Numero 4 - 1992

RATIO MATHEMATICA

Atti del Convegno di
Matematica Applicata all'Economia e all'Ingegneria
Ovindoli - Maggio 1991

a cura di

Franco Eugeni e Mario Gionfriddo

Comitato Scientifico

Direttore Responsabile: Prof. FRANCO EUGENI

Autorizzazione n. 9/90 del 10- 07-1990

Stampato nel mese di ottobre 1992
da Novastampa srl, Parma, per:

BIBLOS scrl
via dei Peligni, 102 - Pescara
Tel. 085/65920

Ricordiamo affettuosamente

Questi atti e i lavori in essi contenuti sono dedicati alla memoria di due nostri amati Colleghi immaturamente scomparsi dopo il Convegno di Ovindoli. Di questo e del precedente essi erano stati, con noi, animatori e sostenitori. A loro va il nostro affettuoso ricordo: saranno sempre nel nostro cuore!

ILIO ADORISIO

Il prof. Adorisio nelle sue molteplici attività di studioso era andato ben oltre la sua figura di Ingegnere e del suo antico desiderio di essere un Fisico. Era divenuto uno scienziato completo nel senso rinascimentale del termine. Si occupava di fatto di tutte le discipline scientifiche: dalla Matematica all'Economia per l'Ingegneria ma anche di Storia, Scienze Politiche, Antropologia, Psichiatria, e recentemente, non pago di discipline "inquadrate", aveva anche scritto e rappresentato una magnifica ed allegorica commedia, magistralmente interpretata dalla figlia.

Sembrava a noi amici che avesse l'interesse a stupirci sempre, e ci stupiva!.

Il prof. Ilio Adorisio, nato a Cirò (Catanzaro) il 25 aprile 1925, era professore ordinario di Economia Matematica nella Facoltà di Ingegneria della Sapienza. In qualità di consulente della Banca Mondiale dal 1958 al 1974 ha redatto studi economici per conto di paesi del terzo mondo ed ha diretto l'elaborazione di piani di investimento in Sud-America, Africa ed Asia. Oltre che di economia si è occupato di analisi teoriche relative a discipline ingegneristiche.

Il nostro caro Ilio non si può certo dire che fu solo un tecnico: sempre affascinato dai problemi filosofici posti dalla Scienza fu un moderatore-provocatore, forse più provocatore, in tutti i gruppi di ricerca che magistralmente condusse.

GIOVANNI MELZI

Il prof. Melzi era una figura di studioso puro, con interessi multidisciplinari, profondo per lo scibile del passato e attento a tutte le nuove discipline di frontiera, discipline che puntualmente riversava nel suo splendido impegno didattico.

Lo ricordiamo nei Convegni Nazionali della Mathesis, al suo arrivo era tutta una festa: è arrivato Melzi, è arrivato il Melzi, è arrivato Nino!. Nell'ultimo di Cattolica ci parlò con entusiasmo di un Corso che stava facendo in una scuola a fini speciali: "Forse il prossimo anno ne avrò delle belle da raccontarvi". L'anno successivo 1992, a Fermo, non venne: era ricoverato. Noi amici parlammo a lungo di Lui, ci mancava. Pensiamolo ancora a Pandino, in giro con il calesse come soleva fare!.

Il prof. Melzi aveva 60 anni.

Nel 1967 divenne Ordinario di Geometria nella Facoltà di Scienze dell'Università di Milano, successivamente fu Preside della Facoltà di Scienze dell'Università Cattolica di Brescia e anche professore di Logica Matematica prima e Matematica Generale poi.

Dopo gli studi iniziali di Geometria Differenziale si occupò di varie questioni quali ad esempio la sua Teoria delle Neuromacchine che andava dalla Logica alla Cibernetica ed alla Teoria dei Sistemi.

Del convegno di Ovindoli tutti ricordano la magnifica musica ottenuta da un sistema che aveva applicazioni in... economia.

A tutti quelli che hanno avuto il piacere ed il privilegio di conoscerLo rimane il ricordo di un amato Maestro e di un coraggioso ed eclettico Uomo di Scienza.

La sua dotta e pacata voce con cui ci spiegava semplicemente le sue più ardite riflessioni ci manca e ci mancherà!

Indice

ON THE PROFILE REDUCTION OF SPARSE SYMMETRIC MATRICES

Antonio BALDI, Antonio DE PAULIS

Summary. After a short presentation of the profile reduction and band optimization methods for sparse symmetric matrices, a new algorithm is presented. Numerical tests on typical Finite Elements problems are made and the results are discussed.

1. INTRODUCTION

To find the numerical solution to many engineering problems the first step is the discretization of the physical problem. Usually this discretization process leads to algebraic systems of the type

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

where the matrix $\mathbf{A}$ is usually symmetric and sparse.
The systems (1) are usually solved using direct methods i.e. the Gauss method or variations such as the Choleskj method. Good results from all these algorithms strongly depend on the matrix structure; some methods for optimizing this structure are outlined below.
Among the principal discretization techniques is the Finite Element Method (F.E.M.).
For Finite Element problems involving a small number of nodes, or a small number of variables x, it is not difficult to order the nodes efficiently; with more complex grids this is often more difficult.
An efficient node ordering is particularly important in non-linear computations where a sequence of equations must be solved many times or when the F.E. meshes are automatically generated and leading to an unnecessarily large band β.
Today there are numerous euristhic methods to minimize approximately the structure of a matrix. These methods can be described with the Graph Theory which provides a succinct means of discussing the various ordering schemes.
A *Graph* $G = (X,E)$ consists of a finite set of *nodes* (or *vertices*) X together with a set E of *edges* , which are unordered pairs of nodes. A mapping of $\{ 1, 2, 3, \ldots , N \}$ onto X , where N denotes the number of nodes of G , is an *ordering* of G

$$G^{\alpha} = G (X^{\alpha}, E)$$

Let $\mathbf{A}$ be an N by N symmetric matrix. An ordered graph G^{A} is one for which the N vertices are numbered from 1 to N and the graph edge $E = \{ x_i, x_j \}$ belongs to E^{A} if and only if $a_{ij} = a_{ji} \neq 0$.
The same graph can be constructed directly by the F.E. mesh; in this case X is the set of the mesh nodes while an edge $E = \{ x_i, x_j \}$ belongs to the graph only if the mesh nodes are connected by one element. Note that in the F.E. method each node usually has more than one degree of freedom, but for our purposes it is sufficient and simpler to consider each of them as having just one degree of freedom. Figure 2 illustrates a simple mesh with its associated graph.
Each graph which is generated in this way can be seen as a *tree*, when one has defined a node as a root. A generical $k + 1$ tree level is defined as the set of nodes which are adjacent to the nodes at k level.
Two nodes x and y are *adjacent* if $\{ x, y \} \in E$.
For a set of node $Y \in X$, the adjacent set, denoted by $Adj(Y)$, is

$$Adj(Y) = \{ x \in X - Y \mid \{ x, y \} \in E \text{ for some } y \in Y \}$$

while the *degree* of Y, denoted by $Deg(Y)$, is the number of members of $Adj(Y)$. Clearly, if Y consists of a single node (y), the adjacent set is formed by the nodes x_i connected to y and its degree is the number of $E_i = \{ x_i, y \}$.

Let i be the i-th row of matrix $\mathbf{A}$. Let

$$f_i(\mathbf{A}) = \max \{ j \mid a_{ij} \neq 0 \},$$

that is the column index of the last non zero element in row i and let

$$\beta_i(\mathbf{A}) = f_i(\mathbf{A}) - i.$$

The *bandwidth* of the matrix $\mathbf{A}$ is defined by

$$\beta(\mathbf{A}) = \max \{ \beta_i(\mathbf{A}) \mid i \leq N \}$$

while the *profile* P is

$$P(\mathbf{A}) = \sum_{i=1}^{N} \beta_i$$

Note that an ordering algorithm which gives the lower profile does not give the smaller bandwidth: in solving a linear system of equations for numerical stabilility purposes is best to have the bandwidth β as small as possible, while to minimize the execution time is best to have a low profile P.

Let x, y be a pair of nodes of a connected graph G. A *path* of length k from node x to y is a set of k vertices $\{ x, x_1, x_2, \ldots, y \}$ such that $x_i \in Adj(x_{i+1})$ for $0 \leq i \leq k - 1$.

The *distance* $d(x,y)$ between x and y is simply the length of the shortest path joining x and y.

If $\lambda(x) = max \{ d(x,y) \mid x,y \in X \}$, then a diameter of the graph G can be given by $\delta(G) = max \{ \lambda(x) \mid x \in X \}$. A node $x \in X$ is *peripheral* if $\lambda(x) = \delta(G)$.

A diameter found by a numerical algorithm is a pseudo-diameter; the first and last nodes of the path defining the pseudo-diameter are pseudo-peripheral nodes. Note that a pseudo-diameter is not guaranteed to be a diameter but only to have a large λ.

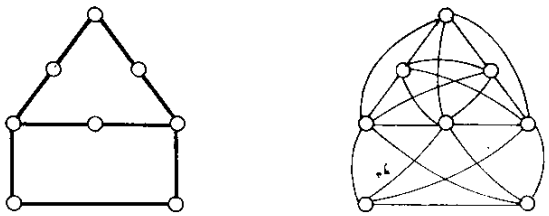


Fig. 1 - A simple F.E.M. mesh and the associated graph.

2. SOME PRACTICAL CONSIDERATIONS

As the algorithms described in subsequent paragraphs operate on Graphs, a computer representation of the adjacency properties of each node, economical in storage and easy to use, is needed.

Let $G(X,E)$ be a graph with N nodes. An *adjacent structure*, i.e. the *list of adjacency* for each $x \in X$, can be implemented quite simply by storing the *adjacency lists* (AL) sequentially in a one dimensional array along with a *index array* containing pointers to the beginning of each list.

Although this method gives the best storage, it is quite complex to use; moreover, it is difficult to construct the connection array when the graph is provided as a list of edges because the final size of each adjacent list is not usually known.

A simpler approach involves a connection table with N rows and $m+1$ columns, where $m = max \{ Deg(x) \mid x \in X \}$.

Each row *i* contains the adjacency list for a node and, in column 0, the node degree. This scheme may be inefficient from a storage point of view (a problem overcome by using the dynamic allocation) but the construction of the adjacent structure can be made easy as, for example, in this Pascal code fragment:

```
for i := 1 to Element_Number do begin
    for j := 1 to Element_Size [Element[i].El_Type]  do begin
        k := 1;  m := Element [i].Node_List[j] ;
         repeat
          if k< >j then begin
          sw := true;
            for i := 1 to AdjT[m,0]  do
              if AdjT[m,i]  = Element [i].Node_List [k]  then
                sw := false;
              if sw then begin
                 AdjT[m,0]  := AdjT[m,0]  + 1;
                 AdjT [m,AdjT[m, 0]]  := Element [i].Node_List[k]
              end
          end;
           k := k+1
        until k > Element_Size [Element[i].El_Type]
end;
```

Here *AdjT* is the adjacency table, *Element_Size* is a table containing the nodes number for each element type and *Element* is an array of records which gives the type (*El_type*) and the list of nodes (*Node_List*) for each of them.

3. ORDERING METHODS

A central concept to many successful ordering algorithms is the *rooted level structure*.
This is a partitioning of the nodes such that each node is assigned to one of the levels $l_1, l_2, \ldots, l_n$ with its distance from a specified *root node* s in the graph. All nodes which lie at the same distance from the root node belong to the same level.
The *depth* of a rooted level structure is simply its total number of levels.
The *width* is the maximum number of nodes which belong to a single level.
The labelling algorithms comprise two distinct steps:

a) the selection of pseudo-peripheral nodes and

b) node labelling.

In step a) a pair of pseudo-peripheral nodes which lie at opposite ends of a pseudo-diameter are found in this way:

1) select a node s with the smallest degree as starting node;
2) generate a level structure rooted at node s;
3) for each node q of the last level generate a level structure; if the height of this structure is higher than the original one, set q as the starting node (s = q) and jump to 1.

Computational experiments indicate that the last level node set is usually quite large and in this set many nodes have the same degree. To reduce the number of nodes in the last level set, nodes for which a rooted level structure must be generated, *shrinking* strategies (which are supposed not to affect the resulting pseudo-peripheral node quality) were adopted by almost all authors. Sloane, for example /4/, creates a list with only one node for each degree; Duff, Reid, Scott /5/ constructed a list with nodes in the last level with ties broken arbitrarily. This strategy led to a slight increase in the computed reduced profile for a class of problems; for other problems they only found a small reduction.

The second step, node labelling, is a field with a wide spectrum of possible choices: usually one starts from the root node and crosses the tree structure in a moving wavefront to end at the tree last level.

Cuthill-McKee's well known algorithm labels, for each tree level node, all the following level unnumbered nodes in increasing order of degree.
Although this method works, it only uses a local parameter, so many authors have presented more sophisticated algorithms. The most effective is the Sloan algorithm, which, during the labelling process, defines active, inactive, pre-active and post-active node states; in this process the current wavefront advancement is controlled by a quantity called the current degree. This label method works dynamically by combining a local parameter (the node degree) with a global one (the node level).
At each stage in the labelling process, the list of eligible nodes (the wavefront) includes those nodes which are either adjacent to a node which has been relabelled or are adjacent to a node which is itself adjacent to a relabelled node.
The next node to be given a new number is the node with the highest priority, where the priority P_i is defined as

$$P_i = - W_1 \, cdeg(i) + W_2 \, d(e,i)$$

Here W_1, W_2 are positive integer weights and the current degree *cdeg(i)* is the number of nodes adjacent to i, not including any node that has been relabelled or is itself adjacent to a relabelled node.
The priorities of the eligible nodes are updated at each step.

To explain the proposed algorithm some new terms are introduced:

in a generic step of the labelling process, all of the previously labelled nodes connected to a node n are *n's fathers*, while all the unlabelled nodes at the next tree level are *n's sons*. The father state is clearly independent of the tree level; nevertheless, the ordering scheme creates an advancing front so that the *fathers* usually belong to the same or former level of node n.

To develop the algorithm an eligible queue has been introduced, which contains all the vertices that can be labelled at the current step. The selection within the queue is made using an index called the *node current priority*, defined as

$$Np(n) = W_F \, N_F - W_S \, N_S$$

where N_F and N_S are, respectively, the number of fathers and sons at the current ordering step ($N_F(n) + N_S(n) \leq Deg(n)$) while W_F and W_S are two positive integer weights.
Numerical tests proved that a good choice is the value 2 for W_F and 1 for W_S , as in the Sloan algorithm, although the two weights refer to parameters with different meanings.
This priority formulation is based on the observation that one must label nodes with few sons first (so nodes with a lot of them are nearest to their sons), but at the same time label as early as possible nodes with many fathers too (Fig. 2).
Numerical tests gave as a good choice the value of 2 for Wf and 1 for Ws.
Although this values are the same as in Sloan algorithm, this is merely a coincidence; in fact they refer to parameters with different meaning.

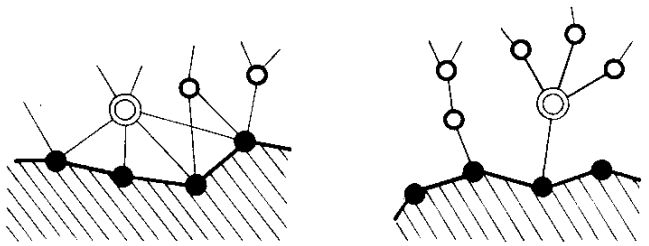


Fig. 2 - The priority scheme.

The algorithm can be summarized in the following steps:

1) look for a starting node *r* using the Gibbs' algorithm or its modifications,

2) construct the tree structure rooted at r.

3) Set start-up priorities: no nodes are labelled: there is no father. All vertices have a current priority $Np(n_i) = -W_s \times N_s(n_i)$ where $N_s(n_i)$ is the number of next level neighbours.

4) For each node set *TRUE* the queue insertion control item (*IcL*); this is an array whose items are *TRUE* if nodes have not been inserted in the *eligibly queue*.

5) Initialize queue:
set $Next\_label \leftarrow 1$, $Queue\_Size \leftarrow 1$, $Queue[1] \leftarrow r$, $IcL(r) \leftarrow false$.

6) Loop: create a list MpL of all the maximum priorities nodes in queue.

In this loop, when there are many nodes with the same priority, the node with minimum distance to the root is selected. This selection method is the same as Sloan used,, but with a different meaning.

a) Select the node k of lower level from MpL;
b) update priorities; for each neighbour i of k
i) set $Np[i] \leftarrow Np[i] + W_F$, (node i gains a father)
ii) if $level[i] > level[k]$ and $Np[i]$ is *TRUE*, insert i in queue and update Np.
iii) if $level[i] < level[k]$ set $Np[i] \leftarrow Np[i] - W_s$, (node i loses a son).
iv) Label k by setting $k \leftarrow Next\_label$,
$Next\_label \leftarrow Next\_label + 1$,
$Queue[k] \leftarrow Queue[Queue\_size]$,
$Queue\_Size \leftarrow Queue\_Size - 1$.
v) Test for termination: if $Queue\_Size > 0$ go to Step 6).

Numerical experiments have proved that the best starting priority setting is with $N_s(n_i)$ (ALG3 b) substituted by $Deg(n_i)$ at step 3) (ALG3 a). This means that not only the next level nodes are considered as being sons but also all the neighbours.

4. NUMERICAL TESTS

To test the performance of the proposed ordering schemes, F.E. meshes were used. They were divided into three groups:

a) Problems that cannot be improved (Test No. 0).

b) Problems for various theoretical configuration (Mostly taken from /2/ : Test Nos. 1, 2, 3, 4, 5).

c) Examples for typical engineering calculations (Test Nos. 6, 7, 8).

Test No. 0 was performed to see how bad the labelling created by an ordering scheme was compared to the original one. Group b) contains small meshes, with particular characteristics. Group c) to see how the algorithms perform in real problems.

Obviously these problems do not cover all the cases and it is easy to find meshes where the ordering scheme performances are quite different from those summarized in Table I; however they are sufficiently reppresentative to extrapolate average algorithm results /4/.

In Table I the results are arranged as follows: each row refers to a different ordering scheme, each column refers to a single test problem. For each problem there is the matrix profile and the ratio obtained by dividing the actual profile by the best result. The original matrix profile is in the first row.

The results are from algorithms RCM, Sloan, ALG3a, ALG3b.

The Table shows that the proposed algorithm nearly always gives good results than the others.

The suggested modification in the start-up priorities generally improves performances, except in test 7.

The Reverse Cuthill McKee method only gave the best performance for test 8 where the exceptionally high and tight graph promotes local against global optimization.

Although the Sloan method only got the best result once with the present tests, it usually performs very well, better than the other algorithms except ALG3b; it got the best result in the quasi optimal test 0.

TABLE I

P =	Test 0 1090	Test 1 403	Test 2 364	Test 3 675	Test 4 156
RCM	1222 (1.121)	370 (1.439)	300 (1.111)	445 (1.093)	136 (1.295)
SLOAN	1199 (1.1)	284 (1.105)	289 (1.070)	480 (1.179)	116 (1.104)
ALG3a	1228 (1.126)	265 (1.031)	389 (1.003)	426 (1.046)	107 (1.019)
ALG3b	1245 (1.142)	257 (1)	270 (1)	407 (1)	105 (1)

P =	Test 5 3218	Test 6 305	Test 7 3764	Test 8 2957
RCM	1251 (1.356)	303 (1.048)	3929 (1.150)	1873 (1)
SLOAN	1080 (1.171)	321 (1.110)	3464 (1.014)	2120 (1.072)
ALG3a	973 (1.055)	298 (1.031)	3415 (1)	2190 (1.169)
ALG3b	922 (1)	289 (1)	3506 (1.026)	1976 (1.054)

APPENDIX: Computer Code for ALG3.

In this Appendix there is the Pascal code to implement ALG3.
Before the listing the definitions of the global variables are reported.

Global routines to use

1 - **Function FindStartNode : integer;**
This function returns a pseudo peripheral node.
2 - **Function CreateTree (root: integer) : integer;**
This function creates a tree structure (in **Nd** see point 3) rooted at the choosen pseudo peripheral node. It returns the tree height (Note that we do not need this information, but it can be useful for other ordering scheme).

Global Variables

3 - **Nd** : array [1..*MAXNODI*, 1..2] of integer;
contains the node label (column 1) and the tree level for each node (column 2).
4 - **AdjT** : array [1..*MAXNODI*, 0..*MAXCON*] of integer;

adjacency Table as described in text; MAXCON is the max number of nodes connected to a node. The nodes are ordered with the degree.

5 - **Nodes** : integer;
is the number of all nodes in the graph.

Global types

6 - **NodeList** = array [1..*MAXNODI*] of integer;
7 - **NodeCtrl** = array [1..*MAXNODI*] of boolean;

The computer codes are

```
procedure ALG3
Const    Ws = 1; Wf = 2;
Var

    Np,Q      : NodeList  { Priority and Queue }
    MaxP      : NodeList  { Max priority List }
    IcL       : NodeCtrl  { Queue insertion control list }
    Ds,Th     : Integer   { Tree root, height }
    qSize     : Integer   { Elements in queue }
    nName     : Integer   { Next node labelled name }
    i,j,k,d,l,n : Integer { Miscellany }
    lvl,q2l   : Integer   { Miscellany }

    procedure UpdatePriority(Wn,Lev:Integer);
    Var i,j : integer;
    begin
        for i := 1 to AdjT [Wn,0] do begin
            j := AdjT [Wn,i];
            Np [j] := Np [j] + Wf;
            if (Nd [j,2] > Lev) and IcL [j] then begin
              qSize := qSize + 1; Q [qSize] := j; IcL [j] := FALSE;
            end ;
            if Nd [j,2] < Lev then Np [j] := Np [j] - Ws;
        end
            end;

begin                    { ALG3 }
    Ds := FindStartNode;                    { Step 1 }
    Th := CreateTree(Ds);                   { Step 2 }

{ ** Init Priorities: use this code fragment for a theoretically consistent algorithm
    for i := 1 to Nodes do begin
        k := Nd [i,2] + 1; l := 0;
        IcL[i] := TRUE;
        for i := 1 to AdjT [i,0] do
        if Nd [AdjT[i,0],2] = k then l := l + 1;
        Np[i] := - Ws * l
    end;                              ** }

    for i := 1 to Nodes do begin
        IcL[i] := TRUE;
        Np[i] := -Ws*AdjT [i,0]
    end;

    qSize := 1; Q [1] := Ds; NName := 1; IcL [Ds] := FALSE; { Step 5 }
    while qSize > 0 do begin    { Main loop }
        d := Np [Q[1]]; n := 1; MaxP [1] := 1;
        for i := 2 to qSize do begin
            k := Np [Q[i]];
            if k = d then begin
                n := n + 1; MaxP [n] := i;
            end else if k > d then begin
```

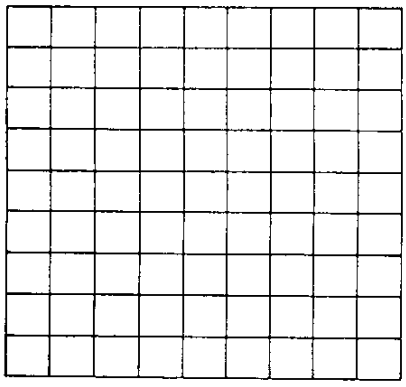
```
                n := 1;  MaxP [1]  := i;  d := k
            end
        end;

        q2l := MaxP [1] ;  lvl := Nd [Q[q2l],2];   { Step 7 }
        for i := 2 to n do
            if Nd [Q[MaxP[i]],2]   < lvl then begin
            q2l := MaxP[i];  lvl := Nd [Q[q2l],2]
            end;

        UpDatePriority(Q [q2l],lvl)                    { Step 8 }
        Nd [Q[q2l],1]  := nName;  nName := nName + 1;   { Step 9 }
        Q [q2l]  := Q [qSize] ;  qSize := qSize - 1;
    end                          { - while loop }
end;
```

REFERENCES

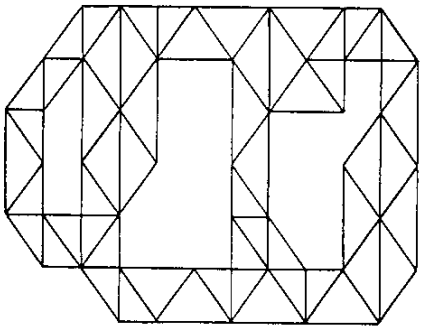
1. A. George, Joseph W.H. Liu
   An Implementation of a Pseudoperipheral Node Finder
   ACM Transactions on Mathematical Software,
   vol. 5, no. 3 (1979), pp 284-295
2. A. George, Joseph W.H. Liu
   Computer Solution of Large Sparse Positive Definite System
   Prentice Hall, Englewood Cliffs, 1981
3. S. W. Sloan
   An Algorithm for Profile and Wavefront Reduction of Sparse Matrices
   Int. J. for Numerical Methods in Engineering, vol 23, 239-251 (1986)
4. S. W. Sloan
   A Fortran Program for Profile and Wavefront Reduction
   Int. J. for Numerical Methods in Engineering, vol 28, 2651-2679 (1989)
5. I.S. Duff, J.K. Reid, J.A. Scott
   The Use of Profile Reduction Algorithms with a Frontal Code
   Int. J. for Numerical Methods in Engineering, vol 28, 2555-2568 (1989)

Author Address

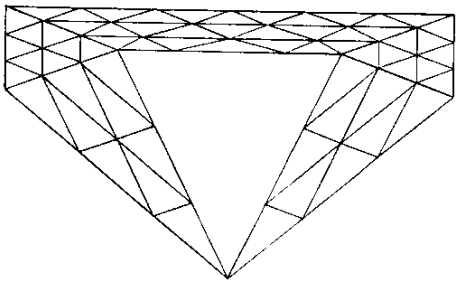
Antonio DE PAULIS
Dipartimento di Energetica
Facolta' di Ingegneria
67040 ROIO POGGIO (AQ) - Italy
FAX number: (0)862-432633



Test 0



Test 1



Test 2

Fig. 3 - F.E. models for Test 0, 1, 2.

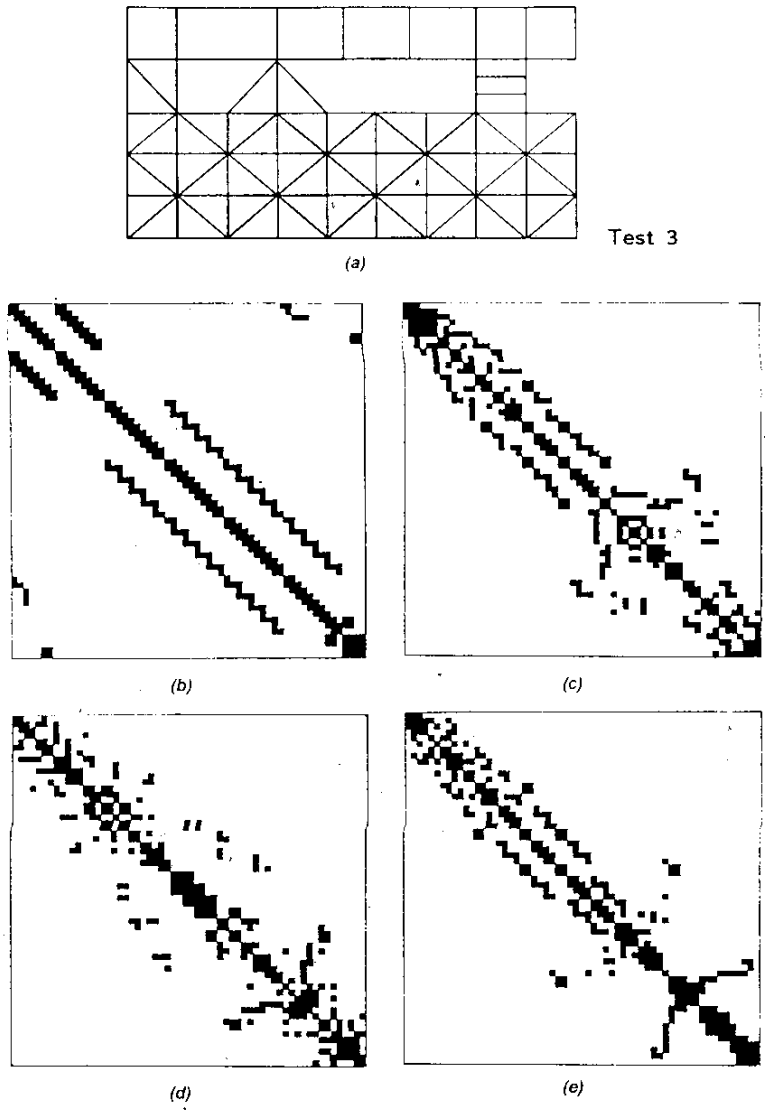


Fig. 4 - Geometrical rappresentation of an ordered matrix. (a) and (b) mesh and original matrix of Test no. 3. The same matrix optimized with (c) Cuthill-Mc Kee, (d) Sloan and (e) ALG3 algorithm.

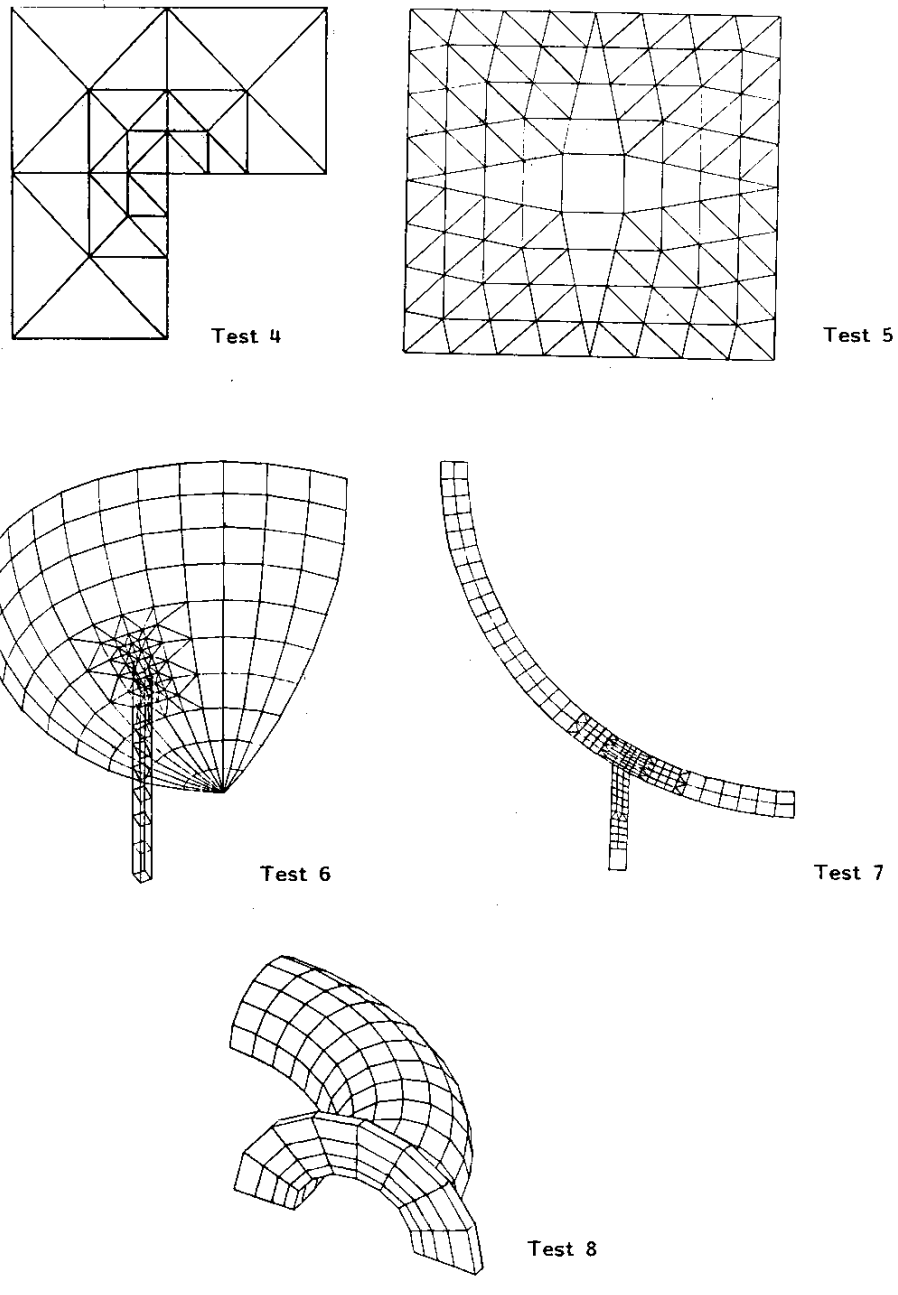


Fig. 5 - F.E. models for Test 4, 5, 6, 7, 8.