Utilización de MATRICES DISPERSAS en el Método de los Elementos Finitos

Gino Natale Primero
Ingeniero Civil
Universidad Santo Tomás
Jefe de la División de
Organización y Sistemas
INGEOMINAS

Jesús Rafael García Núñez
Ing. Civil MSc. en Geotecnia
Universidad Nacional de Colombia
División de Geología Ambiental
INGEOMINAS

Alvaro Correa Arroyave
Ingeniero de Minas y Metalurgia Doctor - Ingeniero Mecánica de Rocas
Profesor Asociado Universidad Nacional
Jefe Unidad de Publicaciones - Facultad de Ingeniería

INTRODUCCION

El presente es el primero de una serie de artículos relacionados con el tema de los Elementos Finitos y su aplicación en el campo de la ingeniería, y en particular de la Ingniería Geotécnica, que publicaremos en esta sección.

En el primero de ellos estamos presentando el método ilustrándolo con un ejemplo muy sencillo en el que se determinan los esfuerzos de una viga constituída por un material CHILE bajo el supuesto de deformaciones planas y esfuerzos biaxiales.

La finalidad de este primer artículo es mostrar las ventajas que ofrecen las matrices dispersas para el manejo de la matriz banda que genera un tipo de problemas como el planteado, frente a los sistemas tradicionales de Choleski-Jordan y Gauss-Seidel, entre otros.

Posteriores artículos ilustrarán el método para materiales CHOLE (Continuos, Homogéneos, Ortotrópicos y linealmente elásticos) y otros tipos de materiales en donde el comportamiento elástico no cumple con la linearidad.

INTRODUCTION

This paper is the first of a series of articles relating with the topic of Finite Elements and their application to the engineering field and in particular to geotechnical engineering, which will be published in this section.

In this first article we are presenting the method, ilustrating a very simple example in wich the stresses are determined in a beam made with a CHILE material (continuos, homogeneous, isotropic, linearly elastic), under the assumption of planar deformations and biaxial stresses.

The objetive of the this first article is to show the advantages that the disperse matrices offer in the handling of the banded matrices that is generated by the type of problem presented, with respect to the traditional methods of Choleski-Jordan and Gauss-Seidel, amongst others.

Future articles will illustrate the method for CHOLE materials (Continuos, homogeneous, orthotropic and linearly elastic) and other type of materials where the elastic behaviour does not comply with linearity.

INTRODUCTION

L'article suivant est le premier d'une série d'articles qui vont traiter sur le sujet des éléments finis et son application aux domaines du Génie Civil et tout particuliérement du Génie Géotechnique que nous publicarons dans cette section.

Au premier de ces articles nous sommes en train de présenter la méthode en l'ilustrant avec un exemple trés simple ol l'on cherche les contraintes dans une poutre constituée d'un matériau CHILE sous la supposition des d)formations planes et des contraintes en deux directions.

Le but de ce premier article c'est de montrer les avantages qui offrent les matrices dispersées sur la manipulation de la matrice bande qui génere un type de probléme comme cet-ci face aux syst/mes traditionels de Choleski-Jordan et de Gauss-Seidel, d'entre autres.

Aprés cet-ci ils viendront d'autres articles qui montreront la méthode pour les mat)riaux CHOLE (continus, homogénes, orthotropes et élastiques lineaires) et d'autres types de matériaux ol le comportement élastique n'est plus linéaire.

EL METODO DEL ELEMENTO FINITO

El método del elemento finito es una técnica numérica para resolver problemas de energía, ecuaciones diferenciales complejas, muchas de las cuales describen problemas físicos tales como transferencia de calor, flujo de fluídos, campos magnéticos, distribución de esfuerzos y deformaciones en un medio continuo, consolidación, etc.

A manera de ilustración en la Figura 1.a se presenta una estructura plana en la que interesa calcular los esfuerzos y las deformaciones originadas por la carga distribuida P. para ello, se procede a discretizar la estructura mediante pequeños triángulos y cuadriláteros, Figura 1.b. Los puntos negros que representan la unión entre los diferentes elementos se denominan "puntos nodales" o simplemente 'nodos'. Para el modelo ilustrado cada nodo presenta dos grados de libertad, puesto que cada uno se puede desplazar tanto en la dirección x como en la y, Figura 1.c

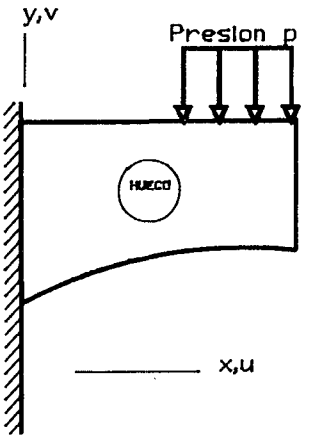


Figura 1.a
Estructura de forma arbitraria.

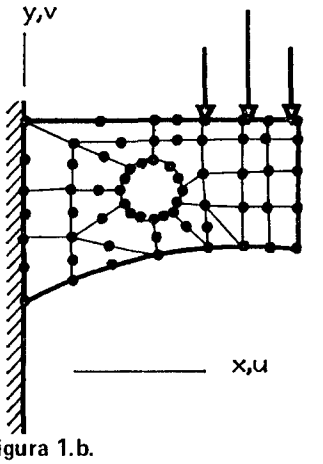


Figura 1.b.
Discretización de la estructura.

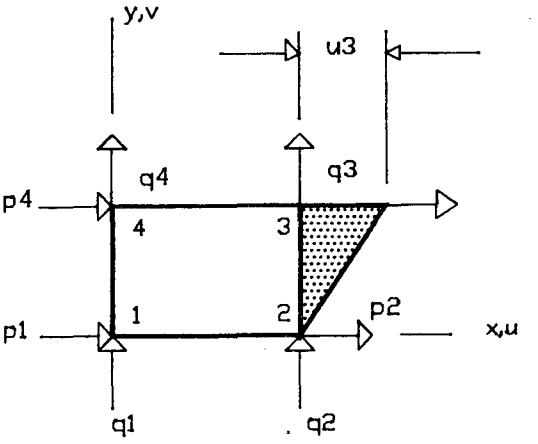
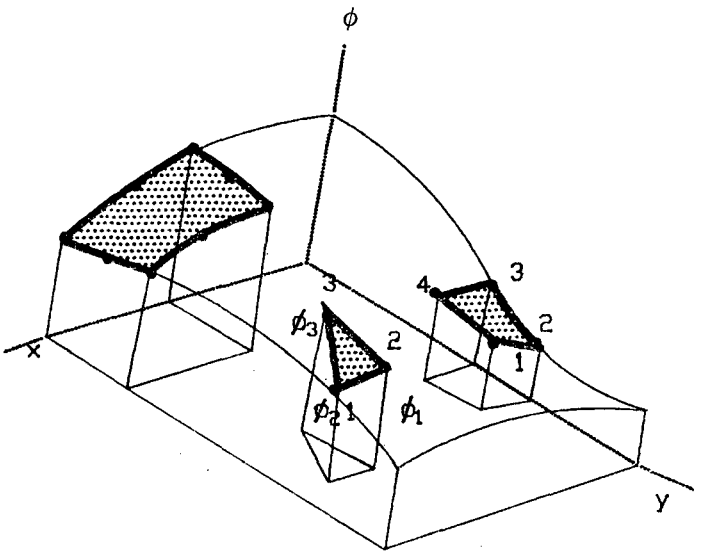


Figura 1.c.
Elemento de forma rectangular mostrando las fuerzas nodales p' y q'. El área punteada representa la deformación asociada con el desplazamiento del nodo 3 (Cook, 1991).

Una propiedad importante del método del elemento finito es que los resultados dependen del tipo de elemento utilizado; es decir, entre más elaborado sea el tipo de elemento, o más pequeños, mejores serán los resultados obtenidos. Entre los elementos más típicos se tienen el elemento triángulo, el elemento cuadrilátero y el isoparamétrico de ocho nodos. Cada uno de estos elementos se representa mediante una función que generalmente corresponde a un polinomio simple como se observa en la Figura 2.



Significado de ϕ en varios problemas:
Torsión: Función de ábaco.
Flujo de fluidos: función de corriente o velocidad potencial.
Infiltración: cabeza hidráulica.
Magnetostática: potencial magnético.
Campo eléctrico: voltaje.
Conducción de calor: temperatura.

Figura 2. La función $\phi = \phi(x,y)$ varía suavemente sobre una región rectangular en el plano x,y. Algunos de los elementos típicos utilizados se ilustran en la figura (Cook, 1991).

Matriz de rigidez del elemento. La conformación de la matriz de rigidez elemental pasa por las siguientes etapas:

1)Formulación del campo de desplazamientos. Esto es, funciones de interpolación que permiten expresar los desplazamientos nodales u y v en términos de los movimientos de los nodos del elemento; para el caso del elemento triángulo de 3 nodos se tienen las siguientes expresiones:

$$u = a_1 + b_1 x + c_1 y$$

$$v = a_2 + b_2 x + c_2 y$$

En forma matricial,

$$u = [x\ y\ 1] \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}$$

$$v = [x\ y\ 1] \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$$

En donde u, v son las componentes de desplazamiento en la dirección x-y.

$a_1 \ldots c_2$ = coeficientes de desplazamiento.

x, y = coordenadas.

Para el elemento triángulo ilustrado en la Figura 3, las coordenadas (x_i,y_i), (x_j,y_j), y (x_k,y_k) determinan la posición de los puntos nodales i, j y k respectivamente, siendo los desplazamientos de cada nodo:

$$u_i = a_1 + b_1 x_i + c_1 y_i$$
$$u_j = a_1 + b_1 x_j + c_1 y_j$$
$$u_k = a_1 + b_1 x_k + c_1 y_k$$
$$v_i = a_2 + b_2 x_i + c_2 y_i$$
$$v_j = a_2 + b_2 x_j + c_2 y_j$$
$$v_k = a_2 + b_2 x_k + c_2 y_k$$

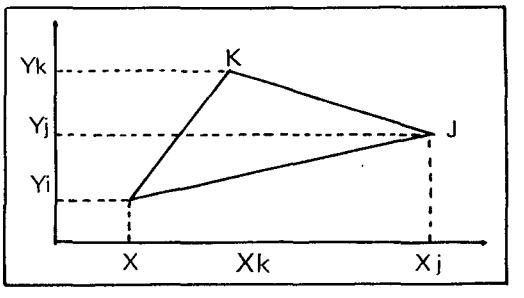


Figura 3. Elemento triangular definido por los nodos i, j, y k, con coordenadas (x_i, y_i), (x_j, y_j) y (x_k, y_k) respectivamente.

Que en forma matricial se expresan:

$$\begin{bmatrix} u_i \\ u_j \\ u_k \end{bmatrix} = \begin{bmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}$$

De donde:

$$\begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = [X^{-1}] \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

De manera similar:

$$\begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = [X^{-1}] \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Agrupando y reorganizando se llega a :

$$\begin{bmatrix} u \\ v \end{bmatrix} = [N] \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix}$$

ó

$$\begin{bmatrix} u \\ v \end{bmatrix} = [N]\ [W]$$

en donde :

[N] = Matriz de interpolación que permite hallar u y v en cualquier parte del elemento, en términos de los desplazamientos de los vértices.

[W] = desplazamientos en los nodos del elemento (ui,vi, uj, vj, uk, vk).

2) Determinación de las deformaciones unitarias en términos de los desplazamientos de los nodos. Las deformaciones unitarias son por definición.

$$\epsilon_x = \frac{\partial u}{\partial x} ; \epsilon_y \frac{\partial v}{\partial y}$$

$$\gamma_{xy} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

Luego la relación existente entre los desplazamientos nodales y las deformaciones unitarias es,

$$\begin{bmatrix} \epsilon x \\ \epsilon y \\ \gamma xy \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} \frac{\partial N}{\partial x} & 0 & \frac{\partial N}{\partial x} & 0 & \frac{\partial N}{\partial x} & 0 \\ 0 & \frac{\partial N}{\partial y} & 0 & \frac{\partial N}{\partial y} & 0 & \frac{\partial N}{\partial y} \\ \frac{\partial N}{\partial y} & \frac{\partial N}{\partial x} & \frac{\partial N}{\partial y} & \frac{\partial N}{\partial x} & \frac{\partial N}{\partial y} & \frac{\partial N}{\partial x} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix}$$

Siendo A = área del elemento en consideración.

Que se acostumbra expresar como:

$[\ \epsilon\ ] = [B][w]$

$[\ \epsilon\ ]$ = Matriz de deformaciones unitarias.

[**B**] = Matriz que contiene solo constantes (caso triángulo) o funciones simples en el caso de elementos más complejos.

3)Relación esfuerzo-deformación. Para un material linealmente elástico el análisis bidimensional presenta las siguientes opciones:

A. Estado plano de deformaciones. En geotecnia un estado de deformaciones plano se presenta cuando se analiza un túnel, un terraplén, un cimiento corrido, un talud, etc. a condición que las deformaciones normales al plano sean nulas. Para este caso la matriz de constantes elásticas está dada por:

$$D = \frac{E}{(1+\mu)(1-2\mu)} \begin{bmatrix} 1-\mu & \mu & 0 \\ \mu & 1-\mu & 0 \\ 0 & 0 & \frac{1-2\mu}{2} \end{bmatrix}$$

B. Estado plano de esfuerzos. En estructuras, por ejemplo, el estado de esfuerzos plano se presenta en vigas, placas con orificio, muros de corte y ensayos eje-simétricos. Para este caso la matriz de constantes elásticas está definida por

$$D=\frac{E}{1-\mu^2}\begin{bmatrix}1 & \mu & 0\\ \mu & 1 & 0\\ 0 & 0 & \frac{1-\mu}{2}\end{bmatrix}$$

La forma general de la expresión anterior es :

$$[\ \sigma\ ] = [D][\ \epsilon\ ]$$

siendo:

[D] = matriz de constantes elásticas.

$[\epsilon\ ]$ = vector de deformaciones unitarias.

$[\sigma]$ = vector de esfuerzos.

4)**Formulación de un sistema de Ecuaciones.** Para ello se utiliza el teorema de la energía potencial mínima, cuyo enunciado es: de todas las posibles deformaciones de una estructura ante unas cargas dadas, aquella que corresponda al equilibrio hace que la energía potencial total sea un valor estacionario y mínimo. La energía por unidad de volumen, U, se define mediante:

$$\delta u=\sigma^T \delta\epsilon$$

$$U=\int_0^{\epsilon} \sigma^T \delta\epsilon$$

(Area bajo la curva Esfuerzo - Deformación)

Además,

$$[\sigma] = [D]\ [\epsilon]$$

$$[\sigma]^T =[\epsilon]^T[D]$$

En un volumen se tendría entonces:

$$U=\tfrac{1}{2}\int_v [\epsilon]^T[D]\ [\epsilon]\delta V$$

Pero,

$$[\epsilon]=[B]\ [W]\Rightarrow[\epsilon]^T=[W]^T[B]^T$$

Reèmplazando en la expresión anterior se obtiene :

$$U=\tfrac{1}{2}\int_v [W]^T[B]^T[D][B][W]\delta V$$

$$U=\tfrac{1}{2}\ [W]^T\ t\iint [B]^T[D]\ [B]\ dxdy]\ [W] \quad (A)$$

Ahora bien, la energía de deformación de un conjunto de barras viene dado por:

$U=\tfrac{1}{2}[P]^T[\delta]$ = Area bajo la curva carga-desplazamiento

Pero,

$$[P] = [K]\ [\delta]\ y$$

$$[P]^T=[\delta]^T[K]^T$$

Siendo

[P] = vector de cargas

[K] = Matriz de rigidez del elemento

$[\delta]$ = Vector de desplazamiento

Reemplazando se obtiene

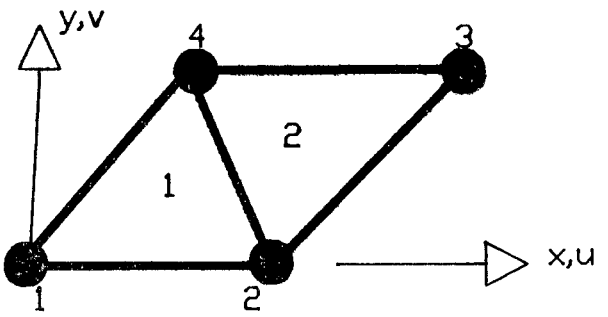
$$U=\tfrac{1}{2}[\delta]^T[K]\ [\delta] \qquad (B)$$

Comparando A y B :

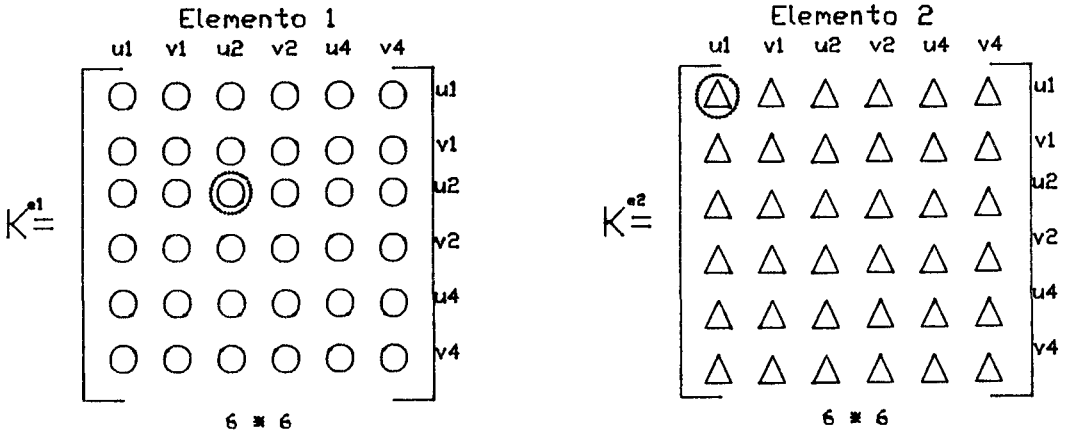
$$[K]=t\iint [B]^T[D]\ [B]\delta x\,\delta y$$

Expresión que representa la matriz de rigidez del elemento.

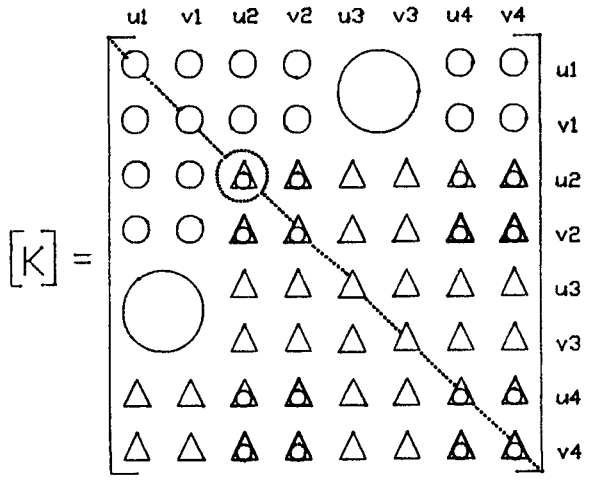
Matriz de rigidez del ensamblaje. Una vez calculada la matriz de rigidez del elemento, se procede a tener en cuenta las respectivas contribuciones para finalmente formar la matriz de rigidez del ensamblaje; este procedimiento es ilustrado esquemáticamente en la Figura 4. Obsérvese que en los sectores donde coinciden las contribuciones, se procede a sumar los valores numéricos correspondientes. Nótese además que existen sectores donde no se producen contribuciones.



a. Continuo conformado por dos elementos.



b. Matriz de rigidez de los elementos.



$\triangle$ = $K_{11}^{e1} + K_{22}^{e2}$ etc, se suman las contribuciones de los elementos.

$\bigcirc$ = Valores nulos.

Figura 4. Matriz de rigidez del ensamble de una estructura hipotética de dos elementos que presenta dos grados de libertad por nodo.

6)**Solución del Sistema de Ecuaciones.** Una vez conformada la matriz de rigidez del ensamblaje, se llega al siguiente sistema:

$$[Q]=[K][\delta]$$

Donde:

$[K]$ = Matriz de rigidez del ensamblaje

$[Q]$ = Matriz de cargas

$[\delta]$ = Matriz de desplazamiento

Al realizar los productos de las matrices se obtiene un sistema de ecuaciones lineales que pueden ser resueltas por diversos métodos. Los más comúnmente utilizados para la solución de dichos sistemas de ecuaciones se ilustran en la Tabla 1.

Métodos Directos	Métodos Iterativos
Eliminación de Gauss	Jacobiano
Eliminación del semiancho de banda	Gauss-Seidel
Eliminación frontal	Gradiente Conjugado

TABLA 1

El método de eliminación de Gauss es tal vez el procedimiento más empleado para la solución de ecuaciones lineales.

En la eliminación directa la matriz [K] se reduce a una forma triangular en la que los desplazamientos desconocidos [δ] son encontrados directamente. Los procedimientos iterativos como Gauss-Seidel y Gradiente Conjugado, incluyen una serie de aproximaciones en la que se estima un valor inicial que es corregido sucesivamente hasta llegar a la solución real.

Incógnitas. Estas cantidades generalmente corresponden a las derivadas de los parámetros ($\epsilon x = \frac{\partial u}{\partial x}$, $\epsilon y = \frac{\partial v}{\partial y}$ y el caso que nos compete): esfuerzos y deformaciones en un elemento discreto.

Propiedades de la matriz de rigidez del ensamblaje. La matriz de rigidez [K], es simétrica y definida positiva; es decir, los coeficientes Kii de la diagonal son siempre positivos y relativamente grandes en comparación con los valores de la misma fila. Estas características son el resultado de la formulación matemática del modelo.

Una característica importante que se utiliza para simplificar los cálculos numéricos es que la matriz de rigidez es una matriz banda; ésto es, todos los coeficientes no nulos están relativamente cerca a la diagonal principal; sin embargo es posible encontrar coeficientes nulos dentro de la banda. La Figura 5 ilustra la manera como queda la matriz de rigidez del ensamblaje una vez realizadas las respectivas contribuciones de los diferentes elementos; de la anterior figura se deduce:

- El ancho de banda depende de la numeración de los nodos; la expresión general para calcular el ancho de banda de [K] es:

$$AB=2x(e_{max}\ [DN^e]+1)$$

Donde:

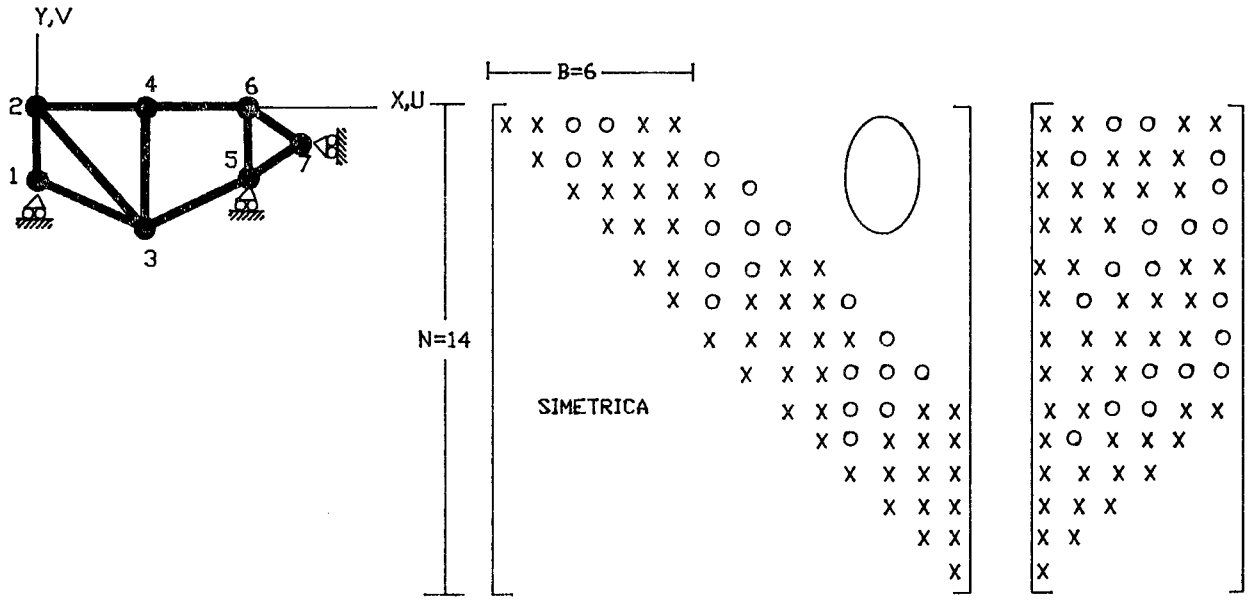
AB = Ancho de banda

DN = Diferencia entre el nodo mayor y menor del elemento "e". El máximo valor de DN es el que se utiliza en la expresión. Para el ejemplo de la figura se tiene:

$$AB=2x[(6-4)+1]=6$$

Luego la manera de minimizar el ancho de banda es numerando los nodos de manera que DN sea el menor posible para todos los elementos.

- Dentro de la banda de la matriz de rigidez existen varias "pseudodiagonales" nulas y por lo tanto se está empleando una buena parte de la memoria del computador para almacenar elementos nulos. Para el ejemplo de la figura la matriz de rigidez consta de 196 valores, pero debido a la simetría de la matriz [K] se pueden simplificar los cálculos analizando únicamente los valores que están por encima de la diagonal principal que son 98. Ahora bien, teniendo en cuenta que la matriz es banda, con un ancho de banda de seis (B=6), la nueva matriz almacenada constará de 69 elementos de los cuales 18 son nulos (26%). Este efecto de valores nulos se incrementa con el número de nodos, restringiéndose de esta manera la memoria del computador.



a. Representación esquemática de la cercha.

b. Matriz de rigidez del ensamblaje (x= coeficientes diferentes de cero).

c. Forma de almacenamiento de la matriz banda. (Cook 1981)

Figura 5. Matrices correspondientes al esquema de la cercha mostrada en a.

OPTIMACION DE LA MEMORIA

Dentro de las principales técnicas numéricas existentes para optimizar la memoria del computador se cuenta con:

El Método del Gradiente Conjugado. El método del gradiente conjugado (MGC) es una técnica para minimizar funciones, pero se puede utilizar para resolver sistemas de ecuaciones lineales cuya matriz sea definida positiva, como es el caso de la matriz de rigidez. En el MGC la matriz ni se modifica ni se descompone (Mora Hector, 1984). Por esta razón los elementos o columnas siguen siéndolo hasta acabar de resolver el sistema. Mediante el método del gradiente conjugado se puede resolver el siguiente sistema de ecuaciones:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2}nx_n = b2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = bn$$

Que se puede escribir en forma matricial:

$$\begin{matrix} a_{11} & a_{12} & \ldots & a_{1n} & x_1 & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & x_2 & b_2 \\ \vdots & & & & \vdots & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{nn} & x_n & b_n \end{matrix}$$

o abreviadamente,

$$[A][x] = [b]$$

En realidad el MGC está enfocado a resolver el siguiente problema equivalente:

$$f(x) = \tfrac{1}{2}[x]^t[A][x] - [x]^t[b]$$

es decir, minimizar

$$f(x_1, x_2 \ldots x_n) = \tfrac{1}{2}\Sigma a_{ii}x_1^2 + \Sigma\Sigma a_{ij} - \Sigma b_i x_i$$

o sea, encontrar

$$x^* = (x_1^*, x_2^*, \ldots, x_n^*)$$

tal que,

$f(x^*) \leqslant f(x)$ para todo x

este $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ será también solución del sistema [A] [x] = [b]

Aunque el método del gradiente conjugado proporciona un ahorro considerable de memoria, presenta las siguientes limitaciones:

- Menor precisión en los cálculos numéricos.
- Mayor tiempo de cálculo debido a que su código presenta muchos ciclos (For, While: en Pascal, Basic o Turbo C; DO: en Fortran). Esto lleva a que la solución de un problema mediante el MGC dure mucho más que utilizando el método de Choleski o el de Gauss.

El Método de Matrices Dispersas. Una matriz dispersa es una estructura de datos que se caracteriza porque su tamaño es relativamente grande. Al hablar de matrices dispersas estamos refiriéndonos por ejemplo a arreglos que contienen más de 400 filas por 400 columnas, en la cual la mayoría de sus elementos contienen el número 0. Por ejemplo una matriz de 700 filas por 800 columnas en la cual 448.000 elementos (80%) almacenan el valor cero, puede considerarse dispersa. Con base en lo anterior, se deducen tres diferencias fundamentales entre una matriz tradicional y una matriz dispersa:

-El tamaño
-El número de elementos con valor cero.
-La forma de almacenamiento y manejo de la memoria.

Representacion de una Matriz Dispersa. Supongamos que deseamos analizar la matriz ilustrada en la Figura 6. Obsérvese que de los 36 elementos que conforman la matriz, solamente 24 son diferentes de cero. Para el caso que nos compete, hay que tener en cuenta que la matriz es definida positiva y simétrica por lo que se requiere construir un algoritmo que solamente almacene los elementos diferentes de cero, para lo cual se utiliza la técnica de apuntadores de memoria definiendo un tipo de dato llamado nodo definido por las variables que se muestran en la Figura 6. Cada elemento almacenado de esta manera gasta 18 bytes de memoria, de manera que en 456 K se podrían manejar 25941 elementos de una matriz dispersa.

PROGRAMA DE ELEMENTOS FINITOS

Cuando se elabora un programa de elementos finitos se deben tener presente las restricciones de memoria que se manifiestan directamente en el número de elementos que conforman la malla. Por ejemplo, para correr un programa de elementos finitos en un microcomputador que tenga 640K de memoria RAM, se requiere como mínimo la siguiente memoria:

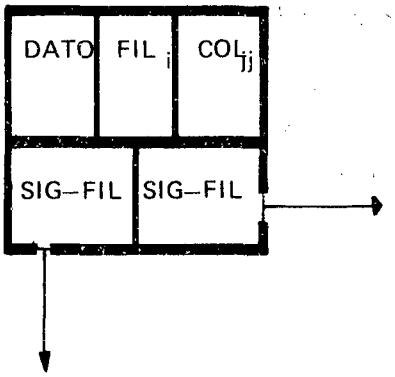
-Cargar sistema operativo
(DOS) 60K
-Cargar programa ejecutable
(xx.EXE) 100K
-Manejo de datos (matrices
Vectores)..... 64K

-Total memoria requerida 224K
-**Memoria Libre** "sobrante" . . **416K**

Los 416 K libres no se utilizarán durante la ejecución del programa debido a restricciones mismas del software (Basic, por ejemplo); sin embargo empleando algunos lenguajes como Pascal y C es posible utilizar esa memoria "sobrante". Para llevar a cabo esta tarea, dichos lenguajes hacen uso de la técnica de apuntadores a memoria que permite el acceso a esa memoria que queda desperdiciada y que algunos autores denominan memoria dinámica o memoria del montón. De esta manera se utilizan mucho mejor los recursos de memoria RAM instalada en el microcomputador. En problemas muy grandes, (con muchos nodos) con seguridad ninguna memoria RAM convencional podrá manejarlos: en estos casos se pueden simular las mismas matrices dispersas que se manejan en memoria, en el disco duro del equipo. De esta manera, si se tienen por ejemplo 10Mb libres en el disco duro se podrá manejar una matriz dispersa con aproximadamente 5'688,888 elementos (ver : BECERRA C., Estructuras de datos en disco duro, Bogotá, Por computador Ltda. 1990, 366 p.).

El diagrama de flujo del programa se ilustra en la Figura 7. El algoritmo utilizado para crear la matriz dispersa se presenta en la Figura 8. Teniendo en cuenta que la matriz de rigidez del ensamblaje es simétrica, solamente se almacenaron los elementos localizados por encima de la diagonal principal y diferentes de cero. Posteriormente se utilizó el método de Gauss para la solución del sistema de ecuaciones pero orientándolo totalmente al uso en memoria dinámica.

ESTRUCTURA DE DATOS BASICA PARA UNA MATRIZ DISPERSA



EN LENGUAJE C

```
struct nodo { double info;
              int    fil;
              int    col;
              struct nodo *sig_fil;
              struct nodo *sig_col; } ;
```

SIG-FIL: siguiente fila
SIG-COL: siguiente columna

E J E M P L O:

MATRIZ SIMETRICA DE 6x6:

$$\begin{bmatrix} 1.11 & 5.25 & 0.00 & 0.00 & 1.55 & 1.66 \\ 5.25 & 2.22 & 6.78 & 0.00 & 0.00 & 6.00 \\ 0.00 & 6.78 & 3.33 & 1.23 & 1.34 & 0.00 \\ 0.00 & 0.00 & 1.23 & 4.44 & 0.000 & 9.65 \\ 1.55 & 0.00 & 1.34 & 0.00 & 5.55 & 5.66 \\ 1.66 & 6.00 & 0.00 & 9.65 & 5.66 & 6.66 \end{bmatrix}$$

REPRESENTACION EN MEMORIA:

VECTOR APUNTADOR A FILAS

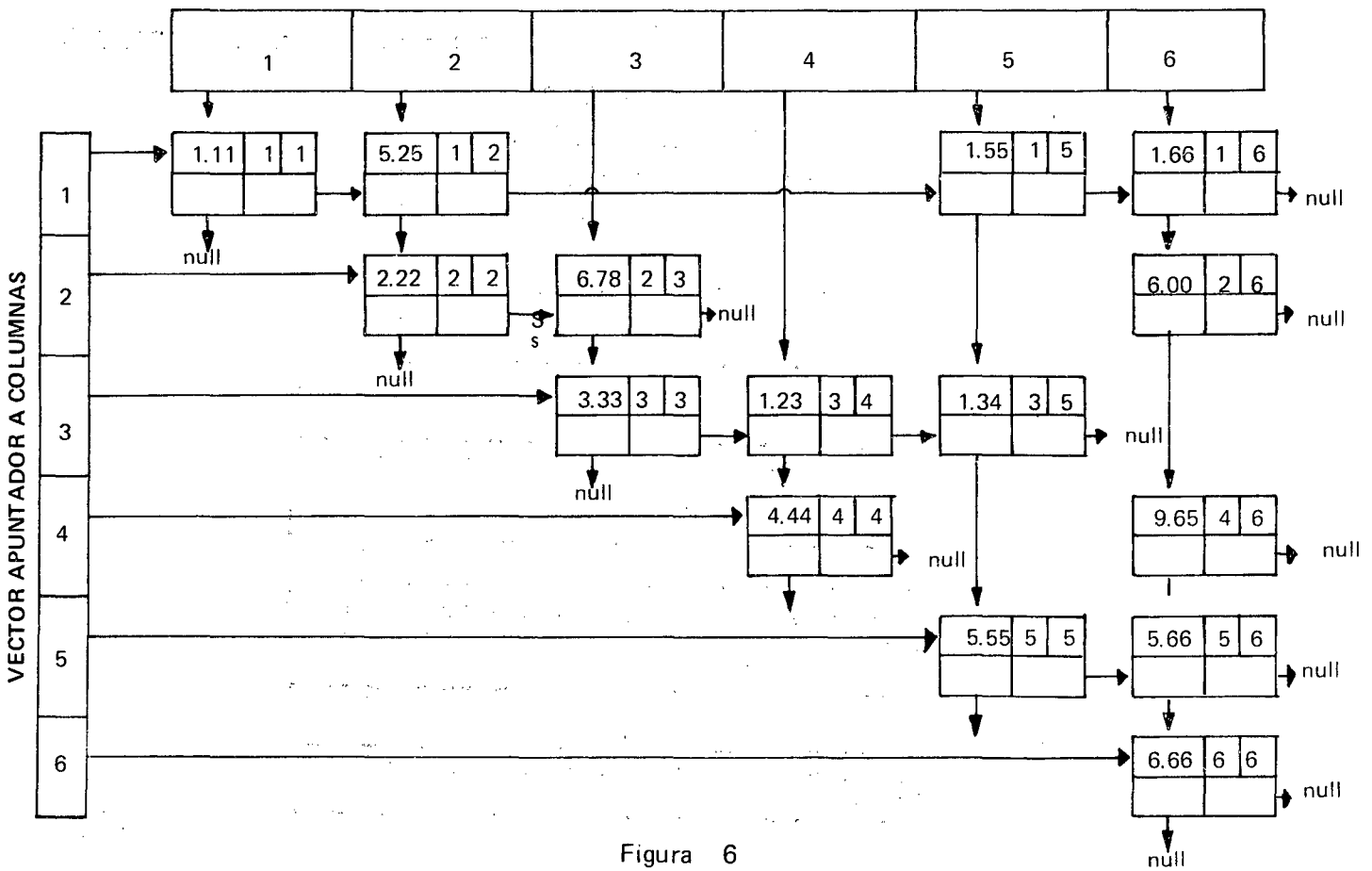
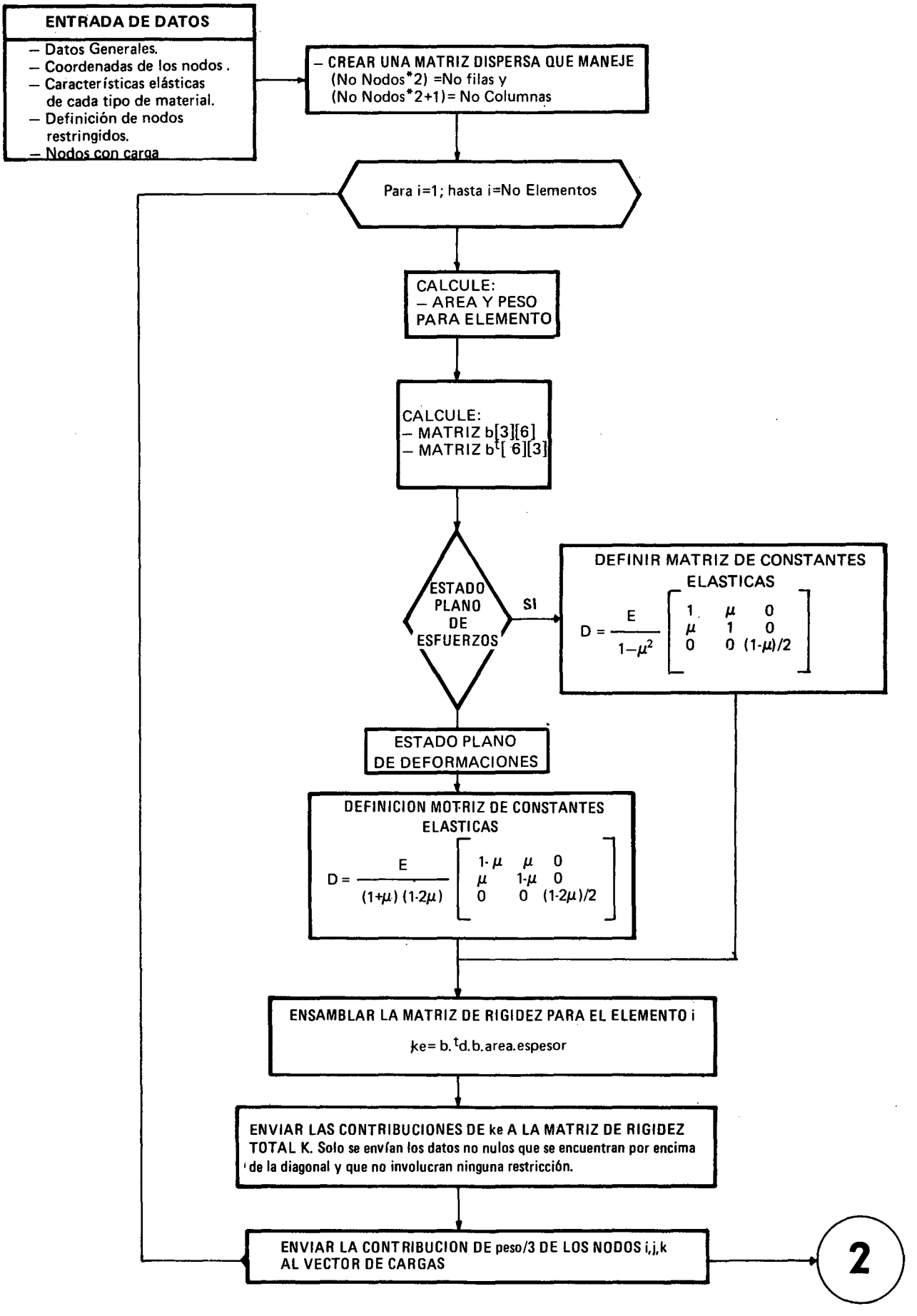


Figura 6

DIAGRAMA DE FLUJO MACRO



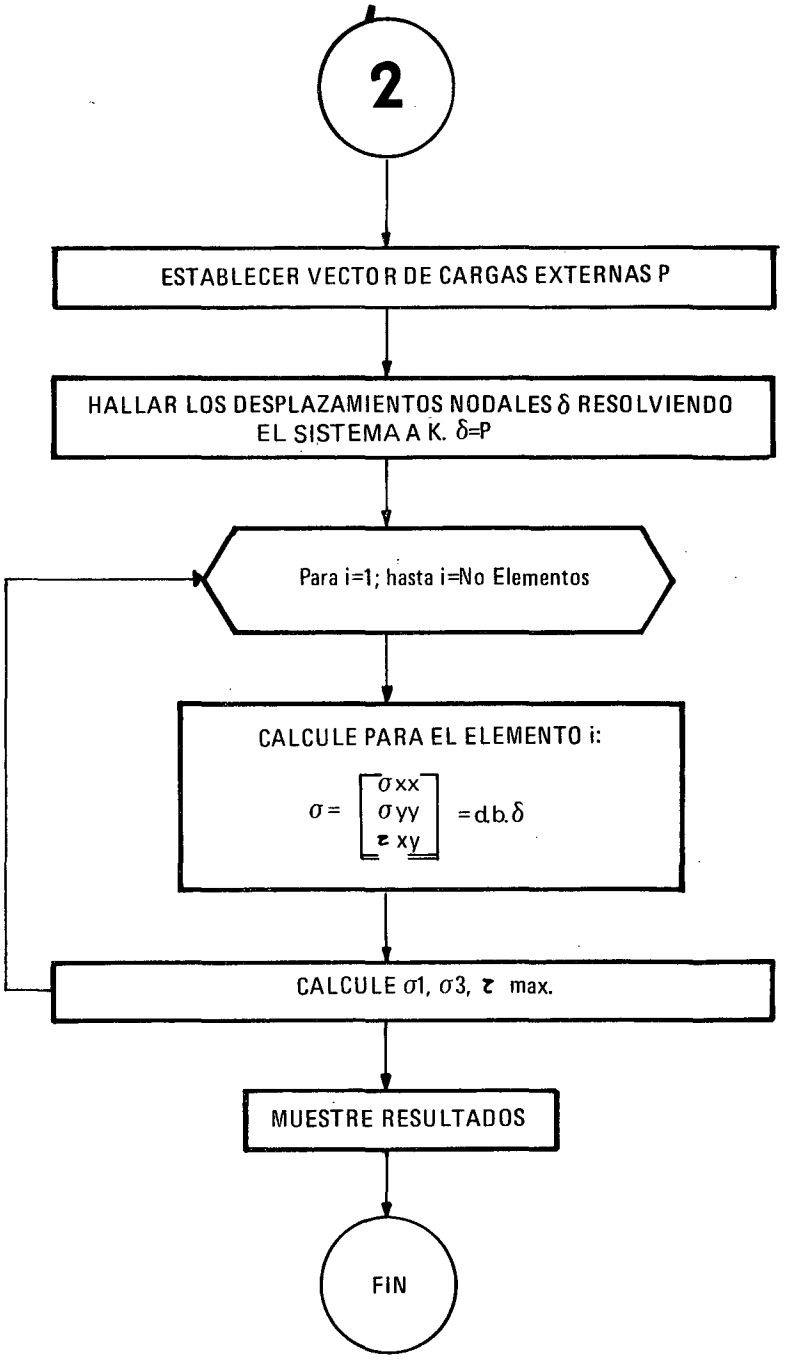

2
ESTABLECER VECTOR DE CARGAS EXTERNAS P
HALLAR LOS DESPLAZAMIENTOS NODALES δ RESOLVIENDO EL SISTEMA A K. δ=P
Para i=1; hasta i=No Elementos
CALCULE PARA EL ELEMENTO i:
σ = [σxx σyy τxy] = d.b.δ
CALCULE σ1, σ3, τ max.
MUESTRE RESULTADOS
FIN

Figura 7

Figura 8. Algoritmo para el manejo de la Matriz Dispersa.

```
//═══════════════════════════════════════════════════════════//
//  Programa : GDispers.c  MAR-91 version 1.0 para C++       //
//  Por      : Gino Natale Primero.                          //
//  Objeto   : Serie de funciones que permiten el manejo de matrices  //
//                dispersas y resolución de ecuaciones por el método de Gauss //
//             para sistemas simetricos                      //
//═══════════════════════════════════════════════════════════//
#include <alloc.h>

static struct nodo { double info;
                     int fil; int col;
                     struct nodo *sigfil;
                     struct nodo *sigcol; };

static struct nodo *vecfil[400],*veccol[400],*arr,*izq;
static int restringido[400];
static double x[400]; //vector solucion//

//═══════════════════════════════════════════════════════════//
//                    funciones prototipos                   //
//═══════════════════════════════════════════════════════════//
void Cree_Dispersa(int nfil,int ncol);
struct nodo *Ubique_Apuntador_Leer(int nfil, int ncol);
struct nodo *Ubique_Apuntador_arr(int nfil, int ncol);
struct nodo *Ubique_Apuntador_izq(int nfil, int ncol);
void Inserte_Nodo(int nfil, int ncol, double dato);
double Disp(int nfil, int ncol);
void Listar_Dispersa(int nfil, int ncol);
void Borre_Dispersa(int nfil);
void Reste_Filas(int Fi, int Fj, int col_ini, int ncol, double mult);
void Gauss_Dispersa(int nfil, int ncol);
//═══════════════════════════════════════════════════════════//

//-----------------------------------------------------------//
//                                                           //
//          Cree_Dispersa                                    //
//                                                           //
void Cree_Dispersa(int nfil,int ncol)
//inicia los vectores vecfil y veccol con los valores pertinentes//
{
  register f,c;

  for (f=1; f<=nfil; f++) vecfil[f]=NULL;
  for (c=1; c<=ncol; c++) veccol[c]=NULL;
}//Cree_Dispersa//
//-----------------------------------------------------------//
//                                                           //
//     Ubique_Apuntador_Leer                                 //
//                                                           //
struct nodo *Ubique_Apuntador_Leer(int nfil, int ncol)
```

```
{
  struct nodo *p_izq,*p,*ant,*izq;
  //busqueda por la izquierda hasta llegar al nodo a leer//
  p_izq=vecfil[nfil];
  if (p_izq==NULL) return(NULL);
  else {  p=p_izq->sigcol;
          while((p_izq->col<ncol) && (p!=NULL))
           { ant=p_izq; p_izq=p_izq->sigcol;
             if (p_izq!=NULL) p=p_izq->sigcol;
                         else p=NULL;
           }//while//
          if (p_izq->col<ncol) izq=ant;
                          else izq=p_izq;
          if ((izq->fil!=nfil) || (izq->col!=ncol)) izq=NULL;
       }//else//
  return(izq);
}//Ubique_Apuntador_Leer//
//-------------------------------------------------------------------------//
//                                                                         //
//   Ubique_Apuntador_arr                                                  //
//                                                                         //
struct nodo *Ubique_Apuntador_arr(int nfil, int ncol)
{
  struct nodo *p_arr,*p,*ant;

  //búsqueda por arriba hasta llegar al nodo anterior al de insertar//
  p_arr=veccol[ncol];
  if (p_arr==NULL) return(NULL);
              else {
                     p=p_arr->sigfil; ant=p_arr;
                     while ((p_arr->fil<nfil) && (p!=NULL))
                       { ant=p_arr; p_arr=p_arr->sigfil; p=p_arr->sigfil; }
                     if (p_arr->fil<nfil) return(p_arr);
                                     else return(ant);
                   }//else//
}//fin Ubique_Apuntador_arr//
//-------------------------------------------------------------------------//
//                                                                         //
//   Ubique_Apuntador_izq                                                  //
//                                                                         //
struct nodo *Ubique_Apuntador_izq(int nfil, int ncol)
{
  struct nodo *p_izq,*p,*ant;

  //búsqueda por la izquierda hasta llegar al nodo anterior al de insertar//
  p_izq=vecfil[nfil];
  if (p_izq==NULL) return(NULL);
              else {
                     p=p_izq->sigcol; ant=p_izq;
                     while ((p_izq->col<ncol) && (p!=NULL))
                       { ant=p_izq; p_izq=p_izq->sigcol; p=p_izq->sigcol; }
                     if (p_izq->col<ncol) return(p_izq);
```

```
                                        else return(ant);
                    }//else//
}//fin Ubique_Apuntador_izq//
//-------------------------------------------------------------------//
//                                                                   //
//              Inserte_Nodo                                         //
//                                                                   //
void Inserte_Nodo(int nfil, int ncol, double dato)
{//Inserta un nodo tipo "struc nodo" en la matriz dispersa; si ya existe//
 //suma el dato anterior con en "dato"                               //

 struct nodo  *nuevo,*prov;

 //Ubicar apuntadores antes de insertar (en el nodo anterior o siguiente)//
 arr=Ubique_Apuntador_arr(nfil,ncol);
 izq=Ubique_Apuntador_izq(nfil,ncol);

 //verificar si se intenta insertar un nodo ya existente, si es el caso sume//
 if (arr->fil==nfil) arr->info+=dato;
 else //proceder a insertar//
  {//pedir memoria ://
   if (( nuevo=(struct nodo *) malloc(sizeof(struct nodo)) )==NULL)
     {clrscr();cprintf("ERROR:\n\r^^^^^^\n\rNo hay memoria disponible!!\n\r");
      exit(1);}
   nuevo->info=dato; nuevo->fil=nfil; nuevo->col=ncol;

   //ajustar apuntadores por arriba://
   if ((arr!=NULL) && (arr->fil>nfil))
      {  prov=veccol[ncol];
         veccol[ncol]=nuevo;
         nuevo->sigfil=prov; }
   else { if (arr==NULL) {veccol[ncol]=nuevo; nuevo->sigfil=NULL; }
                  else { prov=arr->sigfil;
                         arr->sigfil=nuevo;
                         nuevo->sigfil=prov; }
        }//else//

   //ajustar apuntadores por la izquierda://
   if ((izq!=NULL) && (izq->col>ncol))
      {  prov=vecfil[nfil];
         vecfil[nfil]=nuevo;
         nuevo->sigcol=prov; }
   else { if (izq==NULL) {vecfil[nfil]=nuevo; nuevo->sigcol=NULL; }
                  else { prov=izq->sigcol;
                         izq->sigcol=nuevo;
                         nuevo->sigcol=prov; }
        }//else//
 }//fin else nodo existente//
}//Inserte_Nodo//
//-------------------------------------------------------------------//
//                                                                   //
//  f         Disp                                                   //
```

```
//                                                                    //
double Disp(int nfil, int ncol)
//funcion que devuelve el valor de la posicion NFIL,NCOL de la matriz//
{
izq=Ubique_Apuntador_Leer(nfil,ncol);
if (izq==NULL) return(0.0);
         else return(izq->info);
}//Disp//
//--------------------------------------------------------------------//
//                                                                    //
//          Listar_Dispersa                                           //
//                                                                    //
void Listar_Dispersa(int nfil, int ncol)
{
 struct nodo *Fij;
 register f,c;

 for (f=1; f<=nfil; f++)
   {  cprintf("%2d)",f); Fij=vecfil[f];
      for (c=1; c<=ncol; c++)
        { if (Fij->col!=c) cprintf("      ");
                   else { cprintf("%11.3f ",Fij->info);
                          Fij=Fij->sigcol;
                        }//else//
        }//for c//
        cprintf("\n\r");
   }//for f//
}//Listar_Dispersa//
//--------------------------------------------------------------------//
//                                                                    //
//          Borre_Dispersa                                            //
//                                                                    //
void Borre_Dispersa(int nfil)
{
  struct nodo *Fij,*sig;
  register f;

  for (f=1; f<=nfil; f++)
    { Fij=vecfil[f]; if (Fij!=NULL) sig=Fij->sigcol;
                                else sig=NULL;
      while (Fij!=NULL)
        { free(Fij); Fij=sig;
          if (sig!=NULL) sig=sig->sigcol;
        }//while//
    }//for f//
}//Borre_Dispersa//
//--------------------------------------------------------------------//
// LOS SIGUIENTES PROCEDIMIENTOS SON UTILIZADOS POR GAUSS_DISPERSA    //
//--------------------------------------------------------------------//
//                                                                    //
//          Reste_Filas                                               //
//                                                                    //
```

```
//realiza filaj=filaj-mult*filai desde una columna dada//
void Reste_Filas(int Fi, int Fj, int col_ini, int ncol, double mult)
{
  register c;
  struct nodo *p_Fi,*p_Fj;
  int si_hay_j,false=0,true=1;
  double resta,datoi,datoj;

  p_Fi=vecfil[Fi]; p_Fj=vecfil[Fj]; //punteros iniciales por la izquierda//

  //ubicar apuntadores en la columna inicial en la que inicia la resta//
  if (p_Fi->col<col_ini) while (p_Fi->col<col_ini) p_Fi=p_Fi->sigcol;

  if (p_Fj->col<col_ini) while (p_Fj->col<col_ini) p_Fj=p_Fj->sigcol;

  for (c=col_ini; c<=ncol; c++)
    { if (!restringido[c])
        { si_hay_j=false;

          if (p_Fi->col==c) { datoi=p_Fi->info; p_Fi=p_Fi->sigcol; }
                        else datoi=0;

          if (p_Fj->col==c) { datoj=p_Fj->info; si_hay_j=true; }
                        else datoj=0;

          resta=datoj-mult*datoi;
          if (si_hay_j) { p_Fj->info=resta; p_Fj=p_Fj->sigcol; }
          if ((resta!=0) && (si_hay_j==false)) Inserte_Nodo(Fj,c,resta);
        }//if !//
    }//for c//
}//Reste_Filas//
//-------------------------------------------------------------------//
//                                                                   //
//          Gauss_Dispersa                                           //
//                                                                   //
void Gauss_Dispersa(int nfil, int ncol)
//se supone que ningún pivote es igual a cero//
{
//double casi_cero=1.0E-20; fabs molesta//
double tope, pivote, mult;
register ff,cc,k;

for (ff=1; ff<=nfil-1; )
  { // toma como pivote cada elemento de la diagonal, no restringido//
    while (restringido[ff]) ff++;
    pivote=Disp(ff,ff);
    for (cc=ff+1; cc<=nfil; )
      { while (restringido[cc]) cc++;
        mult=Disp(ff,cc)/pivote;

        // filac=filac-mult*filaf//
        //if (fabs(mult)>casi_cero)//
```

```
            Reste_Filas(ff,cc,cc,ncol,mult);
            cc++;
          }//for cc//
        ff++;
      } //for ff//

//sustitución hacia atrás//
ff=nfil;  while(restringido[ff]) ff--;
tope=Disp(ff,ff);
if (tope==0) x[ff]=0;
        else x[ff]=Disp(ff,ncol)/tope;

for (ff=nfil-1; ff>=1; )
  { while(restringido[ff]) ff--;
    tope=Disp(ff,ncol);
    for (k=ff+1; k<=nfil; )
      { while(restringido[k]) k++;
        tope-=Disp(ff,k)*x[k]; k++;}

    mult=Disp(ff,ff);
    if (mult==0) x[ff]=0;
           else x[ff]=tope/mult;
    ff--;
  } //for ff//
} //Gauss_Dispersa//
//-------------------------------------
```

EJEMPLO DE APLICACION

Con el fin de mostrar la eficiencia y rapidez del método se analizó la viga simplemente apoyada ilustrada en la Figura 9. Las características geométricas de la malla son las siguientes:

-Número de elementos. 72
-Número de nodos 52
-Número de nodos restringidos 5
-Número de nodos cargados 13
-Número de ecuaciones 98

Los materiales utilizados en el problema tienen las siguientes características :

-Módulo de elasticidad 29000 PSI
-Relación de Poisson 0.3
-Peso Unitario 0.

Durante la ejecución del programa se utilizaron 31136 bytes de 427920 bytes disponibles; el tiempo de ejecución del programa fue de 13 segundos en un microcomputador DTK 80386 a 25 Mhz con coprocesador matemático. El mismo programa ejecutado en un AT 80286 a 12 Mhz sin coprocesador necesitó 121 segundos.

Finalmente los resultados obtenidos se ilustran en la Figura 10. Para el presente caso solamente se analizó el esfuerzo principal mayor (σ_1), el esfuerzo principal menor (σ_3) y el esfuerzo cortante. Para la elaboración de las gráficas el programa utiliza una subrutina que genera un archivo, que puede ser leído desde un programa gráfico (Surfer).

BIBLIOGRAFIA

Abel, J. Introduction to the Finite Element Method. New York; Van Nostrand Reinhold Company, 1985, 219 p.

Alarcón A.E., Alvarez R., Cálculo matricial de estructuras. Barcelona: Reverté S.A, 1986, 410 p.

Becerra C., Estructuras de datos en disco duro. Bogotá: computador Ltda. 1990 366 p.

Becerra C., Lenguaje C. El nuevo concepto (4a. ed.) Bogotá. Por computador Ltda., 1990, 532 p.

Burden L.R., Faires J.D., Análisis numérico. México: Grupo Editorial Iberiamérica S.A., 1985. 274 p.

C.A. Debria. JJ. Connor. Métodos de los elementos finitos en la Ingeniería Civil. España: Edix S.A., 1975 279 p.

Cohn D. Autocad. Addison - Wesley Publishing Company. 1988.

Cook R. Malkus D. Concepts and Aplications of Finite Element Analysis. Jonh Wiley, 1989.

Gouri Dhatt, Gilbert T. The finite element method displayed. New York: A Wiley-interscience publication, 1985,509 p.

Jamsa, K. Nameroff. S. Turbo pascal programer's library. USA.: McGraw-Hill, 1987. 539 p

Knight. R., Using Autocad. Que Corporation. 1989.

Livesley. R. K. Elementos finitos Introducción para Ingenieros. México D.F.: Limusa S.A., 1988, 224 p.

Mora H. Aplicación del método de Gradiente conjugado al programa PORTRECT. Universidad Nacional de Colombia, Tesis de Grado. 1984.

Parra F. Notas sobre la solución de problemas de Mecánica del continuo. Universidad Nacional. 1986.

Sagerlind L. Applied Finite Element Analysis. John Wiley and Sons. 1984.

Schildt. H., Turbo C The complet reference (2ndEd). Usa: McGraw-Hill, 1990, 749 p.

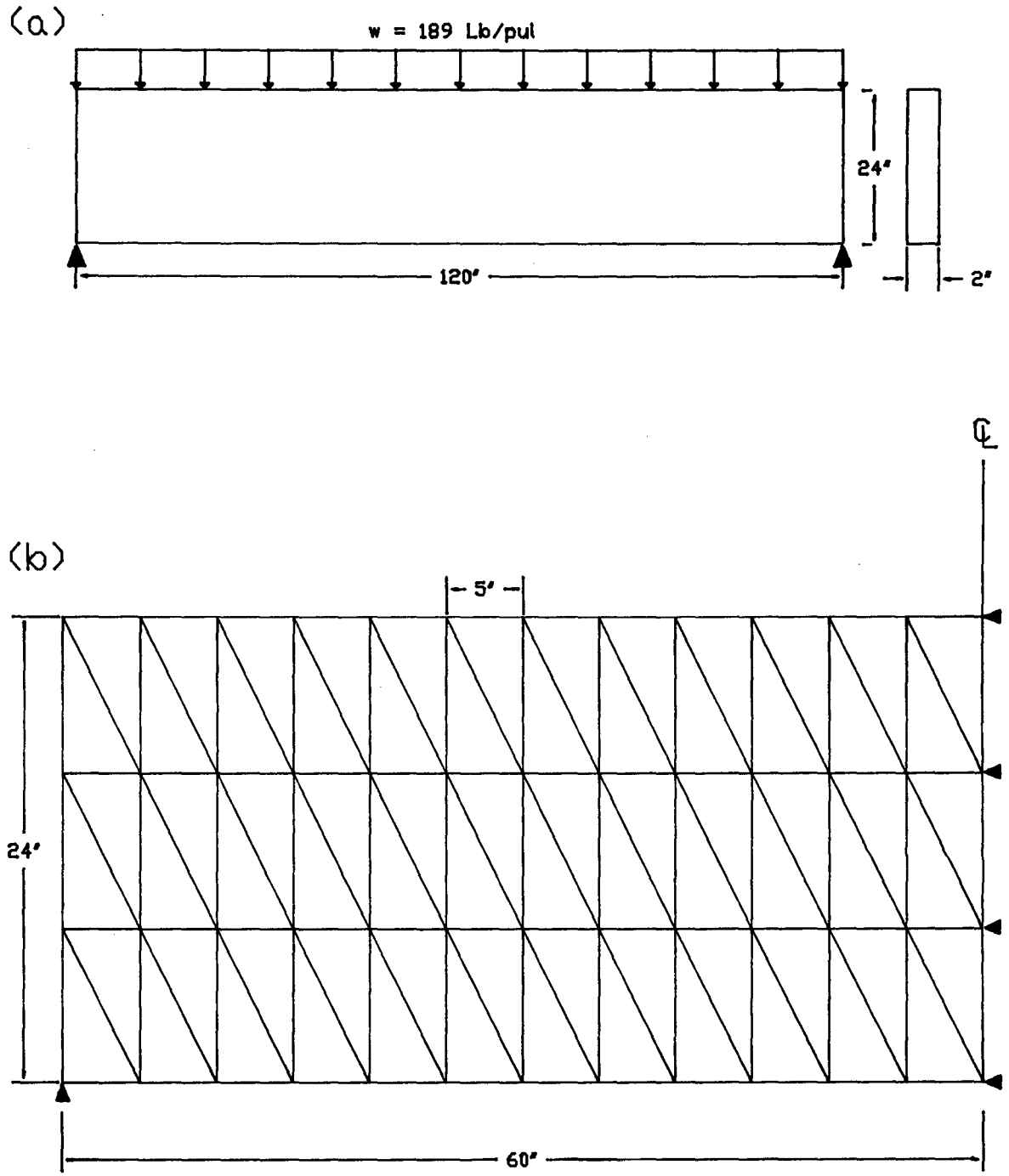
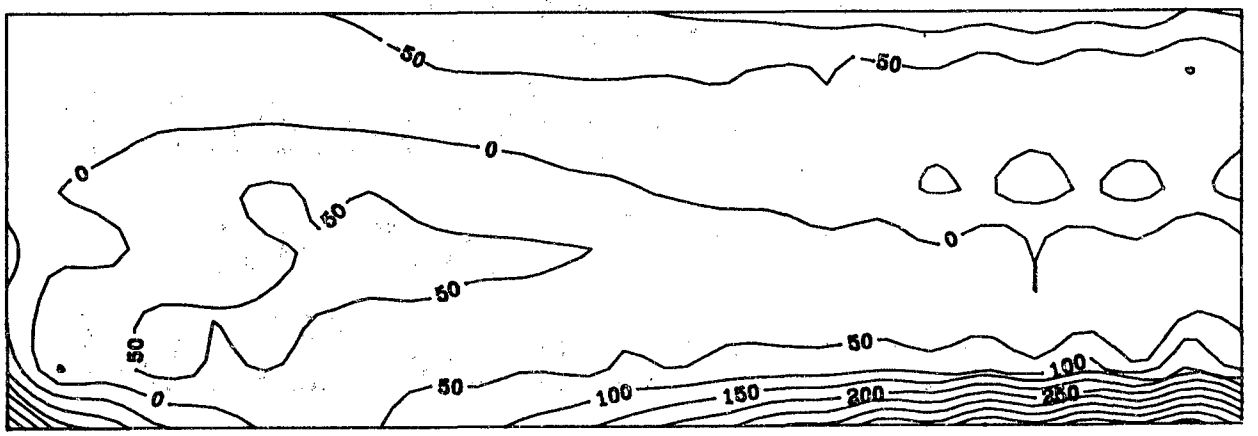
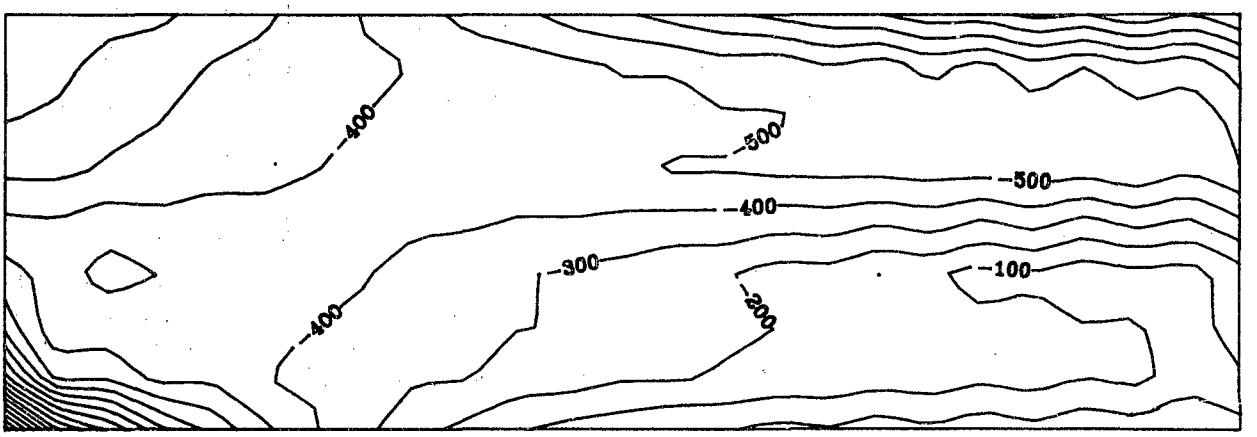


Figura 9. (a) Viga simplemente apoyada. (b) discretización de la viga; dada la simetría de la viga se analizó la mitad.

DISTRIBUCION DEL ESFUERZO PRINCIPAL MAYOR



DISTRIBUCION DEL ESFUERZO PRINCIPAL MENOR



DISTRIBUCION DEL ESFUERZO CORTANTE

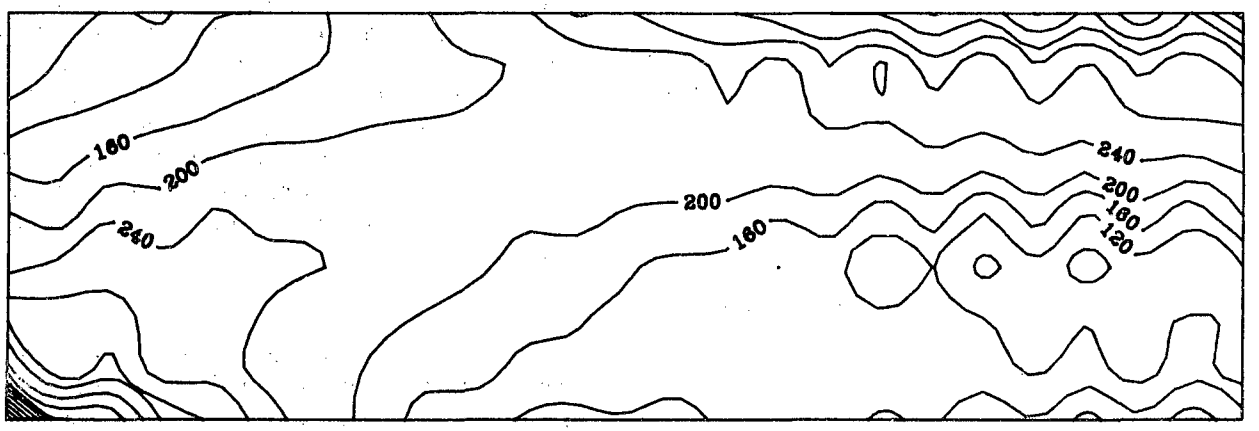


Figura 10. Distribución de esfuerzos en la viga.