Mecánica Computacional Volumen XV
Compilado por A. Larreteguy y M. Vénere
Bariloche, noviembre de 1995

UMA IMPLEMENTAÇÃO DO MÉTODO "SPARSE" E A COMPARAÇÃO DE SUA EFICIÊNCIA COM O MÉTODO "SKYLINE"

Jorge Abeid Neto, Francisco Antonio Menezes
Departamento de Construção Civil, Faculdade de Engenharia Civil
Unicamp - Universidade Estadual de Campinas, Campinas, São Paulo-Brasil

RESUMO

A análise de fenomenos físicos via Método dos Elementos Finitos termina por produzir sistemas de equações lineares de grandes proporções, exigindo a armazenagem de um imenso número de variáveis tornando moroso o processamento, quando não, inviabilizando-o, dependendo do tipo e configuração do equipamento que se tem disponível.

O presente trabalho se vale da Análise Matricial de Estruturas no cálculo de estruturas reticuladas planas como veículo gerador do problema, a criação de um grande sistema de equações lineares para ser solucionado e, a partir de um algoritmo básico, que armazena a matriz de rigidez e resolve o sistema explorando a fatorização LU por eliminação Gaussiana, constrói uma versão via método skyline, que se aproveita da formação em banda dos valores significativos da matriz de rigidez, e outra denominada sparse, que armazena apenas os valores significativos de uma metade mais os elementos da diagonal daquela matriz de rigidez.Os algoritmos foram implementados em Pascal, usando alocação dinâmica de memória.

ABSTRACT

The analysis of physical phenomenon by the Finite Element Method, leads away to large linear systems requiring an excessive number of variables to be stored becoming the process slow or making it unfeasible, depending on the type and configuration of the available equipment.

This paper uses the Matricial Strutural Analysis in the analysis of bidimensional structural frameworks as a generator of the problem, the prodution of a large system of equations , and from a basic algorithm, wich stores the stiffness matrix elements and solves the system by LU fatorization in Gaussian elimination, builds a version through the skyline method, wich, uses the shape band of the nonzero elements of the stiffness matrix, and another called sparse, stores the diagonal elements and the nonzero elements of half of that matrix. The algorithms were implemented in Pascal using dynamic alocation method.

INTRODUÇÃO

A Análise Matricial de Estruturas, aplicável às estruturas de barras é definida por MOREIRA[1], com muita propriedade, como sendo o primeiro capítulo do Método dos Elementos Finitos, uma vez que, trabalhando com funções exatas, pois o fenômeno estrutural opera com funções suaves, constroe a denominada matriz de rigidez local e a partir dela usa o processo para a complementação do cálculo. A estrutura já está discretizada por natureza, tem suas barras constituintes como elementos finitos.

Como o objetivo deste trabalho é o trato da solução dos sistemas lineares gerados, escolheu-se, tendo em vista a sua simplicidade, a análise de estruturas planas pelo método dos deslocamentos, como veículo gerador do problema a ser aqui abordado.

Assim, no caso da análise de estruturas reticuladas planas o método tem por objetivo construir uma matriz-fator com os esforços necessários para que os nós, ou pontos de início e/ou de fim das barras, sofram deslocamentos unitários em cada uma das coordenadas globais definidas previamente, de tal forma que os deslocamentos observados devidos a um carregamento atuante, quando multiplicados por essa matriz-fator, reproduzam em cada coordenada as ações que compõem o tal carregamento.

O processo conduzirá, desta forma, à solução de um sistema de equações lineares que pode ser escrito matricialmente como:

$$[K].\{x\} = \{b\} \qquad (1.1)$$

onde [K] é a matriz dos coeficientes, à qual se dá o nome de matriz de rigidez global, que é a matriz fator como acima foi denominada, $\{x\}$ é o vetor dos deslocamentos a ser calculado e $\{b\}$ é o vetor das ações externas atuantes em cada nó.

No caso em análise, que contempla estruturas planas e reticuladas, pode-se facilmente verificar que a matriz de rigidez global terá uma dimensão de três vezes o número de nós por três vezes o número de nós,ou seja, se está sendo tratada uma estrutura com 50 nós, terá que ser armazenada uma matriz de rigidez de 150 linhas por 150 colunas, o que torna obrigatória a reserva de 22500 posições de variáveis do tipo real na memória do equipamento. O problema surge quando se deseja utilizar microcomputadores do tipo PC, gerenciados por sistema DOS, utilizando-se compiladores como o Pascal, C ou C++, para a análise deste tipo de problema. Conforme apresentado por Roth[2], a arquitetura das máquinas das séries 80*86 acessam a memória em segmentos de 64 Kbytes e os compiladores, C, C++, Pascal, incluindo o Borland Pascal 7.0, reservam apenas um segmento de 64 Kbytes, para ser utilizado como área de dados acessada diretamente pela linguagem. Acrescido o fato de que cada variável do tipo real necessita 6 bytes para seu armazenamento, no caso da utilização do Borland Pascal, verifica-se que se estivesse disponível esse segmento apenas para a matriz de rigidez global, o equipamento estaria limitado a estruturas de 34 nós e se for aventado ainda o fato de que são necessários espaços de memória para os outros dados e resultados da análise e que, a bem da precisão dos resultados, o ideal seria se trabalhar com variáveis com dupla precisão, que ocupam 8 bytes, chegar-se-á à conclusão que a análise ficaria restrita a estruturas ainda menores, o que tornaria os microcomputadores do tipo PC impróprios para esse tipo de utilização.

O ALGORÍTMO BÁSICO

O algoritmo básico tem por objetivo a análise de estruturas aonde se armazena a matriz de rigidez plena e sem reordenação de índices. A partir do básico foram construídas as versões que contemplam o método "skyline" e "sparse" de tal forma que os resultados são perfeitamente comparáveis.
Obtida a matriz [K] e o vetor das ações, já devidamente modificados pelas condições de contorno, o algoritmo calcula os deslocamentos, resolvendo o sistema (1.1) . Para tanto, ele se vale do processo de eliminação de Gauss.A eliminação de Gauss permite, como se pode ver em Golub [3] e em Duff et. al [4] , que se faça a fatorização da matriz dos coeficientes [K] (1.1) em duas matrizes triangulares, uma denominada de [L] e outra de [U] . A primeira é triangular inferior e a outra triangular superior, de tal forma que a matriz chamada de [U], contenha os coeficientes resultantes

das diversas eliminações do sistema gaussiano.Duff et al, op. cit p. 46, propõe que se construa uma matriz [L], fazendo para $i \rangle k$,

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad (2.1)$$

sendo essa uma relação perfeitamente válida, uma vez que para problemas estruturais elástico lineares a matriz dos coeficientes é sempre positiva definida e portanto $a_{kk}^{(k)} \neq 0$. A inversa da matriz [L] multiplicada pela matriz dos coeficientes fornece uma matriz [U], que armazena as diversas etapas da eliminação gaussiana. Encontrada essa matriz [U], o problema está resolvido e as incógnitas determinadas de forma rápida e eficiente. A formulação da solução do problema por essa rota, pode ser apresentada como em (2.2):

$$[L]^{(-1)} \times [K] \times \{x\} = [L]^{(-1)} \times \{b\} \quad (2.2)$$

cujo algoritmo, pode ser visto em (2.3):

```
 L [1]:=1;   (*Vetor que será utilizado para armazenar as colunas que
                 contenham os valores significativos da matriz L*)
 For x:=2 to 3*N do      (* N= número de nós da estrutura em análise*)
 L[x]:=0;
 i:=1; j:=1;
 For y:=1 to 3*N-1
 begin
     while j<3*N do
     begin
      L_{j+1} = a_{j+1,i} / a_{ii}   ;(* a_ij é elemento da matriz [K] dos coeficientes*)
      j:=j+1;
    end;
    For A:=i+1 to 3*N do
    b_A := b_A + b_i × L_A ;(* b_A é elemento do vetor {b} das ações*)
    For A:=y+1 to 3*N do
    begin
       If L_A ≠0 then
       begin
          For k:=1 to 3*N do
          begin
             If a_yk ≠0 then
             begin
                a_{A,k} := a_Ak + L_A × a_yk
             end;
          end;
       end;
 end;
 i:=i+1;  j:=i;
end;
```

(2.3)

Ao final da execução de (2.3), ter-se-á a matriz [U] armazenada por sobre a matriz [K].

O algoritmo básico, com o objetivo de escapar da limitação imposta pelo bloco de 64 kbytes citado na introdução deste trabalho, implementa a técnica denominada de "Alocação Dinâmica de Memória", permitida pelo Pascal.

Trata-se de criar uma estrutura para dados do mesmo tipo, na qual se armazena no bloco reservado pelo compilador para variáveis, apenas o endereço físico de memória da máquina, obtido pelo compilador fora de suas áreas reservadas, no qual foram ou estão gravados os dados de interesse do programa. Sendo assim, pode-se guardar na área limitada apenas um endereço e a partir dele construir uma pilha de dados.

A estrutura de dados parte da criação de um registro no qual serão gravados, no caso da matriz de rigidez, o valor que se deseja armazenar e o endereço do próximo registro. A operação se inicia com a definição do tipo do registro como ilustrado em 2.4:

```
type
aij_ptr=^aij_rec;                    (*1*)
aij_rec=record                       (*2*)        (2.4)
          valor_aij:real;
          next:aij_ptr;
end;
```

em (*1*), foi criada uma variável do tipo ponteiro (aij_ptr), que servirá para armazenar o endereço no qual estará o registro aij_rec, ou pode-se ainda dizer que "aponta" para o registro mencionado. Em (*2*) criou-se o registro, com um campo do tipo real, que servirá para armazenar um valor pertencente a uma posição da matriz de rigidez e o campo denominado "next", que será utilizado para a guarda do ponteiro, ou endereço do próximo registro que armazenará a próxima posição.

Cada vez que se desejar armazenar uma posição, chama-se a subrotina (procedure) **new**, como em 2.5.

```
new(aij_ptr)          (2.5)
```

Por esse procedimento o compilador aloca um segmento de memória suficiente para armazenar um registro e dentro da área reservada para dados (64Kbytes), armazena o endereço do segmento alocado podendo-se guardar o valor a ser armazenado em **aij_ptr^.valor**.

O conceito de pilha se assemelharia, como bem ilustrado por JONES & HARROW[6], a uma pilha de pratos num restaurante, ou seja, o próximo sempre por sobre o anterior, de tal forma que o último a entrar na pilha será o primeiro a sair.

Dessa forma ter-se-á, na memória reservada para dados pelo compilador, sempre o endereço do último segmento alocado, de tal forma a exigir a necessidade de se criar uma sistematica de ligação entre os segmentos, para se poder recuperá-los em algum instante e para isso será necessário aquele campo **next**.

Para se recuperar um elemento da matriz armazenada como pilha, como visto em 4.1, necessita-se apenas saber seu número de ordem, uma vez que, os elementos foram armazenados linha por linha e da primeira à última posição dentro de cada linha. Ora, para se recuperar então o valor de um elemento a_{ij} , basta calcular sua posição de ordem dentro da pilha que pode ser dada por:

$$O = n \times n - \left[(i-1) \times n + j\right] \qquad (2.6)$$

onde: O= número de vezes que se deve saltar registros na pilha;
n= número de linhas ou colunas do sistema;
i= linha a que pertence o elemento procurado;
j= coluna a que pertence o elemento procurado.

Calculado "O", pode-se entrar num laço como indicado em (2.7):

```
topo:=aij_ptr;
for x:=1 to O do                (2.7)
aij_ptr:= aij_ptr^.next;
aij_ptr:=topo;
```

onde: **topo**=variável tipo ponteiro criada para armazenar o endereço do topo da pilha.

É fácil verificar que ao final do laço, se terá em aij_ptr o endereço da posição procurada e nela o valor que se deseja.

Como se pode constatar, cada vez que houver a necessidade de recuperar o valor de uma posição da matriz na pilha, será necessário ir ao seu topo e percorrer através dela até que se chegue ao valor procurado.

É bom lembrar que os problemas comumente analisados, geram matrizes com dezenas de milhares de posições, tornando o processo de procura de elementos na pilha extremamente moroso ou até mesmo impraticável e a solução é criar mais um campo do tipo inteiro dentro do registro definido em (2.4) e estabelecer dessa forma uma dupla ligação da pilha, criando mais um campo do tipo ponteiro naquele registro, de tal modo a se ter a indicação também do registro anterior. No novo campo inteiro lança-se o número de ordem do elemento da matriz transformada em vetor que poderia ser calculado como em (2.8).

$$\text{Ordem:} = (i-1)\times n + j \qquad (2.8)$$

onde : Ordem = número de ordem do elemento a_{ij} na matriz armazenada como vetor;
i =número da linha em que está o elemento procurado;
j =número da coluna em que está o elemento procurado;
n =número de linhas e/ou colunas da matriz.

A pilha de dados estruturada dessa forma permitirá uma varredura em busca de uma determinada posição de forma extremamente mais rápida, uma vez que não será mais necessária a ida ao topo da pilha pois pode-se andar por ela para frente ou para trás.

Assim um bom procedimento para busca de uma determinada posição seria como em (2.9).

```
Procedure Tripaij (posição; var aij_ptr);
begin
while posição < aij_ptr do
    aij_ptr:=aij_ptr^.next;                (2.9)
    while posição > aij_ptr do
    aij_ptr:=aij_ptr^.anterior
end;
```

A versão do Pascal aqui contemplada, oferece três alternativas de operação, sendo que duas delas são apresentadas em forma de menu, quando acionado o ícone "compiler" da barra de menu superior da tela, na versão para DOS, que são :

- REAL MODE

Nessa opção o compilador se restringe à área abrangida pelo sistema operacional, ou seja, dentro dos 640 kbytes como já mencionado.

- PROTECT MODE

Já nessa situação o compilador proteje as áreas de atuação dele próprio e do sistema operacional, a fim de que não sejam acessadas pela alocação dinâmica tais segmentos e opera em toda a memória RAM do equipamento .

O MÉTODO "SPARSE"

A matriz de rigidez global, da forma como é obtida, resulta simétrica e se assim é, não há necessidade de armazená-la por inteiro, pois sempre que for necessária a recuperação de uma posição não armazenada, basta que se invertam seus índices de localização na matriz plena (i e j) e se terá o valor procurado, sendo, portanto, necessário guardar pouco mais da metade da mesma.

Pode-se estocar desta forma os elementos não repetitivos, que assim o sejam por força da similaridade da matriz de rigidez, em um vetor, como proposto em Golub & Van Loan, op. cit. p.22, onde a correlação desejada pode ser formulada como em (3.1).

$$a_{ij} = \left((i-1).n - i.(i-1) \div 2 + j\right) \qquad \text{para todo } j \geq i \qquad (3.1)$$

onde: a_{ij} = número de ordem do elemento da matriz de rigidez da linha i e coluna j, no vetor de armazenagem; n = número de linhas ou de colunas da matriz;

Além da simetria a matriz dos coeficientes tem uma certa esparsidade, ou seja, guarda uma imensa quantidade de zeros, o que permite a construção de um vetor que armazene apenas as posições não nulas de uma de suas metades e ainda procedendo ao seu armazenamento através de um processo de alocação dinâmica de memória, criando um mecanismo que recupere a posição original na matriz de rigidez, desses elementos do vetor proposto posto sob a forma de um conjunto de registros.

Não é difícil criar o algoritmo imaginado quando se conhece a matriz. Mas se ela é conhecida, é sinal de que foi construída e armazenada por inteiro, não interessando mais qualquer recurso para economizar memória de armazenamento, ou seja, há que se criar esse mecanismo sonhado, sem conhecer a matriz de rigidez.

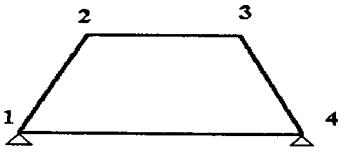
A forma encontrada para vencer esse aparente impasse foi a de montar a matriz linha por linha, ou seja, procedendo a uma varredura da estrutura em busca de barras, que contribuam para uma determinada linha, passando-se à construção dessa linha e armazenando-a no vetor mencionado. A partir da linha construída se faz a busca das posições não zero e o número de ordem que esse elemento teria se fosse armazenado pelo algoritmo (3.1).

Cuidado especial deve ser tomado durante a eliminação gaussiana, pois a mesma altera a esparsidade da matriz de rigidez, de maneira que muitas posições que originalmente guardavam zeros passam a ter não-zeros, exigindo assim um novo registro que deve ser inserido na pilha.

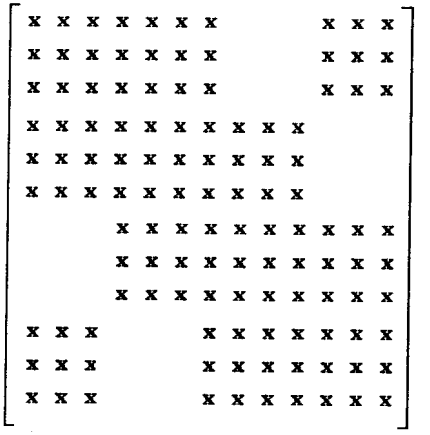
EXPLORANDO A FORMAÇÃO EM BANDA

Os elementos significativos da matriz dos coeficientes, para uma numeração conveniente dos nós da estrutura a ser analisada, acabam definindo uma banda em relação à diagonal e dois métodos de armazenagem dos elementos da referida matriz exploram esse fenômeno que são o chamado método retangular e o "skyline". O método retangular torna-se totalmente inócuo para estruturas ditas fechadas, pois como se pode constatar, o exemplo da figura da página seguinte não proporciona formação em banda o que acaba produzindo o armazenamento total da matriz dos coeficientes.

O método "Skyline", muito utilizado e considerado por muitos autores como bastante eficiente explora, não de forma tão radical como o retangular, a formação em banda da matriz de rigidez. Bahte & Wilson [7] e Wilson et. al [8] discorrem sobre o método e como o utilizaram nos programas SAP4, NONSAP e ADINA. Consiste em definir colunas úteis de gravação tomando como topo da coluna o primeiro elemento não zero e como base o elemento da diagonal armazenando-se então todos os elementos da coluna assim definida.



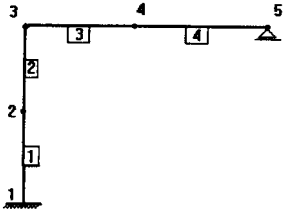
estrutura fechada



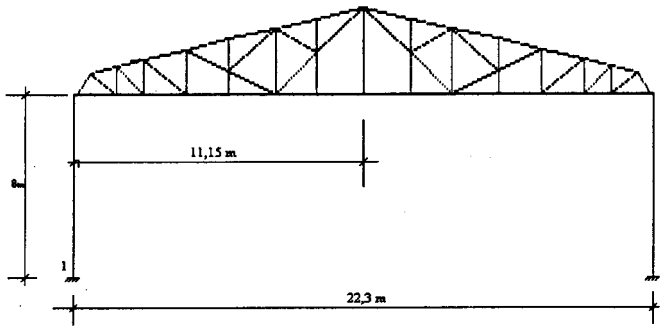
esquemático da matriz dos coeficientes

RESULTADOS OBTIDOS E CONCLUSÕES

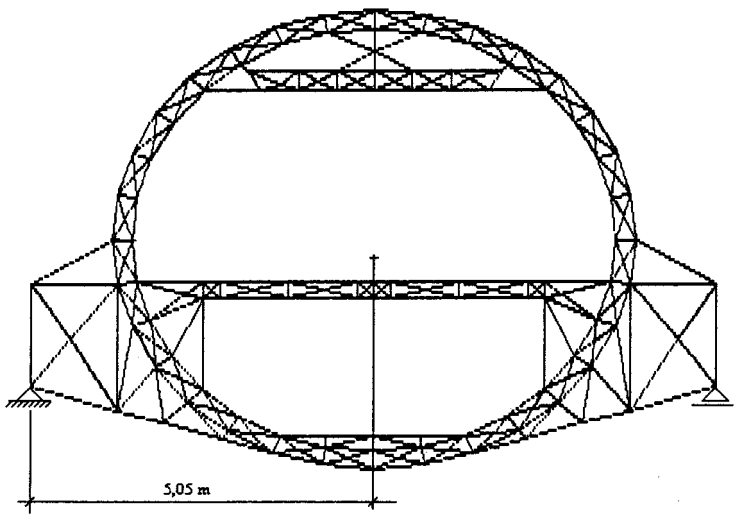
Construídas as duas versões, "Sparse" e "Skyline" a partir do algoritmo básico, foram selecionadas três estruturas para a análise comparativa de desempenho entre as duas versões. A primeira sendo um pequeno pórtico composto por um pilar e uma viga discretizado em quatro barras:



A segunda estrutura é um pórtico de porte médio, tipicamente utilizado na indústria, contendo 69 barras conectadas por 38 nós:



Como terceiro caso uma grande estrutura fechada composta de 195 barras conectadas por 160 nós, impedindo a formação de banda compactada da matriz dos coeficientes:



Tendo em vista a melhor visualização comparativa entre os métodos, foram reunidos alguns resultados em planilhas apresentadas a seguir:

ESTRUTURA 1

	número de elementos da matriz de rigidez	número de elementos armazenados antes da eliminação gaussiana	número de elementos armazenados após a eliminação gaussiana	percentagem de armazengem antes da eliminação gaussiana	percentagem de armazenagem após a eliminação gaussiana	tempo de processamento em segundos
Skyline	225	43	43	19,11	19,11	0,21
Sparse	225	21	37	9,3	16,44	0,17

ESTRUTURA 2

Skyline	12.996	945	945	7,27	7,27	2,52
Sparse	12.996	683	944	5,3	7,26	2,47

ESTRUTURA 3

Skyline	230.400	20.948	20.948	9,09	9,09	49,49
Sparse	230.400	4.181	20.879	1,8	9,06	44,1

Podemos constatar um melhor desempenho em termos de tempo de processamento do método "skyline", sobre o "sparse".

A segunda constatação se refere à grande alteração provocada pelo algoritmo gaussiano na esparsidade da matriz de rigidez. Como pode ser observado na análise do terceiro caso pelo método Sparse, após a intervenção do algoritmo, o número de posições armazenadas foi quintuplicado. Ou seja o número de posições zero no interior da banda definida pelo método Skyline se reduz a quantidades quase insignificantes na matriz [U].

A comparação entre os tempos de processamento, que aponta para uma maior eficiência do método Sparse, obriga a comentar um pouco sobre a elaboração dos programas. Quando foi construida a primeira versão do Sparse, a pilha dinamicamente alocada, como já foi comentado anteriormente, era simplesmente ligada e o tempo gasto para processar a estrutura do caso 2, foi de mais de três horas. Promovendo a dupla ligação entre os elementos da pilha o tempo de processamento caiu para três minutos e armazenando convenientemente alguns endereços ao longo da eliminação gaussiana, com o intuito de reduzir as "viagens" através da estrutura de dados, chegou-se aos pouco mais de 2 segundos observados na versão final. Isso conduz à conclusão de que é uma operação demorada em termos computacionais o manuseio de uma grande estrutura de dados, daí a vantagem do método Sparse sobre o Skyline, pois o segundo inicia a eliminação gaussiana com uma pilha cinco vezes maior.

REFERÊNCIAS

[1]**MOREIRA, Domício Falcão**. *Análise matricial de estruturas*. Universidade de São Paulo Livros Técnicos e Científicos, Rio de Janeiro ,1977.

[2] - **ROTH, P. N.** *A fast pointer based-solved solver for simultaneous linear simultaneous linear equations.* Computer & Structures., vol 4, 1990, págs. 585-612.

[3] - **GOLUB, Gene H. & VAN LOAN, CHARLES F.** *Matrix computations.,* The John Hopkins University, Baltimore, 1989.

[4] - **DUFF, I. S., ERISMAN, A. M., REID, J. K.** *Direct methods for sparse matrices*. Oxford Science Publications, Great Britain, 1986.

[5] - **WILSON, Edward L., Habibullah, A. SAP90**: *A series of computer programs for the static and dinamic finite element analysis of structure, users manual.* Bekerley, 1988.

[6] - **JONES, Jaqueline A. & HARROW, Keith**. *Problem solving with turbo Pascal*. Prentice Hall, New Jersey, 1986.

[7] - **BAHTE, K. J. & WILSON, E. L.** *Numerical methods in finite element analysis*. Prentice Hall, Englewood Cliffs, 1976.

[8] - **WILSON, E. L., BAHTE, K. J. , DOHERTY, W. P.** *Direct solution of large of linear equations.* Computer & Structures. Great Britain, Vol 4, 1974, págs. 363-372.

[9] - **CUTHIL, E. & McKEE, J.** *Reducing the bandwidth of sparse symetric matrices*. Proceedings 24th National Conference of the Association for Computing Machinery. New Jersey, Brandon, p. 157-172, 1969.

[10]- **GEORGE, A.***Computer implementation on the finite element method.* Ph. D thesis. California, Department of Computer Science-Stanford University, 1971.