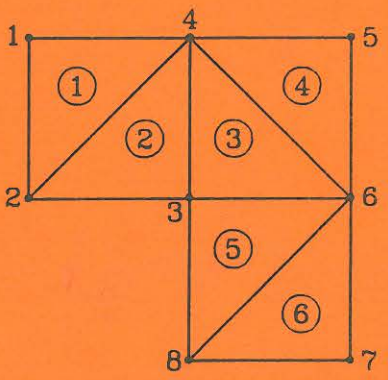
Solution of Finite Element Equations

Krenk, Steen

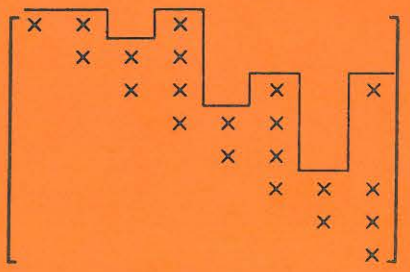
Publication date:
1991

Document Version
Publisher's PDF, also known as Version of record

INSTITUTTET FOR BYGNINGSTEKNIK
DEPT. OF BUILDING TECHNOLOGY AND STRUCTURAL ENGINEERING
AALBORG UNIVERSITETSCENTER • AUC • AALBORG • DANMARK



$$A_i =$$



S. KRENK
SOLUTION OF FINITE ELEMENT EQUATIONS
OCTOBER 1991
ISSN 0902-7513 R9137

SOLUTION OF FINITE ELEMENT EQUATIONS

Steen Krenk

October 1990

Department of Building Technology and Structural Engineering
University of Aalborg, Sohngaardsholmsvej 57
DK-9000 Aalborg

An important step in solving any problem by the finite element method is the solution of the global equations. Numerical solution of linear equations is a subject covered in most courses in numerical analysis. However, the equations encountered in most finite element applications have some special features that justify the development of specialized solution algorithms.

The special properties encountered in many finite element applications are illustrated by a simple example. The equivalence between the standard Gauss elimination procedure and triangular **LU** factorization is described. The symmetric $\mathbf{LDL}^T$ form is derived, and the algorithm is modified to account for boundary conditions. Finally, the profile - or skyline - storage scheme is developed in detail. Procedures for solution of symmetric systems of equations are included in a full-matrix as well as a profile version. The presentation is somewhat more detailed than that usually found in the finite element literature, see e.g. Bathe (1982) and Hughes (1987).

1. AN INTRODUCTORY EXAMPLE

Finite element equation systems are built from the contributions of individual elements. Usually the global equations are set up without any regard of the boundary conditions first, and the boundary conditions are then introduced as a modification of the full global matrix. The procedure is illustrated by the example shown in Fig. 1. A rectangular domain is modelled by four linear triangles. The side 1-2 has a linear heat source distribution and the side 5-6 is kept at constant temperature $T = 0$.

With constant conductivity k and thickness t of all elements the expanded element conductivity matrices are, Krenk (1989),

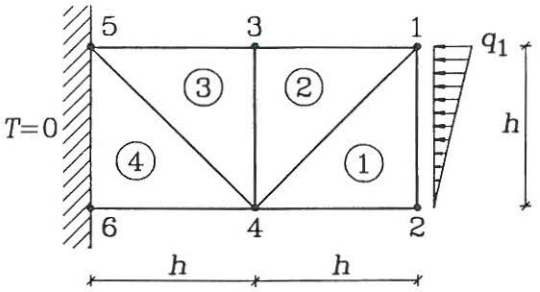


Fig. 1. Heat conduction by triangular elements.

$$[K_1^{ee}] = \frac{kt}{2}\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[K_2^{ee}] = \frac{kt}{2}\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[K_3^{ee}] = \frac{kt}{2}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[K_4^{ee}] = \frac{kt}{2}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

After assembly the conductivity matrix is

$$[K] = \sum_{j=1}^{4} [K_j^{ee}] = \frac{kt}{2}\begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 \\ -1 & 0 & 4 & -2 & -1 & 0 \\ 0 & -1 & -2 & 4 & 0 & -1 \\ 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

The load vector is

$$\{f\} = \frac{q_1 h}{6}\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In addition there are unknown loads at nodes 5 and 6 in order to maintain the prescribed temperature along 5-6. The resulting equations are

$$\frac{kt}{2}\begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 \\ -1 & 0 & 4 & -2 & -1 & 0 \\ 0 & -1 & -2 & 4 & 0 & -1 \\ 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \frac{q_1 h}{6}\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ Q_5 \\ Q_6 \end{bmatrix}$$

In terms of the non-dimensional variables $x_j = 3(kt/q_1 h)\, T_j, j = 1, \ldots, 6$ and $r_i = 6Q_i/q_1 h$, $i = 5, 6$, the system of equations is

$$\begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 \\ -1 & 0 & 4 & -2 & -1 & 0 \\ 0 & -1 & -2 & 4 & 0 & -1 \\ 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ r_5 \\ r_6 \end{bmatrix}$$

This system of equations has the following properties

- Some of the entries x_j are known a priori.
- The matrix is symmetric.
- The matrix has band structure.

These properties are characteristic of most finite element equation systems and must be dealt with specifically.

2. GAUSS ELIMINATION

Let a linear system of equation be given in the form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

$\mathbf{x}$ is a vector containing the n unknown parameters

$$\mathbf{x}^T = (x_1, \ldots, x_n) \tag{2}$$

$\mathbf{A}$ is an $n \times n$ matrix with elements a_{ij} and $\mathbf{b}$ is a known vector,

$$\mathbf{b}^T = (b_1, \ldots, b_n) \tag{3}$$

When written in full the equations (1) are

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & \dots & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{4}$$

Gauss elimination is a solution procedure in which the equation matrix is reduced to upper triangular form by successive row operations. Subsequently, the triangular system is solved by back substitution. The method is illustrated with reference to the introductory example.

Example 1. Gauss elimination

In Fig. 1 $T_5 = T_6 = 0$, and thus only (T_1, T_2, T_3, T_4) are unknown. Similarly, Q_5 and Q_6 are unknown, and therefore only the first 4 equations are used to determine $T_1, \dots, T_4$.

The equations are written in the form of an $n \times (n+1)$ matrix.

$$\left[\begin{array}{cccc|c} 2 & -1 & -1 & 0 & 2 \\ -1 & 2 & 0 & -1 & 1 \\ -1 & 0 & 4 & -2 & 0 \\ 0 & -1 & -2 & 4 & 0 \end{array}\right]$$

First the elements below the diagonal in the first column are zeroed by subtracting the first row multiplied by a suitable factor.

$$\left[\begin{array}{cccc|c} 2 & -1 & -1 & 0 & 2 \\ 0 & 3/2 & -1/2 & -1 & 2 \\ 0 & -1/2 & 7/2 & -2 & 1 \\ 0 & -1 & -2 & 4 & 0 \end{array}\right]$$

Now zeros are introduced in the elements below the diagonal in the second column by use of the second row.

$$\left[\begin{array}{cccc|c} 2 & -1 & -1 & 0 & 2 \\ 0 & 3/2 & -1/2 & -1 & 2 \\ 0 & 0 & 10/3 & -7/3 & 5/3 \\ 0 & 0 & -7/3 & 10/3 & 4/3 \end{array}\right]$$

Finally the single sub-diagonal element in column 3 is zeroed.

$$\left[\begin{array}{cccc|c} 2 & -1 & -1 & 0 & 2 \\ 0 & 3/2 & -1/2 & -1 & 2 \\ 0 & 0 & 10/3 & -7/3 & 5/3 \\ 0 & 0 & 0 & 17/10 & 5/2 \end{array}\right]$$

This completes the reduction to triangular form.

The solution is obtained starting from x_4. The last equation is

$$\frac{17}{10}x_4 = \frac{5}{2}$$

from which

$$x_4 = \frac{10}{17}\,\frac{5}{2} = \frac{25}{17}$$

The next equation from below is

$$\frac{10}{3}x_3 - \frac{7}{3}x_4 = \frac{5}{3}$$

At this stage x_4 is known, whereby

$$x_3 = \frac{3}{10}\left(\frac{5}{3} + \frac{7}{3}\,\frac{25}{17}\right) = \frac{26}{17}$$

Similarly the second and the first equation give

$$x_2 = \frac{2}{3}\left(2 + \frac{25}{17} + \frac{1}{2}\,\frac{26}{17}\right) = \frac{48}{17}$$

and

$$x_1 = \frac{1}{2}\left(2 + \frac{26}{17} + \frac{48}{17}\right) = \frac{54}{17}$$

When the unknown vector $\mathbf{x}$ has been determined the last two equations, that were originally disregarded, can be used to determine the two unknown source intensities (Q_5, Q_6). The last equations are

$$r_5 = -1x_3 + 2x_5 - 1x_6 = -\frac{26}{17}$$

$$r_6 = -1x_4 - 1x_5 + 2x_6 = -\frac{25}{17}$$

This completes the solution of the equations in the problem of Fig. 1.

* * *

The Gauss elimination procedure used in Example 1 is easily developed in the form of a general algorithm. It consists of two parts: forward reduction and back substitution. The forward reduction consists of $n-1$ steps $i = 1, 2, \ldots, n-1$. In step i zeros are introduced below the diagonal of column i. This requires updating of the lower part of the matrix $\mathbf{A}$ and the $\mathbf{b}$. The Gauss forward reduction is expressed in pseudo code in the following algorithm.

Algorithm: Gauss Forward Reduction

for $i := 1$ *to* $n-1$ *(step i)*

for $k := i+1$ *to* n *(row k)*

$l := a_{ki}/a_{ii}$

for $j := i+1$ *to* n *(column j)*

$a_{kj} := a_{kj} - l * a_{ij}$

$b_k := b_k - lb_i$

* * *

The result of this algorithm is a triangular equation system

$$\mathbf{U}\mathbf{x} = \mathbf{z} \tag{5}$$

U is an upper triangular matrix obtained by systematic modification of the original matrix **A**.

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \dots & \vdots \\ & & \dots & u_{nn} \end{bmatrix} \tag{6}$$

The vector $\mathbf{z}^T = (z_1, \dots, z_n)$ is obtained in the process by modification of the original vector **b**.

The back substitution part of Gauss elimination is the stepwise solution of the equation system (5), starting from the last equation. The pseudo code for back substitution is given below.

Algorithm: Gauss Back Substitution

```
for  i := n  down to  1                          (row i)
     for  j := i + 1  to  n                      (column j)
          b_i :=  b_i - a_ij * b_j
     b_i :=  b_i / a_ii
```

* * *

Note that the upper triangular matrix **U** is stored in the upper triangular part of **A**. Upon exit the solution **x** is contained in the vector **b**.

The Gauss elimination solution procedure is simple and direct. However, one disadvantage of the algorithms shown above is that the right hand side **b** is modified during both parts of the solution procedure. In practice

Gauss elimination is most often implemented in finite element programs in the form of matrix factorization.

Before proceeding it should be noticed that the forward reduction algorithm will break down, if a zero diagonal element a_{ii} is encountered. In practice even a small diagonal element may cause problems with loss of accuracy. Such problems are usually countered by equilibration or pivoting, see e.g. Press et al. (1986). Here the prime concern is with symmetric systems of equations, where these problems are not likely to arise. On the contrary any scaling or reordening of the equations may destroy the symmetry, and therefore only direct methods are dealt with here.

3. LU - FACTORISATION

In the solution of linear equations it is often advantageous to separate the forward reduction part from the back substitution part. The result of the forward reduction was to produce a system of equations in the triangular form (5). This system was obtained by simultaneous modification of the matrix **A** and the right hand side **b**. It is demonstated below that this modification can be obtained by decomposing the matrix **A** into two factors **L** and **U**, where **U** is the upper triangular matrix (6), and **L** is a lower triangular matrix with unit diagonal elements.

$$\mathbf{L} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & & & \\ l_{n1} & l_{n2} & \dots & \dots & 1 \end{bmatrix} \tag{7}$$

The factorisation amounts to determination of two triangular matrices **L** and **U** such that

$$\mathbf{A} = \mathbf{LU} \tag{8}$$

With this factorisation the equation system (1) takes the form

$$\mathbf{LUx} = \mathbf{b} \tag{9}$$

The matrix $\mathbf{L}$ is triangular, and it is therefore easy to reduce (9) to

$$\mathbf{U}\mathbf{x} = \mathbf{L}^{-1}\mathbf{b} = \mathbf{z} \tag{10}$$

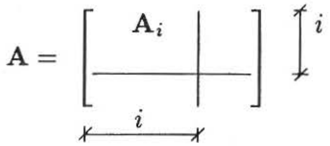
by direct manipulation, i.e. without explicit inversion of $\mathbf{L}$. The final solution $\mathbf{x}$ is obtained by the back substitution algorithm described above.

In the factorisation method of solution the following steps are used.

- Perform the factorisation $\mathbf{A} = \mathbf{L}\mathbf{U}$.
- Solve $\mathbf{L}\mathbf{z} = \mathbf{b}$ for $\mathbf{z}$ by direct manipulation.
- Solve $\mathbf{U}\mathbf{x} = \mathbf{z}$ for $\mathbf{x}$ by direct manipulation.

Note that the factorisation does not involve the right-hand side vector $\mathbf{b}$. The two last steps are usually jointly called back substitution or solution.

The factorisation property $\mathbf{A} = \mathbf{L}\mathbf{U}$ is proved by a constructive method, i.e. by actually doing it. Thereby, the necessary algorithm is obtained directly. Let $\mathbf{A}_i$ denote the upper $i \times i$ part of the $n \times n$ matrix $\mathbf{A}$



(11)

Similarly $\mathbf{L}_i$ and $\mathbf{U}_i$ denote the upper $i \times i$ part of $\mathbf{L}$ and $\mathbf{U}$, respectively. Now assume that the submatrix $\mathbf{A}_{i-1}$ has already been factored in the form

$$\mathbf{A}_{i-1} = \mathbf{L}_{i-1}\mathbf{U}_{i-1} \tag{12}$$

The submatrix $\mathbf{A}_i$ with one more row and column can then be factored as

$$\mathbf{A}_i = \mathbf{L}_i\mathbf{U}_i \tag{13}$$

where the triangular matrices $\mathbf{L}_i$ and $\mathbf{U}_i$ are of the form

$$\mathbf{L}_i = \left[\begin{array}{c|c} \mathbf{L}_{i-1} & \mathbf{0} \\ \hline \mathbf{l}_i^T & 1 \end{array}\right] \tag{14}$$

and

$$\mathbf{U}_i = \left[\begin{array}{c|c} \mathbf{U}_{i-1} & \mathbf{u}_i \\ \hline \mathbf{0}^T & d_i \end{array}\right] \tag{15}$$

The proof follows from writing the matrix $\mathbf{A}_i$ as

$$\mathbf{A}_i = \left[\begin{array}{c|c} \mathbf{A}_i & \mathbf{a}_i \\ \hline \mathbf{b}_i^T & c_i \end{array}\right] \tag{16}$$

and then substituting $\mathbf{L}_i$ and $\mathbf{U}_i$ into (13). The result is

$$\mathbf{A}_i = \left[\begin{array}{c|c} \mathbf{L}_{i-1}\mathbf{U}_{i-1} & \mathbf{L}_{i-1}\mathbf{u}_i \\ \hline \mathbf{l}_i^T\mathbf{U}_{i-1} & d_i + \mathbf{l}_i^T\mathbf{u}_i \end{array}\right] \tag{17}$$

Comparison with (16) gives the following equation for the new i'th row $\mathbf{l}_i^T$ of $\mathbf{L}_i$, the new i'th column $\mathbf{u}_i$ of $\mathbf{U}_i$, and the new diagonal element d_i.

$$\mathbf{L}_{i-1}\mathbf{u}_i = \mathbf{a}_i \tag{18}$$

$$\mathbf{U}_{i-1}^T\mathbf{l}_i = \mathbf{b}_i \tag{19}$$

$$d_i + \mathbf{l}_i^T\mathbf{u}_i = c_i \tag{20}$$

The equations (18) and (19) for $\mathbf{u}_i$ and $\mathbf{l}_i$ are in lower triangular form, and can therefore be solved directly.

The three relations (18) - (20) comprise step i of the factorisation algorithm. In step i the factored part is increased in size from $(i-1)\times(i-1)$ to $i \times i$ by adding a new row and column. Note that the new row and column do not change the parts, $\mathbf{L}_{i-1}$ and $\mathbf{U}_{i-1}$, that have already been factored. Thus only row i and column i are "active" in step i. This is

illustrated in Fig. 2. The part $\mathbf{A}_{i-1}$ of the original matrix, corresponding to the factored part, is not needed in the further steps, and in most implementations $\mathbf{L}_i$ and $\mathbf{U}_i$ are written into the corresponding storage positions of $\mathbf{A}_i$ as they are created. This is shown in the following **LU** factorisation algorithm.

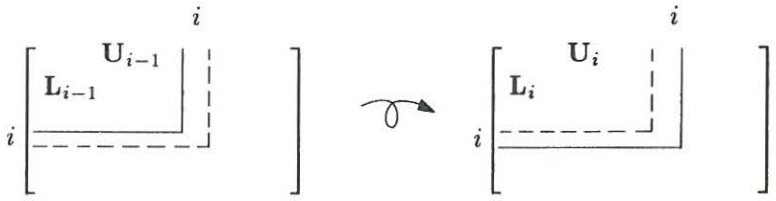


Fig. 2. Progress of front in step i of **LU**-factorisation.

Algorithm: LU-factorisation.

for $i := 2$ *to* n (*step* i)

for $j := 1$ *to* $i - 1$ (*eqn.* j)

for $k := 1$ *to* $j - 1$

$a_{ji} := a_{ji} - a_{jk} * a_{ki}$ (u_{ji} *from* (18))

$a_{ij} := a_{ij} - a_{kj} * a_{ik}$ } (l_{ij} *from* (19))

$a_{ij} := a_{ij}/a_{jj}$

$a_{ii} := a_{ii} - a_{ij} * a_{ji}$ (u_{ii} *from* (20))

* * *

Upon exit from the **LU**-factorisation algorithm **L** and **U** are available in the corresponding storage locations of the matrix **A**. The back substitution consists of the following two steps: find **z** from

$$\mathbf{Lz} = \mathbf{b} \tag{21}$$

and then find **x** from

$$\mathbf{Ux} = \mathbf{z} \tag{22}$$

The corresponding pseudo code can be written down directly.

Algortithm: LU solution.

for $i := 2$ *to* n (*eqn.* i *of* (21))

for $j := 1$ *to* $i - 1$

$b_i := b_i - a_{ij} * b_j$

for $i := n$ *down to* 1 (*eqn.* i *of* (22))

for $j := i + 1$ *to* n

$b_i := b_i - a_{ij} * b_j$

$b_i := b_i / a_{ii}$

* * *

The first part of the **LU** solution algorithm places **z** in the storage positions of **b**, and the second part replaces **z** with **x**. It is notable that the **LU** factorisation algorithm replaces the matrix **A** with the two triangular factors **L** and **U** "in place", i.e. without need for any additional storage. Similarly the **LU** solution algorithm replaces **b** by **x** "in place".

Example 2. LU factorisation.

The upper 4×4 part of the matrix from the problem of Fig. 1 is

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 4 & -2 \\ 0 & -1 & -2 & 4 \end{bmatrix}$$

This matrix is now factored in **LU** form by the **LU** factorisation algorithm.

The first step simply recognizes without computation that the first element is $u_{11} = 2$. In the next step $i = 2$ and $j = 1$

$$u_{12} = a_{12} = -1$$

$$l_{21} = \frac{1}{u_{11}} a_{21} = \frac{-1}{2} = -\frac{1}{2}$$

$$u_{22} = a_{22} - l_{21} \cdot u_{12} = 2 - \left(-\frac{1}{2}\right)(-1) = \frac{3}{2}$$

Thus the factored part after step 2 is

$$\begin{bmatrix} 2 & -1 & \cdot & \cdot \\ -\frac{1}{2} & \frac{3}{2} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

In the third step $i = 3$ and $j = 1, 2$.

$$u_{13} = a_{13} = -1$$

$$l_{31} = \frac{1}{u_{11}} a_{31} = \frac{-1}{2} = -\frac{1}{2}$$

$$u_{23} = a_{23} - l_{21} \cdot u_{13} = 0 - \left(-\frac{1}{2}\right)(-1) = -\frac{1}{2}$$

$$l_{32} = \frac{1}{u_{22}}(a_{32} - u_{12} \cdot l_{31}) = \frac{1}{\frac{3}{2}}\left(0 - (-1)\left(-\frac{1}{2}\right)\right) = -\frac{1}{3}$$

$$\begin{aligned} u_{33} &= a_{33} - l_{31} \cdot u_{13} - l_{32} \cdot u_{23} \\ &= 4 - \left(-\frac{1}{2}\right)(-1) - \left(-\frac{1}{3}\right)\left(-\frac{1}{2}\right) = \frac{10}{3} \end{aligned}$$

The factored part of the matrix now is

$$\begin{bmatrix} 2 & -1 & -1 & \cdot \\ -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} & \cdot \\ -\frac{1}{2} & -\frac{1}{3} & \frac{10}{3} & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

The final step proceeds similarly to produce the fully factored matrix

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} & -1 \\ -\frac{1}{2} & -\frac{1}{3} & \frac{10}{3} & -\frac{7}{3} \\ 0 & -\frac{2}{3} & \frac{7}{10} & \frac{17}{10} \end{bmatrix}$$

Clearly the upper triangular matrix **U** is identical to that determined by Gauss elimination in Example 1.

* * *

In finite element computations the time spent in solving the equations is often an important part of the total computation time. It is therefore of interest to know the number of operations used in the solution. The division may be considered equivalent to a multiplication. Addition and subtraction are so fast that they can be neglected without appreciable error. The number of operations in step i of the **LU**-factorisation algorithm is

$$N_i = 2\sum_{j=1}^{i-1}(j-1) + 2(i-1) = 2\sum_{j=1}^{i-1} j = i(i-1) \qquad (23)$$

The total operation count then is[1]

$$N = \sum_{i=1}^{n} N_i = \frac{n}{3}(n^2 - 1) \sim \frac{1}{3}n^3 \tag{24}$$

For fairly sized finite element applications N is a very large number – 10^4 degrees of freedom gives $N \sim 3 \cdot 10^{11}$. It is therefore important to exploit the special properties of the equation system, such as symmetry and band structure, to decrease computer time as well as storage requirements.

4. LDL^T FACTORISATION.

A matrix **A** is called symmetric if $a_{ij} = a_{ji}$ for all elements. A symmetric matrix requires only about half the storage of an unsymmetric matrix of the same size, and it is therefore of interest to develop a symmetric factorisation algorithm.

Let **D** be a diagonal matrix

$$\mathbf{D} = \begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{nn} \end{bmatrix} \tag{25}$$

The symmetric matrix **A** can then be factored in the form

$$\mathbf{A} = \mathbf{LU} = \mathbf{LDL}^T \tag{26}$$

where the upper triangular matrix is

$$\mathbf{U} = \mathbf{DL}^T \tag{27}$$

and $\mathbf{L}^T$ is the transpose of the lower triangular matrix **L**.

[1] see Graham et al. (1989) p. 50 for an interesting discussion on the summation of power series.

A modified **LU**-factorisation algorithm for symmetric matrices can now be developed, in which only the lower triangular matrix **L** and the diagonal matrix **D** are computed and stored.

Write the submatrix $\mathbf{A}_i$ in the form

$$\mathbf{A}_i = \left[\begin{array}{c|c} \mathbf{A}_{i-1} & \mathbf{a}_i \\ \hline \mathbf{a}_i^T & c_i \end{array}\right] \tag{28}$$

The factored form

$$\mathbf{A}_i = \mathbf{L}_i \mathbf{D}_i \mathbf{L}_i^T \tag{29}$$

contains the submatrices

$$\mathbf{L}_i = \left[\begin{array}{c|c} \mathbf{L}_{i-1} & \mathbf{0} \\ \hline \mathbf{l}_i^T & 1 \end{array}\right] \tag{30}$$

and

$$\mathbf{D}_i = \left[\begin{array}{c|c} \mathbf{D}_{i-1} & \mathbf{0} \\ \hline \mathbf{0}^T & d_i \end{array}\right] \tag{31}$$

Substitution of these expressions into (29) gives

$$\mathbf{A}_i = \left[\begin{array}{c|c} \mathbf{L}_{i-1}\mathbf{D}_{i-1}\mathbf{L}_{i-1}^T & \mathbf{L}_{i-1}\mathbf{D}_{i-1}\mathbf{l}_i \\ \hline \mathbf{l}_i^T\mathbf{D}_{i-1}\mathbf{L}_{i-1}^T & d_i + \mathbf{l}_i^T\mathbf{D}_{i-1}\mathbf{l}_i \end{array}\right] \tag{32}$$

Comparison with (28) gives the equations

$$(\mathbf{L}_{i-1}\mathbf{D}_{i-1})\mathbf{l}_i = \mathbf{a}_i \tag{33}$$

$$d_i + \mathbf{l}_i^T\mathbf{D}_{i-1}\mathbf{l}_i = \mathbf{c}_i \tag{34}$$

These relations are similar to (18) - (20) for the unsymmetric case, but now there are only i instead of $2i-1$ unknown parameters to be determined.

In the solution of (33) it is convenient first to determine the vector

$$\mathbf{u}_i = \mathbf{D}_i \mathbf{l}_i \tag{35}$$

from the equations

$$\mathbf{L}_{i-1} \mathbf{u}_i = \mathbf{a}_i \tag{36}$$

Then $\mathbf{l}_i$ is determined from (35), and finally d_i follows from (34). This procedure is very close to that used in **LU** factorisation. Indeed $\mathbf{u}_i$ is the i'th column of the upper triangular matrix **U**. The difference is that in this case $\mathbf{u}_i$ is overwritten by $\mathbf{l}_i$, when using (35). This way of computing $\mathbf{l}_i$ saves about half the number of operations and half the storage.

Algorithm: LDL^T-factorisation.

```
for  i := 2  to  n                                   (step i)
     for  j := 2  to  i - 1                          (eqn. j)
     for  k := 1  to  j - 1
          a_ji := a_ji - a_kj * a_ki                 (u_ji from (36))
     for  j := 1  to  i - 1
          u := a_ji                                  (temporary u_ji)
          a_ji := a_ji / a_jj                        (l_ij = u_ji / d_jj)
          a_ii := a_ii - a_ji * u                    (d_ii from (34))
```

* * *

In the $\mathbf{LDL}^T$ algorithm only the upper half of $\mathbf{A}$ is used, and upon exit this upper half has been replaced by $\mathbf{L}^T$ and $\mathbf{D}$. The algorithm works essentially "in place" with only a single temporary storage location needed. The operation count is

$$N \sim \frac{1}{6}n^3 \tag{37}$$

about half of the $\mathbf{LU}$ algorithm.

The $\mathbf{LDL}^T$ solution part has three phases

$$\mathbf{Lz} = \mathbf{b} \tag{38}$$

$$\mathbf{Dy} = \mathbf{z} \tag{39}$$

$$\mathbf{L}^T\mathbf{x} = \mathbf{y} \tag{40}$$

The implementation of the corresponding algorithm is straight forward, when it is remembered that $\mathbf{L}^T$, and not $\mathbf{L}$, is stored in the locations of the matrix $\mathbf{A}$.

Algorithm: $\mathbf{LDL}^T$ solution.

for $i := 2$ *to* n

for $j := 1$ *to* $i - 1$

$b_i := b_i - a_{ji} * b_j$ ($\mathbf{z}$ *from* (38))

for $i := 1$ *to* n

$b_i := b_i / a_{ii}$ ($\mathbf{y}$ *from* (39))

for $i := n - 1$ *down to* 1

for $j := i + 1$ *to* n

$b_i := b_i - a_{ij} * b_j$ ($\mathbf{x}$ *from* (40))

* * *

Example 3. $\mathbf{LDL}^T$ factorisation.

When the 4 × 4 matrix considered in Example 1 and 2 is factored by the $\mathbf{LDL}^T$ algorithm, the upper half of $\mathbf{A}$ is replaced by $\mathbf{D}$ and $\mathbf{L}^T$. The result is

$$\begin{bmatrix} 2 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ & \frac{3}{2} & -\frac{1}{3} & -\frac{2}{3} \\ & & \frac{10}{3} & -\frac{7}{10} \\ & & & \frac{17}{10} \end{bmatrix}$$

This form should be compared with that obtained in Example 2.

* * *

5. PROFILE STORAGE ALGORITHM.

In finite element problems it is usually possible to obtain a banded structure of the equations by suitable numbering of the degrees of freedom. Automatic procedures for optimising the band structure have also been developed, see e.g. Schwarz (1988). The main point is illustrated in the following example.

Example 4. Bandwidth and profile.

The element mesh in Fig. 3 is made up of simple triangles with linear interpolation. When the nodes are numbered as shown in the figure - this is *not* the optimal numbering – the upper half of the conductivity matrix has the entries.

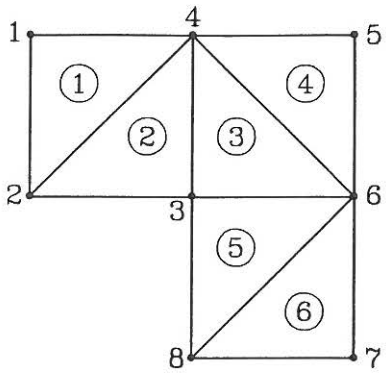
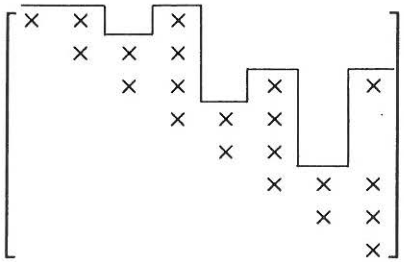


Fig. 3. A simple triangular mesh.

$A_i =$



The matrix has band structure, but the "profile" indicated in the figure appears to be more characteristic. The profile describes the height of each column above the diagonal. Note that zero entries may be included within the profile.

* * *

This section gives a description of $\mathbf{LDL}^T$ factorisation of a symmetric matrix stored columnwise inside the matrix profile. The method is easily extended to $\mathbf{LU}$ factorisation of non-symmetric matrices, but in the case of non-symmetric matrices pivoting may be necessary, and that would complicate the method.

The idea of profile storage of a symmetric matrix $\mathbf{A}$ and the corresponding $\mathbf{LDL}^T$ factorisation algorithm follows from the relation (36),

$$\mathbf{L}_{i-1}\mathbf{u}_i = \mathbf{a}_i \tag{41}$$

$\mathbf{L}_{i-1}$ is a lower triangular matrix, and it then follows that of the first i_0 elements of $\mathbf{a}_i$ are zero, then the first i_0 elements of $\mathbf{u}_i$ must also be zero. It follows from (35) that $\mathbf{l}_i$ also starts with i_0 zero elements. As a consequence of this the factored form of $\mathbf{A}$ given by $\mathbf{D}$ and $\mathbf{L}^T$ fits within the profile of the original matrix $\mathbf{A}$.

In profile storage of a matrix $\mathbf{A}$ the entries between the profile and the diagonal – including the diagonal – are stored columnwise in a one-dimensional array A_i. The order of storage of the matrix of Example 4 is

$$A_i = \begin{bmatrix} 1 & 2 & & 6 & & & & \\ & 3 & 4 & 7 & & & & \\ & & 5 & 8 & & 12 & & 18 \\ & & & 9 & 10 & 13 & & 19 \\ & & & & 11 & 14 & & 20 \\ & & & & & 15 & 16 & 21 \\ & & & & & & 17 & 22 \\ & & & & & & & 23 \end{bmatrix} \tag{42}$$

The number of each diagonal element is stored in the index array

$$m_i = (m_1, m_2, \ldots, m_n) \tag{43}$$

For the matrix (42) the index array m_i is

$$m_i = (1, 3, 5, 9, 11, 15, 17, 23) \tag{44}$$

The index array determines the two-dimensional structure of the matrix completely. This column no. 6 starts with $A(m_5 + 1) = A(12)$ and ends with $A(m_6) = A(15)$.

Two modifications of the $\mathbf{LDL}^T$ algorithm are needed, when profile storage is used:

- a correspondance between the double subscript of the algorithm and the single storage subscript must be established, and
- checks must be incorporated to prevent the algorithm from addressing locations outside the profile.

Let i_0 denote the number of zero elements above the profile in column No. i. The number of elements inside the profile is $m_i - m_{i-1}$. The total number of elements in column No. i is i, and thus

$$i_0 = i - (m_i - m_{i-1}) \tag{45}$$

For column No. 6 in (42)

$$i_0 = 6 - (15 - 11) = 2$$

Similarly j_0 defines the number of zero elements in column No. j. i_0 and j_0 serve as limits for any operation involving columns i and j.

The correspondance between elements in column i is

$$a_{ki} = A_{(m_{i-1}-i_0+k)} = A_{(m_i-i+k)}, \quad k = i_0 + 1, \ldots, i \tag{46}$$

In the algorithm it is important to restrict the range of the subscript k to $(i_0 + 1, \ldots, i)$.

In the $\mathbf{LDL}^T$ algorithm two columns, i and j, are in use simultaneously when forming products $a_{kj} \cdot a_{ki}$, $k = 1, \ldots, j-1$. Here the subscript k must start at $max(j_0, i_0)+1$. When these modifications are incorporated in the $\mathbf{LDT}^T$ algorithm the following profile factorisation algorithm is obtained.

Algorithm: Profile factorisation.

$$
\begin{aligned}
&for \quad i := 2 \quad to \quad n \\
&\qquad i_s := m_i - i && (sum\ index\ i) \\
&\qquad i_0 := m_{i-1} - i_s && (zeros\ in\ i) \\
&\qquad for \quad j := i_0 + 2 \quad to \quad i - 1 \\
&\qquad\qquad j_s := m_j - j && (sum\ index\ j) \\
&\qquad\qquad j_0 := m_{j-1} - j_s && (zeros\ in\ j) \\
&\qquad\qquad k_0 := max\,(i_0, j_0) && (max\ zeros\ i, j) \\
&\qquad\qquad ji := i_s + j && (index\ (j, i)) \\
&\qquad\qquad for \quad k := k_0 + 1 \quad to \quad j - 1 \\
&\qquad\qquad\qquad A(ji) := A(ji) - A(j_s + k) * A(i_s + k) \\
&\qquad for \quad j := i_0 + 1 \quad to \quad i - 1 \\
&\qquad\qquad ji := i_s + j \\
&\qquad\qquad U := A(ji) \\
&\qquad\qquad A(ji) := A(ji)/A(m_j) \\
&\qquad\qquad A(m_i) := A(m_i) - A(ji) * U
\end{aligned}
$$

* * *

The profile solution procedure makes use of the same indexing scheme.

Algorithm: Profile solution.

for $i := 2$ *to* n

$i_s := m_i - i$ *(sum index i)*

$i_0 := m_{i-1} - i_s$ *(zeros in i)*

for $j := i_0 + 1$ *to* $i - 1$

$b_i := b_i - A(i_s + j) * b_j$

for $i := 1$ *to* n

$b_i := b_i / A(m_i)$

for $i := n - 1$ *down to* 1

for $j := i + 1$ *to* n

$j_s := m_j - j$ *(sum index j)*

$j_0 := m_{j-1} - js$ *(zero in j)*

if $i > j_0$ *then* *(profile)*

$b_i := b_i - A(j_s + i) * b_j$

* * *

In order to use the profile storage and solution scheme in a finite element calculation the index array m_i must first be calculated by the program. This calculation is performed in two steps: first all elements are scanned and the array m_i is used to accumulate the column heights, and finally the index values are calculated from the column heights. The accumulation of column heights is conveniently implemented as a dummy run of the assembly routine.

Algorithm: Get profile.

$for \quad i := 1 \quad to \quad n$ *(initialize m_i)*

$\quad m_i := 1$

$for \quad k := 1 \quad to \quad NoofElem$

$\quad AssmElem\ (m_i, \ldots, k)$ *(get heights)*

$for \quad i := 2 \quad to \quad n$

$\quad m_i := m_{i-1} + m_i$ *(height to index)*

$AssmElem(m_i, \ldots, k)$:

$n_1 :=$ *first global DOF*
$\vdots$
$n_{Ne} :=$ *last global DOF*
(find N_e global DOF)

$for \quad i := 1 \quad to \quad N_e$
$for \quad j := i + 1 \quad to \quad N_e$
(all combinations)

$\quad h := \mid n_i - n_j \mid + 1$ *(new height)*

$\quad if \quad n_i < n_j \quad then$

$\quad\quad if \quad m_{n_j} < h \quad then \quad m_{n_j} := h$

$\quad else$

$\quad\quad if \quad m_{n_i} < h \quad then \quad m_{n_i} := h$

* * *

Example 5. Get profile.

Consider the example of Fig. 3. In this example $n = 8$ and the triangular elements have $N_e = 3$. First the array m_i is initialized,

$$m_i = (1, 1, 1, 1, 1, 1, 1, 1)$$

The first call to AssmElem concerns element $k = 1$ with global node references

$$n_i = (1, 2, 4)$$

This leads to the update

$$m_i = (1, 2, 1, 4, 1, 1, 1, 1)$$

When all elements have been called the array m_i contains the column heights,

$$m_i = (1, 2, 2, 4, 2, 4, 2, 6)$$

Finally sequential addition of column heights gives the index array

$$m_i = (1, 3, 5, 9, 11, 15, 17, 23)$$

This index array was already obtained directly from the matrix entries in (42).

* * *

When the size of the inner loops of the factorisation algorithm are limited by the bandwidth or the profile the typical number of computations in one step is approximately $\frac{1}{2}B^2$ as indicated in Fig. 4.

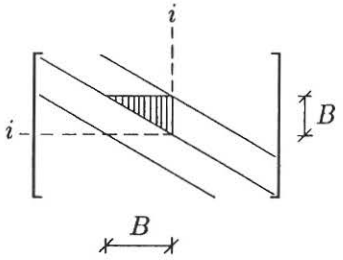


Fig. 4. Active area $\simeq \frac{1}{2}B^2$ in step i.

This gives a total number of operations proportional with nB^2, where B is the "effective bandwidth". In most practical finite element applications $B \ll n$, making the use of band or profile algorithms important.

The implementation of the profile factorisation and solution algorithm described here makes use of a long single array $A(\ )$. This implementation is suitable for traditional programming practice in a language such as FORTRAN 77. In PASCAL or C the profile algorithms are conveniently implemented by use of pointer-variables, whereby each column may be allocated dynamically. The corresponding pointer implementations of the profile algorithms are easily obtained from the present version.

A further step in the development of solution algorithms for banded systems is the so-called Front Solution Technique developed by Irons (1970), see e.g. Irons & Ahmad (1980). In this method only the part of the global matrix necessary to define the active area is assembled in step i. The part of the banded matrix before $i - B$ is fully factored and can be transferred sequentially to background storage. In this way the algorithm is only limited in size by the maximum size of the active area during factorisation. The front solution technique is implemented in many commercial finite element programs and is of particular importance on small computers.

6. PRESCRIBED VALUES.

In the finite element method it is customary to assemble the full global matrix, including also degrees of freedom that may have prescribed values. If the value x_i of a specific degree of freedom i is known a priori, the corresponding value b_i of the right hand side is unknown. Thus the equation essentially is an expression for b_i. If the corresponding row $(a_{i1}, a_{i2}, \ldots a_{in})$ of the matrix $\mathbf{A}$ is left untouched by the factorisation algorithm, the value of b_i can be calculated, when the vector $\mathbf{x}$ has been determined. This only requires a slight modification of the factorisation algorithm. Let the logical array $f_i = (f_1, f_2 \ldots, f_n)$ indicate prescribed (fixed) degrees of freedom by $f_i = \mathit{true}$. The $\mathbf{LDL}^T$ algorithm is modified by including the statement $\mathit{if}\ (\mathit{not}\ f_i)$ at the beginning of each loop, e.g.

$$\mathit{for}\quad i := 2\quad \mathit{to}\quad n\quad \mathit{if}\ (\mathit{not}\ f_i),\ \mathit{etc.}$$

This will leave rows and columns corresponding to the prescribed degrees of freedom untouched and available for later analysis.

Alternatively the prescribed values $x_i = x_i^*$ may be left as unknowns, and the equations matrix modified to enforce the relation $x_i = x_i^*$. This is done by adding $a_i^* x_i = a_i^* x_i^*$ to equation i, where a_{ii}^* is large compared with the original diagonal element a_{ii}.

$$\begin{bmatrix} a_{11} & & a_{1i} & & a_{1n} \\ & & \vdots & & \\ a_{i1} & \ldots & a_{ii} + a_i^* & \ldots & a_{in} \\ & & \vdots & & \\ a_{n1} & & a_{ni} & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \\ x_i \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_i + a_i^* u_i^* \\ \vdots \\ b_n \end{bmatrix} \tag{47}$$

In practice it is convenient to introduce a_i^* in the form $a_i^* = c_i a_{ii}$, where c_i is a large non-dimensional constant. The advantage of this method is its simplicity. A disadvantage is that the accuracy depends on the matrix $\mathbf{A}$ as well as on the constant c_i.

REFERENCES.

Bathe, K.-J. (1982): *Finite Element Procedures in Engineering Analysis.* Prentice–Hall, Englewood Cliffs.

Graham, R. L., Knuth, D. E. and Patashnik, O. (1989): *Concrete Mathematics.* Addison-Wesley, Reading, Mass.

Hughes, J. R. H. (1987): *The Finite Element Method.* Prentice–Hall, Englewood Cliffs.

Irons, B. (1970): A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering.* **2**, pp. 5-32.

Irons, B. and Ahmed, S. (1980): *Techniques of Finite Elements.* Ellis Horwood, Chichester.

Krenk, S. (1989): *Simple trekantelementer.* Department of Building Technology and Structural Engineering, University of Aalborg.

Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1986): *Numerical Recipes.* Cambridge University Press, Cambridge.

Schwarz, H. R. (1988): *Finite Element Methods.* Academic Press, London.