concepts of this data partitioning and memory management for application in direct solution algorithms are described.

3.1 Virtual Memory Management

A severe limitation of the FORTRAN programming language is that additional memory space cannot be allocated while a program is executing, because the dimensions of arrays containing data must be specified at compile time. To overcome this limitation programs utilize database management schemes. This section describes the Dynamic Allocation Scheme (DAS) [17] which provides a virtual memory management facility for user application programs.

3.1.1 A Data Pool and Paging Scheme

The DAS package is a suite of FORTRAN routines that implements a virtual memory system. The DAS-package uses a paging scheme to administer a virtual memory space, which is divided into a specified

number of fixed-size blocks, commonly called pages. These pages are records in a direct access disk file on secondary storage. Before being used, such a file must first be specially formatted and is then known as a DYNA-file. A DYNA-file consists of a specified number of fixed-size records, with each record containing a particular page of the virtual memory space.

The DAS-package also administers a primary memory pool, which consists of a "static" area, which contains the control information, and "dynamic" area, which contains the data pool. The data pool is divided into a number of fixed-size slots. The size of each slot and the size of every record of the DYNA-files currently in use are the same.

The DAS-package has routines which can allocate space for a new array on a particular page of the virtual memory system, as well as to release it again for re-use when it is no longer required. Whenever the application program needs to access an array which has previously been allocated it invokes a routine in the DAS-package which will swap this particular page with the array into a slot of the memory pool and return its starting address in the memory pool to the calling routine.

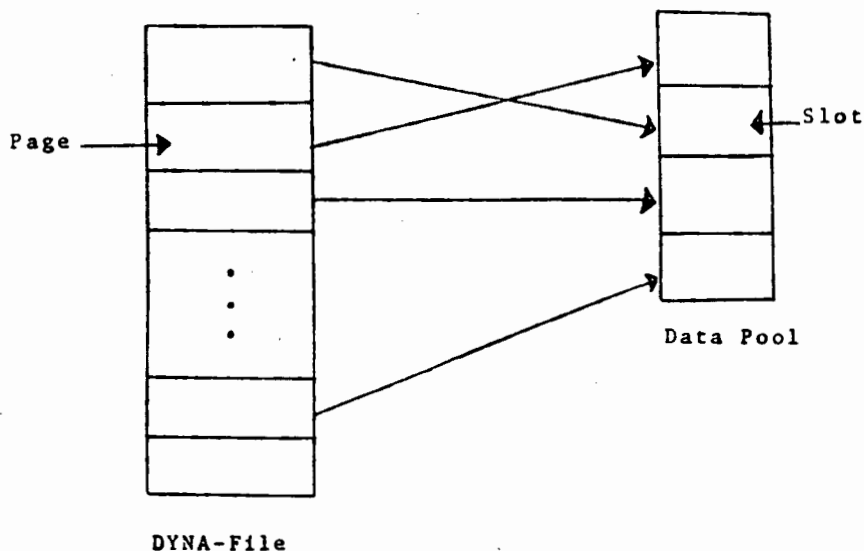


Figure 3.1: The Virtual address space of the DAS-package

The DAS-package includes programmed strategies to determine which one of the pages currently in the memory pool should be swapped out to the disk file in order to make room for the requested page. Once the requested page is swapped into the memory pool, the package returns the starting address of the array in the memory pool to the application program. The program uses this address to retrieve data values from the array, or to store new ones in it.

The largest contiguous space that can be allocated by the DAS-package is equal to the actual page size minus one word, in which the DAS-package stores certain control information for every page of the virtual memory.

3.1.2 Hierarchical Data Structures

Although the DAS-package is a dynamic allocator that implements a virtual memory system, it has one severe limitation. It cannot allocate space for a data entity which exceeds the currently defined page size. This severely limits the maximum dimensions of data entities, such as arrays and tables, which are to be stored in virtual memory. However, the DAS-package does facilitate the creation of hierarchical data structures by using the indirect addressing mechanism provided by pointers, as well as by using vectors of pointers to reference other vectors containing either data values or some more pointers.

Figure 3.6 illustrates an n-level hierarchical data structure, which is characterized by a base pointer which points to a vector of pointers at level 1. Each entry in the level 1 vector points in turn to a

different vector of pointers at level 2. This process is repeated down to level $n-1$. Each entry in the level $n-1$ vectors points to a vector of actual data.

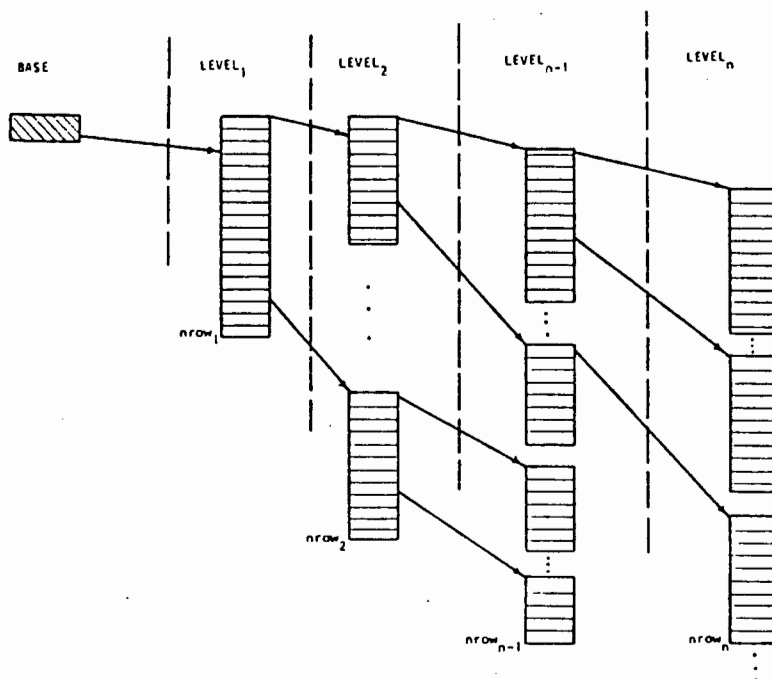


Figure 3.2: Hierarchical Data Structures

3.2 Partitioning the data

When an equation solution scheme is developed for the finite element method the memory requirement of the program is usually directly dependent on the problem size. When solving small problems, this presents no difficulty as the global stiffness matrix and right hand side vector(s) can be stored in the primary memory of the computer. However, for large systems of equations the memory requirement of the solution algorithm may easily exceed the primary memory available storage available.

The only way these large systems of equations may be solved is to make use of secondary storage, such as tapes or disks. Magnetic tapes are sequential devices with very large capacity but are used primarily for backup facilities. Disks are random access devices and are well suited for implementing partitioning schemes that require random access to data.

A program can only process data that is located in primary memory. This requires that the large data entities, such as the global stiffness matrix and right hand side vector(s), be partitioned into smaller data entities each of which must fit into the primary memory. These entities will reside on secondary storage and must be brought into primary memory only when it is required. The algorithm must now ensure that each entity is copied from secondary storage to primary memory before it is actually operated upon. Once the algorithm has modified a particular entity, it must be copied back to its proper place on secondary storage before the next partitioned data entity can be copied into primary memory.

The partitioning of data means the solution algorithm will have to be modified as well in order to operate on these partitioned entities, instead of the original one. Thus, the complexity of the solution algorithm escalates due to the constraints imposed by the limited amount of available memory and because of the data handling aspects involved when dealing with data entities that are too large to fit into primary memory as a whole.

3.3 Partitioning Schemes

In this section we consider the methods of partitioning the global stiffness matrix by rows (or columns) into blocks of data which may be swapped in and out of primary memory.

3.3.1 Individual row (or column) partitioning

The data may be partitioned into single rows (or columns). The solution algorithm is then accomplished by single row combinations, see figure 3.1. However, this method has the disadvantage of excessive I/O activity [33], due to the swapping of rows in and out of primary memory during Gauss elimination.

$\begin{bmatrix} X & X & X & & \\ & X & 0 & X & \\ & & X & 0 & X \\ & & & X & X \\ & & & & X \end{bmatrix}$	$\begin{bmatrix} \text{XXX} \\ \text{XOX} \\ \text{XOX} \\ \text{XX} \\ \text{X} \end{bmatrix}$	Block 1
		Block 2
		Block 3
		Block 4
		Block 5
Stiffness Matrix	Storage	

Figure 3.3: Individual Row Partitioning

3.3.2 Partitioning by rows

Each partition contains a number of rows, (figure 3.4), and is read or written with a single I/O instruction. This method may be combined with row by row solution algorithms, eg. Gauss elimination. This method of partitioning is particularly suited to band solution schemes

as the bandwidth is known for a particular problem and therefore the block or record size may be fixed for the particular solution.

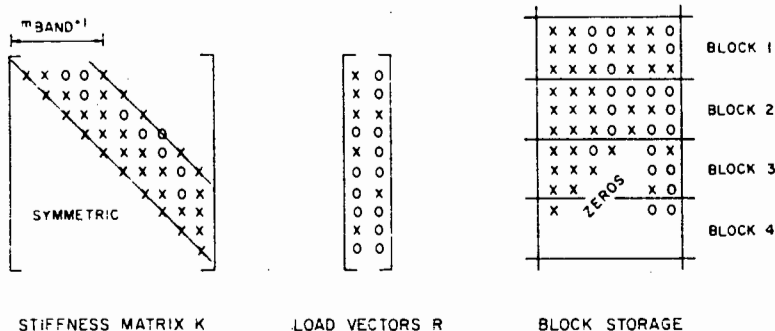


Figure 3.4: Partitioning by rows

3.3.3 Partitioning by columns

This method of partitioning is particularly attractive as a system of compacted column storage may be included in the scheme. This enables the exclusion of zero coefficients above the first non-zero coefficient in a particular column. This method is referred to as the skyline storage technique and is generally used for the storage of the global stiffness matrix for Cholesky decomposition.

Figure 3.5 shows the block partitioning for the compacted storage scheme. The data in a block of the coefficient matrix consists of both the stiffness coefficients and the address of the diagonal terms, as shown in figure 3.5. These are the addresses within the block of the diagonal terms, and are computed simply by adding the effective heights of successive columns. Similarly the load-vectors may be partitioned with the number of rows in a load block equal to the number of columns in the corresponding matrix block, as illustrated in figure 3.5.

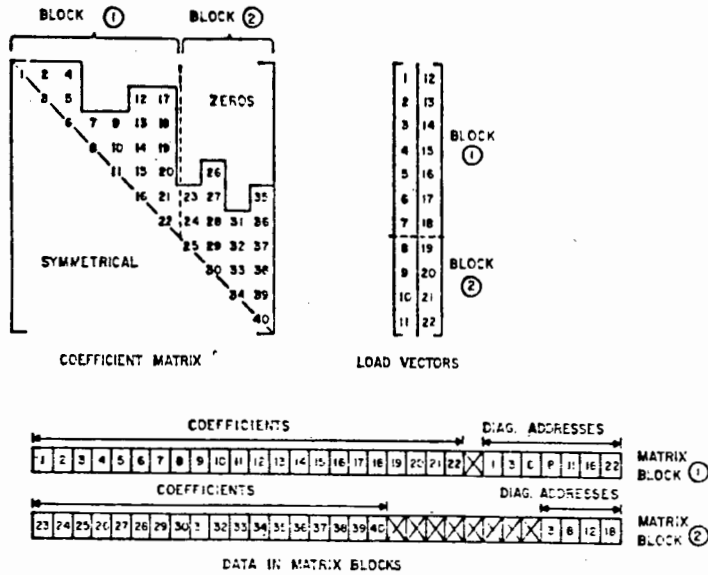


Figure 3.5: Partitioning with compacted column storage

3.3.4 Restrictions of Partitioning Methods

A problem size restriction is imposed by the row or column partitioning schemes. If each partition is considered to be a record on fixed disc then the maximum problem size will be restricted by the record size, since at the limit a single row or column will need to fit onto a single record. If record sizes are set to fit the problem then difficulties arise because the program requires an I/O buffer to read the record from disc and this increases the memory requirements of the program as these buffer sizes are problem dependent.

The method of storing a number of columns in a single partition which is to be transferred as a whole to and from secondary storage is quite effective and simple to program. However, in every computation step a considerable portion of data within the partition is superfluous because it is lacking a counterpart in the complementary partition [36], see figure 3.6. This does not affect CPU-time, but storage

allocation and data transfers increase unnecessarily. This deficiency can be easily avoided if the matrix is divided not only by columns, but also by rows. This leads to the hypermatrix partitioning scheme.

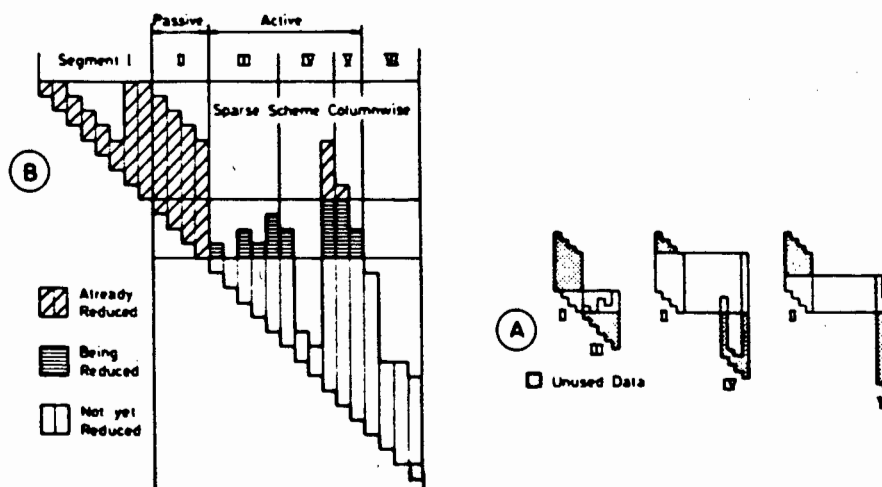


Figure 3.6: Compacted column storage for Cholesky decomposition

3.4 Hypermatrix Schemes

Every direct method for the solution of linear equations is required to work both by rows and by columns. This demand is met in the most natural way by partitioning the global stiffness matrix in a symmetric way both by rows and by columns into compatible submatrices. The method allows the handling of large matrices in a problem independent manner. The size of the submatrices does not depend on the problem size but can be chosen to be an optimum size which matches the hardware block or record size for I/O operations. The data structure developed is modular and the data management scheme is easily separated from the solution algorithm.

Each submatrix is treated as a record to be moved in or out of memory with a single I/O operation. A number of pointers are utilized to indicate the existence, or non-existence, of the submatrices and their disk addresses. These pointers may in themselves be regarded as a higher level matrix, figure 3.7.

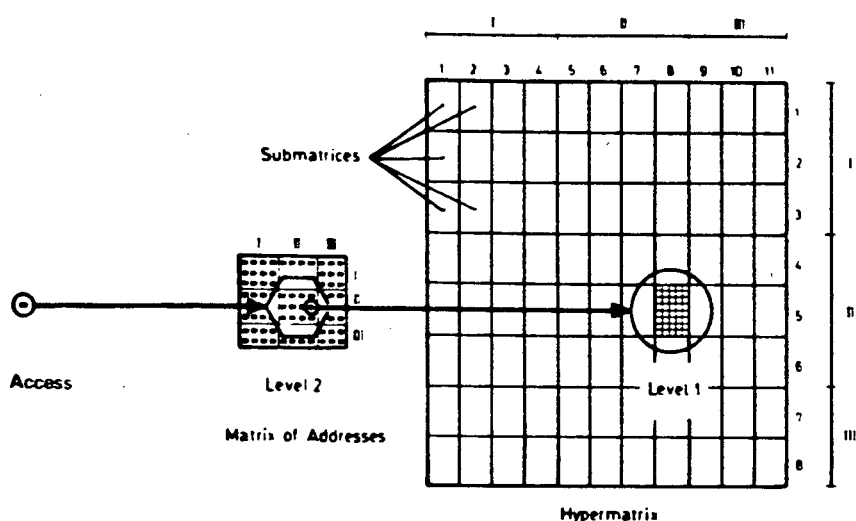


Figure 3.7: Hypermatrix Partitioning

3.4.1 Restrictions on problem size

When using the DAS-package, the dimensions of each data entity for which space is to be allocated must be such that it does not exceed the page size of virtual memory. The method of accommodating the global stiffness matrix is to divide it into a number of submatrices, where the dimensions of each of these submatrices are such that they can fit onto a single page. Figure 3.8 depicts the basic concept of dividing a large matrix, called a hypermatrix, both by rows and columns, into submatrices. These submatrices represent the second level of the

hierarchical data structure. The data structure on the first level is a two-dimensional array containing the addresses of the submatrices, or zeros for null submatrices.

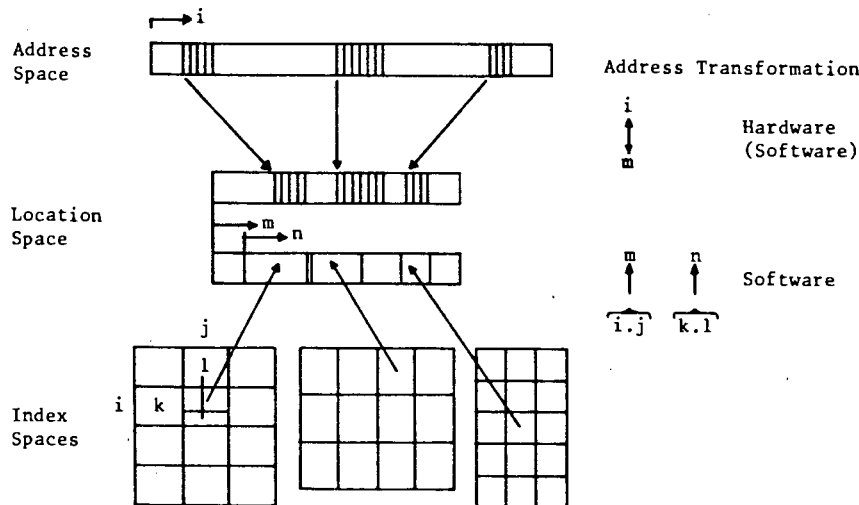


Figure 3.8: Two-level hierarchy of matrices

This scheme can be extended recursively to any number of levels beyond two. However, Schrem [12] states that the experiences with the ASKA system indicate that a two-level hierarchy of storage for hypermatrices is quite sufficient in almost all cases.

3.4.2 Hypermatrix concepts

Hypermatrices can be defined as matrices which are partitioned by rows and columns into submatrices, refer to figure 3.7. The size of the individual submatrices should be arranged so that one set of operand and result submatrices of numbers and corresponding submatrices of addresses fit simultaneously in primary memory. Further advantages result if the size of the submatrices of numbers coincides exactly with the size of the modular divisions (i.e. pages) of primary memory and

secondary storage. The size of a page may in turn coincide with hardware characteristics on secondary storage such as the size of a disk track.

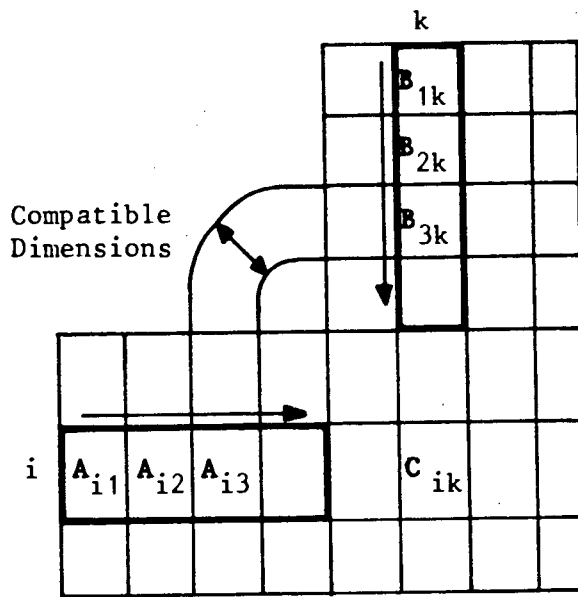
It should be emphasized that when using direct access storage, the submatrices of a hypermatrix can be accessed just as efficiently column-wise as row-wise. Thus in contrast to sparse schemes with partitioning exclusively by rows or by columns, a hypermatrix can be used equally effectively as either a pre-multiplier or a post-multiplier or as its own transpose. For example, hypermatrix back-substitution is just as efficient as either triangularization or forward-substitution.

Hypermatrix schemes can be most naturally applied to algorithms which are recursive. A recursive matrix operation can be defined as one which permits analogous relations between the submatrices to those which exist at the numerical data level. In other words, one tries to develop algorithms at the hypermatrix level which reduce to known algorithms in the case of 1 by 1 submatrices, i.e. numbers. For example, the matrix multiplication operation $C = AB$ defined on the submatrix level (figure 3.9.) as

$$C_{ik} = \sum_m A_{im} B_{mk} \quad (3.1)$$

reduces at the number level simply to the recognizable expression

$$C_{ik} = \sum_m a_{im} b_{mk} \quad (3.2)$$



$$C = AB$$

$$\text{Level 2: } C_{ik} = \sum_l A_{il} B_{lk} V_{i,k}$$

$$\text{Level 1: } c_{ik} = \sum_l a_{il} b_{lk} V_{i,k}$$

Figure 3.9: Hypermatrix Multiplication

Many other operations including addition, subtraction, triangularization of a positive-definite matrix (using Gauss, or Cholesky methods), forward-substitution, back-substitution etc. are recursive and can be readily programmed in hypermatrix form.

Appropriately applied recursive procedures normally involve precisely the same number of arithmetical operations as the equivalent standard procedures at the number level [12]. This fact has not usually been exploited in previously presented partitioning methods, most of which require more arithmetic than needed by the number algorithms, particularly if the partitions are large.

3.4.3 Organization of hypermatrix calculations

The operation $C = AB$, shown in figure 3.9, can conveniently illustrate the calculation steps and data transfers in a typical hypermatrix algorithm. It can be observed that the result submatrix C_{ik} remains in primary memory during the complete series of multiplications with the i^{th} row and k^{th} column of submatrices in hypermatrices A and B respectively, after which submatrix C_{ik} can be output once and for all in final form. After the typical intermediate step at row or column m of accumulating the product $A_{im} B_{mk}$ into C_{ik} , submatrices $A_{i,m+1}$ and $B_{m+1,k}$ can immediately be input to over-write A_{im} and B_{mk} thus allowing a systematic progression through the algorithm and simple book-keeping. In particular during the triangularization of the structural stiffness matrix, the majority of the operand submatrices will be densely populated. Thus $O(n^3)$ multiply/addition operations will be performed after the input of only two n by n submatrices. This fact leads to quite desirable ratios of data transfer to calculation times, particularly if the submatrices are fairly large, for example 20 to 50. Also the operations with the submatrices of numbers are carried out with simple matrix multiplication subroutines for which the inner loops can be readily written in assembler language, thus speeding up the calculations considerably for some types of computer.

If matrices A and B were partitioned exclusively by rows (or exclusively by columns), as recommended in many sparse schemes, the above operation $C = AB$ would require excessive data transfer time. If A were partitioned row-wise and B column-wise, this difficulty would be removed. However, if A had to be used as a post-multiplier or B as a pre-multiplier or either as their transpose in later calculation steps,

either an extra copy of one operand matrix would have to be formed on backing-store, or one of the operand matrices and/or the result matrix would have to be progressively unpacked into expanded form in executable memory. Due to the symmetric partitioning by both rows and columns, this problem does not exist in hypermatrix programs using direct access storage.

3.4.4 Advantages of hypermatrix schemes for sparse matrices

The address matrix of a hypermatrix represents a macro image of the sparsity distribution within the matrix as a whole. Zero submatrices are neither generated nor stored, and are merely identified by a zero address [12] in the appropriate location at the address level. In this way (i) most of the large contiguous zero areas within a given hypermatrix need not be stored and (ii) the rapid recognition of zero addresses automatically prevents the superfluous arithmetic on submatrices containing only zeroes.

One of the main difficulties in practical application of optimal ordering algorithms is that in the very large problems where such a procedure should be most beneficial, the cost of the re-ordering process may exceed the resulting savings, especially when ordering is performed on single rows and columns. However, re-ordering algorithms [12] working on the address matrix of the structural stiffness matrix may produce significant savings. This address matrix, which provides a macro representation of the sparsity, is rarely larger than 200 by 200, even for large problems. The cost of the re-ordering process itself takes only 1% to 2% of the triangularization time [12].

It might be concluded that a finer partitioning, by enabling more thorough exclusion of zero areas, would increase the efficiency of a hypermatrix scheme on sparse matrices. However, operations on very finely partitioned matrices incur overhead costs due to increased book-keeping. Fortunately, for certain processes such as triangularization of the structural stiffness matrix, the recognition of zeroes within certain sub-matrices (even during the outer loop of the most frequent matrix routines) enables skipping the majority of calculations on zeroes within these sub-matrices. For this reason the tendency is to use bigger sub-matrices of numbers for large problems (e.g. 40 by 40 up to 75 by 75) [12]. When using larger pages, the number of data transfers is naturally reduced. However, as the use of extremely large pages usually not only increases the number of zeroes which must be stored (especially for irregular sparsity patterns), but also leads to higher unit data transfer costs in a multi-programming installation due to the corresponding increase in required primary memory storage, the page size should not increase indefinitely. One can arrive by experience at a reasonably optimum page size for a given computer.

3.4.5 Hypermatrix and Hypertable Management Routines

This section describes the hypermatrix and hypertable routines, developed by Vos [17], which are used for the data management scheme in the Cholesky Hypermatrix Equation Solution Scheme as described in Chapter Five. The hypermatrix routines are used for the storage of the structural stiffness matrices, the load matrices and the resultant displacement matrices. The hypertable routines are used for the mapping of free, prescribed and suppressed degrees of freedom into their respective hypermatrices (see section on assembly of global stiffness matrix).

The module of FORTRAN routines was developed to be able to create and access two-level hypermatrices. The routines in the general hypermatrix module call the routines in the DAS-package in turn to allocate space in virtual memory, as well as to store information therein and to retrieve it again. This module can be considered as an additional layer of software between the DAS-package and the equation solution routines, (figure 3.10).

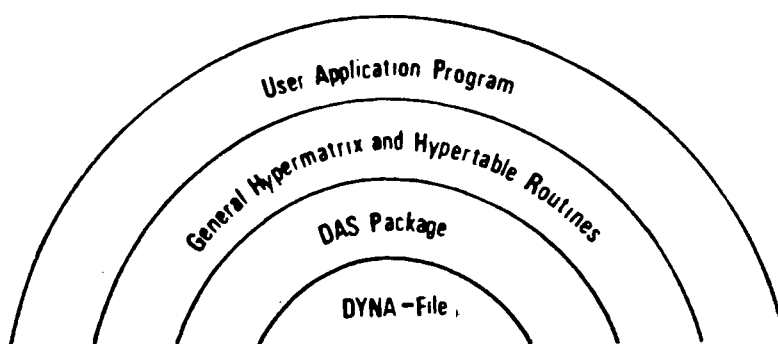


Figure 3.10: Layers of Software

The name and purpose of each individual routine is similar for both modules, therefore only a description of the hypermatrix module is given:

- HMGINI - Initializes the internal variables of the module
- HMGDMP - Dumps the internal variables of the module
- HMGCRT - Creates a new hypermatrix
- HMGOPN - Opens an existing hypermatrix
- HMGCLS - Removes the control information about a hypermatrix
from the table
- HMGPUT - Stores a submatrix of values in a hypermatrix
- HMGGET - Retrieves a submatrix of values from the hypermatrix

- HMGSIZE - Calculates the dimensions of a submatrix and the number of words occupied by the submatrix
- HMGINF - Retrieves control information about a hypermatrix
- HMGNDX - Calculates the index of a submatrix with control information area
- HMGEXT - Creates a submatrix for a hypermatrix and inserts the pointer to it in the control information area
- HMGDEL - Deletes the submatrices and the control information area of a hypermatrix
- HMGESV - Stores a single value in a hypermatrix
- HMGESG - Retrieves the value of an element in the hypermatrix
- HMGLOC - Determines in which submatrix partition a hypermatrix element is in and what the offset from the first one is.

A full description of each of these routines is given in appendix E. A full description of hypertable routines is presented in [17].

CHAPTER FOUR

A PARTITIONED FRONTAL SOLUTION SCHEME

In this chapter the investigation of the frontal solution technique and in particular a scheme for partitioning the frontal storage is described. Problems encountered and experience gained during this development are outlined.

4.1 The Frontal Solution Scheme

The frontal equation solution technique was originated by Irons [2] and has earned the reputation of being easy and inexpensive to use. It can be considered as a technique for interweaving the assembly and elimination of equations, based on the fact that an equation can be eliminated if all the coefficients of the equation are assembled. Once all the equations have been eliminated the unknown displacements may be solved by means of a backsubstitution process. It is designed to minimise primary memory requirements, the number of arithmetic operations and the use of secondary storage. Its efficiency and advantages over other algorithms, such as the banded solution schemes, are derived from its method of implicit assembly of the global stiffness matrix and the resulting exploitation of the sparsity within the matrix bandwidth.

4.1.1 General Description of the Frontal Solution Scheme

The main objective of the frontal solution scheme is to assemble the equations and eliminate the variables at the same time. As soon as the

coefficients of an equation are completely assembled from the contributions of all the relevant elements, the corresponding variable can be eliminated. Therefore the complete structural stiffness matrix is never formed as such, since after elimination the reduced equation is immediately transferred to secondary storage.

The primary memory contains, at any given instant, the upper triangular part of a square matrix containing the equations which have been formed up to that particular time. These equations, their corresponding nodes and degrees of freedom are termed the front. The number of unknowns in the front is the frontwidth, and this length generally changes continually during the assembly/reduction process. The maximum size of the problem which can be solved is governed by the maximum frontwidth. The equations, nodes and degrees of freedom belonging to the front are termed active; those which are yet to be considered are inactive and those which have passed through the front and have been eliminated are said to be deactivated or decomposed, see figure 4.1.

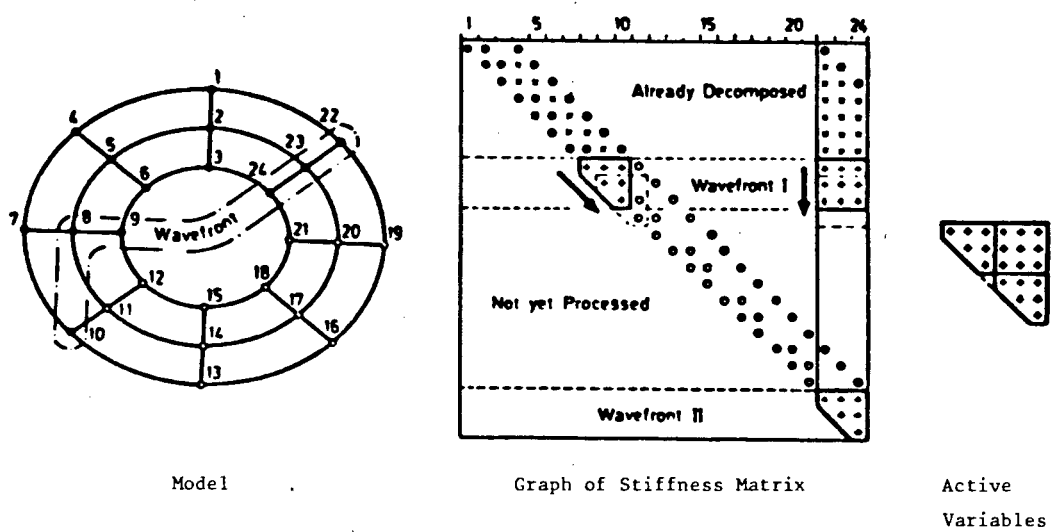


Figure 4.1: Example of frontal solution method

During the assembly/elimination process the elements are considered each in turn according to a prescribed order. Whenever a new element is called in, its stiffness coefficients are read from secondary storage and summed either into existing equations, if the nodes are already active, or into new equations which have to be included in the front if the nodes are being activated for the first time. If some nodes are appearing for the last time, the corresponding equations can be eliminated and written away to secondary storage and are thus deactivated. In so doing they free space in the front which can be employed during assembly of the next element.

4.1.2 The Frontal Solution Algorithm

For a well documented explanation of the frontal solution scheme, refer to 'Finite Element Programming' by Hinton and Owen [2].

The basic steps of the frontal solution scheme are, figure 4.3:-

1. The solution process begins by initializing several 'housekeeping' parameters.
2. Begin the element by element assembly and reduction process.
3. Each element variable is assigned a location in the front, the frontwidth being adjusted if necessary.
4. The element stiffness and load terms are assembled into the global equations (front) as governed by the position of the variable in the front.
5. If a node is appearing for the last time the corresponding degrees of freedom are eliminated by equation reduction. The elimination of a row s leads to a modification of the elements in the remaining rows according to

$$k_{ij}^* = k_{ij} - \frac{k_{si}^* k_{sj}^*}{k_{ss}^*} \tag{4.1}$$

where stored coefficients are fully summed, figure 4.2.

- 6. The stiffness and load terms corresponding to the eliminated variables are written to secondary storage.
- 7. The displacements are obtained by backsubstitution in reverse order.

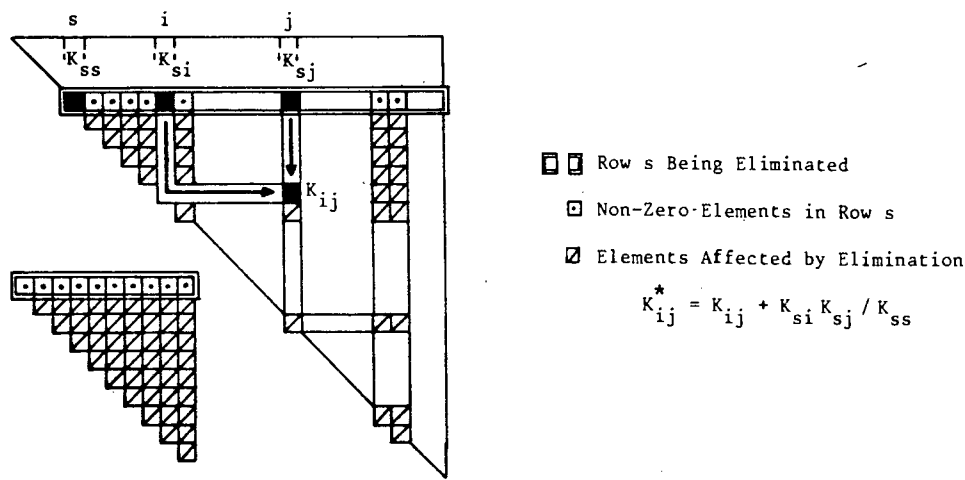


Figure 4.2: Matrix elements directly affected by elimination of row s .

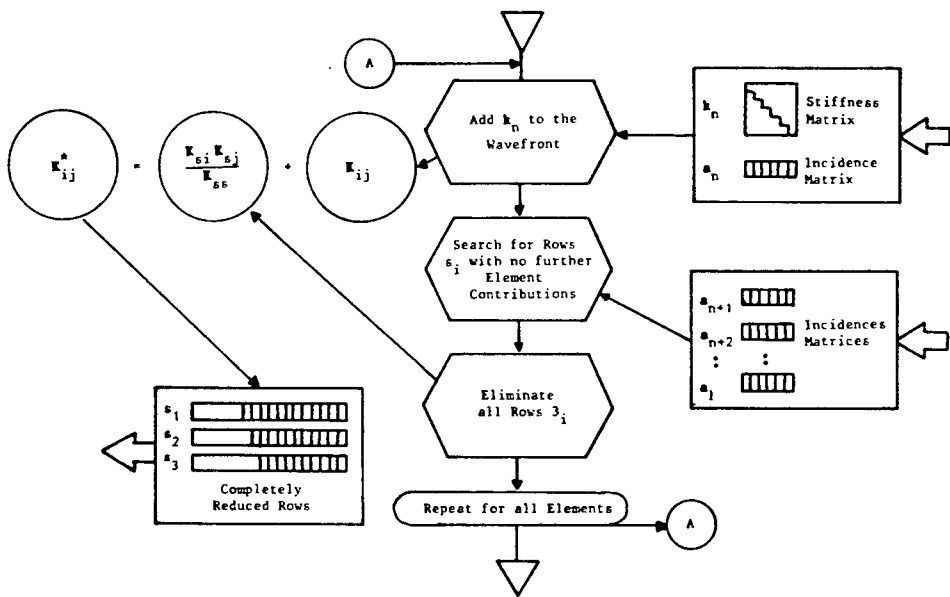


Figure 4.3: Frontal Solution Method, variant of IRONS

4.1.3 Discussion of the frontal solution technique

The front solution is a very efficient direct solution process. Its main attraction is that variables are introduced at a later stage and eliminated earlier than in most other methods. The active life of a node lasts from the time in which it first appears in an element to the time in which it last appears in an element. This has the following consequences.

The ordering of elements is crucial and the ordering of nodal numbering is irrelevant. This is a different requirement to that for the banded solution schemes and is an easier one to satisfy since, invariably, there are fewer elements than nodes, especially in three-dimensional problems. Furthermore if a mesh is found to be too coarse in some region, its modification does not require extensive nodal point renumbering.

The primary memory requirements are at most the same as those of a banded Gaussian solution. The primary memory required is less for structures analysed by means of elements with midside nodes [4,11]. An example of the efficiency of the solution scheme is the closed ring illustrated in figure 4.4.

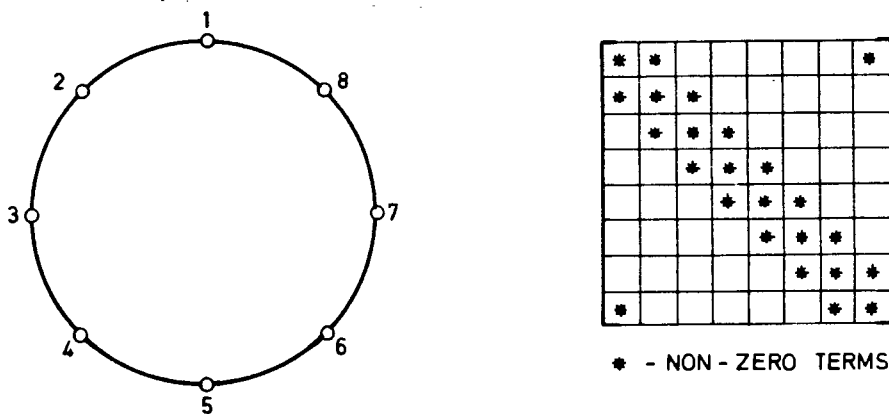


Figure 4.4: Example of a non-banded equation system

With the nodal points numbered as shown an ordinary banded solver leads to the half band width being equal to the total number of equations. This is effectively reduced to 3 by the frontal solution. (Of course, the bandwidth for a banded solution can also be reduced by judicious nodal numbering).

Because of the compact nature of the front and because variables are eliminated as soon as conceivably possible, the operations on zero coefficients are minimised and total arithmetic operations are fewer than with other methods [2]. On the other hand, an elaborate bookkeeping system is required for frontal solution. However, since the bookkeeping is entirely performed using integer variables, little storage and computer time is used for manipulating these variables.

Because any new equation occupies the first available space in the front, there is no need for a bodily shift of the in-core equations as in many other large capacity equation solvers.

Primary memory requirements may also be reduced if the maximum frontwidth is kept to a minimum. The frontwidth size may be reduced by using Rashid iteration which converges with conjugate Newton [4].

4.2 Solution with a Partitioned Frontal Solver

This section describes the investigation made of a partitioned frontal solver presented by Beer and Haas [6] which was based on the original work of Irons [4]. The advantages of the frontal solution scheme over other solution schemes have already been pointed out; however, one of

the disadvantages of published frontal solvers is that the matrix of 'active' coefficients has to fit into central memory. To be able to solve large finite element systems on small computers the front matrix has to be partitioned. This means that only a part of the front matrix resides in primary memory at any one time. While the idea of partitioning is not new [28], the efficient implementation is complex. In a partitioned solver (and this applies to other out-of-core methods), the I/O times become significant and so the main emphasis should be to minimize and improve disk I/O procedures. The purpose of this work was to investigate the data structure requirements for the frontal solution and replace the existing partitioning scheme in [6] with a more structured paging scheme using the DAS package.

The maximum storage requirement for the front is determined by the maximum front size, ω . Only one-half of the symmetric front matrix including the diagonal terms has to be stored, the storage requirement is

$$\text{LIMIT} = \omega(\omega+1)/2 \quad (4.2)$$

The paper published by Beer and Haas [6] presented a program in which this restriction had been overcome by partitioning the front matrix and which enabled the solution of a large 3-D stress analysis problem with 10000 degrees-of-freedom on a mini-computer with only 64 kbytes of memory. The program is divided into three sections:-

1. PREFRONT routine
2. FRONTAL SOLUTION routine
3. RESOLUTION routine

4.2.1 The PREFRONT Routine

All the information necessary during the elimination process, i.e. maximum front width, the order of elimination of equations and the destination addresses for the coefficient in the front, are computed in the so-called PREFRONT routine.

This separation of planning and execution allows the implementation of new facilities by modifications to the PREFRONT routine. Among them are longevity considerations and other techniques which make the elimination process faster by reducing the number of zero coefficients in the front matrix.

A measure of flexibility which is introduced is the consideration of longevity, i.e. the span of elements for which a variable remains active. Variables which have a short "life-span" are the last variables in an element to be introduced into the front. This is particularly useful when a partitioning scheme for the front storage is used as the number of partitions required in the front is kept to a minimum during the solution.

4.2.2 The FRONTAL Routine

The frontal routines initialize the partition handling algorithm and perform the assembly and elimination procedure. Similar to the original code of Irons, a working vector ELPA was used. ELPA is divided into three main areas, see figure 4.5:-

1. the element stiffness matrix of the element to be assembled
2. the front matrix or a partition of the front matrix
3. a buffer for the equations which have ceased to be active

The space for these areas are dynamically allocated at run time. The first area is set to the size of the element stiffness matrix and the space allocation for areas 2 and 3 must be adjusted to give optimal solution times within primary memory limitations of the computer used.

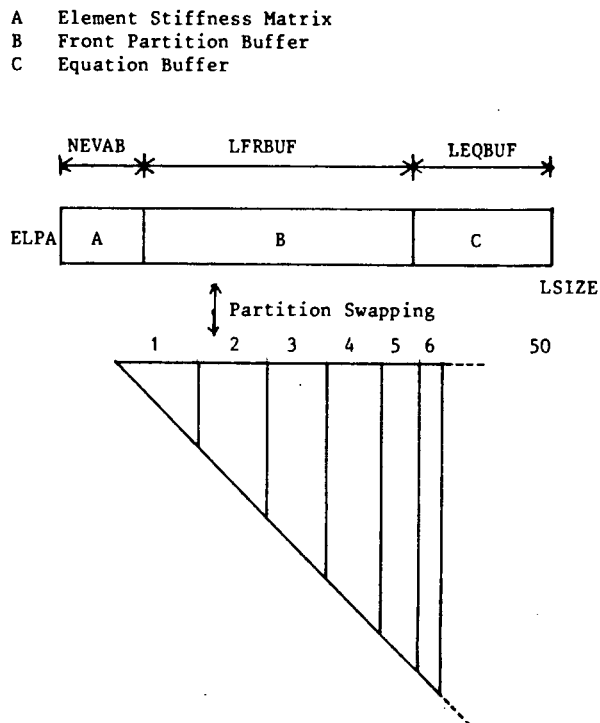


Figure 4.5: Frontal solution partitioning scheme

As long as the front matrix is small and fits into primary memory, the program strategy is simple and follows three basic steps for each element:

1. Read element stiffness matrix and assembly information (destination vector).
2. Assemble its coefficients into the front matrix.
3. Eliminate variables which are ready by adjusting the other 'active' equation coefficients using eq 4.1.

The elimination essentially consists of two steps. First, the equation coefficients of the variable which is to be eliminated are moved to the equation buffer (when the buffer is full, its contents are written on disk for re-solution). Then all the equations in the front matrix are modified by eq (4.1).

However, when the current front matrix becomes large and no longer fits in the allocated space, partitioning is necessary. Basically, two strategies are possible.

- A. Assembly and elimination sequence of the variables of each finite element is governed by the sequence in which partitions are transferred into core.
- B. The sequence in which partitions are transferred into core is governed by the sequence in which variables are assembled or eliminated.

Strategy A is the more efficient strategy [6] since the number of swapping operations of the partitions is kept to a minimum and this was examined in more detail.

The object in the following is to reduce the I/O operations to a minimum even if this means a slight increase of in-core operations. So for the assembly and elimination a strategy is applied which is basically different from that presented by Irons [4] or other partitioned frontal solvers [28]. For each finite element, the following basic steps are carried out.

- 1. Read element stiffness matrix and assembly information.
- 2. If not already in core, swap first partition into core and assemble all coefficients which are resident in this partition.

3. Eliminate variables which are ready in this partition and modify all coefficients resident there.
4. Swap subsequent partition into core and assemble further coefficients.
5. Modify all coefficients in this partition for variables eliminated in the previous partition(s) using the coefficients k_{nl} in the equation buffer.
6. Eliminate all variables which have become ready in the current partition.

Steps 4, 5 and 6 are repeated for all further partitions which are currently needed.

If variables have been eliminated from other than the first partition, the coefficients in some partitions will not have been modified completely in the first loop and a further pass is required to modify them.

For strategy A, if NPA is the number of partitions currently used, in the worst case, $2NPA-1$ passes are required through the partitions; this is independent of the number of variables eliminated in the element loop.

For strategy B, one pass through all the partitions is required for each variable which can be eliminated in each finite element. This could mean up to $60*NPA$ passes if, for example, 20-node solid elements are used.

In many cases, the number of swapping operations per finite element for strategy A will be less than 2NPA-1, especially when a large number of partitions is involved. For example, swapping of partitions is not required when there are no coefficients to be assembled into a partition and no variables have been eliminated yet in the current element loop. If all the coefficients resident in a partition are zero, this partition is declared inactive and completely ignored in the elimination process until filled with non-zero coefficients. This will effectively prevent operations on zero rows when partition limits are determined dynamically and is an economic alternative to the moving of coefficients.

The partitions are processed from the last one backwards and this allows the implementation of the 'elimination from the end of the front' as suggested by Abbas[30]. This, together with a pre-frontal routine based on longevity, should ensure that most variables are eliminated from the back of the last partition, resulting in an immediate reduction of the size of the front matrix and a minimum number of swapping operations.

4.2.3 The RESOLUTION Routine

The resolution routine performs the forward and backsubstitution in the solution process thus separating it completely from the reduction of the left-hand side. This enables maximum flexibility to compute load cases specified after the elimination process, and leaves as much program space as possible for the simultaneous computation of many right-hand sides. Because the size of the vector needed for each load case is equal to the maximum front width, no partitioning is required even for large problems.

The resolution consists of a forward and back substitution part. The basic steps for each element in the forward substitution are:

1. If loads are applied to the element read the load vector.
Otherwise go to step 3.
2. Assemble loads into front-RHS.
3. Using the coefficients in the equation file modify the front coefficients by:

$$F_i' = F_i - \frac{K_{in}^*}{K_{nn}^*} F_n^* \quad (4.3)$$

In the back substitution the results are obtained for the last element first by

$$x_n = \frac{1}{K_{nn}} \left[\sum_{i=1}^{n-1} K_{ni} x_i - F_n \right] \quad (4.4)$$

4.2.4 Selection of Front Partition and Equation Buffer Length

When selecting the size of the buffer for the front matrix, the objective is to avoid partitioning since it is expensive. If no partitioning is involved, the equation buffer may be made small to accommodate a big front matrix, the only restriction being that the buffer has to be at least two blocks long and accommodate the largest equation. On the other hand, when partitioning of the front is unavoidable because of primary memory restrictions or the size of the problem, there is a case for decreasing the partition size in favour of a large equation buffer.

4.2.5 Treatment of Suppressed and Prescribed Displacements

Distinction can be made between restraints and prescribed displacements. Suppressed degrees of freedom are simply deleted from the list of variables (i.e. given the destination zero). For non-zero prescribed displacements, a technique attributed to King [7] is used. The implementation is as follows:

1. In the reduction of the LHS the equation corresponding to the prescribed variable is moved to the equation buffer but no 'active' equations are modified.
2. In the forward substitution, the prescribed value multiplied by its equation coefficient is subtracted from the RHS value.
3. In back substitution, the prescribed values are entered in the list of results.

The procedure described above is most efficient in storage requirement and computation time but does not give reactions. However, boundary conditions need to be checked during the solution process. Therefore there is no clean separation between the free, prescribed and suppressed degrees of freedom in the solution and they each need to be dealt with in turn as special cases.

4.2.6 Frontal Partitioning Data Structure

A partitioning by columns data handling scheme was implemented, as shown in figure 4.5, to store the front matrix. A partition control array, PALI, was used to determine which partition needed to be swapped

in or out of core. A maximum of fifty partitions is allowed, with, each partition being stored as a single record in a direct access file on secondary storage.

The theoretical limits of the program were as follows:-

1. Maximum number of elements 512 (NELSU)
2. Maximum number of degrees of freedom per node 6 (IFG)
3. Maximum number of nodes 4000 (LNIC,LRCO)
4. Maximum number of partitions 50 (LPAL)
5. Maximum partition size was also limited by the size of a single record on the direct access file.
6. The working buffer (KLPA) was limited to 5000 and is required to store the upper triangular stiffness matrix, a front partition and the equation buffer.
7. The maximum front size is determined by the front partition size with 50 partitions available.

4.3 Difficulties Encountered with the Implementation

The FORTRAN code presented by Beer and Haas [6] was implemented and tested using an example which had been verified by the frontal solution scheme of Hinton and Owen [2]. It was found that the code would not run to completion and that several of the intermediate results were not correct. Several programming errors were discovered in the code. For example subroutine DCODE which should have decoded an integer number with the restraint code of a node was incorrect and did not produce the result described in the documentation, as illustrated in figure 4.6.

```

C      NOTE: EACH NODE IS GIVEN A RESTRAINT CODE WHICH INDICATES
C      HOW MANY DEGREES OF FREEDOM IT HAS AND WHICH ARE
C      RESTRAINED.
C
C      AN UNRESTRAINED NODE HAS THE CODE '1' FOR EACH D.O.F.
C
C      FOR EXAMPLE AN UNRESTRAINED NODE WITH THREE D.O.F. HAS
C      A CODE '111'.
C
C      FOR EACH DEGREE OF FREEDOM WHICH IS RESTRAINED
C      A '0' IS ENTERED.
C
C      FOR EXAMPLE A THREE D.O.F. NODE WHICH HAS ITS FIRST D.O.F.
C      RESTRAINED HAS A CODE OF '011'.
C
C      FOR EACH DEGREE OF FREEDOM WHICH WHICH HAS A PRECIBED VALUE
C      A '2' IS ENTERED.
C
C      FOR EXAMPLE A THREE D.O.F. NODE WHICH HAS A PRESCRIBED
C      DISPLACEMENT ON THE FIRST DEGREE OF FREEDOM HAS A
C      CODE OF '211'.
C      :
C      :
C      SUBROUTINE DCODE(IFG,ICODE)
C-----
C SUBROUTINE TO DECODE RESTRAINT CODE FOR PREFRONT
C-----
C      COMMON IREF(256)
C      DIMENSION IFG(6)
C      EQUIVALENCE (NDFMAX,IREF(5))
C      EQUIVALENCE (NULLS,IREF(160))
C      I= NULLS
C      DO 1 N=1,NDFMAX
C      IFG(N)= 1
C      IF(ICODE-I .LT. 0) GOTO 2
C      IFG(N)= 0
C      ICODE= ICODE-I
C      IF(ICODE-I .LT. 0) GOTO 2
C      IFG(N)= 2
C      ICODE= ICODE-I
C 2 CONTINUE
C      I= I/10
C 1 CONTINUE
C      RETURN
C      END
C

```

Figure 4.6: Subroutine DCODE from Beer and Haas [6]

Several attempts were made to debug the code, but without success due to its complexity and a lack of complete comprehension of the implemented algorithm. This can be attributed to several factors:

1. There were no test examples or diagrams published.
2. The documentation was poor and did not clearly explain the sections of the code.
3. The program was monolithic and lacked modularity as several routines had multiple purposes, eg. subroutine PALI is used to set up the partition limits and also zeros array INDABL.
4. Several variable names were also used for the same variable eg. NTYPE and NTYP, they were also generally short and non-descriptive.
5. The code also contained unnecessary complications as illustrated in figure 4.7.

```

C  FIND ADDRESSES FOR ALL DEGREES OF FREEDOM
C  WHICH ARE NOT COMPLETELY RESTRAINED
      LDES= 0
      DO 60 LOC=1,NTYP
      IF(LOC .LE. LOCH) LDES= 0
      N= ISORT(LOC)
      NIC= NICN(N)
      IRDY= 0
C  CHECK IF THIS IS LAST APPEARANCE OF NIC
      IF(NIC .GT. 0) GOTO 65
C  OR IF NODE IS FULLY RESTRAINED
      IF(NIC .EQ. 0) GOTO 70
      NIC= -NIC
      IRDY= 1
      65 CONTINUE
      NICI= 0
C  IF THIS IS FIRST APPEARANCE OF THE NODE ("+"VE ICODE)
C  WE HAVE TO LOOK FOR EMPTY SPACES IN INDABL
      ICODE= NRCC(N)
      IF(ICODE .GT. 0) GOTO 75
C  OTHERWISE WE HAVE TO LOOK FOR EXISTING ADDRESS
      ICODE= -ICODE
      NICI= NIC
      75 CONTINUE

```

```

      CALL DCODE(IFG,ICODE)
C FIND NDOF DESTINATIONS
      DO 80 ND=1,NDFMAX
        IF(IFG(ND) .EQ. 1) GOTO 95
90    CONTINUE
        LDES=LDES+1
        IF(LDES .GT. KURPA) KURPA= LDES
        IF(KURPA .LE. INDEND) GOTO 4
        WRITE(NT0,3000)
        WRITE(NT0,3003) KURPA,INDEND
3003  FORMAT(' FRONT WIDTH',I5,' >',I5)
        STOP
        4 CONTINUE
        IF(INDABL(LDES) .EQ. NICI) GOTO 85
        GOTO 90
        85 CONTINUE
C FOUND DESTINATION FOR ND-TH D.O.F. , N-TH NODE
        INDABL(LDES)= NIC
        ISYM(LDES)= 1H+
C CODE LDEST FOR PRESCRIBED DISPLACEMENT (+1000)
        LDEC= LDES
        IF(IRDY .EQ. 0) GOTO 86
        LDE= LDES
        IF(IFG(ND) .EQ. 2) LDE= LDES + 1000
        LDEC= -LDE
86    CONTINUE
        L= N+NTYP*(ND-1)
        LDEST(L)= LDEC
        GOTO 80
95    CONTINUE
        L= N + NTYP*(ND-1)
        LDEST(L)= 0
80    CONTINUE
        GOTO 60
70    CONTINUE
C A FULLY RESTRAINED NODE
      DO 110 ND=1,NDFMAX
        L= N + NTYP*(ND-1)
        LDEST(L)= 0
110   CONTINUE
60    CONTINUE

```

Figure 4.7: Calculation of addresses in front, Beer & Haas [6]

Although several letters were sent to Beer and Haas in an attempt to obtain a copy of the source code on tape, some test examples and any further documentation, no reply was received.

4.4 Implementation of a frontal solver with a hypermatrix scheme

Due to the problems encountered with the implementation of the code of Beer and Haas [6], it was decided to incorporate a hypermatrix scheme into the frontal solution scheme presented by Hinton and Owen [2], to gain some experience with the implementation of the hypermatrix routines. This hypermatrix scheme was implemented and tested successfully but as expected it was found to be inefficient due to the random accessing of coefficients in the front matrix and the high number of resulting I/O operations. It also meant integrating two types of data structures, which decreased the modularity of the program and greatly increased the complexity of the code. Therefore this solution scheme was not investigated any further.

4.5 Discussion and Conclusions

An equation solver should not only be looked at separately but in the overall context of the finite element program system in which it is used. Many of its advantages or disadvantages depend on the data handling and overall organisation of the program structure. The conclusion was reached that the band, skyline and frontal solution schemes are too closely integrated with the data structures used by the algorithm and a more modular and easily adaptable solution scheme was required. Therefore, it was decided to investigate the implementation of a hypermatrix solution scheme, which has a modular data structure and lends itself to a modular implementation of a suitable mathematical algorithm such as Cholesky decomposition.

As a complete package the solution scheme required a module of interface routines which would isolate the user from the actual solution method. These interface routines would provide a means of describing the boundary conditions, assembling the equations, solving the system of equations and retrieving the results with a minimum core requirement and a modular (layered) implementation.

CHAPTER FIVE

THE CHOLSKY HYPERMATRIX EQUATION SOLUTION SCHEME

This chapter describes the Cholesky solution procedure for symmetric positive-definite matrices and its implementation with a Hypermatrix Data Management Scheme for the solution of systems of linear equations in the finite element method. The program suite described, CHESS - Cholesky Hypermatrix Equation Solution Scheme, has been implemented with the NOSTRUM development at the University of Cape Town.

The method of Data handling is provided by the DAS-package [17] which uses a virtual memory management and paging scheme. The paging scheme is used to swap pages between the data pool in primary memory and a DYNA-file on secondary storage. The DYNA-file is a direct access file with fixed record sizes which corresponds to the page size in the data pool.

A 2-level hypermatrix and hypertable data handling scheme, which accesses the DAS-package, was implemented. The module of hypermatrix routines is used for the storage of the global stiffness matrix, the global load vector and the unknown displacements. The module of hypertable routines is used for the storage of the global degree of freedom look-up table.

The global degree of freedom look-up table is used to assemble the element stiffness matrices into the various global stiffness hypermatrices. These various hypermatrices together represent the

total global stiffness matrix. Element stiffness coefficients are placed in particular hypermatrices depending on their attributed global degrees of freedom.

The Cholesky procedure for hypermatrices is implemented using a row-wise evaluation scheme as described by Schrem [12] for the ASKA program system. The Cholesky procedure is used for the inverse multiplication of two hypermatrices while a general hypermatrix routine is used for the standard multiplication of two hypermatrices.

Finally, a module of interface routines is provided for the application program to define the system of equations, to assemble the global stiffness matrix element by element, to initiate the solution procedure and to retrieve the calculated displacements and reactions.

5.1 The Cholesky Solution Procedure for Symmetric Matrices

Given the system of equations

$$Kx = F \quad (5.1)$$

where K is the symmetric positive-definite structural stiffness matrix, x the unknown displacements, and F is the loading matrix. It is assumed that the suppressed and non-zero prescribed displacements have already been removed from eq. (5.1); in addition, the loading matrix F is presumed to be correspondingly corrected for the effect of any non-zero prescribed displacements (see later the method of assembly for global stiffness matrix).

K can be triangularised by the Cholesky procedure to an upper triangular matrix U, as follows

$$U^T U = K \quad (5.2)$$

By equating corresponding terms on both sides of eq. (5.2), recursive relations for evaluating the terms of U are readily derived.

$i = 1$ to n

$$U_{ii} = \sqrt{(K_{ii} - \sum_{m=1}^{i-1} U_{mi}^2)}$$

$$U_{ij} = (K_{ij} - \sum_{m=1}^{i-1} U_{mi} U_{mj}) / U_{ii} \quad j = i+1 \text{ to } n \quad (5.3)$$

where K can be progressively over-written by U.

The next step of the Cholesky procedure is the "forward-substitution" process on the loading matrix F, thus producing the intermediate result Y.

$$U^T Y = F \quad (5.4)$$

Eq. (5.4) represents the solution of a lower triangular set of equations, and by equating corresponding terms on both sides of eq. (5.4), the following recursive relations for the terms of Y are readily obtained:

$i = 1$ to n

$$Y_{ij} = (r_{ij} - \sum_{k=1}^{i-1} U_{ki} Y_{kj}) / U_{ii} \quad j = 1 \text{ to } m \quad (5.5)$$

where m is the number of loading conditions and Y can overwrite F .

The final displacement matrix x is obtained from the back-substitution step

$$Ux = Y \quad (5.6)$$

which is simply derived using eqs (5.1), (5.2) and (5.4). In this case eq. (5.6) clearly represents the solution of an upper triangular set of equations, which is performed using the following recurrence relations:

$$i = n \text{ to } 1$$

$$x_{ij} = (Y_{ij} - \sum_{k=i+1}^n U_{ik}x_{kj})/U_{ii} \quad j = 1 \text{ to } m \quad (5.7)$$

where x can over-write Y .

5.2 Cholesky algorithm for Hypermatrices

In this section it is assumed that the symmetric positive-definite structural stiffness matrix K has already been formed as a hypermatrix using the form of static condensation described in the next section. Some methods such as [4,2] can avoid the physical formation of the matrix K . However, Fuchs, et al [12] indicate that for large problems using the more sophisticated elements, the assembly of K takes less than one quarter of the time used in the Cholesky triangularization. Methods using simultaneous assembly and triangularization may sometimes be more efficient (i.e. requiring less total data transfers); however, the resulting combined programs become very complicated, and the

advantages of modularity are lost. The pre-assembly of K permits a smooth flow through the triangularization process, itself easily the most expensive calculation step, for large problems.

Either the Gauss, Crout or Cholesky procedures could have been used for the hypermatrix triangularization. The Crout method replaced the Gauss method upon the introduction of desk calculating machines, because it required less saving of intermediate information, i.e. the results of a series of multiplications could be successively accumulated into the one result location. By the same reasoning the hypermatrix Crout algorithm requires considerably less data transfers than the corresponding Gauss algorithm [12]. In fact, it can be simply shown that on average, the hypermatrix Crout method uses only about 50% of the data transfers as needed for the hypermatrix Gauss approach. The Cholesky procedure, which is really a symmetric variant of Crout, is somewhat more efficient for positive-definite symmetric hypermatrices than Crout's method [12]. In addition, only the upper triangle of matrix K must be assembled when using the Cholesky approach, thus achieving major savings in storage. The triangularized result matrix U , eq. (5.2), can be separately formed as an upper triangular matrix or can directly overwrite matrix K , leading to further storage savings.

The recursive relations for the Cholesky hypermatrix algorithm [12] can be simply derived by considering the matrices in eq. (5.2) as partitioned into submatrices and equating corresponding submatrices on both sides of the equation. Eq. (5.3) is generalized to

$I = 1 \text{ to } N$

$$U_{II}^T U_{II} = (K_{II} - \sum_{M=1}^{I-1} U_{MI}^T U_{MI}) \tag{5.8}$$

solved for the N diagonal submatrices U_{II} ,

$$U_{II}^T U_{IK} = (K_{IK} - \sum_{M=1}^{I-1} U_{MI}^T U_{MK}) \quad K = I+1 \text{ to } N \tag{5.9}$$

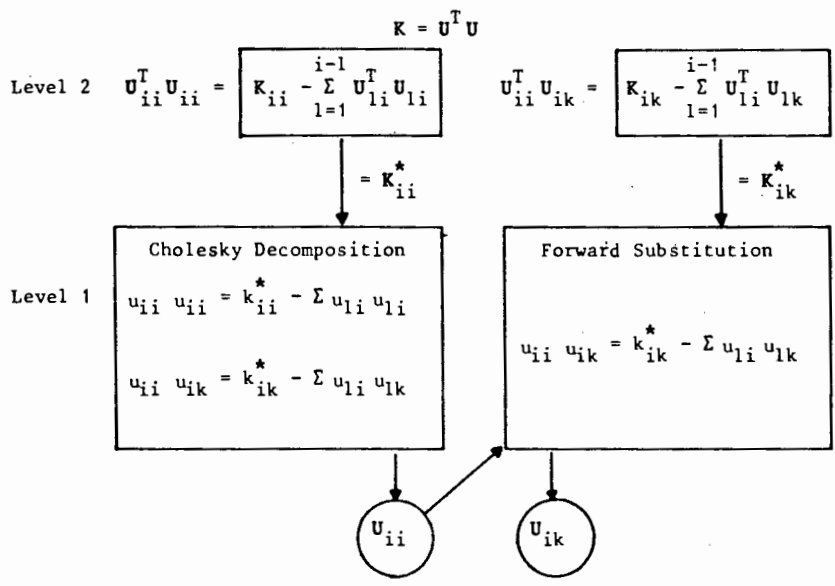
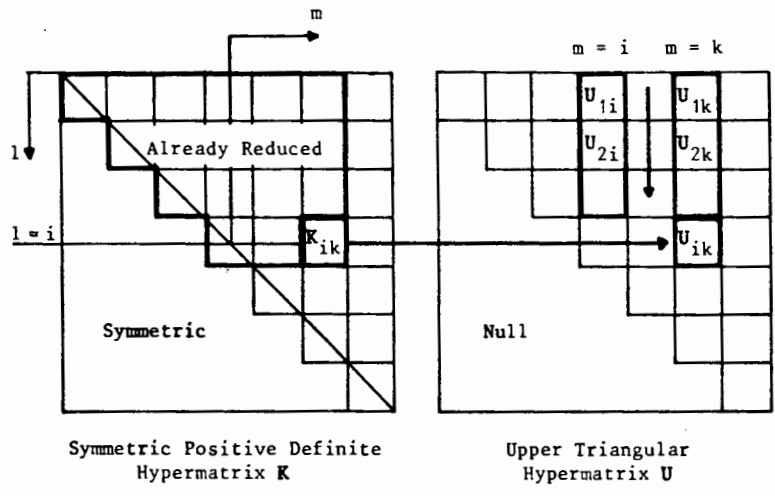


Figure 5.1: Cholesky Hypermatrix Decomposition

solved for the off-diagonal submatrices U_{IK} in row I. After evaluating the symmetric right-hand side of eq. (5.8) called K_{II}^* (figure 5.1), then K_{II}^* is itself triangularized according to eq. (5.3), producing U_{II} . For a typical off-diagonal element U_{IK} in row I, the right-hand side of eq. (5.9), called K_{IK}^* is first evaluated. Comparing eq. (5.4) with eq. (5.9), it is seen that a forward-substitution operation on K_{IK}^* using the triangular matrix U_{II} will produce the result matrix U_{IK}^* , similarly as detailed in eq. (5.5) for the matrices F, U and Y respectively.

After the result matrix U has been produced, the forward-substitution step eq. (5.4) is performed on the loading matrix F. Equating corresponding submatrices in the partitioned form of eq. (5.4), the submatrix scheme of figure (5.2) can be written as follows

$$I = 1 \text{ to } N$$

$$U_{II}^T Y_{IJ} = (F_{IJ} - \sum_{K=1}^{I-1} U_{KI}^T Y_{KJ}) \quad J = 1 \text{ to } M \quad (5.10)$$

where M is the number of vertical partitions (often not more than one) in the loading direction. It is noted that eq. (5.10) is equivalent to eq. (5.9) and after evaluating the right-hand side, it can be solved by the same forward-substitution routine as obtained in eq. (5.5). Finally, as shown in eq. (5.6), the back-substitution process on Y using U produces the displacement matrix x. The hypermatrix formula is

$$I = N \text{ to } 1$$

$$U_{II} x_{IJ} = (Y_{IJ} - \sum_{K=I+1}^N U_{IK} x_{KJ}) \quad J = 1 \text{ to } M \quad (5.11)$$

After evaluating the right-hand side of eq. (5.11) as x_{IJ}^* , the result matrix x_{IJ} can be simply evaluated using the back-substitution routine eq. (5.7) with matrices U , Y and x replaced by U_{II} , x_{IJ}^* and x_{IJ} respectively.

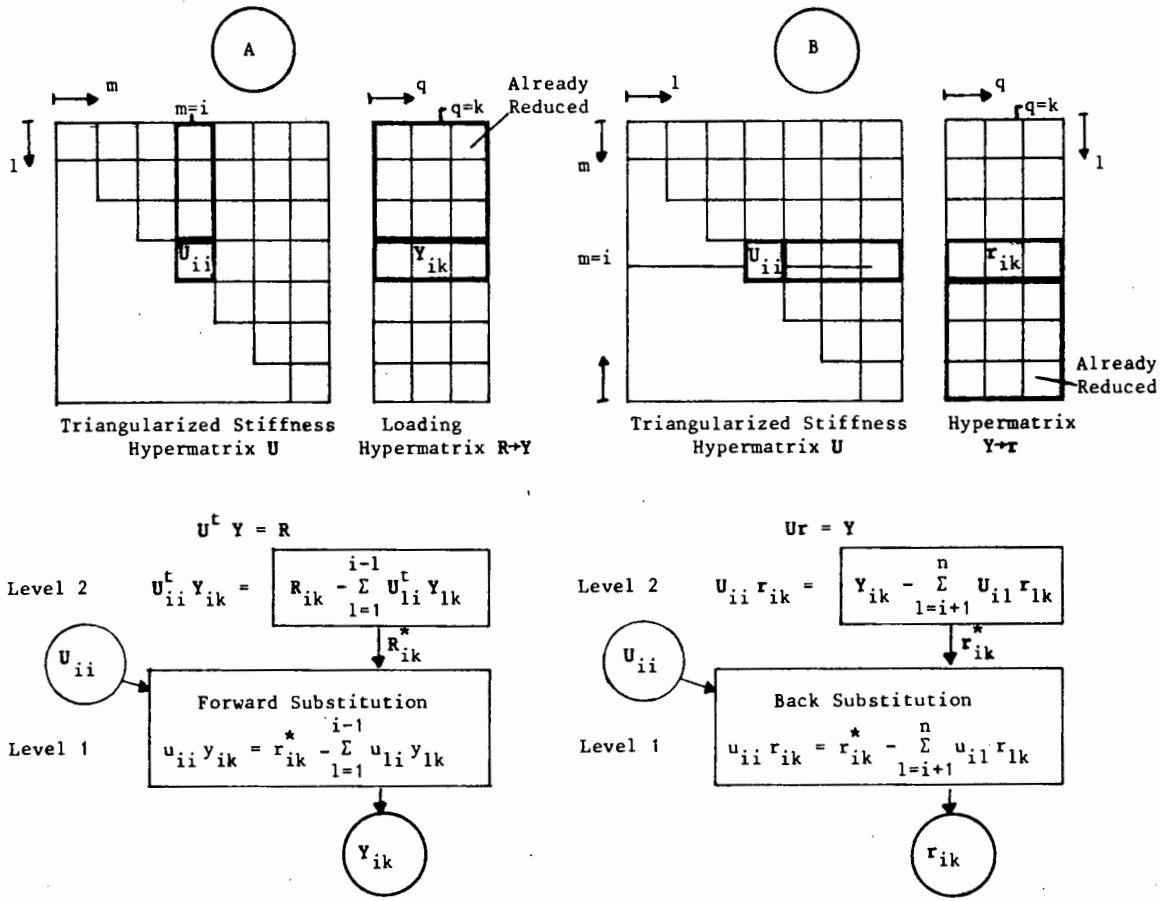


Figure 5.2: Hypermatrix Cholesky Forwarded and Back Substitution

Normally, the intermediate result matrix Y is over-written by the final result matrix x . The loading matrix F can also be optionally over-written by result matrix x , (r in figure 5.2). Since matrix U may

also optionally over-write matrix K, all summation operations in eqs. (5.8) to (5.11) can be regarded as negative multiply-accumulations (decumulations) from the original submatrix, to finally produce the result insitu.

5.3 Assembly of the Global stiffness Matrix

This section describes the static condensation technique used in the assembly of the global stiffness matrix. The objective is to separate the matrix coefficients into different hypermatrices depending on whether the associated degrees of freedom are free, prescribed or suppressed. This means that instead of assembling all the coefficients into a single hypermatrix, they are assembled into six hypermatrices which represent the upper triangular global stiffness matrix.

5.3.1 Concept of static condensation

Given the system of equations

$$Kx = F \quad (5.12)$$

where K is the global stiffness matrix, F is the load vector and x is the unknown displacements. [x] may be represented as

$$[x] = \begin{bmatrix} x_f \\ x_p \\ x_s \end{bmatrix} \quad (5.13)$$

where $[x_f]$ is the vector of unrestrained (free) degrees of freedom, $[x_p]$ is the vector of prescribed degrees of freedom and $[x_s]$ the vector of suppressed degrees of freedom. Similarly we have

$$[K] = \begin{bmatrix} K_{ff} & K_{fp} & K_{fs} \\ K_{pf} & K_{pp} & K_{ps} \\ K_{sf} & K_{sp} & K_{ss} \end{bmatrix} \quad (5.14)$$

and

$$[F] = \begin{bmatrix} F_f \\ F_p \\ F_s \end{bmatrix} \quad (5.15)$$

replacing (5.13), (5.14) and (5.15) into (5.12) we get

$$\begin{bmatrix} K_{ff} & K_{fp} & K_{fs} \\ K_{pf} & K_{pp} & K_{ps} \\ K_{sf} & K_{sp} & K_{ss} \end{bmatrix} \begin{bmatrix} x_f \\ x_p \\ x_s \end{bmatrix} = \begin{bmatrix} F_f \\ F_p \\ F_s \end{bmatrix} \quad (5.16)$$

We now have $[x_f]$ as the unknown displacements, with $[x_p]$ and $[x_s]$ (prescribed or suppressed) known. In order to calculate the unknown displacements we need to solve the following from eq (5.16)

$$x_f = K_{ff}^{-1} \left[F_f - K_{pf} \cdot x_p - K_{fs} \cdot x_s \right] \quad (5.17)$$

Using these values of x_f the reactions may now be calculated as follows:

$$R_p = -F_p + K_{pf} \cdot x_f + K_{pp} \cdot x_p + K_{ps} \cdot x_s \quad (5.18)$$

and

$$R_s = -F_s + K_{sf} \cdot x_f + K_{sp} \cdot x_p + K_{ss} \cdot x_s \quad (5.19)$$

R_p and R_s are the reactions at prescribed and suppressed degrees of freedom. Since x_s is a zero matrix (suppressed degrees of freedom) all multiplication with this matrix may be eliminated from eqs. (5.17) to (5.19). Also since we are considering only symmetric K matrices and only the upper triangular global stiffness matrix is formed the equations may be re-written as:

$$x_f = K_{ff}^{-1} \left[F_f - K_{fp} \cdot x_p \right] \quad (5.20)$$

$$R_p = -F_p + K_{fp}^T \cdot x_f + K_{pp} \cdot x_p \quad (5.21)$$

$$R_s = -F_s + K_{fs}^T \cdot x_f + K_{ps}^T \cdot x_p \quad (5.22)$$

The unknown displacements x_f in eq (6.20) are solved using the Cholesky decomposition scheme. The reactions R_p and R_s are calculated using a general hypermatrix multiplication routine.

5.3.2 Implementation of the Assembly of the Global Stiffness Matrix

The mapping of the element stiffness matrix, k^e , into the global stiffness hypermatrices is implemented with the use of two look-up tables.

The first table, the global degree of freedom indicator table, has two values per tuple. The first value indicates the status of the global degree of freedom, i.e. whether it is free (1), prescribed (2) or suppressed (3). The second value in the tuple is a row/column offset within the respective hypermatrix, see table 5.1. This table was implemented using the hypertable management routines. An illustrative example of how this table is set up is given in the next section.

Global degree of freedom	1	1	1	Column 1:- 1 free degree of freedom
	2	2	1	
	3	1	2	
	4	1	3	Column 2:- Offset within hypermatrix
	5	3	1	

Table 5.1: Global degree of freedom look-up table

The second table, the hypermatrix pointer table, holds the pointers to the various hypermatrices that comprise the total global stiffness matrix, see table 5.2. This table was implemented using a 3 x 3 integer matrix.

PTR_{Kff}	PTR_{Kfp}	PTR_{Kfs}
PTR_{Kpf}	PTR_{Kpp}	PTR_{Kps}
PTR_{Ksf}	PTR_{Ksp}	PTR_{Kss}

where PTR is the pointer
to a hypermatrix

Table 5.2: Hypermatrix Pointer Table

5.3.3 Numerical Example of the Assembly of the Global Stiffness Matrix

In order to demonstrate the global stiffness matrix assembly, an example from 'Finite Elements in Plasticity' [5], (pp. 43) is considered. The structure, a simple axially loaded structure, with a prescribed displacement δ_p at node 2, is subdivided into four elements in each of which a linear displacement variation is assumed, see figure (5.3). At each node i of the element there is only an axial displacement degree of freedom δ_i .

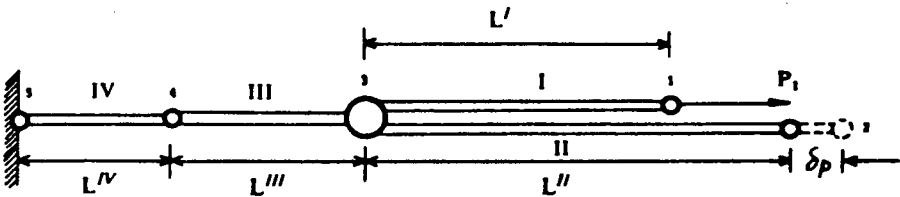


Figure 5.3: Structural example for illustration of assembly of global stiffness matrix

The element stiffness matrix are assumed to be

$$\begin{aligned} k_1 &= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} & k_2 &= \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} \\ k_3 &= \begin{bmatrix} 3 & -3 \\ -3 & 3 \end{bmatrix} & k_4 &= \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix} \end{aligned} \quad (5.23)$$

The element incidences are assumed

$$x_1 = \begin{bmatrix} \delta_3 \\ \delta_1 \end{bmatrix} \quad x_2 = \begin{bmatrix} \delta_3 \\ \delta_2 \end{bmatrix} \quad x_3 = \begin{bmatrix} \delta_4 \\ \delta_3 \end{bmatrix} \quad x_4 = \begin{bmatrix} \delta_5 \\ \delta_4 \end{bmatrix} \quad (5.24)$$

The vector of applied nodal forces for each element is

$$F_1 = \begin{bmatrix} P_1 \\ 0 \end{bmatrix} \quad F_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad F_3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad F_4 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5.25)$$

where $P_1 = 10$

The following prescribed and suppressed values are assumed

$$\delta_2 = \delta_p = 2 \text{ and } \delta_5 = 0 \quad (5.26)$$

The global degree of freedom table is now set up as shown in table 5.3.

	1	1	1	$\delta_1, \delta_3, \delta_4$ free
Global	2	2	1	δ_2 prescribed
degree of	3	1	2	δ_5 suppressed
freedom	4	1	3	
	5	3	1	

Table 5.3: Global degree of freedom look-up table

Each element may now be assembled into the global stiffness matrix. The steps for adding each element coefficient to the global stiffness matrix are:

1. Retrieve global degree of freedom values from the element incidences.
2. From column 1 of the global degree of freedom table find the index in the hypermatrix pointer table.
3. Retrieve the pointer to the particular hypermatrix.
4. From column two of the global degree of freedom table find the row/column offset within the particular hypermatrix.
5. Add the element coefficient to the particular hypermatrix using the hypermatrix pointer and the offset, i.e. row and column index.

Table 5.3 shows the calculation of addresses for the assembly of the element coefficients into the global stiffness matrices, and figure 5.5 shows a flow diagram for the assembly of a single element coefficient.

Element Coefficient	Global Dofs	Hypermatrix Table Index Row, Column	Hypermatrix Pointer	Hypermatrix Offset Row, Column
k_{11}^1	3,3	1,1	K_{ff}	2,2
k_{12}^1	3,1	1,1	K_{ff}	1,2
k_{22}^1	1,1	1,1	K_{ff}	1,1
k_{11}^2	3,3	1,1	K_{ff}	2,2
k_{12}^2	3,2	1,2	K_{fp}	2,1
k_{22}^2	2,2	2,2	K_{pp}	1,1
k_{11}^3	4,4	1,1	K_{ff}	3,3
k_{12}^3	4,3	1,1	K_{ff}	2,3
k_{22}^3	3,3	1,1	K_{ff}	2,2
k_{11}^4	5,5	3,3	K_{ss}	1,1
k_{12}^4	5,4	1,3	K_{fs}	3,1
k_{22}^4	4,4	1,1	K_{ff}	3,3

Note: Only upper triangular element coefficients are assembled

Table 5.3: Assembly of Global Stiffness Matrix

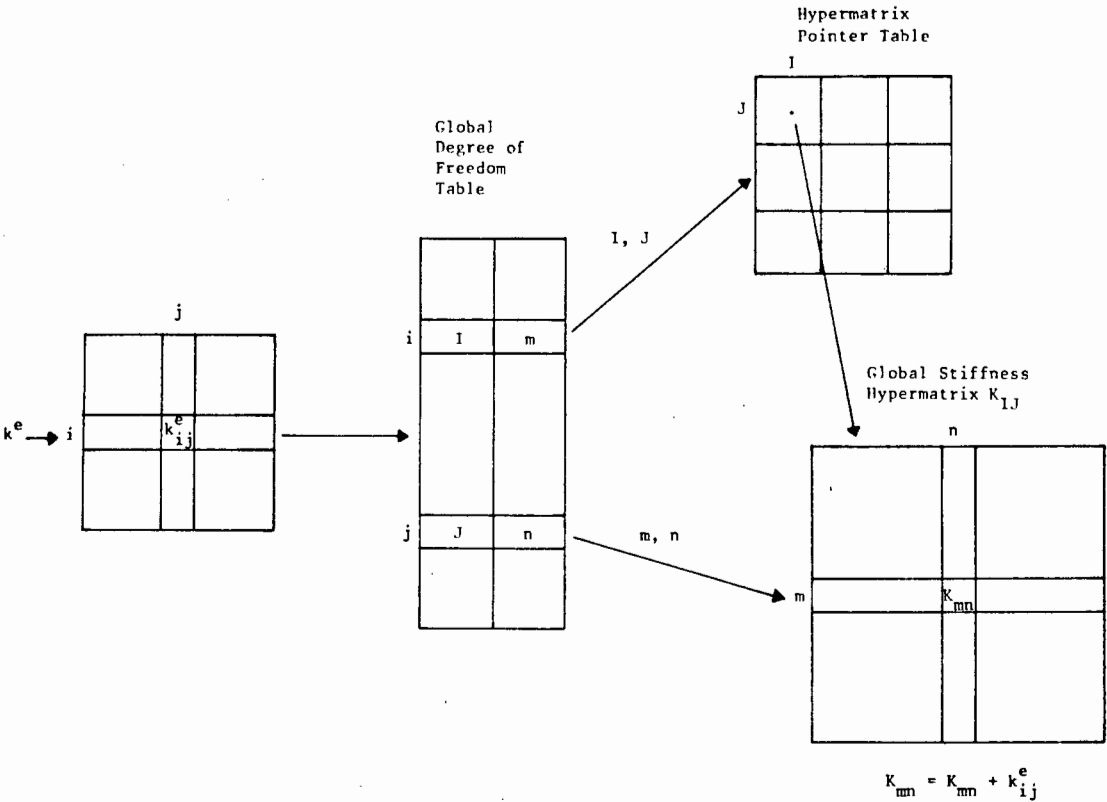


Figure 5.4: Mapping of global dof into hypermatrices

The procedure for the assembly of the global load vector is similar except that the load hypermatrix pointer table is 1 x 3 integer array.

The fully assembled global stiffness matrices and load vector would be in the form

$$K_{ff} = \begin{bmatrix} 1 & -1 & 0 \\ & 6 & -3 \\ & & 7 \end{bmatrix} \quad K_{fp} = \begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix} \quad K_{fs} = \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix} \quad F_s = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}$$
$$K_{pp} = \begin{bmatrix} 2 \end{bmatrix} \quad K_{ps} = \begin{bmatrix} 0 \end{bmatrix} \quad F_p = \begin{bmatrix} 0 \end{bmatrix}$$
$$K_{ss} = \begin{bmatrix} 4 \end{bmatrix} \quad F_s = \begin{bmatrix} 0 \end{bmatrix}$$

Figure 5.5: Assembled global stiffness matrix

5.4 The Equation Solution Routines

The Cholesky Hypermatrix Equation Solution Scheme CHESS, was implemented using several layers of software, see figure 5.6. The data management routines of the DAS package and the hypermatrix and hypertable routines were supplied by Vos [17]. The equation solution routines were implemented as the next layer of software. Finally the equation solution interface routines were developed as a layer of software between the lower level routines and the user application program.

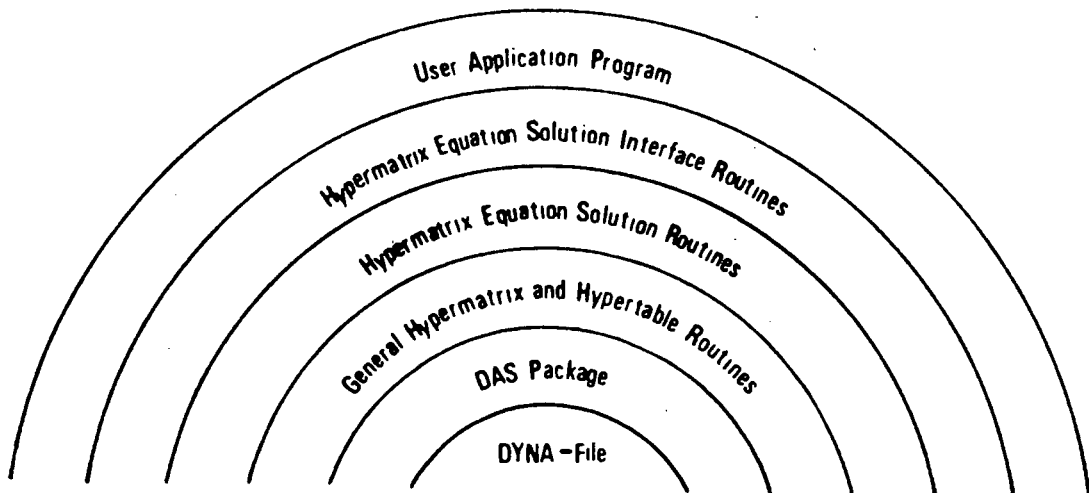


Figure 5.6: Layers of software for CHESS package

5.4.1 Equation Solution Routines Description

This section gives a brief description of the purpose of the routines in the equation solution package.

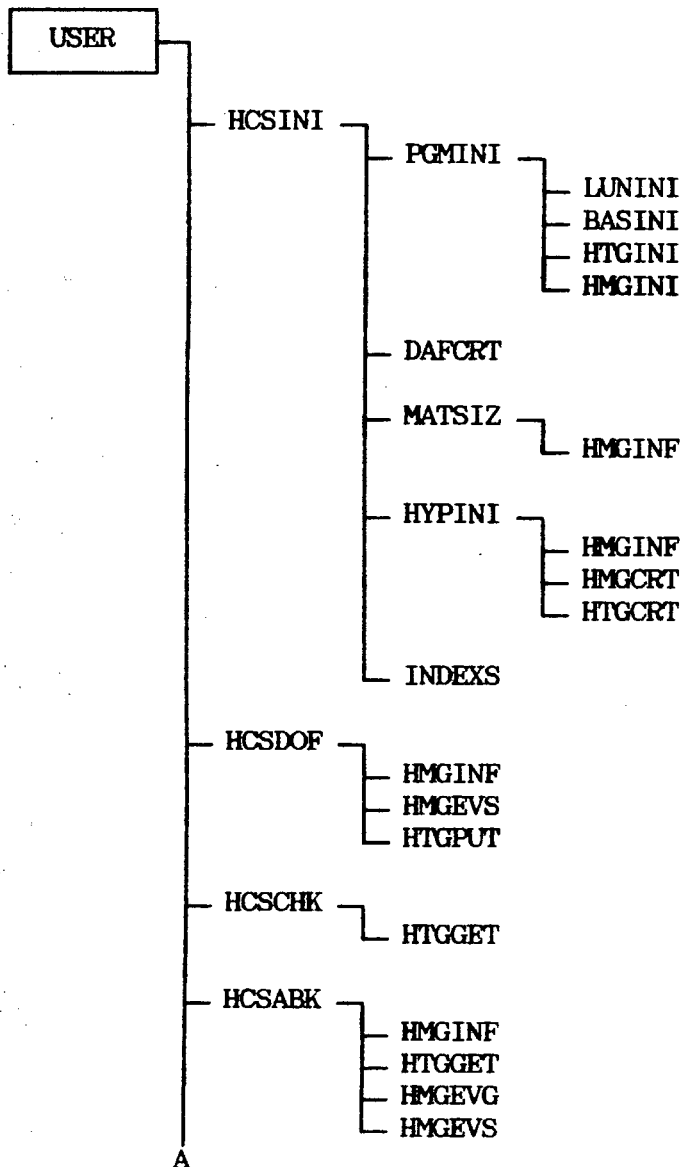
HCSINI	Initializes the equation solution procedure
HCSDOF	Specifies a global degree of freedom in hypertable
HCSCHR	Checks that the global degree of freedom table is completely defined

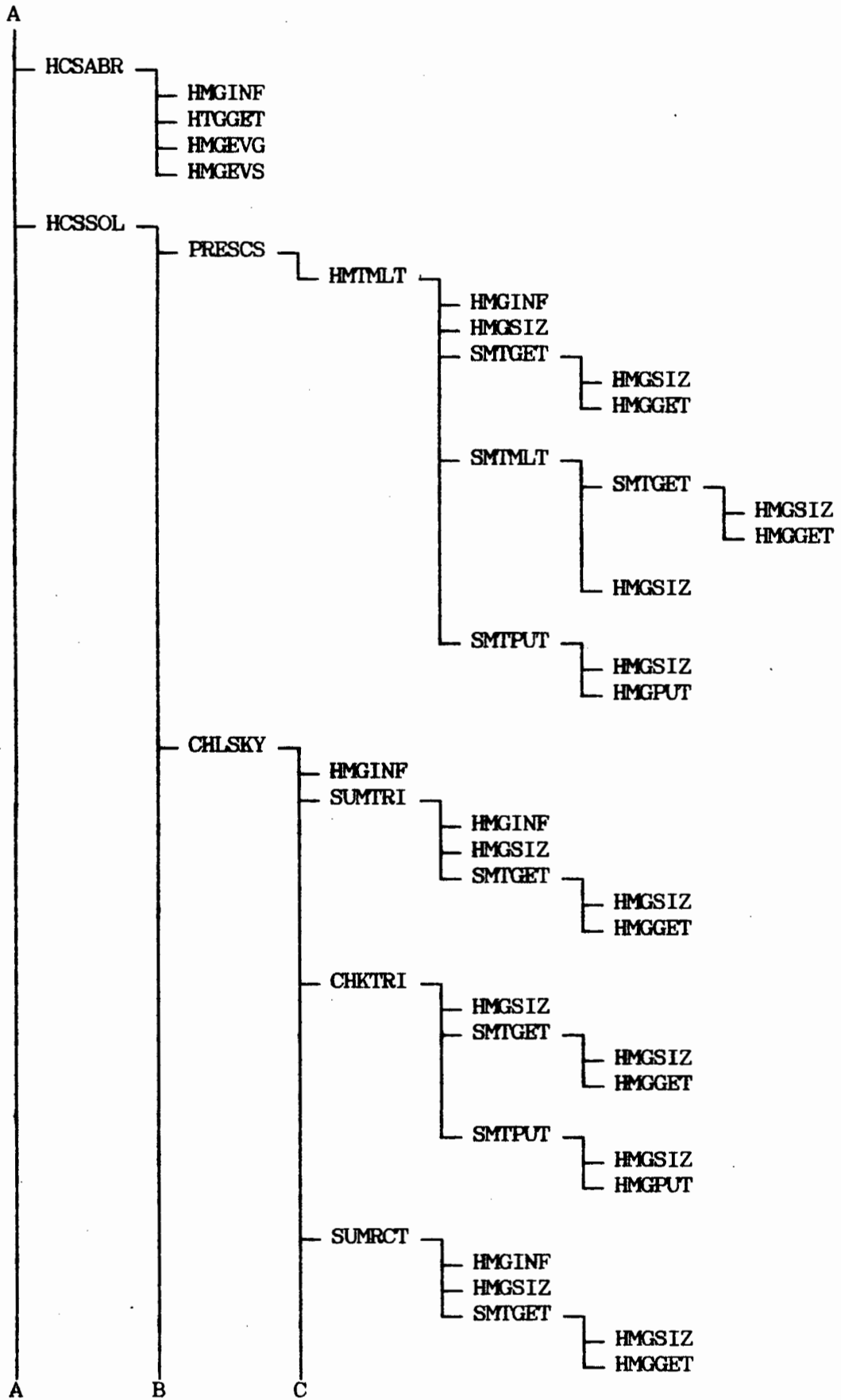
HCSABK	Assembles element stiffness matrix into global stiffness hypermatrices
HCSABR	Assembles element load vector in global load vector
HCSSOL	Controls solution of equations and calculation of reactions
HCSUGT	Retrieves calculated displacement
HCSRGT	Retrieves calculated reaction
HCSDMP	Dumps specified hypermatrix
PGMINI	Initializes program module
MATSIZ	Calculates size of sub-matrice for this solution
DAFCRT	Creates and format DYNA-file
HYPINI	Initializes hypermatrices and hypertable
INDEXS	Initializes indirect pointer look-up tables
PRESCS	Adjusts load vector for prescribed displacement
CHLSKY	Driver routine for Cholesky equation solution procedure
CHKTRI	Calculates L matrix for diagonal sub-matrices
CHKRCT	Calculates L matrix for non-diagonal sub-matrices
CHKYYY	Calculates Y matrix, forward substitution
CHKXXX	Calculates X matrix, backward substitution
SUMTRI	Calculates summation for diagonal sub-matrices
SUMRCT	Calculates summation for non-diagonal sub-matrices
SUMYYY	Calculates summation for Y submatrix
SUMXXX	Calculates summation for X submatrix
SMTPUT	Stores submatrix in DYNA-file
SMTGET	Retrieves submatrix from DYNA-file
CREACT	Calculates reactions
DPIHMT	Dumps integer hypermatrix
DPRHMT	Dumps real hypermatrix
HMTMLT	Multiplies hypermatrices
SMTMLT	Multiplies submatrices

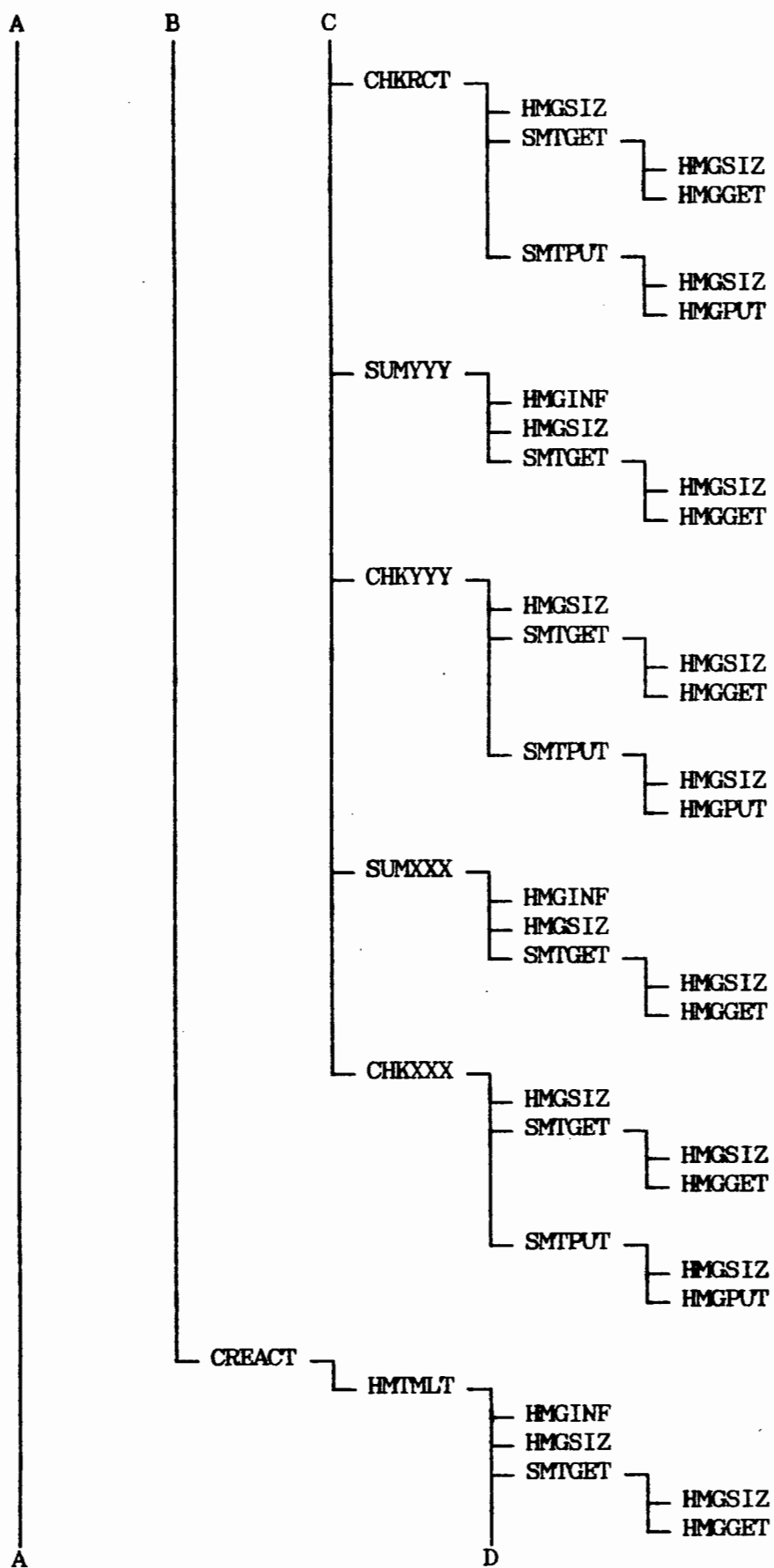
A full description of all the routines and their usage is given in Appendix B. A listing of the source code for the routines is given in Appendix C.

5.4.2 Structure Chart for Equation Solution Routines

The call chart from the user level to the hypermatrix and hypertable routines is shown in figure 5.7, depicting the inter-relationship between the routines:







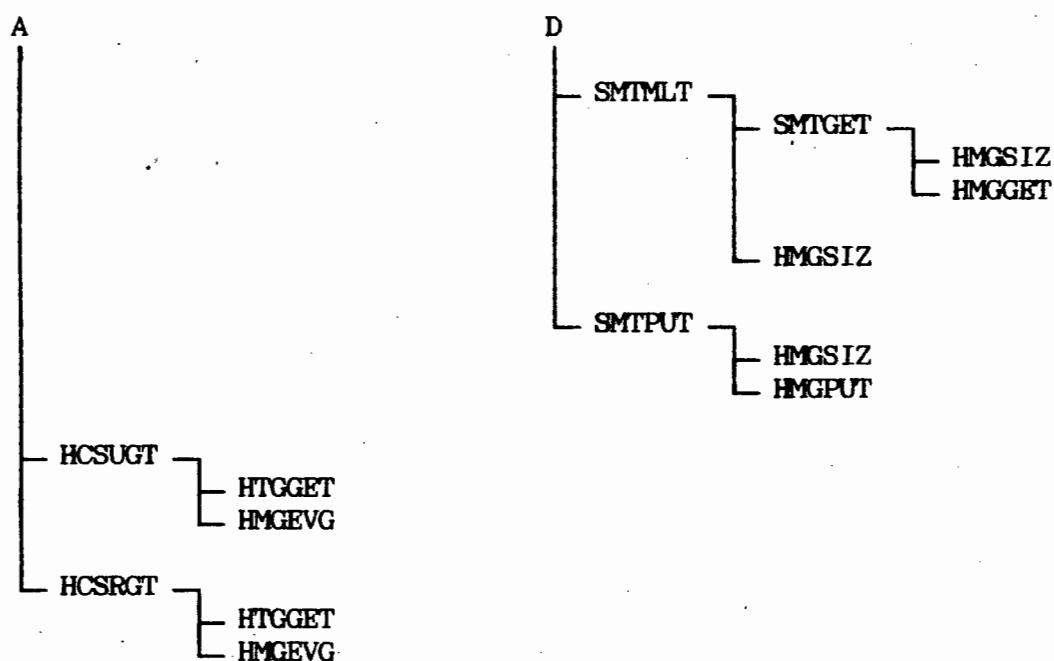


Figure 5.7: Call chart for equation solution package

5.4.3 The Chess Suite: A Sample Program

This section describes the implementation of a simple program to demonstrate how the equation assembly and solution process may be performed.

The problem considered is the one described in section 5.3.3. The free global degrees of freedom are δ_1 , δ_3 and δ_4 . The prescribed degree of freedom is $\delta_2=2$. The suppressed degree of freedom is $\delta_5=0$.

The program listing is:

```

c
c.....
c
c   Description of the program TEST:
c   -----
c
c   This program is designed to test the use of the hypermatrix
c   solver for symmetric matrices using the cholesky algorithm.
c   It uses the following test matrix (from Finite Elements in
c   Plasticity by DRJ Owen and E Hinton, page 43):
c
c   Where:
c    $K1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$   $K2 = \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix}$   $K3 = \begin{bmatrix} 3 & -3 \\ -3 & 3 \end{bmatrix}$   $K4 = \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix}$ 
c
c    $x1 = [u3 \ u1]$   $x2 = [u3 \ u2]$   $x3 = [u4 \ u3]$   $x4 = [u5 \ u4]$ 
c
c    $F1 = [0 \ 10]$   $F2 = [0 \ 0]$   $F3 = [0 \ 0]$   $F4 = [0 \ 0]$ 
c
c   With prescribed constraints:
c       u2 = 2 and u5 = 0
c
c   Displacements:
c   u1 = 13.7692 u2 = 2.000 u3 = 3.7692 u4 = 1.6154 u5 = 0.000
c   Reactions:
c   R2 = -3.5385 R5 = -6.4615
c.....
c
c   program TEST
c
c   double precision estif1(2,2),estif2(2,2),estif3(2,2),estif4(2,2),
c   *           eload1(2), eload2(2), eload3(2), eload4(2),
c   *           fixed(5), bu(5), br(5)
c   double precision HCSUGT,HCSRGT
c   integer iedof1(2),iedof2(2),iedof3(2),iedof4(2),ibcdof(5)
c
c--- element stiffnesses
c   data estif1 / 1.0d0, -1.0d0,
c   *           -1.0d0, 1.0d0/
c   data estif2 / 2.0d0, -2.0d0,
c   *           -2.0d0, 2.0d0/
c   data estif3 / 3.0d0, -3.0d0,
c   *           -3.0d0, 3.0d0/
c   data estif4 / 4.0d0, -4.0d0,
c   *           -4.0d0, 4.0d0/
c
c--- element loads
c   data eload1 / 0.0d0,10.0d0/
c   data eload2 / 0.0d0, 0.0d0/
c   data eload3 / 0.0d0, 0.0d0/
c   data eload4 / 0.0d0, 0.0d0/
c
c--- boundary condition fixities
c   data fixed / 0.0d0, 2.0d0, 0.0d0, 0.0d0, 0.0d0 /

```

```

c
c--- boundary conditions
data ibcdof / 1,2,1,1,3 /
c
c--- element incidences
data iedof1 / 3,1 /
data iedof2 / 3,2 /
data iedof3 / 4,3 /
data iedof4 / 5,4 /
c
c--- initialize hypermatrix cholesky solver module
nfree = 3
npres = 1
nsupp = 1
isymm = 1
ncase = 1
luerr = 6
c
call HCSINI (nfree,npres,nsupp,isymm,ncase,luerr)
c
c--- set up global degree of freedom table
ntotv = nfree + npres + nsupp
do 10 k = 1,ntotv
    call HCSDOF (k,ibcdof(k),fixed(k))
10 continue
call HCSDMP ('D')
c
c--- assemble global stiffness matrix
nevab = 2
call HCSABK (nevab,estif1,iedof1)
call HCSABK (nevab,estif2,iedof2)
call HCSABK (nevab,estif3,iedof3)
call HCSABK (nevab,estif4,iedof4)
call HCSDMP ('K')
c
c--- assemble global load vector
iload = 1
call HCSABR (2,eload1,iedof1,iload)
call HCSABR (2,eload2,iedof2,iload)
call HCSABR (2,eload3,iedof3,iload)
call HCSABR (2,eload4,iedof4,iload)
call HCSDMP ('R')
c
c--- solve
call HCSSOL
c
c--- get displacements and reactions and display them
lunot = 6
do 20 k = 1,ntotv
    bu(k) = HCSUGT (k,iload)
    br(k) = HCSRGT (k,iload)
20 continue

```

```

20    continue
      write(lunot,1013)
1013  format(/lh ,'Results:')
      do 30 k = 1,ntotv
          write(lunot,1003) k,bu(k),br(k)
1003  format(lh ,'Dof=',i2,' U=',F12.5,' R=',F12.5)
30    continue
c
c---  terminate
      stop ' > End of TEST'
      end
c

```

Figure 5.8: Sample Program Listing - TEST

The results are:

Kff	matrix	3 by	3
address matrix		1 by	1
submatrices		3 by	3
matrix type		2	
words per element		2	
Kfp	matrix	3 by	1
address matrix		1 by	1
submatrices		3 by	1
matrix type		2	
words per element		2	
Kpp	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Kfs	matrix	3 by	1
address matrix		1 by	1
submatrices		3 by	1
matrix type		2	
words per element		2	
Kps	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Uf	matrix	3 by	1
address matrix		1 by	1
submatrices		3 by	1
matrix type		2	
words per element		2	

Up	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	

Rp	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	

Rs	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	

Fixity	table	5 by	2
--------	-------	------	---

Dof	Table	Index
1	1	1
2	2	1
3	1	2
4	1	3
5	3	1

Hypermatrix Kff

> Submatrix(1, 1) :

	1	2	3
1	1.0000	-1.0000	0.00000E+00
2	0.00000E+00	6.0000	-3.0000
3	0.00000E+00	0.00000E+00	7.0000

Hypermatrix Kfp

> Submatrix(1, 1) :

	1
1	0.00000E+00
2	-2.0000
3	0.00000E+00

Hypermatrix Kfs

> Submatrix(1, 1) :

	1
1	0.00000E+00
2	0.00000E+00
3	-4.0000

Hypermatrix Kpf

> Submatrix(1, 1) :

	1
1	0.00000E+00
2	-2.0000
3	0.00000E+00

Hypermatrix Kpp

```
> Submatrix( 1, 1) :
      1
1      2.0000
```

Hypermatrix Kps

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Hypermatrix Ksf

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
2      0.00000E+00
3      -4.0000
```

Hypermatrix Ksp

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Hypermatrix Rf

```
> Submatrix( 1, 1) :
      1
1      10.000
2      0.00000E+00
3      0.00000E+00
```

Hypermatrix Rp

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Hypermatrix Rs

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Results:

Dof= 1 U=	13.76923	R=	0.00000
Dof= 2 U=	2.00000	R=	-3.53846
Dof= 3 U=	3.76923	R=	0.00000
Dof= 4 U=	1.61538	R=	0.00000
Dof= 5 U=	0.00000	R=	-6.46154

Figure 5.9: Sample program output file

5.5 Discussion

The CHESS suite has been implemented and tested, on a MicroVAX II, in the NEW NOSTRUM program package. Results are compared with ABAQUS in Appendix D, and the comparison is favourable. The package should be considered as a prototype and therefore only the numerical accuracy of the results has been compared with other finite element packages. Methods of improving the efficiency, as described in section 6.3.2, still need to be implemented before a comparison of solution times may be made. The package has been implemented as a double precision version, however the changes for a single precision version are easily implemented for a different machine.

CHAPTER SIX

SUMMARY AND CONCLUSIONS

This chapter describes the software development strategy employed, as well as problems encountered and experience gained from this project. The work is summarized and conclusions drawn. Finally, suggestions for possible future work are made.

6.1 Software development strategy

6.1.1 Software design

One of the major criteria of the development of the program suite was that the software modules that were developed had to be highly modular, in order that they may be re-used in different contexts. This required that modules exhibit high functional or informational strength (that is, each module contains a strong internal relationship) and low module coupling (that is, each module has weak intermodule relationships with all the other modules in the program) [32].

The principle of information hiding was also a major design consideration for the modules (that is, modules were designed to consist of several routines which performed related functions on a particular resource, such as a common data structure) [17].

Another consideration was the design of the user interface to these modules. These had to be as simple as possible to allow the program suite to be implemented easily in any finite element program. For

example, the interface routines to the equation solution module have to provide information on the system of equations, assemble the element stiffness matrices into the global stiffness matrix, initiate the solution process and be able to retrieve results.

An important aspect of the software design and implementation was that all code be extensively documented. Code that is not documented or is poorly documented becomes 'throw-away' code and has to be redone once the original developer is no longer available for consultation. Therefore it was required that all software should be documented both internally and externally.

Internal documentation includes any explanations that are inserted directly into the code in the form of comment statements. Every module of code should have a comprehensive descriptive header block prologue which describes the module's purpose, includes routines, the developer, etc. Each routine in the module has a prologue describing the routine's purpose and its input/output variables, as well as any necessary remarks deemed necessary to make the code as easy to read as possible. Comment statements are inserted throughout the code to explain the various processing steps.

External documentation consists of a description of the algorithms implemented in the code, diagrams of the data structures, call charts, etc. Test examples must be explained with proper references. If more than one version of the code exists then the version numbers and the different capabilities of the update must be fully documented. Test examples for particular version numbers must also be kept.

6.1.2 Software Implementation

Initial software development and testing was done on the University of Cape Town's SPERRY UNIVAC 1100 computer, while the final implementation of the CHESS package was done on a MicroVAX II (in the Applied Mechanics Research Unit) and integrated into the NEW NOSTRUM development package. The testbed used during development on the UNIVAC 1100 was derived from the program PLANE [2].

Several smaller programs were coded and tested before the CHESS suite was developed.

The finite element program PLANE [2], which has an incore frontal solver, for the plane stress/strain problems was implemented and tested. Results for the given plane strain example were good, unlike the results for plane stress problems. Thus the plane strain examples were used primarily for comparison and checking further developments.

The original frontal solution code of Irons [4] was implemented and tested as this had been the basis for the Beer and Haas [6] package and it was hoped this would provide the necessary understanding of the use of variables and the underlying method in the partitioned frontal solution. Next the code published by Beer and Haas was implemented but would not run to completion. Several attempts were made to debug the code, but due to lack of sufficient documentation and the complexity of the code, it was decided not to proceed with that particular solution procedure.

A hypermatrix scheme for the storage of the frontal stiffness matrix was then implemented in the frontal solution routine of the program PLANE in order to gain some experience with the use of hypermatrices. As expected, this was not efficient due to the random accessing of submatrices and the resulting high number of I/O operations.

It was then decided to implement the Cholesky decomposition procedure with a hypermatrix data management scheme. Initially an incore solution algorithm was developed, which is available for use with problems with small numbers of degrees of freedom. The hypermatrix management scheme and the Cholesky procedure were then integrated into a single program suite. Next the program suite interface routines were developed and thoroughly tested. Finally the code was transferred to the Applied Mechanics Research Unit's MicroVAX II and integrated into the NEW NOSTRUM development code and the results compared with the ABAQUS finite element package.

6.2 Summary

During the first stage of this project an investigation of the various equation solution techniques was done. From this survey it was found that at present direct methods are favoured over iterative methods, however this may change with improving hardware technology. The various triangularisation and factorisation methods are mathematically closely related and no one method appears to have any significant computational advantage over any other method. The advantage of one solution scheme over the other is related to the data handling scheme.

Efficiency may be achieved by exploiting the sparsity of the system of equations which results in a minimum number of I/O operations.

For a robust and reliable code which is easy to upgrade the solution technique needs to be as modular and straight-forward as possible. Firstly a data management scheme should provide the first layer of software to store and retrieve the equations. The solution scheme should be possible through a series of procedures which are controlled by high level routines and performed by lower level routines. A set of interface routines to the equation solution scheme needs to be implemented which enables any user to simply initiate the few processes (such as initialisation, assembly, solution and result retrieval) without having any knowledge of the solution method itself.

The second stage of the project involved the detailed investigation and implementation of an in-core wavefront solver with the intention of using a partitioning scheme to remove the problem size limitations and to provide some more enhanced features. It was found that the data management of the frontal solution scheme was so closely enmeshed with the solution algorithm and that the partitioning scheme would not provide the desired level of modularity.

The hypermatrix scheme was developed which provided the basic data handling capability for a simpler and more modular solution technique. General hypermatrix and hypertable routines provided the interface to the DAS package which implements a virtual memory paging scheme. Hypermatrix procedures were developed as a module for the assembly, solution and retrieval of results. Finally a module of interface routines was provided for the easy application of the equation solution scheme in finite element programs.

6.3 Suggestions For Further Improvements

6.3.1 Unsymmetric matrices

The equation solution suite presented already has many of the facilities required for the solution of a system of unsymmetric matrices except for the solution of the $x = K^{-1}F$ where K is unsymmetric. This limitation may be removed if the Cholesky algorithm, for symmetric matrices, is complemented with the more general Crout's algorithm, for unsymmetric matrices.

An implementation of the Crout algorithm as presented by Taylor[18] has been implemented as an incore solver. Taylor has chosen to use the profile scheme for data management, but an implementation using the existing hypermatrix scheme based on the Cholesky algorithm would be simpler.

6.3.2 Further means of improving efficiency

Several methods of speeding up calculation times, by the appropriate avoidance of zero multiplications within the operations of submatrices, are described by Schrem [12]. In order to avoid wasteful tests on zeroes within the innermost loops, the sparsity structure of the various submatrices must be related to the required matrix operations, particularly during the triangularization. The five necessary operations on submatrices during the equation solution, observed from eqs. (5.8) to (5.11), can be examined as to special action to exploit sparsity:

1. The matrix multiplication $R = R - A^T B$ occurs in eqs. (5.8), (5.9) and (5.10), and in problems with relatively large bandwidths usually accounts for more than 80% of the total numerical operations during the equation solution. Submatrix B always lies further to the outside of the "band" than submatrix A, and will thus have the greater likelihood of being only partially filled. During the forward-substitution, matrix B (i.e. matrix Y_{KJ} in eq. (5.10)) can also contain many zeroes if the loading conditions are differently distributed. Thus the majority of the zero operations can be detected if the columns of matrix B are tested for zero values. In addition, because the submatrices in both the triangularization and forward-substitution always tend to fill up in a downward direction, the zero test can be reasonably terminated upon the occurrence of the first non-zero value in a column of B.

2. The Cholesky triangularization operation $U_{II}^T U_{II} = K_{II}^*$ occurs in eq. (5.8) on the diagonal submatrices. As the diagonal submatrices are nearly always full for larger problems, tests on zeroes are not justified for this procedure.

3. The forward-substitution operation $U_{II}^T X = A$ occurs in both eqs. (5.9) and (5.10). Although matrix U_{II} is normally full, the result matrix X may sometimes be only partially full, especially near the outside of the band, and also fills up in a downward direction. If the outer loop of this operation is made over the number of columns in X, then a simple downward test to detect the first non-zero element in each column of X shortens the two inner loops of the operation correspondingly.

4. The multiplication operation $R = R - AB$ occurs in the back-substitution procedure eq. (5.11). Although the "result" matrix B, i.e. x_{KJ} in eq (5.11) is always full, the pre-multiplier A may only be partly filled near the edge of the band. In this case rows of A are involved. The innermost loop can be completely skipped upon the identification in the next outermost loop of individual zero values in row i of matrix A. Although the accumulation feature into the result matrix R is thereby lost, the net savings are usually significant, especially when several loading conditions are present.
5. For the back-substitution operation $U_{II}x_{IJ} = x_{IJ}^*$ in eq (5.11) no tests should be made, because x_{IJ} is always full and U_{II} also usually full.

The execution speed may be improved by writing some of the low level routines and I/O routines in machine code, however, as a previously mentioned this would complicate the code and reduce its portability. The equivalent FORTRAN routine should always be supplied for initial installation and the machine code routines programmed by the systems programmer.

6.3.3 Renumbering Algorithms

One of the main difficulties in practical application of optimal ordering algorithms is that in the very large problems where such a procedure should be most beneficial, the cost of the re-ordering process may exceed the resulting savings, especially when ordering is performed on single rows and columns [12]. However, re-ordering

algorithms working on the address matrix of the structural stiffness matrix may produce significant savings. This address matrix, which provides a macro representation of the sparsity, is rarely larger than 200 by 200, even for large problems. The cost of the re-ordering process itself takes only 1% to 2% of the triangularization time [12].

6.3.4 Concurrent Processing

Methods performing concurrent Cholesky factorisation of positive-definite matrices have been developed [9]. The technology of concurrent processing is relatively new but appears to be the direction of future developments. This new technology should be exploited as far as possible in the development and implementation of any new code in the future.

CHAPTER SEVEN

REFERENCES

- [1] Wilkinson, J.H., 'The Algebraic Eigenvalue Problem', Clarendon Press, Oxford, England, 1965.
- [2] Hinton, E., Owen, D.R.J., 'Finite Element Programming', Academic Press Inc., Swansea, England, 1977.
- [3] Lindgren, L., 'Design of Small and Effective Finite Element Systems', Lulea University, Sweden, Eng. Computing, 3, pp 202-208, September, 1986.
- [4] Irons, B., 'A Frontal Solution Program for Finite Element Analysis', Uni. Wales, Swansea, Int. J. Num. Meth. Eng., 2, pp 5-32, 1970.
- [5] Owen, D.R.J., Hinton, E., 'Finite Elements in Plasticity. Theory and Practice', Pineridge Press Ltd., Swansea, U.K., 1977.
- [6] Beer, G., Haas, W., 'A Partitioned Frontal Solver for Finite Element Analysis', Int. J. Num. Meth. Eng., 18, pp 1623-1654, 1982.
- [7] Irons, B., Ahmad, S., 'Techniques of Finite Elements', John Wiley & Sons, Chichester, 1986.
- [8] Bettess, J.A., 'A Data Structure for Finite Element Analysis', Int. J. Num. Meth. Eng., 11, pp 1779-1799, 1977.

- [9] Utka, S., Salama, M., Melosh, R.J., 'Concurrent Factorization of Positive Definite Banded Hermitian Matrices', Int. J. Num. Meth. Eng., 23, pp 2137-2152, 1986.
- [10] George, A., 'Sparse Matrix Methods of the Finite Element Method', Dept. Comp. Sci., Uni. Waterloo, Waterloo, Ontario, Canada.
- [11] Melosh, R.J., Bamford, R.M., 'Efficient Solution of Load-deflection Equations', J. Struct. Div. ASCE, 95, pp 661-676, 1969.
- [12] Fuchs, G.v., Roy, J.R., Schrem, E., 'Hypermatrix Solution of the Stiffness Equations', ISD ASKA Report PM 109, Institut fur Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, University of Stuttgart, Stuttgart, 1972.
- [13] Felippa, A.C., 'Solution of Linear Equations with Skyline-stored Symmetric Matrix', Comp. & Struct., 5, pp 13-29, 1975.
- [14] Agrawal, A.B., Mufti, A.A., Jaeger, L.G., 'Band-schemes vs. Frontal-routines in Nonlinear Structural Analysis', Int. J. Num. Meth. Eng., 15, pp 753-766, 1980.
- [15] Kreyszig, E., 'Advanced Engineering Mathematics', John Wiley and Sons, New York, 4th edition pp 805-809, 1979.
- [16] Vos, J., 'Struktur Leer', 4th Year Engineering Course Notes, University of Stellenbosch, pp 1.22-1.27.

- [17] Vos, J., 'Implementing a Finite Element Program on a Microcomputer using a Dynamic Memory Allocation Scheme', Master's Thesis, Uni. Stellenbosch, 1985.
- [18] Taylor, R.L., 'Solution of Equations by a Profile Solver', Eng. Comput., 2, pp 344-350, 1985.
- [19] George, A., Liu, J.W.H., 'The Design of a User Interface for a Sparse Matrix Package', ACM Transactions on Mathematical Software, 5, no.2, pp 139-152, 1979.
- [20] Servranckx, D., Mufti, A.A., 'Data Structures for Finite Element Modelling', Eng.Comput., 3, pp 27-35, 1986.
- [21] Dhatt, G., Touzot, G., 'The Finite Element Displayed', John Wiley & Sons, 1984.
- [22] Zienkiewicz, O.C., 'The Finite Element Method', 3rd Edt., pp 677-757.
- [23] Hood, P., 'Frontal Solution Program for Unsymmetric Matrices', Int. J. Num. Meth. Eng., 10, pp 379-399, 1976.
- [24] Cuthill, E., McKee, J., 'Reducing the Bandwidth of Sparse Symmetric Matrices', Proc. 24th Nat. Conf. Assoc. Comput. Mach., ACM Publ., pp 69, 1969.
- [25] Wilson, E.L., Bathe, K.L., Doherty, W.P., 'Direct Solution of Large Systems of Linear Equations', Comp. and Struct., 4, pp 363-372, 1974.

- [26] Jennings, A., 'A Compact Storage Scheme for the Solution of Symmetric Simultaneous Equations', *Comput. J.*, 9, pp 281-285, 1966.
- [27] Pina, H.L.G., 'An Algorithm for Frontwidth Reduction', *Int. J. Num. Meth. Eng.*, 17, pp 1539-1546, 1981.
- [28] Alizadeh, A., Will, G.T., 'A Substructured Frontal Solver and its Application to Localized Material Non-linearity', *Comp. Struct.*, 10, pp 225-231, 1979.
- [30] Abbas, S.F., 'Some Novel Applications of the Frontal Concept', *Int. J. Num. Meth. Eng.*, 15, pp 519-536, 1980.
- [31] Bell, K., Hatlestad, B., Hansteen, O.E., Araldsen, P.O., 'NORSAM-Users Manual', February 1973.
- [32] Myers, G.J., *Composite/Structural Design*, Van Nostrand Reinhold, New York, 1978.
- [33] Kamel, H.A., McCabe, M.W., 'Direct Numerical Solution of Large Sets of Simultaneous Equations', *Computers and Structures*, 9, pp 113 - 123, 1978.
- [34] Rice, J.R., 'Matrices and Linear Equations'. In *Numerical Methods, Software and Analysis*, McGraw-Hill, New York, 1983.
- [35] Lestingi, J., Prachuktam, S., 'A Blocking Technique for Large Scale Structural Analysis', *Computers and Structures*, 3, pp 669-714, 1973.

- [36] Schrem, E., 'Computer Implementation of the Finite Element Procedure'.
In Numerical and Computer Methods in Structural Mechanics (Edited by
S.J. Fenves, N. Peronne, A.R. Robinson and W.C. Schnobrich). Proc. ONR
- Conf. 1971, pp 79-121. Academic Press, New York, 1973.

APPENDIX A

COURSEWORK PASSED

To comply with the requirements for the degree of Master of Science in Engineering, approved coursework of no less than twenty credits has to be passed. The courses attended are briefly described below.

A.1 AM 363 - NUMERICAL ANALYSIS

Introduction to the theory and practice of numerical approximation. Interpolation and approximation. Numerical integration. Numerical solution of non-linear equations and of initial value problems. Accuracy of numerical solutions.

(3 credits)

A.2 CE 5B6 - FRAME ANALYSIS

The application of the force method of analysis to framed structures of straight and curved members. The stability of equilibrium of framed structures.

(2 credits)

A.3 CE 5B16 - COMPUTER AIDED DESIGN OF STRUCTURES

The modelling of structures using finite element analysis methods. The application of computer aided design (CAD) techniques to finite element data preparation. Computer programming of the finite element method.

APPENDIX B

CHOLESKY HYPERMATRIX EQUATION SOLUTION ROUTINES

In this appendix the FORTRAN routines in the Cholesky Hypermatrix Equation Solution Scheme suite are described in detail. The routines access a two-level hypermatrix scheme which call routines in a virtual memory management scheme. The hypermatrix routines are described in appendix E. The FORTRAN source code for the routines is contained in appendix C.

B.1 Cholesky Hypermatrix Equation Solution Scheme Interface Routines

B.1.1 HCSINI - Initializes the Equation Solution Procedure

Purpose:

HCSINI initializes the equation solution procedure by performing three tasks. Firstly the size of the sub-matrices to be used for the equation solution is calculated, secondly the required hypermatrices are created and thirdly the hypermatrix pointers are stored in the hypermatrix pointer table.

Usage: CALL HCSINI (NFREE, NPRES, NSUPP, IFSYM, NCASE, LUNERR)

Description of Parameters:

Input:

NFREE	Number of free global degrees of freedom
NPRES	Number of prescribed global degrees of freedom
NSUPP	Number of suppressed global degrees of freedom
IFSYM	Flag (0 or 1) to indicate non-symmetric or symmetric problem
NCASE	Number of load cases
LUNERR	Logical unit number for printing any equation solution error message

Output:

None

Remarks:

The input parameters are store in a common block /HCSCOM/ which is common only to the module interface of routines. The hypermatrix pointers are also stored in this common block. A separate common block is initialized which holds the counters for the global degrees of freedom in the hypertable.

Calls: MATSIZ, HYPINI, INDEXS

Called by: User application program

Commons: /HCSCOM/
/DOFCOM/

B.1.2 HCSDOF - Specifies a Tuple in the Global Degree of Freedom Hypertable**Purpose:**

The two values of a tuple in the global degree of freedom hypertable are calculated and stored in the hypertable.

Usage: CALL HCSDOF (ITOTV, IFFPS, FIXED)

Description of parameters**Input:**

ITOTV	Global degree of freedom number
IFFPS	Flag (1, 2, 3) to indicate free, prescribed or suppressed degree of freedom
FIXED	Value of prescribed displacement

Output:

None

Remarks:

Checks are made on the validity of the type of degree of freedom indicator. The value of the prescribed displacement is stored in the prescribed displacement hypermatrix.

Calls: HMGINF, HMGEVS, HMGPOT

Called by: User application program

Commons: /HCSCOM/
/DOFCOM/

B.1.3 HCSCHK - Check Global Degree of Freedom Hypertable**Purpose:**

Routine performs a check to see that all the degrees of freedom in the hypertable have been specified. If a degree of freedom has not been specified then it by default set to be free.

Usage: CALL HCSCHK

Description of parameters:

None:

Remarks:

The use of this routine is optional. This routine may be used for the case when the user wishes to only define the prescribed and suppressed degrees of freedom and then may call this routine to set all the remaining degrees of freedom to be free by default.

Calls: HMGINF, HMGPOT, INFINT

Called by: User application program

Common: /HCSCOM/
/DOFCOM/

B.1.4 HCSABK - Assemble Element Stiffness Matrix into Global Stiffness Hypermatrices

Purpose:

Assembles an element stiffness matrix into the correct global stiffness hypermatrices.

Usage: CALL HCSABK (NEVAB, ESTIF, ELDOF)

Description of parameters:

Input:

NEVAB	Number of degrees of freedom in element
ESTIF	Element stiffness matrix
ELDOF	Global degree of freedom numbers for element degrees of freedom

Output:

None

Remarks:

If the problem is symmetric then the element stiffness matrix need only be upper triangular, however it will not matter if the element matrix is full.

Calls: HMGINF, HMGGET, HMGEVG, HMGEVS

Called by: User application program

Common: /HCSOOM/

B.1.5 HCSABR - Assemble Element Load Vector into Global Load Hypermatrices

Purpose:

Assembles an element load vector into the correct global load hypermatrix.

Usage: CALL HCSABR (NEVAB, ELOAD, ELDOF, ILOAD)

Description of parameters:

Input:

NEVAB	Number of degrees of freedom in element
ELOAD	Element load vector
ELDOF	Global degree of freedom numbers for element degrees of freedom
ILOAD	Load case number

Output:

None

Remarks:

If the global degree of freedom is free then the value is added to the global load vector, however if the degree of freedom is prescribed or suppressed then the value is subtracted from the global load vector in preparation for the calculation of reactions with eqs. (6.21) or (6.22).

Calls: HMGINF, HMGGET, HMGEVS

Called by: User application program

Common: /HCSCOM/

B.1.6 HCSSL - Solution of System of Equations

Purpose:

Controls the three steps of the solution of the system of equations. Firstly the load vector is adjusted for the prescribed degrees of freedom, secondly the Cholesky procedure is used to solve for the inverse multiplication in eq. (6.20) and the calculation of reactions is done from eqs. (6.21) and (6.22).

Usage: CALL HCSSL

Description of parameters:

None

Remarks:

The three sections of this procedure are each performed by separate routines.

Calls: PRESCS, CHLSKY, CREAT

Called by: User application program

Common: /HCSCOM/

B.1.7 HCSUGT - Retrieve Displacement

Purpose:

Retrieves a single displacement for given global degree of freedom and load case number.

Usage: DOUBLE PRECISION HCSUGT, AVALUE

:
AVALUE = HCSUGT (ITOTV, ILOAD)

Description of parameters

Input:

ITOTV	Global degree of freedom number
ILOAD	Load case number

Output:

None

Remarks:

This is a double precision function. If an attempt is made to retrieve a value from a hypermatrix that does not exist then a value of 0.0d0 is returned. This would be the case of trying to retrieve a value from the hypermatrix of suppressed degrees of freedom which is by default not created.

Calls: HTGGET, HMGINF, HMGEVG

Called by: User application program

Common: /HCSCOM/

B.1.8 HCSRGT - Retrieve Reaction

Purpose:

Retrieves a single resultant reaction for a given global degree of freedom and load case number.

Usage: DOUBLE PRECISION HCSRGT, AVALUE
:
AVALUE = HCSRGT (ITOTV, ILOAD)

Description of Parameters

Input:

ITOTV Global degree of freedom number
ILOAD Load case number

Output:

None

Remarks:

This is a double precision function. If an attempt is made to retrieve a value from a hypermatrix that does not exist then a value of 0.0d0 is returned. This would be the case of trying to retrieve a value from the hypermatrix of free degrees of freedom, in which case there is no reaction.

Calls: HTGGET, HMGINF, HMGEVG

Called by: User application program

Common: /HCSCOM/

B.2 Cholesky Hypermatrix Equation Solution Scheme Routines

B.2.1 CHLSKY - Controlling routine for Cholesky Procedure

Purpose:

CHLSKY controls, at partitioning level (level 2), the flow of the Cholesky algorithm through the decomposition, forward-substitution and back substitution stages.

Usage: INTEGER GSTNDX, GLDNDX
:
CALL CHLSKY(GSTNDX, GLDNDX)

Description of parameters:

Input:-

GSTNDX Index to K_{ff} hypermatrix (stiffness matrix)

GLDNDX Index to F_f hypermatrix (load matrix)

Output:-

GLDNDX Index to U_f hypermatrix (result matrix)

Remarks:-

The stiffness matrix is assumed to be symmetric positive-definite and prescribed and suppressed degrees of freedom have been condensed out. The result matrix will overwrite the load matrix.

Calls: HMGINF, SUMTRI, CHKTRI, SUMRCT, SUMYYY, CHKYYY, SUMXXX, CHKXXX.

Called by: HCSSOL

Commons: none

B.2.2 CHKTRI - Cholesky Decomposition for Diagonal Submatrix

Purpose:

Performs Cholesky decomposition on a submatrix on the diagonal of the partitioned hypermatrix.

Usage: INTEGER GSTNDX
 :
 CALL CHKTRI(GSTNDX, IROWH, SUMMA, PIVOT)

Description of Parameters:

Input:-

GSTNDX Index to K_{ff} hypermatrix (global stiffness matrix)
IROWH Partition row number (also column number because pivotal partition)
SUMMA Summation for this partitioned sub-matrix

Output:-

GSTNDX Index to hypermatrix U
PIVOT Pivot vector for this row partition

Remarks:-

Summation is updated as the solution proceeds through the rows and columns

Calls: HMGSI2, SMTGET, SMTPUT

Called by: CHLSKY

Commons: /WORK/

B.2.3 CHKRCT - Cholesky Decomposition for Non-Diagonal Submatrix

Purpose:

Performs Cholesky decomposition on submatrices which do not lie on the diagonal of the partitioned hypermatrix.

Usage: INTEGER GSTNDX
 :
 CALL CHKRCT (GSTNDX, IROWH, ICOLH, SUMMA, PIVOT)

Description of Parameters:

Input:
 GSTNDX Index to K_{ff} hypermatrix (stiffness matrix)
 IROWH Partition row number
 ICOLH Partition column number
 SUMMA Summation up until this partitioned sub-matrix
 PIVOT Pivot vector for this row partition

Output:

Summation is updated as the solution proceeds through the rows and columns of the submatrix.

Calls: HMGSIZ, SMTGET, SMTPUT

Called by: CHLSKY

Commons: /WORK/

B.2.4 CHKYYY - Cholesky Forward-Substitution for a Submatrix

Purpose:

Performs Cholesky "forward-substitution" for a submatrix.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL CHKYYY (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 GLDNDX Index to load matrix
 IROWH Partition row number in load hypermatrix
 ICOLI Partition column number in load hypermatrix
 SUMMA Summation for current partition submatrix (IROWH, ICOLI)

Output:

GLDNDX Index to hypermatrix Y

Remarks:

Summation is updated as the solution proceeds through the rows and columns.

Calls: HMGSIZ, SMTGET, SMTPUT

Called by: CHLSKY

Commons: /WORK/

B.2.5 CHKXXX - Cholesky Back-Substitution of a Submatrix

Purpose:

Performs Cholesky "back-substitution" on a submatrix.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL CHKXXX (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:

GSTNDX Index to hypermatrix U
 GLDNDX Index to hypermatrix Y
 IROWH Partition row number in Y hypermatrix
 ICOI Partition column number in Y hypermatrix
 SUMMA Summation for current submatrix (IROWH, ICOI)

Output:

GLDNDX Index to result hypermatrix X

Remarks:

Summation is updated as the solution proceeds through the rows and columns.

Calls: HMGSIZ, SMIGET, SMIPUT

Called by: CHLSKY

Commons: /WORK/

B.2.6 SUMTRI - Calculates Summation for the Decomposition of a Diagonal Submatrix

Purpose:

Calculates the summation for the decomposition of a diagonal submatrix.

Usage:

INTEGER GSTNDX
 :
 CALL SUMTRI (GSTNDX, IROWH, SUMMA)

Description of Parameters:

Input:

GSTNDX Index to hypermatrix U
 IROWH Partition row number (also column number because pivotal partition) in hypermatrix U

Output:

SUMMA Summation up to partition (IROWH, IROWH)

Remarks:

Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMIGET

Called by: CHLSKY

Commons: /WORK/

B.2.7 SUMRCT - Calculates Summation for the Decomposition of a Non-Diagonal Submatrix

Purpose:

Calculates the summation for the decomposition of a non-diagonal hypermatrix.

Usage: INTEGER GSTNDX
 :
 CALL SUMRCT (GSTNDX, IROWH, ICOLH, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 IROWH Partition row number in hypermatrix U
 ICOLH Partition column number in hypermatrix U

Output:
 SUMMA Summation up to partition (IROWH, ICOLH)

Remarks:
 Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.8 SUMYYY - Calculates Summation for the Forward Substitution of a Submatrix

Purpose:
 Calculates summation for a submatrix in hypermatrix Y.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL SUMYYY (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 GLDNDX Index to hypermatrix Y
 IROWH Partition row number in hypermatrix Y
 ICOLI Partition column number in hypermatrix Y

Output:
 SUMMA Summation up to partition (IROWH, ICOLI)

Remarks:
 Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.9 SUMXXX - Calculates Summation for the Back Substitution of a Submatrix

Purpose:
 Calculates summation for a submatrix in hypermatrix X.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL SUMXXX (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 GLDNDX Index to hypermatrix Y
 IROWH Partition row number in hypermatrix Y
 ICOLI Partition column number in hypermatrix Y

Output:
 SUMMA Summation up to partition (IROWH, ICOLI)

Remarks:
 Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.10 HMTMLT - Hypermatrix Multiplication Routine

Purpose:
 Multiply two hypermatrices together and either adds or subtracts the result to the resultant hypermatrix.

Usage: INTEGER TRANA, TRANB
 :
 CALL HMTMLT (NDXHMA, NDXHMB, NDXHMC, FACTR, TRANA, TRANB,
 ISYMA, ISYMB)

Description of Parameters:

Input:
 NDXHMA Index to hypermatrix A
 NDXHMB Index to hypermatrix B
 NDXHMC Index to hypermatrix C
 FACTR A factor (+1.0 or -1.0) to indicate addition or subtraction
 TRANA Flag (0 or 1) to indicate do not or do transpose A before
 multiplication
 TRANB Flag (0 or 1) to indicate do not or do transpose B before
 multiplication
 ISYMA Flag (0 or 1) to indicate if A is unsymmetric or symmetric
 ISYMB Flag (0 or 1) to indicate if B is unsymmetric or symmetric

Output:
 NDXHMC Index to resultant hypermatrix C

Remarks:
 Only the index to the hypermatrices and whether they are to be transpose and if they are symmetric need be passed to the routine, the remaining hypermatrix information may be retrieved by the routine itself. Checks are done on the validity of the multiplication by checking the compatibility of the hypermatrices.

Calls: HMGINF, HMGSIZ, SMTGET, SMIMLT, SMIPUT

Called by: Application program

Commons: None

B.2.11 SMIMLT - Submatrix Multiplication Routine

Purpose

Multiplies two submatrices together and either adds or subtracts the result from the resultant submatrix.

Usage: INTEGER TRANSA, TRANS

```

      :
      CALL SMIMLT (NDXHMA, IROWA, ICOLA,
                   NDXHMB, IROWB, ICOLB,
                   ARAYC, NROWSC, NCOLSC,
                   FACTR, TRANSA, TRANSB,
                   ISYMA, ISYMB)

```

Description of Parameters:

Input:

NDXHMA	Index to hypermatrix A
IROWA	Partition row number in hypermatrix A
ICOLA	Partition column number in hypermatrix A
NDXHMB	Index to Hypermatrix B
IROWB	Partition row number in hypermatrix B
ICOLB	Partition column number in hypermatrix B
ARAYC	The result submatrix C
NROWSC	Number of rows in submatrix C
NCOLSC	Number of columns in submatrix C
FACTR	A factor (+1.0 or -1.0) to indicate addition or subtraction
TRANSA	Flag (0 or 1) to indicate do not or do transpose A before multiplication
TRANSB	Flag (0 or 1) to indicate do not or do transpose B before multiplication

Remarks

This routine has subsequently been split into several separate routines so that the checks for symmetry and transposition may be checked at a higher level and then the relevant routine called which then does not need to perform any checks before multiplication. These routines could be written in assembler language to improve efficiency.

Calls: SMTGET, HMGSIZ

Called by: SMIMLT

Common: /WORK/

B.2.12 SMIPUT - Store Submatrix in Database

Purpose:

SMIPUT stores a submatrix in the database

Usage: CALL SMIPUT (NDXHMG, IROW, ICOL, ARRAY)

Description of Parameters:

Input:

NDXHMG Index to hypermatrix
 IROW Partition row number of submatrix
 ICOL Partition column number of submatrix
 ARRAY Submatrix to be stored

Output:

None

Remarks:

Before the submatrix may be stored a check is done to see if the actual number of rows in the submatrix corresponds to the number of rows dimensioned for the submatrix. If the row dimensions do not correspond a copy is done with the matrix to ensure values are correctly stored. The column dimensions do not need to be checked because matrices are stored column-wise in FORTRAN.

Calls: HMGsiz, HMGPUT

Called by: General routines

Commons: None

B.2.13 SMTGET - Retrieve Submatrix from Database

Purpose:

SMTGET retrieves a submatrix from the database

Usage: CALL SMTGET (NDXHMG, IROW, ICOL, ARRAY)

Description of Parameters:

Input:

NDXHMG Index to hypermatrix
 IROW Partition row number of submatrix
 ICOL Partition column number of submatrix

Output:

ARRAY Submatrix retrieved from database

Remarks:

After the submatrix has been retrieved a check is done to see if the actual number of rows in the submatrix corresponds to the number of rows dimensioned for the submatrix. If the row dimensions do not correspond a copy is done with the submatrix to ensure values are correctly retrieved. The column dimensions do not need to be checked because matrices are stored column-wise in FORTRAN.

Calls: HMGsiz, HMGGET

Called by: General routines

Commons: None

B.2.14 CREAT - Calculate Reactions

Purpose:

CREACT controls the calculation of the reactions at prescribed and suppressed degrees of freedom.

Usage: CALL CREAT (NDXRP, NDXKPF, NDXUF, NDXKPP, NDXUP,
NDXRS, NDXKSF, NDXKSP, IFSYM)

Description of Parameters:

Input:

NDXRP Index to R_p hypermatrix
 NDXKPF Index to K_{pf} hypermatrix
 NDXUF Index to KU_f hypermatrix
 NDXKPP Index to K_{pp} hypermatrix
 NDXUP Index to U_p hypermatrix
 NDXRS Index to R_s hypermatrix
 NDXKSF Index to K_{sf} hypermatrix
 IFSYM Flag (0 or 1) to indicate non-symmetric or symmetric problem.

Output:

None

Remarks:

Reactions are calculated by performing hypermatrix multiplication. The reactions are solved from

$$R_s = -F_s + K_{sf} \cdot U_f + K_{sp} \cdot U_p$$

and

$$R_s = -F_s + K_{sf} \cdot U_f + K_{sp} \cdot U_p$$

Calls: HMTMLT

Called by: HCSSL

Common: None

B.2.15 PGMINI - Initialize Equation Solution Modules

Purpose:

PGMINI calls all other initialization routines to initialize the internal variables of the modules.

Usage: CALL PGMINI

Description of Parameters

None

Remarks:

There may only be one PGMINI routine in the program. If the program uses any DAS routines within other parts of the program then this routine should be called by the user at the start of the program.

Calls: LUNINI, BASINI, HTGINI, HMGINI

Called by: HCSINI (or user program)

Commons: None

B.2.16 MATSIZ - Calculate Size of Submatrices for this Solution**Purpose:**

MATSIZ calculates the size of the submatrices that will be used for this solution.

Usage: CALL MATSIZ (NFREE, NPRES, NSUPP, NSIZS)

Description of Parameters:**Input:**

NFREE	Number of free degrees of freedom
NPRES	Number of prescribed degrees of freedom
NSUPP	Number of suppressed degrees of freedom

Output:

NSIZS	Size of submatrices
-------	---------------------

Remarks:

The submatrix size is calculated from the number of free degrees of freedom, the page size of the DYNA-file and a maximum optimal size.

Calls: HMGINF, DAINFI

Called by: HCSINI

Commons: None

B.2.17 DAFCRT - Create and Format DYNA-File**Purpose**

Create and format a DYNA-file for use as the database file

Usage: CALL DAFCRT

Description of Parameters:

None

Remarks:

If the DYNA-file is used by other parts of the program then DAFCRT should be called by the user program.

Calls: DAS package

Called by: HCSINI (or user program)

Common: None

B.2.18 HYPINI - Initialize Hypermatrices and Hypertable

Purpose:

HYPINI initializes all the hypermatrices for the global stiffness matrix, global load vector and result vectore. HYPINI also initializes the global degree of freedom hypertable.

Usage: CALL HYPINI (NDXKFF, NDXKFP, NDXKFS, NDXUF, NDXRF, NDXKPF, NDXDPP, NDXKPS, NDXUP, NDXRP, NDXKSF, NDXKSP, NDXKSS, NDXUS, NDXRS, NDXFIX, NFREE, NPRES, NSUPP, NLOAD, NSIZS, IFSYM)

Description of Parameters

Input:

NFREE	Number of free degrees of freedom
NPRES	Number of prescribed degrees of freedom
NSUPP	Number of suppressed degrees of freedom
NLOAD	Number of load cases
NSIZS	Dimension of submatrices
IFSYM	Flag (0 or 1) to indicate non-symmetry or symmetry

Output:

NDXKFF	Index to K_{ff} hypermatrix
NDXKFP	Index to K_{fp} hypermatrix
NDXKFS	Index to K_{fs} hypermatrix
NDXUF	Index to U_f hypermatrix
NDXRF	Index to R_f hypermatrix
NDXKPF	Index to K_{pf} hypermatrix
NDXDPP	Index to K_{pp} hypermatrix
NDXKPS	Index to K_{ps} hypermatrix
NDXUP	Index to U_p hypermatrix
NDXRP	Index to R_p hypermatrix
NDXKSF	Index to K_{sf} hypermatrix
NDXKSP	Index to K_{sp} hypermatrix
NDXKSS	Index to K_{ss} hypermatrix
NDXUS	Index to U_s hypermatrix
NDXRS,	Index to K_s hypermatrix
NDXFIX	Index to fixity hypertable

Remarks:

Not all the hypermatrices are created because some of them may be zero or will not be used.

Calls: HMGCRT, HMGINF, HTGCRT

Called by: HCSINI

COMMON: None

B.2.19 INDEXS - Initialize Pointer Look-up Tables

Purpose:

Initializes the pointer look-up tables for the global stiffness matrix, global load vector and displacement vector.

Usage: CALL INDEXS (INDEXK, INDEXU, INDEXR, IFSYM,
 NDXKFF, NDXKFP, NDXKFS, NDXUF, NDXRF,
 NDXKPF, NDXDPP, NDXKPS, NDXUP, NDXRP,
 NDXKSF, NDXKSP, NDXKSS, NDXUS, NDXRS)

Description of parameters

INDEXK	Stiffness pointer table
INDEXU	Displacements pointer table
INDEXR	Loads pointer table
IFSYM	Flag (0 or 1) for non-symmetry of symmetry
NDXKFF	Index to K_{ff} hypermatrix
NDXKFP	Index to K_{fp} hypermatrix
NDXKFS	Index to K_{fs} hypermatrix
NDXUF	Index to U_f hypermatrix
NDXRF	Index to R_f hypermatrix
NDXKPF	Index to K_{pf} hypermatrix
NDXDPP	Index to K_{pp} hypermatrix
NDXKPS	Index to K_{ps} hypermatrix
NDXUP	Index to U_p hypermatrix
NDXRP	Index to R_p hypermatrix
NDXKSF	Index to K_{sf} hypermatrix
NDXKSP	Index to K_{sp} hypermatrix
NDXKSS	Index to K_{ss} hypermatrix
NDXUS	Index to U_s hypermatrix
NDXRS	Index to K_s hypermatrix

Remarks:

The tables are initialized using the hypermatrix pointers.

Calls: None

Called by: HCSINI

COMMON: None

B.2.20 PRESCTS - Adjust Load Vector for Prescribed Displacements**Purpose:**

PRESCTS adjusts the load vector for the prescribed degrees of freedom in the problem.

Usage: CALL PRESCTS (NDXRF, NDXKFP, NDXUP)

Description of Parameters:**Input:**

NDXRF Index to R_f hypermatrix
NDXKFP Index to K_{fp} hypermatrix
NDXUP Index to U_p hypermatrix

Output:

None

Remarks:

The load hypermatrix is adjusted by performing a hypermatrix multiplication.

Calls: HMTMLT

Called by: HCSSL

Common: None

APPENDIX C

CHOLSKY HYPERMATRIX EQUATION SOLUTION ROUTINES SOURCE LISTING

This appendix contains the FORTRAN source listing of the routines in the Cholesky Hypermatrix Equation Solution Scheme package. A description of the routines is contained in appendix B.

```

c
c.....
c   File = [USERS.ASICS.HCSMOD]HCSMOD.FOR
c.....
c
c   Module.....HCSMOD
c   Purpose.....Linear Equation Solver
c   Developer.....M.W.Eastman.   U.C.T.
c   Date.....July 1987
c   References.....DAS Programmer's Manual
c   Language.....FORTRAN 77
c   Installation.....MicroVax II.  U.C.T.
c
c.....
c
c   Description of the module:
c   -----
c
c   This module solves a system of symmetric positive-definite
c   linear equations derived from the finite element method. The
c   solution method involves Cholesky decomposition with a
c   hypermatrix data management scheme
c
c   Description of routines
c
c   HCSINI   Initializes the equation solution procedure
c   HCSDOF   Specifies a global degree of freedom in hypertable
c   HCSCHR   Checks that the global degree of freedom table is
c           completely defined
c   HCSABK   Assembles element stiffness matrix into global stiffness
c           hypermatrices
c   HCSABR   Assembles element load vector in global load vector
c   HCSSOL   Controls solution of equations and calculation of
c           reactions
c   HCSUGT   Retrieves calculated displacement
c   HCSRGT   Retrieves calculated reaction
c   HCSDMP   Dumps specified hypermatrix
c   PGMINI   Initializes program module
c   MATSIZ   Calculates size of the submatrices for the solution
c   DAFORT   Creates and format DYNA-file
c   HYPINI   Initializes hypermatrices and hypertable
c   INDEXS   Initializes indirect pointer look-up tables
c   PRESCTS   Adjusts load vector for prescribed displacement
c   CHLSKY   Driver routine for Cholesky equation solution procedure
c   CHKTRI   Calculates L matrix for diagonal submatrices
c   CHKRCT   Calculates L matrix for non-diagonal sub-matrices
c   CHKYYY   Calculates Y matrix, forward substitution
c   CHKXXX   Calculates X matrix, backward substitution
c   SUMTRI   Calculates summation for diagonal submatrices

```

```

c      SUMRCT  Calculates summation for non-diagonal submatrices
c      SUMYYY  Calculates summation for Y submatrix
c      SUMXXX  Calculates summation for X submatrix
c      SMTPUT  Stores submatrix in DYNA-file
c      SMTGET  Retrieves submatrix from DYNA-file
c      CREAT  Calculates reactions
c      DPIHMT  Dumps integer hypermatrix
c      DPRHMT  Dumps real hypermatrix
c      HMTMLT  Multiplies hypermatrices
c      SMTMLT  Multiplies submatrices
c.....
c
c
c      subroutine HCSINI ( ndfre, ndpre, ndsup, isymm, ncase, luerr )
c
c.....
c      purpose:
c      initialises hypermatrix cholesky solver parameters
c
c      arguments:
c i      ndfre  - number of free dof
c i      ndpre  - number of prescribed dof
c i      ndsup  - number of suppressed dof
c i      isymm  - symmetry indicator 0 - unsymmetric 1 - symmetric
c i      ncase  - no. of load cases
c i      luerr  - logical unit for error printing
c.....
c
c      include 'hcscom.inc'
c      include 'dofcom.inc'
c
c--- set values in 'hcscom' common
c      nfree = ndfre
c      npres = ndpre
c      nsupp = ndsup
c      ifsym = isymm
c      nload = ncase
c      lunerr = luerr
c
c--- calculate submatrix size
c      call MATSIZ ( nfree, npres, nsupp, nsizs )
c
c--- initialise dof counters
c      ifree = 0
c      ipres = 0
c      isupp = 0
c
c--- create hypermatrices
c      call HYPINI ( ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
c      &             ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
c      &             ndxksf, ndxksp, ndxkss, ndxus, ndxrs,
c      &             ndxfix, nfree, npres, nsupp, nload,
c      &             nsizs, ifsym )
c

```

```

c--- set pointer tables for stiffness, displacements and loads
      call INDEXS ( indexk, indexu, indexr, ifsym,
&                ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
&                ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
&                ndxksf, ndxksp, ndxkss, ndxus, ndxrs )

c
      return
      end

c
c
      subroutine HCSDOF ( itotv, iffps, fixed )
c
c.....
c      purpose :
c      specifies degrees of freedom
c
c      arguments:
c i   itotv - global dof number
c i   iffps - flag to indicate free(1)/prescribed(2)/suppressed(3)
c i   fixed - prescribed value
c.....
c
      implicit double precision(a-h,o-z)
c
      include 'hcscom.inc'
      include 'dofcom.inc'
c
      integer HMGINF
      integer iffix(2)
c
c--- retrieve lun no.
      lunerr = LUNGET(7)
c
      nupwpe = HMGINF(ndxup,3)
c
      iffix(1) = iffps
c
      if(iffps.eq.1) then
          ifree = ifree + 1
          iffix(2) = ifree
      else
          if(iffps.eq.2) then
              ipres = ipres + 1
              iffix(2) = ipres
c--- set prescribed value in hypermatrix
              do 10 iload = 1,nload
10         call HMGEVS(ndxup,ipres,iload,fixed,nupwpe)
          else
              if(iffps.eq.3) then
                  isupp = isupp + 1
                  iffix(2) = isupp
              else
                  write(lunerr,*) 'Incorrect degree of freedom,iffps',iffps
                  iffps = iffps/INFINT(0)
              end if
          end if
      end if
  
```

```

C
C--- store tuple
      call HTGPUT(ndxfix,itotv,ifix,2)
C
      return
      end
C
C
      subroutine HCSCHK
C
C.....
C      purpose:
C      checks that the dof hypermatrix has been correctly set up
C
C      arguments
C      none
C.....
C
      include 'hcscom.inc'
      include 'dofcom.inc'
C
      integer HMGINF
      integer ifix(2),ifps(2)
C
C--- retrieve lun no.
      lunerr = LUNGET(7)
C
      ntotv = nfree + npres + nsupp
C
      do 100 itotv = 1,ntotv
      call HTGGET(ndxfix,itotv,ifps,2)
      if(ifps(1).eq.0) then
         ifree = ifree + 1
         ifix(1) = 1
         ifix(2) = ifree
         call HTGPUT(ndxfix,itotv,ifix,2)
      else
         if(ifps(1).gt.3) then
            write(lunerr,*) 'Error in dof specification:ifps ',ifps(1)
            ifps(1) = ifps(1)/INFINT(0)
         end if
      end if
100  continue
C
      if(ifree.ne.nfree) then
         write(lunerr,*) 'Error in free dof specification,ifree,nfree',
&                        ifree,nfree
         ifree = ifree/INFINT(0)
      end if
C
      if(ipres.ne.npres) then
         write(lunerr,*) 'Error in pres dof specification,ipres,npres',
&                        ipres,npres
         ipres = ipres/INFINT(0)
      end if

```

```

c      if(isupp.ne.nsupp) then
c          write(lunerr,*) 'Error in supp dof specification,isupp,nsupp',
&              isupp,nsupp
c          isupp = isupp/INFINT(0)
c      end if

c      return
c      end

c
c
c      subroutine HCSABK ( nevab,  estif,  eldof )
c
c.....
c      purpose :
c      assembles element stiffness matrix into global stiffness matrix
c
c      arguments:
c i      nevab  - no. of variables in element
c i      estif  - element stiffness matrix
c i      eldof  - element degrees of freedom codes
c.....
c
c      implicit double precision(a-h,o-z)
c
c      include 'hcscom.inc'
c
c      real*8  estif(nevab,nevab)
c      integer HMGINF
c      integer eldof(nevab)
c      integer ifrow(2),ifcol(2),iftmp(2),hmgwpe
c
c---   check for symmetry
c      if(ifsym.eq.0) then
c---   unsymmetric
c          do 100 ievab = 1,nevab
c              do 110 jevab = 1,nevab
c
c                  nrowg = eldof(ievab)
c                  ncolg = eldof(jevab)
c
c---   retrieve fixity code
c          call HTGGET(ndxfix,nrowg,ifrow,2)
c          call HTGGET(ndxfix,ncolg,ifcol,2)
c
c---   check fixity code and place in correct matrix
c          ndxhmg = indexk(ifrow(1),ifcol(1))
c          if(ndxhmg.gt.0) then
c              hmgwpe = HMGINF(ndxhmg,3)
c              call HMGEVG(ndxhmg,ifrow(2),ifcol(2),gstif,hmgwpe)
c              gstif = gstif + estif(ievab,jevab)
c              call HMGEVS(ndxhmg,ifrow(2),ifcol(2),gstif,hmgwpe)
c          end if
110      continue
100      continue
      else

```

```

c
c--- symmetric
do 200 ievab = 1,nevab
do 210 jevab = ievab,nevab
c
nrowg = eldof(ievab)
ncolg = eldof(jevab)
c
c--- retrieve fixity code
call HTGGET(ndxfix,nrowg,ifrow,2)
call HTGGET(ndxfix,ncolg,ifcol,2)
c
if(ifrow(1).gt.ifcol(1)) then
    iftmp(1) = ifrow(1)
    iftmp(2) = ifrow(2)
    ifrow(1) = ifcol(1)
    ifrow(2) = ifcol(2)
    ifcol(1) = iftmp(1)
    ifcol(2) = iftmp(2)
else
    if(ifrow(1).eq.ifcol(1)) then
        if(ifrow(2).gt.ifcol(2)) then
            iftmp(2) = ifrow(2)
            ifrow(2) = ifcol(2)
            ifcol(2) = iftmp(2)
        end if
    end if
end if
c
c--- check fixity code and place in correct matrix
ndxhmg = indexk(ifrow(1),ifcol(1))
if(ndxhmg.gt.0) then
    hmgwpe = HMGINF(ndxhmg,3)
    call HMGEVG(ndxhmg,ifrow(2),ifcol(2),gstif,hmgwpe)
    gstif = gstif + estif(ievab,jevab)
    call HMGEVS(ndxhmg,ifrow(2),ifcol(2),gstif,hmgwpe)
end if
210 continue
200 continue
end if
c
return
end
c
c
c
subroutine HCSABR ( nevab, eload, eldof, iload )
c
c.....
c    purpose :
c    assembles element stiffness matrix into global stiffness matrix
c
c    arguments:
c    nevab - no. of variables in element
c    estif - element stiffness matrix
c    eldof - element degrees of freedom codes
c    iload - load case number
c.....

```

```

C      implicit double precision(a-h,o-z)
C
C      include 'hcscom.inc'
C
C      real*8  eload(nevab)
C      integer HMGINF
C      integer eldof(nevab)
C      integer ifrow(2),hmgwpe
C
C      do 100 ievab = 1,nevab
C
C      nrowg = eldof(ievab)
C
C      c--- retrieve fixity code
C      call HIGGET(ndxfix,nrowg,ifrow,2)
C
C      c--- check fixity code and place in correct matrix
C      ndxhmg = indexr(ifrow(1))
C      if(ndxhmg.gt.0) then
C          hmgwpe = HMGINF(ndxhmg,3)
C          call HMGEVG(ndxhmg,ifrow(2),iload,gload,hmgwpe)
C          if(ifrow(1).eq.1) then
C              gload = gload + eload(ievab)
C          else
C              gload = gload - eload(ievab)
C          end if
C          call HMGEVS(ndxhmg,ifrow(2),iload,gload,hmgwpe)
C      end if
110      continue
100      continue
C
C      return
C      end
C
C      subroutine HCSSOL
C
C      .....
C      purpose:
C      solves equations using Cholesky decomposition and
C      hypermatrix multiplication
C
C      arguments
C      none
C      .....
C
C      include 'hcscom.inc'
C
C      c--- alter gload for prescribed displacements ( Rf = Rf - Kfp.Up )
C      call PRESCS ( ndxuf, ndxkfp, ndxup )
C
C      c--- call Cholesky subroutine
C      call CHLSKY ( ndxkff, ndxuf )
C
C      c--- calculate reactions
C      call CREAT ( ndxrp, ndxkpf, ndxuf, ndxkpp, ndxup,
C      &            ndxrs, ndxksf, ndxksp, ifsym )
C
C      return
C      end

```

```

c
c
c      double precision function HCSUGT ( itotv, iload )
c
c.....
c      Purpose
c      retrieves global displacement
c
c      arguments:
c i   itotv - global dof number
c i   iload - load case number
c.....
c
c      implicit double precision(a-h,o-z)
c
c      include 'hcscom.inc'
c
c      integer HMGINF
c      integer iffix(2)
c      integer hmgwpe
c
c      call HTGGET(ndxfix,itotv,iffix,2)
c
c---  check if hypermatrix exists
c      ndxhmg = indexu(iffix(1))
c      if(ndxhmg.gt.0) then
c          hmgwpe = HMGINF(ndxhmg,3)
c          call HMGVVG(ndxhmg,iffix(2),iload,value,hmgwpe)
c          HCSUGT = value
c      else
c          HCSUGT = 0.0d0
c      end if
c
c      return
c      end
c
c
c      double precision function HCSRGT ( itotv, iload )
c
c.....
c      purpose
c      retrieves global reaction
c
c      arguments:
c i   itotv - global dof number
c i   iload - load case number
c.....
c
c      implicit double precision(a-h,o-z)
c      include 'hcscom.inc'
c
c      integer HMGINF
c      integer iffix(2)
c      integer hmgwpe
c
c      call HTGGET(ndxfix,itotv,iffix,2)

```

```

c
c--- check if hypermatrix exists
      ndxhmg = indexr(iffix(1))
      if(ndxhmg.gt.0.and.iffix(1).ne.1) then
         hmgwpe = HMGINF(ndxhmg,3)
         call HMGEVG(ndxhmg,iffix(2),iload,value,hmgwpe)
         HCSRGT = value
      else
         HCSRGT = 0.0d0
      end if

c
      return
      end

c
c
      subroutine HCSDMP ( string )
c
c .....
c      purpose:
c      dumps hypermatrices
c
c      arguments:
c      string - hypermatrix identifier
c .....
c
      character*1 string
c
      include 'hcscom.inc'
c
      if(string.eq.'K') then
         call DPRHMT('Kff',ndxkff)
         call DPRHMT('Kfp',ndxkfp)
         call DPRHMT('Kfs',ndxkfs)
         call DPRHMT('Kpf',ndxkpf)
         call DPRHMT('Kpp',ndxkpp)
         call DPRHMT('Kps',ndxkps)
         call DPRHMT('Ksf',ndxksf)
         call DPRHMT('Ksp',ndxksp)
         call DPRHMT('Kss',ndxkss)
      end if
c
      if(string.eq.'U') then
         call DPRHMT('Uf',ndxuf)
         call DPRHMT('Up',ndxup)
         call DPRHMT('Us',ndxus)
      end if
c
      if(string.eq.'R') then
         call DPRHMT('Rf',ndxrf)
         call DPRHMT('Rp',ndxrp)
         call DPRHMT('Rs',ndxrs)
      end if
c
      return
      end

```

```

subroutine MATSIZ ( nfree, npres, nsupp, nsizs )
C
C.....
C  purpose:
C  calculates size of sub-matrices for solution
C
C  arguments:
C  nfree  - size of Kff hypermatrix
C  npres  - size of Kpp hypermatrix
C  nsupp  - size of Kss hypermatrix
C  nsizs  - size of sub-matrices
C.....
C
C  parameter MSIZS=15
C  parameter NWDPE=2
C  integer HMGINF,DAINF1,INFINT
C
C--- retrieve lun no.
C  lunerr = LUNGET(6)
C
C--- retrieve header sizes for hypermatrix control
C  nwdhm = HMGINF(0,1)
C  nwdsn = HMGINF(0,2)
C
C--- retrieve page size
C  ipagsz = DAINFI(1)
C
C--- calculate size sub-matrices
C  nsizs = sqrt(float(ipagsz - nwdsn)/NWDPE)
C
C--- check if larger than Kff stiffness matrix
C  if(nsizs.gt.nfree) nsizs = nfree
C
C--- check if larger than maximum allowed
C  if(nsizs.gt.MSIZS) nsizs = MSIZS
C
C--- check if Kff control info fits on a single page
C  nsizh = (nfree + nsizs - 1)/nsizs
C  nsm = nsizh*nsizh
C  if(ipagsz.lt.nsm+nwdhm) then
C    write(lunerr,1003) 'Kff',nsm+nwdhm,ipagsz
C    nsm = nsm/INFINT(0)
C  end if
C
C--- check if Kpp control info fits on a single page
C  nsizh = (npres + nsizs - 1)/nsizs
C  nsm = nsizh*nsizh
C  if (ipagsz.lt.(nsm + nwdhm)) then
C    write(lunerr,1003) 'Kpp',nsm + nwdhm,ipagsz
C    nsm = nsm / INFINT(0)
C  end if
C
C--- check if Kss control info fits on a single page
C  nsizh = (nsupp + nsizs - 1)/nsizs
C  nsm = nsizh*nsizh

```

```

if (ipagsz.lt.(nsm + nwdhm)) then
  write(lunerr,1003) 'Kss',nsm + nwdhm,ipagsz
  nsm = nsm / INFINT(0)
end if

```

c

```

return

```

c

```

1003 format(1h ,'=> Unable to create control information for',
& ' hypermatrix ',a3,/,
& 1h , ' Number of words required for control',
& ' information = ',i5,/,
& 1h , ' Number of words available on a page = ',i5)

```

c

```

end

```

c

c

```

subroutine HYPINI ( ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
&                  ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
&                  ndxksf, ndxksp, ndxkss, ndxus, ndxrs,
&                  ndxfix, nfree, npres, nsupp, nload,
&                  nsizs, ifsym )

```

c

c.....

```

c  purpose :
c  initialise all hyper matrices needed for cholesky solution

```

c

```

c  arguments:

```

```

c  ndxkff - index to kff hypermatrix

```

```

c  ndxkfp - index to kfp hypermatrix

```

```

c  ndxkfs - index to kfs hypermatrix

```

```

c  ndxuf - index to uf hypermatrix

```

```

c  ndxrf - index to rf hypermatrix

```

```

c  ndxkpf - index to kpf hypermatrix

```

```

c  ndxkpp - index to kpp hypermatrix

```

```

c  ndxkps - index to kps hypermatrix

```

```

c  ndxup - index to up hypermatrix

```

```

c  ndxrp - index to rp hypermatrix

```

```

c  ndxksf - index to ksf hypermatrix

```

```

c  ndxksp - index to ksp hypermatrix

```

```

c  ndxkss - index to kss hypermatrix

```

```

c  ndxus - index to us hypermatrix

```

```

c  ndxrs - index to rs hypermatrix

```

```

c  ndxfix - index to fix hypermatrix

```

```

c  nfree - number of free dof

```

```

c  npres - number of prescribed dof

```

```

c  nsupp - number of suppressed dof

```

```

c  nload - number of load cases

```

```

c  nsizs - dimension of standard sub-matrix

```

```

c  ifsym - symmetry flag

```

c.....

c

```

c  parameter NWDPE=2

```

```

c  integer HMGINF

```

```

c  integer mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,mtndx

```

c

```

c--- retrieve io lun no.

```

```

c  lundmp = LUNGET(8)

```

```

C
C--- calculate size of reduced stiffness matrix
      ntotv = nfree + npres + nsupp

C
C*** hypermatrix Kff
      mthed = 2004287489
      mttyp = 2
      mtwpe = NWDPE
      nrowm = nfree
      ncolm = nfree
      nrows = nsizs
      ncols = nsizs
C--- check sizes
      if((nrowm.le.0).or.(ncolm.le.0)) then
         ndxkff = 0
         write(lundmp,9010) 'Kff ',nrowm,ncolm
      else
C--- create hypermatrix
         call HMGCR1(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
          &          mtndx)
         ndxkff = mtndx
         nrowa = HMGINF(ndxkff,8)
         ncola = HMGINF(ndxkff,9)
C--- print stiffness hypermatrix control parameters
         write(lundmp,9000) 'Kff ',nrowm,ncolm,nrowa,ncola,nrows,
          &          ncols,mttyp,mtwpe
      end if

C
C*** hypermatrix Kpf
      ndxkpf = 0
      if(ifsym.eq.0) then
         mthed = 2004287498
         mttyp = 2
         mtwpe = NWDPE
         nrowm = npres
         ncolm = nfree
         nrows = min(nsizs,npres)
         ncols = nsizs
C--- check sizes
         if((nrowm.le.0).or.(ncolm.le.0)) then
            ndxkpf = 0
            write(lundmp,9010) 'Kpf ',nrowm,ncolm
         else
C--- create hypermatrix
            call HMGCR1(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
             &          mtndx)
            ndxkpf = mtndx
            nrowa = HMGINF(ndxkpf,8)
            ncola = HMGINF(ndxkpf,9)
C--- print stiffness hypermatrix control parameters
            write(lundmp,9000) 'Kpf ',nrowm,ncolm,nrowa,ncola,nrows,
             &          ncols,mttyp,mtwpe
         end if
      end if

```

```

c
c*** hypermatrix Ksf
    ndxksf = 0
    if(ifsym.eq.0) then
        mthed = 2004287491
        mttyp = 2
        mtwpe = NWDPE
        nrowm = nsupp
        ncolm = nfree
        nrows = min(nsizs,nsupp)
        ncols = nsizs
c--- check sizes
        if((nrowm.le.0).or.(ncolm.le.0)) then
            ndxksf = 0
            write(lundmp,9010) 'Ksf ',nrowm,ncolm
        else
c--- create hypermatrix
            call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                mtn dx)
            ndxksf = mtn dx
            nrowa = HMGINF(ndxksf,8)
            ncola = HMGINF(ndxksf,9)
c--- print stiffness hypermatrix control parameters
            write(lundmp,9000) 'Ksf ',nrowm,ncolm,nrowa,ncola,nrows,
&                ncols,mttyp,mtwpe
            end if
        end if
c
c*** hypermatrix Kfp
    mthed = 2004287499
    mttyp = 2
    mtwpe = NWDPE
    nrowm = nfree
    ncolm = npres
    nrows = nsizs
    ncols = min(nsizs,npres)
c--- check sizes
    if((nrowm.le.0).or.(ncolm.le.0)) then
        ndxkfp = 0
        write(lundmp,9010) 'Kfp ',nrowm,ncolm
    else
c--- create hypermatrix
        call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                mtn dx)
        ndxkfp = mtn dx
        nrowa = HMGINF(ndxkfp,8)
        ncola = HMGINF(ndxkfp,9)
c--- print stiffness hypermatrix control parameters
        write(lundmp,9000) 'kfp ',nrowm,ncolm,nrowa,ncola,nrows,
&                ncols,mttyp,mtwpe
        end if
c
c*** hypermatrix kpp
    mthed = 2004287500
    mttyp = 2
    mtwpe = NWDPE
    nrowm = npres
    ncolm = npres
    nrows = min(nsizs,npres)
    ncols = min(nsizs,npres)

```

```

c--- check sizes
      if((nrowm.le.0).or.(ncolm.le.0)) then
        ndxkpp = 0
        write(lundmp,9010) 'Kpp ',nrowm,ncolm
      else
c--- create hypermatrix
        call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
          &          mtndx)
        ndxkpp = mtndx
        nrowa = HMGINF(ndxkpp,8)
        ncola = HMGINF(ndxkpp,9)
c--- print stiffness hypermatrix control parameters
        write(lundmp,9000) 'kpp ',nrowm,ncolm,nrowa,ncola,nrows,
          &          ncols,mttyp,mtwpe
        end if
c
c*** hypermatrix ksp
      ndxksp = 0
      if(ifsym.eq.0) then
        mthed = 2004287501
        mttyp = 2
        mtwpe = NWDPE
        nrowm = nsupp
        ncolm = npres
        nrows = min(nsizs,nsupp)
        ncols = min(nsizs,npres)
c--- check sizes
        if((nrowm.le.0).or.(ncolm.le.0)) then
          ndxksp = 0
          write(lundmp,9010) 'Ksp ',nrowm,ncolm
        else
c--- create hypermatrix
          call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
            &          mtndx)
          ndxksp = mtndx
          nrowa = HMGINF(ndxksp,8)
          ncola = HMGINF(ndxksp,9)
c--- print stiffness hypermatrix control parameters
          write(lundmp,9000) 'Ksp ',nrowm,ncolm,nrowa,ncola,nrows,
            &          ncols,mttyp,mtwpe
          end if
        end if
c
c*** hypermatrix kfs
      mthed = 2004287490
      mttyp = 2
      mtwpe = NWDPE
      nrowm = nfree
      ncolm = nsupp
      nrows = nsizs
      ncols = min(nsizs,nsupp)
c--- check sizes
      if((nrowm.le.0).or.(ncolm.le.0)) then
        ndxkfs = 0
        write(lundmp,9010) 'Kfs ',nrowm,ncolm
      else

```

```

c---      create hypermatrix
          call HMGCR7(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
          ndbkfs = mtndx
          nrowa = HMGINF(ndbkfs,8)
          ncola = HMGINF(ndbkfs,9)
c---      print stiffness hypermatrix control parameters
          write(lundmp,9000) 'Kfs ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
          end if

c
c***      hypermatrix kps
          mthed = 2004287502
          mttyp = 2
          mtwpe = NWDPE
          nrowm = npres
          ncolm = nsupp
          nrows = min(nsizs,npres)
          ncols = min(nsizs,nsupp)
c---      check sizes
          if((nrowm.le.0).or.(ncolm.le.0)) then
              ndbkps = 0
              write(lundmp,9010) 'Kps ',nrowm,ncolm
          else
c---      create hypermatrix
              call HMGCR7(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&              mtndx)
              ndbkps = mtndx
              nrowa = HMGINF(ndbkps,8)
              ncola = HMGINF(ndbkps,9)
c---      print stiffness hypermatrix control parameters
              write(lundmp,9000) 'Kps ',nrowm,ncolm,nrowa,ncola,nrows,
&              ncols,mttyp,mtwpe
              end if

c
c***      hypermatrix kss
          ndbkss = 0
          if(ndbkss.gt.0) then
              mthed = 2004287492
              mttyp = 2
              mtwpe = NWDPE
              nrowm = nsupp
              ncolm = nsupp
              nrows = min(nsizs,nsupp)
              ncols = min(nsizs,nsupp)
c---      check sizes
              if((nrowm.le.0).or.(ncolm.le.0)) then
                  ndbkss = 0
                  write(lundmp,9010) 'Kss ',nrowm,ncolm
              else
c---      create hypermatrix
                  call HMGCR7(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                  mtndx)
                  ndbkss = mtndx
                  nrowa = HMGINF(ndbkss,8)
                  ncola = HMGINF(ndbkss,9)

```

```

c---      print stiffness hypermatrix control parameters
          write(lundmp,9000) 'kss  ',nrowm,ncolm,nrowa,ncola,nrows,
&                                ncols,mttyp,mtwpe
          end if
          end if
c
c***      hypermatrix uf
          mthed = 2004287493
          mttyp = 2
          mtwpe = NWDPE
          nrowm = nfree
          ncolm = nload
          nrows = nsizs
          ncols = min(nsizs,nload)
c---      check sizes
          if((nrowm.le.0).or.(ncolm.le.0)) then
              ndxuf = 0
              write(lundmp,9010) 'Uf  ',nrowm,ncolm
          else
c---          create hypermatrix
              call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                                mtndx)
              ndxuf = mtndx
              nrowa = HMGINF(ndxuf,8)
              ncola = HMGINF(ndxuf,9)
c---          print stiffness hypermatrix control parameters
              write(lundmp,9000) 'Uf  ',nrowm,ncolm,nrowa,ncola,nrows,
&                                ncols,mttyp,mtwpe
              end if
c
c***      hypermatrix up
          mthed = 2004287503
          mttyp = 2
          mtwpe = NWDPE
          nrowm = npres
          ncolm = nload
          nrows = min(nsizs,npres)
          ncols = min(nsizs,nload)
c---      check sizes
          if((nrowm.le.0).or.(ncolm.le.0)) then
              ndxup = 0
              write(lundmp,9010) 'Up  ',nrowm,ncolm
          else
c---          create hypermatrix
              call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                                mtndx)
              ndxup = mtndx
              nrowa = HMGINF(ndxup,8)
              ncola = HMGINF(ndxup,9)
c---          print stiffness hypermatrix control parameters
              write(lundmp,9000) 'Up  ',nrowm,ncolm,nrowa,ncola,nrows,
&                                ncols,mttyp,mtwpe
              end if
c

```

```

c*** hypermatrix us
ndxus = 0
if(ndxus.gt.0) then
mthed = 2004287494
mttyp = 2
mtwpe = NWDPE
nrowm = nsupp
ncolm = nload
nrows = min(nsizs,nsupp)
ncols = min(nsizs,nload)
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
    ndxus = 0
    write(lundmp,9010) 'Us  ',nrowm,ncolm
else
c--- create hypermatrix
    call HMGCRN(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
    ndxus = mtndx
    nrowa = HMGINF(ndxus,8)
    ncola = HMGINF(ndxus,9)
c--- print stiffness hypermatrix control parameters
    write(lundmp,9000) 'Us  ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
    end if
    end if
c
c*** hypermatrix rf
ndxrf = 0
if(ndxrf.gt.0) then
mthed = 2004287495
mttyp = 2
mtwpe = NWDPE
nrowm = nfree
ncolm = nload
nrows = nsizs
ncols = min(nsizs,nload)
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
    ndxrf = 0
    write(lundmp,9010) 'Rf  ',nrowm,ncolm
else
c--- create hypermatrix
    call HMGCRN(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
    ndxrf = mtndx
    nrowa = HMGINF(ndxrf,8)
    ncola = HMGINF(ndxrf,9)
c--- print stiffness hypermatrix control parameters
    write(lundmp,9000) 'Rf  ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
    end if
    end if
c

```

```

c*** hypermatrix rp
mthed = 2004287504
mttyp = 2
mtwpe = NWDPE
nrowm = npres
ncolm = nload
nrows = min(nsizs,npres)
ncols = min(nsizs,nload)
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
  ndxrp = 0
  write(lundmp,9010) 'Rp ',nrowm,ncolm
else
c--- create hypermatrix
  call HMGCRt(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
  ndxrp = mtndx
  nrowa = HMGINF(ndxrp,8)
  ncola = HMGINF(ndxrp,9)
c--- print stiffness hypermatrix control parameters
  write(lundmp,9000) 'Rp ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
  end if
c
c*** hypermatrix Rs
mthed = 2004287496
mttyp = 2
mtwpe = NWDPE
nrowm = nsupp
ncolm = nload
nrows = min(nsizs,nsupp)
ncols = min(nsizs,nload)
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
  ndxrs = 0
  write(lundmp,9010) 'Rs ',nrowm,ncolm
else
c--- create hypermatrix
  call HMGCRt(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
  ndxrs = mtndx
  nrowa = HMGINF(ndxrs,8)
  ncola = HMGINF(ndxrs,9)
c--- print stiffness hypermatrix control parameters
  write(lundmp,9000) 'Rs ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
  end if
c
c*** global degree of freedom table
mthed = 2004287497
nrowm = ntotv
ncolm = 2
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
  ndxfix = 0
  write(lundmp,9010) 'Fix ',nrowm,ncolm
else

```

```

c---      create hypertable
          call HTGCRT(mthed,ncolm,nrowm,mtptr,mtndx)
          ndxfix = mtndx
c---      print stiffness hypermatrix control parameters
          write(lundmp,9020) 'Fix ',nrowm,ncolm
          end if
c
          return
c
9000      format(1h0,1a5,4x      , ' matrix      ',i5,' by ',i5,/,
&          1h , 'address matrix      ',i5,' by ',i5,/,
&          1h , 'submatrices          ',i5,' by ',i5,/,
&          1h , 'matrix type          ',i5,/,
&          1h , 'words per element    ',i5)
c
9010      format(1h0,1a5,4x      , ' matrix not created ',/,
&          1h , 'Number of rows      : ',i5,/,
&          1h , 'Number of columns : ',i5)
c
9020      format(1h0,1a5,4x      , ' table      ',i5,' by ',i5)
c
          end
c
c
          subroutine INDEXS ( indexk, indexu, indexr, ifsym,
&                          ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
&                          ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
&                          ndxksf, ndxksp, ndxkss, ndxus, ndxrs )
c
c.....
c      purpose :
c      sets up look-up table for stiffness,displacement and loads
c
c      arguments
c o      indexk - stiffness look-up table
c o      indexu - displacement look-up table
c o      indexr - loads look-up table
c i      ifsym - symmetry flag
c i      ndxkff - index to kff hypermatrix
c i      ndxkfp - index to kfp hypermatrix
c i      ndxkfs - index to kfs hypermatrix
c i      ndxuf - index to uf hypermatrix
c i      ndxrf - index to rf hypermatrix
c i      ndxkpf - index to kpf hypermatrix
c i      ndxkpp - index to kpp hypermatrix
c i      ndxkps - index to kps hypermatrix
c i      ndxup - index to up hypermatrix
c i      ndxrp - index to rp hypermatrix
c i      ndxksf - index to ksf hypermatrix
c i      ndxksp - index to ksp hypermatrix
c i      ndxkss - index to kss hypermatrix
c i      ndxus - index to us hypermatrix
c i      ndxrs - index to rs hypermatrix
c.....
c
          integer indexk(3,3),
&          indexu(3),
&          indexr(3)

```

```

c
c--- check for symmetry
      if(ifsym.eq.1) then
         ndxkpf = ndxkfp
         ndxksf = ndxkfs
         ndxksp = ndxkps
      end if

c
c--- stiffnesses
      indexk(1,1) = ndxkff
      indexk(2,1) = ndxkpf
      indexk(3,1) = ndxksf
      indexk(1,2) = ndxkfp
      indexk(2,2) = ndxkpp
      indexk(3,2) = ndxksp
      indexk(1,3) = ndxkfs
      indexk(2,3) = ndxkps
      indexk(3,3) = ndxkss

c
c--- displacements
      indexu(1)  = ndxauf
      indexu(2)  = ndxup
      indexu(3)  = ndxus

c
c--- loads
      indexr(1)  = ndxrf
      indexr(1)  = ndxuf
      indexr(2)  = ndxrp
      indexr(3)  = ndxrs

c
      return
      end

c
c
c      subroutine PRESICS ( ndxrf, ndxkfp, ndxup )
c
c.....
c      purpose:
c      alters gload for prescribed displacements
c      performs  $R_f = R_f - K_{fs}.U_s$ 
c
c      arguments :
c      ndxrf - index to rf hypermatrix
c      ndxkfp - index to kfp hypermatrix
c      ndxup - index to up hypermatrix
c.....
c
c      implicit double precision(a-h,o-z)

c
c--- set flag to subtract matrix multiplication from result
      factr = -1.0d0
c--- set flag not to transpose rf
      kfptp = 0
c--- set flag not to transpose kfs
      nuptp = 0
c--- Kfp not symmetric
      kfpsym = 0

```

```

c--- Up not symmetric
      nupsym = 0
c--- perform operation
      call HMMLT ( ndbkfp, ndxup, ndxrf, factr,
&                kfptp, nuftp, kfpsym, nupsym )
c
      return
      end
c
c
      subroutine CHLSKY ( gstndx, gldndx )
c
c.....
c      purpose:
c      routine to solve set of linear equation of a symmetric matrix
c
c i   gstndx - index to stiffness hypermatrix
c i   gldndx - index to load hypermatrix
c.....
c
      implicit double precision(a-h,o-z)
      parameter MSIZS=15
      real*8  summa(MSIZS,MSIZS),
&           pivot(MSIZS)
c
      integer HMGINF
      integer gstndx,gldndx
c
c--- dimension of stiffness address matrix
      nsizh = HMGINF(gstndx,8)
c--- dimension of load address matrix
      nsizi = HMGINF(gldndx,9)
c
c--- calculate matrix L
c
      do 100 irowh = 1,nsizh
      call SUMTRI ( gstndx, irowh,  summa )
      call CHKTRI ( gstndx, irowh,  summa,  pivot )
c
      do 200 icolh = irowh+1,nsizh
      call SUMRCT ( gstndx, irowh, icolh,  summa )
      call CHKRCT ( gstndx, irowh, icolh,  summa,  pivot )
200  continue
100  continue
c
c--- calculate Y
c
      do 400 icoli = 1,nsizi
      do 410 irowh = 1,nsizh
      call SUMYYY ( gstndx, gldndx, irowh, icoli,  summa )
      call CHKYYY ( gstndx, gldndx, irowh, icoli,  summa )
410  continue
400  continue
c

```

```

c--- calculate X
c
  do 500 icoli = 1,nsizi
  do 510 irowh = nsizh,1,-1
  call SUMXXX ( gstndx, gldndx, irowh, icoli, summa )
  call CHKXXX ( gstndx, gldndx, irowh, icoli, summa )
510 continue
500 continue
c
  return
end

c
c
c   subroutine CHKTRI ( gstndx, irowh, summa, pivot )
c
c.....
c   purpose:
c   calculates the L values in the cholesky solution for
c   the pivot submatrix
c
c i   gstndx - index to global stiffness hypermatrix
c i   irowh  - row in addressing matrix
c i   summa  - summation for (irowh,irowh)
c o   pivot  - pivot vector for submatrix
c.....
c
c   parameter NWDPE=2
c   parameter MSIZS=15
c   parameter MPOOL=MSIZS*MSIZS*NWDPE
c   common /JUNK/ pool(MPOOL)
c   real*8 leqns(MSIZS,MSIZS),
c   &      summa(MSIZS,MSIZS),
c   &      pivot(MSIZS)
c   equivalence (leqns(1,1),pool(1))
c   integer gstndx
c
c--- retrieve size of submatrix
c   call HMCISZ(gstndx,irowh,irowh,nrowp,ncolp,nsds)
c
c--- retrieve submatrix
c   call SMTGET(gstndx,irowh,irowh,leqns)
c
c   do 100 ieqns = 1,nrowp
c
c--- calculate pivot and store
c   leqns(ieqns,ieqns) = sqrt(leqns(ieqns,ieqns)-summa(ieqns,ieqns))
c   pivot(ieqns) = leqns(ieqns,ieqns)
c
c--- calculate the rest of the row
c   do 200 icolp = ieqns+1,ncolp
c
c   leqns(ieqns,icolp) = (leqns(ieqns,icolp) - summa(ieqns,icolp))/
c   &      pivot(ieqns)
c

```

```

c--- update summation
do 300 irowp = ieqns+1,icolp
  summa(irowp,icolp) = summa(irowp,icolp) +
&                      leqns(ieqns,irowp)*leqns(ieqns,icolp)
300  continue
200  continue
100  continue
c
c--- save submatrix
call SMTPUT(gstndx,irowh,irowh,leqns)
c
c  return
c  end
c
c
c  subroutine CHKRCT ( gstndx, irowh, icolh, summa, pivot )
c.....
c  purpose:
c  calculates the L values in the cholesky solution
c  for the rectangular part of the address matrix
c
c i  gstndx - index to global stiffness hypermatrix
c i  irowh  - row in addressing matrix
c i  icolh  - column in addressing matrix
c i  summa  - summation for (irowh,icolh)
c i  pivot  - pivot vector for submatrix
c.....
c
c  implicit double precision(a-h,o-z)
c  parameter NWDPE=2
c  parameter MSIZS=15
c  parameter MPOOL=MSIZS*MSIZS*NWDPE*2
c  parameter LOCP1=MSIZS*MSIZS*NWDPE+1
c  common /JUNK/ pool(MPOOL)
c  real*8 leqns(MSIZS,MSIZS),
&         lrows(MSIZS,MSIZS),
&         summa(MSIZS,MSIZS),
&         pivot(MSIZS)
c  equivalence (leqns(1,1),pool(1))
c  equivalence (lrows(1,1),pool(LOCP1))
c  integer gstndx
c
c--- retrieve size of sub matrix
call HMGSIZ(gstndx,irowh,icolh,nrowp,ncolp,nwds)
c
c--- retrieve submatrices
call SMIGET(gstndx,irowh,icolh,leqns)
call SMIGET(gstndx,irowh,irowh,lrows)
c
c--- loop over submatrix
do 100 ieqns = 1,nrowp
c
c--- calculate L values
do 200 icolp = 1,ncolp
  leqns(ieqns,icolp) = (leqns(ieqns,icolp) - summa(ieqns,icolp))/
&                      pivot(ieqns)
c

```

```

c--- update summation
do 300 irowp = ieqns+1,nrowp
  summa(irowp,icolp) = summa(irowp,icolp) +
&                      lrows(ieqns,irowp)*leqns(ieqns,icolp)
300  continue
200  continue
100  continue
c
c--- save submatrix
call SMIPUT(gstndx,irowh,icolh,leqns)
c
  return
end
c
c
  subroutine CHKYYY ( gstndx, gldndx, irowh, icoli, summa )
c
c.....
c  purpose:
c  calculates the l values in the cholesky solution
c  for the pivotal part of the address matrix
c
c i  gstndx - index to global stiffness hypermatrix
c i  gldndx - index to load  stiffness hypermatrix
c i  irowh  - row in stiffness addressing matrix
c i  icoli  - column in load addressing matrix
c i  summa  - summation for (irowh,icolh)
c.....
c
  implicit double precision(a-h,o-z)
  parameter NWDPE=2
  parameter MSIZS=15
  parameter MPOOL=MSIZS*MSIZS*NWDPE*2
  parameter LOCP1=MSIZS*MSIZS*NWDPE+1
  common /JUNK/ pool(MPOOL)
  real*8 leqns(MSIZS,MSIZS),
&        summa(MSIZS,MSIZS),
&        yeqns(MSIZS,MSIZS)
  equivalence (leqns(1,1),pool(1))
  equivalence (yeqns(1,1),pool(LOCP1))
  integer gstndx,gldndx
c
c--- retrieve stiffness submatrix
call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
call SMIGET(gstndx,irowh,irowh,leqns)
c
c--- retrieve load submatrix
call HMGSIZ(gldndx,irowh,icoli,nrowl,ncoll,nwds)
call SMIGET(gldndx,irowh,icoli,yeqns)
c
  do 100 ieqns = 1,nrowp
c
c--- loop over loads
do 200 icoll = 1,ncoll
c
c--- calculate Y vector
  yeqns(ieqns,icoll) = (yeqns(ieqns,icoll) - summa(ieqns,icoll))/
&                      leqns(ieqns,ieqns)

```

```

c
c--- update summation
      do 300 irowp = ieqns+1,nrowp
        summa(irowp,icoll) = summa(irowp,icoll) +
&          leqns(ieqns,irowp)*yeqns(ieqns,icoll)
300   continue
200   continue
100   continue
c
c--- save submatrix
      call SMTPUT(gldndx,irowh,icoli,yeqns)
c
      return
      end
c
c
      subroutine CHKXXX ( gstndx, gldndx, irowh, icoli, summa )
c.....
c      purpose:
c      calculates the l values in the cholesky solution
c      for the pivotal part of the address matrix
c
c i      gstndx - index to global stiffness hypermatrix
c i      gldndx - index to load stiffness hypermatrix
c i      irowh  - row in stiffness addressing matrix
c i      icoli  - column in load addressing matrix
c i      summa  - summation for (irowh,icolh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS),
&          yeqns(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      equivalence (yeqns(1,1),pool(LOCP1))
      integer gstndx,gldndx
c
c--- retrieve stiffness submatrix
      call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
      call SMTGET(gstndx,irowh,irowh,leqns)
c
c--- retrieve load submatrix
      call HMGSIZ(gldndx,irowh,icoli,nrowl,ncoll,nwds)
      call SMTGET(gldndx,irowh,icoli,yeqns)
c
      do 100 ieqns = nrowp,1,-1
c
c--- loop over loads
      do 200 icoll = 1,ncoll
c
c--- calculate X vector
      yeqns(ieqns,icoll) = (yeqns(ieqns,icoll) - summa(ieqns,icoll))/
&          leqns(ieqns,ieqns)

```

```

c
c--- update summation
      do 300 irowp = ieqns-1,1,-1
        summa(irowp,icoll) = summa(irowp,icoll) +
&          leqns(irowp,ieqns)*yeqns(ieqns,icoll)
300   continue
200   continue
100   continue
c
c--- save submatrix
      call SMTPUT(gldndx,irowh,icoli,yeqns)
c
      return
      end
c
c
      subroutine SUMRI ( gstndx, irowh, summa )
c
c.....
c   purpose:
c   calculates summation for sub-matrix (irowh,icolh)
c
c i   gstndx - index to global stiffness hypermatrix
c i   irowh  - row in addressing matrix
c o   summa  - summation array for address block (irowh,irowh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter AZERO=0.0d0
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      integer gstndx
      integer HMGINF
      nrows = HMGINF(gstndx,6)
c
c--- retrieve submatrix parameters
      call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
c
c--- zero summation
      do 10 irowp = 1,nrowp
        do 10 icolp = 1,ncolp
10     summa(irowp,icolp) = AZERO
c
c--- loop over submatrices
      do 500 jrowh = 1,irowh-1
c
c--- retrieve submatrices
      call SMTGET(gstndx,jrowh,irowh,leqns)
c

```

```

c--- calculate summation
      do 300 irows = 1,nrows
      do 100 irowp = 1,nrowp
      do 200 icolp = irowp,ncolp
      summa(irowp,icolp) = summa(irowp,icolp) +
&                                leqns(irows,irowp)*leqns(irows,icolp)
200  continue
100  continue
300  continue
c
500  continue
c
      return
      end

c
c
      subroutine SUMRCT ( gstndx, irowh, icolh, summa )
c
c.....
c  purpose:
c  calculates summation for sub-matrix (irowh,icolh)
c
c i  gstndx - index to global stiffness hypermatrix
c i  irowh  - row in addressing matrix
c i  icolh  - column in addressing matrix
c o  summa  - summation array for address block (irowh,icolh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter AZERO=0.0d0
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          lrows(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      equivalence (lrows(1,1),pool(LOCP1))
      integer gstndx
      integer HMGINF
      nrows = HMGINF(gstndx,6)

c
c--- retrieve submatrix parameters
      call HMGSIZ(gstndx,irowh,icolh,nrowp,ncolp,nwds)

c
c--- zero summation
      do 10 irowp = 1,nrowp
      do 10 icolp = 1,ncolp
10  summa(irowp,icolp) = AZERO
c
c--- loop over submatrices
      do 500 jrowh = 1,irowh-1
c
c--- retrieve submatrices
      call SMTGET(gstndx,jrowh,icolh,leqns)
      call SMTGET(gstndx,jrowh,irowh,lrows)

```

```

c
c--- calculate summation
do 300 irows = 1,nrows
do 100 irowp = 1,nrowp
do 200 icolp = 1,ncolp
  summa(irowp,icolp) = summa(irowp,icolp) +
&                      lrows(irows,irowp)*leqns(irows,icolp)
200 continue
100 continue
300 continue
500 continue
c
  return
  end

c
c
  subroutine SUMYYY ( gstndx, gldndx, irowh, icoli, summa )
c.....
c  purpose:
c  calculates summation for sub-matrix (irowh,icolh)
c
c i  gstndx - index to global stiffness hypermatrix
c i  gldndx - index to load stiffness hypermatrix
c i  irowh - row in stiffness addressing matrix
c i  icoli - row in load addressing matrix
c o  summa - summation for (irowh,icolh)
c.....
c
  implicit double precision(a-h,o-z)
  parameter NWDPE=2
  parameter AZERO=0.0d0
  parameter MSIZS=15
  parameter MPOOL=MSIZS*MSIZS*NWDPE*2
  parameter LOCP1=MSIZS*MSIZS*NWDPE+1
  common /JUNK/ pool(MPOOL)
  real*8 leqns(MSIZS,MSIZS),
&        summa(MSIZS,MSIZS),
&        yeqns(MSIZS,MSIZS)
  equivalence (leqns(1,1),pool(1))
  equivalence (yeqns(1,1),pool(LOCP1))
  integer gstndx,gldndx
  integer HMGINF
  nrows = HMGINF(gstndx,6)

c
c--- retrieve submatrix parameters
call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
call HMGSIZ(gldndx,irowh,icoli,nrowl,ncoll,nwds)

c
c--- zero summation
do 10 irowl = 1,nrowl
do 10 icoll = 1,ncoll
10  summa(irowl,icoll) = AZERO
c
c--- loop over submatrices
do 500 jrowh = 1,irowh-1
c

```

```

c--- retrieve submatrices
call SMGET(gstndx,jrowh,irowh,leqns)
call SMGET(gldndx,jrowh,icoli,yeqns)
c
c--- calculate summation
do 300 irows = 1,nrows
do 100 irowp = 1,nrowp
do 200 icoll = 1,ncoll
  summa(irowp,icoll) = summa(irowp,icoll) +
&                      leqns(irows,irowp)*yeqns(irows,icoll)
200 continue
100 continue
300 continue
c
500 continue
c
  return
  end
c
c
  subroutine SUMXXX ( gstndx, gldndx, irowh, icoli, summa )
c
c.....
c  purpose:
c  calculates summation for submatrix (irowh,icolh)
c
c i  gstndx - index to global stiffness hypermatrix
c i  gldndx - index to load stiffness hypermatrix
c i  irowh - row in stiffness addressing matrix
c i  icoli - column in load addressing matrix
c o  summa - summation vector (irowh)
c.....
c
  implicit double precision(a-h,o-z)
  parameter NWDPE=2
  parameter AZERO=0.0d0
  parameter MSIZS=15
  parameter MPOOL=MSIZS*MSIZS*NWDPE*2
  parameter LOCP1=MSIZS*MSIZS*NWDPE+1
  common /JUNK/ pool(MPOOL)
  real*8 leqns(MSIZS,MSIZS),
&        summa(MSIZS,MSIZS),
&        yeqns(MSIZS,MSIZS)
  equivalence (leqns(1,1),pool(1))
  equivalence (yeqns(1,1),pool(LOCP1))
  integer HMGINF
  integer gstndx,gldndx
c
c--- retrieve size of stiffness address matrix
nsizh = HMGINF(gstndx,8)
c
c--- retrieve load submatrix size
call HMGSIZ(gldndx,1,icoli,nrowl,ncoll,nwds)
c
c--- zero summation
do 10 irowl = 1,nrowl
do 10 icoll = 1,ncoll
10  summa(irowl,icoll) = AZERO

```

```

c
c--- loop over submatrices
do 500 jrowh = nsizh,irowh+1,-1
c
c--- retrieve submatrix parameters
call HMGSIZ(gstndx,irowh,jrowh,nrowp,ncolp,nwds)
c
c--- retrieve submatrices
call SMTGET(gstndx,irowh,jrowh,leqns)
call SMTGET(gldndx,jrowh,icoli,yeqns)
c
c--- calculate summation
do 200 icolp = ncolp,1,-1
do 100 irowp = nrowp,1,-1
do 300 icoll = 1,ncoll
  summa(irowp,icoll) = summa(irowp,icoll) +
&                      leqns(irowp,icolp)*yeqns(icoll,icoll)
300  continue
100  continue
200  continue
c
500  continue
c
      return
      end

c
c
      subroutine SMTGET ( ndxhmg, irowh, icolh, array )
c
c.....
c      purpose
c      retrieves a submatrix , ensuring that the mapping is correct
c
c i   ndxhmg - index to hypermatrix
c i   irowh  - submatrix row
c i   icolh  - submatrix column
c o   array  - matrix retrieved
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter MSIZS=15
      parameter MARAY=MSIZS*MSIZS*NWDPE
      real*8 array(MARAY)

c
      call HMGSIZ(ndxhmg,irowh,icolh,nrowp,ncolp,nwdhmg)
c
c--- check if mapping is needed
      if(nrowp.ne.MSIZS) then
          call HMGGET(ndxhmg,irowh,icolh,array,nwdhmg)
c--- do mapping
          do 100 icolp = ncolp,2,-1
          do 100 irowp = nrowp,1,-1
              nloca = (icolp-1)*nrowp + irowp
              nlocb = (icolp-1)*MSIZS + irowp
              array(nlocb) = array(nloca)
100      continue
          else
              call HMGGET(ndxhmg,irowh,icolh,array,nwdhmg)
          end if

```

```

c      return
c      end

c
c
c      subroutine SMTPUT ( ndxhmg, irowh , icolh, array )
c.....
c      purpose
c      replaces a submatrix , ensuring that the mapping is correct
c
c i    ndxhmg - index to hypermatrix
c i    irowh  - submatrix row
c i    icolh  - submatrix column
c i    array  - matrix replaced
c.....
c
c      implicit double precision(a-h,o-z)
c      parameter NWDPE=2
c      parameter MSIZS=15
c      parameter MARAY=MSIZS*MSIZS*NWDPE
c      real*8 array(MARAY)
c
c---   retrieve submatrix size
c      call HMGSIZ(ndxhmg,irowh,icolh,nrowp,ncolp,nwdhmg)
c
c---   check if copy is needed
c      if(nrowp.ne.MSIZS) then
c---       do copy
c           do 100 icolp = 2,ncolp
c           do 100 irowp = 1,nrowp
c               nloca = (icolp-1)*MSIZS + irowp
c               nlocb = (icolp-1)*nrowp + irowp
c               array(nlocb) = array(nloca)
100      continue
c           call HMGPOT(ndxhmg,irowh,icolh,array,nwdhmg)
c       else
c           call HMGPOT(ndxhmg,irowh,icolh,array,nwdhmg)
c       end if
c
c      return
c      end

c
c
c      subroutine CREAT ( ndxrp, ndxkpf, ndxuf, ndxkpp, ndxup,
c      & ndxrs, ndxksf, ndxksp, ifsym )
c.....
c      purpose:
c      calculates reactions
c
c      ndxrs - index to rs hypermatrix
c      ndxksf - index to ksf hypermatrix
c      ndxuf - index to uf hypermatrix
c      ndxkss - index to kss hypermatrix
c      ndxus - index to us hypermatrix
c.....
c

```

```

        implicit double precision(a-h,o-z)
        integer trana,tranb

c
c---- calculate  $R_p = -R_p' + K_{pf}.U_f + K_{pp}.U_p$ 
c
c---- set flag for addition
        factr = 1.0d0
c---- set flag not to transpose A
        if(ifsym.eq.0) then
            trana = 0
        else
            trana = 1
        end if
c---- set flag not to transpose B
        tranb = 0
c----  $K_{pf}$  not symmetric
        kpfsym = 0
c----  $U_f$  not symmetric
        nufsym = 0
c
c---- multiply hypermatrices
        call HMTMLT ( ndxkpf, ndxuf, ndxrp, factr,
&                    trana, tranb, kpfsym, nufsym )
c
c---- set flag for addition
        factr = 1.0d0
c---- set flag not to transpose A
        trana = 0
c---- set flag not to transpose B
        tranb = 0
c----  $K_{pp}$  symmetric
        kppsym = ifsym
c----  $U_p$  not symmetric
        nupsym = 0
c
c---- multiply hypermatrices
        call HMTMLT ( ndxkpp, ndxup, ndxrp, factr,
&                    trana, tranb, kppsym, nupsym )
c
c---- calculate  $R_s = -R_s' + K_{sf}.U_f + K_{sp}.U_p$ 
c
c---- set flag for addition
        factr = 1.0d0
c---- set flag not to transpose A
        if(ifsym.eq.0) then
            trana = 0
        else
            trana = 1
        end if
c---- set flag not to transpose B
        tranb = 0
c----  $K_{sf}$  not symmetric
        ksfsym = 0
c----  $U_f$  not symmetric
        nufsym = 0
c

```

```

c--- multiply hypermatrices
      call HMIMLT ( ndxksf, ndxuf, ndxrs, factr,
&                  trana, tranb, ksfsym, nufsym )
c
c--- set flag for addition
      factr = 1.0d0
c--- set flag not to transpose A
      if(ifsym.eq.0) then
        tranb = 0
      else
        tranb = 1
      end if
c--- set flag not to transpose B
      tranb = 0
c--- Ksp not symmetric
      kspsym = 0
c--- Up not symmetric
      nupsym = 0
c
c--- multiply hypermatrices
      call HMIMLT ( ndxksp, ndxup, ndxrs, factr,
&                  trana, tranb, kspsym, nupsym )
c
      return
      end
c
c
      subroutine DPIHMT(string,hmgndx)
c
c.....
c      purpose:
c      dumps integer hypermatrix
c
c      arguments:
c      string - character header
c      hmgndx - index to hypermatrix
c.....
c
      implicit double precision(a-h,o-z)
      character*(*) string
      integer HMGINF
      integer hmgndx,hmgwpe
      integer value
c
c--- retrieve lun no.
      lundmp = LUNGET(8)
c
      write(lundmp,9000) string
c
      if(hmgndx.eq.0) then
        write(lundmp,9000) 'hypertable does not exit'
      else
        return
      end if
c
9000 format(1h ,a)
      end

```

```

c
c      subroutine DPRHMT(string,hmgndx)
c
c.....
c      purpose:
c      dumps real hypermatrix
c
c      arguments:
c      string - character header
c      hmgndx - index to hypermatrix
c.....
c
c      implicit double precision(a-h,o-z)
c      character*(*) string
c      integer HMGINF
c      integer hmgndx,hmgwpe
c
c--- retrieve lun no.
c      lundmp = LUNGET(8)
c
c      write(lundmp,9000) string
c
c      if(hmgndx.eq.0) then
c         write(lundmp,9000) 'hypermatrix does not exit'
c      else
c
c--- retrieve number of words per element
c      hmgwpe = HMGINF(hmgndx,3)
c
c--- retrieve number of rows and columns in matrix
c      nrowm = HMGINF(hmgndx,4)
c      ncolm = HMGINF(hmgndx,5)
c      do 100 irowm = 1,nrowm
c         do 100 icolm = 1,ncolm
c            call HMGEVG(hmgndx,irowm,icolm,value,hmgwpe)
c            write(lundmp,*) irowm,icolm,value
c100    continue
c         end if
c
c      return
c
c9000  format(1h ,a)
c      end
c
c      subroutine HMTMLT ( ndxhma, ndxhmb, ndxhmc, factr,
c      &                  trana, tranb, isyma, isymb )
c
c.....
c      purpose
c      subroutine multiplies two hypermatrices and adds or subtracts the
c      result from the resultant hypermatrix
c

```

```

c      arguments:
c
c      ndxhma - index to first hypermatrix
c      ndxhmb - index to second hypermatrix
c      ndxhmc - index to resultant hypermatrix
c      factr  - indicator for addition/subtraction to result matrix
c      trana  - flag to indicate transform A before multiplication
c      tranb  - flag to indicate transform B before multiplication
c      isyma  - flag to indicate that A is symmetric
c      isymb  - flag to indicate that B is symmetric
c.....
c
c      implicit double precision(a-h,o-z)
c      parameter MSIZS=15
c      real*8 arrayc(MSIZS,MSIZS)
c      integer HMGINF
c      integer trana,tranb,transa,transb
c
c--- retrieve lun no.
c      lunerr = LUNGET(6)
c
c--- check for zero index
c      if(ndxhma.eq.0) goto 9999
c      if(ndxhmb.eq.0) goto 9999
c
c--- retrieve sizes matrices
c      nrowa = HMGINF(ndxhma,4)
c      ncola = HMGINF(ndxhma,5)
c      nrowb = HMGINF(ndxhmb,4)
c      ncolb = HMGINF(ndxhmb,5)
c      nrowc = HMGINF(ndxhmc,4)
c      ncolc = HMGINF(ndxhmc,5)
c
c--- check if matrices may be multiplied
c--- compare A and B
c      if(trana.eq.0) then
c          if(tranb.eq.0) then
c              if(ncola.ne.nrowb) then
c                  write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c                  stop
c              end if
c          else
c              if(ncola.ne.ncolb) then
c                  write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c                  stop
c              end if
c          end if
c      else
c          if(tranb.eq.0) then
c              if(nrowa.ne.nrowb) then
c                  write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c                  stop
c              end if
c          else
c              if(nrowa.ne.ncolb) then
c                  write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c                  stop
c              end if
c          end if
c      end if

```

```

c
c--- compare A and C
  if(trana.eq.0) then
    if(nrowa.ne.nrowc) then
      write(lunerr,9000) 'Hma',nrowa,ncola,'Hmc',nrowc,ncolc
      stop
    endif
  else
    if(ncola.ne.nrowc) then
      write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
      stop
    end if
  end if

c
c--- compare B and C
  if(tranb.eq.0) then
    if(ncolb.ne.ncolc) then
      write(lunerr,9000) 'Hmb',nrowb,ncolb,'Hmc',nrowc,ncolc
      stop
    endif
  else
    if(nrowb.ne.ncolc) then
      write(lunerr,9000) 'Hmb',nrowb,ncolb,'Hmc',nrowc,ncolc
      stop
    end if
  end if

c
c--- retrieve number of rows and columns in address matrices
  nrowa = HMGINF(ndxhma,8)
  ncola = HMGINF(ndxhma,9)
  nrowb = HMGINF(ndxhmb,8)
  ncolb = HMGINF(ndxhmb,9)
  nrowc = HMGINF(ndxhmc,8)
  ncolc = HMGINF(ndxhmc,9)

c
  if(trana.eq.0) then
    nmult = ncola
  else
    nmult = nrowa
  end if

c
c--- loop over address matrix
  do 100 irowc = 1,nrowc
  do 110 icolc = 1,ncolc

c
c--- retrieve resultant sub-matrix
  call HMGSIZ(ndxhmc,irowc,icolc,nrowsc,ncolsc,nwds)
  call SMTGET(ndxhmc,irowc,icolc,araye)

c
  do 120 imult = 1,nmult

c
c--- select correct hypermatrices
  if(isyma.eq.0) then
    transa = trana
    if(trana.eq.0) then
      irowa = irowc
      icola = imult
    else

```

```

        irowa = imult
        icola = irowc
    end if
else
    if(irowc.le.imult) then
        irowa = irowc
        icola = imult
        transa = 0
    else
        irowa = imult
        icola = irowc
        transa = 1
    end if
end if

c
if(isymb.eq.0) then
    transb = tranb
    if(tranb.eq.0) then
        irowb = imult
        icolb = icolc
    else
        irowb = icolc
        icolb = imult
    end if
else
    if(imult.le.icolc) then
        irowb = imult
        icolb = icolc
        transa = 0
    else
        irowb = icolc
        icolb = imult
        transb = 1
    end if
end if

c
c--- multiply sub-matrices
call SMIMLT ( ndxhma, irowa, icola,
&             ndxhmb, irowb, icolb,
&             arrayc, nrowsc, ncolsc,
&             factr, transa, transb,
&             isyma, isymb )
120 continue
c
    call SMIPUT(ndxhmc,irowc,icolc,arrayc)
c
110 continue
100 continue
c
9000 format(1h , 'Hypermatrices do not match for multiplication',/,
&          1h , 'Hypermatix ',a3,' : nrow = ',i5,/,
&          1h ,14x,           ' : ncol = ',i5,/,
&          1h , 'Hypermatix ',a3,' : nrow = ',i5,/,
&          1h ,14x,           ' : ncol = ',i5)
c
9999 continue
c
    return
end

```

```

c
c
c      subroutine SMIMLT ( ndxhma, irowa, icola,
c      &                  ndxhmb, irowb, icolb,
c      &                  arrayc, nrowsc, ncolsc,
c      &                  factr, transa, transb,
c      &                  isyma, isymb )
c.....
c      purpose:
c      multiplies two submatrices
c
c      arguments:
c      ndxhma - index to first hypermatyrix
c      irowa  - row in address matrix
c      icola  - column in address matrix
c      ndxhmb - index to second hypermatyrix
c      irowb  - row in address matrix
c      icolb  - column in address matrix
c      arrayc - resultant matrix
c      nrowsc - number of rows in submatrix
c      ncolsc - number of columns in submatrix
c      factr  - flag to indicate multiplication/subtraction
c      transa - flag to indicate transpose A before multiplication
c      transb - flag to indicate transpose B before multiplication
c      isyma  - flag to indicate that A is symmetric
c      isymb  - flag to indicate that B is symmetric
c.....
c
c      implicit double precision(a-h,o-z)
c      parameter NWDPE=2
c      parameter MSIZS=15
c      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
c      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
c      common /JUNK/ pool(MPOOL)
c      real*8 araya(MSIZS,MSIZS),
c      &      arrayb(MSIZS,MSIZS),
c      &      arrayc(MSIZS,MSIZS)
c      equivalence(araya(1,1),pool(1))
c      equivalence(arrayb(1,1),pool(LOCP1))
c      integer transa,transb
c
c--- retrieve submatrices
c      call SMTGET(ndxhma,irowa,icola,araya)
c      call SMTGET(ndxhmb,irowb,icolb,arrayb)
c
c      call HMGSIZ(ndxhma,irowa,icola,nrowsa,ncolsa,nwds)
c      if(transa.eq.0) then
c          nmult = ncolsa
c      else
c          nmult = nrowsa
c      end if
c
c--- do multiplication
c      do 100 irowsc = 1,nrowsc
c      do 100 icolsc = 1,ncolsc
c      do 100 imult = 1,nmult
c

```

```

if(isyma.eq.0) then
  if(transa.eq.0) then
    irowsa = irowsc
    icolsa = imult
  else
    irowsa = imult
    icolsa = irowsc
  end if
else
  if(irowa.eq.icola) then
    irowsa = min(irowsc,imult)
    icolsa = max(irowsc,imult)
  else
    if(transa.eq.0) then
      irowsa = irowsc
      icolsa = imult
    else
      irowsa = imult
      icolsa = irowsc
    end if
  end if
end if

```

c

```

if(isymb.eq.0) then
  if(transb.eq.0) then
    irowsb = imult
    icolsb = icolsc
  else
    irowsb = icolsc
    icolsb = imult
  end if
else
  if(irowb.eq.icolb) then
    irowsb = min(imult,icolsc)
    icolsb = max(imult,icolsc)
  else
    if(transb.eq.0) then
      irowsb = imult
      icolsb = icolsc
    else
      irowsb = icolsc
      icolsb = imult
    end if
  end if
end if

```

c

```

arrayc(irowsc,icolsc) = arrayc(irowsc,icolsc) + factr*
& arraya(irowsa,icolsa)*arrayb(irowsb,icolsb)

```

c

100 continue

c

```

return
end

```

APPENDIX D

A SAMPLE PROBLEM

This appendix contains the description of a sample problem and the solution that was obtained with the Cholesky Hypermatrix Solution Scheme implemented in the NEW NOSTRUM research code. These results are also compared with those obtained from equivalent ABAQUS examples. Both of the programs were implemented on the MicroVax II at the University of Cape Town.

D.1 The Finite Element Model

The sample problem, from Hinton and Owen [2], is a simple cantilevered beam with an end point load. The finite element model of the structural system to be analysed is shown in figure D.1.



Figure D.1: Finite element model of cantilevered beam

The original problem, example 1, used a single row of 10 elements, which has 106 degrees of freedom. To test the equation solution package two further discretisations were used. Figure D.2 shows the discretization for example 2 with 3 rows of 10 elements, which has 234 degrees of freedom.

21	22	23	24	25	26	27	28	29	30
11	12	13	14	15	16	17	18	19	20
1	2	3	4	5	6	7	8	9	10

Figure D.2: Cantilevered beam 3x10 elements

Figure D.3 shows the discretisation for example 3 with 3 rows of 30 elements, which has 674 degrees of freedom.

61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Figure D.3: Cantilevered beam 3x30 elements

D.2 Discussion

The accuracy of the results obtained with the NEW NOSTRUM suite are compared with those obtained with the ABAQUS program and theoretical values based on simple beam bending theory. The results for the beam end deflection are illustrated in table D.1.

Example	Theoretical	NOSTRUM	ABAQUS
1: 1x10	-0.135	-0.13427187	-0.1343
2: 3x10	-0.135	-0.13486302	-0.1349
3: 3x30	-0.135	-0.13520581	-0.1352

Table D.1: End deflections of the cantilevered beam.

APPENDIX E

HYPERMATRIX ROUTINES DESCRIPTION

In this appendix the FORTRAN routines in the hypermatrix module are described in detail. The routines were developed by Vos [17] and can be used by an application program to create and access two-level hypermatrices. The routines in the general hypermatrix module call the routines in the DAS-package [17] in turn to allocate space in virtual memory, as well as to store information therein and to retrieve it again.

E.1 HMGINI - Initialize the Internal Data Structure

Purpose:

HMGINI initializes the internal data structure and variables of the general two-level hypertable module. It must be called prior to any other routine in this module.

Usage: CALL HMGINI

Description of Parameters:

None

Remarks:

HMGINI initializes the internal table which contains an entry for each general two-level hypermatrix specified by the application program. Each two-level hypermatrix consists of one root matrix, which contains the necessary control information for the hypermatrix, as well as pointers to all the subtables with the actual data. Each entry in the internal table contains the following control information about a specified hypermatrix:

- > Identifier of the root matrix and each submatrix of the hypermatrix
- > Hypermatrix type (rectangular, symmetric, diagonal)
- > Number of words per entry in the hypermatrix
- > Number of rows in the hypermatrix
- > Number of columns in the hypermatrix
- > Number of rows in the submatrix
- > Number of columns in the submatrix
- > Number of row partitions in the hypermatrix
- > Number of column partitions in the hypermatrix

Calls: None

Called by: Application program

Commons: /COMHMG/

E.2 HTGDMP - Dumps the Variables in COMMON /COMHMG/

Purpose:

HTGDMP displays a formatted dump of all the internal variables and the data structure in COMMON /COMHMG/. It is used primarily for debugging purposes.

Usage: INTEGER NPR
 :
 CALL HMGDMP (NPR)

Description of Parameters:

NRP Logical unit specifier of file to receive the formatted
 dump

Remarks:

This routine is used for debugging purposes and may be called at any stage by the application program to determine the status of variables in COMMON /COMHMG/. It calls routine DAHEX to convert a binary integer to a string of hexadecimal characters.

Calls: DAHEX

Called by: Application program.

Commons: /COMHMG/.

E.3 HMGCRT - Creates A New Hypertable

Purpose:

HMGCRT creates a new two-level hypermatrix, that is, it allocates space in virtual memory for the root matrix and all the matrices of the specified hypermatrix. It returns the index of the appropriate entry in the internal table where all the control information for this hypermatrix are stored to the calling routine, as well as a pointer to the root matrix of this hypermatrix.

Usage: CALL HMGCRT (IDHED, MATTYP, NELTWD, IROWHM, ICOLHM, IROWSM,
 & ICOLSM, IHMPTR, NOWHMG)

Description of Parameters:

IDHED	Header for hypermatrix
MATTYP	Matrix type
NELTWD	Number of word per element
IROWHM	Total number of rows in hypermatrix
ICOLHM	Total number of columns in hypermatrix
IROWSM	Number of rows in submatrix
ICOLSM	Number of columns in submatrix
IHMPTR	Pointer to hypermatrix in dyna file
NOWHMG	Current index to control table for hypermatrices

Calls: DACRET, DAHEX, DAPUTV, INFINT, HMGSIZ

Called by: Application program

Common: /COMHMG/

E.4 HMGOPN - Opens an existing hypermatrix

Purpose:

HMGOPN opens an existing two-level hypermatrix which had previously been created by calling HMGCRT. It returns to the calling routine the index of the appropriate entry in the internal table where the control information of the hypertable is located.

Usage: CALL HMGOPN (IDHED, MATTYP, NELTWD, IROWHM, ICOLHM, IROWSM,
& ICOLSM, IHMPTR, NOWHMG)

Description of Parameters:

IDHED Header for hypermatrix
MATTYP Matrix type
NELTWD Number of words per element
IROWHM Total number of rows in hypermatrix
ICOLHM Total number of columns in hypermatrix
IROWSM Number of rows in submatrix
ICOLSM Number of columns in submatrix
IHMPTR Pointer to hypermatrix in dyna file
NOWHMG Current index in control table for hypermatrices

Calls: DACRET, DAHEX, DAPUTV, INFINT, HMGSIZ

Called by:

Common: /COMHMG/

E.5 HMGPUT - Assigns values to an entry in a hypermatrix

Purpose:

HMGPUT assigns the data values in a given matrix to a submatrix in the specified hypermatrix.

Usage: CALL HMGPUT (NOWHMG, IROWP, ICOLP, IARAY, NWDS)

Description of Parameters

NOWHMG Index in control table for hypermatrices
IROWP Row partition of submatrix in hypermatrix
ICOLP Column partition of submatrix in hypermatrix
IARAY Submatrix with values
NWDS Number of words in submatrix

Calls: DALOC, HMGNDX, INFINT, DAGETV, HMGEXT, DAHEX

Called by: Application Program

Common: /COMHMG/

E.6 HMGGET - Retrieves Values from an Entry in a Hypermatrix

Purpose:

HMGGET retrieves the data values from a given submatrix in the specified hypermatrix.

Usage: CALL HMGGET (NOWHMG, IROWP, ICOLP, IARAY, NWDS)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
IROWP Row partition of submatrix in hypermatrix
ICOLP Column partition of submatrix in hypermatrix
IARAY Submatrix with values
NWDS Number of words in submatrix

Calls: DALOC, HMCNDX, INFINT, DAGETV, HMGEXT, DANEX

Called by: Application Program

Common: /COMHMG/

E.7 HMGSIZ - Calculates number of entries for a submatrix

Purpose:

HMGSIZ calculates the dimensions of a submatrix and the number of words occupied by the submatrix.

Usage: CALL HMGSIZ (NOWHMG, IROWP, ICOLP, IROWS, ICOLS, NUMWDS)

Description of Parameters:

NOWHMG	Index in control table for hypermatrices
IROWP	Row partition of the submatrix in the hypermatrix
ICOLP	Column partition of the submatrix in the hypermatrix
IROWS	Number of rows in submatrix
ICOLS	Number of columns in submatrix
NUMWDS	Number of words occupied by submatrix

Calls: None

Called by: Application Program

Common: /COMHMG/

E.8 HMGINF - Information about a hypermatrix

Purpose:

Retrieves specified control information about a hypermatrix

Usage: INTEGER HMGINF, IVALUE
:
IVALUE = HMGINF (NOWHM, INDX)

Description of Parameters:

NOWHMG	Index in control table for hypermatrices
INDX	Indicator for which control information parameter

Calls: None

Called by: Application Program

Common: /COMHMG/

E.9 HMCNDX - Calculates index

Purpose:

Calculates the index of a submatrix in the control information area.

Usage: INTEGER HMCNDX, IVALUE
:
IVALUE = HMCNDX (NOWHMG, IROWP, ICOLP)

Description of Parameters:

NOWHMG Index in hypermatrices table
 IROWP Row partition number of submatrix
 ICOLP Column partition number of submatrix

Calls: None

Called by: HMGPOT, HMGGET, HMGEVS, HMGEVG

Common: /COMHMG/

E.10 HMGEXT - Creates a submatrix

Purpose:

Creates a submatrix for a hypermatrix and inserts the pointer to it in the control information area.

Usage: CALL HMGEXT (NOWHMG, IROWP, ICOLP, ISPTR)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROWP Row partition of the submatrix in the hypermatrix
 ICOLP Column partition of the submatrix in the hypermatrix
 ISPTR Pointer to submatrix in dynafile

Calls: HMGNDX, HMGSIZ, DACRET, DAPUTV

Called by: HMGGET, HMGPOT, HMGEVS, HMGEVG

Common: /COMHMG/

E.11 HMGDEL - Delete a hypermatrix

Purpose:

Deletes the submatrices and the control information area of a hypermatrix.

Usage: CALL HMGDEL (NOWHMG)

Description of Parameters:

NOWHMG Index in control table for hypermatrices

Calls: INFINT, HMGNDX, HMGSIZ, DAGETV, DAREL, DAULAD, DADEL

Called by: Application program

Common: /COMHMG/

E.12 HMGEVS - Store a value in a hypermatrix

Purpose:

Stores the value of an element in a hypermatrix.

Usage: CALL HMGEVS (NOWHMG, IROW, ICOL, IVALUE, NELTWD)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROW Row index of element in hypermatrix
 ICOL Column index of element in hypermatrix
 IVALUE Element value
 NELTWD Number of words in element

Calls: INFINT, DALOC, HMCNDX, HMGLOC, HMGsiz, DAGETV, HMGEXT
 DAHEX, DAPUTV

Called by: Application program

Common: /COMHMG/

E.13 HMGEVG - Retrieve a value in a hypermatrix

Purpose:

Retrieves the value of an element in the hypermatrix.

Usage: CALL HMGEVG (NOWHMG, IROW, ICOL, IVALUE, NELTWD)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROW Row index of element in hypermatrix
 ICOL Column index of element in hypermatrix
 IVALUE Element value
 NELTWD Number of words in element

Calls: INFINT, DALOC, HMCNDX, HMGLOC, HMGsiz, DAGETV, HMGEXT
 DAHEX, DAPUTV

Called by: Application program

Common: /COMHMG/

E.14 HMGLOC - Calculate element location in hypermatrix

Purpose:

HMGLOC determines in which submatrix partition a hypermatrix element is in and what the offset from the first one is.

Usage: CALL HMGLOC (NOWHMG, IROW, ICOL, IROWP, ICOLP, IOFSET)

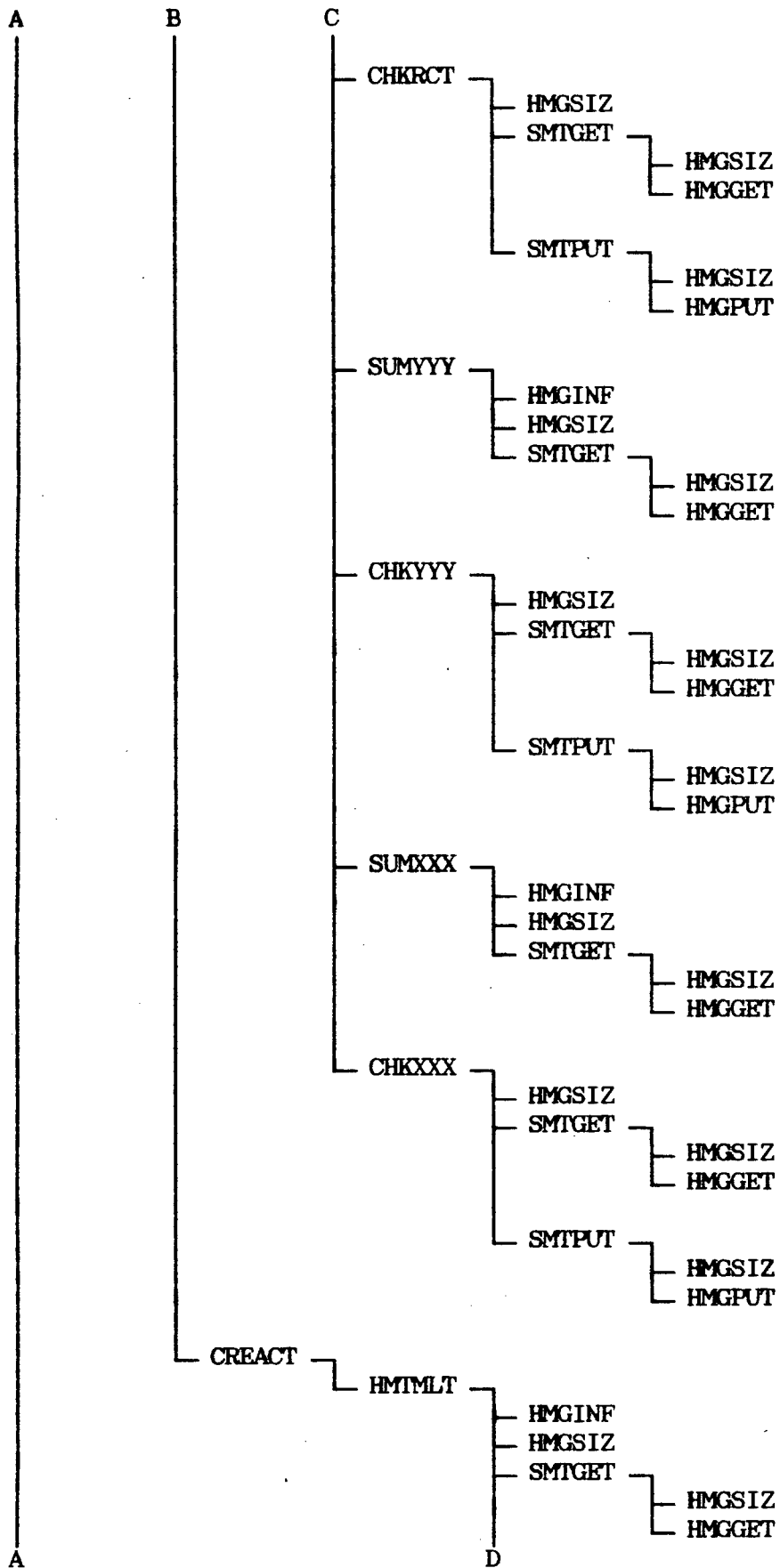
Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROW Row index of element in hypermatrix
 ICOL Column index of element in hypermatrix
 IROWP Row partition number of submatrix
 ICOLP Column partition number of submatrix
 IOFSET Number of words from first element in submatrix

Calls: HMGsiz

Called by: HMGEVS, HMGEVG

Common: /COMHMG/



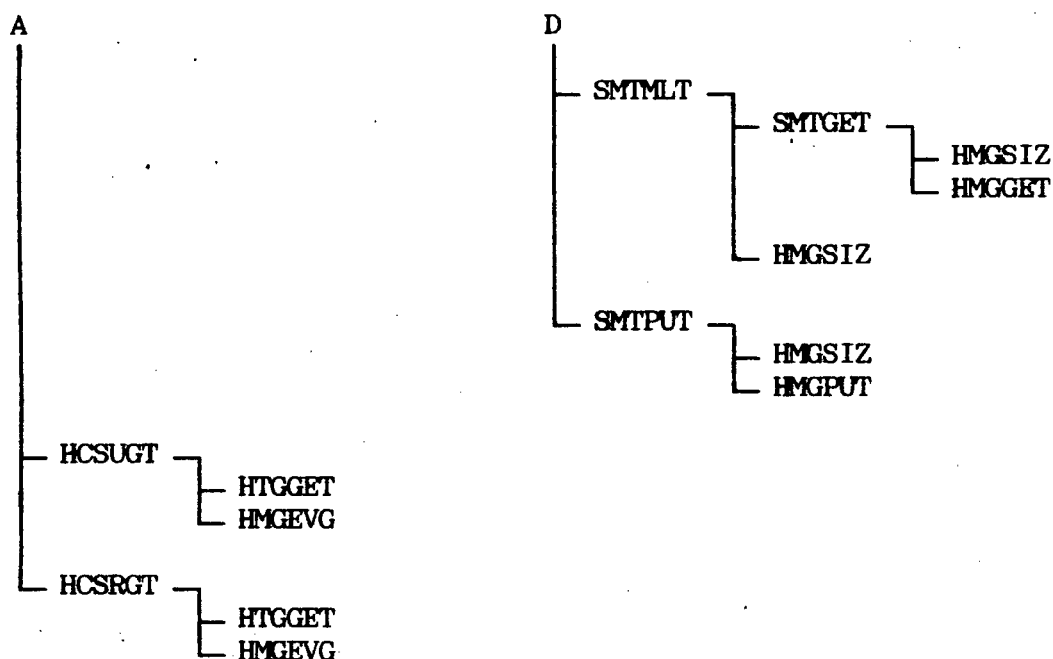


Figure 5.7: Call chart for equation solution package

5.4.3 The Chess Suite: A Sample Program

This section describes the implementation of a simple program to demonstrate how the equation assembly and solution process may be performed.

The problem considered is the one described in section 5.3.3. The free global degrees of freedom are δ_1 , δ_3 and δ_4 . The prescribed degree of freedom is $\delta_2=2$. The suppressed degree of freedom is $\delta_5=0$.

The program listing is:

```

c
c.....
c
c   Description of the program TEST:
c   -----
c
c   This program is designed to test the use of the hypermatrix
c   solver for symmetric matrices using the cholesky algorithm.
c   It uses the following test matrix (from Finite Elements in
c   Plasticity by DRJ Owen and E Hinton, page 43):
c
c   Where:
c    $K1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$   $K2 = \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix}$   $K3 = \begin{bmatrix} 3 & -3 \\ -3 & 3 \end{bmatrix}$   $K4 = \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix}$ 
c
c    $x1 = [u3 \ u1]$   $x2 = [u3 \ u2]$   $x3 = [u4 \ u3]$   $x4 = [u5 \ u4]$ 
c
c    $F1 = [0 \ 10]$   $F2 = [0 \ 0]$   $F3 = [0 \ 0]$   $F4 = [0 \ 0]$ 
c
c   With prescribed constraints:
c       u2 = 2 and u5 = 0
c
c   Displacements:
c   u1 = 13.7692 u2 = 2.000 u3 = 3.7692 u4 = 1.6154 u5 = 0.000
c   Reactions:
c   R2 = -3.5385 R5 = -6.4615
c.....
c
c   program TEST
c
c   double precision estif1(2,2),estif2(2,2),estif3(2,2),estif4(2,2),
c   *           eload1(2), eload2(2), eload3(2), eload4(2),
c   *           fixed(5), bu(5), br(5)
c   double precision HCSUGT,HCSRGT
c   integer iedof1(2),iedof2(2),iedof3(2),iedof4(2),ibcdof(5)
c
c--- element stiffnesses
c   data estif1 / 1.0d0, -1.0d0,
c   *           -1.0d0, 1.0d0/
c   data estif2 / 2.0d0, -2.0d0,
c   *           -2.0d0, 2.0d0/
c   data estif3 / 3.0d0, -3.0d0,
c   *           -3.0d0, 3.0d0/
c   data estif4 / 4.0d0, -4.0d0,
c   *           -4.0d0, 4.0d0/
c
c--- element loads
c   data eload1 / 0.0d0,10.0d0/
c   data eload2 / 0.0d0, 0.0d0/
c   data eload3 / 0.0d0, 0.0d0/
c   data eload4 / 0.0d0, 0.0d0/
c
c--- boundary condition fixities
c   data fixed / 0.0d0, 2.0d0, 0.0d0, 0.0d0, 0.0d0 /

```

```

c
c---- boundary conditions
data ibcdof / 1,2,1,1,3 /

c
c---- element incidences
data iedof1 / 3,1 /
data iedof2 / 3,2 /
data iedof3 / 4,3 /
data iedof4 / 5,4 /

c
c---- initialize hypermatrix cholesky solver module
nfree = 3
npres = 1
nsupp = 1
isymm = 1
ncase = 1
luerr = 6

c
call HCSINI (nfree,npres,nsupp,isymm,ncase,luerr)

c
c---- set up global degree of freedom table
ntotv = nfree + npres + nsupp
do 10 k = 1,ntotv
    call HCSDOF (k,ibcdof(k),fixed(k))
10 continue
call HCSDMP ('D')

c
c---- assemble global stiffness matrix
nevab = 2
call HCSABK (nevab,estif1,iedof1)
call HCSABK (nevab,estif2,iedof2)
call HCSABK (nevab,estif3,iedof3)
call HCSABK (nevab,estif4,iedof4)
call HCSDMP ('K')

c
c---- assemble global load vector
iload = 1
call HCSABR (2,eload1,iedof1,iload)
call HCSABR (2,eload2,iedof2,iload)
call HCSABR (2,eload3,iedof3,iload)
call HCSABR (2,eload4,iedof4,iload)
call HCSDMP ('R')

c
c---- solve
call HCSSOL

c
c---- get displacements and reactions and display them
lunot = 6
do 20 k = 1,ntotv
    bu(k) = HCSUGT (k,iload)
    br(k) = HCSRGT (k,iload)
20 continue

```

```

20    continue
      write(lunot,1013)
1013  format(/1h ,'Results:')
      do 30 k = 1,ntotv
          write(lunot,1003) k,bu(k),br(k)
1003  format(1h ,'Dof=',i2,' U=',F12.5,' R=',F12.5)
30    continue
c
c---  terminate
      stop ' > End of TEST'
      end
c

```

Figure 5.8: Sample Program Listing - TEST

The results are:

Kff	matrix	3 by	3
address matrix		1 by	1
submatrices		3 by	3
matrix type		2	
words per element		2	
Kfp	matrix	3 by	1
address matrix		1 by	1
submatrices		3 by	1
matrix type		2	
words per element		2	
Kpp	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Kfs	matrix	3 by	1
address matrix		1 by	1
submatrices		3 by	1
matrix type		2	
words per element		2	
Kps	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Uf	matrix	3 by	1
address matrix		1 by	1
submatrices		3 by	1
matrix type		2	
words per element		2	

Up	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Rp	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Rs	matrix	1 by	1
address matrix		1 by	1
submatrices		1 by	1
matrix type		2	
words per element		2	
Fixity	table	5 by	2

Dof	Table	Index
1	1	1
2	2	1
3	1	2
4	1	3
5	3	1

Hypermatrix Kff

> Submatrix(1, 1) :			
	1	2	3
1	1.0000	-1.0000	0.00000E+00
2	0.00000E+00	6.0000	-3.0000
3	0.00000E+00	0.00000E+00	7.0000

Hypermatrix Kfp

> Submatrix(1, 1) :	
	1
1	0.00000E+00
2	-2.0000
3	0.00000E+00

Hypermatrix Kfs

> Submatrix(1, 1) :	
	1
1	0.00000E+00
2	0.00000E+00
3	-4.0000

Hypermatrix Kpf

> Submatrix(1, 1) :	
	1
1	0.00000E+00
2	-2.0000
3	0.00000E+00

Hypermatrix Kpp

```
> Submatrix( 1, 1) :
      1
1      2.0000
```

Hypermatrix Kps

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Hypermatrix Ksf

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
2      0.00000E+00
3      -4.0000
```

Hypermatrix Ksp

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Hypermatrix Rf

```
> Submatrix( 1, 1) :
      1
1      10.000
2      0.00000E+00
3      0.00000E+00
```

Hypermatrix Rp

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Hypermatrix Rs

```
> Submatrix( 1, 1) :
      1
1      0.00000E+00
```

Results:

```
Dof= 1 U=    13.76923 R=    0.00000
Dof= 2 U=     2.00000 R=   -3.53846
Dof= 3 U=     3.76923 R=    0.00000
Dof= 4 U=     1.61538 R=    0.00000
Dof= 5 U=     0.00000 R=   -6.46154
```

Figure 5.9: Sample program output file

5.5 Discussion

The CHESS suite has been implemented and tested, on a MicroVAX II, in the NEW NOSTRUM program package. Results are compared with ABAQUS in Appendix D, and the comparison is favourable. The package should be considered as a prototype and therefore only the numerical accuracy of the results has been compared with other finite element packages. Methods of improving the efficiency, as described in section 6.3.2, still need to be implemented before a comparison of solution times may be made. The package has been implemented as a double precision version, however the changes for a single precision version are easily implemented for a different machine.

CHAPTER SIX

SUMMARY AND CONCLUSIONS

This chapter describes the software development strategy employed, as well as problems encountered and experience gained from this project. The work is summarized and conclusions drawn. Finally, suggestions for possible future work are made.

6.1 Software development strategy

6.1.1 Software design

One of the major criteria of the development of the program suite was that the software modules that were developed had to be highly modular, in order that they may be re-used in different contexts. This required that modules exhibit high functional or informational strength (that is, each module contains a strong internal relationship) and low module coupling (that is, each module has weak intermodule relationships with all the other modules in the program) [32].

The principle of information hiding was also a major design consideration for the modules (that is, modules were designed to consist of several routines which performed related functions on a particular resource, such as a common data structure) [17].

Another consideration was the design of the user interface to these modules. These had to be as simple as possible to allow the program suite to be implemented easily in any finite element program. For

example, the interface routines to the equation solution module have to provide information on the system of equations, assemble the element stiffness matrices into the global stiffness matrix, initiate the solution process and be able to retrieve results.

An important aspect of the software design and implementation was that all code be extensively documented. Code that is not documented or is poorly documented becomes 'throw-away' code and has to be redone once the original developer is no longer available for consultation. Therefore it was required that all software should be documented both internally and externally.

Internal documentation includes any explanations that are inserted directly into the code in the form of comment statements. Every module of code should have a comprehensive descriptive header block prologue which describes the module's purpose, includes routines, the developer, etc. Each routine in the module has a prologue describing the routine's purpose and its input/output variables, as well as any necessary remarks deemed necessary to make the code as easy to read as possible. Comment statements are inserted throughout the code to explain the various processing steps.

External documentation consists of a description of the algorithms implemented in the code, diagrams of the data structures, call charts, etc. Test examples must be explained with proper references. If more than one version of the code exists then the version numbers and the different capabilities of the update must be fully documented. Test examples for particular version numbers must also be kept.

6.1.2 Software Implementation

Initial software development and testing was done on the University of Cape Town's SPERRY UNIVAC 1100 computer, while the final implementation of the CHESS package was done on a MicroVAX II (in the Applied Mechanics Research Unit) and integrated into the NEW NOSTRUM development package. The testbed used during development on the UNIVAC 1100 was derived from the program PLANE [2].

Several smaller programs were coded and tested before the CHESS suite was developed.

The finite element program PLANE [2], which has an incore frontal solver, for the plane stress/strain problems was implemented and tested. Results for the given plane strain example were good, unlike the results for plane stress problems. Thus the plane strain examples were used primarily for comparison and checking further developments.

The original frontal solution code of Irons [4] was implemented and tested as this had been the basis for the Beer and Haas [6] package and it was hoped this would provide the necessary understanding of the use of variables and the underlying method in the partitioned frontal solution. Next the code published by Beer and Haas was implemented but would not run to completion. Several attempts were made to debug the code, but due to lack of sufficient documentation and the complexity of the code, it was decided not to proceed with that particular solution procedure.

A hypermatrix scheme for the storage of the frontal stiffness matrix was then implemented in the frontal solution routine of the program PLANE in order to gain some experience with the use of hypermatrices. As expected, this was not efficient due to the random accessing of submatrices and the resulting high number of I/O operations.

It was then decided to implement the Cholesky decomposition procedure with a hypermatrix data management scheme. Initially an incore solution algorithm was developed, which is available for use with problems with small numbers of degrees of freedom. The hypermatrix management scheme and the Cholesky procedure were then integrated into a single program suite. Next the program suite interface routines were developed and thoroughly tested. Finally the code was transferred to the Applied Mechanics Research Unit's MicroVAX II and integrated into the NEW NOSTRUM development code and the results compared with the ABAQUS finite element package.

6.2 Summary

During the first stage of this project an investigation of the various equation solution techniques was done. From this survey it was found that at present direct methods are favoured over iterative methods, however this may change with improving hardware technology. The various triangularisation and factorisation methods are mathematically closely related and no one method appears to have any significant computational advantage over any other method. The advantage of one solution scheme over the other is related to the data handling scheme.

Efficiency may be achieved by exploiting the sparsity of the system of equations which results in a minimum number of I/O operations.

For a robust and reliable code which is easy to upgrade the solution technique needs to be as modular and straight-forward as possible. Firstly a data management scheme should provide the first layer of software to store and retrieve the equations. The solution scheme should be possible through a series of procedures which are controlled by high level routines and performed by lower level routines. A set of interface routines to the equation solution scheme needs to be implemented which enables any user to simply initiate the few processes (such as initialisation, assembly, solution and result retrieval) without having any knowledge of the solution method itself.

The second stage of the project involved the detailed investigation and implementation of an in-core wavefront solver with the intention of using a partitioning scheme to remove the problem size limitations and to provide some more enhanced features. It was found that the data management of the frontal solution scheme was so closely enmeshed with the solution algorithm and that the partitioning scheme would not provide the desired level of modularity.

The hypermatrix scheme was developed which provided the basic data handling capability for a simpler and more modular solution technique. General hypermatrix and hypertable routines provided the interface to the DAS package which implements a virtual memory paging scheme. Hypermatrix procedures were developed as a module for the assembly, solution and retrieval of results. Finally a module of interface routines was provided for the easy application of the equation solution scheme in finite element programs.

6.3 Suggestions For Further Improvements

6.3.1 Unsymmetric matrices

The equation solution suite presented already has many of the facilities required for the solution of a system of unsymmetric matrices except for the solution of the $x = K^{-1}F$ where K is unsymmetric. This limitation may be removed if the Cholesky algorithm, for symmetric matrices, is complemented with the more general Crout's algorithm, for unsymmetric matrices.

An implementation of the Crout algorithm as presented by Taylor[18] has been implemented as an incore solver. Taylor has chosen to use the profile scheme for data management, but an implementation using the existing hypermatrix scheme based on the Cholesky algorithm would be simpler.

6.3.2 Further means of improving efficiency

Several methods of speeding up calculation times, by the appropriate avoidance of zero multiplications within the operations of submatrices, are described by Schrem [12]. In order to avoid wasteful tests on zeroes within the innermost loops, the sparsity structure of the various submatrices must be related to the required matrix operations, particularly during the triangularization. The five necessary operations on submatrices during the equation solution, observed from eqs. (5.8) to (5.11), can be examined as to special action to exploit sparsity:

1. The matrix multiplication $R = R - A^T B$ occurs in eqs. (5.8), (5.9) and (5.10), and in problems with relatively large bandwidths usually accounts for more than 80% of the total numerical operations during the equation solution. Submatrix B always lies further to the outside of the "band" than submatrix A, and will thus have the greater likelihood of being only partially filled. During the forward-substitution, matrix B (i.e. matrix Y_{KJ} in eq. (5.10)) can also contain many zeroes if the loading conditions are differently distributed. Thus the majority of the zero operations can be detected if the columns of matrix B are tested for zero values. In addition, because the submatrices in both the triangularization and forward-substitution always tend to fill up in a downward direction, the zero test can be reasonably terminated upon the occurrence of the first non-zero value in a column of B.

2. The Cholesky triangularization operation $U_{II}^T U_{II} = K_{II}^*$ occurs in eq. (5.8) on the diagonal submatrices. As the diagonal submatrices are nearly always full for larger problems, tests on zeroes are not justified for this procedure.

3. The forward-substitution operation $U_{II}^T X = A$ occurs in both eqs. (5.9) and (5.10). Although matrix U_{II} is normally full, the result matrix X may sometimes be only partially full, especially near the outside of the band, and also fills up in a downward direction. If the outer loop of this operation is made over the number of columns in X, then a simple downward test to detect the first non-zero element in each column of X shortens the two inner loops of the operation correspondingly.

4. The multiplication operation $R = R - AB$ occurs in the back-substitution procedure eq. (5.11). Although the "result" matrix B, i.e. x_{KJ} in eq (5.11) is always full, the pre-multiplier A may only be partly filled near the edge of the band. In this case rows of A are involved. The innermost loop can be completely skipped upon the identification in the next outermost loop of individual zero values in row i of matrix A. Although the accumulation feature into the result matrix R is thereby lost, the net savings are usually significant, especially when several loading conditions are present.
5. For the back-substitution operation $U_{II}x_{IJ} = x_{IJ}^*$ in eq (5.11) no tests should be made, because x_{IJ} is always full and U_{II} also usually full.

The execution speed may be improved by writing some of the low level routines and I/O routines in machine code, however, as a previously mentioned this would complicate the code and reduce its portability. The equivalent FORTRAN routine should always be supplied for initial installation and the machine code routines programmed by the systems programmer.

6.3.3 Renumbering Algorithms

One of the main difficulties in practical application of optimal ordering algorithms is that in the very large problems where such a procedure should be most beneficial, the cost of the re-ordering process may exceed the resulting savings, especially when ordering is performed on single rows and columns [12]. However, re-ordering

algorithms working on the address matrix of the structural stiffness matrix may produce significant savings. This address matrix, which provides a macro representation of the sparsity, is rarely larger than 200 by 200, even for large problems. The cost of the re-ordering process itself takes only 1% to 2% of the triangularization time [12].

6.3.4 Concurrent Processing

Methods performing concurrent Cholesky factorisation of positive-definite matrices have been developed [9]. The technology of concurrent processing is relatively new but appears to be the direction of future developments. This new technology should be exploited as far as possible in the development and implementation of any new code in the future.

CHAPTER SEVEN

REFERENCES

- [1] Wilkinson, J.H., 'The Algebraic Eigenvalue Problem', Clarendon Press, Oxford, England, 1965.
- [2] Hinton, E., Owen, D.R.J., 'Finite Element Programming', Academic Press Inc., Swansea, England, 1977.
- [3] Lindgren, L., 'Design of Small and Effective Finite Element Systems', Lulea University, Sweden, Eng. Computing, 3, pp 202-208, September, 1986.
- [4] Irons, B., 'A Frontal Solution Program for Finite Element Analysis', Uni. Wales, Swansea, Int. J. Num. Meth. Eng., 2, pp 5-32, 1970.
- [5] Owen, D.R.J., Hinton, E., 'Finite Elements in Plasticity. Theory and Practice', Pineridge Press Ltd., Swansea, U.K., 1977.
- [6] Beer, G., Haas, W., 'A Partitioned Frontal Solver for Finite Element Analysis', Int. J. Num. Meth. Eng., 18, pp 1623-1654, 1982.
- [7] Irons, B., Ahmad, S., 'Techniques of Finite Elements', John Wiley & Sons, Chichester, 1986.
- [8] Bettess, J.A., 'A Data Structure for Finite Element Analysis', Int. J. Num. Meth. Eng., 11, pp 1779-1799, 1977.

- [9] Utkar, S., Salama, M., Melosh, R.J., 'Concurrent Factorization of Positive Definite Banded Hermitian Matrices', Int. J. Num. Meth. Eng., 23, pp 2137-2152, 1986.
- [10] George, A., 'Sparse Matrix Methods of the Finite Element Method', Dept. Comp. Sci., Uni. Waterloo, Waterloo, Ontario, Canada.
- [11] Melosh, R.J., Bamford, R.M., 'Efficient Solution of Load-deflection Equations', J. Struct. Div. ASCE, 95, pp 661-676, 1969.
- [12] Fuchs, G.v., Roy, J.R., Schrem, E., 'Hypermatrix Solution of the Stiffness Equations', ISD ASKA Report PM 109, Institut fur Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, University of Stuttgart, Stuttgart, 1972.
- [13] Felippa, A.C., 'Solution of Linear Equations with Skyline-stored Symmetric Matrix', Comp. & Struct., 5, pp 13-29, 1975.
- [14] Agrawal, A.B., Mufti, A.A., Jaeger, L.G., 'Band-schemes vs. Frontal-routines in Nonlinear Structural Analysis', Int. J. Num. Meth. Eng., 15, pp 753-766, 1980.
- [15] Kreyszig, E., 'Advanced Engineering Mathematics', John Wiley and Sons, New York, 4th edition pp 805-809, 1979.
- [16] Vos, J., 'Struktuur Leer', 4th Year Engineering Course Notes, University of Stellenbosch, pp 1.22-1.27.

- [17] Vos, J., 'Implementing a Finite Element Program on a Microcomputer using a Dynamic Memory Allocation Scheme', Master's Thesis, Uni. Stellenbosch, 1985.
- [18] Taylor, R.L., 'Solution of Equations by a Profile Solver', Eng. Comput., 2, pp 344-350, 1985.
- [19] George, A., Liu, J.W.H., 'The Design of a User Interface for a Sparse Matrix Package', ACM Transactions on Mathematical Software, 5, no.2, pp 139-152, 1979.
- [20] Servranckx, D., Mufti, A.A., 'Data Structures for Finite Element Modelling', Eng.Comput., 3, pp 27-35, 1986.
- [21] Dhatt, G., Touzot, G., 'The Finite Element Displayed', John Wiley & Sons, 1984.
- [22] Zienkiewicz, O.C., 'The Finite Element Method', 3rd Edt., pp 677-757.
- [23] Hood, P., 'Frontal Solution Program for Unsymmetric Matrices', Int. J. Num. Meth. Eng., 10, pp 379-399, 1976.
- [24] Cuthill, E., McKee, J., 'Reducing the Bandwidth of Sparse Symmetric Matrices', Proc. 24th Nat. Conf. Assoc. Comput. Mach., ACM Publ., pp 69, 1969.
- [25] Wilson, E.L., Bathe, K.L., Doherty, W.P., 'Direct Solution of Large Systems of Linear Equations', Comp. and Struct., 4, pp 363-372, 1974.

- [26] Jennings, A., 'A Compact Storage Scheme for the Solution of Symmetric Simultaneous Equations', *Comput. J.*, 9, pp 281-285, 1966.
- [27] Pina, H.L.G., 'An Algorithm for Frontwidth Reduction', *Int. J. Num. Meth. Eng.*, 17, pp 1539-1546, 1981.
- [28] Alizadeh, A., Will, G.T., 'A Substructured Frontal Solver and its Application to Localized Material Non-linearity', *Comp. Struct.*, 10, pp 225-231, 1979.
- [30] Abbas, S.F., 'Some Novel Applications of the Frontal Concept', *Int. J. Num. Meth. Eng.*, 15, pp 519-536, 1980.
- [31] Bell, K., Hatlestad, B., Hansteen, O.E., Araldsen, P.O., 'NORSAM-Users Manual', February 1973.
- [32] Myers, G.J., *Composite/Structural Design*, Van Nostrand Reinhold, New York, 1978.
- [33] Kamel, H.A., McCabe, M.W., 'Direct Numerical Solution of Large Sets of Simultaneous Equations', *Computers and Structures*, 9, pp 113 - 123, 1978.
- [34] Rice, J.R., 'Matrices and Linear Equations'. In *Numerical Methods, Software and Analysis*, McGraw-Hill, New York, 1983.
- [35] Lestingi, J., Prachuktam, S., 'A Blocking Technique for Large Scale Structural Analysis', *Computers and Structures*, 3, pp 669-714, 1973.

- [36] Schrem, E., 'Computer Implementation of the Finite Element Procedure'.
In Numerical and Computer Methods in Structural Mechanics (Edited by
S.J. Fenves, N. Peronne, A.R. Robinson and W.C. Schnobrich). Proc. ONR
- Conf. 1971, pp 79-121. Academic Press, New York, 1973.

APPENDIX A

COURSEWORK PASSED

To comply with the requirements for the degree of Master of Science in Engineering, approved coursework of no less than twenty credits has to be passed. The courses attended are briefly described below.

A.1 AM 363 - NUMERICAL ANALYSIS

Introduction to the theory and practice of numerical approximation. Interpolation and approximation. Numerical integration. Numerical solution of non-linear equations and of initial value problems. Accuracy of numerical solutions.

(3 credits)

A.2 CE 5B6 - FRAME ANALYSIS

The application of the force method of analysis to framed structures of straight and curved members. The stability of equilibrium of framed structures.

(2 credits)

A.3 CE 5B16 - COMPUTER AIDED DESIGN OF STRUCTURES

The modelling of structures using finite element analysis methods. The application of computer aided design (CAD) techniques to finite element data preparation. Computer programming of the finite element method.

APPENDIX B

CHOLESKY HYPERMATRIX EQUATION SOLUTION ROUTINES

In this appendix the FORTRAN routines in the Cholesky Hypermatrix Equation Solution Scheme suite are described in detail. The routines access a two-level hypermatrix scheme which call routines in a virtual memory management scheme. The hypermatrix routines are described in appendix E. The FORTRAN source code for the routines is contained in appendix C.

B.1 Cholesky Hypermatrix Equation Solution Scheme Interface Routines

B.1.1 HCSINI - Initializes the Equation Solution Procedure

Purpose:

HCSINI initializes the equation solution procedure by performing three tasks. Firstly the size of the sub-matrices to be used for the equation solution is calculated, secondly the required hypermatrices are created and thirdly the hypermatrix pointers are stored in the hypermatrix pointer table.

Usage: CALL HCSINI (NFREE, NPRES, NSUPP, IFSYM, NCASE, LUNERR)

Description of Parameters:

Input:

NFREE	Number of free global degrees of freedom
NPRES	Number of prescribed global degrees of freedom
NSUPP	Number of suppressed global degrees of freedom
IFSYM	Flag (0 or 1) to indicate non-symmetric or symmetric problem
NCASE	Number of load cases
LUNERR	Logical unit number for printing any equation solution error message

Output:

None

Remarks:

The input parameters are store in a common block /HCSCOM/ which is common only to the module interface of routines. The hypermatrix pointers are also stored in this common block. A separate common block is initialized which holds the counters for the global degrees of freedom in the hypertable.

Calls: MATSIZ, HYPINI, INDEXS

Called by: User application program

Commons: /HCSCOM/
/DOFCOM/

B.1.2 HCSDOF - Specifies a Tuple in the Global Degree of Freedom Hypertable

Purpose:

The two values of a tuple in the global degree of freedom hypertable are calculated and stored in the hypertable.

Usage: CALL HCSDOF (ITOTV, IFFPS, FIXED)

Description of parameters

Input:

ITOTV	Global degree of freedom number
IFFPS	Flag (1, 2, 3) to indicate free, prescribed or suppressed degree of freedom
FIXED	Value of prescribed displacement

Output:

None

Remarks:

Checks are made on the validity of the type of degree of freedom indicator. The value of the prescribed displacement is stored in the prescribed displacement hypermatrix.

Calls: HMGINF, HMGEVS, HMGPOT

Called by: User application program

Commons: /HCSCOM/
/DOFCOM/

B.1.3 HCSCHK - Check Global Degree of Freedom Hypertable

Purpose:

Routine performs a check to see that all the degrees of freedom in the hypertable have been specified. If a degree of freedom has not been specified then it by default set to be free.

Usage: CALL HCSCHK

Description of parameters:

None:

Remarks:

The use of this routine is optional. This routine may be used for the case when the user wishes to only define the prescribed and suppressed degrees of freedom and then may call this routine to set all the remaining degrees of freedom to be free by default.

Calls: HMGINF, HMGPOT, INFINT

Called by: User application program

Common: /HCSCOM/
/DOFCOM/

B.1.4 HCSABK - Assemble Element Stiffness Matrix into Global Stiffness Hypermatrices

Purpose:

Assembles an element stiffness matrix into the correct global stiffness hypermatrices.

Usage: CALL HCSABK (NEVAB, ESTIF, ELDOF)

Description of parameters:

Input:

NEVAB	Number of degrees of freedom in element
ESTIF	Element stiffness matrix
ELDOF	Global degree of freedom numbers for element degrees of freedom

Output:

None

Remarks:

If the problem is symmetric then the element stiffness matrix need only be upper triangular, however it will not matter if the element matrix is full.

Calls: HMGINF, HMGGET, HMGEVG, HMGEVS

Called by: User application program

Common: /HCSOOM/

B.1.5 HCSABR - Assemble Element Load Vector into Global Load Hypermatrices

Purpose:

Assembles an element load vector into the correct global load hypermatrix.

Usage: CALL HCSABR (NEVAB, ELOAD, ELDOF, ILOAD)

Description of parameters:

Input:

NEVAB	Number of degrees of freedom in element
ELOAD	Element load vector
ELDOF	Global degree of freedom numbers for element degrees of freedom
ILOAD	Load case number

Output:

None

Remarks:

If the global degree of freedom is free then the value is added to the global load vector, however if the degree of freedom is prescribed or suppressed then the value is subtracted from the global load vector in preparation for the calculation of reactions with eqs. (6.21) or (6.22).

Calls: HMGINF, HMGGET, HMGEVS
 Called by: User application program
 Common: /HCSCOM/

B.1.6 HCSSL - Solution of System of Equations

Purpose:

Controls the three steps of the solution of the system of equations. Firstly the load vector is adjusted for the prescribed degrees of freedom, secondly the Cholesky procedure is used to solve for the inverse multiplication in eq. (6.20) and the calculation of reactions is done from eqs. (6.21) and (6.22).

Usage: CALL HCSSL

Description of parameters:

None

Remarks:

The three sections of this procedure are each performed by separate routines.

Calls: PRESCS, CHLSKY, CREAT

Called by: User application program

Common: /HCSCOM/

B.1.7 HCSUGT - Retrieve Displacement

Purpose:

Retrieves a single displacement for given global degree of freedom and load case number.

Usage: DOUBLE PRECISION HCSUGT, AVALUE
 :
 AVALUE = HCSUGT (ITOTV, ILOAD)

Description of parameters

Input:

ITOTV Global degree of freedom number
 ILOAD Load case number

Output:

None

Remarks:

This is a double precision function. If an attempt is made to retrieve a value from a hypermatrix that does not exist then a value of 0.0d0 is returned. This would be the case of trying to retrieve a value from the hypermatrix of suppressed degrees of freedom which is by default not created.

Calls: HTGGET, HMGINF, HMGEVG
 Called by: User application program
 Common: /HCSCOM/

B.1.8 HCSRGT - Retrieve Reaction

Purpose:
 Retrieves a single resultant reaction for a given global degree of freedom and load case number.

Usage: DOUBLE PRECISION HCSRGT, AVALUE
 :
 AVALUE = HCSRGT (ITOTV, ILOAD)

Description of Parameters

Input:
 ITOTV Global degree of freedom number
 ILOAD Load case number

Output:
 None

Remarks:
 This is a double precision function. If an attempt is made to retrieve a value from a hypermatrix that does not exist then a value of 0.0d0 is returned. This would be the case of trying to retrieve a value from the hypermatrix of free degrees of freedom, in which case there is no reaction.

Calls: HTGGET, HMGINF, HMGEVG
 Called by: User application program
 Common: /HCSCOM/

B.2 Cholesky Hypermatrix Equation Solution Scheme Routines

B.2.1 CHLSKY - Controlling routine for Cholesky Procedure

Purpose:
 CHLSKY controls, at partitioning level (level 2), the flow of the Cholesky algorithm through the decomposition, forward-substitution and back substitution stages.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL CHLSKY(GSTNDX, GLDNDX)

Description of parameters:

Input:-
 GSTNDX Index to K_{ff} hypermatrix (stiffness matrix)
 GLDNDX Index to F_f hypermatrix (load matrix)

Output:-

GLDNDX Index to U_f hypermatrix (result matrix)

Remarks:-

The stiffness matrix is assumed to be symmetric positive-definite and prescribed and suppressed degrees of freedom have been condensed out. The result matrix will overwrite the load matrix.

Calls: HMGINF, SUMTRI, CHKTRI, SUMRCT, SUMYYY, CHKYYY, SUMXXX, CHKXXX.

Called by: HCSSOL

Commons: none

B.2.2 CHKTRI - Cholesky Decomposition for Diagonal Submatrix

Purpose:

Performs Cholesky decomposition on a submatrix on the diagonal of the partitioned hypermatrix.

Usage: INTEGER GSTNDX
 :
 CALL CHKTRI(GSTNDX, IROWH, SUMMA, PIVOT)

Description of Parameters:

Input:-

GSTNDX Index to K_{ff} hypermatrix (global stiffness matrix)
IROWH Partition row number (also column number because pivotal partition)
SUMMA Summation for this partitioned sub-matrix

Output:-

GSTNDX Index to hypermatrix U
PIVOT Pivot vector for this row partition

Remarks:-

Summation is updated as the solution proceeds through the rows and columns

Calls: HMGISZ, SMTGET, SMTPUT

Called by: CHLSKY

Commons: /WORK/

B.2.3 CHKRCT - Cholesky Decomposition for Non-Diagonal Submatrix

Purpose:

Performs Cholesky decomposition on submatrices which do not lie on the diagonal of the partitioned hypermatrix.

Usage: INTEGER GSTNDX
 :
 CALL CHKRCT (GSTNDX, IROWH, ICOLH, SUMMA, PIVOT)

Description of Parameters:

Input:
 GSTNDX Index to K_{ff} hypermatrix (stiffness matrix)
 IROWH Partition row number
 ICOLH Partition column number
 SUMMA Summation up until this partitioned sub-matrix
 PIVOT Pivot vector for this row partition

Output:

Summation is updated as the solution proceeds through the rows and columns of the submatrix.

Calls: FMGSIZ, SMIGET, SMIPUT

Called by: CHLSKY

Commons: /WORK/

B.2.4 CHKYYY - Cholesky Forward-Substitution for a Submatrix

Purpose:

Performs Cholesky "forward-substitution" for a submatrix.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL CHKYYY (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 GLDNDX Index to load matrix
 IROWH Partition row number in load hypermatrix
 ICOLI Partition column number in load hypermatrix
 SUMMA Summation for current partition submatrix (IROWH, ICOLI)

Output:

GLDNDX Index to hypermatrix Y

Remarks:

Summation is updated as the solution proceeds through the rows and columns.

Calls: FMGSIZ, SMIGET, SMIPUT

Called by: CHLSKY

Commons: /WORK/

B.2.5 CHKXXX - Cholesky Back-Substitution of a Submatrix

Purpose:

Performs Cholesky "back-substitution" on a submatrix.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL CHKXXX (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:

GSTNDX Index to hypermatrix U
 GLDNDX Index to hypermatrix Y
 IROWH Partition row number in Y hypermatrix
 ICOLI Partition column number in Y hypermatrix
 SUMMA Summation for current submatrix (IROWH, ICOLI)

Output:

GLDNDX Index to result hypermatrix X

Remarks:

Summation is updated as the solution proceeds through the rows and columns.

Calls: HMGSIZ, SMTGET, SMTPUT

Called by: CHLSKY

Commons: /WORK/

B.2.6 SUMTRI - Calculates Summation for the Decomposition of a Diagonal Submatrix

Purpose:

Calculates the summation for the decomposition of a diagonal submatrix.

Usage:

INTEGER GSTNDX
 :
 CALL SUMTRI (GSTNDX, IROWH, SUMMA)

Description of Parameters:

Input:

GSTNDX Index to hypermatrix U
 IROWH Partition row number (also column number because pivotal partition) in hypermatrix U

Output:

SUMMA Summation up to partition (IROWH, IROWH)

Remarks:

Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.7 SUMRCT - Calculates Summation for the Decomposition of a Non-Diagonal Submatrix

Purpose:

Calculates the summation for the decomposition of a non-diagonal hypermatrix.

Usage: INTEGER GSTNDX
 :
 CALL SUMRCT (GSTNDX, IROWH, ICOLH, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 IROWH Partition row number in hypermatrix U
 ICOLH Partition column number in hypermatrix U

Output:
 SUMMA Summation up to partition (IROWH, ICOLH)

Remarks:
 Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.8 SUMYYY - Calculates Summation for the Forward Substitution of a Submatrix

Purpose:
 Calculates summation for a submatrix in hypermatrix Y.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL SUMYYY (GSTNDX, GLDNDX, IROWH, ICOLI, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 GLDNDX Index to hypermatrix Y
 IROWH Partition row number in hypermatrix Y
 ICOLI Partition column number in hypermatrix Y

Output:
 SUMMA Summation up to partition (IROWH, ICOLI)

Remarks:
 Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMGSIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.9 SUMXXX - Calculates Summation for the Back Substitution of a Submatrix

Purpose:
 Calculates summation for a submatrix in hypermatrix X.

Usage: INTEGER GSTNDX, GLDNDX
 :
 CALL SUMXX (GSTNDX, GLDNDX, IROWH, ICO LI, SUMMA)

Description of Parameters:

Input:
 GSTNDX Index to hypermatrix U
 GLDNDX Index to hypermatrix Y
 IROWH Partition row number in hypermatrix Y
 ICO LI Partition column number in hypermatrix Y

Output:
 SUMMA Summation up to partition (IROWH, ICO LI)

Remarks:
 Matrix SUMMA must be zeroed before start of summation

Calls: HMGINF, HMG SIZ, SMTGET

Called by: CHLSKY

Commons: /WORK/

B.2.10 HYTIMLT - Hypermatrix Multiplication Routine

Purpose:
 Multiply two hypermatrices together and either adds or subtracts the result to the resultant hypermatrix.

Usage: INTEGER TRANA, TRANB
 :
 CALL HYTIMLT (NDXHMA, NDXHMB, NDXHMC, FACTR, TRANA, TRANB, ISYMA, ISYMB)

Description of Parameters:

Input:
 NDXHMA Index to hypermatrix A
 NDXHMB Index to hypermatrix B
 NDXHMC Index to hypermatrix C
 FACTR A factor (+1.0 or -1.0) to indicate addition or subtraction
 TRANA Flag (0 or 1) to indicate do not or do transpose A before multiplication
 TRANB Flag (0 or 1) to indicate do not or do transpose B before multiplication
 ISYMA Flag (0 or 1) to indicate if A is unsymmetric or symmetric
 ISYMB Flag (0 or 1) to indicate if B is unsymmetric or symmetric

Output:
 NDXHMC Index to resultant hypermatrix C

Remarks:
 Only the index to the hypermatrices and whether they are to be transpose and if they are symmetric need be passed to the routine, the remaining hypermatrix information may be retrieved by the routine itself. Checks are done on the validity of the multiplication by checking the compatibility of the hypermatrices.

Calls: HMGINF, HMGSIZ, SMTGET, SMTMLT, SMTPUT

Called by: Application program

Commons: None

B.2.11 SMTMLT - Submatrix Multiplication Routine

Purpose

Multiplies two submatrices together and either adds or subtracts the result from the resultant submatrix.

Usage: INTEGER TRANSA, TRANS

```

      :
      CALL SMTMLT (NDXHMA, IROWA, ICOLA,
                   NDXHMB, IROWB, ICOLB,
                   ARAYC, NROWSC, NCOLSC,
                   FACTR, TRANSA, TRANSB,
                   ISYMA, ISYMB)

```

Description of Parameters:

Input:

NDXHMA	Index to hypermatrix A
IROWA	Partition row number in hypermatrix A
ICOLA	Partition column number in hypermatrix A
NDXHMB	Index to Hypermatrix B
IROWB	Partition row number in hypermatrix B
ICOLB	Partition column number in hypermatrix B
ARAYC	The result submatrix C
NROWSC	Number of rows in submatrix C
NCOLSC	Number of columns in submatrix C
FACTR	A factor (+1.0 or -1.0) to indicate addition or subtraction
TRANSA	Flag (0 or 1) to indicate do not or do transpose A before multiplication
TRANSB	Flag (0 or 1) to indicate do not or do transpose B before multiplication

Remarks

This routine has subsequently been split into several separate routines so that the checks for symmetry and transposition may be checked at a higher level and then the relevant routine called which then does not need to perform any checks before multiplication. These routines could be written in assembler language to improve efficiency.

Calls: SMTGET, HMGSIZ

Called by: HMTMLT

Common: /WORK/

B.2.12 SMTPUT - Store Submatrix in Database

Purpose:

SMTPUT stores a submatrix in the database

Usage: CALL SMTPUT (NDXHMG, IROW, ICOL, ARRAY)

Description of Parameters:

Input:

NDXHMG	Index to hypermatrix
IROW	Partition row number of submatrix
ICOL	Partition column number of submatrix
ARRAY	Submatrix to be stored

Output:

None

Remarks:

Before the submatrix may be stored a check is done to see if the actual number of rows in the submatrix corresponds to the number of rows dimensioned for the submatrix. If the row dimensions do not correspond a copy is done with the matrix to ensure values are correctly stored. The column dimensions do not need to be checked because matrices are stored column-wise in FORTRAN.

Calls: HMGSIZ, HMGPOT

Called by: General routines

Commons: None

B.2.13 SMTGET - Retrieve Submatrix from Database

Purpose:

SMTGET retrieves a submatrix from the database

Usage: CALL SMTGET (NDXHMG, IROW, ICOL, ARRAY)

Description of Parameters:

Input:

NDXHMG	Index to hypermatrix
IROW	Partition row number of submatrix
ICOL	Partition column number of submatrix

Output:

ARRAY Submatrix retrieved from database

Remarks:

After the submatrix has been retrieved a check is done to see if the actual number of rows in the submatrix corresponds to the number of rows dimensioned for the submatrix. If the row dimensions do not correspond a copy is done with the submatrix to ensure values are correctly retrieved. The column dimensions do not need to be checked because matrices are stored column-wise in FORTRAN.

Calls: HMGSIZ, HMGET

Called by: General routines

Commons: None

B.2.14 CREAT - Calculate Reactions

Purpose:

CREACT controls the calculation of the reactions at prescribed and suppressed degrees of freedom.

Usage: CALL CREAT (NDXRP, NDXKPF, NDXUF, NDXKPP, NDXUP,
NDXRS, NDXKSF, NDXKSP, IFSYM)

Description of Parameters:

Input:

NDXRP Index to R_p hypermatrix
 NDXKPF Index to K_{pf} hypermatrix
 NDXUF Index to KU_f hypermatrix
 NDXKPP Index to K_{pp} hypermatrix
 NDXUP Index to U_p hypermatrix
 NDXRS Index to R_s hypermatrix
 NDXKSF Index to K_{sf} hypermatrix
 IFSYM Flag (0 or 1) to indicate non-symmetric or symmetric problem.

Output:

None

Remarks:

Reactions are calculated by performing hypermatrix multiplication. The reactions are solved from

$$R_s = -F_s + K_{sf} \cdot U_f + K_{sp} \cdot U_p$$

and

$$R_s = -F_s + K_{sf} \cdot U_f + K_{sp} \cdot U_p$$

Calls: HMTMLT

Called by: HCSSL

Common: None

B.2.15 PGMINI - Initialize Equation Solution Modules

Purpose:

PGMINI calls all other initialization routines to initialize the internal variables of the modules.

Usage: CALL PGMINI

Description of Parameters

None

Remarks:

There may only be one PGMINI routine in the program. If the program uses any DAS routines within other parts of the program then this routine should be called by the user at the start of the program.

Calls: LUNINI, BASINI, HTGINI, HMGINI

Called by: HCSINI (or user program)

Commons: None

B.2.16 MATSIZ - Calculate Size of Submatrices for this Solution**Purpose:**

MATSIZ calculates the size of the submatrices that will be used for this solution.

Usage: CALL MATSIZ (NFREE, NPRES, NSUPP, NSIZS)

Description of Parameters:**Input:**

NFREE	Number of free degrees of freedom
NPRES	Number of prescribed degrees of freedom
NSUPP	Number of suppressed degrees of freedom

Output:

NSIZS	Size of submatrices
-------	---------------------

Remarks:

The submatrix size is calculated from the number of free degrees of freedom, the page size of the DYNA-file and a maximum optimal size.

Calls: HMGINF, DAINFI

Called by: HCSINI

Commons: None

B.2.17 DAFCRT - Create and Format DYNA-File**Purpose**

Create and format a DYNA-file for use as the database file

Usage: CALL DAFCRT

Description of Parameters:

None

Remarks:

If the DYNA-file is used by other parts of the program then DAFCRT should be called by the user program.

Calls: DAS package

Called by: HCSINI (or user program)

Common: None

B.2.18 HYPINI - Initialize Hypermatrices and Hypertable

Purpose:

HYPINI initializes all the hypermatrices for the global stiffness matrix, global load vector and result vectore. HYPINI also initializes the global degree of freedom hypertable.

Usage: CALL HYPINI (NDXKFF, NDXKFP, NDXKFS, NDXUF, NDXRF,
 NDXKPF, NDXDPP, NDXKPS, NDXUP, NDXRP,
 NDXKSF, NDXKSP, NDXKSS, NDXUS, NDXRS,
 NDXFIX, NFREE, NPRES, NSUPP, NLOAD
 NSIZS, IFSYM)

Description of Parameters

Input:

NFREE	Number of free degrees of freedom
NPRES	Number of prescribed degrees of freedom
NSUPP	Number of suppressed degrees of freedom
NLOAD	Number of load cases
NSIZS	Dimension of submatrices
IFSYM	Flag (0 or 1) to indicate non-symmetry or symmetry

Output:

NDXKFF	Index to K_{ff} hypermatrix
NDXKFP	Index to K_{fp} hypermatrix
NDXKFS	Index to K_{fs} hypermatrix
NDXUF	Index to U_f hypermatrix
NDXRF	Index to R_f hypermatrix
NDXKPF	Index to K_{pf} hypermatrix
NDXDPP	Index to K_{pp} hypermatrix
NDXKPS	Index to K_{ps} hypermatrix
NDXUP	Index to U_p hypermatrix
NDXRP	Index to R_p hypermatrix
NDXKSF	Index to K_{sf} hypermatrix
NDXKSP	Index to K_{sp} hypermatrix
NDXKSS	Index to K_{ss} hypermatrix
NDXUS	Index to U_s hypermatrix
NDXRS,	Index to K_s hypermatrix
NDXFIX	Index to fixity hypertable

Remarks:

Not all the hypermatrices are created because some of them may be zero or will not be used.

Calls: HMGCRT, HMGINF, HTGCRT

Called by: HCSINI

COMMON: None

B.2.19 INDEXS - Initialize Pointer Look-up Tables

Purpose:

Initializes the pointer look-up tables for the global stiffness matrix, global load vector and displacement vector.

Usage: CALL INDEXS (INDEXK, INDEXU, INDEXR, IFSYM,
 NDXKFF, NDXKFP, NDXKFS, NDXUF, NDXRF,
 NDXKPF, NDXDPP, NDXKPS, NDXUP, NDXRP,
 NDXKSF, NDXKSP, NDXKSS, NDXUS, NDXRS)

Description of parameters

INDEXK	Stiffness pointer table
INDEXU	Displacements pointer table
INDEXR	Loads pointer table
IFSYM	Flag (0 or 1) for non-symmetry of symmetry
NDXKFF	Index to K_{ff} hypermatrix
NDXKFP	Index to K_{fp} hypermatrix
NDXKFS	Index to K_{fs} hypermatrix
NDXUF	Index to U_f hypermatrix
NDXRF	Index to R_f hypermatrix
NDXKPF	Index to K_{pf} hypermatrix
NDXDPP	Index to K_{pp} hypermatrix
NDXKPS	Index to K_{ps} hypermatrix
NDXUP	Index to U_p hypermatrix
NDXRP	Index to R_p hypermatrix
NDXKSF	Index to K_{sf} hypermatrix
NDXKSP	Index to K_{sp} hypermatrix
NDXKSS	Index to K_{ss} hypermatrix
NDXUS	Index to U_s hypermatrix
NDXRS	Index to K_s hypermatrix

Remarks:

The tables are initialized using the hypermatrix pointers.

Calls: None

Called by: HCSINI

COMMON: None

B.2.20 PRESCS - Adjust Load Vector for Prescribed Displacements**Purpose:**

PRESCS adjusts the load vector for the prescribed degrees of freedom in the problem.

Usage: CALL PRESCS (NDXRF, NDXKFP, NDXUP)

Description of Parameters:**Input:**

NDXRF Index to R_f hypermatrix
NDXKFP Index to K_{fp} hypermatrix
NDXUP Index to U_p hypermatrix

Output:

None

Remarks:

The load hypermatrix is adjusted by performing a hypermatrix multiplication.

Calls: HMTMLT

Called by: HCSSOL

Common: None

APPENDIX C

CHOLESKY HYPERMATRIX EQUATION SOLUTION ROUTINES SOURCE LISTING

This appendix contains the FORTRAN source listing of the routines in the Cholesky Hypermatrix Equation Solution Scheme package. A description of the routines is contained in appendix B.

```

c
c.....
c   File = [USERS.ASICS.HCSMOD]HCSMOD.FOR
c.....
c
c   Module.....HCSMOD
c   Purpose.....Linear Equation Solver
c   Developer.....M.W.Eastman.   U.C.T.
c   Date.....July 1987
c   References.....DAS Programmer's Manual
c   Language.....FORTRAN 77
c   Installation.....MicroVax II.  U.C.T.
c
c.....
c
c   Description of the module:
c   -----
c
c   This module solves a system of symmetric positive-definite
c   linear equations derived from the finite element method. The
c   solution method involves Cholesky decomposition with a
c   hypermatrix data management scheme
c
c   Description of routines
c
c   HCSINI  Initializes the equation solution procedure
c   HCSDOF  Specifies a global degree of freedom in hypertable
c   HCSCHR  Checks that the global degree of freedom table is
c           completely defined
c   HCSABK  Assembles element stiffness matrix into global stiffness
c           hypermatrices
c   HCSABR  Assembles element load vector in global load vector
c   HCSSOL  Controls solution of equations and calculation of
c           reactions
c   HCSUGT  Retrieves calculated displacement
c   HCSRGT  Retrieves calculated reaction
c   HCSDMP  Dumps specified hypermatrix
c   PGMINI  Initializes program module
c   MATSIZ  Calculates size of the submatrices for the solution
c   DAFCRT  Creates and format DYNA-file
c   HYPINI  Initializes hypermatrices and hypertable
c   INDEXS  Initializes indirect pointer look-up tables
c   PRESCS  Adjusts load vector for prescribed displacement
c   CHLSKY  Driver routine for Cholesky equation solution procedure
c   CHKTRI  Calculates L matrix for diagonal submatrices
c   CHKRCT  Calculates L matrix for non-diagonal sub-matrices
c   CHKYYY  Calculates Y matrix, forward substitution
c   CHKXXX  Calculates X matrix, backward substitution
c   SUMTRI  Calculates summation for diagonal submatrices

```

```

c      SUMRCT  Calculates summation for non-diagonal submatrices
c      SUMYYY  Calculates summation for Y submatrix
c      SUMXXX  Calculates summation for X submatrix
c      SMTPUT  Stores submatrix in DYNA-file
c      SMTGET  Retrieves submatrix from DYNA-file
c      CREAT  Calculates reactions
c      DPIHMT  Dumps integer hypermatrix
c      DPRHMT  Dumps real hypermatrix
c      HMTMLT  Multiplies hypermatrices
c      SMTMLT  Multiplies submatrices
c.....
c
c      subroutine HCSINI ( ndfre, ndpre, ndsup, isymm, ncase, luerr )
c
c.....
c      purpose:
c      initialises hypermatrix cholesky solver parameters
c
c      arguments:
c i   ndfre  - number of free dof
c i   ndpre  - number of prescribed dof
c i   ndsup  - number of suppressed dof
c i   isymm  - symmetry indicator 0 - unsymmetric 1 - symmetric
c i   ncase  - no. of load cases
c i   luerr  - logical unit for error printing
c.....
c
c      include 'hcscom.inc'
c      include 'dofcom.inc'
c
c---  set values in 'hcscom' common
c      nfree = ndfre
c      npres = ndpre
c      nsupp = ndsup
c      ifsym = isymm
c      nload = ncase
c      lunerr = luerr
c
c---  calculate submatrix size
c      call MATSIZ ( nfree, npres, nsupp, nsizs )
c
c---  initialise dof counters
c      ifree = 0
c      ipres = 0
c      isupp = 0
c
c---  create hypermatrices
c      call HYPINI ( ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
c      &              ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
c      &              ndxksf, ndxksp, ndxkss, ndxus, ndxrs,
c      &              ndxfix, nfree, npres, nsupp, nload,
c      &              nsizs, ifsym )
c

```

```

c--- set pointer tables for stiffness, displacements and loads
      call INDEXS ( indexk, indexu, indexr, ifsym,
&                ndbkff, ndbkfp, ndbkfs, ndxuf, ndxrf,
&                ndbkpf, ndbkpp, ndbkps, ndxup, ndxrp,
&                ndbksp, ndbkss, ndxus, ndxrs )
c
      return
      end
c
c
      subroutine HCSDOF ( itotv, iffps, fixed )
c
c.....
c      purpose :
c      specifies degrees of freedom
c
c      arguments:
c i   itotv - global dof number
c i   iffps - flag to indicate free(1)/prescribed(2)/suppressed(3)
c i   fixed - prescribed value
c.....
c
      implicit double precision(a-h,o-z)
c
      include 'hcscom.inc'
      include 'dofcom.inc'
c
      integer HMGINF
      integer iffix(2)
c
c--- retrieve lun no.
      lunerr = LUNGET(7)
c
      nupwpe = HMGINF(ndxup,3)
c
      iffix(1) = iffps
c
      if(iffps.eq.1) then
         ifree = ifree + 1
         iffix(2) = ifree
      else
         if(iffps.eq.2) then
            ipres = ipres + 1
            iffix(2) = ipres
c--- set prescribed value in hypermatrix
            do 10 iload = 1,nload
10       call HMGFVS(ndxup,ipres,iload,fixed,nupwpe)
         else
            if(iffps.eq.3) then
               isupp = isupp + 1
               iffix(2) = isupp
            else
               write(lunerr,*) 'Incorrect degree of freedom,iffps',iffps
               iffix(2) = iffix(2)/INFINT(0)
            end if
         end if
      end if
end if

```

```

c
c--- store tuple
      call HIGPUT(ndxfix,itotv,ifix,2)
c
      return
      end
c
c
      subroutine HCSCHK
c
c.....
c  purpose:
c  checks that the dof hypermatrix has been correctly set up
c
c  arguments
c  none
c.....
c
      include 'hcscom.inc'
      include 'dofcom.inc'
c
      integer HMGINF
      integer ifix(2),ifps(2)
c
c--- retrieve lun no.
      lunerr = LUNGET(7)
c
      ntotv = nfree + npres + nsupp
c
      do 100 itotv = 1,ntotv
      call HIGGET(ndxfix,itotv,ifps,2)
      if(ifps(1).eq.0) then
         ifree = ifree + 1
         ifix(1) = 1
         ifix(2) = ifree
         call HIGPUT(ndxfix,itotv,ifix,2)
      else
         if(ifps(1).gt.3) then
            write(lunerr,*) 'Error in dof specification:ifps ',ifps(1)
            ifps(1) = ifps(1)/INFINT(0)
         end if
      end if
100  continue
c
      if(ifree.ne.nfree) then
         write(lunerr,*) 'Error in free dof specification,ifree,nfree',
&             ifree,nfree
         ifree = ifree/INFINT(0)
      end if
c
      if(ipres.ne.npres) then
         write(lunerr,*) 'Error in pres dof specification,ipres,npres',
&             ipres,npres
         ipres = ipres/INFINT(0)
      end if

```

```

c      if(isupp.ne.nsupp) then
c          write(lunerr,*) 'Error in supp dof specification,isupp,nsupp',
c      &          isupp,nsupp
c          isupp = isupp/INFINT(0)
c      end if
c
c      return
c      end
c
c      subroutine HCSABK ( nevab,  estif,  eldof )
c
c.....
c      purpose :
c      assembles element stiffness matrix into global stiffness matrix
c
c      arguments:
c i  nevab  - no. of variables in element
c i  estif  - element stiffness matrix
c i  eldof  - element degrees of freedom codes
c.....
c
c      implicit double precision(a-h,o-z)
c
c      include 'hcscom.inc'
c
c      real*8  estif(nevab,nevab)
c      integer HMGINF
c      integer eldof(nevab)
c      integer ifrow(2),ifcol(2),iftmp(2),hmgwpe
c
c---  check for symmetry
c      if(ifsym.eq.0) then
c---  unsymmetric
c          do 100 ievab = 1,nevab
c              do 110 jevab = 1,nevab
c
c                  nrowg = eldof(ievab)
c                  ncolg = eldof(jevab)
c
c---  retrieve fixity code
c          call HITGET(ndxfix,nrowg,ifrow,2)
c          call HITGET(ndxfix,ncolg,ifcol,2)
c
c---  check fixity code and place in correct matrix
c          ndxhmg = indexk(ifrow(1),ifcol(1))
c          if(ndxhmg.gt.0) then
c              hmgwpe = HMGINF(ndxhmg,3)
c              call HMGEVG(ndxhmg,ifrow(2),ifcol(2),gstif,hmgwpe)
c              gstif = gstif + estif(ievab,jevab)
c              call HMGEVS(ndxhmg,ifrow(2),ifcol(2),gstif,hmgwpe)
c          end if
c      110  continue
c      100  continue
c      else

```

```

c
c--- symmetric
do 200 ievab = 1,nevab
do 210 jevab = ievab,nevab
c
nrowg = eldof(ievab)
ncolg = eldof(jevab)
c
c--- retrieve fixity code
call HIGGET(ndxfix,nrowg,ifrow,2)
call HIGGET(ndxfix,ncolg,ifcol,2)
c
if(ifrow(1).gt.ifcol(1)) then
  iftmp(1) = ifrow(1)
  iftmp(2) = ifrow(2)
  ifrow(1) = ifcol(1)
  ifrow(2) = ifcol(2)
  ifcol(1) = iftmp(1)
  ifcol(2) = iftmp(2)
else
  if(ifrow(1).eq.ifcol(1)) then
    if(ifrow(2).gt.ifcol(2)) then
      iftmp(2) = ifrow(2)
      ifrow(2) = ifcol(2)
      ifcol(2) = iftmp(2)
    end if
  end if
end if
c
c--- check fixity code and place in correct matrix
ndxbhng = indexk(ifrow(1),ifcol(1))
if(ndxbhng.gt.0) then
  hmgwpe = HMGINF(ndxbhng,3)
  call HMGEVG(ndxbhng,ifrow(2),ifcol(2),gstif,hmgwpe)
  gstif = gstif + estif(ievab,jevab)
  call HMGEVS(ndxbhng,ifrow(2),ifcol(2),gstif,hmgwpe)
end if
210 continue
200 continue
end if
c
return
end
c
c
c
subroutine HCSABR ( nevab, eload, eldof, iload )
c
c.....
c  purpose :
c  assembles element stiffness matrix into global stiffness matrix
c
c  arguments:
c  nevab - no. of variables in element
c  estif - element stiffness matrix
c  eldof - element degrees of freedom codes
c  iload - load case number
c.....

```

```

c      implicit double precision(a-h,o-z)
c
c      include 'hcscom.inc'
c
c      real*8  eload(nevab)
c      integer HMGINF
c      integer eldof(nevab)
c      integer ifrow(2),hmgwpe
c
c      do 100 ievab = 1,nevab
c
c      nrowg = eldof(ievab)
c
c      retrieve fixity code
c--- call HIGGET(ndxfix,nrowg,ifrow,2)
c
c      check fixity code and place in correct matrix
c--- ndxhmg = indexr(ifrow(1))
c      if(ndxhmg.gt.0) then
c          hmgwpe = HMGINF(ndxhmg,3)
c          call HMGVVG(ndxhmg,ifrow(2),iload,gload,hmgwpe)
c          if(ifrow(1).eq.1) then
c              gload = gload + eload(ievab)
c          else
c              gload = gload - eload(ievab)
c          end if
c          call HMGVVS(ndxhmg,ifrow(2),iload,gload,hmgwpe)
c      end if
110  continue
100  continue
c
c      return
c      end
c
c      subroutine HCSSOL
c
c      .....
c      purpose:
c      solves equations using Cholesky decomposition and
c      hypermatrix multiplication
c
c      arguments
c      none
c      .....
c
c      include 'hcscom.inc'
c
c--- alter gload for prescribed displacements ( Rf = Rf - Kfp.Up )
c      call PRESOS ( ndxuf, ndxkfp, ndxup )
c
c--- call Cholesky subroutine
c      call CHLSKY ( ndxkff, ndxuf )
c
c--- calculate reactions
c      call CREAT ( ndxrp, ndxkpf, ndxuf, ndxkpp, ndxup,
&                  ndxrs, ndxksf, ndxksp, ifsym )
c
c      return
c      end

```

```

c
c
c      double precision function HCSUGT ( itotv, iload )
c
c.....
c      Purpose
c      retrieves global displacement
c
c      arguments:
c i   itotv - global dof number
c i   iload - load case number
c.....
c
c      implicit double precision(a-h,o-z)
c
c      include 'hcscom.inc'
c
c      integer HMGINF
c      integer iffix(2)
c      integer hmgwpe
c
c      call HTGGET(ndxfix,itotv,iffix,2)
c
c---  check if hypermatrix exists
c      ndxhmg = indexu(iffix(1))
c      if(ndxhmg.gt.0) then
c          hmgwpe = HMGINF(ndxhmg,3)
c          call HMGVEVG(ndxhmg,iffix(2),iload,value,hmgwpe)
c          HCSUGT = value
c      else
c          HCSUGT = 0.0d0
c      end if
c
c      return
c      end
c
c
c      double precision function HCSRGT ( itotv, iload )
c
c.....
c      purpose
c      retrieves global reaction
c
c      arguments:
c i   itotv - global dof number
c i   iload - load case number
c.....
c
c      implicit double precision(a-h,o-z)
c      include 'hcscom.inc'
c
c      integer HMGINF
c      integer iffix(2)
c      integer hmgwpe
c
c      call HTGGET(ndxfix,itotv,iffix,2)

```

```

C
C--- check if hypermatrix exists
      ndxhmg = indexr(iffix(1))
      if(ndxhmg.gt.0.and.iffix(1).ne.1) then
         hmgwpe = HMGINF(ndxhmg,3)
         call HMGEVG(ndxhmg,iffix(2),iload,value,hmgwpe)
         HCSRGT = value
      else
         HCSRGT = 0.0d0
      end if

C
      return
      end

C
C
      subroutine HCSDMP ( string )
C
C.....
C      purpose:
C      dumps hypermatrices
C
C      arguments:
C      string - hypermatrix identifier
C.....
C
C      character*1 string
C
C      include 'hcscom.inc'
C
C      if(string.eq.'K') then
         call DPRHMT('Kff',ndxkff)
         call DPRHMT('Kfp',ndxkfp)
         call DPRHMT('Kfs',ndxkfs)
         call DPRHMT('Kpf',ndxkpf)
         call DPRHMT('Kpp',ndxkpp)
         call DPRHMT('Kps',ndxkps)
         call DPRHMT('Ksf',ndxksf)
         call DPRHMT('Ksp',ndxksp)
         call DPRHMT('Kss',ndxkss)
      end if
C
C      if(string.eq.'U') then
         call DPRHMT('Uf',ndxuf)
         call DPRHMT('Up',ndxup)
         call DPRHMT('Us',ndxus)
      end if
C
C      if(string.eq.'R') then
         call DPRHMT('Rf',ndxrf)
         call DPRHMT('Rp',ndxrp)
         call DPRHMT('Rs',ndxrs)
      end if
C
      return
      end

```

```

      subroutine MATSIZ ( nfree, npres, nsupp, nsizs )
c
c.....
c      purpose:
c      calculates size of sub-matrices for solution
c
c      arguments:
c      nfree  - size of Kff hypermatrix
c      npres  - size of Kpp hypermatrix
c      nsupp  - size of Kss hypermatrix
c      nsizs  - size of sub-matrices
c.....
c
c      parameter MSIZS=15
c      parameter NWDPE=2
c      integer HMGINF,DAINF,INFINT
c
c--- retrieve lun no.
c      lunerr = LUNGET(6)
c
c--- retrieve header sizes for hypermatrix control
c      nwdhm = HMGINF(0,1)
c      nwdsm = HMGINF(0,2)
c
c--- retrieve page size
c      ipagsz = DAINFI(1)
c
c--- calculate size sub-matrices
c      nsizs = sqrt(float(ipagsz - nwdsm)/NWDPE)
c
c--- check if larger than Kff stiffness matrix
c      if(nsizs.gt.nfree) nsizs = nfree
c
c--- check if larger than maximum allowed
c      if(nsizs.gt.MSIZS) nsizs = MSIZS
c
c--- check if Kff control info fits on a single page
c      nsizh = (nfree + nsizs - 1)/nsizs
c      nsm = nsizh*nsizh
c      if(ipagsz.lt.nsm+nwdhm) then
c         write(lunerr,1003) 'Kff',nsm+nwdhm,ipagsz
c         nsm = nsm/INFINT(0)
c      end if
c
c--- check if Kpp control info fits on a single page
c      nsizh = (npres + nsizs - 1)/nsizs
c      nsm = nsizh*nsizh
c      if (ipagsz.lt.(nsm + nwdhm)) then
c         write(lunerr,1003) 'Kpp',nsm + nwdhm,ipagsz
c         nsm = nsm / INFINT(0)
c      end if
c
c--- check if Kss control info fits on a single page
c      nsizh = (nsupp + nsizs - 1)/nsizs
c      nsm = nsizh*nsizh

```

```

        if (ipagsz.lt.(nsm + nwdhm)) then
            write(lunerr,1003) 'Kss',nsm + nwdhm,ipagsz
            nsm = nsm / INFINT(0)
        end if

c
        return

c
1003 format(1h ,'=> Unable to create control information for',
& ' hypermatrix ',a3,/,
& 1h , ' Number of words required for control',
& ' information = ',i5,/,
& 1h , ' Number of words available on a page = ',i5)

c
        end

c
c
c
        subroutine HYPINI ( ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
&                          ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
&                          ndxksf, ndxksp, ndxkss, ndxus, ndxrs,
&                          ndxfix, nfree, npres, nsupp, nload,
&                          nsizs, ifsym )

c
c.....
c    purpose :
c    initialise all hyper matrices needed for cholesky solution
c
c    arguments:
c    ndxkff - index to kff hypermatrix
c    ndxkfp - index to kfp hypermatrix
c    ndxkfs - index to kfs hypermatrix
c    ndxuf  - index to uf hypermatrix
c    ndxrf  - index to rf hypermatrix
c    ndxkpf - index to kpf hypermatrix
c    ndxkpp - index to kpp hypermatrix
c    ndxkps - index to kps hypermatrix
c    ndxup  - index to up hypermatrix
c    ndxrp  - index to rp hypermatrix
c    ndxksf - index to ksf hypermatrix
c    ndxksp - index to ksp hypermatrix
c    ndxkss - index to kss hypermatrix
c    ndxus  - index to us hypermatrix
c    ndxrs  - index to rs hypermatrix
c    ndxfix - index to fix hypermatrix
c    nfree  - number of free dof
c    npres  - number of prescribed dof
c    nsupp  - number of suppressed dof
c    nload  - number of load cases
c    nsizs  - dimension of standard sub-matrix
c    ifsym  - symmetry flag
c.....
c
        parameter NWDPE=2
        integer HMGINF
        integer mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,mtndx

c
c--- retrieve io lun no.
        lundmp = LUNGET(8)

```

```

c
c--- calculate size of reduced stiffness matrix
      ntotv = nfree + npres + nsupp
c
c*** hypermatrix Kff
      mthed = 2004287489
      mttyp = 2
      mtwpe = NWDPE
      nrowm = nfree
      ncolm = nfree
      nrows = nsizs
      ncols = nsizs
c--- check sizes
      if((nrowm.le.0).or.(ncolm.le.0)) then
         ndbkff = 0
         write(lundmp,9010) 'Kff ',nrowm,ncolm
      else
c--- create hypermatrix
         call HMGCRD(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
          &          mtndx)
         ndbkff = mtndx
         nrowa = HMGINF(ndbkff,8)
         ncola = HMGINF(ndbkff,9)
c--- print stiffness hypermatrix control parameters
         write(lundmp,9000) 'Kff ',nrowm,ncolm,nrowa,ncola,nrows,
          &          ncols,mttyp,mtwpe
      end if
c
c*** hypermatrix Kpf
      ndbkpf = 0
      if(ifsym.eq.0) then
         mthed = 2004287498
         mttyp = 2
         mtwpe = NWDPE
         nrowm = npres
         ncolm = nfree
         nrows = min(nsizs,npres)
         ncols = nsizs
c--- check sizes
         if((nrowm.le.0).or.(ncolm.le.0)) then
            ndbkpf = 0
            write(lundmp,9010) 'Kpf ',nrowm,ncolm
         else
c--- create hypermatrix
            call HMGCRD(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
             &          mtndx)
            ndbkpf = mtndx
            nrowa = HMGINF(ndbkpf,8)
            ncola = HMGINF(ndbkpf,9)
c--- print stiffness hypermatrix control parameters
            write(lundmp,9000) 'Kpf ',nrowm,ncolm,nrowa,ncola,nrows,
             &          ncols,mttyp,mtwpe
         end if
      end if

```

```

c
c*** hypermatrix Ksf
    ndxksf = 0
    if(ifsym.eq.0) then
        mthed = 2004287491
        mttyp = 2
        mtwpe = NWDPE
        nrowm = nsupp
        ncolm = nfree
        nrow = min(nsizs,nsupp)
        ncol = nsizs
c--- check sizes
        if((nrowm.le.0).or.(ncolm.le.0)) then
            ndbksf = 0
            write(lundmp,9010) 'Ksf ',nrowm,ncolm
        else
c--- create hypermatrix
            call HMGRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrow,ncol,mtptr,
&                    mtndx)
            ndxksf = mtndx
            nrowa = HMGINF(ndbksf,8)
            ncola = HMGINF(ndbksf,9)
c--- print stiffness hypermatrix control parameters
            write(lundmp,9000) 'Ksf ',nrowm,ncolm,nrowa,ncola,nrow,
&                                ncol,mttyp,mtwpe
            end if
        end if
c
c*** hypermatrix Kfp
    mthed = 2004287499
    mttyp = 2
    mtwpe = NWDPE
    nrowm = nfree
    ncolm = npres
    nrow = nsizs
    ncol = min(nsizs,npres)
c--- check sizes
    if((nrowm.le.0).or.(ncolm.le.0)) then
        ndbkfp = 0
        write(lundmp,9010) 'Kfp ',nrowm,ncolm
    else
c--- create hypermatrix
        call HMGRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrow,ncol,mtptr,
&                    mtndx)
        ndbkfp = mtndx
        nrowa = HMGINF(ndbkfp,8)
        ncola = HMGINF(ndbkfp,9)
c--- print stiffness hypermatrix control parameters
        write(lundmp,9000) 'kfp ',nrowm,ncolm,nrowa,ncola,nrow,
&                                ncol,mttyp,mtwpe
        end if
c
c*** hypermatrix kpp
    mthed = 2004287500
    mttyp = 2
    mtwpe = NWDPE
    nrowm = npres
    ncolm = npres
    nrow = min(nsizs,npres)
    ncol = min(nsizs,npres)

```

```

c--- check sizes
      if((nrowm.le.0).or.(ncolm.le.0)) then
        ndbkpp = 0
        write(lundmp,9010) 'Kpp ',nrowm,ncolm
      else
c--- create hypermatrix
        call HMGCR1(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
          &          mtndx)
        ndbkpp = mtndx
        nrowa = HMGINF(ndbkpp,8)
        ncola = HMGINF(ndbkpp,9)
c--- print stiffness hypermatrix control parameters
        write(lundmp,9000) 'kpp ',nrowm,ncolm,nrowa,ncola,nrows,
          &          ncols,mttyp,mtwpe
        end if
c
c*** hypermatrix ksp
      ndbksp = 0
      if(ifsym.eq.0) then
        mthed = 2004287501
        mttyp = 2
        mtwpe = NWDPE
        nrowm = nsupp
        ncolm = npres
        nrows = min(nsizs,nsupp)
        ncols = min(nsizs,npres)
c--- check sizes
        if((nrowm.le.0).or.(ncolm.le.0)) then
          ndbksp = 0
          write(lundmp,9010) 'Ksp ',nrowm,ncolm
        else
c--- create hypermatrix
          call HMGCR1(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
            &          mtndx)
          ndbksp = mtndx
          nrowa = HMGINF(ndbksp,8)
          ncola = HMGINF(ndbksp,9)
c--- print stiffness hypermatrix control parameters
          write(lundmp,9000) 'Ksp ',nrowm,ncolm,nrowa,ncola,nrows,
            &          ncols,mttyp,mtwpe
          end if
        end if
c
c*** hypermatrix kfs
      mthed = 2004287490
      mttyp = 2
      mtwpe = NWDPE
      nrowm = nfree
      ncolm = nsupp
      nrows = nsizs
      ncols = min(nsizs,nsupp)
c--- check sizes
      if((nrowm.le.0).or.(ncolm.le.0)) then
        ndbkfs = 0
        write(lundmp,9010) 'Kfs ',nrowm,ncolm
      else

```

```

c---      create hypermatrix
          call HMGCRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
          nbdkfs = mtndx
          nrowa = HMGINF(nbdkfs,8)
          ncola = HMGINF(nbdkfs,9)
c---      print stiffness hypermatrix control parameters
          write(lundmp,9000) 'Kfs ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
          end if
c
c***      hypermatrix kps
          mthed = 2004287502
          mttyp = 2
          mtwpe = NWDPE
          nrowm = npres
          ncolm = nsupp
          nrows = min(nsizs,npres)
          ncols = min(nsizs,nsupp)
c---      check sizes
          if((nrowm.le.0).or.(ncolm.le.0)) then
              nbdkps = 0
              write(lundmp,9010) 'Kps ',nrowm,ncolm
          else
c---      create hypermatrix
              call HMGCRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&              mtndx)
              nbdkps = mtndx
              nrowa = HMGINF(nbdkps,8)
              ncola = HMGINF(nbdkps,9)
c---      print stiffness hypermatrix control parameters
              write(lundmp,9000) 'Kps ',nrowm,ncolm,nrowa,ncola,nrows,
&              ncols,mttyp,mtwpe
              end if
c
c***      hypermatrix kss
          nbdkss = 0
          if(nbdkss.gt.0) then
              mthed = 2004287492
              mttyp = 2
              mtwpe = NWDPE
              nrowm = nsupp
              ncolm = nsupp
              nrows = min(nsizs,nsupp)
              ncols = min(nsizs,nsupp)
c---      check sizes
              if((nrowm.le.0).or.(ncolm.le.0)) then
                  nbdkss = 0
                  write(lundmp,9010) 'Kss ',nrowm,ncolm
              else
c---      create hypermatrix
                  call HMGCRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                  mtndx)
                  nbdkss = mtndx
                  nrowa = HMGINF(nbdkss,8)
                  ncola = HMGINF(nbdkss,9)

```

```

c---      print stiffness hypermatrix control parameters
          write(lundmp,9000) 'kss ',nrowm,ncolm,nrowa,ncola,nrows,
          &                  ncols,mttyp,mtwpe
          end if
          end if

c
c***      hypermatrix uf
          mthed = 2004287493
          mttyp = 2
          mtwpe = NWDPE
          nrowm = nfree
          ncolm = nload
          nrows = nsizs
          ncols = min(nsizs,nload)
c---      check sizes
          if((nrowm.le.0).or.(ncolm.le.0)) then
              ndxuf = 0
              write(lundmp,9010) 'Uf ',nrowm,ncolm
          else
c---          create hypermatrix
              call HMGCRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
              &                  mtndx)
              ndxuf = mtndx
              nrowa = HMGINF(ndxuf,8)
              ncola = HMGINF(ndxuf,9)
c---          print stiffness hypermatrix control parameters
              write(lundmp,9000) 'Uf ',nrowm,ncolm,nrowa,ncola,nrows,
              &                  ncols,mttyp,mtwpe
              end if
          end if

c
c***      hypermatrix up
          mthed = 2004287503
          mttyp = 2
          mtwpe = NWDPE
          nrowm = npres
          ncolm = nload
          nrows = min(nsizs,npres)
          ncols = min(nsizs,nload)
c---      check sizes
          if((nrowm.le.0).or.(ncolm.le.0)) then
              ndxup = 0
              write(lundmp,9010) 'Up ',nrowm,ncolm
          else
c---          create hypermatrix
              call HMGCRT(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
              &                  mtndx)
              ndxup = mtndx
              nrowa = HMGINF(ndxup,8)
              ncola = HMGINF(ndxup,9)
c---          print stiffness hypermatrix control parameters
              write(lundmp,9000) 'Up ',nrowm,ncolm,nrowa,ncola,nrows,
              &                  ncols,mttyp,mtwpe
              end if
          end if
c

```

```

c*** hypermatrix us
    ndxus = 0
    if(ndxus.gt.0) then
        mthed = 2004287494
        mttyp = 2
        mtwpe = NWDPE
        nrowm = nsupp
        ncolm = nload
        nrows = min(nsizs,nsupp)
        ncols = min(nsizs,nload)
c--- check sizes
        if((nrowm.le.0).or.(ncolm.le.0)) then
            ndxus = 0
            write(lundmp,9010) 'Us  ',nrowm,ncolm
        else
c--- create hypermatrix
            call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                    mtndx)
            ndxus = mtndx
            nrowa = HMGINF(ndxus ,8)
            ncola = HMGINF(ndxus ,9)
c--- print stiffness hypermatrix control parameters
            write(lundmp,9000) 'Us  ',nrowm,ncolm,nrowa,ncola,nrows,
&                    ncols,mttyp,mtwpe
            end if
            end if
c
c*** hypermatrix rf
    ndxrf = 0
    if(ndxrf.gt.0) then
        mthed = 2004287495
        mttyp = 2
        mtwpe = NWDPE
        nrowm = nfree
        ncolm = nload
        nrows = nsizs
        ncols = min(nsizs,nload)
c--- check sizes
        if((nrowm.le.0).or.(ncolm.le.0)) then
            ndxrf = 0
            write(lundmp,9010) 'Rf  ',nrowm,ncolm
        else
c--- create hypermatrix
            call HMGCRF(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&                    mtndx)
            ndxrf = mtndx
            nrowa = HMGINF(ndxrf ,8)
            ncola = HMGINF(ndxrf ,9)
c--- print stiffness hypermatrix control parameters
            write(lundmp,9000) 'Rf  ',nrowm,ncolm,nrowa,ncola,nrows,
&                    ncols,mttyp,mtwpe
            end if
            end if
c

```

```

c*** hypermatrix rp
mthed = 2004287504
mttyp = 2
mtwpe = NWDPE
nrowm = npres
ncolm = nload
nrows = min(nsizs,npres)
ncols = min(nsizs,nload)
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
  ndxrp = 0
  write(lundmp,9010) 'Rp ',nrowm,ncolm
else
c--- create hypermatrix
  call HMGCRt(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
  ndxrp = mtndx
  nrowa = HMGINF(ndxrp,8)
  ncola = HMGINF(ndxrp,9)
c--- print stiffness hypermatrix control parameters
  write(lundmp,9000) 'Rp ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
  end if
c
c*** hypermatrix Rs
mthed = 2004287496
mttyp = 2
mtwpe = NWDPE
nrowm = nsupp
ncolm = nload
nrows = min(nsizs,nsupp)
ncols = min(nsizs,nload)
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
  ndxrs = 0
  write(lundmp,9010) 'Rs ',nrowm,ncolm
else
c--- create hypermatrix
  call HMGCRt(mthed,mttyp,mtwpe,nrowm,ncolm,nrows,ncols,mtptr,
&          mtndx)
  ndxrs = mtndx
  nrowa = HMGINF(ndxrs,8)
  ncola = HMGINF(ndxrs,9)
c--- print stiffness hypermatrix control parameters
  write(lundmp,9000) 'Rs ',nrowm,ncolm,nrowa,ncola,nrows,
&          ncols,mttyp,mtwpe
  end if
c
c*** global degree of freedom table
mthed = 2004287497
nrowm = ntotv
ncolm = 2
c--- check sizes
if((nrowm.le.0).or.(ncolm.le.0)) then
  ndxfix = 0
  write(lundmp,9010) 'Fix ',nrowm,ncolm
else

```

```

c---      create hypertable
          call HTGCRT(mthed,ncolm,nrowm,mtptr,mtndx)
          ndxfix = mtndx
c---      print stiffness hypermatrix control parameters
          write(lundmp,9020) 'Fix ',nrowm,ncolm
end if

c
      return

c
9000      format(1h0,1a5,4x      , ' matrix      ',i5,' by ',i5,/,
&          1h , 'address matrix      ',i5,' by ',i5,/,
&          1h , 'submatrices          ',i5,' by ',i5,/,
&          1h , 'matrix type          ',i5,/,
&          1h , 'words per element    ',i5)

c
9010      format(1h0,1a5,4x      , ' matrix not created ',/,
&          1h , 'Number of rows      : ',i5,/,
&          1h , 'Number of columns   : ',i5)

c
9020      format(1h0,1a5,4x      , ' table      ',i5,' by ',i5)

c
      end

c
c
c
      subroutine INDEXS ( indexk, indexu, indexr, ifsym,
&          ndxkff, ndxkfp, ndxkfs, ndxuf, ndxrf,
&          ndxkpf, ndxkpp, ndxkps, ndxup, ndxrp,
&          ndxksf, ndxksp, ndxkss, ndxus, ndxrs )

c
c.....
c      purpose :
c      sets up look-up table for stiffness,displacement and loads
c
c      arguments
c o      indexk - stiffness look-up table
c o      indexu - displacement look-up table
c o      indexr - loads look-up table
c i      ifsym - symmetry flag
c i      ndxkff - index to kff hypermatrix
c i      ndxkfp - index to kfp hypermatrix
c i      ndxkfs - index to kfs hypermatrix
c i      ndxuf - index to uf hypermatrix
c i      ndxrf - index to rf hypermatrix
c i      ndxkpf - index to kpf hypermatrix
c i      ndxkpp - index to kpp hypermatrix
c i      ndxkps - index to kps hypermatrix
c i      ndxup - index to up hypermatrix
c i      ndxrp - index to rp hypermatrix
c i      ndxksf - index to ksf hypermatrix
c i      ndxksp - index to ksp hypermatrix
c i      ndxkss - index to kss hypermatrix
c i      ndxus - index to us hypermatrix
c i      ndxrs - index to rs hypermatrix
c.....
c
      integer indexk(3,3),
&          indexu(3),
&          indexr(3)

```

```

c
c--- check for symmetry
      if(ifsym.eq.1) then
         ndbkpf = ndbkfp
         ndbksf = ndbkfs
         ndbksp = ndbkps
      end if

c
c--- stiffnesses
      indexk(1,1) = ndbkff
      indexk(2,1) = ndbkpf
      indexk(3,1) = ndbksf
      indexk(1,2) = ndbkfp
      indexk(2,2) = ndbkpp
      indexk(3,2) = ndbksp
      indexk(1,3) = ndbkfs
      indexk(2,3) = ndbkps
      indexk(3,3) = ndbkss

c
c--- displacements
      indexu(1) = ndxuf
      indexu(2) = ndxup
      indexu(3) = ndxus

c
c--- loads
      indexr(1) = ndxrf
      indexr(1) = ndxuf
      indexr(2) = ndbrp
      indexr(3) = ndbrs

c
      return
      end

c
c
      subroutine PRESQS ( ndbrf, ndbkfp, ndxup )

c.....
c      purpose:
c      alters glod for prescribed displacements
c      performs  $Rf = Rf - Kfs.U_s$ 
c
c      arguments :
c      ndbrf - index to rf hypermatrix
c      ndbkfp - index to kfp hypermatrix
c      ndxup - index to up hypermatrix
c.....
c
      implicit double precision(a-h,o-z)

c
c--- set flag to subtract matrix multiplication from result
      factr = -1.0d0
c--- set flag not to transpose rf
      kfptp = 0
c--- set flag not to transpose kfs
      nuptp = 0
c--- Kfp not symmetric
      kfpsym = 0

```

```

c--- Up not symmetric
      nupsym = 0
c--- perform operation
      call HMTMLT ( ndbkfp, ndbkup, ndbkrf, factr,
&                  kfptp, nuftp, kfpsym, nupsym )
c
      return
      end
c
c
      subroutine CHLSKY ( gstndx, gldndx )
c
c.....
c      purpose:
c      routine to solve set of linear equation of a symmetric matrix
c
c i   gstndx - index to stiffness hypermatrix
c i   gldndx - index to load hypermatrix
c.....
c
      implicit double precision(a-h,o-z)
      parameter MSIZS=15
      real*8 summa(MSIZS,MSIZS),
&          pivot(MSIZS)
c
      integer HMGINF
      integer gstndx,gldndx
c
c--- dimension of stiffness address matrix
      nsizh = HMGINF(gstndx,8)
c--- dimension of load address matrix
      nsizi = HMGINF(gldndx,9)
c
c--- calculate matrix L
c
      do 100 irowh = 1,nsizh
      call SUMTRI ( gstndx, irowh, summa )
      call CHKTRI ( gstndx, irowh, summa, pivot )
c
      do 200 icolh = irowh+1,nsizh
      call SUMRCT ( gstndx, irowh, icolh, summa )
      call CHKRCT ( gstndx, irowh, icolh, summa, pivot )
200  continue
100  continue
c
c--- calculate Y
c
      do 400 icoli = 1,nsizi
      do 410 irowh = 1,nsizh
      call SUMYYY ( gstndx, gldndx, irowh, icoli, summa )
      call CHKYYY ( gstndx, gldndx, irowh, icoli, summa )
410  continue
400  continue
c

```

```

c--- calculate X
c
  do 500 icoli = 1,nsizi
  do 510 irowh = nsizh,1,-1
  call SUMXXX ( gstndx, gldndx, irowh, icoli, summa )
  call CHKXXX ( gstndx, gldndx, irowh, icoli, summa )
510 continue
500 continue
c
  return
  end
c
c
  subroutine CHKTRI ( gstndx, irowh, summa, pivot )
c
c.....
c  purpose:
c  calculates the L values in the cholesky solution for
c  the pivot submatrix
c
c i  gstndx - index to global stiffness hypermatrix
c i  irowh  - row in addressing matrix
c i  summa  - summation for (irowh,irowh)
c o  pivot  - pivot vector for submatrix
c.....
c
  parameter NWDPE=2
  parameter MSIZS=15
  parameter MPOOL=MSIZS*MSIZS*NWDPE
  common /JUNK/ pool(MPOOL)
  real*8 leqns(MSIZS,MSIZS),
&        summa(MSIZS,MSIZS),
&        pivot(MSIZS)
c  equivalence (leqns(1,1),pool(1))
  integer gstndx
c
c--- retrieve size of submatrix
  call HMGISZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
c
c--- retrieve submatrix
  call SMTGET(gstndx,irowh,irowh,leqns)
c
  do 100 ieqns = 1,nrowp
c
c--- calculate pivot and store
  leqns(ieqns,ieqns) = sqrt(leqns(ieqns,ieqns)-summa(ieqns,ieqns))
  pivot(ieqns) = leqns(ieqns,ieqns)
c
c--- calculate the rest of the row
  do 200 icolp = ieqns+1,ncolp
c
  leqns(ieqns,icolp) = (leqns(ieqns,icolp) - summa(ieqns,icolp))/
&                      pivot(ieqns)
c

```

```

c--- update summation
      do 300 irowp = ieqns+1,icolp
        summa(irowp,icolp) = summa(irowp,icolp) +
&          leqns(ieqns,irowp)*leqns(ieqns,icolp)
300   continue
200   continue
100   continue
c
c--- save submatrix
      call SMTPUT(gstndx,irowh,irowh,leqns)
c
      return
      end
c
c
      subroutine CHKRCT ( gstndx, irowh, icolh, summa, pivot )
c.....
c      purpose:
c      calculates the L values in the cholesky solution
c      for the rectangular part of the address matrix
c
c i   gstndx - index to global stiffness hypermatrix
c i   irowh  - row in addressing matrix
c i   icolh  - column in addressing matrix
c i   summa  - summation for (irowh,icolh)
c i   pivot  - pivot vector for submatrix
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          lrows(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS),
&          pivot(MSIZS)
      equivalence (leqns(1,1),pool(1))
      equivalence (lrows(1,1),pool(LOCP1))
      integer gstndx
c
c--- retrieve size of sub matrix
      call HMGSIZ(gstndx,irowh,icolh,nrowp,ncolp,nwds)
c
c--- retrieve submatrices
      call SMTGET(gstndx,irowh,icolh,leqns)
      call SMTGET(gstndx,irowh,irowh,lrows)
c
c--- loop over submatrix
      do 100 ieqns = 1,nrowp
c
c--- calculate L values
      do 200 icolp = 1,ncolp
        leqns(ieqns,icolp) = (leqns(ieqns,icolp) - summa(ieqns,icolp))/
&          pivot(ieqns)
c

```

```

c--- update summation
do 300 irowp = ieqns+1,nrowp
  summa(irowp,icolp) = summa(irowp,icolp) +
&                      lrows(ieqns,irowp)*leqns(ieqns,icolp)
300 continue
200 continue
100 continue
c
c--- save submatrix
call SMTPUT(gstndx,irowh,icolh,leqns)
c
  return
end
c
c
  subroutine CHKYYY ( gstndx, gldndx, irowh, icoli, summa )
c.....
c  purpose:
c  calculates the l values in the cholesky solution
c  for the pivotal part of the address matrix
c
c i  gstndx - index to global stiffness hypermatrix
c i  gldndx - index to load stiffness hypermatrix
c i  irowh - row in stiffness addressing matrix
c i  icoli - column in load addressing matrix
c i  summa - summation for (irowh,icolh)
c.....
c
  implicit double precision(a-h,o-z)
  parameter NWDPE=2
  parameter MSIZS=15
  parameter MPOOL=MSIZS*MSIZS*NWDPE*2
  parameter LOCP1=MSIZS*MSIZS*NWDPE+1
  common /JUNK/ pool(MPOOL)
  real*8 leqns(MSIZS,MSIZS),
&        summa(MSIZS,MSIZS),
&        yeqns(MSIZS,MSIZS)
  equivalence (leqns(1,1),pool(1))
  equivalence (yeqns(1,1),pool(LOCP1))
  integer gstndx,gldndx
c
c--- retrieve stiffness submatrix
call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
call SMIGET(gstndx,irowh,irowh,leqns)
c
c--- retrieve load submatrix
call HMGSIZ(gldndx,irowh,icoli,nrowl,ncoll,nwds)
call SMIGET(gldndx,irowh,icoli,yeqns)
c
  do 100 ieqns = 1,nrowp
c
c--- loop over loads
  do 200 icoll = 1,ncoll
c
c--- calculate Y vector
    yeqns(ieqns,icoll) = (yeqns(ieqns,icoll) - summa(ieqns,icoll))/
&                        leqns(ieqns,ieqns)

```

```

c
c--- update summation
      do 300 irowp = ieqns+1,nrowp
        summa(irowp,icoll) = summa(irowp,icoll) +
&                                leqns(ieqns,irowp)*yeqns(ieqns,icoll)
300   continue
200   continue
100   continue
c
c--- save submatrix
      call SMTPUT(gldndx,irowh,icoli,yeqns)
c
      return
      end
c
c
      subroutine CHKXXX ( gstndx, gldndx, irowh, icoli, summa )
c.....
c      purpose:
c      calculates the l values in the cholesky solution
c      for the pivotal part of the address matrix
c
c i   gstndx - index to global stiffness hypermatrix
c i   gldndx - index to load  stiffness hypermatrix
c i   irowh  - row in stiffness addressing matrix
c i   icoli  - column in load addressing matrix
c i   summa  - summation for (irowh,icolh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS),
&          yeqns(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      equivalence (yeqns(1,1),pool(LOCP1))
      integer gstndx,gldndx
c
c--- retrieve stiffness submatrix
      call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
      call SMTGET(gstndx,irowh,irowh,leqns)
c
c--- retrieve load submatrix
      call HMGSIZ(gldndx,irowh,icoli,nrowl,ncoll,nwds)
      call SMTGET(gldndx,irowh,icoli,yeqns)
c
      do 100 ieqns = nrowp,1,-1
c
c--- loop over loads
      do 200 icoll = 1,ncoll
c
c--- calculate X vector
      yeqns(ieqns,icoll) = (yeqns(ieqns,icoll) - summa(ieqns,icoll))/
&                                leqns(ieqns,ieqns)

```

```

c
c--- update summation
      do 300 irowp = ieqns-1,1,-1
        summa(irowp,icoll) = summa(irowp,icoll) +
&                                leqns(irowp,ieqns)*yeqns(ieqns,icoll)
300   continue
200   continue
100   continue
c
c--- save submatrix
      call SMTPUT(gstndx,irowh,icoli,yeqns)
c
      return
      end
c
c
      subroutine SUMTRI ( gstndx, irowh, summa )
c
c.....
c  purpose:
c  calculates summation for sub-matrix (irowh,icolh)
c
c i  gstndx - index to global stiffness hypermatrix
c i  irowh  - row in addressing matrix
c o  summa  - summation array for address block (irowh,irowh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter AZERO=0.0d0
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      integer gstndx
      integer HMGINF
      nrows = HMGINF(gstndx,6)
c
c--- retrieve submatrix parameters
      call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
c
c--- zero summation
      do 10 irowp = 1,nrowp
        do 10 icolp = 1,ncolp
10      summa(irowp,icolp) = AZERO
c
c--- loop over submatrices
      do 500 jrowh = 1,irowh-1
c
c--- retrieve submatrices
      call SMTGET(gstndx,jrowh,irowh,leqns)
c

```

```

c--- calculate summation
      do 300 irows = 1,nrows
      do 100 irowp = 1,nrowp
      do 200 icolp = irowp,ncolp
      summa(irowp,icolp) = summa(irowp,icolp) +
&                                leqns(irows,irowp)*leqns(irows,icolp)
200  continue
100  continue
300  continue
c
500  continue
c
      return
      end

c
c
      subroutine SUMRCT ( gstndx, irowh, icolh, summa )
c
c.....
c      purpose:
c      calculates summation for sub-matrix (irowh,icolh)
c
c i   gstndx - index to global stiffness hypermatrix
c i   irowh  - row in addressing matrix
c i   icolh  - column in addressing matrix
c o   summa  - summation array for address block (irowh,icolh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter AZERO=0.0d0
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          lrows(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      equivalence (lrows(1,1),pool(LOCP1))
      integer gstndx
      integer HMGINF
      nrows = HMGINF(gstndx,6)

c
c--- retrieve submatrix parameters
      call HMGSIZ(gstndx,irowh,icolh,nrowp,ncolp,nwds)
c
c--- zero summation
      do 10 irowp = 1,nrowp
      do 10 icolp = 1,ncolp
10    summa(irowp,icolp) = AZERO
c
c--- loop over submatrices
      do 500 jrowh = 1,irowh-1
c
c--- retrieve submatrices
      call SMIGET(gstndx,jrowh,icolh,leqns)
      call SMIGET(gstndx,jrowh,irowh,lrows)

```

```

c
c--- calculate summation
      do 300 irows = 1,nrows
      do 100 irowp = 1,nrowp
      do 200 icolp = 1,ncolp
        summa(irowp,icolp) = summa(irowp,icolp) +
&          lrows(irows,irowp)*leqns(irows,icolp)
200    continue
100    continue
300    continue
500    continue
c
      return
      end

c
c
      subroutine SUMYYY ( gstndx, gldndx, irowh, icoli, summa )
c.....
c      purpose:
c      calculates summation for sub-matrix (irowh,icolh)
c
c i      gstndx - index to global stiffness hypermatrix
c i      gldndx - index to load stiffness hypermatrix
c i      irowh  - row in stiffness addressing matrix
c i      icoli  - row in load addressing matrix
c o      summa  - summation for (irowh,icolh)
c.....
c
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter AZERO=0.0d0
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 leqns(MSIZS,MSIZS),
&          summa(MSIZS,MSIZS),
&          yeqns(MSIZS,MSIZS)
      equivalence (leqns(1,1),pool(1))
      equivalence (yeqns(1,1),pool(LOCP1))
      integer gstndx,gldndx
      integer HMGINF
      nrows = HMGINF(gstndx,6)

c
c--- retrieve submatrix parameters
      call HMGSIZ(gstndx,irowh,irowh,nrowp,ncolp,nwds)
      call HMGSIZ(gldndx,irowh,icoli,nrowl,ncoll,nwds)

c
c--- zero summation
      do 10 irowl = 1,nrowl
      do 10 icoll = 1,ncoll
10    summa(irowl,icoll) = AZERO
c
c--- loop over submatrices
      do 500 jrowh = 1,irowh-1
c

```

```

c--- retrieve submatrices
call SMTGET(gstndx,jrowh,irowh,leqns)
call SMTGET(gldndx,jrowh,icoli,yeqns)
c
c--- calculate summation
do 300 irows = 1,nrows
do 100 irowp = 1,nrowp
do 200 icoll = 1,ncoll
  summa(irowp,icoll) = summa(irowp,icoll) +
&                      leqns(irows,irowp)*yeqns(irows,icoll)
200 continue
100 continue
300 continue
c
500 continue
c
  return
end
c
c
  subroutine SUMXXX ( gstndx, gldndx, irowh, icoli, summa )
c
c.....
c  purpose:
c  calculates summation for submatrix (irowh,icolh)
c
c i  gstndx - index to global stiffness hypermatrix
c i  gldndx - index to load stiffness hypermatrix
c i  irowh - row in stiffness addressing matrix
c i  icoli - column in load addressing matrix
c o  summa - summation vector (irowh)
c.....
c
  implicit double precision(a-h,o-z)
  parameter NWDPE=2
  parameter AZERO=0.0d0
  parameter MSIZS=15
  parameter MPOOL=MSIZS*MSIZS*NWDPE*2
  parameter LOCP1=MSIZS*MSIZS*NWDPE+1
  common /JUNK/ pool(MPOOL)
  real*8 leqns(MSIZS,MSIZS),
&        summa(MSIZS,MSIZS),
&        yeqns(MSIZS,MSIZS)
  equivalence (leqns(1,1),pool(1))
  equivalence (yeqns(1,1),pool(LOCP1))
  integer HMGINF
  integer gstndx,gldndx
c
c--- retrieve size of stiffness address matrix
nsizh = HMGINF(gstndx,8)
c
c--- retrieve load submatrix size
call HMGSIZ(gldndx,1,icoli,nrowl,ncoll,nwds)
c
c--- zero summation
do 10 irowl = 1,nrowl
do 10 icoll = 1,ncoll
10  summa(irowl,icoll) = AZERO

```

```

c
c--- loop over submatrices
do 500 jrowh = nsizh,irowh+1,-1
c
c--- retrieve submatrix parameters
call HMGSIZ(gstndx,irowh,jrowh,nrowp,ncolp,nwds)
c
c--- retrieve submatrices
call SMTGET(gstndx,irowh,jrowh,leqns)
call SMTGET(gldndx,jrowh,icoli,yeqns)
c
c--- calculate summation
do 200 icolp = ncolp,1,-1
do 100 irowp = nrowp,1,-1
do 300 icoll = 1,ncoll
  summa(irowp,icoll) = summa(irowp,icoll) +
&      leqns(irowp,icolp)*yeqns(icoll,icoll)
300 continue
100 continue
200 continue
c
500 continue
c
  return
end

c
c
  subroutine SMTGET ( ndxhmg, irowh, icolh, array )
c
c.....
c  purpose
c  retrieves a submatrix , ensuring that the mapping is correct
c
c i  ndxhmg - index to hypermatrix
c i  irowh  - submatrix row
c i  icolh  - submatrix column
c o  array  - matrix retrieved
c.....
c
  implicit double precision(a-h,o-z)
  parameter NWDPE=2
  parameter MSIZS=15
  parameter MARAY=MSIZS*MSIZS*NWDPE
  real*8 array(MARAY)
c
  call HMGSIZ(ndxhmg,irowh,icolh,nrowp,ncolp,nwdhmg)
c
c--- check if mapping is needed
if(nrowp.ne.MSIZS) then
  call HMGGET(ndxhmg,irowh,icolh,array,nwdhmg)
c--- do mapping
do 100 icolp = ncolp,2,-1
do 100 irowp = nrowp,1,-1
  nlocb = (icolp-1)*nrowp + irowp
  nlocb = (icolp-1)*MSIZS + irowp
  array(nlocb) = array(nloca)
100 continue
else
  call HMGGET(ndxhmg,irowh,icolh,array,nwdhmg)
end if

```

```

c      return
c      end

c
c
c      subroutine SMTPUT ( ndxhmg, irowh , icolh, array )
c.....
c      purpose
c      replaces a submatrix , ensuring that the mapping is correct
c
c i    ndxhmg - index to hypermatrix
c i    irowh  - submatrix row
c i    icolh  - submatrix column
c i    array  - matrix replaced
c.....
c
c      implicit double precision(a-h,o-z)
c      parameter NWDPE=2
c      parameter MSIZS=15
c      parameter MARAY=MSIZS*MSIZS*NWDPE
c      real*8 array(MARAY)
c
c---   retrieve submatrix size
c      call HMGSIZ(ndxhmg,irowh,icolh,nrowp,ncolp,nwdhmg)
c
c---   check if copy is needed
c      if(nrowp.ne.MSIZS) then
c---       do copy
c           do 100 icolp = 2,ncolp
c           do 100 irowp = 1,nrowp
c               nloca = (icolp-1)*MSIZS + irowp
c               nlocb = (icolp-1)*nrowp + irowp
c               array(nlocb) = array(nloca)
100      continue
c           call HMGPOT(ndxhmg,irowh,icolh,array,nwdhmg)
c       else
c           call HMGPOT(ndxhmg,irowh,icolh,array,nwdhmg)
c       end if
c
c      return
c      end

c
c
c      subroutine CREAT ( ndxrp, ndxkpf, ndxuf, ndxkpp, ndxup,
c      &                  ndxrs, ndxksf, ndxksp, ifsym )
c.....
c      purpose:
c      calculates reactions
c
c      ndxrs - index to rs hypermatrix
c      ndxksf - index to ksf hypermatrix
c      ndxuf - index to uf hypermatrix
c      ndxkss - index to kss hypermatrix
c      ndxus - index to us hypermatrix
c.....
c

```

```

implicit double precision(a-h,o-z)
integer trana,tranb

c
c--- calculate  $R_p = -R_p' + K_{pf}.U_f + K_{pp}.U_p$ 
c
c--- set flag for addition
      factr = 1.0d0
c--- set flag not to transpose A
      if(ifsym.eq.0) then
        trana = 0
      else
        trana = 1
      end if
c--- set flag not to transpose B
      tranb = 0
c--- Kpf not symmetric
      kpfsym = 0
c--- Uf not symmetric
      nufsym = 0
c
c--- multiply hypermatrices
      call HMTMLT ( ndxkpf, ndxuf, ndxrp, factr,
&                  trana, tranb, kpfsym, nufsym )
c
c--- set flag for addition
      factr = 1.0d0
c--- set flag not to transpose A
      trana = 0
c--- set flag not to transpose B
      tranb = 0
c--- Kpp symmetric
      kppsym = ifsym
c--- Up not symmetric
      nupsym = 0
c
c--- multiply hypermatrices
      call HMTMLT ( ndxkpp, ndxup, ndxrp, factr,
&                  trana, tranb, kppsym, nupsym )
c
c--- calculate  $R_s = -R_s' + K_{sf}.U_f + K_{sp}.U_p$ 
c
c--- set flag for addition
      factr = 1.0d0
c--- set flag not to transpose A
      if(ifsym.eq.0) then
        trana = 0
      else
        trana = 1
      end if
c--- set flag not to transpose B
      tranb = 0
c--- Ksf not symmetric
      ksfsym = 0
c--- Uf not symmetric
      nufsym = 0
c

```

```

c--- multiply hypermatrices
      call HMIMLT ( ndxksf, ndxuf, ndxrs, factr,
&                  trana, tranb, ksfsym, nufsym )
c
c--- set flag for addition
      factr = 1.0d0
c--- set flag not to transpose A
      if(ifsym.eq.0) then
        tranb = 0
      else
        tranb = 1
      end if
c--- set flag not to transpose B
      tranb = 0
c--- Ksp not symmetric
      kspsym = 0
c--- Up not symmetric
      nupsym = 0
c
c--- multiply hypermatrices
      call HMIMLT ( ndbksp, ndxup, ndxrs, factr,
&                  trana, tranb, kspsym, nupsym )
c
      return
      end
c
c
      subroutine DPIHMT(string,hmgndx)
c
c.....
c      purpose:
c      dumps integer hypermatrix
c
c      arguments:
c      string - character header
c      hmgndx - index to hypermatrix
c.....
c
      implicit double precision(a-h,o-z)
      character*(*) string
      integer HMGINF
      integer hmgndx,hmgwpe
      integer value
c
c--- retrieve lun no.
      lundmp = LUNGET(8)
c
      write(lundmp,9000) string
c
      if(hmgndx.eq.0) then
        write(lundmp,9000) 'hypertable does not exit'
      else
        return
      end if
c
9000 format(1h ,a)
      end

```

```

c
c
c      subroutine DPRHMT(string,hmgndx)
c
c.....
c      purpose:
c      dumps real hypermatrix
c
c      arguments:
c      string - character header
c      hmgndx - index to hypermatrix
c.....
c
c      implicit double precision(a-h,o-z)
c      character*(*) string
c      integer HMGINF
c      integer hmgndx,hmgwpe
c
c--- retrieve lun no.
c      lundmp = LUNGET(8)
c
c      write(lundmp,9000) string
c
c      if(hmgndx.eq.0) then
c         write(lundmp,9000) 'hypermatrix does not exit'
c      else
c
c--- retrieve number of words per element
c      hmgwpe = HMGINF(hmgndx,3)
c
c--- retrieve number of rows and columns in matrix
c      nrowm = HMGINF(hmgndx,4)
c      ncolm = HMGINF(hmgndx,5)
c      do 100 irowm = 1,nrowm
c      do 100 icolm = 1,ncolm
c      call HMGKVG(hmgndx,irowm,icolm,value,hmgwpe)
c      write(lundmp,*) irowm,icolm,value
100  continue
c      end if
c
c      return
c
c      9000 format(1h ,a)
c      end
c
c
c      subroutine HMTMLT ( ndxhma, ndxhmb, ndxhmc, factr,
c      &                  trana, tranb, isyma, isymb )
c
c.....
c      purpose
c      subroutine multiplies two hypermatrices and adds or subtracts the
c      result from the resultant hypermatrix
c

```

```

c      arguments:
c
c      ndxhma - index to first hypermatrix
c      ndxhmb - index to second hypermatrix
c      ndxhmc - index to resultant hypermatrix
c      factr  - indicator for addition/subtraction to result matrix
c      trana  - flag to indicate transform A before multiplication
c      tranb  - flag to indicate transform B before multiplication
c      isyma  - flag to indicate that A is symmetric
c      isymb  - flag to indicate that B is symmetric
c.....
c
c      implicit double precision(a-h,o-z)
c      parameter MSIZS=15
c      real*8 arrayc(MSIZS,MSIZS)
c      integer HMGINF
c      integer trana,tranb,transa,transb
c
c---   retrieve lun no.
c      lunerr = LUNGET(6)
c
c---   check for zero index
c      if(ndxhma.eq.0) goto 9999
c      if(ndxhmb.eq.0) goto 9999
c
c---   retrieve sizes matrices
c      nrowa = HMGINF(ndxhma,4)
c      ncola = HMGINF(ndxhma,5)
c      nrowb = HMGINF(ndxhmb,4)
c      ncolb = HMGINF(ndxhmb,5)
c      nrowc = HMGINF(ndxhmc,4)
c      ncolc = HMGINF(ndxhmc,5)
c
c---   check if matrices may be multiplied
c---   compare A and B
c      if(trana.eq.0) then
c         if(tranb.eq.0) then
c            if(ncola.ne.nrowb) then
c               write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c               stop
c            end if
c         else
c            if(ncola.ne.ncolb) then
c               write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c               stop
c            end if
c         end if
c      else
c         if(tranb.eq.0) then
c            if(nrowa.ne.nrowb) then
c               write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c               stop
c            end if
c         else
c            if(nrowa.ne.ncolb) then
c               write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
c               stop
c            end if
c         end if
c      end if
c
c      end if

```

```

c
c--- compare A and C
  if(trana.eq.0) then
    if(nrowa.ne.nrowc) then
      write(lunerr,9000) 'Hma',nrowa,ncola,'Hmc',nrowc,ncolc
      stop
    endif
  else
    if(ncola.ne.nrowc) then
      write(lunerr,9000) 'Hma',nrowa,ncola,'Hmb',nrowb,ncolb
      stop
    end if
  end if

c
c--- compare B and C
  if(tranb.eq.0) then
    if(ncolb.ne.ncolc) then
      write(lunerr,9000) 'Hmb',nrowb,ncolb,'Hmc',nrowc,ncolc
      stop
    endif
  else
    if(nrowb.ne.ncolc) then
      write(lunerr,9000) 'Hmb',nrowb,ncolb,'Hmc',nrowc,ncolc
      stop
    end if
  end if

c
c--- retrieve number of rows and columns in address matrices
  nrowa = HMGINF(ndxhma,8)
  ncola = HMGINF(ndxhma,9)
  nrowb = HMGINF(ndxhmb,8)
  ncolb = HMGINF(ndxhmb,9)
  nrowc = HMGINF(ndxhmc,8)
  ncolc = HMGINF(ndxhmc,9)

c
  if(trana.eq.0) then
    nmult = ncola
  else
    nmult = nrowa
  end if

c
c--- loop over address matrix
  do 100 irowc = 1,nrowc
  do 110 icolc = 1,ncolc

c
c--- retrieve resultant sub-matrix
  call HMGSIZ(ndxhmc,irowc,icolc,nrowsc,ncolsc,nwds)
  call SMTGET(ndxhmc,irowc,icolc,araye)

c
  do 120 imult = 1,nmult

c
c--- select correct hypermatrices
  if(isyma.eq.0) then
    transa = trana
    if(trana.eq.0) then
      irowa = irowc
      icola = imult
    else

```

```

        irowa = imult
        icola = irowc
    end if
else
    if(irowc.le.imult) then
        irowa = irowc
        icola = imult
        transa = 0
    else
        irowa = imult
        icola = irowc
        transa = 1
    end if
end if

c
if(isymb.eq.0) then
    transb = tranb
    if(tranb.eq.0) then
        irowb = imult
        icolb = icolc
    else
        irowb = icolc
        icolb = imult
    end if
else
    if(imult.le.icolc) then
        irowb = imult
        icolb = icolc
        transa = 0
    else
        irowb = icolc
        icolb = imult
        transb = 1
    end if
end if

c
c---- multiply sub-matrices
call SMIMLT ( ndxhma, irowa, icola,
&             ndxhmb, irowb, icolb,
&             arauc, nrowsc, ncolsc,
&             factr, transa, transb,
&             isyma, isymb )
120 continue
c
    call SMTPUT(ndxhmc,irowc,icolc,arauc)
c
110 continue
100 continue
c
9000 format(1h , 'Hypermatrices do not match for multiplication',/,
&          1h , 'Hypermatix ',a3,' : nrow = ',i5,/,
&          1h ,14x,           ' : ncol = ',i5,/,
&          1h , 'Hypermatix ',a3,' : nrow = ',i5,/,
&          1h ,14x,           ' : ncol = ',i5)
c
9999 continue
c
    return
end

```

```

C
C
      subroutine SMMLT ( ndxhma, irowa, icola,
&                      ndxhmb, irowb, icolb,
&                      arrayc, nrowsc, ncolsc,
&                      factr, transa, transb,
&                      isyma, isymb )
C.....
C      purpose:
C      multiplies two submatrices
C
C      arguments:
C      ndxhma - index to first hypermatyrix
C      irowa  - row in address matrix
C      icola  - column in address matrix
C      ndxhmb - index to second hypermatyrix
C      irowb  - row in address matrix
C      icolb  - column in address matrix
C      arrayc - resultant matrix
C      nrowsc - number of rows in submatrix
C      ncolsc - number of columns in submatrix
C      factr  - flag to indicate multiplication/subtraction
C      transa - flag to indicate transpose A before muliplication
C      transb - flag to indicate transpose B before muliplication
C      isyma  - flag to indicate that A is symmetric
C      isymb  - flag to indicate that B is symmetric
C.....
C
      implicit double precision(a-h,o-z)
      parameter NWDPE=2
      parameter MSIZS=15
      parameter MPOOL=MSIZS*MSIZS*NWDPE*2
      parameter LOCP1=MSIZS*MSIZS*NWDPE+1
      common /JUNK/ pool(MPOOL)
      real*8 arraya(MSIZS,MSIZS),
&          arrayb(MSIZS,MSIZS),
&          arrayc(MSIZS,MSIZS)
      equivalence(arraya(1,1),pool(1))
      equivalence(arrayb(1,1),pool(LOCP1))
      integer transa,transb
C
C--- retrieve submatrices
      call SMTGET(ndxhma,irowa,icola,arraya)
      call SMTGET(ndxhmb,irowb,icolb,arrayb)
C
      call HMGSIZ(ndxhma,irowa,icola,nrowa,ncolsa,nwds)
      if(transa.eq.0) then
         nmult = ncolsa
      else
         nmult = nrowa
      end if
C
C--- do multiplication
      do 100 irowsc = 1,nrowsc
      do 100 icolsc = 1,ncolsc
      do 100 imult = 1,nmult
C

```

```

if(isyma.eq.0) then
  if(transa.eq.0) then
    irowsa = irowsc
    icolsa = imult
  else
    irowsa = imult
    icolsa = irowsc
  end if
else
  if(irowa.eq.icola) then
    irowsa = min(irowsc,imult)
    icolsa = max(irowsc,imult)
  else
    if(transa.eq.0) then
      irowsa = irowsc
      icolsa = imult
    else
      irowsa = imult
      icolsa = irowsc
    end if
  end if
end if
c
if(isymb.eq.0) then
  if(transb.eq.0) then
    irowsb = imult
    icolsb = icolsc
  else
    irowsb = icolsc
    icolsb = imult
  end if
else
  if(irowb.eq.icolb) then
    irowsb = min(imult,icolsc)
    icolsb = max(imult,icolsc)
  else
    if(transb.eq.0) then
      irowsb = imult
      icolsb = icolsc
    else
      irowsb = icolsc
      icolsb = imult
    end if
  end if
end if
c
arrayc(irowsc,icolsc) = arrayc(irowsc,icolsc) + factr*
&                                arraya(irowsa,icolsa)*arrayb(irowsb,icolsb)
c
100 continue
c
return
end

```

APPENDIX D

A SAMPLE PROBLEM

This appendix contains the description of a sample problem and the solution that was obtained with the Cholesky Hypermatrix Solution Scheme implemented in the NEW NOSTRUM research code. These results are also compared with those obtained from equivalent ABAQUS examples. Both of the programs were implemented on the MicroVax II at the University of Cape Town.

D.1 The Finite Element Model

The sample problem, from Hinton and Owen [2], is a simple cantilevered beam with an end point load. The finite element model of the structural system to be analysed is shown in figure D.1.



Figure D.1: Finite element model of cantilevered beam

The original problem, example 1, used a single row of 10 elements, which has 106 degrees of freedom. To test the equation solution package two further discretisations were used. Figure D.2 shows the discretization for example 2 with 3 rows of 10 elements, which has 234 degrees of freedom.

21	22	23	24	25	26	27	28	29	30
11	12	13	14	15	16	17	18	19	20
1	2	3	4	5	6	7	8	9	10

Figure D.2: Cantilevered beam 3x10 elements

Figure D.3 shows the discretisation for example 3 with 3 rows of 30 elements, which has 674 degrees of freedom.

61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Figure D.3: Cantilevered beam 3x30 elements

D.2 Discussion

The accuracy of the results obtained with the NEW NOSTRUM suite are compared with those obtained with the ABAQUS program and theoretical values based on simple beam bending theory. The results for the beam end deflection are illustrated in table D.1.

Example	Theoretical	NOSTRUM	ABAQUS
1: 1x10	-0.135	-0.13427187	-0.1343
2: 3x10	-0.135	-0.13486302	-0.1349
3: 3x30	-0.135	-0.13520581	-0.1352

Table D.1: End deflections of the cantilevered beam.

APPENDIX E

HYPERMATRIX ROUTINES DESCRIPTION

In this appendix the FORTRAN routines in the hypermatrix module are described in detail. The routines were developed by Vos [17] and can be used by an application program to create and access two-level hypermatrices. The routines in the general hypermatrix module call the routines in the DAS-package [17] in turn to allocate space in virtual memory, as well as to store information therein and to retrieve it again.

E.1 HMGINI - Initialize the Internal Data Structure

Purpose:

HMGINI initializes the internal data structure and variables of the general two-level hypertable module. It must be called prior to any other routine in this module.

Usage: CALL HMGINI

Description of Parameters:

None

Remarks:

HMGINI initializes the internal table which contains an entry for each general two-level hypermatrix specified by the application program. Each two-level hypermatrix consists of one root matrix, which contains the necessary control information for the hypermatrix, as well as pointers to all the subtables with the actual data. Each entry in the internal table contains the following control information about a specified hypermatrix:

- > Identifier of the root matrix and each submatrix of the hypermatrix
- > Hypermatrix type (rectangular, symmetric, diagonal)
- > Number of words per entry in the hypermatrix
- > Number of rows in the hypermatrix
- > Number of columns in the hypermatrix
- > Number of rows in the submatrix
- > Number of columns in the submatrix
- > Number of row partitions in the hypermatrix
- > Number of column partitions in the hypermatrix

Calls: None

Called by: Application program

Commons: /COMHMG/

E.2 HTGDMP - Dumps the Variables in COMMON /COMHMG/

Purpose:

HTGDMP displays a formatted dump of all the internal variables and the data structure in COMMON /COMHMG/. It is used primarily for debugging purposes.

Usage: INTEGER NPR
 :
 CALL HMGDMP (NPR)

Description of Parameters:

NRP Logical unit specifier of file to receive the formatted
 dump

Remarks:

This routine is used for debugging purposes and may be called at any stage by the application program to determine the status of variables in COMMON /COMHMG/. It calls routine DAHEX to convert a binary integer to a string of hexadecimal characters.

Calls: DAHEX

Called by: Application program.

Commons: /COMHMG/.

E.3 HMGCRT - Creates A New Hypertable

Purpose:

HMGCRT creates a new two-level hypermatrix, that is, it allocates space in virtual memory for the root matrix and all the matrices of the specified hypermatrix. It returns the index of the appropriate entry in the internal table where all the control information for this hypermatrix are stored to the calling routine, as well as a pointer to the root matrix of this hypermatrix.

Usage: CALL HMGCRT (IDHED, MATTYP, NELTWD, IROWHM, ICOLHM, IROWSM,
 & ICOLSM, IHMPTR, NOWHMG)

Description of Parameters:

IDHED	Header for hypermatrix
MATTYP	Matrix type
NELTWD	Number of word per element
IROWHM	Total number of rows in hypermatrix
ICOLHM	Total number of columns in hypermatrix
IROWSM	Number of rows in submatrix
ICOLSM	Number of columns in submatrix
IHMPTR	Pointer to hypermatrix in dyna file
NOWHMG	Current index to control table for hypermatrices

Calls: DACRET, DAHEX, DAPUTV, INFINT, HMGSIZ

Called by: Application program

Common: /COMHMG/

E.4 HMGOPN - Opens an existing hypermatrix

Purpose:

HMGOPN opens an existing two-level hypermatrix which had previously been created by calling HMGCRT. It returns to the calling routine the index of the appropriate entry in the internal table where the control information of the hypertable is located.

Usage: CALL HMGOPN (IDHED, MATTYP, NELTWD, IROWHM, ICOLHM, IROWSM,
& ICOLSM, IHMPTR, NOWHMG)

Description of Parameters:

IDHED	Header for hypermatrix
MATTYP	Matrix type
NELTWD	Number of words per element
IROWHM	Total number of rows in hypermatrix
ICOLHM	Total number of columns in hypermatrix
IROWSM	Number of rows in submatrix
ICOLSM	Number of columns in submatrix
IHMPTR	Pointer to hypermatrix in dyna file
NOWHMG	Current index in control table for hypermatrices

Calls: DACRET, DAHEX, DAPUTV, INFINT, HMGSIZ

Called by:

Common: /COMHMG/

E.5 HMGPUT - Assigns values to an entry in a hypermatrix

Purpose:

HMGPUT assigns the data values in a given matrix to a submatrix in the specified hypermatrix.

Usage: CALL HMGPUT (NOWHMG, IROWP, ICOLP, IARAY, NWDS)

Description of Parameters

NOWHMG	Index in control table for hypermatrices
IROWP	Row partition of submatrix in hypermatrix
ICOLP	Column partition of submatrix in hypermatrix
IARAY	Submatrix with values
NWDS	Number of words in submatrix

Calls: DALOC, HMGNDX, INFINT, DAGETV, HMGEXT, DAHEX

Called by: Application Program

Common: /COMHMG/

E.6 HMGGET - Retrieves Values from an Entry in a Hypermatrix

Purpose:

HMGGET retrieves the data values from a given submatrix in the specified hypermatrix.

Usage: CALL HMGGET (NOWHMG, IROWP, ICOLP, IARAY, NWDS)

Description of Parameters:

NOWHMG	Index in control table for hypermatrices
IROWP	Row partition of submatrix in hypermatrix
ICOLP	Column partition of submatrix in hypermatrix
IARAY	Submatrix with values
NWDS	Number of words in submatrix

Calls: DALOC, HMGNDX, INFINT, DAGETV, HMGEXT, DANEX

Called by: Application Program

Common: /COMHMG/

E.7 HMGSIZ - Calculates number of entries for a submatrix

Purpose:

HMGSIZ calculates the dimensions of a submatrix and the number of words occupied by the submatrix.

Usage: CALL HMGSIZ (NOWHMG, IROWP, ICOLP, IROWS, ICOLS, NUMWDS)

Description of Parameters:

NOWHMG	Index in control table for hypermatrices
IROWP	Row partition of the submatrix in the hypermatrix
ICOLP	Column partition of the submatrix in the hypermatrix
IROWS	Number of rows in submatrix
ICOLS	Number of columns in submatrix
NUMWDS	Number of words occupied by submatrix

Calls: None

Called by: Application Program

Common: /COMHMG/

E.8 HMGINF - Information about a hypermatrix

Purpose:

Retrieves specified control information about a hypermatrix

Usage: INTEGER HMGINF, IVALUE
:
IVALUE = HMGINF (NOWHM, INDX)

Description of Parameters:

NOWHMG	Index in control table for hypermatrices
INDX	Indicator for which control information parameter

Calls: None

Called by: Application Program

Common: /COMHMG/

E.9 HMGNDX - Calculates index

Purpose:

Calculates the index of a submatrix in the control information area.

Usage: INTEGER HMGNDX, IVALUE
:
IVALUE = HMGNDX (NOWHMG, IROWP, ICOLP)

Description of Parameters:

NOWHMG Index in hypermatrices table
 IROWP Row partition number of submatrix
 ICOLP Column partition number of submatrix

Calls: None

Called by: HMGPOT, HMGGET, HMGEVS, HMGEVG

Common: /COMHMG/

E.10 HMGEXT - Creates a submatrix

Purpose:

Creates a submatrix for a hypermatrix and inserts the pointer to it in the control information area.

Usage: CALL HMGEXT (NOWHMG, IROWP, ICOLP, ISPTR)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROWP Row partition of the submatrix in the hypermatrix
 ICOLP Column partition of the submatrix in the hypermatrix
 ISPTR Pointer to submatrix in dynafile

Calls: HMGNDX, HMGSIZ, DACRET, DAPUTV

Called by: HMGGET, HMGPOT, HMGEVS, HMGEVG

Common: /COMHMG/

E.11 HMGDEL - Delete a hypermatrix

Purpose:

Deletes the submatrices and the control information area of a hypermatrix.

Usage: CALL HMGDEL (NOWHMG)

Description of Parameters:

NOWHMG Index in control table for hypermatrices

Calls: INFINT, HMGNDX, HMGSIZ, DAGETV, DAREL, DAULAD, DADEL

Called by: Application program

Common: /COMHMG/

E.12 HMGEVS - Store a value in a hypermatrix

Purpose:

Stores the value of an element in a hypermatrix.

Usage: CALL HMGEVS (NOWHMG, IROW, ICOL, IVALUE, NELTWD)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROW Row index of element in hypermatrix
 ICOL Column index of element in hypermatrix
 IVALUE Element value
 NELTWD Number of words in element

Calls: INFINT, DALOC, HMGNDX, HMGLOC, HMGsiz, DAGETV, HMGEXT
 DAHEX, DAPUTV

Called by: Application program

Common: /COMHMG/

E.13 HMGHVG - Retrieve a value in a hypermatrix

Purpose:

Retrieves the value of an element in the hypermatrix.

Usage: CALL HMGHVG (NOWHMG, IROW, ICOL, IVALUE, NELTWD)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROW Row index of element in hypermatrix
 ICOL Column index of element in hypermatrix
 IVALUE Element value
 NELTWD Number of words in element

Calls: INFINT, DALOC, HMGNDX, HMGLOC, HMGsiz, DAGETV, HMGEXT
 DAHEX, DAPUTV

Called by: Application program

Common: /COMHMG/

E.14 HMGLOC - Calculate element location in hypermatrix

Purpose:

HMGLOC determines in which submatrix partition a hypermatrix element is in and what the offset from the first one is.

Usage: CALL HMGLOC (NOWHMG, IROW, ICOL, IROWP, ICOLP, IOFSET)

Description of Parameters:

NOWHMG Index in control table for hypermatrices
 IROW Row index of element in hypermatrix
 ICOL Column index of element in hypermatrix
 IROWP Row partition number of submatrix
 ICOLP Column partition number of submatrix
 IOFSET Number of words from first element in submatrix

Calls: HMGsiz

Called by: HMGHVS, HMGHVG

Common: /COMHMG/