AN ALGORITHM FOR ASSEMBLY OF STIFFNESS MATRICES INTO A COMPACTED DATA STRUCTURE

September 1984

An Investigation Conducted by:
DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
Berkeley, CA 95663

N00014-76-C-0013
N62583/83 M T265

Approved for public release; distribution in unlimited.
### Metric Conversion Factors

#### Approximate Conversions to Metric Measures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>When You Know</th>
<th>Multiply by</th>
<th>To Find</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LENGTH</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>inches</td>
<td>2.5</td>
<td>centimeters</td>
</tr>
<tr>
<td>ft</td>
<td>feet</td>
<td>30</td>
<td>centimeters</td>
</tr>
<tr>
<td>yd</td>
<td>yards</td>
<td>0.9</td>
<td>meters</td>
</tr>
<tr>
<td>m</td>
<td>miles</td>
<td>1.6</td>
<td>kilometers</td>
</tr>
<tr>
<td><strong>AREA</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in²</td>
<td>square inches</td>
<td>6.5</td>
<td>square centimeters</td>
</tr>
<tr>
<td>ft²</td>
<td>square feet</td>
<td>0.09</td>
<td>square meters</td>
</tr>
<tr>
<td>yd²</td>
<td>square yards</td>
<td>0.8</td>
<td>square meters</td>
</tr>
<tr>
<td>ac</td>
<td>square miles</td>
<td>2.6</td>
<td>square kilometers</td>
</tr>
<tr>
<td><strong>MASS (weight)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>oz</td>
<td>ounces</td>
<td>28</td>
<td>grams</td>
</tr>
<tr>
<td>lb</td>
<td>pounds</td>
<td>0.45</td>
<td>kilograms</td>
</tr>
<tr>
<td></td>
<td>short tons</td>
<td>0.9</td>
<td>tonnes (2,000 lb)</td>
</tr>
<tr>
<td><strong>VOLUME</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tsp</td>
<td>teaspoons</td>
<td>5</td>
<td>milliliters</td>
</tr>
<tr>
<td>Tbsp</td>
<td>tablespoons</td>
<td>15</td>
<td>milliliters</td>
</tr>
<tr>
<td>fl oz</td>
<td>fluid ounces</td>
<td>30</td>
<td>milliliters</td>
</tr>
<tr>
<td>c</td>
<td>cups</td>
<td>0.24</td>
<td>liters</td>
</tr>
<tr>
<td>pt</td>
<td>pints</td>
<td>0.47</td>
<td>liters</td>
</tr>
<tr>
<td>qt</td>
<td>quarts</td>
<td>0.95</td>
<td>liters</td>
</tr>
<tr>
<td>gal</td>
<td>gallons</td>
<td>3.8</td>
<td>liters</td>
</tr>
<tr>
<td>ft³</td>
<td>cubic feet</td>
<td>0.03</td>
<td>cubic meters</td>
</tr>
<tr>
<td>yd³</td>
<td>cubic yards</td>
<td>0.76</td>
<td>cubic meters</td>
</tr>
<tr>
<td><strong>TEMPERATURE (exact)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>°F</td>
<td>Fahrenheit</td>
<td>5.9</td>
<td>Celsius</td>
</tr>
<tr>
<td></td>
<td>temperature</td>
<td>subtracting</td>
<td>temperature</td>
</tr>
</tbody>
</table>

#### Approximate Conversions from Metric Measures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>When You Know</th>
<th>Multiply by</th>
<th>To Find</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LENGTH</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mm</td>
<td>millimeters</td>
<td>0.04</td>
<td>inches</td>
</tr>
<tr>
<td>cm</td>
<td>centimeters</td>
<td>0.4</td>
<td>inches</td>
</tr>
<tr>
<td>m</td>
<td>meters</td>
<td>3.3</td>
<td>feet</td>
</tr>
<tr>
<td>m</td>
<td>meters</td>
<td>1.1</td>
<td>yards</td>
</tr>
<tr>
<td>m</td>
<td>kilometers</td>
<td>0.6</td>
<td>miles</td>
</tr>
<tr>
<td><strong>AREA</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cm²</td>
<td>square centimeters</td>
<td>0.16</td>
<td>square inches</td>
</tr>
<tr>
<td>m²</td>
<td>square meters</td>
<td>1.2</td>
<td>square yards</td>
</tr>
<tr>
<td>ha</td>
<td>hectares (1000 m²)</td>
<td>2.5</td>
<td>acres</td>
</tr>
<tr>
<td><strong>MASS (weight)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>grams</td>
<td>0.035</td>
<td>ounces</td>
</tr>
<tr>
<td>kg</td>
<td>kilograms</td>
<td>2.2</td>
<td>pounds</td>
</tr>
<tr>
<td>t</td>
<td>tonnes (1,000 kg)</td>
<td>1.1</td>
<td>short tons</td>
</tr>
<tr>
<td><strong>VOLUME</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ml</td>
<td>milliliters</td>
<td>0.03</td>
<td>fluid ounces</td>
</tr>
<tr>
<td>l</td>
<td>liters</td>
<td>2.1</td>
<td>pints</td>
</tr>
<tr>
<td>qt</td>
<td>quarts</td>
<td>1.06</td>
<td>pints</td>
</tr>
<tr>
<td>gal</td>
<td>gallons</td>
<td>0.26</td>
<td>quarts</td>
</tr>
<tr>
<td>ft³</td>
<td>cubic feet</td>
<td>36</td>
<td>cubic yards</td>
</tr>
<tr>
<td>m³</td>
<td>cubic meters</td>
<td>1.3</td>
<td>cubic yards</td>
</tr>
<tr>
<td><strong>TEMPERATURE (exact)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>°C</td>
<td>Celsius</td>
<td>9/5 (then)</td>
<td>Fahrenheit</td>
</tr>
<tr>
<td></td>
<td>temperature</td>
<td>add 32</td>
<td>temperature</td>
</tr>
</tbody>
</table>

*In °C = 2.54 °F + 32 for other exact conversions and more detailed tables, see NBS
Mead Publ. 286, Tables of Weights and Measures, Price 92.75. SO Catalog No. 01-34-10-286.
# An Algorithm for Assembly of Stiffness Matrices into a Compacted Data Structure

## Abstract

A data structure is described that stores only the non-zero terms of the assembled stiffness matrix. This storage scheme results in considerable reduction in memory demand during the assembly phase of a finite element program. Therefore, larger matrices can be formed in the main memory of the computer. When secondary storage must be used this approach reduces the...
I/O cost during the assembly stage.

An algorithm is derived that starts with the element connectivity information and generates the compacted data structure. The element matrices are then assembled to form the stiffness matrix with this storage scheme. The assembly algorithm is described and a FORTRAN listing of the routines are presented. The reduction in storage is demonstrated with the aid of numerical examples.
CONTENTS

Page

INTRODUCTION .................................................. 2

STORAGE SCHEME ............................................... 3

DERIVATION OF THE ASSEMBLY PROCESS ....................... 4

NUMERICAL RESULT ............................................. 8

CONCLUSION ..................................................... 9

REFERENCES .................................................... 13

APPENDIX A. Program Listing ................................ 14

List of Figures

Figure 1. Finite element mesh of example 1 .................. 10

Figure 2. Nodal graph for the mesh of example 1 .......... 10

Figure 3. The graph of unknowns for the mesh of example 1 11

List of Tables

Table 1. Connectivity sets and active degrees of freedom for the mesh in Fig. 1 .......................... 5

Table 2. The result of algorithm for establishing the row index of the nonzero terms in K for example 1 .. 6

Table 3. The row indices of the nonzero terms in the upper triangular part of K ............................. 7

Table 4. Comparison of the storage demands of profile and compacted assembly for K in examples 2 to 6 ...... 8

Table 5. Estimated storage needs for each scheme on regular mesh. \( \eta \) is the number of equations .... 9

Table 6. Description of test examples ........................ 12
Introduction

The analysis of large structures, especially in three dimensions, can result in stiffness matrices that demand an exceptionally large amount of computer storage. The storage needs of these matrices depend to a large extent on their sparsity and the data structure that is used to store them. The choice of the data structure in turn depends on the method that is used to solve the associated system of equations. Presently, most solution schemes used in finite element computer programs are based on direct methods, i.e. triangular factorization of the stiffness matrix, $K$. Starting from a given mesh description, a finite element program performs the following steps:

1. determine the sparsity structure of $K$,
2. renumber the equations to reduce the storage demands of $K$,
3. reserve the required storage for $K$,
4. compute the element matrices,
5. assemble the element matrices into $K$,
6. compute the triangular factorization of $K$,
7. solve the associate system of equations.

In many applications the available primary memory is not sufficient to store the assembled matrix, and therefore secondary storage is used. In this circumstance, steps 4 through 7 involve data transfers between primary and secondary store, often referred to as I/O. In this case, $K$ is partitioned into blocks and each block is assembled and stored on secondary store. The blocks are then brought back into the main memory to form the factors of $K$. For large enough problems the I/O costs can dominate the computation costs.

A great deal of effort has been expended to develop new procedures for reordering the equilibrium equations, thus reducing the overall storage requirements in the solution steps 5, 6 and 7 \cite{1,2,3}. This is motivated by the fact that reduction in storage translates directly into a savings in the I/O costs. Among the many solution schemes used, the frontal method \cite{4} and the profile or skyline method \cite{1} are probably the most popular. In \cite{8} it is shown that when the same nodal
elimination ordering is used the profile method performs the same number of operations as the frontal method; in [5] an algorithm is described that delivers a good frontal node ordering for the profile method. The significant difference between these methods is that the frontal method often combines steps 4, 5 and 6. In this way the I/O during the assembly step is overlapped with the I/O in the factorization step; thus, the frontal method results in a saving that is equal to the cost of I/O in step 4. Alternatively, the assembly of a profile stored matrix which is partitioned into blocks requires a multiple pass through the elements to perform the assembly. The principle differences in the I/O costs of the two methods during factorization may be traced to the above differences in the partitioning of the matrix into a frontal or a profile form. The principle disadvantage of the frontal method is the added overhead to retain a small front width during the triangular factorization in step 6 and subsequent resolutions in step 7. For example, in resolution of equations this added overhead may lead to CPU costs which are several times those of a profile stored resolution.

In this paper we take a different approach. We use the following simple observation:

The assembly process is independent of the solution procedure.

In other words, one should use the most efficient data structure for the assembly process, step 5, and then restructure the data for one’s favorite solution scheme, i.e., either frontal or profile. In this way one can achieve the same reductions in I/O as the frontal method and at the same time maintain high modularity of the program. Here we develop a data structure that stores only the nonzero terms in the stiffness matrix in a compacted form, and present an algorithm for the assembly of K for this storage method. This approach results in considerable reduction in the storage needs during the assembly process. Therefore large matrices often may be fully assembled in-core resulting in a considerable reduction in I/O.

This approach has the added advantage that the program is not built around a single equation solver. One can have many solution procedures by simply expanding the compacted structure of K into a form appropriate for each particular solution method. Furthermore, the compacted structure can be used directly for iterative solution techniques such as the conjugate
gradient type methods [6,7].

Storage Scheme

We now describe the compacted structure that is used to store $K$. We only consider symmetric matrices, although the extension to the nonsymmetric case is trivial, and store only the upper triangular part of $K$ is stored. The diagonal terms of $K$ will be stored separately in a single array of length $n$, where $n$ is the total number of equations. The remaining off-diagonals will be placed in a second array of length $r$, where $r$ is the total number of nonzero off-diagonal terms in the upper triangular part of $K$. All the elements in the same column will be placed consecutively in this array, starting from the top of the column down to the diagonal (excluding the diagonal term). The columns are stored consecutively from the second to the last. For each entry in this array we store its row number in a corresponding integer array of length $r$. The example below demonstrates the final storage scheme.

Example:

Consider the matrix

$$
K = \begin{bmatrix}
1 & 9 & 12 & 15 \\
2 & 10 & 11 & 13 & 14 \\
3 & 16 & 17 \\
4 & 5 & 19 & 20 \\
6 & 7 & 22 & 23 \\
8 & & & & \\
\end{bmatrix}
$$

The array $\text{diag}$ contains the diagonal terms of $K$ as shown below.

$$
\text{diag} = [1, 2, 3, 4, 5, 6, 7, 8]
$$

A real array then stores the off-diagonal terms of $K$ and a corresponding integer array denotes the row number of each off-diagonal term as shown below.

$$
\text{off-diag} = [9, 10, 11, 12, 16, 13, 17, 18, 14, 19, 20, 15, 21, 22, 23]
$$

$$
\text{irow} = [1, 2, 1, 3, 2, 3, 5, 2, 4, 5, 1, 4, 6, 7]
$$

In addition, a single integer array of length $n$ is also required to point to the end of entries from a given column. For the above example this array is
The total storage requirement is $r + n$ real words and $r + n$ integer words. Using a 16 bit integer word, $\frac{5}{4}(n + r)$ real words (64 bit) will be sufficient. Then the largest number of equations that can be solved this way is $2^{16} - 1 \approx 32000$. With a 32 bit integer word, the total storage required will be $\frac{3}{2}(n + r)$ real words.

**Derivation of the Assembly Process**

In this section we give a step by step derivation of the assembly algorithm. Each step is demonstrated with the aid of the mesh example in Fig. 1. First we introduce some notation. A finite element mesh is denoted by $M = \{E, N\}$ where $E$ and $N$ represent the collection of elements and nodes in the mesh. A part of the input information provided to a finite element program is the set of nodes belonging to an element. This we denote as $NCN$. For example element 4 in Fig. 1 has the connectivity set $N_4 = \{5, 8, 9, 6\}$. In Table 1 we give the complete list of the connectivity sets $N$, for each element in example 1. These data are usually assembled in a single array known as the connectivity array. The complete set $\{N_\forall e \in E\}$ is sufficient to describe the connectivity of a given mesh. Another part of the input data is the boundary conditions that determine the set of the indices of all the active degrees of freedom at node $p$. We denote this set as $U_p$. In Table 1 we give the set $\{U_p \forall p \in N\}$ for the example in Fig. 1. In this example, we assume that there are two degree of freedom at each node. The collection of $N$, column 2, and $U_p$, column 4, given in Table 1 is sufficient to determine the sparsity structure of the stiffness matrix associated with a given mesh.
Table 1. Connectivity sets and active degrees of freedom for the mesh in Fig. 1.

Our objective here is to find the set of indices of the unknowns that are coupled with a given degree of freedom. This is precisely the row number of each nonzero term in a given column of the stiffness matrix, $i_{row}$, that is required for the storage scheme described in the previous section.

First, we must establish the set of elements that are connected to each node. This can be done by inspecting the element connectivity sets. Looking at the second column of Table 1, for example node 4 appears twice, in rows 1 and 3. We then conclude that node 4 is connected to elements 1 and 3. This process must be repeated for each node. The difficulty here is that we do not know a priori the number of storage locations needed to identify the set of elements for each node. For this reason the above process is carried out in two steps. The number of elements connected to each node is determined and stored first. We refer to this as the E-degree (element degree) of each node. In the example the E-degree of node 4 is 2. The E-degree also determines the length of the array that is required to keep the set of elements connected to each node. For each node $p$ we denote this set by $E_p \subset E$. Then the above process is simply to evaluate

$$E_p = \{ e \mid p \in N_e \}$$

for each node $p$. This equation may be thought of as finding the pseudo-inverse of the connectivity array. The E-degree of node $p$ is the number of terms in $E_p$ (the cardinality of $E_p$). See
columns two and three of Table 2 for the E-degree and the complete set of \( E_i \) for the nodes in example 1.

Next, we find for the set of nodes that are adjacent to each node \( p \). We denote this by \( A_p \subset N \). This is the set of all nodes that belong to an element with \( p \) as one of its nodes. Having established the set of elements connected to node \( p \), the adjacent nodes are all the other nodes belonging to these elements. In example 1 node 4 is connected to elements 1 and 3 (see column 2 of Table 2). The set of nodes belonging to elements 1 and 3 are obtained by inspecting column 2 of Table 1; these are \( \{ 1, 4, 5, 2 \} \) and \( \{ 4, 7, 8, 5 \} \) respectively. Then the set of nodes adjacent to 4 is \( A_4 = \{ 1, 5, 2, 7, 8 \} \). The N-degree (nodal degree) of a node is the number of nodes adjacent to it and is the cardinality of \( A_p \). In columns 4 and 5 of Table 2 we give the N-degree and the adjacency set of each node in the nodal graph of example 1 (Fig. 2). This step is simply to evaluate the equation

\[ A_p = \bigcup_{e \in E_p} N_e - n \]  

(2)

Note that both \( A_p \) and N-degree can be obtained in the same loop.

<table>
<thead>
<tr>
<th>Node ( p )</th>
<th>E degree</th>
<th>( E_p )</th>
<th>N degree</th>
<th>( A_p )</th>
<th>S degree</th>
<th>( S_p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>{ 1 }</td>
<td>3</td>
<td>{ 4, 5, 2 }</td>
<td>0</td>
<td>( \phi )</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{ 1, 2 }</td>
<td>5</td>
<td>{ 1, 4, 5, 6, 3 }</td>
<td>1</td>
<td>{ 1 }</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>{ 2 }</td>
<td>3</td>
<td>{ 2, 5, 6 }</td>
<td>1</td>
<td>{ 2 }</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>{ 1, 3 }</td>
<td>5</td>
<td>{ 1, 2, 5, 7, 8 }</td>
<td>2</td>
<td>{ 1, 2 }</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>{ 1, 3, 4, 2 }</td>
<td>8</td>
<td>{ 1, 4, 7, 8, 9, 6, 3, 2 }</td>
<td>4</td>
<td>{ 1, 4, 3, 2 }</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>{ 2, 4 }</td>
<td>5</td>
<td>{ 2, 5, 8, 9, 3 }</td>
<td>3</td>
<td>{ 2, 5, 3 }</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>{ 3, 5, 6 }</td>
<td>5</td>
<td>{ 10, 11, 8, 5, 4 }</td>
<td>2</td>
<td>{ 4, 5 }</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>{ 3, 5, 7, 4 }</td>
<td>6</td>
<td>{ 11, 9, 6, 5, 4, 7 }</td>
<td>4</td>
<td>{ 6, 5, 4, 7 }</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>{ 4, 7, 8 }</td>
<td>5</td>
<td>{ 11, 12, 6, 5, 8 }</td>
<td>3</td>
<td>{ 6, 5, 8 }</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>{ 6 }</td>
<td>2</td>
<td>{ 7, 11 }</td>
<td>1</td>
<td>{ 7 }</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>{ 6, 5, 7, 8 }</td>
<td>5</td>
<td>{ 10, 12, 9, 8, 7 }</td>
<td>4</td>
<td>{ 10, 9, 8, 7 }</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>{ 8 }</td>
<td>2</td>
<td>{ 11, 9 }</td>
<td>2</td>
<td>{ 11, 9 }</td>
</tr>
</tbody>
</table>

Table 2. The result of algorithm for establishing the row index of the nonzero terms in \( K \) for example 1.
Since we want to store only the upper triangular part of $K$ we need to store only a subset of $A_p$. This will be the set of nodes in $A_p$ with an index less than $p$; that is $S_p = \{ i \mid i \in A_p \text{ and } i < p \}$. We refer to the number of terms in $S_p$ as the S-degree (or minor-degree) of a node. The set $S_p$ is only useful when the numbering of the unknowns are such that when $i < j$ all the unknowns at node $i$ have a smaller index than the unknowns at node $j$. Whenever this is not true it is necessary to use the complete set of adjacent nodes $A_p$ together with the numbers of the unknowns for each node (e.g., see listing in Appendix A).

Finally, for a given unknown at node $p$ with index $j \in U_p$ we find the set of the indices of all other unknowns that are coupled with $u_j$. This will be the set of row indices $R_j$ for nonzeros in $j$-th column of $K$. Then

$$R_j = \bigcup_{i \in A_p} U_i$$

(3)

For example 1 $R_j$ is the adjacency set of $j$ in the graph of the unknowns in Fig. 3. Since we only store the upper triangular part of $K$ we scan through $R_j$ and use the subset defined by:

$$R_j = \{ i \mid i \in R_j \text{ and } i < j \}$$

(4)

$R_j$ is the row index of all the nonzero terms in the $j$-th column of the upper triangular part of $K$.

The complete set of $R_j$ for the example problem is presented in Table 3.

<table>
<thead>
<tr>
<th>D.O.F.</th>
<th>Node</th>
<th>$R_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td>$p$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>${ 0 }$</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>${ 1 }$</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>${ 1, 2 }$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>${ 3, 2, 1 }$</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>${ 4, 3, 1 }$</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>${ 3, 4, 2 }$</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>${ 6, 2, 4, 5, 3 }$</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>${ 2, 3, 4, 5, 6, 7 }$</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>${ 7, 4, 5, 3, 8 }$</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>${ 9, 6, 7, 8 }$</td>
</tr>
</tbody>
</table>

Table 3. The row indices of the nonzero terms in the upper triangular part of $K$. 
The listing of a FORTRAN program that performs all the steps that is described in this section is given in Appendix A. In this Appendix we also provide the subroutine that uses the row indices to perform the assembly of $K$.

**Numerical Result**

We use the algorithm described in the previous section to assemble the stiffness matrices of the problems described in Table 6. The total storage required during the assembly step is evaluated. We compare these results to similar results obtained when the assembly is performed directly into a profile data structure. The storage requirement of the compacted assembly is not affected by the node ordering. For the assembly into a profile form, we numbered the nodes across the width of the mesh to reduce the bandwidth of the stiffness matrix. Although, the bandwidth could have been reduced further using a renumbering scheme such as [2,3], we omitted this step for simplicity. The results for examples 2 to 6, given in Table 6, are presented in Table 4 below.

<table>
<thead>
<tr>
<th>Description of Problem</th>
<th>No. of nodes</th>
<th>No. of elements</th>
<th>No. of equations</th>
<th>Stores for profile</th>
<th>Stores for Compacted $K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cantilever type structure</td>
<td>225</td>
<td>184</td>
<td>428</td>
<td>10204</td>
<td>5340</td>
</tr>
<tr>
<td>Small Cylinder structure</td>
<td>231</td>
<td>200</td>
<td>440</td>
<td>10492</td>
<td>5679</td>
</tr>
<tr>
<td>Large Cylinder structure</td>
<td>496</td>
<td>450</td>
<td>960</td>
<td>32542</td>
<td>12715</td>
</tr>
<tr>
<td>$4 \times 4 \times 4$ solid structure</td>
<td>125</td>
<td>64</td>
<td>300</td>
<td>21945</td>
<td>11634</td>
</tr>
<tr>
<td>$8 \times 8 \times 8$ solid structure</td>
<td>729</td>
<td>512</td>
<td>1944</td>
<td>470043</td>
<td>94272</td>
</tr>
</tbody>
</table>

Table 4. Comparison of the storage demands of profile and compacted assembly for $K$ in examples 2 to 6.

The results in Table 4 is obtained based on the assumption that a real word is twice as long as an integer word. We observe a reduction from 40% for two dimensional (2-D) problems to 80% for 3-D problems for these examples. The reductions will be more if short integer words are used. It is interesting to note that the required storage for compacted structure varies linearly with the number of equations. Therefore, the saving will be more for larger problems. In Table 5 we give
the storage counts for the two methods considered here on square mesh in 2-D and cubes in 3-D as a function of the number of equations. To obtain these estimates we assumed that there is only one degree of freedom per node.

<table>
<thead>
<tr>
<th>Dimension of Problem</th>
<th>Half bandwidth</th>
<th>Profile storage</th>
<th>Compacted storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>$5/2 , n$</td>
<td>$3n$</td>
</tr>
<tr>
<td>2</td>
<td>$n^{1/2}$</td>
<td>$n^{3/2}$</td>
<td>$15/2 , n$</td>
</tr>
<tr>
<td>3</td>
<td>$n^{2/3}$</td>
<td>$n^{5/3}$</td>
<td>$21n$</td>
</tr>
</tbody>
</table>

Table 5. Estimated storage needs for each scheme on regular mesh. $n$ is the number of equations.

Conclusion

The essential steps in a finite element program can be modified to make use of the compacted assembly described here. Accordingly, we perform the following steps:

1. Obtain the row indices of the nonzero terms in $K$.
2. Assemble the matrix in compact form.
3. Choose a solution procedure and renumber the equations to reduce the storage demands of the factors of $K$.
4. Expand the compacted $K$ into a data structure suitable for the solution method.
5. Solve the associated system of equations.

When there is insufficient primary storage, the assembly of the matrix in compacted form opens a number of possible avenues that one can take to reduce the I/O cost. The expansion of the compacted form need not be done immediately after its assembly. The matrix can be kept in compact form and put on secondary store and expanded into a full profile form only when a factorization must be performed. This way the number of data entries that is read (in the input phase of I/O) can be reduced considerably, which in turn results in a reduction in the solution time.
Figure 1. Finite element mesh of example 1.

Figure 2. Nodal graph for the mesh of example 1.
Figure 3. The graph of unknowns for the mesh of example 1.
<table>
<thead>
<tr>
<th>Example 2: Cantilever Structure, left end fixed, plane stress elements with 2 degrees of freedom per node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Cantilever Structure" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 3: Small Cylinder, both ends fixed in tangential direction, plane strain elements with 2 degrees of freedom per node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image2.png" alt="Small Cylinder" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 4: Large Cylinder, both ends fixed in tangential direction, plane strain elements with 2 degrees of freedom per node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Large Cylinder" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 5: 4X4X4 Solid cube, fixed base, solid elements with 3 degrees of freedom per node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4.png" alt="4X4X4 Solid cube" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 6: 8X8X8 Solid cube, fixed base, solid elements with 3 degrees of freedom per node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.png" alt="8X8X8 Solid cube" /></td>
</tr>
</tbody>
</table>

Table 6. Description of test examples.
References


SUBROUTINE ELCNT(NUMNP, NUMEL, NEN, NENI, IX, IC)
DIMENSION IX(NENI, 1), IC(1)

C... INPUT
NUMNP TOTAL NO. OF NODES IN THE MESH
NUMEL TOTAL NO. OF ELEMENTS IN THE MESH
NEN MAX. NO. OF NODES PER ELEMENT
NENI DIMENSION OF IX ARRAY
IX ELEMENT CONNECTIVITY ARRAY

C... OUTPUT
ARRAY OF LENGTH NUMNP. IT FIRST HOLDS THE ELEMENT DEGREE
OF EACH NODE, THEN BECOMES A POINTER FOR AN ARRAY THAT
CONTAINS THE SET OF ELEMENTS CONNECTED TO EACH NODE.

C... COUNT THE NUMBER OF ELEMENTS EACH NODE BELONGS TO
CALL IZERO(IC, NUMNP)
DO 110 N = 1, NUMEL
   DO 100 J = 1, NEN
      I = IX(J, N)
      IF (I.GT.0) IC(I) = IC(I) + 1
 100 CONTINUE
110 CONTINUE
C... SET UP POINTERS
DO 120 I = 2, NUMNP
   IC(I) = IC(I) + IC(I-1)
120 CONTINUE
RETURN
END

SUBROUTINE CASSEM(D, A, B, S, P, JCOLE, IROW, LD, ID, NSTNEL, AFL, BFL)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
LOGICAL AFL, BFL
DIMENSION D(1), A(1), B(1), S(NSTNEL), P(1), JCOLE(1), IROW(1), LD(1), ID(1)

C... COMPACT ASSEMBLY OF PROFILE MATRIX
DO 200 J = 1, NSTNEL
   N = LD(J)
   IF (AFL .AND. N .GT. 1) THEN
      DO 150 I = 1, N
         K = LD(I)
         IF (K.GT.0 .AND. K.LT. N) THEN
            INZ = INZA(JCOLE(I)+1, JCOLE(I), IROW, K)
            A(INZ) = A(INZ) + S(I, J)
         END IF
      150 CONTINUE
   END IF
200 CONTINUE
RETURN
END
SUBROUTINE CC3URO(NUMNP, NUMEL, NEN, NEN1, NDF, IX, ID, IC, IROW, IELC, IJCOLE, KP)

DIMENSION IX(NEN1, 1), ID(NDF, 1), IC(1), IROW(1), IELC(1), JCOLE(1)

FOR (NUMNP, NUMEL, NEN, NEN1, IX, IC) SEE SUBROUTINE ELCNT

INPUT
NDF: NUMBER OF UNKNOWNS AT EACH NODE
ID: ACTIVE UNKNOWNS AT EACH NODE

OUTPUT
IELC: HOLDS THE SET OF ELEMENTS CONNECTED TO EACH NODE
IROW: ROW NUMBER OF EACH NONZERO IN THE STIFFNESS MATRIX

JCOLE: END OF ENTRIES IN IROW FORM A GIVEN COLUMN

FIND ELEMENTS CONNECTED TO NODES

CALL IZERO (IELC, IC(NUMNP))
DO 230 N = 1, NUMEL
   DO 220 J = 1, NEN
      I = IX(J, N)
      IF ( I .GT. 0 ) THEN
         KP = IC(I)
         IF ( IELC(KP) .EQ. 0 ) GO TO 210
         KP = KP - 1
         GO TO 200
      END IF
      IELC(KP) = N
   CONTINUE
220 CONTINUE
230 CONTINUE

SET UP COMPRESSED PROFILE POINTERS

KP = 0
NEP = 1
DO 350 I = 1, NUMNP
   NE = IC(I)
   DO 340 J = 1, NDF
      NEQ = ID(II, I)
      IF ( NEQ .GT. 0 ) THEN
         JCOLE(NEQ) = KP
         KPO = KP + 1
         IF ( NEP .LE. NE ) THEN
            DO 330 N = NEP, NE
               NN = IELC(N)
               DO 320 J = 1, NEN
                  K = IX(J, NN)
                  DO 310 JJ = 1, NDF
                     NEQJ = ID(JJ, K)
                     IF ( NEQJ .LTE. NEQ .AND. NEQJ .LT. 0 ) GO TO 310
                  CONTINUE
               CONTINUE
            END IF
            IF ( KPO .LE. KP ) THEN
               DO 300 KK = KPO, KP
                  IF ( IROW(KK) .EQ. NEQJ ) GO TO 310
               CONTINUE
            END IF
         END IF
      END IF
310 CONTINUE
320 CONTINUE
330 CONTINUE
JCOLE(NEQ) = KP
END IF
340 CONTINUE
NEP = NE + 1
350 CONTINUE
RETURN
END
INTEGER FUNCTION INZA(N1,N2,IROW,K)
DIMENSION IROW(1)

C FIND THE TERM FOR THE ASSEMBLY
C
DO 100 N = N1,N2
   IF ( IROW(N) .EQ. K ) THEN
      INZA = N
      RETURN
   END
100 CONTINUE
C ERROR IF LOOP EXITS
STOP
END

SUBROUTINE IZERO(IA,NN)
DIMENSION IA(NN)
DO 100 N = 1,NN
   IA(N) = 0
100 CONTINUE
RETURN
END