Universidad Nacional Autónoma de México

Facultad de Ingeniería

DISEÑO DE UNA LIBRERIA DE PROPOSITO GENERAL PARA EL DESARROLLO DE PROGRAMAS DE ELEMENTO FINITO

T E S I S

Que para obtener el título de

INGENIERO CIVIL

p r e s e n t a

JESUS FAVELA VARA

México, D. F. 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

I. INTRODUCCION

La solución de problemas ha sido identificado como el principal objetivo de la ingeniería, y es al proceso sistemático utilizado para ello al que se conoce como diseño[1].

Este proceso, que tiene por objeto el encontrar un sistema que cumpla satisfactoriamente con ciertos requisitos, incluye las siguientes etapas: la formulación del problema, el planteamiento de uno o varios espacios-solución, el análisis de los mismos y, finalmente una etapa de decisión.

Dado que la toma de decisiones se plantea sobre principios de optimación, el proceso de diseño adquiere un carácter intrínsecamente iterativo en el que el sistema que aún es sujeto a mejoras es modificado y sometido nuevamente a la etapa de análisis para su evaluación.

El objetivo del análisis consiste en determinar el comportamiento que tendrá el sistema en condiciones de operación. El proceso de análisis puede ser separado en tres fases: la primera implica la abstracción del sistema en un

modelo (físico o matemático); en la segunda se determina la respuesta del modelo; y finalmente los resultados obtenidos en la fase anterior son examinados, si éstos son aceptables se extrapolan al sistema físico, de otra manera el modelo utilizado debe ser modificado.

Los modelos matemáticos que describen los sistemas más frecuentes en ingeniería son sumamente complejos y por lo tanto no tienen generalmente solución analítica. Por ello era una práctica común el que el análisis se realizara haciendo grandes simplificaciones al modelo; la etapa de simplificación así como la evaluación del comportamiento del modelo eran sustentados más en la experiencia que en el conocimiento científico. Los modelos matemáticos podían ser planteados, pero no resueltos.

Actualmente, la situación ha cambiado radicalmente, ya que con el avance de la tecnología de las computadoras algunos métodos matemáticos prácticamente olvidados por requerir de un gran número de cálculos, han sido reconsiderados y muchos más se han creado. Así, el análisis matemático y los métodos numéricos han cobrado fuerza convirtiéndose en herramientas útiles al ingeniero.

En particular el método que ha tenido una dependencia directa con el advenimiento y posterior desarrollo de las computadoras es el de los elementos finitos, método que si bien no con ese nombre, se remonta a principios de siglo, y no es, sino hasta la llegada de las computadoras digitales cuando el método se desarrolla vertiginosamente mostrando su verdadero potencial. La razón es clara: sin la ayuda de herramientas que lleven a cabo los numerosos cálculos que el método requiere, su uso sería prácticamente imposible.

Para dar una idea del enorme campo de aplicación del método baste decir que éste se usa en problemas tan variados como el diseño de una presa, la predicción del tiempo, la distribución de temperaturas en un cuerpo, el análisis sísmico de un edificio, etc. Su aplicación es posible a problemas transitorios o estacionarios, lineales o no lineales, con geometrías complejas en una, dos y tres dimensiones y combinando materiales con distintas características.

No obstante lo anterior, el método de los elementos finitos no es muy usado en nuestro medio, ni siquiera en ambientes académicos donde frecuentemente se le menciona, ya

que al método se le asocia gran complejidad tanto teórica como de programación, considerándosele con frecuencia instrumento exclusivo de investigadores y especialistas en el área con acceso a grandes equipos de cómputo. Sin embargo estos inconvenientes no son inherentes al método, sino más bien se derivan de problemas de difusión, ya que la mayoría de los textos existentes presentan los fundamentos teóricos del método obviando los procedimientos computacionales que requieren de un gran esfuerzo de programación y para cuyo desarrollo no se cuenta, en general, con herramientas de apoyo.

Ante este panorama, el presente trabajo tiene como principal objetivo el explicar en forma sencilla (acaso a costa de formulaciones teóricas rigurosas) el método de los elementos finitos, y proveer de una herramienta que facilite su uso: una librería de rutinas o subprogramas de elemento finito.

Con la librería se pretende cambiar la filosofía vigente de diseño de programas de elemento finito, en la que el programador necesita escribir todas los elementos que componen el programa. Por el contrario, el uso de librerías provee de

una serie de rutinas de propósito general en la etapa de programación del método, las que al ser integradas a otros módulos (incluyendo un programa principal) escritos por el usuario permiten estructurar programas en menos tiempo, con menor esfuerzo y generalmente de mayor calidad que el que resultaría si se programase todo desde el principio.

El diseño de una librería con estas características incluye una primera etapa en la que se identifican los procedimientos numéricos de carácter general del método de los elementos finitos, seguida de una etapa en la que estos procedimientos son programados, y terminando con una fase de prueba tanto de las rutinas programadas como de la eficiencia que proporciona el uso de la librería. Estas tres etapas son presentadas a lo largo de este trabajo.

En el capítulo dos se da una breve introducción al método de los elementos finitos, cuya presentación no sigue un planteamiento matemático riguroso, sacrificio que se hace en aras de un mejor entendimiento de los principios generales y procedimientos numéricos que caracterizan al método, prestando especial atención al proceso que se sigue en la implantación del mismo en computadoras digitales.

En el tercer capítulo se destaca la conveniencia del uso de librerías en la programación del método de los elementos finitos y se hacen consideraciones generales para su uso, estableciéndose tanto los principios que se siguieron para su desarrollo, como las características y organización de las mismas.

Los procedimientos numéricos necesarios en el uso del método de los elementos finitos se analizan en el capítulo cuatro. Se presentan algoritmos de operaciones matriciales, solución de sistemas de ecuaciones, integración numérica, interpolación, así como de solución de sistemas de ecuaciones diferenciales ordinarias, dándosele un especial interés a sus aspectos de programación en computadora.

El capítulo cinco se dedica a presentar las aplicaciones del método en distintas áreas, particularmente aquéllas relacionadas con la ingeniería civil. Se desarrollan rutinas para derivar las matrices elementales para cada aplicación y se integran aquellas rutinas elaboradas en el capítulo anterior necesarias al proceso de solución en cada caso, según sea requerido.

En la aplicación del método de los elementos finitos, el mayor esfuerzo y tiempo suele concentrarse en la preparación de los datos y en el análisis de los resultados. En éste campo numerosos esfuerzos han sido realizados para proveer al usuario de utilerías para la generación automática de datos y despliegue gráfico de entradas y resultados; estos elementos, denominados de pre y postprocesamiento se analizan en el capítulo 6.

Con el propósito de ilustrar la ventaja que el uso de librerías ofrece en la programación del método, se presentan en el capítulo 7 algunos ejemplos de aplicación en el área de ingeniería civil.

Finalmente, en el capítulo 8 son presentadas las conclusiones, se evalúan los resultados obtenidos con el uso de la librería y se hacen sugerencias para trabajos que sean desarrollados posteriormente en este mismo campo.

---

1. Krick,E. An Introduction to Engineering & Engineering Design. 1969. U.S.A. John Wiley & Sons, Inc. pág. 28.

II. EL METODO DE LOS ELEMENTOS FINITOS

A. INTRODUCCION

Al presentarse el método de los elementos finitos con este nombre en 1960, era considerado como una extensión de las técnicas de análisis matricial de estructuras. El uso de estas técnicas suponen a la estructura como un conjunto de elementos (barras, trabes, etc.) conectados entre sí en puntos llamados nodos apartir del cual se deriva un conjunto de ecuaciones de equilibrio cuya solución determina los desplazamientos en los nodos y las deformaciones y esfuerzos de la estructura.

El análisis estructural de armaduras y marcos es ahora considerado como un caso particular del método de los elementos finitos con la particularidad de que los esfuerzos y deformaciones obtenidos son exactos de acuerdo con la teoría de la mecánica de los materiales[1].

Existen dos tipos de formulaciones del método de los elementos finitos, una basada en el análisis matricial de estructuras a la que se le conoce como formulación física o directa, y otra en la que un problema en un dominio continuo es modelado ya sea por medio de ecuaciones diferenciales, o bien, como un valor extremo (máximo o mínimo) de un principio variacional, llamada la formulación matemática del método. Ambas formulaciones son equivalentes, obteniendose los mismos resultados si son planteadas correctamente.

Por ser más generales que la formulación directa, las dos formulaciones matemáticas serán presentadas en este capítulo, precedidas de una descripción general del método de los elementos finitos. Se indican finalmente los lineamientos generales para la implantación del método en un sistema de cómputo.

B. DESCRIPCION GENERAL DEL METODO

A diferencia del conocido método de diferencias finitas donde son las ecuaciones que describen el fenómeno las que son discretizadas, con el método de los elementos finitos se

discretiza el dominio de la variable del problema en un número finito de elementos.

El procedimiento general del método puede ser dividido en los siguientes pasos:

1. Discretización del medio contínuo

En esta primera etapa la región en la que ocurre el fenómeno de interés (medio contínuo) es dividida por puntos, líneas o planos imaginarios, formando regiones más pequeñas llamadas elementos finitos. Estos elementos finitos están unidos entre sí en puntos llamados nodos; a cada nodo se le asocia uno o varios grados de libertad. Estos grados de libertad son los parámeros desconocidos que gobiernan al problema en estudio y representan el dominio, ahora discreto, de las variables de campo (incógnitas del problema) que pueden ser por ejemplo desplazamientos, temperaturas, potenciales, etc.

Los elementos finitos pueden tener distintas formas, dependiendo desde luego de la forma que tenga la región global, procurándose que los elementos se ajusten de la

mejor manera posible a ésta. Así, los elementos serán dimensionalmente equivalentes a sus regiones. Los problemas de n dimensiones requieren de elementos también de n dimensiones; líneas en una dimensión; triángulos, cuadriláteros, etc. en dos dimensiones; y tetraedros, hexaedros, etc. en tres dimensiones.

En la figura II.1 se ilustra una región de dos dimensiones discretizada usando triángulos como elementos finitos.

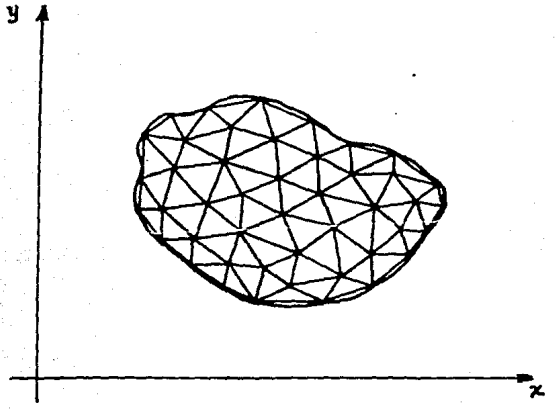

Fig. II.1

2. Selección de las funciones de interpolación

Una vez que se han determinado el número finito de puntos (nodos) en el dominio, se seleccionan las funciones de interpolación las que se representan en términos de parámetros nodales que aproximan adecuadamente a las variables de campo y si el problema lo requiere, a sus derivadas.

De la adecuada selección de las funciones de interpolación dependerá, en gran medida, la convergencia del método para un problema determinado, por lo que este paso es considerado como estratégico en la aplicación del método.

El uso de las funciones de interpolación permite expresar los valores de las variables de campo para cada elemento en función de las coordenadas de los nodos que lo forman. Por ejemlo la variable y puede ser aproximada de la siguiente forma:

$$\{y\} = [N]\{a\}$$

donde [N] representa a las llamadas funciones de forma o de interpolación, que dependen de las coordenadas

de los nodos, y el vector {a} contiene a los parámetros nodales que interesa conocer.

Comunmente se utiliza la interpolación polinomial para obtener las funciones de forma, ya que éstas presentan la ventaja de ser fáciles de integrar y diferenciar, sin embargo en ocasiones llegan a ser utilizadas funciones trascendentes para este efecto.

Las funciones de forma deben también satisfacer las condiciones de frontera locales de cada elemento, cumpliendo con la restricción de tener un valor unitario en un punto nodal y un valor nulo en los demás puntos.

3. Derivación de las ecuaciones de elemento finito

Utilizando métodos variacionales o de residuos pesados, las ecuaciones que gobiernan el problema son transformadas en ecuaciones de elemento finito que relacionan a las variables de interés para cada elemento. Si bien estas ecuaciones pueden en ocasiones plantearse en forma directa a partir del fenómeno físico, esta

formulación no siempre puede aplicarse.

Las ecuaciones elementales, como se verá en el siguiente inciso, se obtienen al sustituir las funciones de interpolación seleccionadas en el paso anterior en las ecuaciones integrales que modelan al fenómeno.

4. Ensamble de las ecuaciones elementales y aplicación de las condiciones de frontera.

Generadas las ecuaciones correspondientes a cada elemento, estas son ensambladas en ecuaciones globales que representan el comportamiento de todo el sistema.

El proceso de ensamble consiste en sumar ordenadamente los coeficientes de las ecuaciones de cada elemento en el lugar que en el sistema de ecuaciones indiquen los grados de libertad asociados a los nodos que pertenecen a tal elemento. El procedimiento es semejante al utilizado en la formación de ecuaciones generales en problemas de redes o en el cálculo matricial de estructuras.

Las condiciones de frontera del sistema son aplicadas durante el proceso de ensamble, o bien, una vez que las ecuaciones globales de equilibrio han sido generadas.

5. Solución de las ecuaciones de equilibrio

El siguiente paso, una vez que las ecuaciones de equilibrio global del sistema han sido generadas, consiste en resolver dichas ecuaciones obteniendose con ello los valores de los parámetros nodales de los que dependen los valores de las variables de campo.

El proceso de solución de las ecuaciones de equilibrio depende en gran medida del problema en cuestión, pudiendo resultar éste un sistema de ecuaciones lineales, un problema de valores característicos, ó un sistema de ecuaciones diferenciales ordinarias; cualquiera que sea el caso, esta etapa resulta ser la que más recursos computacionales necesita, y para ello métodos numéricos especiales como los que se discutirán en el capítulo IV.

6. Cálculo de variables secundarias de campo

A partir de los resultados obtenidos en el punto anterior, otras variables a las que llamaremos variables secundarias de campo son calculadas. Algunos ejemplos de este tipo de variables son los esfuerzos que se calculan a partir de los desplazamientos en problemas estructurales, o bien, las velocidades en un fluído para cuya determinación se requiere del cálculo previo del potencial.

Cuando todos los resultados de interés se han calculado, si se desea estos pueden ser dibujados, utilizando para ello técnicas que permiten resumir grandes cantidades de información en forma gráfica. Con esta etapa, conocida como postprocesamiento, y con el análisis de los resultados obtenidos concluye el procedimiento general seguido en la aplicación del método de los elementos finitos.

Cabe recordar que el método de los elementos finitos es un procedimiento de análisis que puede ser aplicado a una gran variedad de problemas por lo que el algoritmo general presentado no debe ser considerado como un algoritmo riguroso, sino como un esbozo amplio del método. Aplicaciones de carácter particular serán presentadas en el capítulo V de este trabajo.

C. FORMULACION VARIACIONAL Y RESIDUAL DEL METODO

Como se expuso en el inciso anterior, el tercer paso en la solución de un problema con el método de los elementos finitos consiste en derivar las ecuaciones de los elementos finitos a partir de las ecuaciones que gobiernan el dominio de las variables de campo. Para obtener estas ecuaciones se puede utilizar cualquiera de las formulaciones existentes para el método, dentro de estas, las más comunes son las que utilizan métodos variacionales y métodos residuales o de residuos pesados. Ambos métodos son expuestos brevemente a continuación:

Método Variacional

El método variacional es utilizado cuando el fenómeno en estudio es modelado por una ecuación integral o funcional, consistiendo el problema en encontrar los parámetros que definen su valor extremo (mínimo o máximo)[2].

Un problema variacional modelado por el funcional I función de las funciones $y=y(x)$ y $y=dy/dx$ es descrito de la siguiente manera:

$$I(y) = \int_{x_1}^{x_2} f(x, y, y_x)\, dx$$

El funcional definido en toda la región es expresado como la suma de las integrales en el subdominio de cada elemento finito, cada una de estas integrales puede ser expresada en forma aproximada como[3]:

$$y = [N]\{a\}$$

donde el vector $\{a\}$ contiene los parámetros desconocidos asociados a los puntos nodales.

En forma semejante a la utilizada en el cálculo diferencial para obtener máximos o mínimos, en el cálculo variacional el funcional I(y) tendrá un valor estacionario

cuando su variación (δI) valga cero, esto es, cuando una pequeña variación de la función $y(x)$ no provoque cambios en el valor del funcional.

En el método de los elementos fintos en el que el funcional es expresado en términos de parámetros nodales, el principio arriba expuesto equivale a igualar a cero las derivadas del funcional respecto a cada uno de los parámetros nodales con lo que se forma un sistema de ecuaciones algebraicas que al ser resueltas se obtienen los valores de los parámetros nodales que minimizan al funcional.

Método de los residuos pesados

A diferencia del método variacional, en el método de los residuos pesados para la obtención de las ecuaciones de los elementos finitos se parte de una ecuación diferencial del tipo[4]:

$$L(y) = 0$$

con condiciones de frontera:

$$C(y) = 0$$

donde tanto L como C son operadores diferenciales.

En la misma forma que en el método variacional los métodos de residuos pesados suponen una aproximación a la variable y de la forma:

$$y = [N][a]$$

Al sustituir la aproximación en la ecuación diferencial obtenemos la siguiente expresión:

$$L(y) = R$$

donde R es el error o residuo que se genera debido a que la solución propuesta es tan sólo una aproximación de la solución exacta.

El objetivo de los métodos residuales consiste en minimizar el error, para ello, se buscan funciones de peso (W) que hagan mínimo el valor del residuo en toda la región, definiendo de esta manera la siguiente ecuación integral:

$$\int W \cdot R \, dV = 0$$

Esta ecuación puede ser expresada como el producto punto de las funciones de peso por el residuo R igualado a cero con lo que el error es distribuido en toda la región de acuerdo

a la función de peso. Esta expresión lleva finalmente a un sistema de ecuaciones de cuya solución se obtienen los valores de $\{a\}$ de manera semejante al método variacional.

Los distintos métodos residuales difieren entre sí en la forma en que las funciones de peso son determinadas. Dentro de estos métodos el más utilizado es el llamado método de Galerkin, en el cual las funciones de peso son iguales a las funciones de forma[5].

D. IMPLANTACION DEL METODO DE LOS ELEMENTOS FINITOS EN COMPUTADORA DIGITAL

Como ya se expuso la aplicación del método de los elementos finitos demanda una gran cantidad de cálculos, lo que plantea la necesidad de usar herramientas especiales para tales efectos. Por lo tanto una descripción del proceso de la implantación del método en computadoras digitales resulta necesaria en cualquier presentación del mismo, además, en el presente caso, esto permitirá visualizar la manera en que los distintos elementos que conforman un programa de elementos finitos se relacionan. El desarrollo de estos elementos se

discutirá a lo largo de los siguientes capítulos.

El gran uso que se hace de la memoria principal en un programa de elementos finitos hace recomendable manejar en un arreglo unidimensional todas las matrices usadas por el sistema, con lo que se logra un manejo más flexible y óptimo de la memoria disponible.

Si bien los programas de elemento finito pueden variar mucho, existe una estructura general que normalmente conservan. Esta estructura está determinada en su nivel más general por seis grandes módulos que son ilustrados en la figura II.2.

Si el problema varía linealmente con el tiempo, los procesos de solución del sistema de ecuaciones y cálculo de variables secundarias serán repetidos para cada incremento de tiempo realizándose con ello un proceso iterativo representado en la figura II.2 con la línea punteada interna; la otra línea punteada indica el bloque que habrá de repetirse si el problema es no lineal, ya que si este es el caso las matrices elementales deberán ser modificadas en cada iteración. Un problema dinámico no lineal incluirá ambas iteraciones.

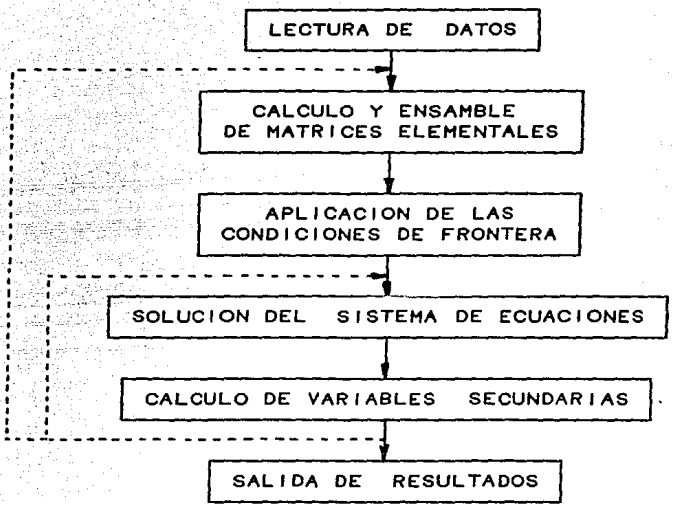

Fig. II.2

El primer bloque comprende la lectura de los datos del problema y la inicialización del sistema. El número de datos por leer es en general grande y puede ser dividido en varias etapas; una primera, en la que se leen los datos de control, como son el número de nodos, de elementos, el tipo de análisis,

etc. Posteriormente un segundo bloque de datos es leído, donde se incluye la geometría de la región, las propiedades de los elementos, las condiciones de frontera y las condiciones iniciales. Leídos los datos de control es posible determinar si el arreglo dimensionado será suficiente para el análisis y marcar con apuntadores la posición en el vector de las matrices que se usarán. Para hacer esto, el semiancho de banda o el vector de acceso deben ser calculados dependiendo del tipo de almacenamiento usado para la matriz del sistema (ver IV.2).

En la segunda parte del programa se realiza un proceso iterativo, en el cual, para cada elemento se calculan las matrices elementales, y quizá alguna otra que vaya a ser usada para el cálculo de variables secundarias, y éstas son ensambladas en las matrices globales. Es en esta parte del programa en el que más difieren las distintas aplicaciones del método, ya que el cálculo de las matrices elementales está en función de los elementos utilizados y las ecuaciones que gobiernan al problema en cuestión.

Las condiciones de frontera son impuestas al sistema de ecuaciones una vez que éste ha sido completado para proceder a resolver el sistema de ecuaciones, obteniéndose con ello los

valores de la variable de campo en los nodos. En problemas de vibración y análisis de estabilidad se tiene un problema de valores característicos por lo que en lugar de resolver el sistema de ecuaciones lineales, se determinan los valores y vectores característicos de las matrices globales.

Usando los resultados obtenidos en el paso anterior y las matrices generadas durante el cálculo de las matrices elementales y almacenadas en memoria auxiliar otros resultados de interés son determinados, como puden ser los valores de una variable secundaria de campo.

Finalmente la última etapa del programa la constituye la salida de los resultados obtenidos. Esta etapa puede ser enriquecida con rutinas de postprocesamiento que permitan concentrar la información de interés en forma gráfica.

Todos los procesos indicados requieren de varias operaciones cada una de las cuales puede ser vista como un proceso en sí. Esta división del sistema en pequeños módulos ofrece grandes ventajas al programador y constituye la idea principal de este trabajo como se verá en el siguiente capítulo.

1. Bathe,K.; Wilson,E. Numerical Methods in Finite Element Analysis. E.U.A. Prentice Hall Inc.1976. Pág 83.

2. Elsgoltz,L. Ecuaciones diferenciales y cálculo variacional. Moscú, Mir. 1983. Pág 287.

3. Ballesteros,P. Modelación Matemática de Sistemas. El Método del Elemento Finito en Ingeniería. D.E.C.F.I., UNAM, 1985. Pág 13.

4. Cervantes,R.; Porras,V. Introducción al método de los elementos finitos. México, 1983. F.I. UNAM. Pág 25.

5. ZienKiewycz,O. The Finite Element Method in Engineering Science. Great Britain, 1971. Mc. Graw Hill. Pág 40.

III. USO DE LIBRERIAS EN LA PROGRAMACION DEL METODO DE LOS ELEMENTOS FINITOS

A. CARACTERISTICAS DESEABLES EN UN PROGRAMA DE ELEMENTO FINITO

Durante muchos años los programas de elemento finito fueron desarrollados solamente en universidades y centros de investigación, donde las principales preocupaciones consistían en lograr implementaciones de rápida ejecución y utilizar modelos que ocuparan al mínimo la memoria.

Con el abaratamiento y creciente desarrollo de las computadoras, más usuarios se interesaron en los programas de elemento finito, sin embargo, los sistemas existentes estaban lejos de satisfacer sus necesidades, ya que presentaban ciertas carencias desde el punto de vista del usuario; entre éstas se encontraban la gran dependencia del programa con el equipo de cómputo, lo que hacia al programa poco transportable resultando muchas veces más conveniente el escribir un nuevo programa que el adaptarlo a otra máquina, otro defecto común en estos programas era la dificultad de uso ya que se daba

poca importancia a los aspectos ergonómicos durante el desarrollo de los mismos. Finalmente el aspecto que más directamente promovió un cambio en la filosofía de diseño de programas de elemento finito fué el del mantenimiento del sistema. En un medio en el que los progresos en los aspectos tanto teóricos como de implantación se daban día a día, surgían nuevos elementos que no podían ser fácilmente incorporados a los sistemas existentes.

A mediados de los años 60, con la llegada de la llamada crisis de la programación (debida básicamente a los crecientes costos en el mantenimiento de programas), se desarrolla la ingeniería de la programación, que pretende dictar filosofías más adecuadas para la elaboración de programas permitiendo reducir los costos de desarrollo y mantenimiento e incrementando al mismo tiempo la calidad de los mismos.

Las prácticas surgidas de la ingeniería de la programación, como son la programacion estructurada, la modularidad, la documentación, etc., frecuentemente usadas en la creación de programas de tipo administrativo empiezan desde hace poco tiempo a cobrar importancia en el desarrollo de programas con orientación científica.

De esta forma, la calidad de un programa puede evaluarse en términos de las siguientes características[12]:

CONFIABLE	Los resultados obtenidos por el programa deben ser correctos.
CLARO	El programa debe poder ser entendido por otra persona de manera que lo pueda utilizar o modificar.
UNIFORME	El programa y las estructuras de datos se definirán en un estilo uniforme.
DOCUMENTADO	Deben existir reportes que indiquen el uso del programa, entradas y salidas del mismo.
FACIL DE USO	Las entradas como las salidas en cada módulo deben ser claras y fáciles de identificar.
EFICIENTE	No debe haber desperdicio de recursos.
EXPANDIBLE	Nuevas funciones deben poder incorporarse al programa.
FLEXIBLE	El programa debe poder satisfacer las necesidades particulares de cada usuario.

TRANSPORTABLE El programa podrá ser ejecutado en otros equipos sufriendo cambios menores.

Existen en ocasiones conflictos entre algunas de las características antes citadas, en particular la eficiencia, en la que estaban basados los primeros programas de elementos finitos incide prácticamente en todas las otras, ya que el programa más eficiente estará sujeto a un equipo y sacrificará legibilidad y facilidad de uso, entre otras cosas.

B. MODULARIZACION Y EL CONCEPTO DE LIBRERIA

El objetivo de lograr programas de elemento finito que cumplan de la mejor manera posible con las características arriba mencionadas es más fácilmente alcanzado, si el programa se divide en varias partes o módulos, cada uno encargado de realizar una funcion específica.

Un módulo es un conjunto de instrucciones que desarrollan una función específica. Un módulo posee los siguientes atributos[3]:

NOMBRE	identificador por medio del cual es llamado
ENTRADAS	datos necesarios para efectuar la función
SALIDAS	datos que genera después de su ejecución
FUNCION	lo que hace con las entradas para producir las salidas.

Además, en cuanto a su estructura interna, un módulo puede llamar a otros módulos y manejar datos internos en variables locales.

En el diseño de programas modulares deben cuidarse dos aspectos: uno hace referencia al alto o bajo grado con el que se relacionan los elementos de un módulo, llamado cohesión, donde se busca que los módulos tengan una cohesión funcional, es decir, que todos los elementos del módulo contribuyan a realizar una y sólo una tarea. El otro aspecto, relacionado con la interdependencia o acoplamiento de un módulo respecto a otro, busca que este sea mínimo, lo cual permite modificar un módulo sin tener que modificar otros.

Con el objeto de disminuir el acoplamiento entre módulos debe evitarse el uso de variables globales o regiones comunes. La mayoría de los programas de elemento finito han sido

escritos en FORTRAN y más recientemente en BASIC; en los primeros generalmente utilizan regiones comunes (COMMON) como estructura de datos principal, en tanto que en los segundos todas las variables son globales. Con ello ambos modelos rompen con el principio arriba expuesto.

El tipo de acoplamiento más deseable se da cuando los datos se pasan por valor, a diferencia de los que se pasan por referencia donde se da la dirección del dato a la que el módulo se refiere permitiéndole afectar el valor de éste, o bien, cuando se pasan como estructuras de datos tales como arreglos, registros, etc. En los módulos aquí desarrollados se usan ambos tipos de paso de parámetros y ocasionalmente son pasadas banderas de control.

La integración de módulos como elementos independientes y compatibles que pueden combinarse en programas que satisfagan distintas necesidades específicas constituyen la idea principal de una librería.

Dos de las características que posee el método de los elementos finitos son su estructura intrínsecamente modular y el gran campo de aplicación del mismo, esta generalidad del

método crea al usuario la necesidad de contar con programas flexibles que puedan ser modificados con facilidad para estudiar nuevos problemas.

Lo arriba expuesto conduce a pensar que el uso de librerías como herramientas básicas en el desarrollo de programas de elemento finito es, no sólo útil, sino necesario. Además, el uso de librerías genera un incremento notable en la productividad, ya que permite, en aplicaciones particulares, que el usuario una los módulos de la librería que considere necesarios, con algunos que él mismo haya escrito; de otra manera tendría que programar todos los módulos, que para el caso de programas de elemento finito, resultaría una labor difícil y tomaría mucho tiempo.

El uso de librerías presenta además ventajas desde el punto de vista educativo, ya que el estudiante puede concentrar su atención en aspectos propios del método y olvidarse de los problemas de programación.

Finalmente el costoso problema del mantenimiento de programas se ve en buena medida superado, pues el uso de librerías permite que personas distintas al autor incorporen a

las mismas otros módulos, ya sean otros tipos de elementos, nuevos modelos o técnicas de solución.

C. SELECCION DEL AMBIENTE DE PROGRAMACION

De la adecuada selección de un ambiente de programación depende la calidad lograda en el producto final. El ambiente incluye las herramientas tanto de programación como de equipo usadas durante el desarrollo de programas, al igual que otros aspectos como es el estilo de programación.

Al seleccionar el ambiente de programación debe tenerse en mente que el producto debe operar en equipos distintos a aquel en el que fue originalmente desarrollado, de otra manera, el programa podría tener un ciclo de vida muy corto al ser reemplazado el equipo o bien, llegar a un sector limitado de usuarios. Seguramente el paso más importante en la implantación de un ambiente de programación corresponde a la selección del lenguaje.

Como ya ha sido mencionado, el lenguaje tradicionalmente usado en la programación del método de los elementos finitos

(y en general en programas de aplicación científica) ha sido FORTRAN, lo anterior obedece, entre otros, a los siguientes méritos: haber sido el primer lenguaje de alto nivel, ser modular, generar código objeto eficiente, ser un lenguaje muy estandarizado y al hecho de haber sufrido modificaciones generando nuevas versiones que le han permitido mantenerse relativamente al paso de los progresos que en materia de lenguajes de programación han habido.

Sin embargo, el uso de FORTRAN en la programación del método de los elementos finitos no es ya plenamente justificado al compararlo con lenguajes que sin tener que estar atados a implementaciones anteriores han podido incorporar avances recientes en programación, como son las estructuras de control propias de la programación estructurada, estructuras de datos complejas (incluyendo manejo dinámico de memoria), comentarios en columnas intermedias, recursividad, etc. Muchos atribuyeron erróneamente estas características a un lenguaje en particular (Pascal), un lenguaje creado con fines didácticos, pero cuyas implementaciones, como su propio autor admite[4], dejaban mucho que desear.

Lo anterior es seguramente una de las causas por las que tardaron en hacer eco entre aquellos encargados del desarrollo de programas de orientación científica e ingeniería muchos de los avances que en materia de programación se habían desarrollado, ya que éstos eran asociados a códigos largos y ejecuciones lentas.

El lenguaje a elegir debe proveer de elementos de programacion que permitan usar las técnicas de la programación estructurada: estructuras de control con una entrada y una salida, manejo de módulos o subrutinas (compilación y prueba por módulos) y estructuras de datos versátiles.

El lenguaje selecionado en este trabajo para la programación de las rutinas que darán cuerpo a las librerías es el lenguaje C, que además de cumplir con los requerimientos anteriores posee otras ventajas:

1. El lenguaje C es generalmente más rápido y genera código objeto más pequeño que otros lenguajes de programación, debido a que cuenta con operadores básicos que tienen correspondencia directa con los de lenguaje de máquina haciendo con ello menos complejo el proceso de compilación.

2. C es además un lenguaje de programación sumamente estándar, lo que le da una gran transportabilidad a los programas en él escritos.

3. Un programa en C esta construído en base a funciones, esta orientación funcional del lenguaje facilta la modularización y promueve el uso de librerias.

4. C ofrece también apuntadores y operadores aritméticos para su manejo, lográndose con su uso códigos mas compactos y eficientes. En particular el manejo de matrices usando apuntadores es altamente recomendado.

La característica que ha sido considerada como la mayor desventaja del lenguaje C es la falta de claridad de su código, lo cual dificulta la comprensión y seguimiento de las acciones de un programa escrito en ese lenguaje, sin embargo este inconveniente puede ser superado si el programa se encuentra adecuadamente documentado. Si bien C ha sido catalogado por muchos como un lenguaje de propósito específico para el desarrollo de sistemas operativos, los argumentos antes expuestos indican que su uso puede extenderse a otros campos de

aplicación pues en realidad se trata de un lenguaje de propósito general.

D. ESTANDARES USADOS EN EL DESARROLLO DE LA LIBRERIA

Los principales objetivos que debe satisfacer una librería son, dar facilidades para su mantenimiento y ser fácil de transportar. En lo que respecta a las subrutinas que contiene la librería, éstas deben usar técnicas modernas, ser eficientes y fáciles de usar.

Con el fin de poder alcanzar los objetivos arriba expuestos deben imponerse algunas reglas o estándares que deberán seguirse al escribir las rutinas.

En la programación de los módulos que formarán las librerías de elemento finito se siguieron los siguientes estándares:

1. Todos los módulos deberán estar escritos en un lenguaje de alto nivel (C) que permita el transporte de los mismos a otras máquinas con un mínimo de modificaciones.

2. Los módulos que sean dependientes del equipo deberán indicarlo especificando los cambios necesarios para adaptarlo a otro sistema de cómputo.

3. Los módulos deberán tener una sola entrada y una sola salida, ambas perfectamente definidas en la lista de argumentos. No habrá comunicación entre módulos con variables globales o regiones comunes.

4. Ningún módulo debera realizar funciones de entrada o salida a excepción de aquellos estrictamente encargados para ello.

5. El paso de parámetros se hará por valor, y sólo en el caso de vectores, el paso se hará por referencia (pasando la dirección del primer elemento del vector).

6. El manejo de matrices, muy frecuente en el método del elemento finito, se realizará en forma vectorial evitando con ello el ineficiente uso de variables con varios subíndices.

7. El paso de argumentos entre módulos guardará el siguiente orden: primero los argumentos de entrada, seguidos de los argumentos de entrada/salida y terminando por los argumentos de salida.

E. ORGANIZACION DE LA LIBRERIA

El número de rutinas necesarias para desarrollar un programa de elementos finitos modular es en general grande, si además consideramos que en una librería se incluyen rutinas que no serán utilizadas en un programa, tendremos un número de rutinas tal que será necesario organizarlas para que su uso se facilite.

Incluir a todas las rutinas en una sola librería resultaría poco práctico, es por ello, que han sido divididas en cinco librerías agrupadas de acuerdo al tipo de función que realizan.

A continuación se presentan las rutinas incluídas en cada librería así como la función de cada una:

MATRIZ — Librería que incluye las operaciones matriciales y vectoriales básicas usadas en la programación el método de los elementos finitos.

limpia_ent	iguala a cero un vector enteros
limpia_real	iguala a cero un vector real
banda	calcula semi-ancho de banda de matriz
perfil	calcula vector de acceso de una matriz
altura	calcula alturas a partir de la diagonal principal
mul_mat	multiplica 2 matrices y sus transpuestas
detinv1	calcula det. e inversa de matriz de 1x1
detinv2	calcula det. e inversa de matriz de 2x2
detinv3	calcula det. e inversa de matriz de 3x3
ensambla_todo	suma matriz elemental a matriz global
ensambla_banda	suma mat. elemental a mat. global en banda
ensambla_perfil	suma matriz elemental a matriz global en perfil
modifica_todo	impone condiciones de frontera en sistema de ecs global con almacenamiento completo
modifica_banda	impone condiciones de frontera en sistema de ecs global con almacenamiento en banda
modifica_perfil	impone condicione de frontera en sistema de ecs global almacenadas en perfil

FUNFORMA Incluye rutinas que calculan funciones de forma y sus derivadas para elementos de las familias lagrangiana y triangular, así como de integración gaussiana para elementos isoparamétricos.

ff12 funciones de forma elemento de 1-D de 2 nodos

ff13 funciones de forma elemento de 1-D de 3 nodos

ff24 funciones de forma elemento de 2-D de 4 nodos

fd24 derivadas de func. de forma elem. 2-D de 4 nodos

ff28 funciones de forma elemento de 2-D de 8 nodos

fd28 derivadas de func. de forma elem. 2-D de 8 nodos

ff38 funciones de forma elemento de 3-D de 8 nodos

fd38 derivadas de func. de forma elem. 3-D de 8 nodos

ff23 funciones de forma elemento traingular de 3 nodos

fd23 derivadas de func. de forma de triang. de 3 nodos

gauss calcula ptos. de integración y factores de peso para integración gaussiana en una dimensión

gauss2 calcula ptos. de integración y factores de peso para integración gaussiana en dos dimensiones

SOLECS Esta librería incluye las rutinas necesarias para resolver las ecuaciones de equilibrio global.

sol_gauss resuelve sistema de ecs. por método de gauss

fact_banda triangulariza matriz almacenada en banda

obtenx_banda obtiene vector solucion de matriz en banda

fact_perfil triangulariza matriz almacenada en perfil

obtenx_perfil obtiene vector solucion de matriz en perfil

APLICACION Las rutinas particulares de la aplicación del mētodo a los problemas presentados se incluyen en esta librería.

barra calcula matriz de elemento barra en 2-D

tri_esf calcula matriz de elemento traingular para edo. plano de esfuerzos

suma_desp suma desplazamientos a coordenadas de los nodos

laplace28 calcula matriz de elemento isoparamētrico de 8 nodos para la ecuación de LaPlace

PREPOST Las rutinas presentadas en el capítulo VI que son usadas para pre y postprocesamiento forman esta librería.

genera_malla genera mallas de elementos rectangulares en forma automática

dibuja_malla dibuja mallas planas de elementos finitos

numera_nodos escribe los numeros de cada nodo

numera_elementos escribe los numeros de cada elemento

contorno grafica curvas de contornos

maxmin . calcula valores máximos y mínimos de cada variable de campo

dibuja_linea grafica una línea

dibuja_vectores grafica vectores de acuerdo a valores y orientaciones definidas

---

1. Ayala, G.; Eudave,J.; Gómez, R. Implantación del Método de los elementos finitos en microcomputadoras. Un estado del arte. 1986. Memorias del V Congreso Nacional de Ing. Estructural. Veracruz. Pág. E5-04.

2. Rice, J. Numerical Methods, Software, and Analysis. 1983. E.U.A. Mc. Graw Hill. pág 426-429.

3. Martínez, A. Diseño Estructurado. México. Pág. 4.

4. Wirth, N. History and Goals of Modula-2. BYTE, agosto, 1984. E.U.A. Pág 145.

IV. ASPECTOS NUMERICOS DEL METODO DE LOS ELEMENTOS FINITOS

A. INTRODUCCION

El método de los elementos finitos requiere para su aplicación del uso de una gran cantidad de métodos numéricos como son: integración numérica, interpolación, solución de sistemas de ecuaciones, etc. y dado que se trata de un método aproximado que utiliza gran cantidad de recursos computacionales, debe buscarse que las técnicas numéricas usadas en su aplicación sean precisas y rápidas.

Después de los primeros años, en los que la formulación teórica del método ocupó la atención de los especialistas, le siguió una etapa en la que el desarrollo de métodos numéricos eficientes se convirtió en la principal preocupación con lo que, en corto tiempo, se desarrollaron numerosos algoritmos en los que se consideraba de suma importancia, desde entonces, el eficiente uso que de los recursos de la máquina éstos hicieran.

El uso de librerías en la programación del método de los elementos finitos permite incorporar nuevos métodos numéricos con suma facilidad y compararlos con los utilizados. Por otra parte, para que el uso de librerías quede plenamente justificado, los algoritmos contenidos en éstas deberán minimizar tiempos de ejecución y garantizar precisión en los cálculos.

Los algoritmos que se presentan en este capítulo tienen características especiales respecto a los algoritmos convencionalmente usados para resolver problemas similares. Esto se debe a que la aplicación del método de los elementos finitos necesita de numerosos cálculos y grandes espacios de memoria, y cualquier intento que se haga por optimizar el uso de estos recursos debe ser tomado en cuenta. El uso de nuevas estructuras de datos puede ser considerado como la principal aportación que en este sentido hace el presente trabajo.

B. MODELOS DE ALMACENAMIENTO DE MATRICES

En lenguajes de alto nivel los vectores y matrices se representan normalmente en estructuras de datos llamados

arreglos, de manera que el manejo de los elementos que lo forman se hace através de índices (escalar cuyo valor apunta a una dirección de memoria relativa a otra), así, la expresión **a[5]** hace referencia al dato que se encuentra 5 espacios después de la dirección de referencia, en donde se encuentra el valor de **a[0]**.

Como los elementos de los arreglos son almacenados en forma secuencial en la memoria principal de la computadora, el manejo de matrices es más complicado que el de vectores. Si en la representación de matrices se utilizan arreglos bidimensionales se necesitaran de dos índices para hacer referencia a un elemento, uno que indique el renglón y otra la columna en que éste se localiza. De acuerdo a esto, el almacenamiento en memoria de un elemento será determinado por una expresión que permita calcular la dirección del elemento en función de los índices, a este tipo de expresiones se les conoce con el nombre de funciones de mapeo. Así, en una matriz **[A]** de **m** renglones por **n** columnas la función de mapeo para localizar el elemento **a[i,j]** sera:

$$dir\ a[i,j] = dir.\ base + m*i + j$$

donde la dirección base es aquélla que contiene el primer elemento del arreglo en el caso de que el índice comience

desde cero como ocurre en el lenguaje C. En el caso de que el índice comience en uno, la dirección base hace referencia a una localidad de memoria anterior al primer elemento y la función de mapeo se transforma en:

$$dir\ a[i,j] = dir.\ base + m*(i-1) + j$$

Calcular las direcciones de memoria en programas que efectúan operaciones matriciales puede representar un porcentaje alto del número total de cálculos que se realizan durante la ejecución del programa, por ello, resulta conveniente manejar las funciones de mapeo desde el programa usando estructuras de datos de más bajo nivel optimizando con ésto el cálculo de las direcciones de los elementos de las matrices y obteniendo códigos objeto más pequeños.

Al representar las matrices con arreglos unidimensionales en lugar de arreglos bidimensionales se cuenta con un modelo más parecido al que se tiene en memoria principal. El uso de un arreglo unidimensional presenta varias ventajas, en la suma de matrices por ejemplo, un ciclo iterativo es eliminado y se evita la operación de multiplicación en la función de mapeo como se puede observar en el pseudocódigo:

arreglo bidimensional

```
for i=1,m
    for j=1,n
        c[i,j]=a[i,j]+b[i,j]
    endfor
endfor
```

arreglo unidimensional

```
for i=1,m*n
    c[i]=a[i]+b[i]
endfor
```

En la multiplicación de matrices que generalmente se realiza con 3 ciclos iterativos anidados, si los arreglos bidimensionales son utilizados, el cálculo de las direcciones de memoria de las dos matrices que se multiplican se realiza en el ciclo más interno; usando arreglos unidimensionales, parte de la dirección puede ser calculada en ciclos externos evitando repetir inecesariamente esta operación.

Si las matrices son almacenadas en arreglos unidimensionales la operación más costosa para determinar la dirección de un elemento lo constituye la multiplicación entre el número de renglones y el índice que hace referencia a la columna en que se encuentra el elemento, ya que a diferencia de la suma, la multiplicación no es una operación directa del procesador. Como en la mayoría de las operaciones matriciales constantemente se requiere de operaciones entre columnas o renglones es recomendable recorrer los elementos de la matriz

con incrementos constantes usando como dirección base la del elemento anterior, eliminando con esto, la multiplicación que aparece en la función de mapeo.

Lo anterior puede ser realizado si se cuenta con estructuras de datos que permitan almacenar la dirección de otra estructura de datos (apuntadores) y si se puden realizar operaciones aritméticas elementales con estas direcciones, como sucede en el lenguaje C.

En C, inclusive, la semejanza en el uso de apuntadores y arreglos es notable, al grado de que las referencias a arreglos son convertidas en apuntadores durante la compilación, esto, aunado a las ventajas arriba expuestas para el uso de apuntadores en operaciones matriciales permite generar códigos objeto más compactos y eficientes.

Por otra parte, las matrices de las ecuaciones de equilibrio que aparecen en el método de los elementos finitos poseen, en general, dos características: la primera de ellas es el ser simétricas; y la segunda, el ser altamente porosas, es decir, la mayoría de los elementos que la forman valen cero. Estas dos circunstancias han sido aprovechadas en los

métodos de solución de las ecuaciones de equilibrio y es por ello que se han originado modelos especiales de almacenamiento para las matrices globales, en los que no todos los elementos de la matriz son almacenados. Dentro de estos modelos los más conocidos son el almacenamiento en banda y el almacenamiento en perfil, los que a continuación se explican.

Almacenamiento en banda

De acuerdo al procedimiento descrito en el capítulo dos para el ensamble de las ecuaciones de equilibrio, las matrices elementales son sumadas a las matrices globales en las posiciones que corresponden a los grados de libertad asociados al elemento finito en cuestión. De esta forma, en la medida en que los grados de libertad de los nodos en que inciden todos los elementos finitos tengan una numeración cercana, la matriz global estará concentrada en la diagonal principal. Esta característica puede ser aprovechada para economizar recursos de la computadora y por ello, existen algoritmos de numeración de nodos[1] que permiten obtener matrices lo más concentradas posibles alrededor de la diagonal principal de acuerdo a la topología de la región en estudio.

En la figura IV.1 se muestra el patrón general de una matriz global, un parámetro importante en este tipo de matrices lo constituye el ancho de banda que indica el número de diagonales que alrededor de la diagonal principal contienen todos los elementos que no son cero de la matriz. En caso de que la matriz sea simétrica, interesará conocer el semi-ancho de banda que incluye de la diagonal principal hasta la última diagonal que contiene elementos no nulos. Estos dos parámetros también son señalados en la figura IV.1.

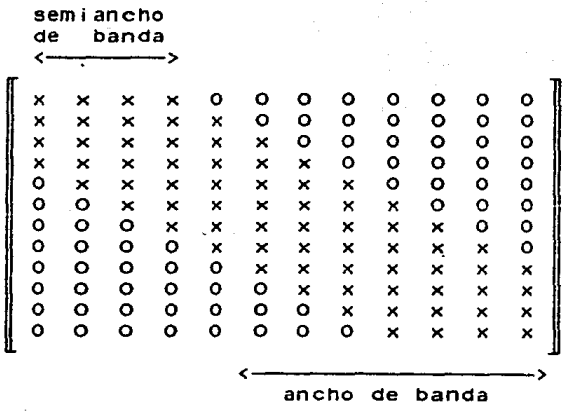

Fig. IV.1

Algunos procedimientos de solución de sistemas de ecuaciones no necesitan de los elementos no nulos por lo que su almacenamiento resulta inecesario. En el modelo de almacenamiento en banda para matrices simétricas sólo los coeficientes que se encuentren dentro del semi-ancho de banda son tomados en cuenta. El almacenamiento se realizará en un arreglo unidimensional por lo que existen dos posibles formas de llevarlo a cabo: una en la que primero se almacenan los elementos de la diagonal principal, después los de la diagonal contigua, y así hasta almacenar los elementos de la última diagonal que contiene elementos no nulos, y otra en la que primero se almacenan los elementos del primer renglón hasta el semi-ancho de banda (en caso de guardarse la triangular superior), después el segundo renglón, etc. hasta llegar al último. De las formas anteriormente presentadas, el segundo caso ofrece ventajas sobre el primero, ya que en los métodos de solución, como la mayoría de las operaciones matriciales, las operaciones entre columnas o renglones son frecuentes, por lo que resulta conveniente tener a los elementos de una misma columna o renglón almacenados en forma contigua. Cuando el tamaño de las matrices globales hace necesario el uso de memoria secundaria, el almacenamiento por renglones o columnas es más importante ya que de esta manera sólo los vectores que

se operan en un momento dado se guardan en memoria principal.

Antes de comenzar a formar las matrices globales se debe calcular el semi-ancho de banda para determinar el espacio que se requiere en el almacenamiento de la matriz. El semi-ancho de banda es función del máximo valor absoluto de la mayor diferencia entre los números de los nodos conectados a cada elemento, este valor incrementado en uno y multiplicado por el número de grados de libertad que se asocian a cada nodo es igual al semi-ancho de banda.

El algoritmo usado para determinar el semi-ancho de banda se muestra a continuación, la subrutina correspondiente se encuentra en el apéndice A con el nombre de *banda*.

```
maxdif = 1
for l=1,num_elem
    m = n-1
    for i=1,m
        for j=1,n
            dif = abs(nodos[i]-nodos[j])
            if (dif > maxdif)
                maxdif = dif
            endif
        endfor
    endfor
endfor
ancho_banda = ng*(maxdif+1)
```

La variable **n** guarda el número de nodos que tiene cada elemento, **ng** el número de grados de libertad de cada nodo y el vector **nodos[]** almacena los números de los nodos que definen la geometría de cada elemento.

La función de mapeo para un elemento **a[i,j]** (con i ≥ j) de una matriz alacenada en banda estará dada por la expresión:

$$dir\ a[i,j] = dir\ base + i * abanda + j$$

donde **abanda** es el semiancho de banda. La condición de que el índice i sea mayor o igual a j indica que se trabaja únicamente con la triangular superior de la matriz, aprovechando la simetría de la misma.

Almacenamiento en perfil

En el método del perfil, se calcula para cada columna la altura a partir de la diagonal principal a la que se encuentra el último elemento no nulo y sólo estos elementos son almacenados. Como en cada columna el número de elementos por almacenar puede ser distinto, el cálculo de las direcciones de los elementos de la matriz se lleva a cabo usando un vector de acceso de longitud igual al número de incógnitas del sistema.

El vector de acceso almacena en cada uno de sus elementos la dirección en que se localizan los elementos de la diagonal principal y debe ser definido antes de empezar a formar la matriz global.

La función de mapeo para encontrar la dirección de un elemento **a[i,j]** (con i ≥ j) almacenado en perfil es:

$$dir\ a[i,j] = va[i] + j$$

siendo **va[i]** el i-ésimo elemento del vector de acceso, el cual contiene la dirección del i-ésimo elemento de la diagonal principal.

En la formación del vector de acceso se obtienen primero las diferencias entre los números de los nodos contenidos en un elemento dado y el menor de éstos, al realizar este cálculo para todos los elementos se determina la máxima altura que tendrán las columnas asociadas a todos los grados de libertad de cada nodo, el vector de acceso se obtendrá al sumar estas alturas.

La subrutina *perfil* es usada para calcular el vector de acceso que ayudará a formar la matriz global y resolver las

ecuaciones de equilibrio. La subrutina *perfil* llama a la subrutina *limpia_ent* que inicializa en ceros al vector de acceso y a la rutina *altura* que calcula las alturas de las columnas de la matriz global para cada elemento finito. El código de las tres rutinas puede encontrarse en el apéndice A de este trabajo.

C. OPERACIONES MATRICIALES BASICAS

En la programación del método de los elementos finitos las operaciones matriciales son, como se vió en el capítulo dos, frecuentemente requeridas, ya que según el procedimiento descrito, para cada elemento del medio contínuo se genera una o varias matrices que son incorporadas posteriormente a una matriz global del sistema de cuya solución depende la determinación de las variables de interés.

En este inciso se tratarán las herramientas matriciales básicas, tanto para la creación de las matrices elementales, como para su ensamble en las matrices globales y la aplicación a éstas de las condiciones de frontera, dejando la solución de las ecuaciones de equilibrio para los siguientes incisos, por

resultar éstos, los algoritmos numéricos más importantes en la programación del método de los elementos finitos.

Multiplicación de Matrices

El algoritmo para multiplicar matrices es sencillo y puede ser reducido a la siguiente expresión:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

donde a_{ik} es el elemento de la matriz que premultiplica colocado en el renglón i columna k, b_{kj} el elemento de la matriz que postmultiplica colocado en el renglón k columna j, y c_{ij}, el elemento colocado en el renglón i columna j de la matriz resultante.

El cálculo de la transpuesta de la matriz [A] multiplicada por una matriz [B] es muy utilizado en la formación de las matrices elementales; para realizarlo, el proceso lógico consiste en transponer primero la matriz que lo requiere y hacer posteriormente la multiplicación de acuerdo a la fórmula arriba presentada. Sin embargo, este proceder tiene

dos inconvenientes: primero, se requiere de espacio en memoria para almacenar la matriz transpuesta, y segundo, es necesario hacer el intercambio de elementos en la matriz para generar la transpuesta.

Una solución alterna consiste en cambiar el algoritmo de multiplicación de matrices quedando de la siguiente manera:

$$c_{ij} = \sum_{k=1}^{n} a_{ki} \cdot b_{kj}$$

con lo que se ahorra la generación de la matriz transpuesta y el espacio para su almacenamiento.

En la librería se decidió incluir en una sola subrutina la multiplicación de dos matrices, la multiplicación de la transpuesta de la primera por la segunda, la primera por la transpuesta de la segunda y el producto de las transpuestas de las dos matrices. Para distinguir la operación que se desea realizar entre los argumentos de la función, se incluye una bandera cuyo valor determina el tipo de multiplicación matricial seleccionado.

Utilizando apuntadores con los que se eliminan las multiplicaciones para calcular las direcciones de los

coeficientes de las matrices, se presenta el programa en C para multiplicar dos matrices:

```
mulmat(a,b,n1,n2,n3,c)

float   *a,     /* apuntador a matriz que premultiplica */
        *b,     /* apuntador a matriz que postmultiplica */
        *c;     /* apuntador a matriz resultante */

int     n1,     /* num de rengl. de matriz [A] */
        n2,     /* num de col de [A] y de rengl de [B] */
        n3;     /* num de columnas de matriz [B] */
{
    float  suma,*ja,*jb;
    int    i,j,K;

    ja = a;
    jb = b;
    for (i=0;i<n1;i++) {
        jb += n2;
        for (j=0;j<n3;j++) {
            jb -= n2;
            ja = a + j;
            suma = 0.0;
            for (K=0;K<n2;K++,ja+=n1,jb++)
                suma += *ja*(*jb);
            *(c++) = suma;
        }
    }
}
```

Las otras operaciones de multiplicación pueden verse codificadas en lenguaje C en el apéndice A bajo el nombre de *m u l m a t .*

Inversa y determinante de una matriz

En el método del elemento finito es deseable contar con elementos que puedan tomar formas irregulares o que tengan lados curvos para poder representar de mejor manera la geometría de la región en estudio. La forma más sencilla de derivar estos elementos resulta si se les concibe como de geometría regular en las coordenadas locales del elemento y luego se mapean a coordenadas cartesianas, donde el elemento podrá tomar una forma irregular.

Cuando los puntos que son usados para definir la geometría del elemento definen también la variable de campo y por lo tanto tienen las mismas funciones de forma, se dice que los elementos finitos son isoparamétricos[2].

En la determinación de las matrices de elementos isoparamétricos comúnmente se requiere calcular derivadas de la variable de interés en coordenadas globales; la transformación de estos valores a partir de derivadas en coordenadas locales necesita la obtención de la inversa de la matriz jacobiana J que a su vez puede obtenerse al multiplicar la matriz de derivadas de las funciones de forma

en el punto de interés por las coordenadas de los nodos del elemento.

El determinante de la matriz jacobiana de transformación depende de la dimensión del elemento en cuestión, así si el elemento es bidimensional la matriz será de 2x2, por lo que no es necesario contar con rutinas generales para el cálculo de determinantes e inversas de matrices, lo que además de dificultar la programación, originaría un desperdicio de recursos computacionales.

En vista de lo anterior y dado que la dimensión del elemento no puede exceder de tres dimensiones, se han programado tres subrutinas, una llamada *detinv1* obtiene inversa y determinante de la matriz jacobiana de un elemento unidimensional, *detinv2* es usada en elementos bidimensionales y para problemas en tres dimensiones se usará la subrutina *detinv3*. En los tres casos primero se calcula el determiante y aprovechando éste se calcula la inversa usando para ello el método de cofactores.

Las subrutinas mencionadas pueden ser consultadas en el apéndice A.

Ensamble de la matriz global del sistema

El proceso de ensamble de las matrices características de cada elemento en matrices globales puede ser descrito como una suma ordenada de matrices en la que a cada uno de los coeficientes de las matrices elementales le son asociados dos grados de libertad que indican las coordenadas de la matriz global en la que el coeficiente será sumado.

El algoritmo para ensamblar cada elemento consistirá entonces, en encontrar a partir de los índices de los coeficientes del elemento la dirección de la matriz global donde cada elemento será incorporado, esto, por supuesto, dependerá del tipo de almacenamiento que se haya elegido para la matriz global.

El primer paso antes de sumar un elemento a la matriz global consiste en calcular los números de los grados de libertad asociados a él, esto se obtiene mediante la expresión:

$$gl = ng*(i-1) + j$$

donde **ng** es el número de grados de libertad que se

tienen en cada nodo, i es el número de nodo asociado al coeficiente de la matriz elemental en cuestión y j es el grado de libertad del nodo i que le corresponde. La subrutina *obtengl* se encarga de encontrar los números de los grados de libertad asociados a cada elemento. Este algoritmo no depende de la forma en que hayan sido almacenadas las matrices generales.

Una vez que los números de los grados de libertad han sido determinados, se ensambla la matriz elemental en forma directa, calculándose la dirección en la matriz global con la función de mapeo correspondiente a cada modelo de almacenamiento y usando como índices los números de grados de libertad que corresponden a cada coeficiente de la matriz elemental.

Si el número de grados de libertad correspondiente al renglón del elemento **me[i,j]** se le llama **mgi** y al que corresponde a la columna **mgj**, cuando la matriz global **[mg]** sea almacenada en un arreglo unidimensional la dirección en que será acumulado el coeficiente **me[i,j]** estará dada por la relación:

```
dir mg[ngi,ngj] = dir mg[0] + ngls*(ngi-1) + ngj
```

siendo **ngls** el número de grados de libertad que tiene el sistema que es igual al producto entre el número total de nodos y el número de grados libertad que se tienen por cada nodo.

En el caso en el que el modelo de almacenamiento seleccionado sea en banda, la función de mapeo será:

dir mg[ngi,ngj] = dir mg[0] + abanda*(ngi-1) + ngj

donde abanda contiene el semiancho de banda calculado por la subrutina *banda*. Si sólo la parte triangular superior va a ser almacenada, aprovechando las condiciones de simetría de la matriz, debe asegurarse de ensamblar solamente aquellos coeficientes de la matriz elemental en los que **ngi** $\geq$ **ngj**.

Finalmente, cuando se utilize el método del perfil la función de mapeo que se utilizará será:

dir mg[ngi,ngj] = diag[ngi] + ngj

donde **diag[]** es el vector de acceso obtenido por la subrutina *perfil*. Nuevamente si las matrices del sistema son simétricas la expresión anterior estará restringida a los casos en que **ngi** $\geq$ **ngj**.

Las rutinas de ensamble para cada modelo son:

ensambla_todo en el caso del arreglo unidimensional, *ensambla_banda* cuando se utiliza el método de almacenamiento en banda y *ensambla_perfil* si la matriz va a ser almacenada de acuerdo al método del perfil. Las tres rutinas comienzan llamando a la rutina *obtengl* descrita anteriormente (ver apéndice A).

Aplicación de condiciones de frontera

La aplicación de las condiciones de frontera a las ecuaciones del sistema puede llevarse a cabo de dos formas: la primera de ellas consiste en eliminar del sistema de ecuaciones los grados de libertad restringidos, en tanto que en la segunda, el sistema de ecuaciones es modificado una vez que ha sido ensamblado para reflejar las condiciones de frontera. El primer método posee la ventaja de que el sistema a resolver es más pequeño y evita calcular los parámetros de los nodos restringidos, a diferencia del segundo método que permite manejar más facilmente distintos tipos de condiciones de frontera.

La generalidad que se busca en la elaboración de herramientas para realizar programas de elemento finito

conduce a elegir el segundo método para ser incorporado a la librería.

Para explicar el método de modificación de matrices globales para aplicar las condiciones de frontera, considérese un nodo i cuyo grado de libertad j tiene el valor prescrito T.

A partir de los indices i y j se calcula el número del grado de libertad del sistema cuyo valor ha sido restringido, esto se logra con la expresión presentada en el inciso anterior:

$$gl = ng(i-1) + j$$

La condición de frontera será satisfecha si sustituimos el elemento correspondiente al grado de libertad gl en el vector de términos independientes por el valor preescrito T, haciendo unitario el coeficiente a[gl,gl] de la matriz global y a los otros elementos del gl-ésimo renglón y columna de la matriz los igualamos a cero. El hacer estos ajustes modifica las relaciones entre las ecuaciones (cuando T no es cero), mismas que pueden ser reestablecidas al restar a los elementos del vector de términos independientes

el correspondiente valor en la columna o renglón gl (si la matriz es simétrica) de la matriz global multiplicado por el valor prescrito T. La modificación que sufre el sistema de ecuaciones debida a valores preescritos en los nodos es ilustrada en la figura IV.2.

$$
\begin{array}{c}
\begin{array}{c} \\ 1 \\ gl \\ k \\ n \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & & gl & & k & & n \end{array} \\
\left[\begin{array}{cccccc}
a_{11} & \cdots & 0 & \cdots & a_{1k} & \cdots & a_{1n} \\
0 & \cdots & 1 & \cdots & 0 & \cdots & 0 \\
a_{k1} & \cdots & 0 & \cdots & a_{kk} & \cdots & a_{kn} \\
a_{n1} & \cdots & 0 & \cdots & a_{nk} & \cdots & a_{nn}
\end{array}\right]
\end{array}
\left\{\begin{array}{c} x_1 \\ x_{gl} \\ x_k \\ x_n \end{array}\right\}
=
\left\{\begin{array}{c} b_1 - T \cdot a_{1gl} \\ T \\ b_k - T \cdot a_{kgl} \\ b_n - T \cdot a_{ngl} \end{array}\right\}
$$

Fig. IV.2

El algoritmo anteriormente descrito, si bien general, debe desarrollarse para cada modelo de almacenamiento considerado. La subrutina *modifica_todo* del apéndice A aplica las condiciones de frontera (valores preescritos en los nodos) en matrices almacenadas en arreglos unidimensionales, en tanto que las subrutinas *modifica_banda* y *modifica_perfil* lo hacen para almacenamientos en banda y en perfil respectivamente.

D. FUNCIONES DE INTERPOLACION

Uno de los aspectos más importantes del método de los elementos finitos lo constituye la selección de las funciones de interpolación con las que se aproxima la variable de campo. Estas funciones de interpolación o funciones de forma estarán en términos de parámetros que en los nodos definen los grados de libertdad del sistema.

Por conveniencia y por facilidad de manejo se acostumbra utilizar tanto en las funciones de interpolación como en la integración numérica coordenadas locales normalizadas que varían en el rango de -1 a +1.

Si bien pueden seleccionarse como funciones de interpolación splines, funciones trigonométricas u otras funciones trascendentes, lo más común es utilizar para ello funciones polinomiales ya que su manejo, como el de sus derivadas es muy sencillo y da en la mayoría de los casos buenos resultados.

De acuerdo con la continuidad entre elementos que deben

satisfacer las funciones de interpolación, se tienen las funciones C^0 que son contínuas en la frontera de los elementos, las funciones C^1 que además de garantizar la continuidad de las funciones también son contínuas sus primeras derivadas, y en general las C^n donde la continuidad se extiende hasta las derivadas de orden n.

Una forma de obtener las funciones de interpolación usando polinomios para el caso unidimensional, es expresando la variable de campo (y) como una expansión polinomial:

$$y = a_0 + a_i x^i \qquad i=1,2,\ldots,n$$

donde el valor i indicara el grado de la aproximación, si i=1 la aproximación será lineal, si i=2 cuadrática, si i=3 cúbica, etc.

La expansión polinomial anterior es evaluada en cada nodo obteniéndose con ello un sistema de ecuaciones cuyas incógnitas son los parámetros a_i.

Los valores a_i son sustituidos en el polinomio original y los términos asociados, llegándose a una ecuación del tipo:

$$y = N_i y_i$$

siendo N_i las funciones de interpolación buscadas.

El procedimiento descrito es laborioso, siendo su principal inconveniente la solución del sistema de ecuaciones que no puede llevarse a cabo numéricamente, sino en forma simbólica en términos de las variables que intervienen.

Un método más conveniente y que lleva a los mismos resultados, consiste en utilizar las funciones de interpolación de Lagrange como funciones de forma.

El polinomio de interpolación de Lagrange de orden n en una dimensión esta definido por la expresión:

$$l_i = \prod_{j=1}^{n} \frac{x - x_j}{x_i - x_j}$$

siendo r la variable adimensional que va de -1 a +1.

Usando los polinomios de interpolación de Lagrange para elementos unidimensionales lineales (fig. IV.3) las funciones de forma de acuerdo a la expresión anterior serán:

$$N_1 = l_1^2 = \frac{r - r_2}{r_1 - r_2} = 1/2 \cdot (1-r)$$

$$N_2 = l_2^2 = \frac{r - r_1}{r_2 - r_1} = 1/2 \cdot (1+r)$$

los mismos resultados serían obtenidos usando el primer procedimiento descrito.

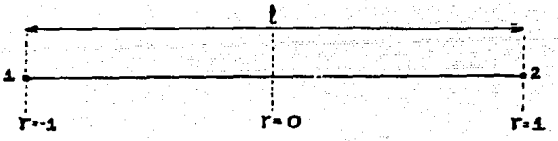

Fig. IV.3

Puede observarse tanto en la expresión general como en las anteriores que las funciones de interpolación de Lagrange tienen la propiedad:

$$l_i(r_j) = \delta_{ij}$$

donde δ_{ij} es la función delta de Kronecker y por lo tanto es igual a uno cuando los subíndices son iguales y es nula cuando estos son distintos. Esta propiedad de las funciones de interpolación es importante ya que garantiza que

sean linealmente independientes y formen una base en el espacio de dimensión **n** (siendo **n** el numéro de funciones de forma).

Las funciones de forma para elementos unidimensionales cuadráticos (con un nodo intermedio) son derivados en forma semejante[3]. La subrutina *ff12* y *ff13* calculan las funciones de forma normalizadas de elementos unidimensionales con 2 y 3 nodos respectivamente.

Los elementos finitos bidimensionales comúnmente usados son triángulos y cuadriláteros (con lados lineales o curvos). En el caso de estos últimos las funciones de forma pueden calcularse con los productos de polinomios unidimensionales obtenidos por Lagrange en las dos coordenadas correspondientes a cada nodo. Estas mismas ideas puden ser extendidas a elementos finitos en tres dimensiones donde las funciones de forma de un hexaedro se obtienen con el producto de las funciones de forma unidimensionales valuadas en cada una de las tres coordenadas de cada nodo del elemento,

Las subrutinas que calculan las funciones de forma de un elemento rectangular de 4 nodos y sus derivadas son

respectivamente *ff24* y *fd24*, para el elemento finito rectangular de 8 nodos las subrutinas correspondientes son *ff28* y *fd28*, en tanto que para el hexaëdro de 8 nodos se usarán las rutinas *fd38* y *fd38*.

Para obtener las funciones de forma de la familia de elementos triangulares se utilizan coordenadas triangulares o coordenadas de área (de volumën en 3 dimensiones), estas coordenadas para el elemento triangular mostrado en la figura IV.4 están definidas por las siguientes expresiones:

$$l_1 = (a_1+b_1x+c_1y)/2A$$

$$l_2 = (a_2+b_2x+c_2y)/2A$$

$$l_3 = (a_3+b_3x+c_3y)/2A$$

donde A es el área del triángulo y las variables a, b y c son función de las coordenadas de los vértices del triángulo.

Las coordenadas de área así calculadas definen las funciones de forma para un elemento triángular de 3 nodos, y pueden calcularse usando la subrutina *ff23* y para sus derivadas la subrutina *fd23*.

Las funciones de forma de un elemento tetraedro lineal

pueden ser determinadas en forma semejante usando para ello coordenadas de volumen.

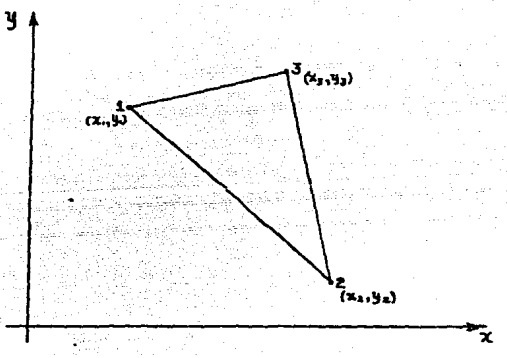

Fig. IV.4

Si en lugar de las funciones de interpolación consideradas se usan otras, nuevas familias de elementos pueden ser creadas además de la familia lagrangiana y la triangular. La familia serendipity, por ejemplo, es utilizada cuando los nodos de los elementos son colocados en la frontera del elemento. Esta última familia debe su nombre a que las funciones fueron obtenidas por inspección.

En caso de que se quiera garantizar la continuidad de las derivadas de las funciones en la frontera de los elementos, se utilizan con frecuencia los polinomios de interpolación de Hermite que pueden ser considerados como una extensión de los polinomios de Lagrange.

Si bien sólo dos familias de elementos han sido consideradas en este trabajo, el uso de librerías permite incorporar fácilmente otras familias, lográndose con ello gran flexibilidad en la programación del método de los elemetos finitos.

E. INTEGRACION NUMERICA

Las ecuaciones de los elementos finitos requieren para su desarrollo de la evaluación de integrales; la solución analítica de estas integrales puede ser un proceso difícil y en ocasiones imposible, por lo que la integración suele hacerse en forma numérica.

Los métodos de integración numérica usados en el método

de los elementos finitos se basan en suponer que se conocen o determinan los valores de la función a integrar en ciertos puntos y al asociársele factores de peso a estos valores y sumarlos, se determina en forma aproximada el valor de la integral.

Supóngase una función $f(x)$. La integral de $f(x)$ puede ser aproximada mediante la expresión:

$$\int f(x)\,dx = \sum_{i=1}^{n} W_i q_i + E$$

donde W_i representa a los factores de peso, q_i son los valores de $f(x)$ en algunos puntos del dominio, y E es el error generado por la aproximación de la integral.

Los dos métodos de integración usados en el método de los elementos finitos son el método de Newton-Cotes y el método de integración gaussiana. El primero de estos métodos supone que los puntos donde se conoce el valor de la función están igualmente espaciados. Las fórmulas de integración obtenidas con el método de Newton-Cotes para 2, 3 y 4 puntos conocidos en el dominio son las llamadas reglas del trapecio y de Simpson, respectivamente

La integración numérica puede tener mejores aproximaciones si en lugar de los puntos de integración definidos previamente se escogen estos en el lugar y número adecuado para tener la mayor precisión posible. En esto se basa el método de integración de Gauss.

Usando los polinomios de Lagrange pueden calcularse los puntos de integración, así como los factores de peso, que conducen a una mejor aproximación de la integral[4]. Estos parámetros son mostrados a continuación para 2, 3 y 4 puntos de integración ya que en el método de los elementos finitos difícilmente se utilizan más puntos de integración.

$\pm x_i$		W_i
	n = 2	
0.577350269		1.000000000
	n = 3	
0.774596669		0.555555556
0.000000000		0.888888889
	n = 4	
0.861136311		0.347854845
0.339981043		0.652145154

Este método de integración puede ser extendido a dos y tres dimensiones, requiriendose para ello de n^2 o

n^3 número de factores de peso y de coordenadas que definen los puntos de integración. En ambos casos tanto los factores de peso como las coordenadas tendrán los mismos valores que para el caso unidimensional, pero estos se repetiran para cada dimensión.

Los puntos de integración así como los factores de peso para los casos presentados en la tabla pueden obtenerse con la rutina *gauss*. Con base en ella, se ha escrito también la rutina *gauss2* que determina los factores de peso y coordenadas de los puntos de integración para integración gaussiana en dos dimensiones, ambas rutinas se encuentran en el apéndice A.

F. SOLUCION DE SISTEMAS DE ECUACIONES LINEALES

Los mayores tiempos de procesamiento en un programa de elementos finitos generalmente se concentran en la etapa de solución de las ecuaciones de equilibrios, por lo que gran parte de los esfuerzos destinados a mejorar las técnicas numéricas usadas por el método han sido dirigidas a este campo, incluso, los modelos de almacenamiento presentados en

el capítulo IV se deben a algoritmos de solución de sistemas de ecuaciones que no requieren para operar de los elementos eliminados en estos modelos.

Los métodos de solución de sistemas de ecuaciones lineales pueden ser divididos en métodos iterativos y métodos directos, entre éstos últimos se encuentran el método de eliminación de Gauss, la factorización de Choleski y el método de Choleski modificado, también conocido como método de Crout.

El método de Choleski se basa en la descomposición de la matriz [A] del sistema

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

que define a un sistema de ecuaciones lineales; siendo [b] el vector de términos independientes y [x] el vector incógnita. La descomposición de la matriz [A] se hace en dos matrices triangulares, una inferior [L] y otra superior [U] lo que se expresa como:

$$[A] = [L][U]$$

Una vez encontradas las matrices [L] y [U] estas sustituyen a la matriz [A] en la expresión original y haciendo

$$\{y\} = [U]\{x\} \qquad (4.1)$$

tenemos:

$$[L]\{y\} = \{b\}$$

como [L] es una matriz triangular la última expresión puede ser resuelta haciendo una sustitución llamada hacia adelante, en la que el primer elemento de {y} es obtenido directamente como b_1/l_{11}, conocido este valor se sustituye en las siguientes ecuaciones del sistema con lo que la segunda ecuación tendrá una sóla incógnita y_2 que al ser determinada permite repetir el procedimiento hasta encontrar el valor de y_n.

Una vez que se conoce {y} se puede obtener {x} con la expresión 4.1 siguiendo un procedimiento semejante al descrito en el parráfo anterior conocido como de sustitución hacia atrás pero con la diferencia de que el último valor de {x} en ser calculado será x_1, ya que la matriz [U] es triangular superior.

Cuando la matriz [A] es simétrica, como sucede en la mayoría de las aplicaciones del método del elemento finito, el algoritmo puede ser simplificado ya que para este caso la matriz [U] es igual a la transpuesta de la matriz [L].

La simplificación se debe a que solamente la matriz [L] tiene que ser calculada y almacenada.

En el método de Gauss, la matriz original es transformada en una triangular superior al hacer nulos los coeficientes abajo de la diagonal principal por medio de operaciones sobre los renglones. Al tenerse solamente la diagonal superior podemos encontrar el vector solución al hacer el procedimiento de sustitución hacia atrás exclusivamente. Si bien el no realizar la sustitución hacia adelante representa una ventaja sobre el método de Choleski el proceso de factorización es más rápido en este último. Otra ventaja del método de Choleski, es que una vez que la matriz [L] es calculada se pueden resolver sistemas de ecuaciones con distintos vectores de términos independientes usando solamente las sustituciones hacia adelante y hacia atrás, ya que a diferencia de lo que ocurre con el método de Gauss, en el método de Choleski la factorización de la matriz no depende del vector {b}. Lo anterior es especialmente importante en el método de los elementos finitos para problemas transitorios y estructurales en los que el vector de términos independientes esta formado por condiciones de frontera y cargas externas con lo que podemos analizar una estructura sometida a distintas

condiciones sin tener que formar y factorizar la matriz de rigideces global en más de una ocasión.

El método de Crout presenta una modificación respecto al de Choleski que lo hace más eficiente al introducir una tercera matriz en el proceso de factorizacion. Esta matriz [D] es una matriz diagonal que provoca que la diagonal principal de [L] sea unitaria. De esta manera el sistema de ecuaciones queda definido de la siguiente manera:

$$([L][D][L]^T)\{x\} = \{b\}$$

Si bien la presencia de una matriz parece requerir de mayor almacenamiento esto se evita al guardar su diagonal principal (únicos valores de interés ya que todos los demás son nulos) en la diagonal principal de [L].

Las expresiones para calcular [D] y [L] para el método de Crout son las siguientes[5]:

$$d_{11}=a_{11}$$

$$d_{ii} = a_{ii} - \Sigma\, d_{kk} \cdot l_{ik}^2 \quad , i>1$$

$$l_{ii} = 1$$

$$l_{ij} = (a_{ij} - \Sigma\, d_{kk} \cdot l_{ik} \cdot l_{jk})/d_{jj} \quad , i>j$$

$$l_{ij} = 0 \quad , i<j$$

de estas expresiones se deduce que para obtener un renglón de [L] y [D] se requiere conocer los coeficientes de estas matrices para renglones superiores y los valores de [A] para el renglón que se esta calculando. Esto es relevante ya que nos permite ir almacenando las matrices [L] y [D] en donde se encontraban los términos de la matriz [A].

Otro característica importante de la matriz [L] es que los coeficientes nulos de la matriz [A] no intervienen en su cálculo, y además permanecen nulos en [L] por lo que no es necesario almacenarlos, esto justifica el uso de modelos de almacenamiento como son el de perfil y en banda.

Se han desarrollado cinco rutinas para resolver sistemas de ecuaciones lineales, la primera *sol_gauss* utiliza el método de Gauss para matrices almacenadas en forma completa; las siguientes dos *fact_banda* y *obtenx_banda* usan el método de Crout para factorizar la primera y realizar las sustituciones hacia adelante y hacia atrás la segunda estando la matriz del sistema almacenada en banda; por último *fact_perfil* y *obtenx_perfil* usan el mismo método para matrices almacenadas usando el método del perfil.

G. SOLUCION DE ECUACIONES DIFERENCIALES ORDINARIAS

La formulación de problemas de difusión por el método de los elementos finitos no conducen un sistema de ecuaciones lineales como sucede en problemas de equilibrio, sino en ecuaciones diferenciales ordinarias del tipo:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{F\}$$

La solución de éstas ecuaciónes puede obtenerse discretizando la variable respecto a la cual se deriva la variable de campo $\{x\}$ (normalmente el tiempo), utilizando para ello, un método de integración paso a paso.

La discretización puede realizarse con métodos como el de diferencias finitas, el de elementos finitos, o algún otro que permita encontrar ecuaciones recurrentes con las que se pueda obtener el valor de $\{x\}$ en función de valores iniciales o previamente calculados.

Para explicar como funcionan estos métodos explicaremos el de diferencias centrales, que se basa en el método de

diferencias finitas, aplicado a un problema dinámico sin amortiguamiento y con incrementos de tiempo Δ_t.

Usando la serie de Taylor y eliminando términos de orden superior obtenemos:

$$\{\dot{x}\} = 1/(2\Delta_t)\ (\{x\}_{i+1} - \{x\}_{i-1})$$

y

$$\{\ddot{x}\} = 1/\Delta_t\ (\{x\}_{i-1} - 2\cdot\{x\}_i + \{x\}_{i+1})$$

expresiones que al ser sustituídas en la ecuación original (con [C] igual a cero) y despejando $\{x\}_{i+1}$ nos dan la siguiente ecuación:

$$(1/\Delta_t[M])\{x\}_{i+1} = \{F\} - ([K] - 2/\Delta_t[M])\{x\}_i - (1/\Delta_t[M])\{x\}_{i-1}$$

Para iniciar los cálculos debemos conocer las condiciones iniciales $\{x\}_0$, $\{\dot{x}\}_0$ y $\{\ddot{x}\}_0$ y sus valores para el instante i-1 los que pueden obtenerse con la expresión:

$$\{x\}_{i-1} = \{x\}_0 - \Delta_t\{\dot{x}\}_0 + \Delta_t/2\{\ddot{x}\}_0$$

El algoritmo que se usará para resolver la ecuación diferencial usando este método se describe a continuación:

1. Se calcula la matriz global [K].

2. Se calcula $\{x\}_{i-1}$ a partir de las condiciones iniciales y conocido Δ_t.

3. Se triangulariza la matriz $1/\Delta_t[M]$.

para cada intervalo de tiempo:

4. Calcular

$$\{R\} = \{F\} - ([K] - 2/\Delta_t[M])\{x\}_i - (1/\Delta_t[M])\{x\}_{i-1}$$

5. Se obtiene $\{x\}_{i+1}$.

6. Se calcula $\{\dot{x}\}_{i+1}$, $\{\ddot{x}\}_{i+1}$ y variables secundarias de campo si estas fueran de interés.

7. Se incrementa t.

---

1. Durocher,L.; Gasper,A. A Versatile Two-Dimensional Mesh Generator With Automatic Bandwidth Reduction. Great Britain. Computers & Structures. 1979. Págs. 561-575.

2. Akin,J. Application and Implementation of Finite Element Methods. Great Britain, Academic Press. 1982. Pág 65.

3. Cervantes, R.; Porras,V. Introducción al Método del Elemento Finito. México,D.E.P.F.I., UNAM. 1983. Págs. 54-56.

4. Zienkiewicz,O. The Finite Element Method in Engineering Science. Great Britain, Mc. Graw Hill, 1971. Pág 147.

5. Perronet, A. Les Methodes de Resolution des Systems Lineaires. Leur Techniques de Stockage. Universite Pierre et Marie Curie, France.

V. APLICACIONES DEL METODO DE LOS ELEMENTOS FINITOS

Como ya fué mencionado, las subrutinas descritas en el capítulo anterior son de carácter general, pudiendo ser utilizadas en la solución de distintos tipos de problemas en los que el método de los elementos finitos es utilizado. En este capítulo se discuten algunas de las principales aplicaciones del método de los elementos finitos en la ingeniería civil, se describen los elementos característicos de cada problema y basándose en ellos se elaboran las subrutinas dependientes de cada aplicación que junto con las presentadas en el capítulo anterior, nos permitirán resolver el problema en cuestión.

Nuevas aplicaciones pueden ser implementadas utilizando las librerías desarrolladas en este trabajo simplificándose enormemente la tarea de programación; esto proporciona una gran flexibilidad en el uso del método, convirtiéndolo en una verdadera herramienta de análisis.

La presentación que se hace de las aplicaciones es operativa, evitándose los desarrollos matemáticos que llevan a las formulaciones del método en cada caso y procurando distinguir los elementos característicos de su implementación, así como su interpretación física.

A. ELEMENTOS FINITOS EN PROBLEMAS DE EQUILIBRIO

Desde su origen, el método de los elementos finitos fué aplicado a problemas de equilibrio de la mecanica estructural. Debido a esto y a su semejanza con el análisis matricial de estructuras se ha dado una gran difusión al método en este campo.

El propósito fundamental de los problemas de equilibrio consiste en encontrar la configuración deformada y la distribución de esfuerzos en un cuerpo sometido a fuerzas externas y con ciertas restricciones en la frontera.

Los problemas de equilibrio encuentran aplicación en la ingeniería civil en las áreas de la mecánica estructural y la mecánica de suelos. En particular en este inciso se trabajará

con materiales elástico-lineales y sin variacion en el tiempo. Los problemas transitorios serán estudiados más adelante, en tanto que los problemas de plasticidad, no se incluyen, ya que generan ecuaciones no lineales cuyo tratamiento cae fuera de los alcances de este trabajo.

Las ecuaciones de elemento finito de un problema de equilibrio pueden formularse usando métodos variacionales a través del principio de la energía potencial mínima; a partir de las ecuaciones diferenciales con un método residual; o bien, con una formulación directa derivada de las técnicas de análisis matricial de estructuras, en las que se supone al medio continuo elástico como un conjunto de elementos interconectados en los nodos.

El método más común para abordar problemas de equilibrio es el método de los desplazamientos, en el cual la variable de campo primaria, los desplazamientos, es aproximada usando las funciones de forma. Otra opción dentro de los métodos de equilibrio es aquella en la que los esfuerzos constituyen la variable de campo. También se usan formulaciones mixtas donde la variable de campo está formada por esfuerzos y deformaciones en distintos puntos del medio contínuo. El

método de los desplazamientos será utilizado en el presente desarrollo.

La aproximación de los desplazamientos en los nodos que definen la malla de elementos finitos {d} estará definida por la expresión:

$$\{d\} = [N]\{d\} \qquad (5.1)$$

donde [N] representa las funciones de forma y {d} los desplazamientos en los nodos. En el caso de un problema en el plano cada nodo tendrá dos desplazamientos en direcciones ortogonales, en problemas tridimensionales serán seis los desplazamientos por cada nodo, correspondientes a las tres direcciones ortogonales y a los giros alrededor de estos ejes.

Usando el principio de continuidad para materiales elásticos, las deformaciones unitarias en un elemento {e} pueden ser calculadas a partir de los desplazamientos por medio de la relación[1]:

$$\{e\} = [B]\{d\}$$

donde [B] será en general un operador diferencial.

Sustituyendo la ecuación (5.1) en la expresión anterior tenemos:

$$\{e\} = [B][N]\{d\}$$

que relaciona los desplazamientos en los nodos de un elemento con sus deformaciones.

El operador diferencial [B] quedará determinado por el tipo de elemento que sea utilizado, de acuerdo a la ecuación:

$$e_{i,j} = 1/2\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

que en notación indicial relaciona para cada nodo del elemento las deformaciones con sus correspondientes desplazamientos, de esta manera para el caso plano [B] quedara definida como sigue:

$$[B] = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

Las ecuaciones constitutivas de los materiales elástico-lineales permiten relacionar en forma directa los esfuerzos y las deformaciones de acuerdo con la ley de Hooke. Esta

relación expresada en forma matricial resulta:

$$[\sigma] = [D][\varepsilon]$$

siendo [D] una matriz rectangular y simétrica definida para el caso tridimensional de la siguiente manera[2]:

$$[D] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & (1-2\nu)/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-2\nu)/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (1-2\nu)/2 \end{bmatrix}$$

donde E es el módulo de elásticidad o de Young y ν la relación de Poisson.

Por otra parte el estado de equilibrio que conserva el cuerpo requiere que las siguientes ecuaciones de equilibrio sean satisfechas:

$$\Sigma F_x = 0$$

$$\Sigma F_y = 0$$

$$\Sigma F_z = 0$$

donde F_x, F_y y F_z representan a las fuerzas externas aplicadas en los nodos. Estas expresiones pueden escribirse en forma matricial:

$$\{F\} = [A]\{\sigma\}$$

donde el operador diferencial [A] es igual al transpuesto de [B] que aparece en el principio de continuidad: Por lo tanto, la ecuación anterior puede ser expresada como:

$$\{F\} = [B]^T\{\sigma\}$$

Si la ecuación de continuidad es sustituida en la ecuación constitutiva de los materiales elástico-lineales y ésta a su vez es sustituida en las ecuaciones de equilibrio apenas presentada obtenemos la relación:

$$\{F\} = [B]^T[D][B][N]\{d\} \qquad (5.2)$$

que expresa una relación entre los desplazamientos nodales y las fuerzas externas actuantes.

Como [B] y su transpuesta son operadores diferenciales se requiere usar alguno de los métodos vistos en el capítulo II para aproximar esta relación como una ecuación de elementos finitos.

Si establecemos las funciones de forma para cada elemento, de modo que:

$$[E] = [B][N]$$

resulta ser [E] una matriz algebraica, ya que las funciones de interpolación [N] se determinan de manera que pudan ser derivadas con lo que la ecuación (5.2) puede transformarse en:

$$[F] = (\int_{v} [E]^T [D] [E] \, dvol) \cdot [d] \qquad (5.3)$$

El resultado de la integral, a la que se le conoce como matriz de rigidez del elemento, puede ser evaluada numéricamente convirtiendo el problema en un sistema de ecuaciones representado por:

$$[F] = [K][d]$$

La matriz de rigideces y el vector de fuerzas de cada elemento será ensamblada en una matriz global con el procedimiento antes descrito, obteniéndose de la solución de este sistema de ecuaciones global los desplazamientos en cada uno de los nodos de la malla.

La determinación de los esfuerzos en los elementos puede realizarse usando la ecuación de continuidad, ya que las deformaciones pueden ser determinadas una vez conocidos los desplazamientos. La expresión que nos permite determinar los esfuerzos en función de los desplazamientos será entonces:

$$\{\sigma\} = [D][E]\{d\}$$

Es de notar que el producto [D][E] puede almacenarse al ser generado como una operación parcial del cálculo de las matrices elementales, ahorrándose con ello el realizar este producto en dos ocasiones.

El vector de fuerzas externas puede incluir también esfuerzos o deformaciones iniciales, o bien, cargas distribuidas, como es el peso propio; este último factor puede ser evaluado mediante la integral:

$$\{F_g\} = \int [N]^T \{g\}\, dv$$

conteniendo el vector {g} a las fuerzas distribuidas por unidad de volumen.

Del breve análisis realizado a las ecuaciones de elemento finito de un problema de equilibrio elástico, resulta necesario obtener las matrices [E] y [D] para el elemento con que se trabaje, con las que la matriz elemental puede ser generada y seguir el procedimiento general del método del elemento finito ya estudiado.

Con el fin de ilustrar lo anterior a continuación se presentan dos elementos finitos, sus ecuaciones constitutivas y la subrutina que genera la matriz elemental de cada uno.

ELEMENTO FINITO BARRA

El elemento finito estructural más sencillo lo constituye la barra, elemento que al ser unido con otros de la misma especie forman las estructuras conocidas como armaduras.

Si bien la barra solo puede tener deformaciones en la dirección de su eje longitudinal, si se considera a las barras en un espacio bi o tridimensional, estas deformaciones tendrán lugar en dos o tres ejes coordenados.

Consideremos un elemento barra situado en un espacio bidimensional, como el mostrado en la figura V.1. Sean los extremos de la barra los nodos i y j, y su ángulo de inclinación respecto a la horizontal α. Sea L la longitud de la barra, E su módulo de elasticidad y A el área de su sección transversal.

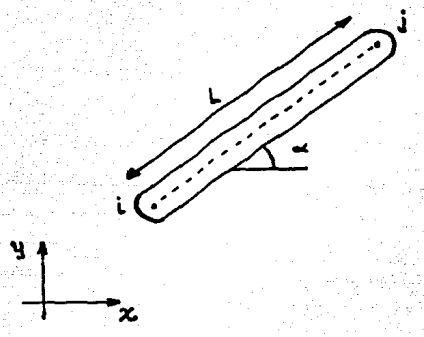

Fig. V.1

Cada elemento barra tendrá cuatro grados de libertad definidos por los desplazamientos de los nodos en cada una de las direcciones ortogonales. De esta manera si llamamos **u** y **v** a los desplazamientos en las direcciones horizontal y vertical respectivamente, el vector desplazamiento del elemento será:

$$[d]^e = \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix}$$

La deformación unitaria de la barra se obtiene con las proyecciones de los desplazamientos en el eje de la barra usando la siguiente expresión:

$$[e_i] = (u_j - u_i)\cos\alpha + (v_j - v_i)\operatorname{sen}\alpha$$

en forma matricial se tiene:

$$[e_i] = [-\cos\alpha \ -\operatorname{sen}\alpha \ \cos\alpha \ \operatorname{sen}\alpha] \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix}$$

ya que las diferencias (u_j-u_i) y (v_j-v_i) son las deformaciones en las direcciones ortogonales.

El elemento finito barra permite deformaciones axiales al eje de la barra exclusivamente, por lo que en la ley de Hooke sólo se considera la fuerza normal σ que es igual a la deformación de la barra por la rigidez de la misma, esta última obtenida como el módulo de elásticidad multiplicado por el área de la sección transversal y dividido entre la longitud de la barra.

Finalmente, las ecuaciones de equilibrio se obtienen igualando la fuerza actuante en el nodo a las fuerzas internas que concurren al nodo. Las proyecciones de las dos fuerzas axiales del elemento definen el equilibrio del elemento de la siguiente forma:

$$\begin{Bmatrix} F_{xi} \\ F_{yi} \\ F_{xj} \\ F_{yj} \end{Bmatrix} = \begin{Bmatrix} -\cos\alpha \\ -\operatorname{sen}\alpha \\ \cos\alpha \\ \operatorname{sen}\alpha \end{Bmatrix} \begin{Bmatrix} \sigma_i \\ \sigma_j \end{Bmatrix}$$

donde se puede ver que la matriz que relaciona fuerzas externas con esfuerzos es igual a la transpuesta de aquélla que relaciona deformaciones con desplazamientos.

La matriz de rigideces del elemento se obtiene a partir de las relaciones planteadas con la expresión (5.3). Así partiendo de:

$$\begin{Bmatrix} -\cos\alpha \\ -\operatorname{sen}\alpha \\ \cos\alpha \\ \operatorname{sen}\alpha \end{Bmatrix} = \frac{EA}{L} \begin{bmatrix} -\cos\alpha & -\operatorname{sen}\alpha & \cos\alpha & \operatorname{sen}\alpha \end{bmatrix}$$

resulta:

$$\begin{bmatrix} \cos^2\alpha & \operatorname{sen}\alpha\cdot\cos\alpha & -\cos^2\alpha & -\operatorname{sen}\alpha\cdot\cos\alpha \\ \operatorname{sin}\alpha\cdot\cos\alpha & \operatorname{sen}^2\alpha & -\operatorname{sen}\alpha\cdot\cos\alpha & -\operatorname{sen}^2\alpha \\ -\cos^2\alpha & -\operatorname{sen}\alpha\cdot\cos\alpha & \cos^2\alpha & \operatorname{sen}\alpha\cdot\cos\alpha \\ -\operatorname{sen}\alpha\cdot\cos\alpha & -\operatorname{sen}^2\alpha & \operatorname{sen}\alpha\cdot\cos\alpha & \cos^2\alpha \end{bmatrix}$$

Una vez que la matriz de cada elemento es calculada, se sigue el procedimiento general determinado en el capítulo II, para el cual las rutinas presentadas en el capítulo IV pueden ser utilizadas. La rutina que calcula la matriz de rigidez de un elemento finito barra se presenta a continuación:

```
barra(coor,prop,matel)
/*-------------------------------------------------------------*
 *    FUNCION:  Calcula matriz de rigidez de un elemento       *
 *              barra                                          *
 *                                                             *
 *    FORMA DE USO: barra(coor,prop,matel);                    *
 *                                                             *
 *    ARGUMENTOS: coor : coordenadas de los nodos del          *
 *                       elemento (ENTRADA).                   *
 *                prop : propiedades del elmento (ENTRADA)     *
 *                       prop[0] = area                        *
 *                       prop[1] = modulo de Young             *
 *                matel: matriz de rigidez del elemento        *
 *                       (SALIDA).                             *
 *                                                             *
 *    RUTINAS LLAMADAS: ninguna                                *
 *                                                 J.F.V. 1987 *
 *-------------------------------------------------------------*/
float   *coor,*prop,*matel;
{
    float  xi,yi,xj,yj,
           a,e,dx,dy,l,k,
           c,s,cc,ss;

    xi = *coord;
    yi = *(coord+1);
    xj = *(coord+2);
    yj = *(coord+3);
    a = *prop;
    e = *(prop+1);
    dx = xj - xi;
    dy = yj - yi;
    l = sqrt(dx*dx + dy*dy);
    c = dx/l;
```

```
        s = dy/l;
        k = e*a/l;
        cc = c*c;
        cs = c*s;
        ss = s*s;
        *matel = k*cc;
        *(matel++) = k*cs;
        *(matel++) = -k*cc;
        *(matel++) = -k*cs;
        *(matel++) = k*cs;
        *(matel++) = k*ss;
        *(matel++) = -k*cs;
        *(matel++) = -k*ss;
        *(matel++) = -k*cc;
        *(matel++) = -k*cs;
        *(matel++) = k*cc;
        *(matel++) = k*cs;
        *(matel++) = -k*cs;
        *(matel++) = -k*ss;
        *(matel++) = k*cs;
        *(matel++) = k*ss;
}
```

En la rutina barra no es necesario incluír otras subrutinas ya que las ecuaciones de elemento finito describen en forma exacta de acuerdo con la teoría de la mecánica de los materiales las relaciones entre las variables.

ELEMENTO FINITO TRIANGULAR

Hasta la aparición de los elementos isoparamétricos, el elemento finito más utilizado era el triangular, debido a la

flexibilidad que otorgaba para ajustarse a geometrías complejas y a la sencillez de su derivación y cálculo, ya que es el elemento finito bidimensional definido por el menor numéro de puntos.

Un elemento triangular como el mostrado en la figura V.2 está definido por tres nodos i, j, k, cada uno de los cuales tiene desplazamientos en dos componentes ortogonales por lo que el vector desplazamiento de cada elemento estará formado por seis componentes.

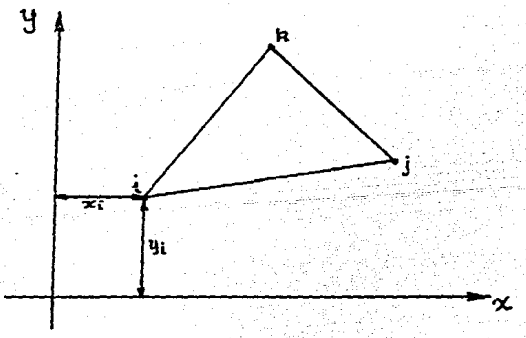

Fig. V.2

La aproximación de la variable de campo puede realizarse con dos polinomios lineales, uno para cada dirección ortogonal, de donde las funciones de forma serán las coordenadas de área presentadas en el capítulo IV.

El operador diferencial [B] que relaciona las deformaciones con los desplazamientos definido para un punto puede ser extendido a los tres nodos que definen al elemento triangular. Así, la matriz [E] resultante del producto de [B] por las funciones de forma será:

$$[E] = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & \frac{\partial N_j}{\partial x} & 0 & \frac{\partial N_k}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 & \frac{\partial N_j}{\partial y} & 0 & \frac{\partial N_k}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & \frac{\partial N_j}{\partial y} & \frac{\partial N_j}{\partial x} & \frac{\partial N_k}{\partial y} & \frac{\partial N_k}{\partial x} \end{bmatrix}$$

La matriz [E] puede formarse usando las derivadas de las funciones de forma que pueden obtenerse con la rutina *fa23* presentada en el capítulo IV.

Para formar la matriz [D] para un estado plano podemos considerar dos casos, el estado plano de esfuerzos o

el estado plano de deformación. En el primero los esfuerzos en la dirección perpendicular al plano xy son iguales a cero, en tanto que en el segundo estos esfuerzos pueden no ser nulos, pero la deformación en esta dirección SÍ lo es.

En ambos casos la matriz [D] puede ser derivada del caso general expresado en (5.3). Considerando un estado plano de esfuerzos, la matriz [D] quedará reducida a:

$$[D] = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix}$$

Una vez definidas las matrices [E] y [D], podemos formar la matriz de rigideces del elemento de acuerdo a la expresión (5.3), pero en lugar de integrar respecto al volumen se integrará respecto al área y multiplicando la expresión por el espesor del elemento que es constante; además, como [E] y [D] son constantes para el elemento, la integral definida en el elemento será igual al área del triángulo, con lo que la expresión (5.3) se transforma para este caso particular en:

$$[K] = [E]^T[D][E] \cdot t \cdot A$$

donde t es el espesor del elemento y A su área.

Auxiliados de las rutinas *fd23* y *mulmat*, y de acuerdo con la última expresión podemos desarrollar una rutina que forme la matriz de rigideces de un elemento triangular para un estado plano de esfuerzo, como a continuación se muestra:

```
tri_esf(coor,prop,matel)
/*-------------------------------------------------------*
 *    FUNCION:  Calcula matriz de rigidez de un elemento  *
 *              triangular para estado plano de esfuerzos *
 *                                                        *
 *    FORMA DE USO: tri_esf(coor,prop,matel);             *
 *                                                        *
 *    ARGUMENTOS: coor : coordenadas de los nodos del     *
 *                       elemento (ENTRADA).              *
 *                prop : propiedades de elemento (ENTRADA)*
 *                       prop[0] = modulo de Young        *
 *                       prop[1] = modulo de Poisson      *
 *                       prop[2] = espesor                *
 *                matel: matriz de rigidez del elemento   *
 *                       (SALIDA).                        *
 *                                                        *
 *    RUTINAS LLAMADAS: fd23     - calcula derivadas de las*
 *                                 funciones de forma     *
 *                      mul_mat - multiplica dos matrices *
 *                                                        *
 *                                          J.F.V. 1987   *
 *-------------------------------------------------------*/
float   *coor,*prop,*matel;
{
    float  e,v,t,area,K;
    float  dn[6],d[9],e[18],eb[18];

    e = *prop;
    v = *(++prop);
    t = *(++prop);
```

```
        /* calcula las derivadas de las funciones de forma */
        fd23(coor,dn);
        area = *dn/(*(coor+5)-(*(coor+3)));

        /* calculo de la matriz [D]*t*area */
        K = t*area*e/(1+v*v);
        d[2] = d[5] = d[6] = d[7] = 0;
        d[0] = d[4] = K;
        d[1] = d[3] = K*v;
        d[8] = K*(1-v)/2;

        /* calculo de la matriz [E] */
        e[1] = e[3] = e[7] = e[9] = e[13] = e[15] = 0;
        e[0] = e[5] = dh[0];
        e[2] = e[4] = dh[1];
        e[6] = e[11] = dh[2];
        e[8] = e[10] = dh[3];
        e[12] = e[17] = dh[4];
        e[14] = e[16] = dh[5];

        /* calculo de [E]T*[D]*t*area */
        mul_mat(e,d,6,3,3,1,ed);

        /* calculo de la matriz de rigidez */
        mul_mat(ed,e,6,3,6,0,matel);
    }
```

B. ELEMENTOS FINITOS EN PROBLEMAS DE CAMPO

Los llamados problemas de campo son aquellos gobernados por la ecuacion de Poisson e incluyen problemas de torsión, conducción de calor y flujo con potencial, entre otros.

La ecuación de Poisson que modela a los problemas de campo es:

$$\frac{\partial}{\partial x}(K_x\frac{\partial \phi}{\partial x}) + \frac{\partial}{\partial y}(K_y\frac{\partial \phi}{\partial y}) + \frac{\partial}{\partial z}(K_z\frac{\partial \phi}{\partial x}) = C$$

o bien, la ecuación de LaPlace para casos particulares cuando C es igual a cero:

$$\frac{\partial}{\partial x}(K_x\frac{\partial \phi}{\partial x}) + \frac{\partial}{\partial y}(K_y\frac{\partial \phi}{\partial y}) + \frac{\partial}{\partial z}(K_z\frac{\partial \phi}{\partial x}) = 0$$

donde ϕ es la función desconocida y C, K_x, K_y, y K_z son parámetros desconocidos que toman significado físico dependiendo del problema en cuestión.

La aproximación de las ecuaciones de Poisson y LaPlace puede llevarse a cabo por métodos variacionales ó residuales, usando cualquier formulación la ecuación de equilibrio de elemento finito será[3]:

$$[H]\{\phi\} + [F] = 0$$

esta expresión es semejante a la encontrada en problemas de elasticidad por lo que en forma general a la matriz [H] y al vector {F} se les conoce como "matriz de rigideces" y "vector de fuerzas" respectivamente, aunque estos nombres no correspondan al significado físico del problema.

La matriz y el vector del sistema estarán definidos por la sumatoria de las matrices y vectores definidos para cada

elemento, de manera que el procedimiento de ensamble presentado en el capítulo IV puede ser aplicado para su formación. La matriz y el vector correspondientes a cada elemento quedarán definidos por las siguientes expresiones[5]:

$$\overset{e}{h}_{ij} = \int_V \left[K_x \cdot \frac{\partial N_i}{\partial x} \cdot \frac{\partial N_j}{\partial x} + K_y \cdot \frac{\partial N_i}{\partial y} \cdot \frac{\partial N_j}{\partial y} + K_z \cdot \frac{\partial N_i}{\partial z} \cdot \frac{\partial N_j}{\partial z}\right] dxdydz \qquad (5.6)$$

$$\overset{e}{f}_i = -\int_V C \cdot N_i \; dV \qquad (5.7)$$

siendo N_i las funciones de forma definidas por la geometría y familia del elemento seleccionado. En problemas en dos dimensiones, la integración se realizará respecto a la superficie, eliminando el último sumando de la expresión (5.6).

En lo que se refiere a las condiciones de frontera, se presentan dos casos, uno en el que los valores de la función toman valores prescritos y otro en el que los valores de las derivadas de la función son previamente definidos.

La aplicación del primer tipo de condición de frontera puede llevarse a cabo siguiendo el procedimiento descrito en el capítulo IV, por otra parte, el segundo tipo de condición

de frontera al ser incluída en el funcional ocasionará que en la formulación variacional aparezca una integral en la formación del vector de términos independientes de la forma:

$$\int_S N_i \cdot N_i \cdot p \, ds$$

donde p representa los valores del flujo en ciertos puntos y N las funciones de interpolación dependientes del tipo de variación que se le considerará a este parámetero. Esta última condición de frontera, por sus características, debe ser definida para cada problema ya que dependerá tanto del elemento usado como de la manera en que la variación del flujo sea aproximada.

De acuerdo con lo anterior, el vector de términos independientes de las ecuaciones de equilibrio del sistema estará formado por el tipo de condiciones de frontera en las que las derivadas toman valores prescritos, conocidas como de Neumann, más los definidos por la expresión integral presentada en la expresión (5.7). Cuando el problema de campo sea modelado por la ecuación de LaPlace, el último componente del vector de términos independientes no aparece, ya que para estos casos C es igual a cero.

Una vez que la variable de campo ha sido determinada resolviendo el sistema de ecuaciones, se calculan las variables secundarias, generalmente definidas por las derivadas de las variables de campo cuyo significado, dependiendo del problema en estudio, puede ser velocidad, gradiente térmico, etc. El cálculo de estas variables secundarias puede llevarse a cabo recuperando la matriz de derivadas de las funciones de interpolación y multiplicándola por los valores de las variables de campo en los nodos correspondientes a cada elemento, para lo cual se requiere que los coeficientes de la matriz de derivadas sean almacenados cuando las matrices elementales son generadas, evitando con ello que su generación se lleve a cabo dos veces.

Utilizando las rutinas desarrolladas en el capítulo IV se construyen otras que calculan las matrices elementales en problemas de campo; como ejemplo, desarrollaremos una rutina que resuelva la ecuación de LaPlace para problemas bidimensionales. De acuerdo a dicha ecuación el parámetro C vale cero, y por lo tanto, el vector de términos independientes quedará integrado por las condiciones de frontera por lo que la expresión (5.7) no se tomará en cuenta.

Se usarán elementos rectangulares cuadráticos (con 8 nodos) isoparamétricos, para los cuales las funciones de interpolación y sus derivadas se podrán calcular con las rutinas *ff28* y *fd28*; se utiliza además, la integración gaussiana de 4 puntos por elemento, para lo cual se llama a la subrutina *gauss2* que calcula factores de peso y coordenadas de integración, y finalmente para el cálculo de la inversa y el determinante de la matriz jacobiana se usa la subrutina *detinv2*.

La subrutina para calcular la matriz elemental se presenta a continuación:

```
laplave28(ng,coor,prop,matel)
/*-------------------------------------------------------------*
 *   FUNCION:  Calcula matriz de un elemento isoparamet.       *
 *             de 8 nodos para la ecuacion de LaPlace          *
 *                                                             *
 *   FORMA DE USO: laplace28(ng,coor,prop,matel);              *
 *                                                             *
 *   ARGUMENTOS: ng    : numero de g.de lib del elem.(ENT)     *
 *               coor  : coordenadas de los nodos del          *
 *                       elemento (ENTRADA).                   *
 *               prop  : propiedades del elmento (ENTRADA)     *
 *                       prop[0] = Kx                          *
 *                       prop[1] = Ky                          *
 *               matel: matriz de rigidez del elemento         *
 *                       (SALIDA).                             *
 *                                                             *
 *   RUTINAS LLAMADAS: fd28     - calcula derivadas de las     *
 *                                funciones de forma           *
 *                     mul_mat - multiplica dos matrices       *
```

```
*                       gauss2  - determina ptos. de int.  *
*                                 y fac. de peso en 2-D    *
*                       detinv2 - calcula determinante e   *
*                                 inversa de mat. de 2x2   *
*                       limpia_real  inicializa a cero un  *
*                                    vector                *
*                                                          *
*                                              J.F.V. 1987 *
*----------------------------------------------------------*/
int     ng;
float   *coor,*prop,*matel;
{
    float  k,kx,ky,npi,det;
    float  pi[8],w[4],dn[16],jacob[4],
           jinv[4],dhg[16];

    kx = *prop;
    ky = *(prop+1);
    npi = 4;
    limpia_real(matel,ng*ng);
    gauss2(2,pi,w);
    for (i=0; i<npi; i++) {
        fd28(pi[i],pi[i+1],dn);           /* deriv. de N */
        mulmat(jacob,dn,coord,2,8,2,0); /* mat. jacob. */
        detinv2(jacob,jinv,det);          /* inversa jac.*/
        mulmat(dhg,jinv,dh,2,2,8,0);      /* deriv. glob.*/
        k = *w*det;
        w++;
        for (j=0;j<ng;j++)
            for(i=0;i<j;i++) {
                *(matel+8*i+j)  += det*(kx*dhg[8*i]*dhg[8*j]
                                       +ky*dhg[8*i+1]*dhg[8*j+1];
                *(matel+8*j+i)  = *(matel+8*i+j);
            }
    }
}
```

C. ELEMENTOS FINITOS EN PROBLEMAS TRANSITORIOS

Hasta aquí los problemas presentados no consideran la variable tiempo, sin embargo, existen muchos fenómenos que dependen del tiempo y para los cuales podemos encontrar una solución aproximada por medio del método de los elementos finitos.

Entre estos problemas de carácter transitorio podemos nombrar la transmisión de ondas en fluídos y el comportamiento dinámico de estructuras y suelos entre otros.

Los problemas transitorios se encuentran gobernados generalmente por ecuaciones diferenciales ordinarias del tipo:

$$[A]\{\ddot{x}\} + [B]\{\dot{x}\} + [C]\{x\} = \{D\} \qquad (5.8)$$

Las matrices [A], [B], y [C], se forman siguiendo el mismo sistema de ensamble presentado, y su significado depende del tipo de aplicación. Así, por ejemplo, en mecánica de sólidos [C] será la matriz de rigideces calculada de acuerdo a lo visto en el primer inciso de este capítulo, [B] representa el amortiguamiento lineal del

sistema y a [A] se le conoce como matriz de masas, si [C] es cero la ecuación (5.8) representa la ecuación de onda. Las nuevas matrices [A] y [B] son calculadas para cada elemento de acuerdo a las siguientes expresiones[6]:

$$[a]^e = \int_v [N]^T \rho [N] \, dV$$

$$[b]^e = \int_v [N]^T \mu [N] \, dV$$

siendo los parámetros μ y ρ constantes del elemento.

La solución de la ecuación diferencial ordinaria que gobierna a los problemas transitorios puede obtenerse por integración directa, usando para ello alguno de los algoritmos numéricos de paso a paso, en los que discretiza el tiempo en puntos situados a intervalos de tiempo constante; en cada uno de estos puntos la variable de campo es determinada siguiendo el mismo procedimiento que en el caso estático.

Los algoritmos de solución de problemas transitorios requieren de condiciones iniciales a partir de las cuales se obtiene la solución para el punto situado un intervalo de tiempo adelante. Siguiendo un procedimiento recurrente se encuentra la solución para el siguiente punto.

Los algoritmos de integración directa pueden obtenerse usando el método de diferencias finitas, el de Euler o algún otro usado para resolver ecuaciones diferenciales ordinarias, como el método predictor-corrector. Uno de estos métodos es el de diferencias centrales explicado en el capítulo IV.

Otra forma de resolver la ecuación (5.8) es por medio del llamado método se superposición modal, que conviene utilizar cuando se quiere obtener la solución del sistema en un número muy grande de puntos, ya que esto resultaría muy costoso en lo que a recursos computacionales se refiere, si se siguiera un método de integración directa.

En el método de superposición modal se deben calcular primero los valores y vectores característicos del sistema desamortiguado, es decir con [B] igual a cero. La ecuación (5.8) define entonces un problema de valores característicos de la forma:

$$([C] - \omega^2[A])\{eo\} = 0 \qquad (5.9)$$

que tiene tantas soluciones como grados de libertad tenga el sistema. Una solución de la ecuación (5.9) estará dada por el valor característico ω y el vector $\{eo\}$ llamado modo del sistema.

Una vez que los valores y vectores característicos son determinados se realiza una superposición de modos definida para el caso de amortiguamiento nulo, como:

$$\{x\} = \sum_{i=1}^{n} \{eo\}_i \{X_i(t)\} \qquad (5.10)$$

donde $\{X_i(t)\}$ es el vector de coordenadas generalizadas para el nodo i dado por la solución del sistema de ecuaciones diferenciales ordinarias desamortiguada y que es función del valor característico ω_i.

La expresión (5.10) permitirá conocer la variable de campo $\{x\}$ para un tiempo t sin haber antes obtenido la solución para valores cercanos del tiempo. Aunque este parece ser un procedimiento más sencillo que el de integración directa, debe tomarse en cuenta que el cálculo de valores y vectores característicos es un problema numérico complicado, por lo que en ocasiones resulta más económico usar un método de paso a paso.

Además de poder obtener la variable de campo por superposición de modos, el cálculo de valores y vectores característicos es requerido en problemas de vibración y análisis de estabilidad[7]. En estos problemas, conocidos como de valores característicos, es importante determinar los modos

del sistema llamados modos naturales de vibrar y los valores característicos que definen la frecuencia natural del sistema para cada modo.

Las aplicaciones vistas en este capítulo no han sido llevadas a casos particulares, con el objeto de poder conservar generalidad, sin embargo, en el capítulo VII se desarrollarán ejemplos auxiliados de las librerías formadas por las rutinas presentadas en este y otros capítulos, con lo que el uso de las mismas pueda ser evaluado.

---

1. Zienkiewicz,O. The Finite Element Method in Engineering Science. Great Britain, Mc. Graw Hill, 1971. Pág 19.

2. ibid, pág. 133.

3. Baker,A.J. Finite Element Computational Fluid Mechanics. Singapore, Mc. Graw Hill, 1985. Pág 25.

4. Tong,P. Finite Element Method. Basic Technique and Implementation. E.U.A., The MIT Press, 1982. Pág 41.

5. Akin,J.E. Application and Implementation of Finite Element Methods. Great Britain, Academic Press, 1982. Pág 323.

6. Thomson,W. Teoría de las Vibraciones. Colombia, Prentice/Hall Int., 1983. Págs. 183-201.

VI. PRE Y POSTPROCESAMIENTO EN LA APLICACION DEL METODO DE LOS ELEMENTOS FINITOS

Al salir el método de los elementos finitos del ambito exclusivo de universidades y centros de investigación, y extenderse su uso a despachos de ingeniería, surge el interés por comercializar los programas de elemento finito. Sin embargo, los códigos existentes resultaban demasiado complicados tanto en su manejo como en la interpretación de los resultados, por lo que surgió la necesidad de crear programas de apoyo al usuario que le faciliten la creación del modelo, la entrada de datos y la interpretación de los resultados del análisis.

La preparación de los datos que requiere para operar un programa de elementos finitos es una tarea laboriosa que debe llevarse a cabo con sumo cuidado; esta preparación incluye: la división del medio contínuo en elementos finitos, la numeración en forma ordenada de nodos y elementos, la determinación de constantes, las restricciones del modelo, las coordenadas de nodos e inicidencias de los elementos. Una vez

que los datos han sido generados, lo que en ocasiones puede llevar días, éstos deben ser introducidos a la computadora, proceso que dada la enorme cantidad de datos que se manejan puede provocar errores en ocasiones difícilmente localizables, haciendo que costosas ejecuciones del programa se lleven a cabo inútilmente.

Con el fin de evitar el problema anterior se han desarrollado programas que generan las mallas de elementos finitos, que numeran nodos con un ancho de banda mínimo, y que mediante el despliegue gráfico del modelo generado permite revisar los datos antes de realizar el análisis de los mismos. A estos programas, frecuentemente ligados con sistemas de CAD (Diseño Asistido por Computadora) se les conoce como de preprocesamiento por ejecutarse antes que el programa de elementos de finitos.

Una vez que el modelo de elementos finitos ha sido analizado, algunas técnicas de postprocesamiento de los resultados son requeridas para extraer la información que será de utilidad, ya que como del análisis se esperan obtener generalmente valores extremos, zonas críticas o el comportamiento en forma general del modelo, no todos los

valores generados por el programa son utilizados.

Entre los programas de postprocesamiento se encuentran la preparación y despliegue en forma gráfica de los valores obtenidos para las variables de campo, lo que puede llevarse a cabo con curvas de contornos o como en el caso de problemas de elasticidad mostrando la configuración deformada del dominio. Otros programas de este tipo realizan el cálculo y despliegue de valores extremos como son, por ejemplo, esfuerzos principales o velocidades máximas, y en problemas transitorios permiten la obtención de gráficas que indiquen el comportamiento respecto al tiempo de un grado de libertad del sistema o de todo el sistema mediante el uso de animación por computadora.

Como podrá notarse al revisar las funciones del pre y el postprocesamiento en el método de los elementos finitos, éstas están generalmente muy ligadas al tipo de problema a tratar. Es por esto, y por razones de espacio, que en este capítulo se presentarán solamente aquéllas que tengan un carácter general, como es el caso de la generación automática de mallas, o bien, de aquéllas técnicas que, pertenecientes a una aplicación particular, permitan ilustrar la ventaja de usar el pre y

postprocesamiento en la aplicación del método de los elementos finitos, ya que de ninguna manera se pretende cubrir totalmente en este capítulo ese tema por demás vasto.

Las técnicas de pre y postprocesamiento hacen uso extensivo de las capacidades gráficas del equipo de cómputo, por lo que muchas de las subrutinas aquí presentadas necesitan utilizar funciones dependientes del equipo que se usa, no obstante, la documentación interna de estas subrutinas señalan cuáles son éstas y qué proceso realizan, para que en caso de requerir incorporarlas a otro ambiente de cómputo solamente sea necesario cambiar el nombre de las funciones llamadas.

A. GENERACION AUTOMATICA DE MALLAS

El objetivo de un programa de generación automática de mallas de elemento finito consiste en disminuir en lo posible la cantidad de información que el usuario debe preparar e introducir a la computadora.

Si bien existen varios métodos de generación automática de mallas, todas se basan en el concepto de superelementos,

que son regiones que conservan propiedades constantes y que definen la geometría del dominio de interés. Estos elementos son divididos de acuerdo a un esquema que permite mapear de una geometría regular definida por los superelementos a la que corresponde al dominio.

La técnica que usaremos es la presentada por Zienkiewicz y Phillips[1], según la cual los elementos interiores a cada elemento se generan siguiendo un mapeo isoparamétrico.

Con el fin de ilustrar la técnica mencionada considérese un superelemento de 8 nodos como el mostrado en la figura VI.1. Las coordenadas de los nodos del superelemento quedarán definidas en términos de coordenadas curvilíneas r y s mediante las expresiones:

$$x = \sum_{i=1}^{8} N_i \cdot x_i \qquad (6.1)$$

$$y = \sum_{i=1}^{8} N_i \cdot y_i \qquad (6.2)$$

donde N_i son las funciones de forma, correspondientes en este caso al elemento rectangular cuadrilátero plano.

Utilizando las coordenadas de los nodos de cada superelemento e indicando el número de subdivisiones

requeridas en cada dirección, se pueden calcular, usando para ello las ecuaciones (6.1) y (6.2), las coordenadas de los nodos intermedios que definen a los elementos generados.

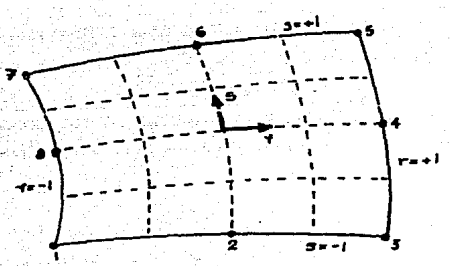

Fig. VI.1

Si la geometría del superelemento no presenta lados curvos las coordenadas de los nodos intermedios no tienen que ser proporcionados, generándose los demás nodos por interpolación lineal, con lo que se simplifica notablemente el cálculo de la generación de la malla. Una geometría puede incluir superelementos de 4 u 8 nodos pero debe cuidarse que las fronteras de todos los superelementos coincidan con las de aquéllas con las que las comparte.

El programa de generación automática de mallas que se presenta determina primero los elementos que se encuentran conectados, de manera que para dos superelementos que comparten un lado si uno de ellos es subdividido, al dividir al segundo superelemento. el lado común no será calculado porque las coordenadas de sus nodos ya han sido generadas.

En la siguiente etapa de cálculo el programa revisa si se proporcionan solo 4 nodos del superelemento, de ser así, las coordenadas de los nodos intermedios se generan por interpolación lineal y se procede a calcular los demás nodos de acuerdo al número de subdivisiones requeridas, haciendo uso de las funciones de forma de un elemento rectangular cuadrilátero. Para esto se hace uso de la rutina *ff28* presentada en el capítulo IV.

Una vez que la malla ha sido generada, ésta puede ser mostrada en forma gráfica empleando de una de las funciones que veremos más adelante, con lo que se puede revisar que la geometría del dominio ha sido respetada y que la malla generada es adecuada.

Se ilustra el uso de la rutina *genera_malla* usando los datos del dominio mostrado en la figura VI.2; la malla generada puede verse en la figura VI.3. En todos los superelementos se han pedido cuatro divisiones en cada dirección y como se puede ver no todos han sido definidos por ocho nodos, ya que sólo parte de la geometría es curva.

B. DIBUJO DE MALLAS

El dibujo de las mallas de elemento finito en algún dispositivo de graficación es útil tanto en pre como en postprocesamiento; en el primero permite revisar en forma visual que los datos que se han proporcionado o que se han generado en forma automática son correctos, permitiéndonos corregirlos en caso de ser erróneos antes de que costosas ejecuciones sean realizadas.

En postprocesamiento el dibujo de mallas permite analizar configuraciones deformadas, sirviendo como fondo para curvas de contornos y para animaciones en problemas transitorios.

Fig. VI.2

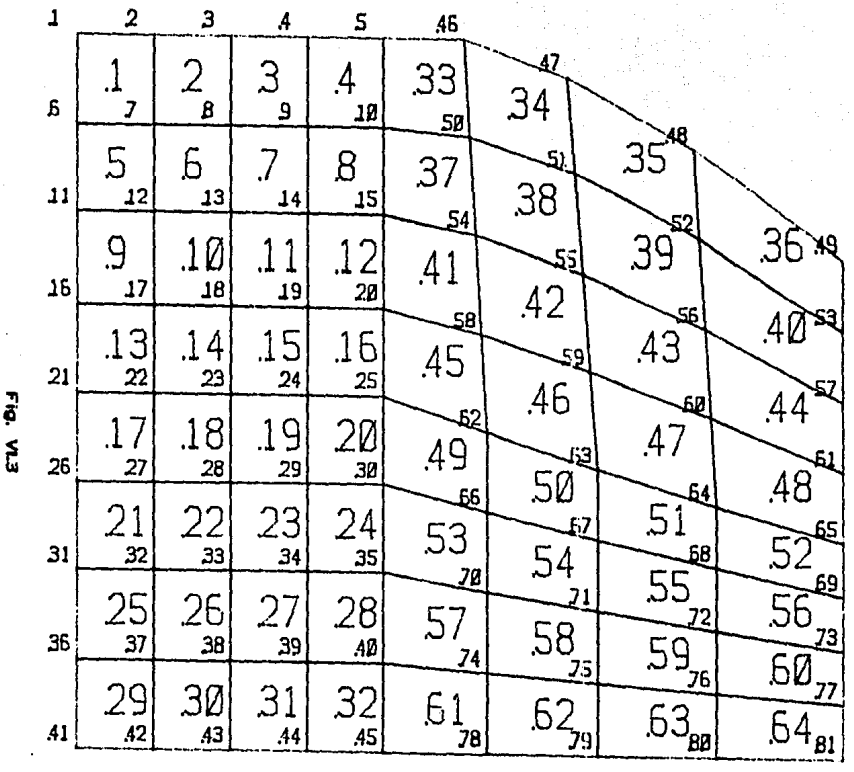
1
2
3
4
5
46
47
48
49
6
11
16
21
26
31
36
41
1
2
3
4
33
34
35
36
5
6
7
8
37
38
39
40
9
10
11
12
41
42
43
44
13
14
15
16
45
46
47
48
17
18
19
20
49
50
51
52
21
22
23
24
53
54
55
56
25
26
27
28
57
58
59
60
29
30
31
32
61
62
63
64

Fig. VL3

La rutina *dibuja_malla* usa varias funciones que dependen del dispositivo de graficación: una que inicializa el dispositivo, otra para cambiar la posición de la pluma del graficador en la región de dibujo, una más para trazar una línea recta del lugar en que se encuentra la pluma a otro punto especificado, y finalmente una rutina que dibuja caracteres especiales, letras y números. Estas rutinas constituyen parte de los elementos básicos de cualquier sistema de graficación, por lo que sólo será necesario sustituir los nombres y argumentos de éstas para transportar la rutina *dibuja_malla* a otro equipo.

En la rutina presentada en el apéndice A se utilizan instrucciones del graficador DMP-29 de Houston Instruments[2] con el que se obtuvieron los dibujos mostrados en el inciso anterior.

Una vez inicializado el dispositivo, y escogida una escala de acuerdo a las coordenadas extremas, la rutina *dibuja_malla* recorre todos los elementos trazando con líneas rectas cada uno de sus lados, con lo que la malla queda definida.

Sobre la malla dibujada es conveniente escribir los números de los nodos, esto puede lograrse usando la rutina *numera_nodos* que fué separada de la rutina *dibuja_malla*, porque en ocasiones puede no ser de interés la numeración de los nodos; lo mismo sucede con la rutina *numera_elementos* que escribe el número de cada elemento en el interior de éste.

La malla deformada en problemas de mecánica de sólidos puede graficarse usando la misma rutina, pero pasando como parámetro las coordenadas de los nodos incrementados por las deformaciones y multiplicadas por un factor de escala que permita visualizarlos. La rutina *suma_desp* que recibe como parámetro al factor de escala, calcula las coordenadas de la malla deformada.

En problemas transitorios para los que se obtienen desplazamientos para incrementos finitos de tiempo, pueden graficarse en un monitor las mallas deformadas para cada tiempo, grabar éstas y desplegarlas una después de otra dando idea del comportamiento dinámico que sigue el cuerpo con el tiempo.

C. DIBUJO DE CURVAS DE CONTORNOS

Una forma rápida y clara de visualizar los valores de una variable de campo, determinados en cada nodo con el método del elemento finito, es utilizando curvas de igual valor de la variable o curvas de contorno, que de manera semejante a las curvas de nivel permitan localizar en forma visual zonas de altas concentraciones o de valores críticos y que además puedan sugerir una nueva discretización del dominio con una malla más fina en las zonas en las que los valores de interés tengan cambios bruscos.

La forma más sencilla de obtener una curva de contorno es considerar dos puntos cercanos, ver si el valor buscado pasa por entre esos puntos, y si es así, obtener las coordenadas del punto por el que debe pasar la curva por interpolación lineal. Este procedimiento es seguido para todos los puntos vecinos con lo que se obtiene una lista de coordenadas que definen a la curva del de interés que puede ser graficada en base a líneas rectas o como curva suave, usando algún método de interpolación.

Antes de llamar a la rutina *contorno* que grafica las curvas de igual valor, debe ejecutarse la rutina *maxmin* que calcula para cada uno de los grados de libertad los valores máximos y mínimos de la variable de campo. Conocidos los valores extremos para el grado de libertad que se quiere graficar, se divide la diferencia de estos valores entre el número de curvas y se suma al valor mínimo, obteniéndose con ello los valores de las curvas por graficar.

Todos los elementos de la malla son recorridos para cada valor, determinándose para cada lado del elemento si la curva debe atravesarlo; de ser así, se calculan las coordenadas del punto por el que pasa la curva por medio de interpolación lineal.

Una vez que se tienen todos los puntos se grafica la curva que los une con la rutina *dibuja_linea*. Esta rutina, utiliza funciones que dependen del dispositivo que se use para graficar y sirve para traza líneas rectas entre los puntos que le son pasados como argumento.

El trazo de las curvas se realiza sobre la malla de elementos finitos que es dibujada antes de iniciar el cálculo

de las curvas, usando la rutina *dibuja_malla*.

La rutina *maxmin*, que al igual que la rutina *contornos* se presenta en el apéndice A, permite determinar rápidamente si los valores obtenidos estan dentro de un rango esperado o permitido, siendo también requerida por otras rutinas como la que grafica la malla deformada en un problema de mecánica de sólidos, donde los valores máximo y mínimo de desplazamientos permiten escalar las deformaciones.

D. DIBUJO DE VECTORES

Algunas de las variables de campo calculadas por el método de los elementos finitos pueden ser graficadas como vectores, este es el caso de las velocidades en un problema de mecánica de fluidos y los esfuerzos principales[3] en problemas de equilibrio.

Si en el modelo del problema se usan elementos isoparamétricos, será conveniente graficar un vector con el valor obtenido para cada punto de integración y no para cada elemento; esta idea puede extenderse también al dibujo de

curvas de contornos, ya que el valor en el elemento es resultado de una aproximación obtenida a partir de los valores en cada punto de integración.

La rutina *dibuja_vectores* calcula para todos los puntos de integración de cada elemento las coordenadas escaladas a la dimensión del dibujo, escalando la dimensión del vector de acuerdo al argumento *factor*, y calculando el ángulo de inclinación del vector a partir de los valores de las derivadas. Conocidos estos valores se grafica la recta inclinada con longitud proporcional al módulo y la flecha que indica su dirección.

La aplicación de las rutinas aquí presentadas se muestra en el capítulo siguiente en el que se ejemplifica el uso de las librerías desarrolladas en este trabajo.

---

1. Zienkiewicz,O.; Phillips,D.V. An authomatic Mesh Generation Scheme for Plane and Curved Surfaces by Isoparametric Coordinates. Int. Journal for Num. Meth. in Eng. Vol 3, 1971.

2. DM/PL Command Language. Houston Instrument. E.U.A. 1985.

3. Zienkiewicz,O. The Finite Element Method in Engineering Science. Great Britain, Mc. Graw Hill, 1971. Págs 471-475.

VII. EJEMPLOS DE APLICACION DE LAS LIBRERIAS

Con la intención de mostrar el uso de las librerías desarrolladas en este trabajo, se presentan en este capítulo dos ejemplos de aplicación del método de los elementos finitos para cuya programación se utilizaron las rutinas desarrolladas y otras adicionales, de propósito particular, que dependen de cada aplicación.

Los ejemplos seleccionados corresponden a los campos de aplicación presentados en el capítulo V con lo que se aprovecha la teoría y las subrutinas en él desarrollados.

La secuencia utilizada en el desarrollo de cada ejemplo puede servir de guía para resolver otros problemas de elemento finito en los que el uso de la librería pueda considerarse de utilidad.

En los ejemplos se puede identificar el algoritmo general del método presentado en el capítulo II para problemas de equilibrio estático.

A. Analisis Plano de Esfuerzos

El cálculo de esfuerzos y deformaciones en un sólido sometido a fuerzas externas es un problema de interés en la ingeniería estructural y en la mecánica de suelos donde el método del elemento finito ha permitido dar solución a casos que por su dificultad no podrían ser resueltos por otros medios.

En particular el análisis de esfuerzos en un dominio bidimensional usando elementos finitos triangulares es utilizado frecuentemente para explicar el método de los elementos finitos.

Las ecuaciones de elemento finito correspondientes a un estado de esfuerzo plano usando elementos triangulares ya han sido presentadas en el capítulo V, al igual que la rutina *tri_esf* que calcula la matriz elemental correspondiente.

Considerando el medio contínuo correspondiente a una viga en voladizo, con propiedades constantes (espesor, módulo de Young, etc.) y sujeto a fuerzas externas, mostrado en la

figura VII.1, éste puede ser dividido en elementos triangulares como puede observarse en la figura VII.2.

Al resolver el sistema de ecuaciones generado por el ensamble de las matrices elementales y las cargas aplicadas en los nodos, se obtienen los desplazamientos en cada nodo y puede calcularse como una segunda variable de campo los esfuerzos axiales y cortante en cada elemento.

El cálculo de los esfuerzos en cada elemento se realiza conforme a lo visto en el capítulo V con la expresión:

$$[\sigma] = [D][E][d]$$

y dado que el producto [D][E] de cada elemento es una operación intermedia al cálculo de las matrices elementales, este puede ser almacenado en un archivo temporal y recuperado cuando los desplazamientos hayan sido determinados para calcular los esfuerzos en forma económica.

Además de la rutina *obten_esf* que calcula los esfuerzos se desarrolló la rutina *aplica_cargas* que forma el vector de términos independientes con las fuerzas aplicadas en los nodos. Las otras rutinas llamadas por el programa

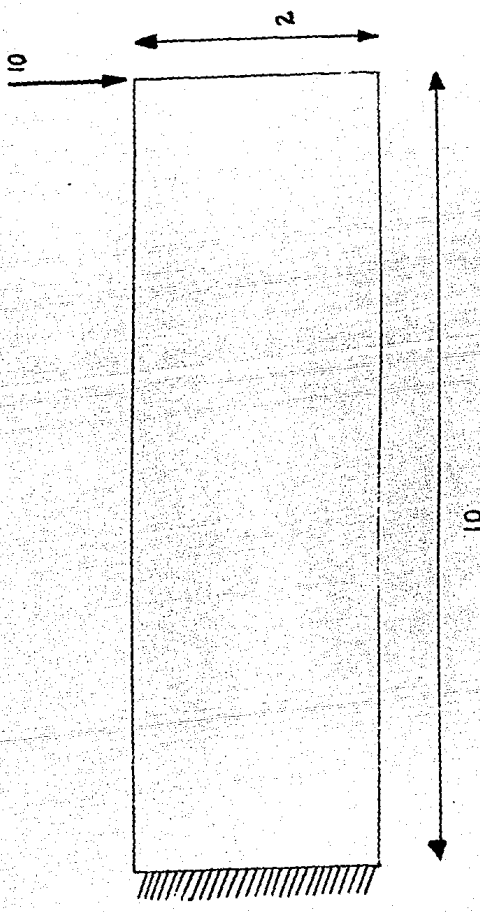

Fig. VII.1

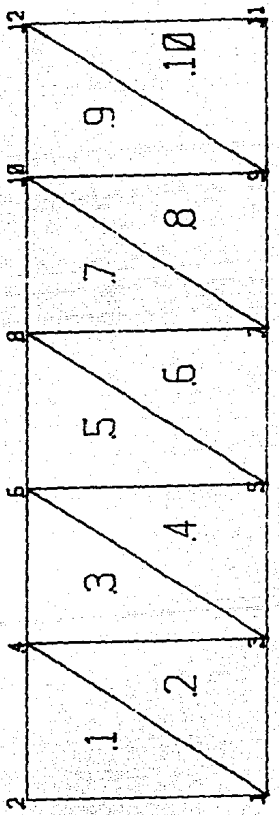

Fig. VII.2

principal tienen un carácter general y ya han sido presentadas.

En cuanto al modelo de almacenamiento de la matriz global del sistema se usarón el completo en un arreglo unidimensional y el de perfil, con el fin de observar las ventajas que ofrece este último, tanto en velocidad como en necesidades de almacenamiento.

El programa principal en el que se usa el método del perfil es mostrado a continuación:

```
#include <stdio.h>
main()
{
    char    entrada[12];

    float   vector[3000],matel[36],front[20],corel[16];

    int     nnudos,nelem,nmat,nprop,nres,ncargas,
            nn,ngls,a1,a2,a3,i,j,k,l;

    int     incid[500],nodres[20],diag[100];

    FILE    *ent;

    printf(" Nombre del archivo de entrada: ");
    scanf("%s",entrada);

    ent = fopen(entrada,"r");

    /* lectura de datos de control */
    fscanf(ent,"%30s",titulo);
```

```
    fscanf(en,"%d %d %d %d %d %d",&nnudos,&nelem,&nmat
                                  ,&nprop,&nres,&ncargas,&nn);

    /* calculo de aputadores del vector */
    ngls = nn*2;
    a1 = nnudos*2;
    a2 = a1 + nmat*nprop;
    a3 = a2 + nnudos*2;

    perfil(nelem,nn,2,nnudos,incid,diag);

    /* lectura de datos del archivo de entrada */
    lee_coor(nnudos 2,vector,ent);
    lee_incid(incid,nelem,nn,ent);
    lee_prop(nmat,nprop,&vector[a1],ent);
    lee_cond(nres,nodres,front,ent);

    /* inicializa arreglos globales */
    limpia_real(&vector[a2],ngls);
    limpia_real(&vector[a3],ngls*ngls);

    /* calculo de la matriz global */
    for(i=0,j=0; i<nelem; i++,j+=nn) {
        obten_corel(vector,&incid[j],nn,corel);
        tri_esf(corel,&vector[a1],matel);
        ensambla_perfil(nn,2,&incid[j],diag,matel,
                                              &vector[a3]);
    }

    /* calculo del vector global */
    aplica_cargas(ncargas,&vector[a2],ent);

    /* aplica condiciones de frontera */
    modifica_perfil(diag,ngls,nres,nodres,front,
                                  &vector[a3],&vector[a2]);

    /* solucion de las ecuaciones globales */
    trian_perfil(&vector[a3],diag,ngls);
    obtenx_perfil(&vector[a3],diag,ngls,&vector[a2]);

    /* calculo de variables secundarias de campo */
    obten_esf(nelem,&vector[a2],incid);
}
```

Al comparar las ejecuciones de los dos programas se observa que el método del perfil utiliza 140 elementos del arreglo para el almacenamiento de la matriz global, en tanto que cuando se almacena la matriz completa son necesarios 496 elementos. En lo que respecta al tiempo el primer modelo es aproximadamente .3 veces más rápido que el segundo. Ambos parámetros presentarán mayor diferencia en la medida en que la malla de elementos finitos sea más fina.

En la figura VII.3 se muestra la configuración deformada de la viga, donde los desplazamientos han sido amplificados para que puedan distinguirse de la configuración original. Los esfurzos axiales en dirección horizontal son graficados con la rutina *contorno* como puede observarse en la figura VII.4, lo que permite observar su comportamiento.

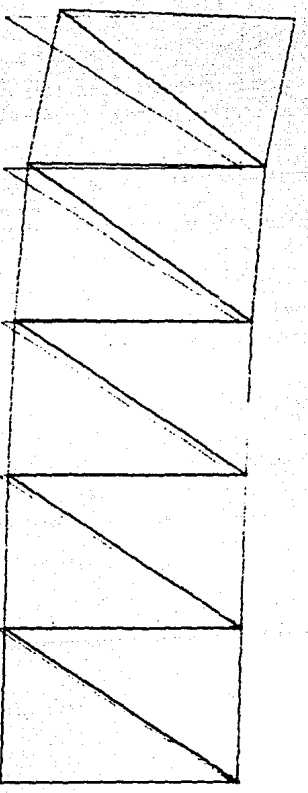

Fig. VII.3

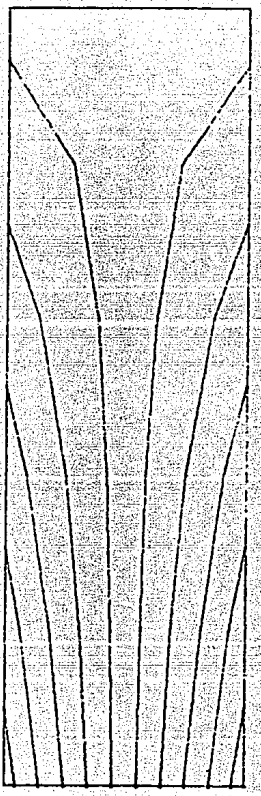

Fig. VII.4

B. FLUJO CON POTENCIAL

Algunos problemas de ingeniería civil requieren de conocer la distribución de velocidades y presiones en un fluído. Este fenómeno conocido como de flujo con potencial se modela para el caso de un fluído ideal, no viscoso y restringido a un flujo irrotacional por la ecuación de LaPlace por lo que se trata de un problema de campo (ver cap. V).

Existen dos planteamientos posibles de un problema de flujo con potencial, ambos modelados por la ecuación de LaPlace, en uno la variable de campo es la función corriente ψ que define las líneas de corriente del fluído, y el otro es a partir de la función potencial ϕ que representa una familia de superficies equipotenciales con un valor constante de potencial de velocidad, y que son perpendiculares a las líneas de corriente definidas por la función corriente.

Como ambos planteamientos se representan por la misma ecuación su desarrollo por elementos fintos resulta semejante, variando únicamente la aplicación de las condiciones de frontera. Así usaremos a la función potencial como variable de campo. Conocidos los valores de la función potencial en los

nodos de los elementos, las velocidades del fluído en cada elemento en las tres direcciones ortogonales podrá obtenerse usando las siguientes ecuaciones:

$$u = \frac{\partial \phi}{\partial x}$$

$$v = \frac{\partial \phi}{\partial y}$$

$$w = \frac{\partial \phi}{\partial y}$$

El ejemplo corresponde al flujo de un fluído ideal alrededor de un cilindro, este es un problema clásico de flujo con potencial bidimensional estudiado por numerosos autores y cuya solución anlítica ha sido encontrada por el método de imagenes[1].

Aprovechando las características de simetría de la superficie en estudio (ver fig. VII.5), el análisis puede reducirse a un cuarto de ésta, en nuestro caso tomaremos el cuarto superior izquierdo como se muestra en la figura VII.6 pudiendose extender los resultados al restante 75% de la región.

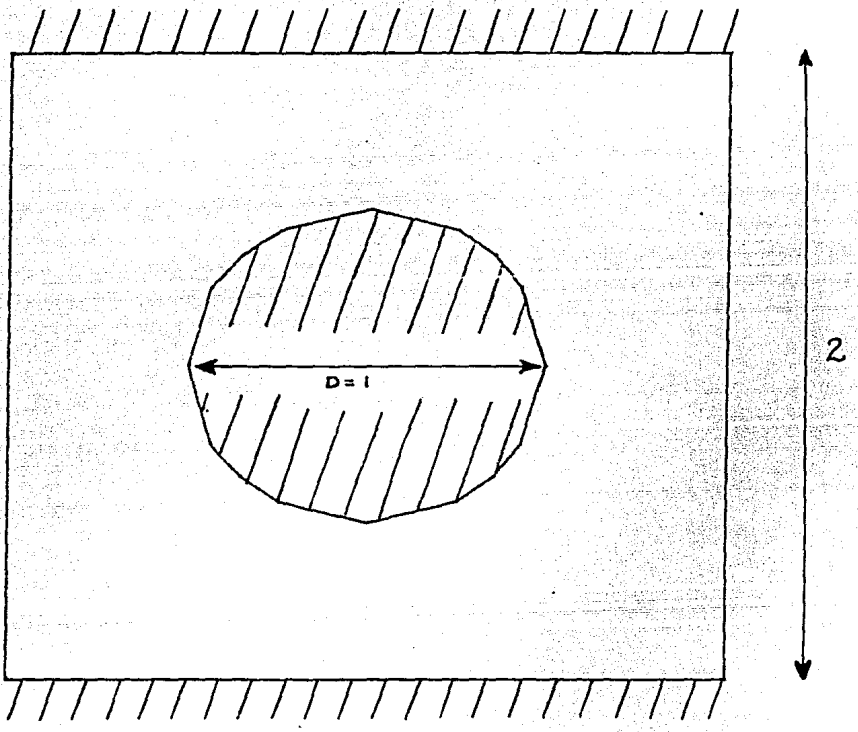

Fig VII.5

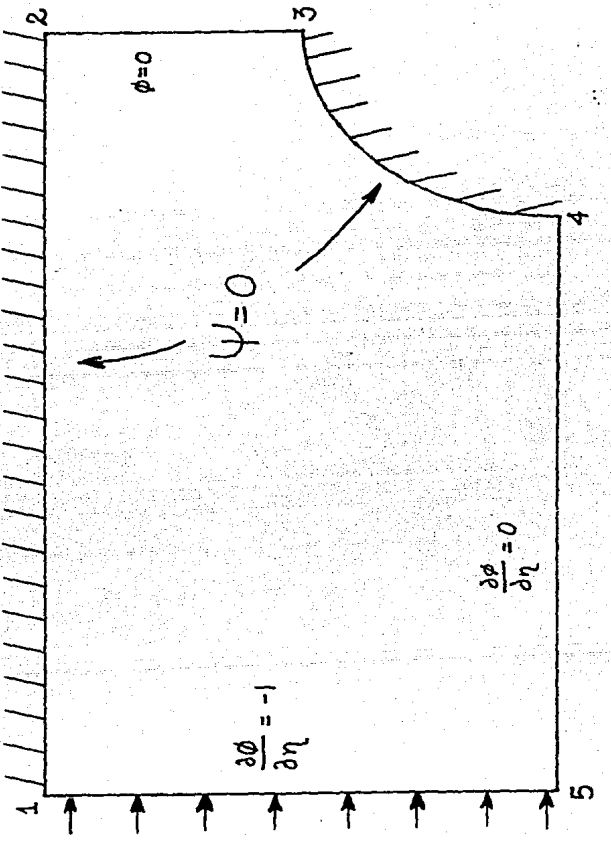
2
3
4
1
5
$\phi=0$
$\psi=0$
$\frac{\partial \phi}{\partial \eta}=-1$
$\frac{\partial \phi}{\partial \eta}=0$

Fig. VII.6

Dado que el problema es bidimensional la ecuación que gobierna el fenómeno quedara reducida a:

$$\frac{\partial}{\partial x}(K_x\frac{\partial \phi}{\partial x}) + \frac{\partial}{\partial y}(K_y\frac{\partial \phi}{\partial y}) = 0$$

La ecuación de elemento finito de la expresión anterior ya ha sido presentada en el capítulo V, así como la rutina *laplace28* que calcula la matriz elemental para estos casos usando elementos rectangulares cuadráticos e isoparamétricos por lo que con el fin de aprovecharla la geometría del problema será dividida usando este tipo de elementos como se muestra en la figura VII.7.

En la figura VII.6 se indican también las condiciones de frontera del problema, en ella se distinguen dos tipos de condiciones, las llamadas condiciones de Neumann y condiciones de Dirichlet.

Las dos fronteras de contacto entre el el fluído y la superficie sólida marcan una línea de flujo al igual que la superficie central (ptos. 4-5), y por la característica de ortogonalidad que tienen las equipotenciales con las líneas de

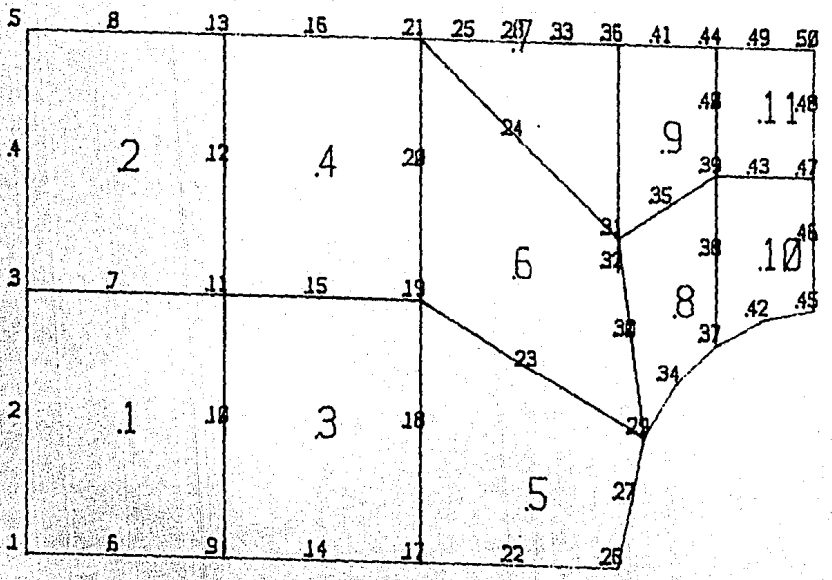

FIG. VIII.7

150

corriente la variación del potencial en una dirección ortogonal a estas superficies valdrá cero; la otra frontera que tiene condiciones de Neumann es la definida por los puntos 1 y 5 donde el fluído entra, y si suponemos que la distancia al obstáculo es suficiente para que las líneas de corriente sean paralelas el gradiente del potencial de velocidad será igual al gasto de entrada que para fines del ejemplo consideraremos unitario.

La última frontera (2-3) tiene condiciones de Dirichlet, ya que forma una equipotencial y por lo mismo podemos igualar el valor del potencial de velocidad en los nodos de esa frontera a una constante que por comodidad será cero.

Las aportaciones de las condiciones de Neumann al vector de términos independientes puede obtenerse con la expresión[2]:

$$f^e = \int_t N^T q_n ds$$

presentada anteriormente en el capítulo V. En este caso q_n es el gasto que entra y N las funciones de forma. Al utilizar las funciones de forma de un elemento unidimensional de tres nodos e integrando en la longitud del elemento obtenemos el siguiente vector elemental:

$$f^e = \frac{l}{6}\begin{Bmatrix}1\\4\\1\end{Bmatrix} q_n$$

Las condiciones de frontera de Neumann para una variación lineal son leídas y su aportación al vector de términos independientes calculada de acuerdo a la expresión anterior en la rutina *cond_neumann.*

En la programación del problema se decidio usar el modelo de almacenamiento de la matriz global en banda, para lo que se pueden utilizar algunas de las rutinas presentadas en el capítulo V.

El programa principal usado para resolver el problema en cuestión es presentado a continuación:

```
#include <stdio.h>
main()
{
    char    entrada[12];

    float   vector[3000],matel[36],front[20],corel[16];

    int     nnudos,nelem,nmat,nprop,nres1,nres2,
            nn,ngls,a1,a2,a3,i,j,K,l,banda;
    int     incid[500],nodres[20];

    FILE    *ent;

    printf("  Nombre del archivo de entrada: ");
    scanf("%s",entrada);
```

```
    ent = fopen(entrada,"r");

    /* lectura de datos de control */
    fscanf(ent,"%30s",titulo);
    fscanf(en,"%d %d %d %d %d %d %d",&nnudos,&nelem,&nmat
                                    ,&nprop,&nres,&nres2,&nn);

    /* calculo de aputadores del vector */
    ngls = nnudos;
    a1 = nnudos*2;
    a2 = a1 + nmat*nprop;
    a3 = a2 + nnudos*2;
    banda = an_banda(nelem,nn,1,incid);

    /* lectura de datos del archivo de entrada */
    lee_coor(nnudos,2,vector,ent);
    lee_incid(incid,nelem,nn,ent);
    lee_prop(nmat,nprop,&vector[a1],ent);
    lee_cond(nres,nodres,front,ent);

    /* inicializa arreglos globales */
    limpia_real(&vector[a2],ngls);
    limpia_real(&vector[a3],ngls*banda);

    /* condiciones de frontera tipo Neumann */
    cond_neumann(nres2,vector,&vector[a3],ent);

    /* calculo de la matriz global */
    for(i=0,j=0;i<nelem;i++,j+=nn) {
        obten_corel(vector,&incid[j],nn,corel);
        laplace28(nn,corel,&vector[a1],matel);
        ensambla_banda(nn,1,banda&incid[j],matel,
                                        &vector[a3]);
    }

    /* aplica condiciones de frontera tipo Dirichlet */
    modifica_banda(banda,nres1,nodres,front,&vector[a3],
                                        &vector[a2]);

    /* solucion de las ecuaciones globales */
    fact_banda(&vector[a3],banda,ngls);
    obtenx_banda(&vector[a3],&vector[a2],ngls,banda);

    /* calculo de variables secundarias de campo */
    obten_vel(nelem,&vector[a2],incid);
}
```

Además de la rutina *cond_neumann* se escribió la rutina *obten_vel* que calcula, a partir del valor del potencial en los nodos, las velocidades en cada punto de integración. Para ello, además del vector solución, la rutina utiliza las matrices de derivadas globales calculadas en la formación de las matrices elementales y que mediante una pequeña modificación en la rutina *laplace28* son almacenadas en un archivo temporal para que no tengan que ser recalculadas en la rutina *obten_vel*.

En el desarrollo del ejemplo se escribieron tres rutinas incluyendo el programa principal con un total de 140 líneas de código, número que contrasta con las 570 líneas que resultan si se consideran también las 20 rutinas de las librerías utilizadas. Lo anterior da una idea de la gran ayuda que puede respresentar el uso de las librerías.

Para mostrar los resultados obtenidos en el ejemplo se hace uso de las rutinas de postprocesamiento dearrolladas en el capítulo VI.

En la figura VII.8 se muestran las equipotenciales obtenidas a partir de la primera variable de campo con la rutina *contorno*.

Las velocidades calculadas en cada punto de integración se grafican como vectores usando la rutina *dibuja_vectores* y son mostradas en la figura VII.9.

---

1. Martín del Campo, E; Sen, H. Aplicaciones del Método de los Elementos Finitos a Problemas de Termofluidos. F.I., UNAM, Abril 1982. Pág 70.

2. Akin,J.E. Application and Implementation of Finite Element Methods. Great Britain, Academic Press, 1982. Pág 54.

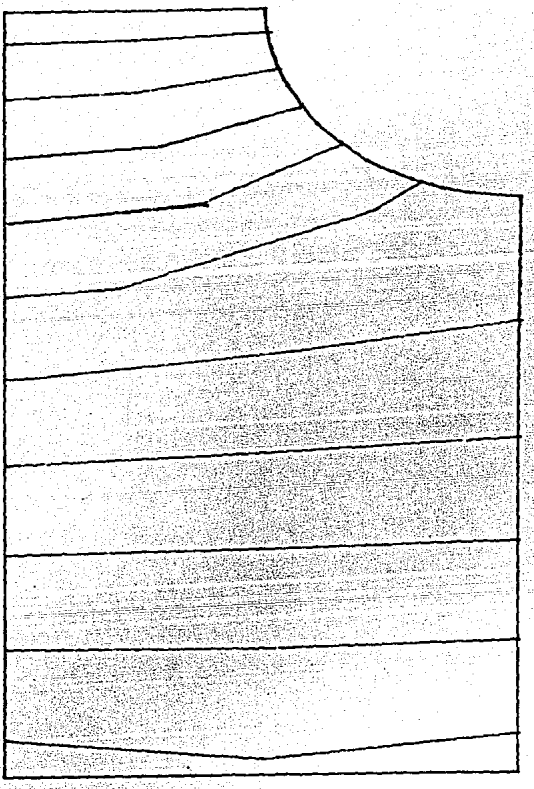

Fig. VII.8

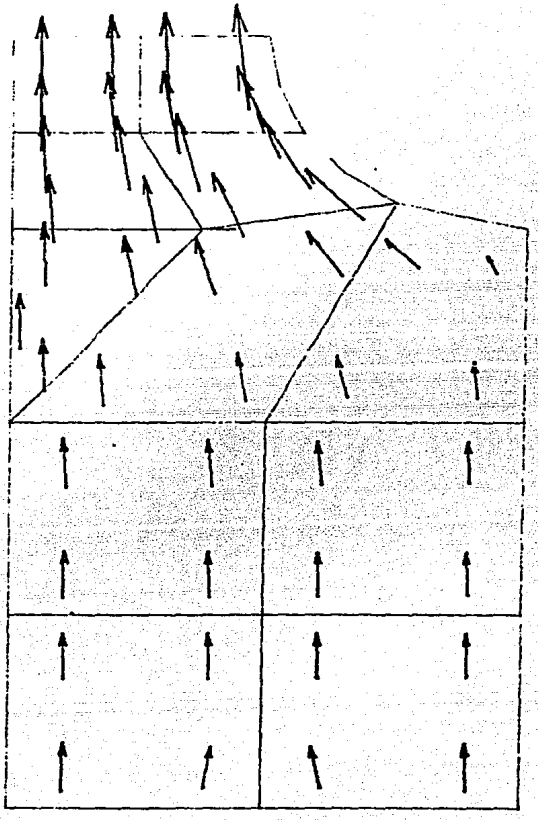

Fig. VII.9

VIII. COMENTARIOS Y CONCLUSIONES

Se ha explicado en forma general el método de los elementos finitos destacándose la importancia de contar con una herramienta que facilita el desarrollo de programas de implantación del mismo. Se han presentado los algoritmos de las rutinas que forman la librería, objeto principal del presente trabajo, cuyos listados se presentan en el apéndice A

Los ejemplos de aplicación desarrollados en el capítulo VII permiten evaluar las rutinas en función de las características deseables en los programas de elemento finito presentados en el capítulo III y con ello destacar las ventajas que el presente trabajo ofrece en la programación del método de los elementos finitos.

La primera de éstas características, la de confiabilidad, ha sido cubierta al probar cada una de las rutinas que integran la librería y verificar que los resultados que arrojan son correctos. La revisión por rutinas que permite la

programación modular facilita en gran medida la corrección de programas, especialmente cuando éstos son grandes y usan algoritmos complicados como sucede en los programas de elemento finito.

La documentación y la división de programas en módulos que realizan una y sólo una función perfectamente definida dan claridad al programa, sin embargo, conocer los algoritmos utilizados en cada rutina es necesario para entenderla y, en su caso modificarla.

Tanto la programación como la documentación siguen un estilo uniforme regido por los estándares mencionados en el capítulo III. En cuanto a la documentación, este trabajo funge como documento externo ya que presenta tanto los algoritmos utilizados como los ejemplos, aunque no deja de ser recomendable elaborar un escrito que sirva exclusivamente para estos fines.

En lo que respecta a la facilidad de uso, ésta estará en función, finalmente, del programa principal y de los módulos interactivos que le puedan ser integrados. Los argumentos de las rutinas son claros, lo que aunado a la organización que se

ha dado a éstas facilita su uso.

Una de las características más importantes de los programas de elemento finito es la eficiencia que en el uso de los recursos de la máquina se tengan, ésta ha sido una preocupación principal tanto en los algoritmos como en las estructuras de datos consideradas; en especial el uso de apuntadores en operaciones matriciales puede considerarse la aportación más importante que en este sentido se hace en el trabajo desarrollado. Por otra parte el uso de algoritmos y estructuras de datos adecuados permite implementar el método de los elementos finitos en equipos de cómputo pequeños para resolver problemas de mediana complejidad como ha podido ser demostrado.

La flexibilidad del sistema quedará determinada por el programa principal, pero la misma generalidad de las rutinas facilita la elaboración de programas de propósito general, o bien, de sistemas que puedan ser expandidos de acuerdo a las necesidades del usuario.

Finalmente la transportabilidad de los programas, así como de las mismas librerías, ha sido garantizada desde la

selección del lenguaje de programación, y al no utilizar, salvo en rutinas de graficación, funciones dependientes del equipo. En el caso de las rutinas que grafican éstas pueden ser fácilmente modificadas para adaptarse a otros equipos como lo demuestra el haberlas usado con pequeños cambios para dibujar, tanto en pantalla como en graficador.

Con base en lo anterior es importante hacer notar que si bien el uso de las librerías facilita el desarrollo de programas que cumplan con las características discutidas, será de la forma en que los distintos módulos sean integrados de lo que dependerá la calidad final del producto. En particular en los programas presentados en el capítulo VII, si bien se ha buscado sencillez en aras de la claridad que como ejemplos deben tener, consideramos que cumplen satisfactoriamente con estos criterios.

Es conveniente advertir que los programas desarrollados no resuelven del todo el problema de la aplicación del método de los elementos finitos, pues el uso de este método requiere no sólo de correr un programa sino un conocimiento profundo del mismo y capacidad para construir el modelo a partir del problema. El uso de los programas como cajas negras puede

generar resultados equivocados, por lo que debe tenerse cuidado de no caer en el sofisma de que un análisis es correcto por haber sido desarrollado en computadora.

Si bien el objetivo principal del trabajo consiste en el desarrollo de una librería de subrutinas de propósito general para la elaboración de programas de elemento finito, y como se ha visto en los parráfos anteriores y en el capítulo VII se ha probado que el uso de la librería puede no solamente facilitar el desarrollo de programas sino también incrementar su calidad, durante la elaboración del trabajo han surgido otras aportaciones que consideramos importantes y que pueden resumirse en los siguientes puntos:

- Las técnicas de la programación estructurada pueden ser incorporadas al desarrollo de programas que requieren de una gran cantidad de operaciones sin sacrificar la eficiencia de los mismos.

- El método de los elementos ^finitos no es herramienta exclusiva de centros de investigación y su uso no esta limitado a grandes equipos de cómputo; el método puede ser implementado en microcomputadoras.

- Se ha destacado la importancia de contar con elementos de pre y postprocesamiento que hagan más accesible al usuario el uso de programas de elemento finito, dándose, además, ejemplos de su aplicación.

- Lenguajes de programación no usuales en el desarrollo de programas de elemento finito como lo es C pueden representar una mejor alternativa que los lenguajes convencionales, ya que facilita la programación y permiten el desarrollo de algoritmos más eficientes.

Durante la investigación y el desarrollo del trabajo se han detectado algunas líneas sobre las cuales consideramos conveniente continuar los desarrollos y la investigación sobre el tema, en la conciencia de que las aportaciones que aquí se hacen no tienen un carácter terminal, sino es más bien un paso adelante en esta área de la ingeniería. Estos temas se proponen a continuación:

El desarrollo de una base de datos de elementos finitos como parte central de un sistema alrededor del cual se encuentren las rutinas que operan sobre los datos. A

diferencia de hace algunos años donde la parte algorítmica constituía la parte principal de un programa, en la actualidad la organización de los datos es considerada como igual o incluso más importante, es por ello que un sistema eficiente debe completarse con una organización modular de los datos sobre los que trabajan los algoritmos, este principio contenido en el concepto de base de datos empieza a tener impulso en la programación de sistemas de carácter científico y debe ser especialmente considerado en la programación del método de los elementos finitos donde la cantidad de datos que se manejan es considerable.

El desarrollo de un preprocesador que evite al usuario el manejo directo de las rutinas de la librería. Lenguajes de alto nivel que permiten manejar las rutinas de una librería de orientación científica de una manera más natural ya han sido desarrollados[1], en los que se incluyen nuevos tipos de variables e instrucciones orientadas a resolver problemas. La realización de un preprocesador con éstas características puede facilitar el uso de las librerías al evitar al usuario un conocimiento de más detalle de la misma. Por otra parte el lenguaje C otorga muchas facilidades para el desarrollo de este tipo de sistemas.

Incorporar rutinas de preprocesamiento a un sistema gráfico interactivo que facilite el modelado de problemas. El contar con un editor gráfico que integre módulos de preprocesamiento y genere archivos de datos para un programa de elemento finito facilitaría no solo la creación y prueba de modelos, sino, el uso de los programas de elemento finito.

La aplicación de sistemas basados en conocimiento al modelado e interpretación de resultados. Tanto el modelado del problema como la interpretación de los resultados constituyen las dos etapas del análisis por elemento finito que en mayor medida requieren del uso del criterio y la experiencia y resultan por lo mismo las etapas más difíciles del proceso. La reciente aplicación de algunas técnicas de la inteligencia artificial en el área de sistemas expertos comienzan a tener éxito[2] en la automatización de estos dos procesos donde aún queda mucho por hacer.

Desarrollo de algoritmos para procesamiento vectorial y en paralelo. En la carrera por elevar las velocidades de procesamiento en las computadoras, se han desarrollado máquinas con arquitecturas no convencionales donde las

operaciones no son efectuadas en forma secuencial que empiezan a ser comercializadas y siendo que en la medida en que se puedan reducir los tiempos de ejecución de los programas de elemento finito dependerá el que nuevos y más complejos problemas puedan ser atacados y resueltos, resulta conveniente comenzar a explotar estas nuevas posibilidades.

1. Rice, John. 1983. Numerical Methods, Software, and Analysis. Mc Graw Hill, pág. 459.

2. Bennett, J.; Creary, L.; Engelmore, R. y Melosh, R. 1978. SACON: A KNOWLEDGE-BASED CONSULTANT FOR STRUCTURAL ANALYSIS. Stanford University.

APENDICE A

Listado de Programas

```
limpia_ent(vector,n)
/*-------------------------------------------------------------------------*
 *   NOMBRE: limpia_ent                                                    *
 *                                                                         *
 *   FUNCION: Inicializa a ceros un arreglo de enteros.                    *
 *                                                                         *
 *   FORMA DE USO: limpia_ent(vector,n);                                   *
 *                                                                         *
 *   ARGUMENTOS: vector : apuntador al primer elemento del arreglo         *
 *                        por inicializar.                                 *
 *               n      : numero de elementos del arreglo.                 *
 *                                                                         *
 *   RUTINAS LLAMADAS: ninguna                                             *
 *                                                                         *
 *                                                          J.F.V.  1987   *
 *-------------------------------------------------------------------------*/

int     *vector;
int     n;
{
    int i;

    for (i=0;i<n;i++,vector++)
        *vector = 0;
}


limpia_real(vector,n)
/*-------------------------------------------------------------------------*
 *   NOMBRE: limpia_real                                                   *
 *                                                                         *
 *   FUNCION: Inicializar en ceros un arreglo de reales                    *
 *                                                                         *
 *   FORMA DE USO: limpia_real(vector,n);                                  *
 *                                                                         *
 *   ARGUMENTOS: vector : apuntador al primer elemento del arreglo         *
 *                        por inicializar.                                 *
 *               n      : numero de elementos del arreglo.                 *
 *                                                                         *
 *   RUTINAS LLAMADAS: ninguna                                             *
 *                                                                         *
```

```
 *                                                          J.F.V.  1987  *
 *-------------------------------------------------------------------------*/

float  *vector;
int    n;
{
    int i;

    for (i=0;i<n;i++,vector++)
        *vector = 0;
}


banda(ne, n, ng, nodos)
/*-------------------------------------------------------------------------*
 *  NOMBRE: banda                                                          *
 *                                                                         *
 *  FUNCION: Calcula el semiancho de banda de la matriz global en          *
 *           funcion de las incidencias de los nodos.                      *
 *                                                                         *
 *  FORMA DE USO: var = banda(ne, n, ng, nodos);                           *
 *                                                                         *
 *  ARGUMENTOS: ne     : numero de elementos.                              *
 *              n      : numero de nodos por elemento.                     *
 *              ng     : numero de grados  de libertad por nodo.           *
 *              nodos  : incidencias de los nodos de cada elemento         *
 *                                                                         *
 *  RUTINAS LLAMADAS: elbanda()                                            *
 *                                                                         *
 *                                                          J.F.V.  1987   *
 *-------------------------------------------------------------------------*/

int ne, n, ng, *nodos;
{
     int maxdif=1, banda, ancho, i;

     /* calcula la maxima diferencia de todos los elementos */
     for (i=0;i<ne;i++) {
          ancho = elbanda(n, nodos);
          nodos+=n;
          if (ancho > maxdif)
              maxdif = ancho;
     }
     /* calcula semiancho de banda */
     banda = ng*(maxdif+1);
     return(banda);
}
```

```
elbanda(n, nodos)
/*-------------------------------------------------------------------*
 *   NOMBRE: elbanda                                                 *
 *                                                                   *
 *   FUNCION: Calcula la maxima diferencia entre los numeros de los  *
 *            nodos de un elemento.                                  *
 *                                                                   *
 *   FORMA DE USO: var = elbanda(n, nodos);                          *
 *                                                                   *
 *   ARGUMENTOS: n      : numero de nodos por elemento.              *
 *               nodos  : incidencias de los nodos de cada elemento. *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                    J. F. V.  1987 *
 *-------------------------------------------------------------------*/

int n, *nodos;
{
     int ancho=1, dif, m, i, j;
     int *ap1, *ap2;

     m = n-1;
     ap1 = nodos;
     for (i=0; i<m; i++) {
          ap1++;
          ap2 = nodos;
          for (j=0; j<n; j++, ap2++)
               if (i != j) {
                    dif = *ap1 - *ap2;
                    if (dif < 0)
                         dif = -dif;
                    if (dif > ancho)
                         ancho = dif;
               }
     }
     return(ancho);
}

perfil(ne, n, ng, nnodos, nodos, diag)
/*-------------------------------------------------------------------*
 *   NOMBRE: perfil                                                  *
 *                                                                   *
 *   FUNCION: Calcula el vector de acceso para el manejo de una      *
 *            por el metodo del perfil.                              *
 *                                                                   *
 *   FORMA DE USO: perfil(ne, n, ng, nnodos, nodos, diag);           *
 *                                                                   *
```

```
 *  ARGUMENTOS: ne      : numero de elementos                              *
 *              n       : numero de nodos por elemento                     *
 *              ng      : numero de grados de libertad por nodo            *
 *              nnodos  : numero total de nodos                            *
 *              nodos   : incidencias de los nodos de cada elemento        *
 *              diag    : apuntador al primer elemento del vector de       *
 *                        acceso                                           *
 *                                                                         *
 *  RUTINAS LLAMADAS: limpia_ent()                                         *
 *                    obtengl()                                            *
 *                    altura_elem()                                        *
 *                                                                         *
 *                                                        J.F.V. 1987      *
 *-------------------------------------------------------------------------*/

int  ne,n,ng,nnodos,*nodos,*diag;
{
    int     altura[60],gl[60],i,j,k,l,ant,ngls,*ap;

    /* inicializa en ceros el vector de acceso */
    limpia_ent(diag,nnodos*ng);

    /* se calculan las maximas alturas de las columnas */
    for (i=0;i<ne;i++) {
        obtengl(n,ng,nodos,gl);
        nodos += n;
        altura_elem(ng,n,gl,altura);
        for (j=0;j<n*ng;j++) {
            if (*(diag+gl[j]) < altura[j])
                *(diag+gl[j]) = altura[j];
        }
    }

    /* calculo a partir de las alturas del vector de acceso */
    ngls = nnodos*ng;
    diag++;
    for (i=1;i<ngls;i++,diag++) {
        *diag += *(diag-1);
    }
}


altura_elem(ng,n,gl,altura)
/*-------------------------------------------------------------------------*
 *  NOMBRE: altura_elem                                                    *
 *                                                                         *
 *  FUNCION: Calcula la altura maxima a partir de la diagonal              *
 *           principal para cada elemento.                                 *
 *                                                                         *
```

```
 *   FORMA DE USO: altura_elem(ng, n, gl, altura);                        *
 *                                                                        *
 *   ARGUMENTOS: ng      : numero de grados de libertad por nodo          *
 *               n       : numero de nodos por elemento                   *
 *               gl      : grados de libertad asociados al elemento       *
 *               altura  : altura de las columnas correspondientes a      *
 *                         los grados de libertad del elementfo           *
 *                                                                        *
 *   RUTINAS LLAMADAS: limpia_ent()                                       *
 *                                                                        *
 *                                                          J.F.V. 1987   *
 *------------------------------------------------------------------------*/

int ng, n, *gl, *altura;
{
    int  i, min, *ap;

    /* calcula el menor numero de nodo del elemento */
    ap = gl;
    min = *gl;
    for (i=0; i<n*ng; i++) {
        ap++;
        if (*ap < min)
            min = *ap;
    }
    limpia_ent(altura, n*ng);

    /* calculo de la altura de cada columna */
    for (i=0; i<n*ng; i++, gl++, altura++)
        *altura = *gl-min+i;
}


mul_mat(a, b, n1, n2, n3, bandera, c)
/*------------------------------------------------------------------------*
 *   NOMBRE: mul_mat                                                      *
 *                                                                        *
 *   FUNCION:   Multiplica dos matrices ya sea en forma normal, o         *
 *              sus transpuestas.                                         *
 *                                                                        *
 *   FORMA DE USO: mul_mat(a, b, n1, n2, n3, band, c);                    *
 *                                                                        *
 *   ARGUMENTOS:   c     : matriz resultante                              *
 *                 a     : matriz que premultiplica                       *
 *                 b     : matriz que postmultiplica                      *
 *                 n1    : numero de renglones de la matriz A             *
 *                 n2    : numero de cols. de A y renglones de B          *
 *                 n3    : numero de columnas de la matriz B              *
```

```
 *                band : si band = 0  c=a*b                                *
 *                       si band = 1  c=aT*b                               *
 *                       si band = 2  c=a*bT                               *
 *                       si band = 3  c=aT*bT                              *
 *                c     : matriz resultante                                *
 *                                                                         *
 *                                                                         *
 *  RUTINAS LLAMADAS: ninguna                                              *
 *                                                                         *
 *                                                          J.F.V. 1987    *
 *-------------------------------------------------------------------------*/

float    *a,
         *b,
         *c;

int      n1,n2,n3,
         bandera;

{
    float  suma;

    int    i,j,k,ia,ib;
    float *ja,*jb;

    ja=a;
    jb=b;
    switch (bandera) {
    case 1:                              /* multiplica aT*b */
         for(i=0;i<n1;i++){
              ja=a;
              jb+=n2;
              for(j=0;j<n3;j++) {
                   jb-=n2;
                   suma=0.0;
                   for (k=0;k<n2;k++,ja++,jb++)
                        suma+=*ja*(*jb);
                   *(c++)=suma;
              }
         }
         break;
    case 2:                              /* multiplica aT*bT */
         for(i=0;i<n3;i++){
              for(j=0;j<n1;j++) {
                   ja=a+j;
                   jb=b;
                   suma=0.0;
                   for (k=0;k<n2;k++,ja+=n1,jb+=n3)
                        suma+=*ja*(*jb);
```

```
                *(c++)=suma;
            }
            jb=b++;
        }
        break;
    case 3:                         /* multiplica aT*bT */
        for(i=0;i<n1;i++){
            ja=a;
            for(j=0;j<n3;j++) {
                jb=b;
                suma=0.0;
                for (k=0;k<n2;k++, ja++, jb+=n3)
                    suma+=*ja*(*jb);
                *(c++)=suma;
            }
            jb=b++;
        }
        break;
    case 0:                         /* multiplica a*b */
        for(i=0;i<n1;i++) {
            jb+=n2;
            for(j=0;j<n3;j++) {
                jb-=n2;
                ja=a+j;
                suma=0.0;
                for (k=0;k<n2;k++, ja+=n1, jb++)
                    suma+=*ja*(*jb);
                *(c++)=suma;
            }
        }
    }
}

detinv1(mat, inv, det)
/*-------------------------------------------------------------------------*
 *   NOMBRE: detinv1                                                       *
 *                                                                         *
 *   FUNCION: Calcula determinante e inversa de una matriz de ixi.         *
 *                                                                         *
 *   FORMA DE USO: detinv1(mat, inv, det);                                 *
 *                                                                         *
 *   ARGUMENTOS: mat : matriz a la que se desea calcular inversa y         *
 *                     determinante                                        *
 *               inv : matriz inversa de la matriz de entrada              *
 *               det : determinante de la matriz de entrada                *
 *                                                                         *
 *   RUTINAS LLAMADAS: ninguna                                             *
 *                                                                         *
```

```
 *                                                          J. F. V.  1987  *
 *-------------------------------------------------------------------------*/

float   *mat, *inv, *det;
{
    *det  = *mat;
    *inv  = 1/(*det);
}


detinv2(mat, inv, det)
/*-------------------------------------------------------------------------*
 *   NOMBRE: detinv2                                                       *
 *                                                                         *
 *   FUNCION: Calcula determinante e inversa de una matriz de 2x2.         *
 *                                                                         *
 *   FORMA DE USO: detinv2(mat, inv, det);                                 *
 *                                                                         *
 *   ARGUMENTOS: mat : matriz a la que se desea calcular inversa y         *
 *                     determinante                                        *
 *               inv : matriz inversa de la matriz de entrada              *
 *               det : determinante de la matriz de entrada                *
 *                                                                         *
 *   RUTINAS LLAMADAS: ninguna                                             *
 *                                                                         *
 *                                                          J. F. V.  1987 *
 *-------------------------------------------------------------------------*/

float   *mat, *inv, *det;
{
    /* calculo del determinante de la matriz */
    *det = *mat*(*(mat+3)) - *(mat+1)*(*(mat+2));

    /* calculo de la matriz inversa por el metodo de cofactores */
    *inv = *(mat+3)/(*det);
    *(inv+1) = -(*(mat+1))/(*det);
    *(inv+2) = -(*(mat+2))/(*det);
    *(inv+3) = *mat/(*det);
}

detinv3(mat, inv, det)
/*-------------------------------------------------------------------------*
 *   NOMBRE: detinv3                                                       *
 *                                                                         *
 *   FUNCION: Calcula determinante e inversa de una matriz de 3x3.         *
 *                                                                         *
 *   FORMA DE USO: detinv3(mat, inv, det);                                 *
 *                                                                         *
```

```
 *   ARGUMENTOS: mat : matriz a la que se desea calcular inversa y    *
 *                     determinante                                   *
 *                 inv : matriz inversa de la matriz de entrada       *
 *                 det : determinante de la matriz de entrada         *
 *                                                                    *
 *   RUTINAS LLAMADAS: ninguna                                        *
 *                                                                    *
 *                                                    J. F. V.  1987  *
 *--------------------------------------------------------------------*/

float  *mat, *inv, *det;
{
    int  i;

    *inv = *(mat+4)*(*(mat+8)) - *(mat+5)*(*(mat+7));
    *(inv+1) = -(*(mat+1))*(*(mat+8)) + *(mat+2)*(*(mat+4));
    *(inv+2) = (*(mat+1))*(*(mat+7)) - *(mat+2)*(*(mat+7));
    *(inv+3) = -(*(mat+3))*(*(mat+8)) + *(mat+5)*(*(mat+2));
    *(inv+4) = *mat*(*(mat+8)) - *(mat+2)*(*(mat+6));
    *(inv+5) = -(*mat)*(*(mat+7)) + *(mat+2)*(*(mat+3));
    *(inv+6) = *(mat+3)*(*(mat+7)) - *(mat+4)*(*(mat+6));
    *(inv+7) = -(*mat)*(*(mat+7)) + *(mat+1)*(*(mat+6));
    *(inv+8) = *mat*(*(mat+4)) - *(mat+3)*(*(mat+1));

    /* calculo del determinante de la matriz */
    *det = *mat*(*inv) + *(mat+3)*(*(inv+1)) + *(mat+6)*(*(inv+2));

    /* calculo de la matriz inversa por el metodo de cofactores */
    for (i=0;i<9;i++)
        *(inv+i) /= *det;
}


ensambla_todo(n, ng, nnodos, nodos, matel, matglo)
/*--------------------------------------------------------------------*
 *   NOMBRE: ensambla_todo                                            *
 *                                                                    *
 *   FUNCION: ensambla una matriz elemental a la matriz global        *
 *            almacenada en forma completa.                           *
 *                                                                    *
 *   FORMA DE USO: ensambla_todo(n, ng, nnodos, nodos, matel, matglo);*
 *                                                                    *
 *   ARGUMENTOS: n      : numero de nodos por elemento                *
 *               ng     : numero de grados de libertas por nodo       *
 *               nnodos : numero total de nodos                       *
 *               nodos  : incidencias del elemento al que corresponde *
 *                        la matriz matel                             *
 *               matel  : matriz elemental por ensamblar              *
 *               matglo : matriz global en la que se almacena matel   *
```

```
 *                                                                      *
 *   RUTINAS LLAMADAS: obtengl()                                        *
 *                                                                      *
 *                                                          J.F.V  1987 *
 *---------------------------------------------------------------------*/

int      n, ng, nnodos, *nodos;
float    *matel, *matglo;
{
    int  i, j, ngle, ngls, ngi;
    int  *ap1, *ap2, gl[60];

    /* se calculan los grados de libertas asociados al elemento */
    obtengl(n, ng, nodos, gl);
    ap1 = gl;
    ngls = nnodos*ng;
    ngle = n*ng;

    /* ensamble de la matriz elemental */
    for (i=0; i<ngle; i++, ap1++) {
        ngi = *ap1*ngls;
        ap2 = gl;
        for (j=0; j<ngle; j++, ap2++) {
            *(matglo+ngi+*ap2) += *matel;
            matel++;
        }
    }
}

ensambla_banda(n, ng, banda, nodos, matel, matglo)
/*---------------------------------------------------------------------*
 *   NOMBRE: ensambla_banda                                             *
 *                                                                      *
 *   FUNCION: ensambla una matriz elemental a la matriz global          *
 *            almacenada en banda.                                      *
 *                                                                      *
 *   FORMA DE USO: ensambla_banda(n, ng, banda, nodos, matel, matglo);  *
 *                                                                      *
 *   ARGUMENTOS: n       : numero de nodos por elemento                 *
 *               ng      : numero de grados de libertas por nodo        *
 *               banda   : semiancho de banda de la matriz matglo       *
 *               nodos   : incidencias del elemento al que corresponde  *
 *                         la matriz matel                              *
 *               matel   : matriz elemental por ensamblar               *
 *               matglo  : matriz global en la que se almacena matel    *
 *                                                                      *
 *   RUTINAS LLAMADAS: obtengl()                                        *
 *                                                                      *
 *                                                          J.F.V  1987 *
 *---------------------------------------------------------------------*/
```

```
int      n, ng, banda, *nodos;
float    *matel, *matglo;
{
     int i, j, ngi, ngle;
     int *api, *ap2, gl[60];

     /* calculo de los grados de libertad asociados al elemento */
     obtengl(n, ng, nodos, gl);
     api = gl;
     ngle = n*ng;

     /* ensamble de la matriz elemental */
     for (i=0; i<ngle; i++, api++) {
          ngi = *api;
          ap2 = gl;
          for (j=0; j<ngle; j++, ap2++) {
               if (*api >= *ap2)
                    *(matglo+(ngi-*ap2)+*ap2*banda) += *matel;
               matel++;
          }
     }
}


ensambla_perfil(n, ng, nodos, diag, matel, matglo)
/*-------------------------------------------------------------------*
 *   NOMBRE: ensambla_perfil                                         *
 *                                                                   *
 *   FUNCION: ensambla una matriz elemental a la matriz global       *
 *            almacenada en perfil.                                  *
 *                                                                   *
 *   FORMA DE USO: ensambla_perfil(n, ng, nodos, diag, matel, matglo);*
 *                                                                   *
 *   ARGUMENTOS: n       : numero de nodos por elemento              *
 *               ng      : numero de grados de libertas por nodo     *
 *               nodos   : incidencias del elemento al que corresponde*
 *                         la matriz matel                           *
 *               diag    : vector de acceso de la matriz gloabal     *
 *               matel   : matriz elemental por ensamblar            *
 *               matglo  : matriz global en la que se almacena matel *
 *                                                                   *
 *   RUTINAS LLAMADAS: obtengl()                                     *
 *                                                                   *
 *                                                    J. F. V  1987  *
 *-------------------------------------------------------------------*/

int      n, ng, *nodos, *diag;
float    *matel, *matglo;
```

```
{
    int  i, j, ngi, ngle;
    int  *api, *ap2, gl[60];

    /* calcula los grados de libertad asociados al elemento */
    obtengl(n, ng, nodos, gl);
    api = gl;
    ngle = n*ng;

    /* ensamble de la matriz elemental */
    for (i=0; i<ngle; i++, api++) {
        ngi = *(diag+*api)-1;
        ap2 = gl;
      . for (j=0; j<ngle; j++, ap2++) {
            if (*api >= *ap2)
                *(matglo+ngi-*api+*ap2) += *matel;
            matel++;
        }
    }
}


obtengl (n, ng, nodos, gl)
/*-------------------------------------------------------------------*
 *   NOMBRE: obtengl                                                 *
 *                                                                   *
 *   FUNCION: Determina los numeros de los grados de libertad        *
 *            asociados a un elemento finito.                        *
 *                                                                   *
 *   FORMA DE USO: obtengl(n, ng, nodos, gl);                        *
 *                                                                   *
 *   ARGUMENTOS: n      : numero de nodos por elemento               *
 *               ng     : numero de grados de libertas por nodo      *
 *               nodos  : incidencias del elemento al que corresponde*
 *                        la matriz matel                            *
 *               gl     : vector en el que se almacenan los numeros  *
 *                        de grados de libertad obtenidos            *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                    J.F.V  1987    *
 *-------------------------------------------------------------------*/

int  n, ng, *nodos, *gl;
{
    int i, j, k;
```

```
    for (i=0;i<n;i++,nodos++) {
        k = ng*(*nodos-1);
        for (j=0;j<ng;j++,gl++)
            *gl = k + j;
    }
}

modifica_todo(ngls,nres,nodo,res,matglo,vetglo)
/*-------------------------------------------------------------------*
 *  NOMBRE: modifica_todo                                            *
 *                                                                   *
 *  FUNCION: Impone condiciones de frontera en un sistema de global  *
 *           de ecuaciones almacenado en forma completa.             *
 *                                                                   *
 *  FORMA DE USO: modifica_todo(ngls,nres,nodo,res,matglo,vetglo);   *
 *                                                                   *
 *  ARGUMENTOS: ngls   : numero de grados de libertad del sistema    *
 *              nres   : num de condiciones de frontera por aplicar  *
 *              nodo   : vector con los numeros de nodos en los que  *
 *                       se aplicaran las condiciones de frontera    *
 *              res    : vector de valores preescritos en los nodos  *
 *              matglo : matriz global almacenada en forma completa  *
 *              vetglo : vector global de terminos independientes    *
 *                                                                   *
 *  RUTINAS LLAMADAS: ninguna                                        *
 *                                                                   *
 *                                                    J. F. V  1987  *
 *-------------------------------------------------------------------*/

int     ngls,nres,*nodo;
float   *res,*matglo,*vetglo;
{
    int     i,j,l;
    float   val,*ap,*avet;

    for (i=0;i<nres;i++) {
        avet = vetglo;
        l = *nodo - 1;
        nodo++;
        val = *res;
        res++;
        ap = matglo+l*ngls;
        /* limpia el renglon l */
        for (j=0;j<ngls;j++,avet++) {
            *avet -= val*(*ap);
            *ap = 0;
            ap++;
        }
```

```
        ap = matglo+l;

        /* limpia la columa l */
        for (j=0;j<ngls;j++,ap+=ngls)
            *ap = 0;

        /* pone el valor preescrito en la diagonal principal */
        *(vetglo+l) = val;
        *(matglo+l*ngls+l) = 1;
    }
}

modifica_banda(banda,nres,nodo,res,matglo,vetglo)
/*-------------------------------------------------------------------*
 *                                                                   *
 *   NOMBRE: modifica_banda                                          *
 *                                                                   *
 *   FUNCION: Impone condiciones de frontera en un sistema de global *
 *            de ecuaciones almacenado en banda.                     *
 *                                                                   *
 *   FORMA DE USO: modifica_banda(banda,nres,nodo,res,matglo,vetglo); *
 *                                                                   *
 *   ARGUMENTOS: banda  : semiancho de banda de la matriz global     *
 *               nres   : num de condiciones de frontera por aplicar *
 *               nodo   : vector con los numeros de nodos en los que *
 *                        se aplicaran las condiciones de frontera   *
 *               res    : vector de valores preescritos en los nodos *
 *               matglo : matriz global almacenada en banda          *
 *               vetglo : vector global de terminos independientes   *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                    J.F.V  1987    *
 *-------------------------------------------------------------------*/

int     banda,nres,*nodo;
float   *res,*matglo,*vetglo;
{
    int       i,j,K,l;
    float     val,*ap,*avet;

    for (i=0;i<nres;i++,res++) {
        avet = vetglo;
        l = *nodo-1;
        nodo++;
        val = *res;
```

```
        k = l;
        if (l>=banda)
            k = banda*(l-banda+1) + banda -1;
        ap = matglo + k;

        /* limpia el renglon l */
        for (j=1; j<banda; j++, avet++) {
            *avet -= val*(*ap);
            *ap = 0;
            ap += banda-1;
        }

        avet = vetglo+l+1;
        ap = matglo+banda*l;

        /* limpia la columna l */
        for (j=1; j<banda; j++, avet++) {
            *avet -= val*(*ap);
            *ap = 0;
            ap++;
        }

        /* pone el valor preescrito en la diagonal principal */
        *(vetglo+l) = val;
        *(matglo+banda*l) = 1;
    }
}

modifica_perfil(perfil, ngls, nres, nodo, res, matglo, vetglo)
/*-----------------------------------------------------------------------*
 *  NOMBRE: modifica_perfil                                              *
 *                                                                       *
 *  FUNCION: Impone condiciones de frontera en un sistema de global      *
 *           de ecuaciones almacenado en perfil.                         *
 *                                                                       *
 *  FORMA DE USO: modifica_perfil(perfil, ngls, nres, nodo, res,         *
 *                                                matglo, vetglo);       *
 *                                                                       *
 *  ARGUMENTOS: perfil : vector de acceso a la matriz global             *
 *              ngls   : numero de grados de libertad del sistema        *
 *              nres   : num de condiciones de frontera por aplicar      *
 *              nodo   : vector con los numeros de nodos en los que      *
 *                       se aplicaran las condiciones de frontera        *
 *              res    : vector de valores preescritos en los nodos      *
 *              matglo : matriz global almacenada en perfil              *
 *              vetglo : vector global de terminos independientes        *
 *                                                                       *
```

```
 *  RUTINAS LLAMADAS: ninguna                                              *
 *                                                                         *
 *                                                          J. F. V  1987  *
 *-------------------------------------------------------------------------*/

int    *perfil, ngls, nres, *nodo;
float  *res, *matglo, *vetglo;
{
    int     i, j, k, l, m, dif, band, inc, pi;
    float   val, *ap, *avet;

    for (i=0; i<nres; i++, res++) {
        avet = vetglo;
        l = *nodo-1;
        nodo++;
        val = *res;
        pi = 0;
        if (l > 0)
            pi = *(perfil+l-1);
        ap = matglo + pi;
        dif = *(perfil+l) - pi;

        /* limpia columna l */
        for (j=0; j<dif; j++, avet++) {
            *avet -= val*(*ap);
            *ap = 0;
            ap++;
        }

        avet = vetglo+l;
        band = 0;
        k = l;
        m = 0;

        /* limpia renglon l */
        while (band == 0) {
            k++;
            m++;
            inc = *(perfil+k) - m;
            if (inc <= *(perfil+k-1) || k == ngls)
                band = -1;
            else {
                avet++;
                ap = matglo +inc-1;
                *avet -= val*(*ap);
                *ap = 0;
            }
        }
```

```
        /* pone valor preescrito en la diagonal principal */
        *(vetglo+1) = val;
        *(matglo + *(perfil+1)-1) = 1;
    }
}


ff12(s,n)
/*-----------------------------------------------------------------------*
 *   NOMBRE: ff12                                                        *
 *                                                                       *
 *   FUNCION: Calcula las funciones de forma de un elemento              *
 *            unidimensional con dos nodos.                              *
 *                                                                       *
 *   FORMA DE USO: ff12(s,n);                                            *
 *                                                                       *
 *   ARGUMENTOS: s : abcisa local del punto donde se desea calcular      *
 *                   las funciones de forma                              *
 *               n : vector con las funciones de forma resultantes       *
 *                                                                       *
 *   RUTINAS LLAMADAS: ninguna                                           *
 *                                                                       *
 *                                                          J.F.V 1987   *
 *-----------------------------------------------------------------------*/

float   s,
        *n;
{

    *n = .5*(1-s);
    *(n+1) = .5*(1+s);
}


ff13(s,n)
/*-----------------------------------------------------------------------*
 *   NOMBRE: ff13                                                        *
 *                                                                       *
 *   FUNCION: Calcula las funciones de forma de un elemento              *
 *            unidimensional con tres nodos.                             *
 *                                                                       *
 *   FORMA DE USO: ff13(n,s);                                            *
 *                                                                       *
 *   ARGUMENTOS: s : abcisa local del punto donde se desea calcular      *
 *                   las funciones de forma                              *
 *               n : vector con las funciones de forma resultantes       *
 *                                                                       *
 *   RUTINAS LLAMADAS: ninguna                                           *
 *                                                                       *
```

```
 *                                                          J. F. V  1987 *
 *-----------------------------------------------------------------------*/

float   s,
        *n;
{

    *n = .5*s*(s-1);
    *(n+1) = 1-s*s;
    *(n+2) = .5*s*(s+1);
}


ff24(s, t, n)
/*-----------------------------------------------------------------------*
 *  NOMBRE: ff24                                                         *
 *                                                                       *
 *  FUNCION: Calcula las funciones de forma de un elemento               *
 *           bidimensional con cuatro nodos.                             *
 *                                                                       *
 *  FORMA DE USO: ff24(s,t,n);                                           *
 *                                                                       *
 *  ARGUMENTOS: s : abcisa local del punto donde se desea calcular       *
 *                  las funciones de forma                               *
 *              t : ordenada local del punto donde se desea calcular     *
 *                  las funciones de forma                               *
 *              n : vector con las funciones de forma resultantes        *
 *                                                                       *
 *  RUTINAS LLAMADAS: ninguna                                            *
 *                                                                       *
 *                                                          J. F. V  1987 *
 *-----------------------------------------------------------------------*/

float   s, t,
        *n;
{
    float   sp, sm, tp, tm;

    sp = 1 + s;
    sm = 1 - s;
    tp = 1 + t;
    tm = 1 - t;

    /* calculo de las funciones de forma */
    *n = .25*sm*tm;
    *(n+1) = .25*sp*tm;
    *(n+2) = .25*sp*tp;
    *(n+3) = .25*sm*tp;
}
```

```
fd24(s, t, dn)
/*-------------------------------------------------------------------*
 *   NOMBRE: fd24                                                    *
 *                                                                   *
 *   FUNCION: Calcula las derivadas de las funciones de forma de un  *
 *            elemento bidimensional con cuatro nodos.               *
 *                                                                   *
 *   FORMA DE USO: ffd24(s, t, dn);                                  *
 *                                                                   *
 *   ARGUMENTOS: s : abcisa local del punto donde se desea calcular  *
 *                   las derivadas de las funciones de forma         *
 *               t : ordenada local del punto donde se desea calcular*
 *                   las derivadas de las funciones de forma         *
 *              dn: vector con las derivadas de las funciones de     *
 *                   forma resultantes                               *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                      J.F.V  1987  *
 *-------------------------------------------------------------------*/

float   s, t,
       *dn;
{
    float   sp, sm, tp, tm;

    sp = 1 + s;
    sm = 1 - s;
    tp = 1 + t;
    tm = 1 - t;

    /* calculo de las derivadas de las funciones de forma */
    *dn = -.25*tm;
    *(dn+1) = .25*tm;
    *(dn+2) = .25*tp;
    *(dn+3) = -.25*tp;
    *(dn+4) = -.25*sm;
    *(dn+5) = -.25*sp;
    *(dn+6) = .25*sp;
    *(dn+7) = .25*sm;
}

ff28(s, t, n)
/*-------------------------------------------------------------------*
 *   NOMBRE: ff28                                                    *
 *                                                                   *
 *   FUNCION: Calcula las funciones de forma de un elemento          *
 *            bidimensional con ocho nodos.                          *
 *                                                                   *
```

```
 *   FORMA DE USO:  ff28(s, t, n);                                          *
 *                                                                          *
 *   ARGUMENTOS:  s :  abcisa local del punto donde se desea calcular       *
 *                       las funciones de forma                             *
 *                t :  ordenada local del punto donde se desea calcular     *
 *                       las funciones de forma                             *
 *                n :  vector con las funciones de forma resultantes        *
 *                                                                          *
 *   RUTINAS LLAMADAS:  ninguna                                             *
 *                                                                          *
 *                                                          J. F. V  1987  *
 *--------------------------------------------------------------------------*/

float   s, t,
        *n;
{
    float   sp, sm, tp, tm;

    sp = 1 + s;
    sm = 1 - s;
    tp = 1 + t;
    tm = 1 - t;

    /* calculo de las funciones de forma */
    *n = .25*sm*tm*(sm+tm-3);
    *(n+1) = .25*sp*tm*(sp+tm-3);
    *(n+2) = .25*sp*tp*(sp+tp-3);
    *(n+3) = .25*sm*tp*(sm+tp-3);
    *(n+4) = 0.5*tm*(1-s*s);
    *(n+5) = 0.5*sp*(1-t*t);
    *(n+6) = 0.5*tp*(1-s*s);
    *(n+7) = 0.5*sm*(1-t*t);
}


fd28(s, t, dn)
/*--------------------------------------------------------------------------*
 *   NOMBRE:  fd28                                                          *
 *                                                                          *
 *   FUNCION:  Calcula las derivadas de las funciones de forma de un        *
 *             elemento bidimensional con ocho nodos.                       *
 *                                                                          *
 *   FORMA DE USO:  fd28(s, t, dn);                                         *
 *                                                                          *
 *   ARGUMENTOS:  s :  abcisa local del punto donde se desea calcular       *
 *                       las derivadas de las funciones de forma            *
 *                t :  ordenada local del punto donde se desea calcular     *
 *                       las derivadas de las funciones de forma            *
 *                n :  vector con las derivadas de las funciones de         *
 *                       forma resultantes                                  *
```

```
 *                                                                      *
 *   RUTINAS LLAMADAS: ninguna                                          *
 *                                                                      *
 *                                                        J.F.V  1987 *
 *----------------------------------------------------------------------*/

float    s, t,
         *dn;
{
     float    sp, sm, tp, tm;

     sp = 1 + s;
     sm = 1 - s;
     tp = 1 + t;
     tm = 1 - t;

     /* calculo de las derivadas de las funciones de forma */
     *dn = -.25*tm*(sm+sm+tm-3);
     *(dn+1) = -.25*sm*(tm+sm+tm-3);
     *(dn+2) = .25*tm*(sp+sp+tm-3);
     *(dn+3) = -.25*sp*(tm+sp+tm-3);
     *(dn+4) = .25*tp*(sp+sp+tp-3);
     *(dn+5) = .25*sp*(tp+sp+tp-3);
     *(dn+6) = -.25*tp*(sm+sm+tp-3);
     *(dn+7) = .25*sm*(tp+sm+tp-3);
     *(dn+8) = -s*tm;
     *(dn+9) = -.5*(1-s*s);
     *(dn+10) = .5*(1-t*t);
     *(dn+11) = -t*sp;
     *(dn+12) = -s*tp;
     *(dn+13) = .5*(1-s*s);
     *(dn+14) = -.5*(1-t*t);
     *(dn+15) = -t*sm;
}


ff38(s, t, r, n)
/*----------------------------------------------------------------------*
 *   NOMBRE: ff38                                                       *
 *                                                                      *
 *   FUNCION: Calcula las funciones de forma de un elemento             *
 *            tridimensional con ocho nodos.                            *
 *                                                                      *
 *   FORMA DE USO: ff38(s, t, r, n);                                    *
 *                                                                      *
 *   ARGUMENTOS: s : abcisa local del punto donde se desea calcular     *
 *                   las funciones de forma                             *
 *               t : ordenada local del punto donde se desea calcular   *
 *                   las funciones de forma                             *
```

```
 *                r : altura local del punto donde se desea calcular  *
 *                     las funciones de forma                         *
 *                n : vector con las funciones de forma resultantes   *
 *                                                                    *
 *   RUTINAS LLAMADAS: ninguna                                        *
 *                                                                    *
 *                                                        J.F.V 1987  *
 *--------------------------------------------------------------------*/

float    s,t,r,
        *n;
{
    float    sp,sm,tp,tm,rp,rm;

    sp = 1 + s;
    sm = 1 - s;
    tp = 1 + t;
    tm = 1 - t;
    rp = 1 + r;
    rm = 1 - r;

    /* calculo de las funciones de forma */
    *n = .125*sp*tp*rp;
    *(n+1) = .125*sp*tm*rp;
    *(n+2) = .125*sp*tm*rm;
    *(n+3) = .125*sp*tp*rm;
    *(n+4) = .125*sm*tp*rp;
    *(n+5) = .125*sm*tm*rp;
    *(n+6) = .125*sm*tm*rm;
    *(n+7) = .125*sm*tp*rm;
}


fd38(s,t,r,dn)
/*--------------------------------------------------------------------*
 *   NOMBRE: fd38                                                     *
 *                                                                    *
 *   FUNCION: Calcula las derivadas de las funciones de forma de un   *
 *            elemento tridimensional con ocho nodos.                 *
 *                                                                    *
 *   FORMA DE USO: fd38(s,t,r,dn);                                    *
 *                                                                    *
 *   ARGUMENTOS: s : abcisa local del punto donde se desea calcular   *
 *                    las derivadas de las funciones de forma         *
 *               t : ordenada local del punto donde se desea calcular *
 *                    las derivadas de las funciones de forma         *
 *               r : altura local del punto donde se desea calcular   *
 *                    las derivadas de las funciones de forma         *
 *               dn: vector con las derivadas de las funciones de     *
 *                    forma resultantes                               *
```

```
*                                                                    *
*  RUTINAS LLAMADAS: ninguna                                         *
*                                                                    *
*                                                        J.F.V 1987  *
*--------------------------------------------------------------------*/

float    s,t,r,
         *dn;
{
    float    sp,sm,tp,tm,rp,rm;

    sp = 1 + s;
    sm = 1 - s;
    tp = 1 + t;
    tm = 1 - t;
    rp = 1 + r;
    rm = 1 - r;

    /* calculo de las derivadas de las funciones de forma */
    *dn = .125*tp*rp;
    *(dn+1) = .125*sp*rp;
    *(dn+2) = .125*sp*tp;
    *(dn+3) = .125*tm*rp;
    *(dn+4) = -.125*sp*rp;
    *(dn+5) = .125*sp*tm;
    *(dn+6) = .125*tm*rm;
    *(dn+7) = -.125*sp*rm;
    *(dn+8) = -.125*sp*tm;
    *(dn+9) = .125*tp*rm;
    *(dn+10) = .125*sp*rm;
    *(dn+11) = -.125*sp*tp;
    *(dn+12) = -.125*tp*rp;
    *(dn+13) = .125*sm*rp;
    *(dn+14) = .125*sm*tp;
    *(dn+15) = -.125*tm*rp;
    *(dn+16) = -.125*sm*rp;
    *(dn+17) = .125*sm*tm;
    *(dn+18) = -.125*tm*rm;
    *(dn+19) = -.125*sm*rm;
    *(dn+20) = -.125*sm*tm;
    *(dn+21) = -.125*tp*rm;
    *(dn+22) = .125*sm*rm;
    *(dn+23) = -.125*sm*tp;
}
```

```
ff23(coor, r, s, l)
/*-------------------------------------------------------------------*
 *   NOMBRE: ff23                                                    *
 *                                                                   *
 *   FUNCION: Calculo de las funciones de forma de un elemento       *
 *            triangular de tres nodos.                              *
 *                                                                   *
 *   FORMA DE USO: ff23(coor, r, s, l);                              *
 *                                                                   *
 *   ARGUMENTOS: coor: coordenadas de los nodos del elemento         *
 *               t   : ordenada local del punto donde se desea       *
 *                     calcular las funciones de forma               *
 *               r   : altura local del punto donde se desea         *
 *                     calcular las funciones de forma               *
 *               l   : vector con las funciones de forma resultantes *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                    J. F. V  1987  *
 *-------------------------------------------------------------------*/

float  *coor, r, s, *l;
{
    float   xi, xj, xk, yi, yj, yk,
            ai, aj, ak, bi, bj, bk,
            ci, cj, ck, area2;

    xi = *coor;
    yi = *(++coor);
    xj = *(++coor);
    yj = *(++coor);
    xk = *(++coor);
    yk = *(++coor);

    ai = xj*yk - xk*yj;
    aj = xk*yi - xi*yk;
    ak = xi*yj - xj*yi;
    bi = yj - yk;
    bj = yk - yi;
    bk = yi - yj;
    ci = xk - xj;
    cj = xi - xk;
    ck = xj - xi;

    /* calculo del area del elemento */
    area2 = 0.5*(ai+aj+ak);

    /* calculo de funciones de forma */
    *l = (ai + bi*r + ci*s)/area2;
```

```
    *(l+1) = (aj + bj*r + cj*s)/area2;
    *(l+2) = (ak + bk*r + ck*s)/area2;
}


fd23(coor, dl)
/*-------------------------------------------------------------------------*
 *   NOMBRE: fd23                                                          *
 *                                                                         *
 *   FUNCION: Calculo de las derivadas de las funciones de forma de        *
 *            un elemento triangular de tres nodos.                        *
 *                                                                         *
 *   FORMA DE USO: fd23(coor, dl);                                         *
 *                                                                         *
 *   ARGUMENTOS: coor: coordenadas de los nodos del elemento               *
 *               dl  : vector con las derivadas de las funciones de        *
 *                     forma resultantes                                   *
 *                                                                         *
 *   RUTINAS LLAMADAS: ninguna                                             *
 *                                                                         *
 *                                                        J.F.V  1987      *
 *-------------------------------------------------------------------------*/

float  *coor,
       *dl;
{
    float  xi, xj, xk, yi, yj, yk,
           area2, bi, bj, bk,
           ci, cj, ck;

    xi = *coor;
    yi = *(++coor);
    xj = *(++coor);
    yj = *(++coor);
    xk = *(++coor);
    yk = *(++coor);

    /* calculo del area del elemento */
    area2 = 0.5*((xj*yk - xk*yj)+(xk*yi - xi*yk)+(xi*yj - xj*yi));

    /* calculo de las derivadas de las funciones de forma */
    *dl = (yk - yj)/area2;
    *(dl+1) = (xk - xj)/area2;
    *(dl+2) = (yk - yi)/area2;
    *(dl+3) = (xi - xk)/area2;
    *(dl+4) = (yi - yj)/area2;
    *(dl+5) = (xj - xi)/area2;
    return(area2/2);
}
```

```
gauss(n, x, w)
/*-------------------------------------------------------------------*
 *   NOMBRE: gauss                                                   *
 *                                                                   *
 *   FUNCION: Calculo de las coordenadas de los puntos de integracion*
 *            y factores de peso para integracion gaussiana en una   *
 *            dimension.                                             *
 *                                                                   *
 *   FORMA DE USO: gauss(n, x, w);                                   *
 *                                                                   *
 *   ARGUMENTOS:  n : numero de puntos de integracion                *
 *                x : coordenadas de los ptos. de integracion        *
 *                w : factores de peso                               *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                    J.F.V  1987    *
 *-------------------------------------------------------------------*/

int     n;
float   *x, *w;
{
    switch(n) {
    /* dos puntos de integracion */
    case 2: *x = 0.577350269;
            *(x+1) = -(*x);
            *w = *(w+1) = 1.0;
            break;

    /* tres puntos de integracion */
    case 3: *x = 0.774596667;
            *(x+1) = 0;
            *(x+2) = -(*x);
            *w = *(w+2) = 0.555555556;
            *(w+1) = 0.888888889;
            break;

    /* cuatro puntos de integracion */
    case 4: *x = 0.861136311;
            *(x+1) = 0.339981044;
            *(x+2) = -(*(x+1));
            *(x+3) = -(*x);
            *w = *(w+3) = 0.347854137;
            *(w+1) = *(w+2) = 0.652145155;
            break;
    }
}
```

```
gauss2(n, x, w)
/*-------------------------------------------------------------------*
 *   NOMBRE: gauss2                                                  *
 *                                                                   *
 *   FUNCION: Calculo de las coordenadas de los puntos de integracion*
 *            y factores de peso para integracion gaussiana en dos   *
 *            dimensiones.                                           *
 *                                                                   *
 *   FORMA DE USO: gauss2(n, x, w);                                  *
 *                                                                   *
 *   ARGUMENTOS: n : numero de puntos de integracion                 *
 *               x : coordenadas de los ptos. de integracion         *
 *               w : factores de.peso                                *
 *                                                                   *
 *   RUTINAS LLAMADAS: gauss()                                       *
 *                                                                   *
 *                                                  J.F.V  1987      *
 *-------------------------------------------------------------------*/

int     n;
float   *x, *w;
{
    int     i, j;
    float   xi[4], wi[4];

    /* obtiene parametros para una dimension */
    gauss(n, xi, wi);

    /* calculo de parametros para dos dimensiones */
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            *w = wi[i]*wi[j];
            w++;
            *x = xi[j];
            x++;
            *x = xi[i];
            x++;
        }
}


sol_gauss(mat, x, n)
/*-------------------------------------------------------------------*
 *   NOMBRE: sol_gauss                                               *
 *                                                                   *
 *   FUNCION: Resuelve un sistema de ecuaciones por el metodo de     *
 *            gauss.                                                 *
 *                                                                   *
 *   FORMA DE USO: sol_gauss(mat, x, n);                             *
```

```
 *                                                                      *
 *  ARGUMENTOS:  mat  : matriz del sistema                              *
 *               x    : vector de terminos independientes y vector      *
 *                      resultante al terminar el programa              *
 *               n    : orden del sistema de ecuaciones                 *
 *                                                                      *
 *  RUTINAS LLAMADAS: ninguna                                           *
 *                                                                      *
 *                                                        J. F. V  1987 *
 *----------------------------------------------------------------------*/

float   *mat,
        *x;

int     n;
{
    float    p, sum;

    int      i, j, k;

    for (k=0; k< n-i; k++) {
        for (i=k+1; i<n; i++) {
            *(mat+k*n+i) = *(mat+k*n+i)/(*(mat+k*n+k));
            p = *(mat+k*n+i);
            for (j=k+1; j<n; j++)
                *(mat+j*n+i)=*(mat+j*n+i)-(*(mat+j*n+k))*p;
            *(x+i)=*(x+i)-(*(x+k))*p;
        }
    }

    *(x+n-1) /= *(mat+n*n-1);
    for (i=n-2; i > -1; i--) {
        sum = 0;
        for (j=i+1; j < n; j++)
            sum += *(mat+j*n+i)*(*(x+j));
        *(x+i) -=sum;
        *(x+i)/=*(mat+i*n+i);
    }
}

fact_banda(mat, banda, n_ren)
/*----------------------------------------------------------------------*
 *  NOMBRE: fact_banda                                                  *
 *                                                                      *
 *  FUNCION: Triangulariza una matriz rectangular simetrica             *
 *           almacenada en banda.                                       *
 *                                                                      *
 *  FORMA DE USO: fact_banda(mat, banda, n_ren);                        *
```

```
 *                                                                      *
 *   ARGUMENTOS: mat    : vector con los coeficientes de la matriz en  *
 *                        banda                                         *
 *               banda  : semiancho de banda de la matriz              *
 *               n_ren  : orden de la matriz                           *
 *                                                                      *
 *   RUTINAS LLAMADAS: ninguna                                          *
 *                                                                      *
 *                                                        J.F.V  1987   *
 *----------------------------------------------------------------------*/

float   *mat;

int     banda,
        n_ren;

{
    float    aux;

    int      n,
             i,i1,ia,in,i1,ij,i2,12;

    for (n=1; n <= n_ren; n++) {
        ia=(n-1)*banda;
        for (i1=2; i1 <= banda; i1++) {
            i1=ia+i1;
            in=ia+1;
            i=n+i1-1;
            ij=(i-1)*banda;
            aux=(*(mat+i1-1))/(*(mat+in-1));
            if (i <= n_ren) {
                for (i2=i1; i2 <= banda; i2++) {
                    ij++;
                    12=ia+i2;
                    *(mat+ij-1) -= aux*(*(mat+12-1));
                }
                *(mat+i1-1) = aux;
            }
    }
}


obtenx_banda(mat, x, n, banda)
/*----------------------------------------------------------------------*
 *   NOMBRE: obtenx_banda                                               *
 *                                                                      *
 *   FUNCION: Obtiene el vector solucion de un sistema de ecuaciones    *
 *             cuya matriz almacenada en banda ha sido previamente      *
 *             triangularizada.                                         *
```

```
*                                                                      *
*   FORMA DE USO: obtenx_banda(mat,x,n,banda);                         *
*                                                                      *
*   ARGUMENTOS: mat    : matriz factorizada del sistema                *
*               x      : vector de terminos independientes             *
*               n      : numero de ecuaciones del sistema              *
*               banda  : semiancho de banda de la matriz del sistema   *
*                                                                      *
*   RUTINAS LLAMADAS: ninguna                                          *
*                                                                      *
*                                                        J.F.V  1987   *
*----------------------------------------------------------------------*/

float  *mat,
       *x;

int    n,
       banda;
{

    int      ren, i, ii, li, la, ln, i2, l2;

    for (ren=1; ren <= n; ren++) {
        la=(ren-1)*banda;
        for (ii=2; ii <= banda; ii++) {
            i=ren+ii-1;
            if (i<=n) {
                li=la+ii;
                *(x+i-1) -= *(mat+li-1)*(*(x+ren-1));
            }
         }
         ln=la+1;
         *(x+ren-1) /= *(mat+ln-1);
    }

    for (ren=n; ren >= 1; ren--) {
        la=(ren-1)*banda;
        for (i2=2; i2 <= banda; i2++) {
            ii=ren+i2-1;
            if (ii <= n) {
               l2=la+i2;
               *(x+ren-1) -= *(mat+l2-1)*(*(x+ii-1));
            }
        }
    }
}
```

```
fact_perfil(mat, diag, n_ren)
/*-------------------------------------------------------------------------*
 *   NOMBRE: fact_perfil                                                   *
 *                                                                         *
 *   FUNCION: Triangulariza una matriz simetrica almacenada en perfil      *
 *                                                                         *
 *   FORMA DE USO: fact_perfil(mat, diag, n_ren);                          *
 *                                                                         *
 *   ARGUMENTOS: mat    : matriz almacenada en perfil                      *
 *               diag   : vector de acceso de la matriz                    *
 *               n_ren  : orden de la matriz                               *
 *                                                                         *
 *   RUTINAS LLAMADAS: ninguna                                             *
 *                                                                         *
 *                                                        J.F.V  1987      *
 *-------------------------------------------------------------------------*/

float   *mat;

int     *diag,
        n_ren;

{
    float     s,vs;

    int       i,j,k,l,m,n,o,ri,v,u,
              band=0;

    for (i=2; i <= n_ren; i++) {
        j = *(diag + i -1) - (*(diag + i - 2));
        if (j > 1) {
            k = l = i-j+1;
            while (band == 0) {
                m = j+1;
                if (l > 1)
                    m = *(diag+l-1) - (*(diag+l-2));
                if (m > (j-i+1))
                    m = j-i+1;
                m--;
                if (m > 0) {
                    n = *(diag+l-1) - m;
                    o = *(diag+l-1) - 1;
                    ri = *(diag+i-1)-i+1;
                    u = ri - *(diag+l-1);
                    s = 0;
                    if (l != i) {
                        for (v=n;v<=(n+m-1);v++)
                            s += *(mat+v-1) * (*(mat+u+v-1));
```

```
                        *(mat+ri-1) -= s;
                    }
                    else
                        band = 1;
                }
                l++;
            }
            for(j=n; j<=o; j++) {
                u = *(diag+k-1);
                k++;
                vs = *(mat+j-1);
                *(mat+j-1) /= *(mat+u-1);
                s += *(mat+j-1)*vs;
            }
            *(mat+ri-1) -= s;
        }
    }
}

obtenx_perfil(mat, diag, n, x)
/*-------------------------------------------------------------------*
 *   NOMBRE: obtenx_perfil                                           *
 *                                                                   *
 *   FUNCION: Obtiene el vector solucion de un sistema de ecuaciones *
 *            cuya matriz previamente fatorizada es almacenada en    *
 *            perfil.                                                *
 *                                                                   *
 *   FORMA DE USO: obtenx_perfil(mat, diag, n, x);                   *
 *                                                                   *
 *   ARGUMENTOS: mat  : matriz triangularizada almacenada en perfil  *
 *               diag : vector de acceso a la matriz del sistema     *
 *               n    : numero de ecuaciones del sistema             *
 *               x    : vector de terminos independientes y vector   *
 *                      solucion al terminar el programa             *
 *                                                                   *
 *   RUTINAS LLAMADAS: ninguna                                       *
 *                                                                   *
 *                                                    J.F.V  1987    *
 *-------------------------------------------------------------------*/

float   *mat,
        *x;

int     *diag,
        n;
{
```

```
    int     i,ni,j,k,l;
    float   s,vs;

    for (i=2; i <= n; i++) {
        j = *(diag+i-1) - (*(diag+i-2)) - 1;
        if (j > 0 ) {
            ni = *(diag+i-1)-j;
            k = i-j;
            s = 0;
            for(l=0;l<j;l++)
                s += *(mat+ni+l-1) * (*(x+k+l-1));
            *(x+i-1) -= s;
        }
    }
    for(i=1;i<n;i++)
        *(x+i-1) /= *(mat + (*(diag+i-1))-1);
    for(i=n;i>=2;i--) {
        j = *(diag+i-1) - (*(diag+i-2)) -1;
        ni = *(diag+i-2) + 1 -1;     /* se resto 1 */
        k = i - (*(diag+i-1)) + ni;
        if (j > 0 ) {
            s -= *(x+i-1);
            for (l=0;l<j;l++)
                *(x+k+l-1) += *(x+ni+l-1)*s;
        }
    }
}

genera_malla(coor, incid, indi, ren, col, mat, nse, nel,
                                        nb, xnod, ynod, incs, mats)
/*-------------------------------------------------------------------------*
 *  NOMBRE: genera_malla                                                   *
 *                                                                         *
 *  FUNCION: Genera mallas de elementos finitos rectangulares en           *
 *           forma automatica.                                             *
 *                                                                         *
 *  FORMA DE USO: genera_malla(coor, incid, indi, ren, col, mat, nse,      *
 *                                        nel, nb, xnod, ynod, incs, mats);*
 *                                                                         *
 *  ARGUMENTOS: coor : coordenadas de nodos de los superelementos          *
 *              incid: incidencias de los superelementos                   *
 *              indi : vector que indica para cada superelemento si        *
 *                     este es descrito por 4 u 8 nodos                    *
 *              ren  : vector con el numero de renglones en que se         *
 *                     dividira cada superelemento                         *
 *              col  : vector con el numero de columnas en que se          *
 *                     quiere dividir cada superelemento                   *
 *              mat  : numero de material de cada superelemento            *
```

```
 *              nse   : numero de superelementos                          *
 *              nel   : numero de los elementos que son generados         *
 *              nb    : numero de los nodos que son generados             *
 *              xnod  : coordenadas en x de los nodos generados           *
 *              ynod  : coordenadas en y de los nodos generados           *
 *              incs  : incidencias de los elementos generados            *
 *              mats  : numero de material de los elementos generados     *
 *                                                                        *
 *  RUTINAS LLAMADAS: ff28()                                              *
 *                                                                        *
 *                                                    J. F. V  1987       *
 *------------------------------------------------------------------------*/

float   *coor;
int     *incid, *indi,
        *ren, *col, *mat,
        nse, *nel, *nb;
float   *xnod, *ynod;
int     *incs, *mats;

{
    int       nse, nm, i1, j1, ni, n3, nc1, nc3, k1, l1, i, j,
              np, is, ie, ii, kn1, ks1, kn2, ks2, nrt, k,
              jk, jl, l, nrts, tr, m, il, j2, j3, j4, r, c;
    int       jt1[84], nn[441],
              nnrb[2520], comp[16];

    float     xc[441], yc[441],
              xrg[8], yrg[8], n[8];
    float     eta, si;

    comp[0]=comp[3]=comp[5]=comp[6]=
    comp[9]=comp[10]=comp[12]=comp[15]=-1;

    comp[1]=comp[2]=comp[4]=comp[7]=
    comp[8]=comp[11]=comp[13]=comp[14]=1;

    *nb = 0;          /* numero de nodos generados */
    *nel = 0;         /* numero de elementos generados */
    nse = 30;         /* numero maximo de superelementos */
    nm = 21;          /* numero max. de divisiones por superelem */

    limpia_ent(jt1, 4*nse);
    limpia_real(xc, nm*nm);
    limpia_real(yc, nm*nm);

    /* se revisa si el elemento por calcular es */
    /* contiguo a uno previamente calculado     */
```

```
 for (i1=0;i1<nse;i1++)
     for (j1=0;j1<4;j1++) {
         n1 = *(incid+i1*8+2*j1);
         if (j1 == 3)
             n3 = *(incid+i1*8);
         else
             n3 = *(incid+i1*8+2*(j1+1));
         /* endif */
         for (k1=0;k1<nse;k1++)
             if (k1 != i1)
                 for (l1=0;l1 < 4;l1++) {
                     nc1 = *(incid+8*k1+2*l1);
                     if (l1 == 3)
                         nc3 = *(incid+k1*8);
                     else
                         nc3 = *(incid+8*k1+2*(l1+1));
                     /* endif */
                     if ((n1==nc3) && (n3==nc1)) {
                         jti[4*i1+j1] = k1+1;
                     }
                 }
     }

for (i=0;i<nse;i++) {
    r = *ren;
    c = *col;
    if (*indi == 2)
        np = 4;
    else
        np = 8;
    /* endif */
    indi++;
    ren++;
    col++;
    for (j=0;j<8;j++)
        if ((np == 4) && ((j==1) || (j==3) || (j==5) || (j==7)))
            is = *(incid+8*i+j-1)-1;
            if (j != 7)
                ie = *(incid+8*i+j+1)-1;
            else
                ie = *(incid+8*i)-1;
            /* endif */
            xrg[j] = (*(coor+2*is) + (*(coor+2*ie)))/2;
            yrg[j] = (*(coor+2*is+1) + (*(coor+2*ie+1)))/2;
        }
        else {
            ii = *(incid+8*i+j)-1;
            xrg[j] = *(coor+2*ii);
            yrg[j] = *(coor+2*ii+1);
```

```
        }
        /* endif */

    /* generacion de los numeros de nodos */
    /* del superelemento                  */
    kn1 = ks1 = 1;
    kn2 = r+1;
    ks2 = c+1;
    for (j=0; j<4; j++) {
        nrt = jti[4*i+j];
        if ((nrt != 0) && (nrt<=i)) {
            for (k=0;k<4;k++)
                if ((jti[4*(nrt-1)+k]) == (i+1))
                    nrts = k;
            ki = c+1;
            if ((j==2) || (j==4))
                ki = r+1;
            jk = comp[4*j+nrts];
            if (jk == -1)
                jl = ki-1;
            else
                jl = 0;
            /* endif */
            for (l=0;l<ki;l++) {
                ii = (nrt-1)*84+nrts*21+jl;
                switch(j) {
                case 0:
                        nn[r*21+l] = nnrb[ii];
                        kn2 = r;
                        break;
                case 1:
                        nn[l*21+c] = nnrb[ii];
                        ks2 = c;
                        break;
                case 2:
                        nn[l] = nnrb[ii];
                        kn1 = 2;
                        break;
                case 3:
                        nn[21*l] = nnrb[ii];
                        ks1 = 2;
                        break;
                }
                jl += jk;
            }
        }
    }
}
if ((kn1<=kn2) && (ks1<=ks2)) {
```

```
        for (k = kn1;k<=kn2;k++) {
            eta = 1.0 - (k-1)*2.0/r;
            for (l=ks1;l<=ks2;l++) {
                si = -1.0 + ((l-1)*2.0)/c;
                /* las funciones de forma son calculadas */
                ff28(eta,si,n[0]);
                xc[21*(k-1)+l-1] = yc[21*(k-1)+l-1] = 0;

                for (m=0;m<8;m++) {
                    xc[21*(k-1)+l-1] += xrg[m]*n[m];
                    yc[21*(k-1)+l-1] += yrg[m]*n[m];
                }
                /* calculo de las coordenadas */
                /* de los nuevos nodos        */
                *(xnod+(*nb)) = xc[21*(k-1)+l-1];
                *(ynod+(*nb)) = yc[21*(k-1)+l-1];
                /* incremento en el numero de nodos */
                (*nb)++;
                nn[21*(k-1)+l-1] = *nb;
            }
        }

        /* almacenamiento de los numeros */
        /* de nodos en la frontera       */

        for (k=0;k<(c+1);k++) {
            nnrb[84*i+k] = nn[21*r+k];
            nnrb[84*i+42+k] = nn[k];
        }
        for (k=0;k<(r+1);k++) {
            nnrb[84*i+21+k] = nn[21*k+c];
            nnrb[84*i+63+k] = nn[k*21];
        }
    }
    for(il=0;il<r;il++)
        for (jl=1;jl<(c+1);jl++) {
            j1 = 21*(il+1)+jl-1;
            j2 = 21*(il+1)+jl;
            j3 = 21*il+jl;
            j4 = 21*il+jl-1;
            k = *nel*4;

            /* se dan las incidencias definitivas */
            *(incs+k) = nn[j1];
            *(incs+k+1) = nn[j2];
            *(incs+k+2) = nn[j3];
            *(incs+k+3) = nn[j4];
```

```
                    /* coloca el numero del material correspondiente */
                    *(mats+*nel) = *(mat+i);
                    /* incremento en el numero de elementos */
                    (*nel)++;
                }
        }
}

dibuja_malla(coor, incs, nn, ex, nelem)
/*-------------------------------------------------------------------*
 *   NOMBRE: dibuja_malla                                            *
 *                                                                   *
 *   FUNCION: Dibuja mallas de elementos finitos de dos dimensiones. *
 *                                                                   *
 *   FORMA DE USO: dibuja_malla(coor, incs, nn, ex, nelem);          *
 *                                                                   *
 *   ARGUMENTOS: coor  : coordenadas de los nodos                    *
 *               incs  : incidencias de los elementos                *
 *               nn    : numero de nodos por elemento                *
 *               ex    : coordenadas maximas y minimas de la malla   *
 *                       por graficar                                *
 *               nelem : numero de elementos de la malla             *
 *                                                                   *
 *   RUTINAS LLAMADAS: Se usan llamadas al graficador:               *
 *                                                                   *
 *       plot=fopen("COM1:", "w")    : inicializa puerto serial      *
 *       fprintf(plot, ";:H A ")     : inicializa graficador         *
 *       fprintf(plot, "Pi")         : selecciona pluma              *
 *       fprintf(plot, "%d, %d", a, b) : dibuja linea a el punto de las *
 *                                       coordenadas (a, b)          *
 *       fprintf(plot, "D")          : baja pluma                    *
 *       fprintf(plot, "U")          : sube pluma                    *
 *                                                                   *
 *                                                   J.F.V 1987      *
 *-------------------------------------------------------------------*/

float   *coor, *ex;
int     *incs,
         nn, nelem;
{
    FILE     *fopen(), *plot;
    int      dx, dy,
             a, b, c, d, i, j;

    /* se inicializa el puerto serial */
    plot=fopen("COM1:", "w");
```

```
    /* se calcula el tama&o del dibujo */
    dx = (*(ex+1) - *ex);
    dy = (*(ex+3) - *(ex+2));

    /* se inicializa el graficador y se selecciona pluma */
    fprintf(plot,";:H A W %d,%d,%d,%d,%d,%d,%d,%d,",
            *ex,*(ex+2),*(ex+1),*(ex+3),*ex,*(ex+2),*(ex+1),*(ex+3));
    fprintf(plot,"P1,");

    for (i=0;i<nelem;i++) {
        a=*(coor+2*(*incs)-2);
        b=*(coor+2*(*incs)-1);
        fprintf(plot,"%d,%d,D,",a,b);

        for (j=1;j<nn;j++) {
            c=*(coor+2*(*(incs+j))-2);
            d=*(coor+2*(*(incs+j))-1);
            fprintf(plot," %d,%d,",c,d);
        }
        fprintf(plot,"%d,%d,U,",a,b);
        incs+=nn;
    }
}


numera_nodos(coor,nnudos,off,escx,escy,plot)
/*-------------------------------------------------------------------*
 *  NOMBRE: numera_nodos                                             *
 *                                                                   *
 *  FUNCION: Escribe los numeros correspondientes a cada nodo.       *
 *                                                                   *
 *  FORMA DE USO: numera_nodos(coor,nnudos,off,escx,escy,plot);      *
 *                                                                   *
 *  ARGUMENTOS: coor  : coordenadas de los nodos por numerar         *
 *              nnudos: numero de nodos en la malla                  *
 *              off   : desviacion respecto al nodo donde se coloca  *
 *                      el numero del mismo                          *
 *              escx  : escalamiento en direccion x de la malla      *
 *              escy  : escalamiento en direccion y de la malla      *
 *              plot  : identificacion del archivo en que se grafica *
 *                                                                   *
 *  RUTINAS LLAMADAS: Se usan llamadas al graficador:                *
 *                                                                   *
 *       fprintf(plot,"%d,%d,Si1+%d_",x,y,i) : escribe el numero i   *
 *                                             en las coordenadas (x,y)*
 *       fprintf(plot,"D")                   : baja pluma            *
 *       fprintf(plot,"U")                   : sube pluma            *
 *                                                                   *
 *                                                    J.F.V  1987    *
 *-------------------------------------------------------------------*/
```

```
FILE     *plot;
float    *coor, *off,
         escx, escy;
int      nnudos;
{
     int     i;
     int     x, y;
     float   f = .09;

     for (i=1;i<=nnudos;i++) {
          /* calculo de las coordenadas donde se escribe el numero */
          x = (*off + (*coor-f))*escx;
          coor++;
          y = (*(off+1) + (*coor+.02))*escy;
          coor++;
          /* se escribe el numero del nodo */
          fprintf(plot, "%d, %d, D, S11+%d_, U, ", x, y, i);
     }
}


numera_elementos(coor, incs, off, escx, escy, nn, nel, plot)
/*-----------------------------------------------------------------*
 *  NOMBRE: numera_elementos                                       *
 *                                                                 *
 *  FUNCION: Escribe los numeros correspondientes a cada elemento. *
 *                                                                 *
 *  FORMA DE USO: numera_elementos(coor, incs, off, escx, escy,    *
 *                                                 nn, nel, plot); *
 *                                                                 *
 *  ARGUMENTOS: coor  : coordenadas de los nodos                   *
 *              incs  : incidencias de los elementos               *
 *              off   : desviacion respecto al nodo donde se coloca *
 *                       el numero del mismo                       *
 *              escx  : escalamiento en direccion x de la malla    *
 *              escy  : escalamiento en direccion y de la malla    *
 *              nn    : numero de nodos por elemento               *
 *              nel   : numero de elementos de la malla            *
 *              plot  : identificacion del archivo en que se grafica*
 *                                                                 *
 *  RUTINAS LLAMADAS: Se usan llamadas al graficador:              *
 *                                                                 *
 *       fprintf(plot, "%d, %d, S12+%d_", x, y, i) : escribe el numero i *
 *                                                 en las coordenadas (x,y)*
 *       fprintf(plot, "D")                         : baja pluma   *
 *       fprintf(plot, "U")                         : sube pluma   *
 *                                                                 *
 *                                                    J. F. V  1987 *
 *-----------------------------------------------------------------*/
```

```
float   *coor, *off,
        escx, escy;
int     *incs,
        nn, nel;
FILE    *plot;
{
    int     n1, n2, i;
    int     op, x, y;
    float   f = .03;

    /* se calculan las coordenadas donde se escribe el numero */
    *off -= f;
    *(off+1) -=f;
    op = nn/2;

    /* se escribe para cada elementos su numero correspondiente */
    for (i=1;i<=nel;i++) {
        n1 = (*incs-1)*2;
        n2 = (*(incs+op)-1)*2;
        x = (*off + ((*(coor+n1)+(*(coor+n2)))/2))*escx;
        y = (*(off+1) + ((*(coor+n1+1)+(*(coor+n2+1)))/2))*escy;
        fprintf(plot, "%d, %d, D, S12+%d_, U, ", x, y, i);
        incs+=nn;
    }
}


contorno (coor, vector, rango, incid, nn, ne, param, ncurva)
/*-------------------------------------------------------------------*
 *  NOMBRE: contorno                                                 *
 *                                                                   *
 *  FUNCION: Grafica curvas de contornos.                            *
 *                                                                   *
 *  FORMA DE USO: contorno(coor, vector, rango, incid, nn, ne,       *
 *                                                  param, ncurva);  *
 *                                                                   *
 *  ARGUMENTOS: coor   : coordenadas de los nodos                    *
 *              vector : vector con los valores de la variable de    *
 *                       campo                                       *
 *              rango  : valores maximo y minimo de las variables    *
 *                       de campo                                    *
 *              incid  : incidencias de los elementos                *
 *              nn     : numero de nodos del sistema                 *
 *              ne     : numero de elementos finitos                 *
 *              param  : numero de la variable de campo a dibujar    *
 *              ncurva : numero de curvas de contorno por dibujar    *
 *                                                                   *
 *  RUTINAS LLAMADAS: grafica_malla()                                *
 *                    obtengl()                                      *
```

```
 *                        dibuja_linea()
*
 *                                                                    *
 *                                                          J. F. V   1987  *
 *--------------------------------------------------------------------*/

float    *coor, *vector, *rango;
int      *incid, nn, ne, param, ncurva;

{    int     k, ic, ie, j, ipt, prim, ult,
             gl[60], gli, gl2, puntos;
     float   dif, vi, limite, radio, valor,
             vi, v2, c1, c2, linea[500];

     /* grafica la malla de elementos finitos*/
     grafica_malla(coor, incid, nn, ne, 1);
     k = 2*param;
     dif = *(rango+k) - *(rango+k+1);
     vi = *(rango+k+1) + 0.05*dif;
     limite = *(rango+k+1) + 0.95*dif;

     /* calculo de cada las coordenadas de cada curva */
     for (ic=0; ic<ncurva; ic++) {
          radio = (ic-1)/(ncurva-1);
          valor = vi + radio*(limite-vi);
          puntos = 0;

          /* para cada valor de la variable de campo se recorren   */
          /* todos los elementos localizando puntos con este valor */
          for (ie=0; ie<ne; ie++) {
               obtengl(nn, param, *(incid+ie*nn), gl);
               for(ipt=0; ipt<nn; ipt++) {
                    prim = ipt;
                    ult = ipt+1;
                    if (ipt == nn-1)
                         ult = 0;
                    gli = gl[param*prim];
                    gl2 = gl[param*ult];
                    vi = *(vector+gli);
                    v2 = *(vector+gl2);

                    /* prueba si el valor pasa entre los dos puntos */
                    if (((valor-vi)*(v2-valor)) >= 0) {
                         puntos++;
                         radio = v2 - vi;
                         if (radio != 0)
                              radio = (valor-vi)/radio;
                         for (j=0; j<2; j++) {
                              c1 = *(coor + 2*prim + j);
                              c2 = *(coor + 2*ult + j);
```

```
                        dif = c2 - c1;
                        linea[2*puntos+j] = c1 + radio*dif;
                    }
                }
            }
        }
        /* dibuja la linea una vez determinadas las */
        /* coordenadas de puntos de igual valor     */
        dibuja_linea(puntos, linea);
    }
}

maxmin(nnudos, ng, vector, rango)
/*-------------------------------------------------------------------*
 *  NOMBRE: maxmin                                                   *
 *                                                                   *
 *  FUNCION: Calcula los valores maximos y minimos de cada variable  *
 *           de campo.                                               *
 *                                                                   *
 *  FORMA DE USO: maxmin(nnudos, ng, vector, rango);                 *
 *                                                                   *
 *  ARGUMENTOS: nnudos : numero de nodos del sistema                 *
 *              ng     : numero de grados de libertad por nodo       *
 *              vector : vector que contiene los valores de cada     *
 *                       variable de campo                           *
 *              rango  : vector con los valores maximo y minimo de   *
 *                       cada variable de campo                      *
 *                                                                   *
 *  RUTINAS LLAMADAS: ninguna                                        *
 *                                                                   *
 *                                                    J. F. V  1987  *
 *-------------------------------------------------------------------*/

int     nnudos, ng;
float   *vector, *rango;
{
    int     i, j;

    /* se inicializa el vector rango */
    for(j=0; j<ng; j++) {
        *(rango+2*j) = *(vector+j);
        *(rango+2*j+1) = *(vector+j);
    }

    /* se comparan todos los valores de las variables */
    /* de campo hasta encontrar los valores extremos  */
    for(i=0; i<nnudos; i++)
        for(j=0; j<ng; j++) {
```

```
            if (*(vector+ng*i+j) < *(rango+2*j))
                *(rango+2*j) = *(vector+ng*i+j);
            if (*(vector+ng*i+j) > *(rango+2*j+1))
                *(rango+2*j+1) = *(vector+ng*i+j);
        }
}


dibuja_linea(puntos, linea)
/*-------------------------------------------------------------------------*
 *  NOMBRE: dibuja_linea                                                   *
 *                                                                         *
 *  FUNCION: Grafica una linea dadas las coordenadas de los puntos         *
 *           que la definen.                                               *
 *                                                                         *
 *  FORMA DE USO: dibuja_linea(puntos, linea);                             *
 *                                                                         *
 *  ARGUMENTOS: puntos : numero de puntos que forman la linea              *
 *              linea  : coordenadas de los puntos                         *
 *                                                                         *
 *  RUTINAS LLAMADAS: Se usan funciones para graficar en pantalla:         *
 *                                                                         *
 *       move(x, y)    : coloca la pluma en las coordenadas (x, y)         *
 *       draw(x, y, 2) : dibuja linea hasta las coordenadas (x, y)         *
 *                                                                         *
 *                                                          J. F. V  1987  *
 *-------------------------------------------------------------------------*/

int     puntos;
float   *linea;
{
    int  i, x, y;

    x = *linea;
    linea++;
    y = *linea;
    linea++;
    move(x, y);

    for (i=1;i<puntos;i++) {
        x = *linea;
        linea++;
        y = *linea;
        linea++;
        draw(x, y, 2);
    }
}
```

```
dibuja_vectores(nelem,npi,factor,ex,leer)
/*-------------------------------------------------------------------*
 *   NOMBRE: dibuja_vectores                                         *
 *                                                                   *
 *   FUNCION: Grafica vectores dados los valores en cada punto de    *
 *            integracion.                                           *
 *                                                                   *
 *   FORMA DE USO: dibuja_vectores(nelem,npi,factor,ex,leer);        *
 *                                                                   *
 *   ARGUMENTOS: nelem : numero de elementos de la malla             *
 *               npi   : numero de puntos de integracion por elemento*
 *               factor: factor de escala para graficar los modulos  *
 *                       de los vectores                             *
 *               ex    : valores extremos del dibujo               *
 *               leer  : archivo que contiene los datos por graficar *
 *                                                                   *
 *   RUTINAS LLAMADAS: Se usan llamadas al graficador:               *
 *                                                                   *
 *       plot=fopen("COM1:","w")    : inicializa puerto serial       *
 *       fprintf(plot,";:H A ")     : inicializa graficador          *
 *       fprintf(plot,"P2")         : selecciona pluma               *
 *       fprintf(plot,"%d,%d",a,b)  : dibuja linea a el punto de     *
 *                                    coordenadas (a,b)              *
 *       fprintf(plot,"D")          : baja pluma                     *
 *       fprintf(plot,"U")          : sube pluma                     *
 *                                                                   *
 *                                                    J.F.V  1987    *
 *-------------------------------------------------------------------*/

int     nelem,npi;
float   factor,*ex;
FILE    *leer;

{
    float   x,y,dx,dy,
            mod,c,s;
    int     i,ptos,a,b,ai,bi;
    double  sqrt();
    FILE    *fopen(),*plot;
    float   off[2],dxi,dyi,escx,escy;

    /* inicializa puerto serial */
    plot=fopen("COM1:","w");

    /* escala grafica a dimensiones del papel */
    dxi = (*(ex+1) - *ex);
    dyi = (*(ex+3) - *(ex+2));
    escx = 2300/dxi;
    escy = 1800/dyi;
```

```
        off[0] = -*ex/1.5;
        off[1] = -*(ex+2)/1.5;

        /* inicializa graficador y selecciona pluma */
        fprintf(plot,";:H A ");
        fprintf(plot,"P2,");

        /* comienza a imprimir vectores cuyos datos */
        /* son leidos del archivo leer              */
        ptos = npi*nelem;
        for(i=0;i<ptos;i++) {
            fscanf(leer,"%f %f %f %f",&x,&y,&dx,&dy);
            mod = sqrt(dx*dx + dy*dy);
            c = dx/mod;
            s = dy/mod;
            mod *= factor/2;
            a = (off[0] + x - mod*c)*escx;
            b = (off[1] + y - mod*s)*escy;

            /* grafica vector */
            fprintf(plot,"%d,%d,D,",a,b);

            a = (off[0] + x + mod*c)*escx;
            b = (off[1] + y + mod*s)*escy;

            /* grafica flecha del vector */
            fprintf(plot,"%d,%d,",a,b);

            mod /= 2.5;
            c += .25;
            s += .25;
            ai = a - (mod*c)*escx;
            bi = b - (mod*s)*escy;

            fprintf(plot,"%d,%d,U,",ai,bi);
            fprintf(plot,"%d,%d,D,",a,b);
            s -= .5;
            ai = a - (mod*c)*escx;
            bi = b - (mod*s)*escy;
            fprintf(plot,"%d,%d,U,",ai,bi);
        }
}
```

BIBLIOGRAFIA.

- Akin, J. E.
  Application and Implementation of Finite Element Methods.
  Academic Press, 1982, Great Britain.

- Ayala M., Gustavo ; Eudave M., Jorge ; Gómez M., Roberto
  Implantación del Método de los Elementos Finitos en
  Microcomputadoras : Un Estado del Arte.
  Memorias de V Congreso Nacional de Ingeniería Estructural
  Sociedad Mexicana de Ingenería Estructural, A.C.
  Veracruz, México 1986.

- Bathe, Klaus Jurgen ; Wilson, Edward L.
  Numerical Methods in Finite Element Analysis.
  Prentice Hall Inc., 1976, E.U.A.

- Cervantes Beltran, Ramón ; Porras Silva, Victor
  Introducción al Método del Elemento Finito.
  D.E.P.F.I., U.N.A.M., 1983, México

- Dodds, R. H. ; Lopez, L. A. ; Pecknold, D. A.
Numerical and Software Requirements for General Nonlinear Finite Element Analysis.
University of Illinois, 1978, E.U.A.

- Dunder, Vera
Software Engineering Modules.
Computers & Structures Vol.10
Pergamon Press, Ltd., 1979, Great Britain

- Durocher, Lawrence L. ; Gasper, Andrew
A Versatile Two - Dimensional Mesh Generator with Automatic Bandwidth Reduction.
Computers & Structures Vol.10
Pergamon Press, Ltd., 1979, Great Britain

- Felippa, Carlos A.
Database Management in Scientific Computing - I
General Description.
Computers & Structures Vol.10
Pergamon Press, Ltd., 1979, Great Britain

- Fong, Henry H.
Interactive Graphics and Commercial Finite Element Codes.
Mechanical Engineering, Vol. 106 # 6, Jun 1984, E.U.A.

- Harrington, Steven
Computer Graphics : a Programming Approach.
Mc. Graw Hill, 1983, E.U.A

- Hughes, Thomas J. P. ; Prevost, Jean-Herve
Dirt II - A Nonlinear Quasi-Static Finite Element
Analysis Program.
California Institute of Technology, 1979, E.U.A.

- Iwaki, Takashi ; Maeda, Akihiko ; Ishii, Tadashi
Misa - A General Purpose FEM Program.
Computers & Structures, Vol. 10
Pergamon Press Ltd., 1979, Great Britain

- Jennings, Alan
Matrix Computation for Engineers and Scientists.
John Wiley & Sons, 1977, E.U.A.

- Krick, Edward V.
An Introduction to Engineering & Engineering Design.
John Wiley & Sons, Inc. 1967, E.U.A.

- Kernighan, Brian W. ; Ritchie, Dennis M.
The C Programming Language.
Prentice Hall, 1978, E.U.A.

- Mason, Philip ; Gregory, David ; Balderes, Theodore
Towards a Realistic Structural Analysis/Design System.
Computers & Structures Vol.10
Pergamon Press, Ltd., 1979, Great Britain

- Merrifield, B. C.
Fortran Subroutines for Finite Element Analysis.
Royal Aircraft Establishment, 1971, E.U.A.

- Noor, Ahmed K. ; Lambiotte, Jules
Finite Element Dynamic Analiysis on CDC Star-100 Computer
Computers & Structures Vol.10
Pergamon Press, Ltd., 1979, Great Britain

- Perronet, A.
Les Methodes de Resolution des Systems Lineaires.
Leur Technique de Stockage.
Universite Pierre et Marie Curie, France

- Rajan, S. D. ; Bhatti, M. A.
Data Management in FEM - Based Optimization Software.
Computers & Structures Vol.16
Pergamon Press, Ltd., 1979, Great Britain

- Rice, John R.
Numerical Methods, Software, and Analysis.
Mc. Graw Hill, 1983, E.U.A.

- Shoppee, G. J. V.; Jeanes, P. J. ; Griffin, T. B.
A Finite Element Modelling and Analysis Language for Engineers - The Program FEMALE.
Advances in Engineering Software, Vol.1 , No.1, 1978, Great Britain.

- Thomson, William.
Teoría de Vibraciones.
Prentice/Hall Int., 1981, Colombia.

- Schremm, E.
Trends and Aspects of the Development of Large Finite. Element Software Systems
Computers & Structures Vol.10
Pergamon Press, Ltd., 1979, Great Britain.

- Tong, Pin ; Rossettos, John N.
Finite Element Method.
Basic Technique and Implementation.
The MIT Press, 1982, E.U.A.

- Zienkiewicz, O. C.
The Finite Element Method in Engineering Science.
Mc. Graw Hill, 1971, England.

- Zienkiewicz, O. C. ; Phillips, D. V.
An Automatic Mesh Generation Scheme for Plane and Curved Surfaces by Isoparametric Coordinates.
Int. Journal for Numerical Methods in Engineering
Vol. 3 , 1971, Great Britain.