42
2 Ejem

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

ANALISIS DE LOS METODOS FRONTAL Y MULTIFRONTAL

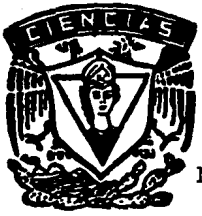
T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I O

P R E S E N T A:

ARTURO LORENZO VALDES



México, D. F. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

M. EN C. VIRGINIA ABRIN BATULE
Jefe de la División de Estudios Profesionales
Facultad de Ciencias
P r e s e n t e

Los abajo firmantes, comunicamos a Usted, que habiendo revisado el trabajo de Tesis que realiz(ó)ron el pasante(s) LORENZO VALDES ARTURO

con número de cuenta 9052137-9 con el Título:

"ANALISIS DE LOS METODOS FRONTAL Y MULTIFRONTAL"

Otorgamos nuestro Voto Aprobatorio y consideramos que a la brevedad deberá presentar su Examen Profesional para obtener el título de ACTUARIO

GRADO	NOMBRE(S)	APELLIDOS COMPLETOS	FIRMA
M. EN C.	VIRGINIA	ABRIN BATULE	Virginia Abrín Batule
Director de Tesis			
M. EN C.	JOSE	GUERRERO GRAJEDA	
MAT.	BENITO FERNANDO	MARTINEZ SALGADO	
MAT.	OSCAR	DAVALOS OROZCO	
Suplente			
ACT.	DAVID	LOPEZ SERVIN	
Suplente			

A mi mama

Por quererme mucho, ayudarme incondicionalmente en todo momento, por su apoyo.

A mi padre

Por sus enseñanzas.

A Elvia

Por brindarme su amor, apoyo y comprensión siempre. Por llenar de felicidad mi vida. Espero que estes orgullosa de mi.

Te amo

A mis hermanos Aura Sylvia, Francisco, Patricia, Maria del Carmen y Luis Ernesto

Por compartir mi niñez y porque los quiero.

A mis amigos

Por cada momento que hemos compartido y por creer en mi.

A Virginia

Por guiarme, explicarme y ayudarme en todo momento en la realizacion de esta tesis. Por su ejemplo.

INDICE

INTRODUCCION

La solución numérica de los sistemas de ecuaciones es uno de los temas que mayor atención ha tenido por parte de los expertos en computación numérica, ya que han influenciado en el diseño de muchas arquitecturas computacionales, compiladores, etc.

Dentro de la gran variedad de sistemas de ecuaciones, estudiaremos los que tienen las siguientes características:

- La matriz sea simétrica
- La matriz sea definida positiva
- La matriz sea rala, es decir, que tiene muchos elementos iguales a cero.

Los pasos para encontrar una solución a los sistemas con las características anteriores se pueden resumir en: dar un ordenamiento, obtener la factorización simbólica y la factorización numérica y encontrar una solución.

En el capítulo 1, se hace un repaso de los temas relacionados como la eliminación Gaussiana, factorización de Cholesky, teoría de gráficas y árboles de eliminación, necesarios para el proceso de eliminación y ensamblaje en el método multifrontal, así como en los reordenamientos equivalentes.

Al final de este capítulo se muestra un algoritmo para obtener la factorización simbólica.

En el segundo capítulo se estudian los ordenamientos para reducir el llenado. Aquí se explican las características de una matriz y de su gráfica asociada para no tener llenado (matriz de eliminación perfecta).

Se presentan dos formas de ordenamiento de matrices para reducir el llenado. El ordenamiento de grado mínimo trabaja sobre las columnas y los renglones de una matriz de acuerdo al grado de cada vértice en la gráfica asociada y el de deficiencia mínima que lo realiza con la deficiencia de cada nodo.

En este capítulo también se definen las gráficas triangularizadas, que son gráficas asociadas a la matriz y que no producen llenado al ser factorizadas.

La última sección del segundo capítulo trata de los ordenamientos equivalentes, es decir, aquellos que tienen el mismo número de llenado pero con su respectivo árbol de eliminación.

El siguiente capítulo trata de la factorización numérica. Se presentan los métodos frontal, desarrollado por Irons en 1970, y el multifrontal, propuesto por Duff y Reid.

Contiene además, una sección donde se presenta el postordenamiento y que incluye un algoritmo con el que se decide el orden para procesar las columnas en el método multifrontal.

Por último se agrega un apéndice en el que se encuentra la instrumentación computacional en lenguaje C de los algoritmos expuestos en este trabajo y con la cual se prueban los ejemplos.

CAPITULO I

SISTEMAS RALOS Y GRAFICAS

1.1 Eliminación Gaussiana

Supongamos que tenemos un sistema de ecuaciones Ax=b donde A es una matriz de nxn.

Una manera simple para resolver este sistema es usar el método de eliminación Gaussiana que se describe a continuación.

El objetivo de la eliminación Gaussiana es obtener una matriz U equivalente a A tal que U sea triangular. Esto se consigue realizando operaciones elementales, tales como intercambiar renglones, multiplicar un renglón por un escalar, etc., en A.

Ejemplificando lo anterior:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix}$$

donde:

$$a_{12}^{(2)} = a_{12} - (a_{11}/a_{11})\, a_{12}$$

$$a_{13}^{(2)} = a_{13} - (a_{11}/a_{11})\, a_{13}$$

$$b_1^{(2)} = b_1 - (a_{11}/a_{11})\, b_1 \qquad \text{con } i \in \{1,2\}$$

(esto se puede si $a_{11} \neq 0$, si no lo es se puede hacer una permutación de renglones).

Luego:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \end{pmatrix}$$

donde:

$$a_{33}^{(3)} = a_{33}^{(2)} - (a_{32}^{(2)}/a_{22}^{(2)})\, a_{23}^{(2)}$$

y

$$b_3^{(3)} = b_3^{(2)} - (a_{32}^{(2)}/a_{22}^{(2)})\, b_2^{(2)}$$

si $a_{22}^{(2)} \neq 0$, si no se puede hacer una permutación de renglones.

Este proceso consiste en crear ceros debajo de la diagonal, de esta forma tenemos que para $k=1,2,\ldots,n-1$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)}/a_{kk}^{(k)})\, a_{kj}^{(k)}, \quad i,j>k$$

y

$$b_i^{(k+1)}=b_i^{(k)}-(a_{ik}^{(k)}/a_{kk}^{(k)})b_k^{(k)} \quad i>k$$

donde

$$a_{ij}^{(1)}=a_{ij} \quad i,j=1,\ldots,n$$

aquí cada $a_{kk}^{(k)}\neq 0$, $k=1,\ldots,n$ son llamados los pivotes en la eliminación Gaussiana.

Entonces el sistema Ax=b se transforma en el sistema Ux=c el que se resuelve como sigue:

$$x_n=\frac{c_n}{U_{nn}}$$

$$x_k=(c_k-\sum_{j=k+1}^{n} U_{kj}x_j)U_{kk} \quad k=n-1,n-2,\ldots,1$$

1.2 Factorización de matrices

El método de eliminación Gaussiana, también se puede expresar en una factorización de matrices, donde A=LU y L es una matriz triangular inferior y U una matriz triangular

superior.[1]

así

$$M_1 = \begin{pmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ -m_{31} & 0 & 1 & & \\ \cdot & & & \cdot & \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ -m_{n1} & 0 & \cdots & & 1 \end{pmatrix}$$

donde

$$m_{i1} = \frac{a_{i1}}{a_{11}^{(1)}}$$

entonces

$$M_1 A = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & & a_{2n}^{(2)} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & & \cdot \\ 0 & a_{n2}^{(2)} & \cdots & & a_{nn}^{(2)} \end{pmatrix}$$

y

$$M_1^{-1} = \begin{pmatrix} 1 & & & & \\ m_{21} & 1 & & & \\ m_{31} & 0 & 1 & & \\ \cdot & & & \cdot & \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ m_{n1} & 0 & \cdots & & 1 \end{pmatrix}$$

tenemos que $M_1^{-1} M_1 = I$.

---

[1] Una matriz A es triangular inferior si $a_{ij} = 0$ para $i<j$, así mismo es triangular superior si $a_{ij} = 0$ para $i>j$.

Generalizando

$$M_k = \begin{pmatrix} 1 & & & & & & \\ 0 & 1 & & & & & \\ \cdot & & \cdot & & & & \\ \cdot & & & \cdot & & & \\ \cdot & & & & 1 & & \\ 0 & & & & -m_{k+1,k} & 1 & \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ 0 & & & & -m_{n,k} & 0 \ . \ . \ . & 1 \end{pmatrix}$$

donde

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{nn}^{(k)}}$$

entonces

$$M_k M_{k-1} \cdots M_2 M_1 A = \begin{pmatrix} a_{11}^{(1)} & \cdot & & \cdot & \cdot & a_{1n}^{(1)} \\ & \cdot & & & & \\ & & \cdot & & & \\ & & & a_{kk}^{(k)} & a_{k,k+1}^{(k)} \ . \ . \ . & a_{kn}^{(k)} \\ & & & 0 & a_{k+1,k+1}^{(k+1)} \ . \ . \ . & a_{k+1,n}^{(2)} \\ & & & \cdot & \cdot \quad \cdot & \cdot \\ & & & \cdot & \cdot \quad \cdot & \cdot \\ & & & \cdot & \cdot \quad \cdot & \cdot \\ & & & 0 & a_{n,k+1}^{(k+1)} \ . \ . \ . & a_{nn}^{(k+1)} \end{pmatrix}$$

y

$$M_k^{-1} = \begin{pmatrix} 1 & & & & & & \\ 0 & 1 & & & & & \\ \cdot & & \cdot & & & & \\ \cdot & & & \cdot & & & \\ \cdot & & & & 1 & & \\ 0 & & & & m_{k+1,k} & 1 & \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ 0 & & & & m_{n,k} & 0 \ . \ . \ . & 1 \end{pmatrix}$$

por lo que tenemos que

$$M_{n-1}M_{n-2}\ldots M_1 A = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ & & \cdot & \cdot \\ & & \cdot & \cdot \\ & 0 & & \cdot \\ & & & a_{nn}^{(2)} \end{pmatrix}$$

y como

$$(M_{n-1}\ldots M_1)^{-1} = M_1^{-1}\ldots M_{n-1}^{-1}$$

entonces

$$M_1^{-1}M_2^{-1}\ldots M_{n-1}^{-1} = \begin{pmatrix} 1 & & & & \\ m_{21} & 1 & & & \\ m_{31} & m_{32} & 1 & & \\ \cdot & & \cdot & & \\ \cdot & & & \cdot & \\ \cdot & & & & \cdot \\ m_{n1} & m_{n2} & \ldots & m_{n,n-1} & 1 \end{pmatrix} = L$$

y como

$$M_{n-1}...M_1A = U$$

entonces

$$A = M_1^{-1}...M_{n-1}^{-1}\ U = LU$$

y el sistema Ax=b se puede expresar como LUx=b y se puede resolver encontrando las soluciones a los sistemas

$$Ly=b$$

$$Ux=y$$

ya que $Ly=L(Ux)=(LU)x=Ax=b$

Las matrices que nos interesan en este trabajo son las matrices simétricas y definidas positivas que se pueden trabajar en una forma especial.

1.3 Factorización de Choleski

Una matriz simétrica A es definida positiva si $x^tAx>0$ para toda x distinta de cero. Todas las entradas de la diagonal son positivas ya que $e_i^tAe_i=a_{ii}>0$ donde e_i es el vector que tiene 1 en la i-ésima entrada y 0 en las demás. Una matriz simétrica A definida positiva, puede ser factorizada en matrices triangulares en lo que se conoce como factorización de Choleski que sigue del siguiente teorema.

Teorema 1.3.1

Si A es una matriz de nxn simétrica y definida positiva, entonces existe una única factorización triangular LL^t de A, donde L es una matriz triangular inferior con entradas positivas en la diagonal.

Demostración

Por inducción sobre el orden de la matriz A.

Para n=1 sabemos que por ser definida positiva $xa_{11}x=a_{11}x^2>0 \rightarrow a_{11}>0$ por lo tanto $L=\sqrt{a_{11}}$ $(\sqrt{a_{11}}>0)$ y $a_{11}=\sqrt{a_{11}}\sqrt{a_{11}}$ por lo tanto existe y es única.

Lo suponemos cierto para n-1.

Sea A una matriz de nxn simétrica y definida positiva. Descomponemos A de la siguiente manera:

$$A=\begin{pmatrix} d & v^t \\ v & H \end{pmatrix}$$

donde d es un escalar positivo y H es una matriz de $(n-1)x(n-1)$; entonces

$$A=\begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I_{n-1} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & H^1 \end{pmatrix}\begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I_{n-1} \end{pmatrix}$$

donde $H^1=H-vv^t/d$

H^1 es simétrica porque H y vv^t/d son simétricas.

H^1 es definida positiva ya que para toda x distinta de 0, con $x \in \mathbb{R}^{n-1}$

$$(-x^t v/d, x^t)\begin{pmatrix} d & v^t \\ v & H \end{pmatrix}\begin{pmatrix} -x^t v/d \\ x \end{pmatrix} = x^t(H - vv^t/d)x =$$

$$= x^t H^1 x > 0$$

Por hipótesis de inducción H^1 tiene una única factorización $L_{H^1}L_{H^1}{}^t$ con diagonal positiva, entonces

$$A = \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I_{n-1} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & L_{H1} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & L_{H1}{}^t \end{pmatrix}\begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I_{n-1} \end{pmatrix} =$$

$$= \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & L_{H1} \end{pmatrix}\begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & L_{H1}{}^t \end{pmatrix} = LL^t$$

donde L es triangular inferior con diagonal positiva ($\sqrt{d} > 0$) y única, ya que L_{H1} es única, $\sqrt{d} > 0$ es única, y $v/\sqrt{d}$ es única ya que de ahí se particionó a A.

El método de Choleski es un caso particular de la eliminación Gaussiana. Ya que el primero se aplica a matrices simétricas y definidas positivas. La factorización correspondiente al método de Choleski se puede llevar a cabo de distintas

maneras, dependiendo el orden en el que las entradas de las matrices pueden ser accesadas y calculadas.
Los párrafos siguientes muestran una breve descripción de cómo llevar a cabo dicho proceso.

1.- Por renglones.

Aquí cada paso calcula un renglón del factor, resolviendo un sistema triangular, esto es:

Para toda i=1,...,n resolver

$$\begin{pmatrix} l_{1,1} & & & 0 \\ \cdot & \cdot & & \\ \cdot & & \cdot & \\ \cdot & & & \cdot \\ l_{i-1,1} & \cdot & \cdot & l_{i-1,i-1} \end{pmatrix} \begin{pmatrix} l_{i,1} \\ \cdot \\ \cdot \\ \cdot \\ l_{i,i-1} \end{pmatrix} = \begin{pmatrix} a_{i,1} \\ \cdot \\ \cdot \\ \cdot \\ a_{i,i-1} \end{pmatrix}$$

calculando

$$l_{i,i}=\left(a_{i,i}-\sum_{k=1}^{i-1} l_{i,k}^{2}\right)^{\frac{1}{2}}$$

Ejemplo

Sea

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 43 \end{pmatrix}$$

Primer paso

$$l_{11}=\sqrt{a_{11}} \rightarrow l_{11}=1$$

Segundo paso

$$l_{11}l_{21}=a_{21} \rightarrow l_{21}=2$$

$$l_{22}=(a_{22}-l_{21}^2)^{\frac{1}{2}} \rightarrow l_{22}=4$$

Tercer paso

$$\begin{pmatrix} l_{11} & \\ l_{21} & l_{22} \end{pmatrix}\begin{pmatrix} l_{31} \\ l_{32} \end{pmatrix}=\begin{pmatrix} a_{31} \\ a_{32} \end{pmatrix} \rightarrow \begin{matrix} l_{31}=3 \\ l_{32}=5 \end{matrix}$$

$$l_{33}=(a_{33}-l_{31}^2-l_{32}^2)^{\frac{1}{2}} \rightarrow l_{33}=3$$

Por lo tanto

$$L=\begin{pmatrix} 1 & & \\ 2 & 4 & \\ 3 & 5 & 3 \end{pmatrix}$$

2.- Por columnas.

Calcula el factor L columna por columna, aplicando todas las actualizaciones de las columnas previas. El algoritmo es el siguiente:

Para toda j=1,...,n encuentra

$$l_{j,j}=\left(a_{j,j}-\sum_{k=1}^{j-1} l_{j,k}^2\right)^{\frac{1}{2}}$$

y para toda i=j+1, j+2, ...n

$$l_{i,j}=\left(a_{i,j}-\sum_{k=1}^{j-1} l_{i,k}\, l_{j,k}\right)/l_{j,j}$$

Ejemplo

Con la misma matriz del ejemplo anterior

Primer paso

$$l_{11}=\sqrt{a_{11}} \rightarrow l_{11}=1$$

$$l_{21}=a_{21}/l_{11} \rightarrow l_{21}=2$$

$$l_{31}=a_{31}/l_{11} \rightarrow l_{31}=3$$

Segundo paso

$$l_{22}=(a_{22}-l_{21}^{2})^{\frac{1}{2}} \rightarrow l_{22}=4$$

$$l_{32}=(a_{32}-l_{31}\, l_{21})/l_{22} \rightarrow l_{32}=5$$

Tercer paso

$$l_{33}=(a_{33}-l_{31}^{2}-l_{32}^{2})^{\frac{1}{2}} \rightarrow l_{33}=3$$

Por lo tanto

$$L=\begin{pmatrix}1 & & \\ 2 & 4 & \\ 3 & 5 & 3\end{pmatrix}$$

3.- Por submatrices.

Inicialmente se encuentran las columnas y lo que queda de la matriz se va actualizando para volver a ser factorizado.

Este esquema se puede describir como sigue:

$$A = A_0 = H_0 = \begin{pmatrix} d & v^t \\ v & H \end{pmatrix} =$$

$$= \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & H - vv^t/d \end{pmatrix} \begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I_{n-1} \end{pmatrix} =$$

$$= L_1 \begin{pmatrix} 1 & 0 \\ 0 & H_1 \end{pmatrix} L_1^t = L_1 A_1 L_1^t$$

donde

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & H_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & d_1 & v_1^t \\ 0 & v_1 & H_2 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sqrt{d_1} & 0 \\ 0 & v_1/\sqrt{d_1} & I_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & H_2 - v_1 v_1^t/d_1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sqrt{d_1} & v_1^t/\sqrt{d_1} \\ 0 & 0 & I_{n-2} \end{pmatrix} =$$

$$= L_2 A_2 L_2^t$$

de esta forma encontramos $A_3, A_4, \ldots, A_{n-1} = L_n I_n L_n^t$

en donde después de n pasos de aplicar el algoritmo tenemos:

$$A = L_1 L_2 \ldots L_n L_n^t \ldots L_2^t L_1^t = L L^t$$

Ejemplo

Con la misma matriz empleada para los casos anteriores:

$$\begin{pmatrix}1&0&0\\2&1&0\\3&0&0\end{pmatrix}\begin{pmatrix}1&0&0\\0&16&20\\0&20&34\end{pmatrix}\begin{pmatrix}1&2&3\\0&1&0\\0&0&1\end{pmatrix} =$$

$$= \begin{pmatrix}1&0&0\\2&4&0\\3&5&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&9\end{pmatrix}\begin{pmatrix}1&2&3\\0&4&5\\0&0&1\end{pmatrix} =$$

$$= \begin{pmatrix}1&0&0\\2&4&0\\3&5&3\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&2&3\\0&4&5\\0&0&3\end{pmatrix} =$$

$$= \begin{pmatrix}1&0&0\\2&4&0\\3&5&3\end{pmatrix}\begin{pmatrix}1&2&3\\0&4&5\\0&0&3\end{pmatrix} = LL^t$$

1.4 Matrices Ralas

Dado un sistema Ax=b, si A tiene un porcentaje pequeño de elementos distintos de cero, se dice es rala. Para ser más precisos, una matriz A de orden n, con n suficientemente grande, es rala si tiene de dos a diez elementos distintos de

cero en cada renglón.

Si al sistema de ecuaciones Ax=b donde A es una matriz triangular, definida positiva, rala y de orden n, aplicamos el método de Cholesky, se factoriza A en:

$$A=LL^t$$

donde L es triangular inferior cuyos elementos de la diagonal mayores son mayores a cero.

Definimos la matriz simétrica $F=L+L^t$ que será la matriz de llenado. La llamaremos así porque puede pasar que si un elemento $a_{ij} \in A$ es igual a cero, el correspondiente $f_{ij} \in F$ sea distinto de cero.

Volviendo al sistema Ax=b, donde A es una matriz rala, es posible almacenarla sin los ceros, siempre y cuando se puedan localizar los elementos de la matriz por medio de subíndices.

Este trabajo sólo se considerarán matrices simétricas y definidas positivas, en las que sólo es necesario almacenar la diagonal y los elementos bajo la misma. A continuación se mencionarán dos métodos de almacenamiento.

Almacenamiento por renglones

Este método consiste en colocar los elementos distintos de cero de la matriz en un arreglo primario por renglones y en orden de aparición como un arreglo secundario. Este último debe ser del mismo tamaño que el primero e indica en qué columna se encuentra la entrada equivalente en el arreglo

primario. Un arreglo de índices de tamaño n es el que indica dónde comienza cada renglón.

Ejemplo

$$A = \begin{pmatrix} 1 & 0 & 6 & 0 & 0 \\ 0 & 2 & 7 & 8 & 0 \\ 6 & 7 & 3 & 0 & 0 \\ 0 & 8 & 0 & 4 & 9 \\ 0 & 0 & 0 & 9 & 5 \end{pmatrix}.$$

Arreglo primario: 1,2,6,7,3,8,4,9,5

Arreglo secundario: 1,2,1,2,3,2,4,4,5

Arreglo de índices: 1,2,3,6,8

Almacenamiento por columnas

Es similar al anterior, solo que en este caso el arreglo primario se hace por columnas mientras que el secundario y el de índices muestran la localización en el renglón y dónde comienza cada uno de éstos.

Basados en el ejemplo anterior, tendremos los siguientes arreglos:

Arreglo primario: 1,6,2,7,8,3,4,9,5

Arreglo secundario: 1,3,2,3,4,3,4,5,5

Arreglo de índices 1,3,6,7,9

Una manera de trabajar con matrices simétricas consiste en asociar a una matriz una "gráfica", ya que como veremos se puede hacer caso omiso del valor numérico de cada entrada y

trabajar con permutaciones de números naturales 1,2,...,n que se definirán y estudiarán en la siguiente sección.

1.5 Gráficas

"Una *gráfica* G=(V(G),A(G)) consiste en un conjunto finito, no vacío, de objetos llamados VERTICES (NODOS) y un conjunto de pares no ordenado de vértices llamados ARISTAS (RAMAS, LINEAS). Los vértices de G se denotan V(G) y sus aristas A(G)".[2]

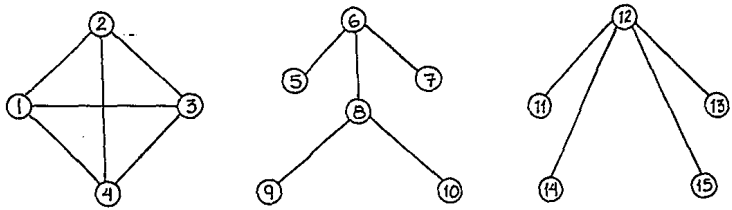


Figura 1

En la figura 1 se muestra una gráfica donde $V(G)=\{1,2,\ldots,15\}$ y

$$A(G) = \left\{ \begin{matrix} \{1,2\} & \{1,4\} & \{2,4\} & \{5,6\} & \{6,8\} & \{8,10\} & \{12,13\} & \{12,15\} \\ \{1,3\} & \{2,3\} & \{3,4\} & \{6,7\} & \{8,9\} & \{11,12\} & \{12,14\} & \end{matrix} \right\}.$$

Si $\{u,v\} \in A(G)$ y u=v entonces se dice que la arista es un *lazo*.

---

[2] CURCO, María del Carmen, Una Introducción a la Teoría de Gráficas, p. 7-8.

A dos o más aristas con los mismos extremos se les llama *aristas múltiples*. En este trabajo sólo nos interesan las gráficas donde no aparecen lazos ni aristas múltiples y que llamaremos *gráficas simples*.

Una *subgráfica* de G es una gráfica H tal que $V(H) \subseteq V(G)$ y $A(H) \subseteq A(G)$. En la figura 1 los vértices 1,2,3,4 y las aristas entre ellos, forman una subgráfica.

Si tenemos dos vértices u,v tal que $\{u,v\} \in A(G)$ entonces se dice que u y v son *adyacentes*. El vértice 5 es adyacente al vértice 6 en nuestra gráfica.

De igual forma, si tenemos un conjunto de vértices W entonces el conjunto de vértices adyacentes de W es:

$$Adj(W) = \{ u \in V-W / \exists\, v \in W \text{ y } \{u,v\} \in A \}.$$

Una gráfica G es *completa* si todo par de vértices es adyacente. Una subgráfica completa de G se dice que es un *clan*. La subgráfica con los vértices 1,2,3,4 representa una gráfica completa, es decir es un clan de la figura 1.

Si $\{u,v\} \in A(G)$ entonces se dice que la arista *incide* en u y en v con esto se define que para todo $v \in V(G)$ el *grado* (o valencia) de v en G es el número de aristas de G que inciden en v y se denota por $g_G(v)$. El grado del vértice 1 es 3.

El *grado mínimo* de una gráfica G es el menor de los grados de sus vértices y se denota por $\delta(G)$. Así mismo, se define el *grado máximo* denotado por $\Delta(G)$. En nuestro ejemplo $\delta(G)=1$ y $\Delta(G)=3$.

"Un *camino* $v_0, a_0, v_1, a_1, \ldots, a_{n-1}, v_n$ en G es una sucesión alternada de vértices y aristas de G tal que $a_i=[v_i, v_{i+1}]$ $i=0,1,\ldots,n-1$".[9]

La *longitud* de un camino es el número de aristas que lo forman. En la figura 1, C=(1,2,4,3,2) es un camino de longitud 4.

Un *paseo* es un camino en el que no se repiten aristas.

Una *trayectoria* es un camino en el que no se repiten vértices.

Un *camino (paseo) cerrado* es donde el vértice final coincide con el inicial.

Un *ciclo* es un camino cerrado $C=(v_0, v_1, \ldots, v_n, v_0)$ donde $v_i \neq v_j \ \forall \ i \neq j$ i,j = 1,2, ...,n. C=(1,2,3,4,1) forman un ciclo.

Una gráfica G es *conexa* si y sólo si para cualquier par de vértices de G existe una trayectoria que los une.

Un *árbol* es una gráfica conexa y acíclica.

La subgráfica con V(G)=(5,5,7,8,9,10) y las aristas que unen los vértices en V(G) es un árbol.

Un *bosque* es una gráfica acíclica. De esto se deduce que un árbol es un bosque conexo. Entonces decimos que una *componente conexa* en una gráfica, es una subgráfica conexa G(X,A(X)) donde $X \subseteq V(G)$, $A(X) \subseteq A(G)$ y $(x,y) \in A(X)$ si $x,y \in X$ y no existe (x,y) en A(G) tal que $x \in X$ y $y \in V(G)-X$.

En la figura la G(V(G),A(G)) donde V(G)=(5,6,...,15) es un

---

[9] Ibid. p. 17

bosque que consta de dos componentes conexas.

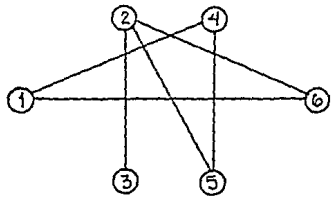
Un *separador* es un conjunto $S \subset V(G)$ separa a dos vértices u y $v \in V(G)$ donde u no es adyacente a v y si u y v están en distintas componentes de G-S.

En el ejemplo, un separador para 5 y 7, es 6.

Sea A una matriz de nxn simétrica a la cual le asociamos la gráfica G(V(G),A(G)), donde V(G)=(1,2,...,n) y $(x,y) \in A(G)$ si $a_{xy} \neq 0$.

Ejemplo

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{pmatrix} X & & X & & X \\ & X & X & X & X \\ & X & X & & \\ X & & & X & X \\ & X & & X & X \\ X & X & & & X \end{pmatrix}$$



1.6 Arboles de Eliminación

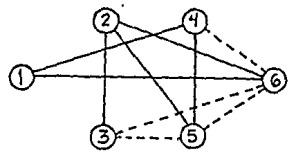
Sea A una matriz de nxn simétrica y definida positiva con factor de Choleski L.

El *árbol de eliminación* de la matriz A es una gráfica G(V(G),A(G)) donde $V(G) = \{1,2,\ldots,n\}$ y tal que $(i,j) \in A$ si $i = \min\{ k > j \ / \ l_{kj} \neq 0 \}$.

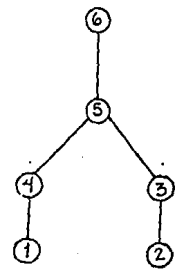
En este caso, i es el padre de j denotándolo i=p(j), y j es hijo de i.

Ejemplo

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} X & & X & & & X \\ & X & X & & X & X \\ & X & X & & & \\ X & & & X & X & \\ & X & & X & X & \\ X & X & & & & X \end{pmatrix} \end{array}$$



G(F)



T(A)

$$F = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} X & & X & & & X \\ & X & X & & X & X \\ & X & X & & \otimes & \otimes \\ X & & & X & X & \otimes \\ & X & \otimes & X & X & \otimes \\ X & X & \otimes & \otimes & \otimes & X \end{pmatrix} \end{array}$$

donde $F=L+L^t$, $\otimes$ y las líneas punteadas representan un llenado en la matriz y la gráfica respectivamente.

Si A es *reducible*, es decir que existe una matriz de permutación P tal que

$$P^t A P = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}$$

entonces el árbol de eliminación es un bosque.
Si A es Irreducible (que no es reducible) entonces es un árbol.

El árbol de eliminación de una matriz A se denota por T(A). Estas estructuras tienen varias propiedades de las cuales se presentarán algunas útiles para este trabajo.

Sea A una matriz de nxn, simétrica, definida positiva e irreducible con su árbol de eliminación T(A).

Proposición 1.6.1

Si el nodo i es un ancestro de j en T(A) entonces i>j.

Demostración

Sea $j, i_1, \ldots, i_r, i$ la trayectoria de i a j en T(A). Como i es ancestro de j entonces i_1 es el padre de j, por lo tanto:

$$i_1=\min\{k>j / l_{kj}\neq 0\} \Rightarrow i_1>j$$

$$i_2=\min\{k>i_1 / l_{k,i_1}\neq 0\} \Rightarrow i_2>i_1$$

$$\vdots$$

$$i_r=\min\{k>i_{r-1} / l_{k,i_{r-1}}\neq 0\} \Rightarrow i_r>i_{r-1}$$

$$i=\min\{k>i_r / l_{k,i_r}\neq 0\} \Rightarrow i>i_r$$

entonces $i>i_r>i_{r-1}>\ldots>i_2>i_1>j$ por lo que $i>j$.

Usaremos T[x] para representar el subárbol de T(A) enraizado en el nodo x, es decir, T[x] es el conjunto de descendentes de x incluyendo a x. Con esto tenemos que T[n] = T(A).

Entre las propiedades que cumple la gráfica asociada a F[4] que tienen que ver con T(A).

Proposición 1.6.2

Sea A una matriz rala, simétrica y definida positiva y sea i>j. Si existe un camino de j a i en G(F) a través de vértices más pequeños que j e i entonces $l_{ij}\neq 0$.

Demostración

Sea $C=j,v_1,\ldots,v_t,i$ un camino tal que $v_k<j<i$ para toda k=1,2,....t.

Sea v_r el menor de las v_k en C. Entonces $l_{v_{r-1},v_r}\neq 0$ y $l_{v_{r+1},v_r}\neq 0$.

De la manera de definir el llenado, si $a_{v_{r-1},v_{r+1}}\neq 0$ entonces $l_{v_{r-1},v_{r+1}}\neq 0$, si no entonces $l_{v_{r-1},v_{r+1}}=(-\sum_{i=1}^{v_{r+1}-1} l_{v_{r-1},i}l_{v_{r+1},i})/l_{v_{r+1},v_{r+1}}=$

$$=(-l_{v_{r-1},v_r}l_{v_{r+1},v_r}-\sum_{i=1,\ldots,v_r-1} l_{v_{r-1},i}l_{v_{r+1},i})/l_{v_{r+1},v_{r+1}}$$

y al ser $l_{v_{r-1},v_r}\neq 0$ y $l_{v_{r+1},v_r}\neq 0$ entonces $l_{v_{r-1},v_{r+1}}\neq 0$.

Con este procedimiento hemos encontrado un camino más corto de j a i (es decir con un elemento menos). Repitiendo el proceso, eliminando un vértice en cada paso, hasta llegar a que C=j,k,i concluyendo con esto que $l_{ij}\neq 0$.

---

[4] Ver matriz de llenado en la sección de matrices ralas (página 18). Donde $l_{ij}\in F$ y $a_{ij}\in A$ cada vez que aparezcan.

Corolario 1.6.3

Si j=p(k) y $l_{ik} \neq 0$ entonces $l_{ij} \neq 0$.

Demostración

Tenemos que por ser j=p(k), $l_{jk} \neq 0$, y $l_{ik} \neq 0$ por lo tanto hay un camino de j a i en G(F) j,k,i tal que k<j<i por la proposición 1.6.2, $l_{ij} \neq 0$.

El siguiente teorema nos muestra la relación de un elemento distinto de cero en L, con el árbol de eliminación.

Teorema 1.6.4

Sea A una matriz rala, simétrica y definida positiva y sea i>j.

Si $l_{ij} \neq 0$, entonces i es un ancestro de j en el árbol de eliminación.

Demostración

Si i=p(j) entonces i es un ancestro de j.

Si no, sea $w_1=p(j)$ entonces $l_{w_1,j} \neq 0$ y $w_1<i$.

Por el corolario 1.6.3 $l_{i,w_1} \neq 0$.

Si $i=p(w_1)$ entonces i es un ancestro de w_1 y por lo tanto de j.

Si no, repetimos el procedimiento anterior hasta encontrar una w_k tal que $i=p(w_k)$, lo cual es posible por tener un número finito de nodos. Entonces tenemos un camino $j, w_1, \ldots, w_r, i$ donde $w_1=p(j)$ y $w_{m+1}=p(w_m)$ para m=2,....r-1 y $i=p(w_r)$ por lo

tanto i es un ancestro de j .

Corolario 1.6.5

Sea T[i] y T[j] tal que $T[i] \cap T[j] = \emptyset$, entonces $\forall\ s \in T[i]$ y $\forall\ t \in T[j]$, $l_{st}=0$ con s>t.

Demostración

Por contradicción. Supongamos que existen s,t tal que $s \in T[i]$, $t \in T[j]$ y $l_{s,t}\neq 0$. Por el teorema 1.6.4 s es un ancestro de t, entonces $t \in T[s]$ pero $T[s] \subseteq T[i]$ $\therefore$ $t \in T[i]$ entonces $T[i] \cap T[j] \neq \emptyset$, pero esto no puede ser. Por lo tanto $l_{st}=0$.

Los siguientes teoremas nos muestran cuando $l_{ij}\neq 0$ en L, a partir de la matriz A y su gráfica asociada.

Teorema 1.6.6

Sea i>j. Entonces $l_{ij}\neq 0$ si y sólo si existe un camino $i,p_1,\ldots,p_t,j$ en G(A) tal que $p_k<j$ para k=1,...,t.

Demostración

=>) Por inducción sobre el número de columnas.

Sea $l_{ij}\neq 0$

Para j=1

como $l_{11}\neq 0$ => $a_{11}\neq 0$ y el camino en G(A) es i,j.

Suponemos cierto para K<j. Falta demostrar para j.

Si $a_{ij}\neq 0$, el camino en G(A) es i,j

Si $a_{ij}=0$ => $l_{ij}=(-\sum_{k=1}^{j-1} l_{ik}l_{jk})/l_{jj}$ para que sea distinto de cero

por lo menos alguna l_{ik} l_{jk} es distinta de cero.

Por hipótesis de inducción, existen caminos en G(A) de i a k o de j a k con elementos menores que k, lo que implica que existe un camino de i a j con elementos menores que k.

<=) Sup. que existe un camino en G(A) $i,p_1,\ldots,p_t,j$ entonces, $i,p_1,\ldots,p_t,j$ existe en G(F) y por el teorema $l_{ij}\neq 0$.

Este teorema nos ayuda para la demostración del siguiente.

Teorema 1.6.7

Sea i>j. Entonces $l_{ij}\neq 0$ si y sólo si existe un camino $i,p_1,\ldots,p_t,j$ en G(A) tal que $\{p_1,\ldots,p_t\} \subseteq T[j]$.

Demostración

=>) Sup. que existe el camino. Dado que j es un ancestro de los nodos $\{p_1,\ldots,p_t\}$, entonces $j>p_k$, k=1,...,t. Entonces por el teorema 1.6.6 $l_{ij}\neq 0$.

<=) Sup. $l_{ij}\neq 0$ por el teorema anterior, existe un camino en G(A) $i,p_1,\ldots,p_t,j$ con $p_k<j$ para k=1,...,t.

Para t=0 $j \in T[j]$. Para t>0, sea s>0 tal que s es el más grande de las t donde $p_t \notin T[j]$, entonces $p_{s+1} \in T[j]$. Dado que el camino está en G(A) => $\{p_s,p_{s+1}\} \in G(A)$ y de ahí a G(F). Pero p_{s+1} no puede ser un ancestro de p_s ya que $p_s \notin T[j]$. Entonces p_s es un ancestro de p_{s+1}. Por lo que p_s y j son ancestros de p_{s+1} y j no es un ancestro de p_s, entonces p_s es un ancestro de j. Pero eso no puede ser ya que $p_s<j$. Por lo tanto

$\{p_1,\ldots,p_t\} \subset T[j]$.

Teorema 1.6.8

$l_{ij}\neq 0$ si y sólo si el nodo j es un ancestro de algún nodox_k en el árbol de eliminación donde $a_{ij}\neq 0$.

Demostración

<=) Sea j ancestro de algún nodo k donde $a_{ik}\neq 0$, si k=j => $l_{ij}\neq 0$.

Si k<j existe un camino de k a j con nodos menores de j en G(F) (tomando el camino de k a j en T(A)) y como$a_{ik}\neq 0$ entonces $l_{ik}\neq 0$ por lo que existe un camino de i a j con elementos distintos de cero en G(F) entonces $l_{ij}\neq 0$.

=>) Sup. $l_{ij}\neq 0$- existe un camino $j,p_1,\ldots,p_t,i$ en G(A) tal que $p_k<j$ para k=1,2,...,t.

Si t=0 => $a_{ij}\neq 0$ porque j es ancestro de él mismo.

Si t>0, tomando $p_t=k$ => $a_{ik}\neq 0$ y por el teorema anterior $\{p_1,\ldots,p_t\} \subset T[j]$ => $p_t=k$ es descendiente de j .

El siguiente algoritmo, obtiene el árbol de eliminación en un vector, que en la posición i contiene el padre de i.

ALGORITMO 1

1. Para i=1,...,n.
   1.1 padre[i]=0.
   1.2 Para todo $k \in Adj(i)$ y k<i.

1.2.1 r=k.

1.2.2 M i e n t r a s $padre(r)\neq 0$ y $padre(r)\neq i$

r=padre(r).

1.2.3 Si padre(r)=0 entonces padre(r)=i.

Ejemplo

Sea

$$A = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{pmatrix} X & & X & & X \\ & X & X & X & X \\ & X & X & & \\ X & & & X & X \\ & X & & X & X \\ X & X & & & X \end{pmatrix}$$

1. i=1,2

   padre[i]=0

2. i=3

   padre[3]=0

   $Adj[3]=\{2\}$ 2<3 padre[2]=3

3. i=4

   padre[4]=0

   $Adj[4]=\{1,5\}$ 1<4 padre[1]=4

4. i=5

   padre[5]=0

   $Adj[5]=\{2,4\}$ 2<5 $padre[2]=3\neq 0$

   r=padre[2]=3 entonces padre[3]=5

   4<5 padre[4]=5

5. i=6

   padre[6]=0

Adj[6]={1,2} 1<6 *padre*[1]=4≠0

r=padre[1]=4 *padre*[4]=5≠0

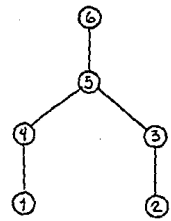
r=padre[4]=5 entonces padre[5]=6

2<6 *padre*[2]=3≠0

r=padre[2]=3 *padre*[3]=5≠0

r=padre[3]=5 *padre*[5]=6≠0

El árbol de eliminación queda de la siguiente manera



$\eta(j)$ es un vector que nos indica cuantos elementos distintos de cero se tienen en la columna j de L, el cual se calcula con el siguiente algoritmo, donde marca es un vector auxiliar.

ALGORITMO 2

1. Para j=1,....,n $\eta(j)=1$
2. Para i=1,....,n
   - 2.1 marca(i)=i;
   - 2.2 Para k<i y $a_{ik}\neq 0$
     - 2.2.1 Mientras *marca*$(j)\neq i$
       - 2.2.1.1 $\eta(j)=\eta(j)+1$
       - 2.2.1.2 marca(j)=i

2.2.1.3 j=padre(j)

Ejemplo

Con el ejemplo utilizado en el algoritmo 1.

1. $\eta[j]=1 \quad j=1,\ldots,6$
2. i=1,2

   marca[i]=i
3. i=3

   marca[3]=3

   $a_{32}\neq 0 \quad marca[2]\neq 3 \quad \eta[2]=2$

   marca[2]=3
4. i=4

   marca[4]=4

   $a_{41}\neq 0 \quad marca[1]\neq 4 \quad \eta[1]=2$

   marca[1]=4
5. i=5

   marca[5]=5

   $a_{52}\neq 0 \quad marca[2]\neq 5 \quad \eta[2]=3$

   marca[2]=5

   j=padre[2]=3

   $marca[3]\neq 5 \quad \eta[3]=2$

   marca[3]=5

   $a_{54}\neq 0 \quad marca[4]\neq 5 \quad \eta[4]=2$

   marca[4]=5
6. i=6

marca[6]=6

$a_{61}\neq 0$ *marca*[1]$\neq$6 $\eta[1]=3$

marca[1]=6

j=padre[1]=4

marca[4]$\neq$6 $\eta[4]=3$

marca[4]=6

j=padre[4]=5

marca[5]$\neq$6 $\eta[5]=2$

marca[5]=6

$a_{62}\neq 0$ *marca*[2]$\neq$6 $\eta[2]=4$

marca[2]=6

j=padre[2]=3

marca[3]$\neq$6 $\eta[3]=3$

marca[3]=6

j=padre[3]=5 marca[5]=6

Con lo que obtenemos $\eta[1]=3$ $\eta[2]=4$ $\eta[3]=3$ $\eta[4]=3$ $\eta[5]=2$ $\eta[6]=1$ que son el número de elementos distintos de cero en cada columna de L.

Nos interesa conocer como es la estructura de cada columna.

Teorema 1.6.9

La estructura de la columna j del factor de Cholesky, está dada por

$$Adj_{G(A)}(T[j]) \cup \{j\} = \{i \;/\; l_{ij}\neq 0,\ i\geq j\}$$

Demostración

$l_{jj} \neq 0$, con lo que j pertenece a la estructura.

Si $l_{ij} \neq 0$ con $i>j$, entonces j es un ancestro de algún nodo k con $a_{ik} \neq 0$. Entonces $i \in Adj_{G(A)}(T[k])$, donde $k \in T[j]$, es decir $i \in Adj_{G(A)}(T[j])$.

Con el siguiente algoritmo obtenemos la estructura de cada columna.

ALGORITMO 3

1. Para $j=1,\ldots,n$

   1.1 $Adj(T[j]) = Adj(j) - \{1,2,\ldots,j-1\}$

   1.2 Para los hijos s de j

   $$Adj(T[j]) = Adj(T[j]) \cup Adj(T[s]) - \{j\}$$

Ejemplo

Siguiendo con la misma matriz.

1. $j=1$

$$Adj[T(1)] = \{4,6\} - \emptyset$$

2. $j=2$

$$Adj[T(2)] = \{3,5,6\} - \emptyset$$

3. $j=3$

$$Adj[T(3)] = \{2\} - \{2\} = \emptyset$$

2 es hijo de 3

$$Adj[T(3)] = \emptyset \cup \{3,5,6\} - \{3\} = \{5,6\}$$

4. j=4

$$Adj[T(4)] = \{1,5\} - \{1\} = \{5\}$$

1 es hijo de 4

$$Adj[T(4)] = \{5\} \cup \{4,6\} - \{4\} = \{5,6\}$$

5. j=5

$$Adj[T(5)] = \{2,4\} - \{2,4\} = \emptyset$$

3 es hijo de 5

$$Adj[T(5)] = \emptyset \cup \{5,6\} - \{5\} = \{6\}$$

4 es hijo de 5

$$Adj[T(5)] = \{6\} \cup \{5,6\} - \{5\} = \{6\}$$

6. j=6

$$Adj[T(6)] = \{1,2\} - \{1,2\} = \emptyset$$

5 es hijo de 6

$$Adj[T(6)] = \emptyset \cup \{6\} - \{6\} = \emptyset$$

CAPITULO 2

ORDENAMIENTOS DE MATRICES Y GRAFICAS

2.1 Ordenamiento

Cuando A es una matriz definida positiva los pivotes pueden ser seleccionados de la diagonal. De ahí los renglones y las columnas de A pueden ser permutados de tal forma que se preserve la simetría. Esto es, si A es simétrica PAP^t también es simétrica.

Si queremos resolver el sistema Ax=b, con A una matriz rala y simétrica y si P es una matriz de permutación, el sistema es equivalente a $PAP^t Px = Pb$ ya que $P^t P = I$.

Un ordenamiento P se dice óptimo con respecto al llenado si el factor L de PAP^t tiene menos llenado que el factor de QAQ^t para las demás Q permutaciones de A.

Si A es una matriz de orden n, existen n! permutaciones diferentes. De estas permutaciones existe al menos una que provoca menos llenado al realizar la factorización.

Sean A y M matrices simétricas, definidas positivas y ralas en

donde $A = PMP^t = L\,L^t$, si para toda $a_{ij}=0$ implica que $l_{ij}=0$ entonces decimos que M es una matriz de eliminación perfecta, es decir L no tiene llenado.

El siguiente resultado nos muestra la estructura de una matriz A, que al ser factorizada no tenga llenado.

Proposición 2.1.1

Sea A una Matriz rala simétrica de nxn.

Si $a_{ij}=0$ implica que $l_{ij}=0$ si y sólo si $\forall\ 1\le k<l<m\le n$

$$a_{mk}\neq 0,\ a_{lk}\neq 0 \rightarrow a_{ml}\neq 0$$

Demostración

<=) Supongamos que si $a_{mk}\neq 0,\ a_{lk}\neq 0 \rightarrow a_{ml}\neq 0$. Y sea $a_{ij}=0$. Hay que demostrar que $l_{ij}=0$. Por inducción sobre j. Para j=1, si $a_{i1}=0$ entonces $l_{i1} = \frac{a_{i1}}{l_{11}} = 0$.

Para j=2. si $a_{12}=0$ entonces $l_{12} = \frac{-l_{11}\, l_{21}}{l_{22}}$ pero $l_{12}=0$

porque si no fuera así $l_{11}\neq 0$ y $l_{21}\neq 0$ => $a_{11}\neq 0$ y $a_{21}\neq 0 \rightarrow a_{12}\neq 0$.

Entonces $l_{12}=0$.

Suponemos cierto para k<j. Por demostrar para j.

Sea $a_{ij}=0$. Entonces $l_{ij}=(-\sum_{k=1}^{j-1} l_{ik}l_{jk})/l_{jj}$.

Pero $l_{ij}=0$ ya que si no fuera así $l_{ik}\neq 0$ y $l_{jk}\neq 0$ para alguna k<j. Por hipótesis de inducción $a_{ik}\neq 0$ y $a_{jk}\neq 0 \rightarrow a_{ij}\neq 0$. Entonces $l_{ij}=0$.

=>) Sup $a_{ij}=0$ implica que $l_{ij}=0$. Es decir si $l_{ij}\neq 0 \rightarrow a_{ij}\neq 0$.

Sea $a_{ik}\neq 0$ y $a_{jk}\neq 0 \rightarrow l_{ik}\neq 0$ y $l_{jk}\neq 0$ para alguna k<j, pero

$l_{ij}=(-\sum_{k=1}^{j-1} l_{ik}l_{jk})/l_{jj}$ entonces $l_{ij}\neq 0 \rightarrow a_{ij}\neq 0$ por hipótesis.

La matriz de llenado F, es una matriz de eliminación perfecta ya que $\forall\ 1 \le k < j < i \le n$

$$l_{ik} \neq 0,\ l_{jk} \neq 0 \rightarrow l_{ij} \neq 0$$

Con el teorema anterior y la definición de matriz de eliminación perfecta, llegamos a la siguiente relación:
M es una matriz de eliminación perfecta, si y sólo si existe una permutación P, tal que si $A = PMP^t$ entonces

$\forall\ 1 \le k < l < m \le n$

$a_{mk} \neq 0,\ a_{lk} \neq 0 \rightarrow a_{ml} \neq 0$

Nos interesa entonces, encontrar una permutación que nos minimice el llenado.
Para esto, es muy útil la relación entre una matriz rala y simétrica, y su gráfica asociada.
Un ordenamiento P es equivalente a permutar el nombre de los vértices en G.
Sea G una gráfica con n vértices. Una *numeración* de G es una función biyectiva

$$f: V(G) \rightarrow \{1, \ldots, n\}$$

En una gráfica con n vértices, existen n! numeraciones

posibles.

Otro concepto necesario es la deficiencia de un nodo x, que es el conjunto de pares distintos de vértices que pertenecen a adj(x) y que no son adyacentes entre ellos, esto es

$$D(x) = \{ \{y, z\} / y, z \in adj(x),\ y \neq z,\ y \notin adj(z) \}$$

Dado un vértice y, se define la gráfica de eliminación de y como:

$$G_y(V(G)-\{y\},\ A(V(G)-\{y\} \cup D(y))$$

Para una gráfica numerada G, el orden de secuencia de gráficas de eliminación es $G_1, \ldots, G_{n-1}$ y se define recursivamente como

$G_1 = G_{x_1}$ y $G_i = (G_{i-1})_{x_i}$ para i=2,....,n-1.

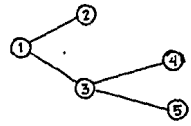
Definimos el proceso de eliminación de una gráfica con numeración f como el conjunto

$$P(G, f) = [G=G_0, G_1, \ldots, G_{n-1}]$$

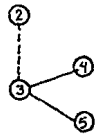
La estructura de la matriz de llenado F y su gráfica asociada G(F) quedan determinadas mediante este proceso.

El siguiente ejemplo nos ilustra lo anterior.

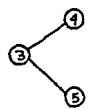
$$M_0 = \begin{pmatrix} x & x & x & & \\ x & x & & & \\ x & & x & x & x \\ & & x & x & \\ & & x & & x \end{pmatrix}$$

$G_0 =$ 

$$M_1 = \begin{pmatrix} x & x & x & & \\ x & x & \otimes & & \\ x & \otimes & x & x & x \\ & & x & x & \\ & & x & & x \end{pmatrix}$$

$G_1 =$ 

$$M_2 = \begin{pmatrix} x & x & x & & \\ x & x & \otimes & & \\ x & \otimes & x & x & x \\ & & x & x & \\ & & x & & x \end{pmatrix}$$

$G_2 =$ 

$$M_3 = \begin{pmatrix} x & x & x & & \\ x & x & \otimes & & \\ x & \otimes & x & x & x \\ & & x & x & \otimes \\ & & x & \otimes & x \end{pmatrix}$$

$G_3 =$ 

$$M_4 = \begin{pmatrix} x & x & x & & \\ x & x & \otimes & & \\ x & \otimes & x & x & x \\ & & x & x & \otimes \\ & & x & \otimes & x \end{pmatrix} \qquad G_{T_4} = \textcircled{5}$$

La estructura de M_4 es la misma estructura que $\bar{F}$ en donde

$\otimes$ significa un llenado.

Decimos que un proceso es de eliminación perfecta si

$$G_i = G\left(V(G) - \bigcup_{j=1}^{i} (x_j) \right)$$

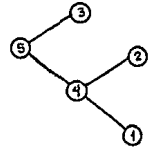
Claramente un proceso es de eliminación perfecta si en cada G_{i-1} , $D(x_i)=\emptyset$ $i=1,\ldots n$.

Y esto sucede si no existe llenado, con lo que tenemos que la numeración f en un proceso de eliminación perfecta es equivalente a un ordenamiento de eliminación perfecta.
Consideremos el ejemplo anterior con un ordenamiento

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

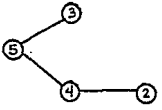
$$A_0 = PM_0P^t = \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & & x \\ & & x & x & x \end{pmatrix}$$

$G_0 =$



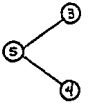
$$A_1 = \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & & x \\ & & x & x & x \end{pmatrix}$$

$G_1 =$



$$A_2 = \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & & x \\ & & x & x & x \end{pmatrix}$$

$G_2 =$



$$A_3 = \begin{pmatrix} x & & x & \\ & x & x & \\ & & x & x \\ x & x & x & x \\ & & x & x \end{pmatrix} \qquad G_3 =$$

$$A_4 = \begin{pmatrix} x & & x & \\ & x & x & \\ & & x & x \\ x & x & x & x \\ & & x & x \end{pmatrix} \qquad G_4 =$$

Como puede observarse en A_4 y G_4, no existe llenado.

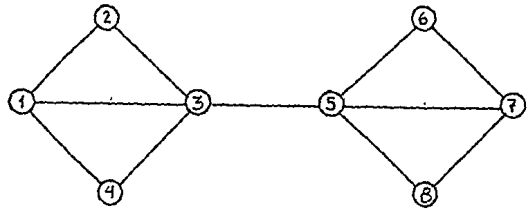
2.2 Gráficas triangularizadas

La relación que existe entre una matriz de eliminación perfecta y su gráfica, surge a partir de la siguiente definición.

Una gráfica G es *triangularizada*, si para todo ciclo $m=[p_1,p_2,\ldots,p_n,p_1]$ de longitud n>3, existe una arista en G (llamada cuerda) que une dos vértices no consecutivos de m.

En el ejemplo siguiente se muestra una gráfica

triangularizada.



Aquí existen dos ciclos con longitud mayor que tres:

$m_1=[1,2,3,4]$ con cuerda (1,3) y

$m_2=[5,6,7,8]$ con cuerda (5,7).

En las gráficas triangularizadas existen vértices tales que su deficiencia es vacía.[1]

En el ejemplo anterior

$$D(2) = D(4) = D(6) = D(8) = \emptyset$$

Con esto podemos llegar a que si G es triangularizada entonces esiste una numeración f tal que P[G,f] es de eliminación perfecta entonces cada G_i es triangularizada.

Por otro lado G(F), también es triangularizada ya que, como

---

[1] D. J. ROSE. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, en Graph Theory and Computing p. 194-199

vimos anteriormente F es una matriz de eliminación perfecta.
La relación que guarda una matriz de eliminación perfecta con su gráfica asociada, se sigue de la siguiente proposición.

Proposición 2.2.1

Sea M una matriz rala, simétrica y definida positiva. M es una matriz de eliminación perfecta, si y sólo si, la gráfica asociada a M es triangularizada.

Demostración

=>) Sea M una matriz de eliminación perfecta y G su gráfica asociada. Por la proposición 2.1.1, existe una permutación P tal que si $A = PMP^t$ entonces $\forall\ 1 \le k < l < m \le n$

$a_{mk} \neq 0,\ a_{lk} \neq 0 \rightarrow a_{ml} \neq 0$.

Tomemos un ciclo en G, $m = [p_1, p_2, \ldots, p_n, p_1]$ con n mayor que 3. Y sea p_j el más chico de acuerdo a nuestro ordenamiento, entonces $p_{j+1} > p_j$ y $p_{j-1} > p_j$. supongamos que $p_{j+1} > p_{j-1}$ entonces $a_{p_{j+1}p_j} \neq 0$, y $a_{p_{j-1}p_j} \neq 0 \rightarrow a_{p_{j+1}p_{j-1}} \neq 0$ con lo que tenemos que existe (p_{j+1}, p_{j-1}) , que es una cuerda, por lo tanto G es

triangularizada.

<=) Por inducción sobre el número de vértices. Para n=1 se cumple ya que G no tiene ciclos y entonces G es triangularizada y al hacer la factorización no habría llenado. Suponemos cierto para n=k, por demostrar para n=k+1.

Sea G una gráfica triangularizada con k+1 vértices. Tomamos un vértice x de tal manera que $D(x)=\emptyset$. Tomamos la subgráfica $G^1(X-\{x\})$, es decir quitar a x y sus aristas incidentes. G^1 es triangularizada ya que todo ciclo que no involucre a x en G tiene una cuerda en G^1 , y si el ciclo involucra a x entonces se rompe ese ciclo y no se necesita una cuerda. Por hipótesis de inducción existe un ordenamiento f^1 en G^1 tal que

$\forall\ 1 \le q < l < m \le k \quad a_{mq} \neq 0,\ a_{lq} \neq 0 \rightarrow a_{ml} \neq 0$.

Entonces en G, tomamos el ordenamiento f(x)=1 y f^1+1 para los nodos en G^1 , se cumple que $\forall\ 2 \le q < l < m \le k+1$ $a_{mq} \neq 0,\ a_{lq} \neq 0 \rightarrow a_{ml} \neq 0$. Sólo falta demostrar que $\forall\ 2 \le l < m \le k+1$

$a_{m1}\neq 0$, $a_{11}\neq 0 \rightarrow a_{m1}\neq 0$, pero esto es cierto por ser $D(x)=\emptyset$.

2.3 Deficiencia mínima y grado mínimo

La sección anterior nos da una idea para obtener un algoritmo que nos reduzca el llenado.
El algoritmo, llamado de deficiencia mínima, es el siguiente:

ALGORITMO 4

1.-Sea G_0 la gráfica correspondiente a la matriz A.

2.- Para i=1,...,n

2.1 Tomar el vértice tal que su deficiencia tenga el menor número de elementos en G_{i-1} y asociarle el número i.

2.2 Eliminar el vértice i, las aristas que lo contienen, y agregar D(i) en G_i .

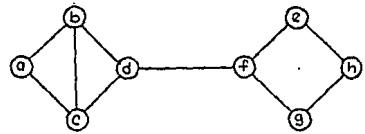
Ejemplo

Sea

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{bmatrix} x & x & x & & & & & \\ x & x & x & x & & & & \\ x & x & x & x & & & & \\ & x & x & x & & x & & \\ & & & & x & x & & x \\ & & & x & x & x & x & \\ & & & & & x & x & x \\ & & & & x & & x & x \end{bmatrix} \end{array}$$

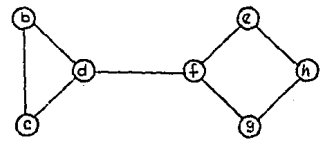
Primer paso.

$G_0 =$



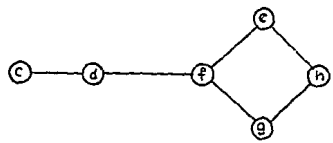
Segundo paso. i=1 a es el vértice que tiene deficiencia mínima en G_0 .

$G_1 =$

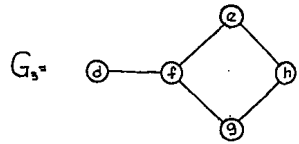


i=2 b es el vértice que tiene deficiencia mínima en G_1 .

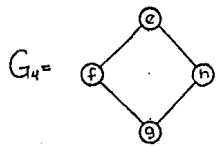
$G_2 =$



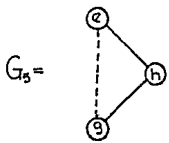
i=3 c es el vértice que tiene deficiencia mínima en G_2 .



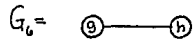
i=4 d es el vértice que tiene deficiencia mínima en G_3 .



i=5 f es el vértice que tiene deficiencia mínima en G_4 .



i=6 e es el vértice que tiene deficiencia mínima en G_5 .



i=7 g es el vértice que tiene deficiencia mínima en G_6 .

G_7 = (h)

i=8 h es el único vértice que queda en en G_7 .

La matriz permutada queda

$$PAP^t = \begin{array}{c} \\ a \\ b \\ c \\ d \\ f \\ e \\ g \\ h \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & & & & & & \end{array} \\ \left[\begin{array}{ccccccccc} x & x & x & & & & & & \\ x & x & x & x & & & & & \\ x & x & x & x & & & & & \\ & x & x & x & x & & & & \\ & & & x & x & x & x & & \\ & & & & x & x & & x & \\ & & & & x & & x & x & \\ & & & & & x & x & x & \end{array}\right] \end{array}$$

La desventaja de este algoritmo es su lentitud, ya que en cada paso, se tiene que calcular la deficiencia de cada nodo, lo que involucra (V(x)V(x)+1)/2 pruebas, donde V(x) es el número de vecinos de x.

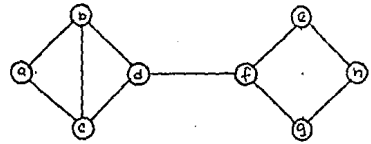
De la forma en que fue definida D(x) para un nodo x, tomando en cuenta que si disminuye el grado del nodo, disminuye la posibilidad de tener muchos elementos en la deficiencia, podemos tomar en cada paso el nodo de grado mínimo en lugar

del nodo con deficiencia mínima.

Tomando el mismo ejemplo para el algoritmo de grado mínimo.

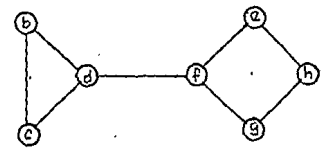
Primer paso.

$G_0 =$



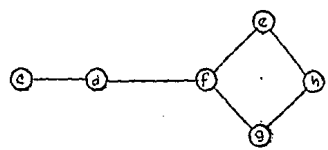
Segundo paso. i=1 a es el vértice que tiene grado mínimo en G_0 .

$G_1 =$



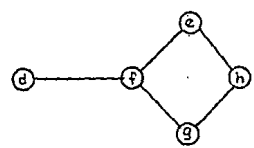
i=2 b es el vértice que tiene grado mínimo en G_1 .

$G_2 =$



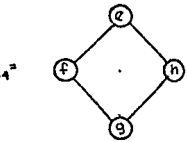
i=3 c es el vértice que tiene grado mínimo en G_2 .

$G_3 =$



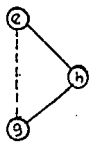
i=4 d es el vértice que tiene grado mínimo en G_3 .

G_4=



i=5 f es el vértice que tiene grado mínimo en G_4 .

G_5=



i=6 e es el vértice que tiene grado mínimo en G_5 .

G_6= g — h

i=7 g es el vértice que tiene grado mínimo en G_6 .

G_7= h

i=8 h es el único vértice que queda en en G_7 .

La matriz permutada obtenida es la misma que usando el algoritmo de deficiencia mínima.
Podemos notar en el ejemplo, que en el primer paso, existen varios vértices con el mismo grado, pero siempre tomamos el primero que nos encontramos en la gráfica.
La desventaja de este algoritmo, es que puede producir un número mayor de llenado.

2.4 Reordenamientos equivalentes

Sea A una matriz de nxn rala y simétrica. Una permutación P se dice que es un reordenamiento equivalente de A si la matriz permutada $M=PAP^t$ tiene el mismo conjunto de llenado que A.

Esto es G(F) es la misma en ambas matrices.
Como lo demuestra Rose[2], para cada nodo x en G(F), existe un ordenamiento de eliminación perfecta P en G(F) tal que el nodo x es numerado al final. Esta P es un reordenamiento equivalente ya que se tiene el mismo llenado.
Con lo anterior tenemos el siguiente corolario.

[2] Ibid. p. 183-217

Corolario 2.3.1

Para todo nodo x en G(A), existe un ordenamiento P en G(A) tal que el nodo x es numerado al final y tal que la gráfica de llenado de $G(PAP^t)$ es una subgráfica de la gráfica de llenado de G(A).

Demostración

Sea G(F) la gráfica de llenado de G(A). Existe un ordenamiento de eliminación perfecta P en G(F) tal que x es numerado al final. Es decir, la factorización de la matriz de llenado PFP^t no crea llenado adicional. Entonces el llenado creado al factorizar PAP^t cuenta para $G(PFP^t)$. De ahí que $G(PAP^t)$ es una subgráfica de $G(PFP^t)=G(F)$ ∎

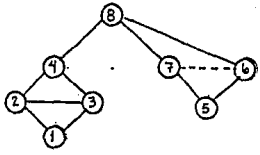
Con este corolario, tenemos que si un nodo es numerado al final, ese nodo será la raíz en el árbol de eliminación.
Existe un algoritmo desarrollado por Liu, en el que dado un nodo x, éste será la raíz en el árbol de eliminación.
La esencia de algoritmo se describe como sigue. Para cualquier nodo en el árbol de eliminación, el ordenamiento obtenido por una rotación del árbol de eliminación en x, mantendrá el ordenamiento de los nodos que no son ancestros de x. Pero los que son ancestros de x serán reordenados tal que los nodos en

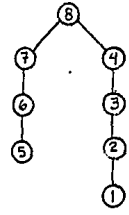
$Adj(T[x]) \cup \{x\}$ son numerados al final.

ALGORITMO 5

1. Sea T el árbol de eliminación.
2. Hacer z=x (x es el nodo que queremos para ser raíz).
3. Mientras z no sea la raíz
   3.1. Ordenar los nodos no numerados de Adj(T[z]) al último antes de los ya renumerados.
   3.2. Marcar los nodos en Adj(T[z]) como numerados.
   3.3. Hacer z=padre de z.
4. Numarar los nodos restantes usando sus ordenamientos originales.

Ejemplo





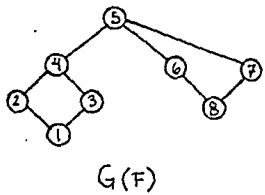
Supongamos que queremos el nodo 5 como raíz.

1. z=5.
2. $Adj(T[5])=\{6,7\}$, al 6 le asignamos el 7 y al 7 el 6.
3. z=6.
4. $Adj(T[6])=\{8,7\}$, al 8 le asignamos el 5.

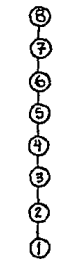
5. z=7.

6. $Adj(T[7]) = \{8\}$.

El nuevo ordenamiento con su árbol de eliminación:



G(F)



T(A)

2.4 Nodos pseudoperiféricos

Para encontrar un nodo apropiado para el reordenamiento de la sección anterior, consideremos las siguientes definiciones.

Dado un vértice $v \in V(G)$, la *estructura de nivel* cimentada en el vértice v, es una partición de V(G)

$$N(v) = \{N_0(v), N_1(v), \ldots, N_{e(v)}(v)\}$$

en donde

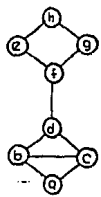
$$N_0(v) = \{v\}, \quad N_1(v) = Adj(N_0(v))$$

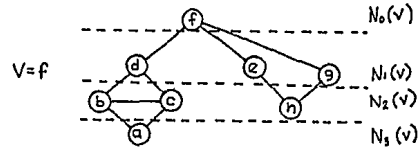
y

$$N_i(v) = Adj(N_{i-1}(v)) - N_{i-2}(v) \quad i=2,3,\ldots,e(v)$$

El entero e(v) se llama la profundidad de la estructura N(v); a $N_0(v), N_1(v), \ldots, N_{e(v)}(v)$ se les llama niveles y al vértice v la raíz.[3]

Ejemplo





La *excentricidad* de un nodo $v \in V(G)$, se define por

$$e(v) = \max\{d(v,u) : v \in V(G)\}$$

donde

$$d(v,u) = \begin{cases} 0 & Si\ u=v \\ \infty & Si\ No\ \exists\ en\ G\ trays.\ de\ v\ a\ u \\ m & Si\ m\ es\ el\ mínimo\ valor\ de\ las\ longs.\ de\ las\ trays.\ de\ u\ a\ v\ en\ G \end{cases}$$

---

[3] ABRIN, Virginia, Algoritmos para reordenar matrices ralas, Tesis de Maestría, p. 41

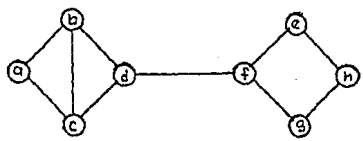
es la distancia entre v y u.[4]

El *diámetro* de una gráfica G está dado por

$$\delta(G) = \max\{e(v) : v \in V(G)\}$$

Un nodo $v \in V(G)$ se dice que es periférico, si su excentricidad es igual al diámetro.

Ejemplo



En la gráfica, la trayectoria (a,b,c,d) tiene longitud 3; d(a,d)=2; $e(a)=e(h)=\delta(G)=5$.

Nos interesa encontrar nodos periféricos v, con lo que tendríamos mayor número de niveles al construir la estructura de nivel cimentada en v.

Encontrar nodos periféricos es una tarea complicada, pero podríamos emplear nodos de excentricidad alta que llamaremos pseudoperiféricos.

El siguiente algoritmo encuentra nodos pseudoperiféricos.

---

[4] CURCO, María del Carmen, Una Introducción a la Teoría de Gráficas, Vínculos Matemáticos, p. 40-41

ALGORITMO 6

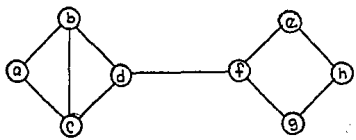
1. Escoger un nodo arbitrario $v \in V(G)$.

2. Generar la estructura de nivel cimentada en v.
3. Elegir un nodo u de grado mínimo en el último nivel de la estructura.
4. Generar la estructura de nivel enraizada en u.

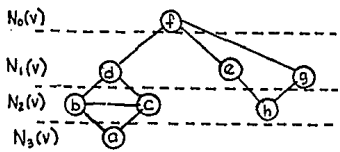
   4.1. Si $e(u)>e(v)$ asignarle a v=u y regresar a 3.

5. El nodo u es un nodo pseudoperiférico.

Ejemplo

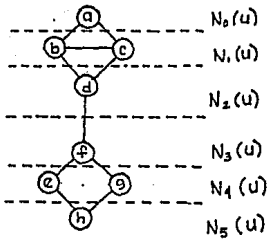


1. v=f
2.



3. u=a .

4.

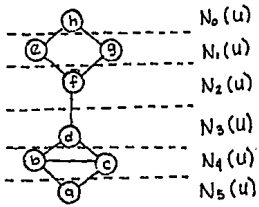


5. $5=e(u)>e(v)=3$

6. v=a

7. u=h

8.



9. h es un nodo pseudoperiférico.

CAPITULO 3

FACTORIZACION NUMERICA

3.1 Método Frontal

La solución de un sistema Ax=b donde A es una matriz simétrica, definida positiva y rala, se consigue con la siguiente serie de pasos:

1. Ordenamiento. Encontrar un ordenamiento P para A; esto es, determinar la matriz de permutación P tal que el factor de Cholesky L de PAP^t sufra menos llenado.
2. Factorización simbólica. Determinar la estructura de L y crear una estructura de datos para almacenar A y calcular los elementos distintos de cero de L.
3. Factorización numérica. Insertar los elementos distintos de cero de A en la estructura de datos, y calcular el factor de Cholesky L de PAP^t.
4. Solución triangular. Resolver $Ly=Pb$, $L^tz=y$, $x=P^tz$.

Los primeros dos pasos se estudiaron en los capítulos anteriores. En este capítulo veremos dos métodos para obtener la factorización numérica.

El *método frontal* consiste en tomar para cada columna los elementos distintos de cero y realizar el primer paso de la factorización de Cholesky por submatrices.

Para esto definimos lo que es la matriz frontal.

Sea A una matriz simétrica, definida positiva y rala, y sean $i_1, i_2, \ldots, i_r$ los índices de los elementos distintos de cero en la columna j después de haber realizado la factorización de Cholesky, j-1 veces por submatrices.

Entonces

$$F_j = \begin{pmatrix} a_{jj}^{\,j} & a_{ji_1}^{\,j} & \cdots & a_{ji_r}^{\,j} \\ a_{i_1j}^{\,j} & a_{i_1i_1}^{\,j} & \cdots & a_{i_1i_r}^{\,j} \\ \vdots & \vdots & & \vdots \\ a_{i_rj}^{\,j} & a_{i_ri_1}^{\,j} & \cdots & a_{i_ri_r}^{\,j} \end{pmatrix}$$

donde $a_{ij}^{\,k}$ es el valor numérico de la entrada i,j después de j-1 pasos.

Con esta definición tenemos que

$$F_j = \begin{pmatrix} l_{jj} & 0 \\ l_{i_1j} & \\ \vdots & I \\ l_{i_ri_1} & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & H \end{pmatrix} \begin{pmatrix} l_{jj} & l_{ji_1} & \cdots & l_{ji_r} \\ & & & \\ 0 & & I & \end{pmatrix}$$

$$H = \begin{pmatrix} a_{ji_1}^{\,j+1} & \cdots & a_{ji_r}^{\,j+1} \\ a_{i_1i_1}^{\,j+1} & \cdots & a_{i_1i_r}^{\,j+1} \\ \vdots & & \vdots \\ a_{i_ri_1}^{\,j+1} & \cdots & a_{i_ri_r}^{\,j+1} \end{pmatrix}$$

donde

$$a_{ij}^{k+1} = a_{ij}^{k} - \frac{a_{ik}^{k}\, a_{kj}^{k}}{a_{kk}^{k}}$$

como se vió en el primer capítulo.

El método frontal realiza la factorización de Cholesky, por submatrices, pero tomando únicamente los elementos distintos de cero, con lo que tenemos matrices densas.

El siguiente ejemplo nos muestra los pasos a seguir.

Sea

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 4 & 0 & 4 \\ 1 & 0 & 10 & 9 \\ 0 & 4 & 9 & 29 \end{pmatrix}$$

$$F_1 = \begin{matrix} 1 \\ 3 \end{matrix}\begin{pmatrix} 1 & 1 \\ 1 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 4 & 0 & 4 \\ 1 & 0 & 9 & 9 \\ 0 & 4 & 9 & 29 \end{pmatrix}$$

$$F_2 = \begin{matrix} 2 \\ 4 \end{matrix}\begin{pmatrix} 4 & 4 \\ 4 & 29 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 25 \end{pmatrix}\begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 0 & 9 & 9 \\ 0 & 2 & 9 & 25 \end{pmatrix}$$

$$F_3 = \begin{matrix} 3 \\ 4 \end{matrix}\begin{pmatrix} 9 & 9 \\ 9 & 25 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 16 \end{pmatrix}\begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 0 & 3 & 3 \\ 0 & 2 & 3 & 16 \end{pmatrix}$$

$$F_4 = 16 - 4 \cdot 4$$

y

$$L = \begin{pmatrix} 1 & & & \\ 0 & 2 & & \\ 1 & 0 & 3 & \\ 0 & 2 & 3 & 4 \end{pmatrix}$$

3.2 Método Multifrontal

Este método reorganiza una matriz rala en una secuencia de factorizaciones parciales de matrices densas pequeñas.

Es una generalización del método anterior. Para la aplicación del método multifrontal, es necesario conocer las matrices frontales y las de actualización; estas serán definidas a continuación.

Sea A una matriz de nxn simétrica y definida positiva, y sea L su factor de Choleski. Tomemos a los elementos $i_0,\ldots,i_r$ de la j-ésima columna de L como los distintos de cero con $i_0=j$, es decir los elementos distintos de cero en la columna j bajo la diagonal.

Definimos la siguiente matriz:

$$\overline{U}_j = -\sum_{k\in T(j)-(j)} \begin{pmatrix} l_{j,k} \\ l_{i_1,k} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,k} \end{pmatrix} \cdot (l_{j,k}, l_{i_1,k}, \ldots, l_{i_r,k})$$

Con lo que definimos a la j-ésima matriz frontal como sigue:

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \cdot & & & \\ \cdot & & 0 & \\ \cdot & & & \\ a_{i_r,j} & & & \end{pmatrix} + \overline{U}_j$$

Donde F_j es una matriz de (r+1)x(r+1).

De esta definición, el primer renglón y la primera columna de F_j están formados por la columna j de A y su actualización.

Es decir:

$${}^{j}F_{11} = a_{jj} - \sum_{k\in T(j)-(j)} l_{jk}^2$$

$${}^{j}F_{*1} = a_{*j} - \sum_{k\in T(j)-(j)} l_{*k} l_{jk} \text{ donde } * \in \{i_1, \ldots, i_r\}$$

con lo que se puede calcular $l_{jj} = ({}^{j}F_{11})^{\frac{1}{2}}$

$$l_{*j} = \frac{{}^{j}F_{*1}}{l_{jj}}$$

Y con esto se puede realizar el primer paso de la factorización de Cholesky por submatrices:

$$F_j=\begin{pmatrix} l_{j,j} & 0 \\ l_{i_1,j} & \\ \cdot & I \\ \cdot & \\ \cdot & \\ l_{i_r,j} & \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & U_j \end{pmatrix}\begin{pmatrix} l_{j,j} & l_{i_1,j} & \cdots & l_{i_r,j} \\ & & & \\ 0 & & I & \end{pmatrix}$$

De esta proceso se desprende la matriz de actualización U_j, cuya estructura nos define el siguiente teorema.

Teorema 3.2.1

$$U_j=-\sum_{k\in T[j]}\begin{pmatrix} l_{i_1,k} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,k} \end{pmatrix}(l_{i_1,k},\ldots,l_{i_r,k})$$

Demostración

Este resultado se sigue de la eliminación de la primera columna de F_j.

$$F_j=\begin{pmatrix} d & v^t \\ v & H \end{pmatrix}=$$

$$=\begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I_r \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & H_1 \end{pmatrix}\begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I_r \end{pmatrix}$$

$H_1 = H - \frac{vv^t}{d}$ y para nuestra demostración $H_1 = U_j$ y

$$\frac{v^t}{\sqrt{d}} = (l_{i_1,r}, \ldots, l_{i_r,k})$$

pero

$$U_j = H - \begin{pmatrix} l_{i_1,j} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,j} \end{pmatrix} (l_{i_1,j}, \ldots, l_{i_r,j})$$

y

$$H = - \sum_{k \in \{j\}^{-}(j)} \begin{pmatrix} l_{i_1,k} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,k} \end{pmatrix} (l_{i_1,k}, \ldots, l_{i_r,k})$$

en la que se tienen los mismos elementos de $\overline{U}_j$, pero sin el primer renglón ni la primera columna.

Y como:

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \cdot & & & \\ \cdot & & 0 & \\ \cdot & & & \\ a_{i_r,j} & & & \end{pmatrix} + \overline{U}_j$$

Entonces:

$$U_j = -\sum_{k \in r[j]-(j)} \begin{pmatrix} l_{i_1,k} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,k} \end{pmatrix} (l_{i_1,k}, \ldots, l_{i_r,k}) -$$

$$- \begin{pmatrix} l_{i_1,j} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,j} \end{pmatrix} (l_{i_1,j}, \ldots, l_{i_r,j}) =$$

$$= -\sum_{k \in r[j]} \begin{pmatrix} l_{i_1,k} \\ \cdot \\ \cdot \\ \cdot \\ l_{i_r,k} \end{pmatrix} (l_{i_1,k}, \ldots, l_{i_r,k})$$

La relación existente entre el conjunto de matrices frontales y matrices de actualización se desprende de la siguiente definición.

Sea A una matriz de nxn y R una matriz de rxr, con $(i_1, \ldots, i_r)$ un conjunto de índices asociado a r con $i_1 \leq \ldots \leq i_r$ y en donde i_j representa el j-ésimo renglón y j-ésima columna de A.

Si tenemos dos matrices R y S de rxr y sxs y conjuntos de

índices $\{i_1,\ldots,i_r\}$, $\{j_1,\ldots,j_s\}$ respectivamente, al realizar la unión de dichos cojuntos de índices obtenemos

$$\{k_1,\ldots,k_t\} = \{i_1,\ldots,i_r\} \cup \{j_1,\ldots,j_s\}$$

con $$k_1 \leq \ldots \leq k_t$$

Entonces "extendemos" R agregando renglones y columnas de ceros en los lugares que se encuentren en $\{k_1,\ldots,k_t\}$ pero que no se encuentren en $\{i_1,\ldots,i_r\}$ y hacemos lo mismo con S sumando ambas matrices obtenidas en una operación llamada suma extendida

$$R \overset{\uparrow}{\underset{\downarrow}{\leftarrow\rightarrow}} S$$

que va a ser una matriz de txt con conjunto de índices $\{k_1,\ldots,k_t\}$.

Ejemplo

Si $R=\begin{pmatrix} p & q \\ u & v \end{pmatrix}$ con conjunto de índices {5,8}

$S=\begin{pmatrix} w & x \\ y & z \end{pmatrix}$ con conjunto de índices {5,9}

$$R \overset{\uparrow}{\underset{\downarrow}{\leftarrow\rightarrow}} S = \begin{matrix} 5 \\ 8 \\ 9 \end{matrix}\begin{pmatrix} p & q & 0 \\ u & v & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} w & 0 & x \\ 0 & 0 & 0 \\ y & 0 & z \end{pmatrix} = \begin{matrix} 5 \\ 8 \\ 9 \end{matrix}\begin{pmatrix} p+w & q & x \\ u & v & 0 \\ y & 0 & z \end{pmatrix}$$

con conjunto de índices {5,8,9}.

La relación entre el conjunto de matrices frontales y matrices

de actualización queda establecida en el siguiente teorema.

Teorema 3.2.2

Sean los nodos $c_1,\ldots,c_s$ los hijos de j en el árbol de eliminación, entonces:

$$F_j=\begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & 0 & \\ a_{i_r,j} & & & \end{pmatrix} \leftarrow\!\!\uparrow\!\!\rightarrow U_{c_1} \leftarrow\!\!\uparrow\!\!\rightarrow \ldots \leftarrow\!\!\uparrow\!\!\rightarrow U_{c_s}$$

Demostración

Por definición:

$$F_j=\begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & 0 & \\ a_{i_r,j} & & & \end{pmatrix} +\overline{U}_j$$

y

$$\overline{U}_j=-\sum_{k\in T[j]-\{j\}} \begin{pmatrix} l_{j,k} \\ l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{pmatrix} (l_{j,k}, l_{i_1,k},\ldots,l_{i_r,k})$$

Pero $T[j]-j = T[c_1] \cup \ldots \cup T[c_s]$ por ser $c_1,\ldots,c_s$ hijos de j

además $T[c_i] \cap T[c_j] \quad \forall \ i \neq j$.

De esto se puede expresar $\bar{U}_j$ como una suma extendida de las matrices U_{c_i} por el teorema anterior, ya que cada U_{c_i} se obtiene por elementos de $T[c_i]$.

Este resultado es la base del método multifrontal y del que surge el siguiente algoritmo para obtener la factorización de Choleski.

Algoritmo 7

1. Para toda columna j = 1, ..., n

   1.1 Sea $j, i_1, \ldots, i_r$ los elementos distintos de cero en L_{*j}.

   1.2 Sea $c_1, \ldots, c_s$ los hijos de j en el árbol de eliminación.

   1.3 Calcular

$$\bar{U} = U_{c_1} \leftarrow\!\downarrow\!\rightarrow \ldots \leftarrow\!\downarrow\!\rightarrow U_{c_s}$$

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & 0 & \\ a_{i_r,j} & & & \end{pmatrix} \leftarrow\!\downarrow\!\rightarrow \bar{U}$$

   1.4 Factorizar F_j en:

$$F_j = \begin{pmatrix} l_{j,j} & 0 \\ l_{i_1,j} & \\ \vdots & I \\ l_{i_r,j} & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & U_j \end{pmatrix} \begin{pmatrix} l_{j,j} & l_{i_1,j} \cdots l_{i_r,j} \\ 0 & I \end{pmatrix}$$

Consideremos el siguiente factor de Cholesky:

$$L = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{pmatrix} X & & & & & & & & \\ & X & & & & & & & \\ & & X & & & & & & \\ & X & & X & & & & & \\ & & X & & X & & & & \\ & X & \otimes & X & X & & & & \\ X & & & & & X & & & \\ X & & X & X & X & \otimes & X & X & \\ X & & & X & & X & X & \otimes & X \end{pmatrix}$$

Para las columnas 1, 2 y 3, las matrices frontales sólo toman los elementos distintos de cero de la matriz original. Para la columna 4 necesita de la columna 2:

$$F_4 = \begin{pmatrix} a_{44}-l_{42}^2 & -l_{42}l_{62} & a_{48} & a_{49} \\ -l_{42}l_{62} & -l_{62}^2 & 0 & 0 \\ a_{84} & 0 & 0 & 0 \\ a_{94} & 0 & 0 & 0 \end{pmatrix}$$

Esto ocurre con todas las columnas que, en el árbol de eliminación tiene hijos, entonces para la columna 6que utiliza a 4 y 5:

$$F_6 = \begin{pmatrix} a_{66}-l_{62}^2-l_{64}^2-l_{65}^2 & -l_{64}l_{84}-l_{65}l_{85} & a_{69}-l_{64}l_{94} \\ -l_{64}l_{84}-l_{65}l_{85} & -l_{83}^2-l_{84}^2-l_{85}^2 & -l_{84}l_{94} \\ a_{69}-l_{64}l_{94} & -l_{84}l_{96} & -l_{94}^2 \end{pmatrix}$$

Y con cada matriz frontal se obtiene la columna

correspondiente al realizar el primer paso de la factorización de Cholesky.

Ejemplo

Sea

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 4 & 0 & 4 \\ 1 & 0 & 10 & 9 \\ 0 & 4 & 9 & 29 \end{pmatrix}$$

1. i=1

$$F_1 = \begin{matrix} 1 \\ 3 \end{matrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$U_1 = -1$$

con renglón 3.

2. i=2

$$F_2 = \begin{matrix} 2 \\ 4 \end{matrix} \begin{pmatrix} 4 & 4 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -4 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix}$$

$$U_2 = -4$$

con renglón 4.

3. i=3;

$$F_3 = \begin{matrix} 3 \\ 4 \end{matrix} \begin{pmatrix} 10 & 9 \\ 9 & 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 9 \\ 9 & 0 \end{pmatrix}$$

$$F_3 = \begin{matrix} 3 \\ 4 \end{matrix} \begin{pmatrix} 9 & 9 \\ 9 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -9 \end{pmatrix} \begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}$$

$$U_3 = -9$$

con renglón 4.

4. i=4

$$F_4 = 29-4-9 = 16 = 4 \cdot 4$$

3.3 Postordenamiento

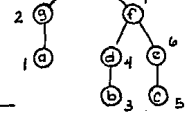
En el método multifrontal las columnas son procesadas en orden creciente de 1 a n; las matrices de actualización $\{U_j\}$ son producidas en ese orden. Si U_j no se usa inmediatamente para formar F_{j+1} se necesita almacenar U_j temporalmente hasta que sea usada.

Para una implementación efectiva del método multifrontal es necesario optimizar el almacenamiento.

Para esto definiremos lo que es un postordenamiento.

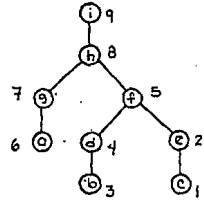
Sea A una matriz simétrica y rala, y T(A) su árbol de eliminación. Un *postordenamiento* es un ordenamiento topólogico[1], tal que los nodos en cada subárbol de T(A) son numerados consecutivamente.

Ejemplo



[1] Un ordenamiento topólogico es un ordenamiento en que los nodos hijos se numeran antes que los nodos padre.

Un postordenamiento requiere que todos los nodos dentro del subárbol sean numerados consecutivamente. Pero para varios subárboles que tienen el mismo nodo padre, en el árbol de eliminación, se tiene la libertad de ordenar los nodos en un subárbol, antes que otros. Con el ejemplo anterior T[g] y T[f] tienen el mismo nodo padre h, entonces se pueden numerar los nodos de T[g] y T[f] con números anteriores a ocho.



Liu[3] proporciona un algoritmo para reordenar los nodos, para obtener un postordenamiento óptimo, con el mismo árbol de eliminación.

Consideremos

$$\Delta(t) = \frac{1}{2}t(t+1)$$

esto es el total necesario para almacenar una matriz triangular inferior.

Sea x_i un nodo de T(A) y $Wstore(x_i)$ el mínimo almacenamiento necesario para ejecutar en forma efectiva la eliminación del

---

[3] J. W. H. Liu, On the storage requirement in the out-of-core multifrontal method for sparse factorization, ACM Trans. Math. Software, 12 (1986) p.1190-1211.

nodo x_i en el método multifrontal.

Si r_i es el número de elementos distintos de cero debajo de la diagonal en la columna i, y si x_i es una hoja[3] entonces

$$Wstore(x_i) = \Delta(r_i+1)$$

que es lo necesario para almacenar F_i.

Supongamos que x_i tiene hijos $x_{s_1}, x_{s_2}, \ldots, x_{s_t}$ en el árbol. Para calcular el almacenamiento requerido antes de eliminar x_i se tiene

$$Wstore(x_i) = \max_k \left\{ \max \{ Wstore(x_{s_k}), \Delta(r_i+1) \} + \sum_{j=1}^{k-1} \Delta(r_{s_j}) \right\}$$

El algoritmo es el siguiente:

Algoritmo 8

1. Para i=1,....n

   1.1. Si x_i no tiene hijos

   $$Wstore(x_i) = \Delta(r_i+1)$$

   1.2. Si tiene hijos

   1.2.1 Reordenar los nodos hijos $x_{s_1}, x_{s_2}, \ldots, x_{s_t}$ en forma descendiente de acuerdo a la fórmula $\{ \max \{ Wstore(x_{s_k}), \Delta(r_i+1) \} - \Delta(r_{s_k}) \}$

---

[3] Una hoja es un nodo que no es la raíz, y que tiene grado uno en un árbol.

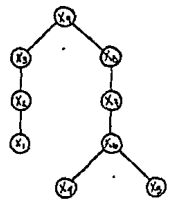
1.2.2 C a l c u l a r

$$Wstore(x_i) = \max_k \left\{ \max \{ Wstore(x_{s_k}), \Delta(r_i+1) \} \right.$$

2. Para i=n,n-1,...,1 generar el nuevo reordenamiento en el árbol.

Ejemplo

$$A = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{bmatrix} X & X & X & & & & & & X \\ X & X & X & & & & & & X \\ X & X & X & & & & & & X \\ & & & X & & X & X & & X \\ & & & & X & X & X & X & X \\ & & & X & X & X & X & X & X \\ & & & X & X & X & X & X & X \\ & & & & X & X & X & X & X \\ X & X & X & X & X & X & X & X & X \end{bmatrix}$$



$Wstore(x_1) = Wstore(x_2) = Wstore(x_3) = \Delta(r_1+1) = 10$

$Wstore(x_4) = \Delta(r_4+1) = 10$

$Wstore(x_5) = \Delta(r_5+1) = 15$

Para el nodo x_6 con nodos hijos x_4 y x_5

$\max \{ Wstore(x_4), \Delta(r_6+1) \} - \Delta(r_4) = 10-6=4$

$\max \{ Wstore(x_5), \Delta(r_6+1) \} - \Delta(r_5) = 15-10=5$

el nodo x_5 debe reordenarse primero que x_4. Con esta nueva secuencia $Wstore(x_6) = 20$. De ahí, al ser x_6 el único hijo de x_7, y este el único de x_8, $Wstore(x_7) = 20$ $Wstore(x_8) = 20$.

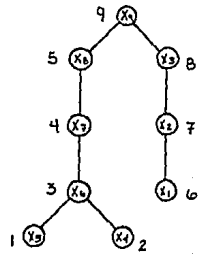
Finalmente, para la raíz x_9, con hijos x_3 y x_8

$\max \{ Wstore(x_3), \Delta(r_9+1) \} - \Delta(r_3) = 10-1=9$

$\max \{ Wstore(x_8), \Delta(r_9+1) \} - \Delta(r_8) = 20-1=19$

y x_8 debe numerarse antes que x_3 en la secuencia de hijos de x_9 y $Wstore(x_9) = 20$.

Generando el nuevo orden en el árbol de eliminación como sigue: $x_9, x_3, x_2, x_1, x_8, x_7, x_6, x_4, x_5$



RESULTADOS Y CONCLUSIONES

De los temas estudiados se probaron tres ejemplos de matrices de orden 35, 40 y 45 respectivamente.

De los ordenamientos utilizados (el de deficiencia mínima y el de grado mínimo), se encontró que el resultado obtenido es prácticamente el mismo en ambos casos, a diferencia de que el algoritmo de grado mínimo ordena más rápido.

De los métodos de factorización numérica, el frontal y el multifrontal, se llegó a la conclusión que en equipos con capacidad de memoria pequeña, el método frontal es muy efectivo, a diferencia del multifrontal que en ocasiones no puede encontrar la solución, debido a la falta de memoria.

Sin embargo, si se tuviera un equipo con gran capacidad de memoria, el método multifrontal promete ser muy efectivo, aunado al postordenamiento, ya que a diferencia del frontal tiene mayor facilidad para localizar datos y obtener buenas soluciones en un tiempo relativamente corto de acuerdo al número de elementos distintos de cero en la matriz.

Por lo tanto, si se quieren resolver matrices no tan grandes y se tiene un equipo con capacidad de memoria limitada, el

método frontal sería una buena opción.

Si se tiene gran capacidad de memoria, el método multifrontal sería el adecuado.

En ambos casos, de acuerdo a los dos algoritmos de ordenamiento expuestos, el de grado mínimo sería una mejor opción.

APENDICE

La instrumentación computacional de los algoritmos presentados en este trabajo se presentan a continuación en procedimientos desarrollados en lenguage C. El esquema al final, señala el orden para dicha instrumentación de acuerdo al método a ser utilizado.

GENGRA

Sea A una matriz rala simétrica y definida positiva. Supongamos que A ha sido almacenadas por columnas, en donde únicamente figuran los elementos $a_{ij} \neq 0$ con $i \geq j$.

El procedimiento GENGRA construye la estructura de adyacencia de la gráfica G(A) asociada a la matriz A.

```
/*************************************************************/
/*                          GENGRA                           */
/*************************************************************/
/* GENERA LA ESTRUCTURA DE ADYACENCIA DE UNA GRAFICA ASO-    */
/* CIADA A UNA MATRIZ SIMETRICA DEFINIDA POSITIVA            */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    IR - ARREGLO DE UNA DIMENSION QUE CONTIENE EL INDI-    */
/*         CE DE RENGLON DE CADA ELEMENTO DISTINTO DE CE-    */
/*         RO AIJ, EN DONDE I ES MAYOR O IGUAL QUE J.        */
/*    AD - ARREGLO DE UNA DIMENSION DE LONGITUD EL NUMERO    */
/*         DE ECUACIONES MAS UNO. CONTIENE APUNTADORES AL    */
/*         PRINCIPIO Y AL FINAL DE CADA COLUMNA              */
/*    NEC - NUMERO DE ECUACIONES.                            */
/* PARAMETROS DE SALIDA                                      */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*         VECINDAD DE CADA VERTICE.                         */
```

```
/*     XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE */
/*          CADA VECINDAD.                                 */
/***********************************************************/
void gengra (int *ir, int *ad, int *adjncy, int *xadj,
             int nec)
{
 int l=1;
 int i=1;
 int n,j,nv,k,k1,m;
 xadj[i]=1;
 n=ad[i+1]-ad[i];
 if (n!=1)
   {
    for (j=ad[i]+1; j<=ad[i+1]-1; j++)
      {
       adjncy[l]=ir[j];
       l++;
       }
    xadj[i+1]=xadj[i]+n-1;
    }
 else
    xadj[i+1]=xadj[i];
 for (i=2; i<=nec;i++)
    {
     nv=0;
     for (k=1; k<=i-1; k++)
     {
       k1=xadj[k+1]-xadj[k];
       if (k1!=0)
       for (m=xadj[k];m<=xadj[k+1]-1;m++)
          {
           if (adjncy[m] > i) break;
           if (adjncy[m] < i) continue;
           adjncy[l]=ir[ad[k]];
           nv++;
           l++;
           m=xadj[k+1]-1;
          }
       }
          n=ad[i+1]-ad[i];
          if (n!=1)
          for (j=ad[i]+1;j<=ad[i+1]-1;j++)
             {
              adjncy[l]=ir[j];
              l++;
             };
          xadj[i+1]=xadj[i]+nv+n-1;
    }
}
```

VECPADRE

Este procedimiento construye el árbol de eliminación T(A) asociado a la matriz A, en forma de un vector que contiene los padres de cada nodo. De acuerdo al algoritmo descrito en el primer capítulo sólo necesita de la matriz A y en este caso de G(A).

```
/*************************************************************/
/*                        VECPADRE                           */
/*************************************************************/
/* GENERA EL ARBOL DE ELIMINACION DE UNA GRAFICA ASOCIADA    */
/* A UNA MATRIZ SIMETRICA DEFINIDA POSITIVA Y RALA           */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*         VECINDAD DE CADA VERTICE.                         */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*         CADA VECINDAD.                                    */
/*    NEC - NUMERO DE ECUACIONES.                            */
/* PARAMETROS DE SALIDA                                      */
/*    PADRE - ARREGLO DE UNA DIMENSION QUE CONTIENE AL       */
/*      PADRE DE CADA NODO EN ELARBOL DE ELIMINACION.        */
/*************************************************************/
void vecpadre (int *adjncy, int *xadj, int *padre,int nec)
{
  int i,k,r;
  for (i=1; i<=nec;i++)
    {
     padre[i]=0;
     for (k=xadj[i]; k<=xadj[i+1]-1; k++)
     if (adjncy[k] < i)
       {
         r=adjncy[k];
         while (padre[r]!=0 && padre[r]!=i)
            r=padre[r];
         if (padre[r]==0)
            padre[r]=i;
       }
     }
}
```

VECNOZERO

Este procedimiento cuenta el número de elementos distintos de cero para cada columna, utilizando el árbol de eliminación (vector PADRE).

```
/*************************************************************/
/*                          VECNOZERO                        */
/*************************************************************/
/* GENERA EL NUMERO DE ELEMENTOS DISTINTOS DE CERO PARA      */
/* COLUMNA DEL FACTOR DE CHOLESKY L                          */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*             VECINDAD DE CADA VERTICE.                     */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*           CADA VECINDAD.                                  */
/*    PADRE - ARREGLO DE UNA DIMENSION QUE CONTIENE AL       */
/*      PADRE DE CADA NODO EN EL ARBOL DE ELIMINACION.       */
/*    NEC - NUMERO DE ECUACIONES.                            */
/* PARAMETROS DE SALIDA                                      */
/*    NOZERO - ARREGLO DE UNA DIMENSION QUE CONTIENE EL      */
/*           NUMERO DE ELEMENTOS DISTINTOS DE CERO PARA CA-  */
/*           COLUMNA DE L.                                   */
/*************************************************************/
void vecnozero(int *adjncy, int *xadj, int *padre,
               int *nozero,int nec)
{
  int *marca,i,k,j;
  marca=calloc(nec,sizeof(int));
  for (j=1; j<=nec;j++)
     nozero[j]=1;
  for (i=1; i<=nec;i++)
    {
     marca[i]=i;
     for (k=xadj[i]; k<=xadj[i+1]-1; k++)
     if (adjncy[k] < i)
       {
        j=adjncy[k];
        while(marca[j]!=i)
            {
             nozero[j]++;
             marca[j]=i;
             j=padre[j];
            }
       }
    }
```

)

PERMUT

Estos procedimientos PERMUT1 y PERMUT2, ordenan la matriz original A, por el método de grado mínimo (PERMUT1) y por el método de deficiencia mínima (PERMUT2), descritos en el capítulo 2. Cada uno de estos procedimientos, tienen procedimientos equivalentes que se mencionarán a continuación.

Los procedimientos GRAMI y DEMI, obtienen un vector que contiene los grados y las deficiencias de cada nodo así como el nodo con el grado y deficiencia mínima.

MOGRAD y MODEV modifican los arreglos de grado y deficiencia mínima en cada paso, es decir actualizan los vectores al eliminar un nodo.

Las funciones VEGM y VEDM actualizan el nuevo vértice de grado mínimo y deficiencia mínima, después de la eliminación del anterior.

```
/*************************************************************/
/*                        PERMUT1                            */
/*************************************************************/
/* ENCUENTRA LA PERMUTACION POR EL METODO DE GRADO MINIMO */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*          VECINDAD DE CADA VERTICE.                        */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*          CADA VECINDAD.                                   */
/*    NEC - NUMERO DE ECUACIONES.                            */
/* PARAMETROS DE SALIDA                                      */
/*    PERM - ARREGLO DE UNA DIMENSION QUE CONTIENE LA        */
/*          PERMUTACION PARA CADA NODO.                      */
```

```
/*    GRAD - ARREGLO DE UNA DIMENSION QUE CONTIENE EL      */
/*           GRADO DE CADA NODO.                           */
/*    GM - ENTERO QUE APUNTA AL NODO DE GRADO MINIMO       */
/*********************************************************/
void permut1(int *perm,int *xadj,int *adjncy,int *grad,
        int gm,int nec)
{
 int i;
 gm=grami(xadj,grad,nec);
 perm[1]=gm;
 grad[gm]=-1;
 mograd(xadj,adjncy,grad,gm);
 for(i=2;i<=nec;i++)
 {
  gm=vegm(grad,nec);
  perm[i]=gm;
  grad[gm]=-1;
  mograd(xadj,adjncy,grad,gm);
 }
}


/*********************************************************/
/*                        MOGRAD                         */
/*********************************************************/
/* MODIFICA EL ARREGLO DE GRADOS                         */
/*********************************************************/
/* PARAMETROS DE ENTRADA                                 */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA  */
/*           VECINDAD DE CADA VERTICE.                   */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE */
/*           CADA VECINDAD.                              */
/*    GM - ENTERO QUE APUNTA AL NODO DE GRADO MINIMO     */
/* PARAMETROS DE ACTUALIZACION                           */
/*    GRAD - ARREGLO DE UNA DIMENSION QUE CONTIENE EL    */
/*           GRADO DE CADA NODO.                         */
/*********************************************************/
void mograd(int *xadj,int *adjncy,int *grad,int gm)
{
 int i,j,m;
 i=gm;
 for (j=xadj[i]; j<=xadj[i+1]-1;j++)
    {
      m=adjncy[j];
      if (grad[m]==-1) continue;
      grad[m]--;
    }
}
```

```
/*************************************************************/
/*                    FUNCION VEGM                           */
/*************************************************************/
/* MODIFICA EL GRADO MINIMO                                  */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    GRAD - ARREGLO DE UNA DIMENSION QUE CONTIENE EL        */
/*         GRADO DE CADA NODO.                               */
/*    NEC - NUMERO DE ECUACIONES                             */
/* PARAMETROS DE ACTUALIZACION                               */
/*    GM - ENTERO QUE APUNTA AL NODO DE GRADO MINIMO         */
/*************************************************************/
int vegm(int *grad,int nec)
{
 int i,gm;
 int md=0;
 for (i=1;i<=nec; i++)
    {
     if(grad[i]==-1) continue;
     md++;
     if (md==1)
       {
     gm=i;
     continue;
       }
     if(grad[i] < grad[gm]) gm=i;
    }
return(gm);
}

/*************************************************************/
/*                        FUNCION GRAMI                      */
/*************************************************************/
/* OBTIENE EL ARREGLO DE GRADOS Y EL GRADO MINIMO            */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*         CADA VECINDAD.                                    */
/*    NEC - NUMERO DE ECUACIONES.                            */
/* PARAMETROS DE SALIDA                                      */
/*    GRAD - ARREGLO DE UNA DIMENSION QUE CONTIENE EL        */
/*         GRADO DE CADA NODO.                               */
/*    GM - ENTERO QUE APUNTA AL NODO DE GRADO MINIMO         */
/*************************************************************/
int grami (int *xadj,int *grad,int nec)
{
 int i,n;
 int gm=1;
 for (i=1;i<=nec; i++)
    {
```

```
    n=xadj[i+1]-xadj[i];
    grad[i]=n;
    if (grad[i] < grad[gm]) gm=i;
   }
 return(gm);
}


/*************************************************************/
/*                          PERMUT2                          */
/*************************************************************/
/* ENCUENTRA LA PERMUTACION POR EL METODO DE DEFICIENCIA     */
/* MINIMA                                                    */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*          VECINDAD DE CADA VERTICE.                        */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*          CADA VECINDAD.                                   */
/*    NEC - NUMERO DE ECUACIONES.                            */
/* PARAMETROS DE SALIDA                                      */
/*    PERM - ARREGLO DE UNA DIMENSION QUE CONTIENE LA        */
/*          PERMUTACION PARA CADA NODO.                      */
/*    DV - ARREGLO DE UNA DIMENSION QUE CONTIENE LA          */
/*          DEFICIENCIA DE CADA NODO.                        */
/*    DM - ENTERO QUE APUNTA AL NODO DE DEFICIENCIA          */
/*          MINIMA                                           */
/*************************************************************/
void permut2(int *perm,int *xadj,int *adjncy,int *dv,int dm,
        int nec)
{
 int i;
 dm=demi(xadj,adjncy,dv,nec);
 perm[1]=dm;
 dv[dm]=-1;
 modev(xadj,adjncy,dv,dm);
 for(i=2;i<=nec;i++)
 {
  dm=vedm(dv,nec);
  perm[i]=dm;
  dv[dm]=-1;
  modev(xadj,adjncy,dv,dm);
 }
}


/*************************************************************/
/*                          MODEV                            */
/*************************************************************/
/* MODIFICA EL ARREGLO DE DEFICIENCIA                        */
/*************************************************************/
```

```
/* PARAMETROS DE ENTRADA                                          */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA           */
/*          VECINDAD DE CADA VERTICE.                             */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE         */
/*          CADA VECINDAD.                                        */
/*    DM - ENTERO QUE APUNTA AL NODO DE DEFICIENCIA               */
/*         MINIMA                                                 */
/* PARAMETROS DE ACTUALIZACION                                    */
/*    DM - ARREGLO DE UNA DIMENSION QUE CONTIENE LA               */
/*          DEFICIENCIA DE CADA NODO.                             */
/****************************************************************/
void modev(int *xadj,int *adjncy,int *dv,int dm)
{
 int i,j,k,m,l,v,w,marca;
 i=dm;
 for (j=xadj[i]; j<=xadj[i+1]-1;j++)
    {
     m=adjncy[j];
     if (dv[m]==-1) continue;
     for (k=xadj[m]; k<=xadj[m+1]-1; k++)
     {
      l=adjncy[k];
      if(dv[l]==-1) continue;
      marca=0;
      for (v=xadj[l]; v<=xadj[l+1]-1;v++)
          {
           w=adjncy[v];
           if (m==w) continue;
           if (i==w)
          {
           marca=1;
           break;
          }
          }
      if (marca==0) dv[m]--;
     }
    }
}


/****************************************************************/
/*                  FUNCION VEDM                                */
/****************************************************************/
/* MODIFICA LA DEFICIENCIA MINIMA                               */
/****************************************************************/
/* PARAMETROS DE ENTRADA                                        */
/*    DV - ARREGLO DE UNA DIMENSION QUE CONTIENE LA             */
/*          DEFICIENCIA DE CADA NODO.                           */
/*    NEC - NUMERO DE ECUACIONES                                */
/* PARAMETROS DE ACTUALIZACION                                  */
/*    DM - ENTERO QUE APUNTA AL NODO DE DEFICIENCIA             */
```

```
/*          MINIMO                                                  */
/************************************************************/
int vedm(int *dv,int nec)
{
 int i,dm;
 int md=0;
 for(i=1;i<=nec;i++)
 {
  if(dv[i]==0)
   {
    dm=i;
    break;
   }
  if(dv[i]==-1) continue;
  md++;
  if (md==1)
    {
     dm=i;
     continue;
     }
  if(dv[i] < dv[dm]) dm=i;
 }
return(dm);
}


/************************************************************/
/*                      FUNCION DEMI                        */
/************************************************************/
/* OBTIENE EL ARREGLO DE DEFICIENCIAS Y LA DEFICIENCIA      */
/* MINIMA                                                   */
/************************************************************/
/* PARAMETROS DE ENTRADA                                    */
/*     XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE  */
/*          CADA VECINDAD.                                  */
/*     NEC - NUMERO DE ECUACIONES.                          */
/* PARAMETROS DE SALIDA                                     */
/*     DM - ARREGLO DE UNA DIMENSION QUE CONTIENE LA        */
/*          DEFICIENCIA DE CADA NODO.                       */
/*     DM - ENTERO QUE APUNTA AL NODO DE DEFICIENCIA        */
/*          MINIMA                                          */
/************************************************************/
int demi (int *xadj,int *adjncy,int *dv,int nec)
{
 int i,j,n,k,l,m,v,marca;
 int dm=1;
 for (i=1;i<=nec; i++)
    {
     dv[i]=0;
     n=xadj[i+1]-xadj[i];
     if (n>1)
```

```
        {
    for (j=xadj[i]; j<=xadj[i+1]-1;j++)
        {
         m=adjncy[j];
         for (k=j+1; k<=xadj[i+1]-1; k++)
             {
             marca=0;
             l=adjncy[k];
             for (v=xadj[m]; v<=xadj[m+1]-1;v++)
                if (l==adjncy[v])
                   {
                    marca=1;
                    break;
                   };
             if (marca==0) dv[i]++;
              }
        }
      }
    if (dv[i] < dv[dm]) dm=i;
   }
 return(dm);
}
```

FNROOT

Este función encuentra un nodo pseudoperiférico. Considera a ROOT como nodo inicial. Trabaja sobre la estructura de nivel construida por ROOT y elige del último nivel un nodo v y construye su estructura de nivel. Si la estructura con raíz en v tiene más niveles que la cimentada en ROOT, elige un vértice en el último nivel de N(v) y construye la estructura correspondiente. En caso de que el número de niveles en N(v) sea menor que los de N(ROOT), se detiene el proceso. De otra forma se continúa hasta encontrar un vértice en el último nivel, que de origen a una estructura de nivel con más niveles que N(v).

```
/*************************************************************/
/*                  FUNCION FNOOT                            */
/*************************************************************/
/* ENCUENTRA NODOS PSEUDOPERIFERICOS. DETERMINA NODOS        */
/* LA SECCION DE LA SUBGRAFICA ESPECIFICADA POR MASK Y       */
/* ROOT.                                                     */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*     ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA     */
/*          VECINDAD DE CADA VERTICE.                        */
/*     XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE   */
/*          CADA VECINDAD.                                   */
/*     MASK - SE USA PARA ESPECIFICAR UNA SECCION DE LA      */
/*          GRAFICA. NODOS CON MASK(I)=0 SE IGNORAN          */
/*     NEC - NUMERO DE ECUACIONES                            */
/*     (XLS,LS) - CONTIENE LA ESTRUCTURA DE NIVEL CIMENTA-   */
/*           DA EN ROOT. XLS ES UN VACTOR DE POSICION DE     */
/*           LA ESTRUCTURA DE NIVEL Y LS ES UN ARREGLO DE    */
/*           UNA DIMENSION QUE GUARDA EN ORDEN SUCESIVO      */
/*           LOS NOMBRES DE LOS VERTICES DE CADA NIVEL.      */
/*************************************************************/
int fnroot(int *xadj,int *adjncy,int *mask,
       int *xls,int *ls,int nec)
{
 int j,k,jstrt,mindeg,node,ndeg,nlvl,nunlvl,ccsize;
 int kstop,kstrt,nabor,root;
 nlvl=rootls(1,xadj,adjncy,mask,xls,ls, nec);
 ccsize=xls[nlvl]-1;
 do
  {
   jstrt=xls[nlvl];
   mindeg=ccsize;
   root=ls[jstrt];
   if (ccsize != jstrt)
   for(j=jstrt;j<=ccsize;j++)
      {
       node=ls[j];
       ndeg=0;
       kstrt=xadj[node];
       kstop=xadj[node+1]-1;
       for (k=kstrt;k<=kstop;k++)
       {
        nabor =adjncy[k];
        if (mask[nabor]>0) ndeg++;
       }
       if(ndeg >= mindeg) continue;
       root=node;
       mindeg=ndeg;
      }
   nunlvl=rootls(root,xadj,adjncy,mask,xls,ls, nec);
   if (nunlvl<=nlvl) break;
```

```
   nlvl=nunlvl;
 }while(nlvl<ccsize);
 return(nlvl);
}
```

ROOTLS

Esta función obtiene el número de niveles de la estructura de nivel cimentada en ROOT, además de construir ésta. El arreglo MASK indica cuales nodos pertenecen a la misma componente que ROOT.

```
/*********************************************************/
/*                  FUNCION ROOTLS                       */
/*********************************************************/
/* GENERA LA ESTRUCTURA DE NIVEL CIMENTADA EN EL VERTICE  */
/* ROOT, UNICAMENTE LOS NODOS PARA LOS CUALES MASK NO ES  */
/* CERO SE CONSIDERAN.                                    */
/* TAMBIEN OBTIENE EL NUMERO DE NIVELES.                  */
/*********************************************************/
/* PARAMETROS DE ENTRADA                                  */
/*    ROOT - EL NODO EN EL CUAL SE CIMENTA LA ESTRUCTURA  */
/*         DE NIVEL.                                      */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA   */
/*         VECINDAD DE CADA VERTICE.                      */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE */
/*         CADA VECINDAD.                                 */
/*    MASK - SE USA PARA ESPECIFICAR UNA SECCION DE LA    */
/*         GRAFICA. NODOS CON MASK(I)=0 SE IGNORAN        */
/*    NEC - NUMERO DE ECUACIONES                          */
/* PARAMETROS DE SALIDA                                   */
/*    NLVL - CUENTA EL NUMERO DE NIVELES                  */
/*    (XLS,LS) - CONTIENE LA ESTRUCTURA DE NIVEL CIMENTA- */
/*          DA EN ROOT. XLS ES UN VACTOR DE POSICION DE   */
/*          LA ESTRUCTURA DE NIVEL Y LS ES UN ARREGLO DE  */
/*          UNA DIMENSION QUE GUARDA EN ORDEN SUCESIVO    */
/*          LOS NOMBRES DE LOS VERTICES DE CADA NIVEL.    */
/*********************************************************/
int rootls(int root,int *xadj,int *adjncy,int *mask,
       int *xls,int *ls,int nec)
{
 int lvlend=0;
 int ccsize=1;
```

```
 int i,j,lbegin,node,jstrt,jstop,nbr,nlvl,lvsize;
 for (i=1;i<=nec;i++) mask[i]=1;
 mask[root]=0;
 ls[1]=root;
 nlvl=0;
 do
    {
/*-------------------------------------------------------------*/
/* LBEGIN ES UN APUNTADOR AL PRINCIPIO DEL NIVEL CORRIENTE     */
/* LVLEND APUNTA AL FINAL DE DICHO NIVEL                       */
/*-------------------------------------------------------------*/
     lbegin=lvlend + 1;
     lvlend=ccsize;
     nlvl++;
     xls[nlvl]=lbegin;
/*-------------------------------------------------------------*/
/* GENERA EL NIVEL SIGUIENTE                                   */
/*-------------------------------------------------------------*/
     for (i=lbegin;i<=lvlend;i++)
     {
      node=ls[i];
      jstrt=xadj[node];
      jstop=xadj[node+1]-1;
      if (jstop < jstrt) continue;
      for (j=jstrt;j<=jstop;j++)
          {
           nbr=adjncy[j];
           if (mask[nbr]==0) continue;
           ccsize++;
           ls[ccsize]=nbr;
           mask[nbr]=0;
          }
     }
/*-------------------------------------------------------------*/
/* CALCULA LA ANCHURA DEL NIVEL CORRIENTE                      */
/*-------------------------------------------------------------*/
       lvsize=ccsize-lvlend;
      }while (lvsize>0);
xls[nlvl+1]=lvlend+1;
for (i=1;i<=ccsize;i++)
   {
    node=ls[i];
    mask[node]=1;
   }
return(nlvl);
}
```

ESTRUCT

Este procedimiento obtiene la factorización simbólica, es decir obtiene la estructura de la matriz L representada por un arreglo que contiene los renglones en cada columna (ADJNCYT), y un arreglo que apunta al principio de cada columna (XADJT). Es una instrumentación del algoritmo visto en el capítulo 1.

```
/*************************************************************/
/*                          ESTRUCT                          */
/*************************************************************/
/* GENERA LA FACTORIZACION SIMBOLICA                         */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*         VECINDAD DE CADA VERTICE.                         */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*         CADA VECINDAD.                                    */
/*    PADRE - ARREGLO DE UNA DIMENSION QUE CONTIENE AL       */
/*      PADRE DE CADA NODO EN EL ARBOL DE ELIMINACION.       */
/*    NOZERO - ARREGLO DE UNA DIMENSION QUE CONTIENE EL      */
/*         NUMERO DE ELEMENTOS DISTINTOS DE CERO PARA CA-    */
/*         COLUMNA DE L.                                     */
/*    NEC - NUMERO DE ECUACIONES                             */
/*    NMAT - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN L      */
/* PARAMETROS DE SALIDA                                      */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*         ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA       */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL       */
/*         CADA COLUMNA DE L.                                */
/*************************************************************/
void estruct(int *xadj,int *adjncy,int *xadjt,int *adjncyt,
          int *padre,int *nozero,int nec, int nmat)
{
 int i,j,k,f,g,a,n,m,h;
 int l=1;
 int w=0;
 for (i=1;i<=nmat;i++)
    adjncyt[i]=0;
 for(j=1;j<=nec;j++)
    {
     m=w+1;
     w=w+nozero[j];
     xadjt[j]=m;
     adjncyt[m]=j;
```

```
      m++;
      for(i=xadj[j];i<=xadj[j+1]-1;i++)
        {
      if (adjncy[i]>j)
        {
         adjncyt[m]=adjncy[i];
         m++;
        }
      }
      for(i=1;i<j;i++)
      {
       n=padre[i];
       if (n!=j) continue;
       for (k=xadjt[i]+1;k<=xadjt[i+1]-1;k++)
           {
            l=adjncyt[k];
            if (j>=l) continue;
            for (f=xadjt[j]+1;f<=w;f++)
             {
              g=adjncyt[f];
              h=adjncyt[f-1];
              if (l==g) break;
              if (l>h && l<g)
             {
              for(a=m;a>f;a--)
                adjncyt[a]=adjncyt[a-1];
              adjncyt[f]=l;
              m++;
              break;
             }
              if (g==0 && h<l)
             {
              adjncyt[f]=l;
              m++;
              break;
             }
             }
           }
      }
    }
}
```

ROTAR

Este procedimiento obtiene un reordenamiento, con la rotación del árbol de eliminación. ROOT es el nodo que se desea como raíz, y por medio de este se van tomando los nodos ancestros

de ROOT para ser numerados después de él. Se obtiene un vector de permutaciones.

```
/*************************************************************/
/*                        ROTAR                              */
/*************************************************************/
/* GENERA EL ORDENAMIENTO DE LA MATRFIZ POR ROTACION DE      */
/* NODOS.                                                    */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ROOT - ELEMENTO DESEADO COMO RAIZ                      */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*         ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA       */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL      */
/*         CADA COLUMNA DE L.                                */
/*    PADRE - ARREGLO DE UNA DIMENSION QUE CONTIENE AL       */
/*      PADRE DE CADA NODO EN EL ARBOL DE ELIMINACION.       */
/*    NEC - NUMERO DE ECUACIONES                             */
/* PARAMETROS DE SALIDA                                      */
/*    PERM - NUEVO ORDEN DELOS NODOS YA ROTADOS              */
/*************************************************************/
void rotar(int root,int *perm,int *xadjt,int *adjncyt,
        int *padre,int nec)
{
 int *mark,i,z,w;
 int k=nec-1;
 mark=calloc(nec,sizeof(int));
 for (i=1;i<=nec;i++)
    mark[i]=0;
 for (i=1;i<=nec;i++)
    perm[i]=0;
 z=root;
 perm[z]=nec;
 mark[z]=1;
 while (z!=nec)
    {
     for (i=xadjt[z+1]-1;i>xadjt[z];i--)
     {
      w=adjncyt[i];
      if (mark[w]==0)
        {
         perm[w]=k;
         mark[w]=1;
         k--;
        }
     }
     z=padre[z];
    }
```

```
k=1;
for (i=1;i<=nec;i++)
   {
    if (mark[i]==0)
      {
     perm[i]=k;
     k++;
      }
    }
}
```

ORDENA

Si A es la matriz original, este procedimiento obtiene $M=PAP^t$ por medio de la permutación obtenida en el vector PERM. La matriz A esta representada por los arreglos JA, IA, AN y la matriz M pro sus correspondientes JAT, IAT, ANT.

```
/*************************************************************/
/*                          ORDENA                           */
/*************************************************************/
/* PERMUTA LA MATRIZ AIJ EN MIJ DE ACUERDO A LA PERMUTA-     */
/* CION DADA.                                                */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    JA - ARREGLO DE UNA DIMENSION QUE CONTIENE EL INDI-    */
/*         CE DE RENGLON DE CADA ELEMENTO DISTINTO DE CE-    */
/*         RO AIJ, EN DONDE I ES MAYOR O IGUAL QUE J.        */
/*    IA - ARREGLO DE UNA DIMENSION DE LONGITUD EL NUMERO    */
/*         DE ECUACIONES MAS UNO. CONTIENE APUNTADORES AL    */
/*         PRINCIPIO Y AL FINAL DE CADA COLUMNA              */
/*    AN - ARREGLO DE UNA DIMENSION QUE CONTIENE EL VALOR    */
/*         NUMERICO DE RENGLON DE CADA ELEMENTO DISTINTO     */
/*         DE CERO AIJ, EN DONDE I ES MAYOR O IGUAL QUE J.   */
/*    ADJNCY - ARREGLO DE UNA DIMENSION QUE CONTIENE LA      */
/*         VECINDAD DE CADA VERTICE.                         */
/*    XADJ - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL DE    */
/*         CADA VECINDAD.                                    */
/*    PERM - ARREGLO DE UNA DIMENSION QUE CONTIENE LA        */
/*         PERMUTACION PARA CADA NODO.                       */
/*    NEC - NUMERO DE ECUACIONES.                            */
/*    NUM - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN L       */
```

```
/* PARAMETROS DE SALIDA                                          */
/*     JAT - ARREGLO DE UNA DIMENSION QUE CONTIENE EL IN-        */
/*           DECE DE RENGLON DE CADA ELEMENTO DISTINTO DE        */
/*           CERO MIJ, EN DONDE I ES MAYOR O IGUAL QUE J Y       */
/*           M ES LA MATRIZ PERMUTADA                            */
/*     IAT - ARREGLO DE UNA DIMENSION DE LONGITUD EL NUME-       */
/*           RO DE ECUACIONES MAS UNO. CONTIENE APUNTADORES      */
/*           AL PRINCIPIO Y AL FINAL DE CADA COLUMNA DE M.       */
/*     ANT - ARREGLO DE UNA DIMENSION QUE CONTIENE EL VA-        */
/*           LOR NUMERICO DE RENGLON DE CADA ELEMENTO DIS-       */
/*           TINTO DE CERO MIJ, EN DONDE I ES MAYOR O            */
/*           IGUAL QUE J.                                        */
/*************************************************************/
void ordena(int *xadj,int *adjncy, int *ia,int *ja,
         float *an,int *iat,int *jat, float *ant,
         int *perm, int nec, int num)
{
 int i,j,p,q,a,l,f,w;
 int m=1;
 int k=1;
 for (i=1;i<=num;i++)
    {
     ant[i]=0.0;
     jat[i]=0;
    }
 do
 {
  for (i=1;i<=nec;i++)
     if (perm[i]==k)
     break;
  iat[k]=m;
  jat[m]=k;
  ant[m]=an[ia[i]];
  m++;
  for(j=xadj[i];j<=xadj[i+1]-1;j++)
     {
      l=adjncy[j];
      if(perm[l]<=k) continue;
      for(p=iat[k]+1;p<=m;p++)
        {
      q=jat[p];
      f=jat[p-1];
      if (perm[l]==q) break;
      if (perm[l]>f && perm[l]<q)
        {
          for(a=m;a>p;a--)
             {
              jat[a]=jat[a-1];
              ant[a]=ant[a-1];
             }
          jat[p]=perm[l];
```

```
        if (i<l)
          for (w=ia[i]+1;w<=ia[i+1]-1;w++)
          if (ja[w]==l)
            {
             ant[p]=an[w];
             break;
            }
        if(l<i)
          for (w=ia[l]+1;w<=ia[l+1]-1;w++)
          if (ja[w]==i)
            {
             ant[p]=an[w];
             break;
            }
        m++;
        break;
       }
     if (q==0 && f<perm[l])
       {
        jat[p]=perm[l];
        if (i<l)
          for (w=ia[i]+1;w<=ia[i+1]-1;w++)
          if (ja[w]==l)
            {
             ant[p]=an[w];
             break;
            }
        if (l<i)
          for (w=ia[l]+1;w<=ia[l+1]-1;w++)
            if (ja[w]==i)
              {
               ant[p]=an[w];
               break;
              }
        m++;
        break;
       }
     }
   }
k++;
}while(k!=nec+1);
}
```

PASAR

Este procedimiento pasa el valor numérico de los elementos distintos de cero, a la estructura de L representada por XADJT

y ADJNCYT. Los valores numéricos son depositados en NVAL.

```
/*************************************************************/
/*                          PASAR                            */
/*************************************************************/
/* PASA LOS ELEMENTOS DISTINTOS DE CERO A LA ESTRUCTURA      */
/* OBTENIDA EN LA FACTORIZACION SIMBOLICA.                   */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*          ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA      */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL      */
/*          CADA COLUMNA DE L.                               */
/*    JA - ARREGLO DE UNA DIMENSION QUE CONTIENE EL INDI-    */
/*          CE DE RENGLON DE CADA ELEMENTO DISTINTO DE CE-   */
/*          RO AIJ, EN DONDE I ES MAYOR O IGUAL QUE J.       */
/*    IA - ARREGLO DE UNA DIMENSION DE LONGITUD EL NUMERO    */
/*          DE ECUACIONES MAS UNO. CONTIENE APUNTADORES AL   */
/*          PRINCIPIO Y AL FINAL DE CADA COLUMNA             */
/*    AN - ARREGLO DE UNA DIMENSION QUE CONTIENE EL VALOR    */
/*          NUMERICO DE RENGLON DE CADA ELEMENTO DISTINTO    */
/*          DE CERO AIJ, EN DONDE I ES MAYOR O IGUAL QUE J.  */
/*    NEC - NUMERO DE ECUACIONES                             */
/*    NMAT - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN L      */
/* PARAMETROS DE SALIDA                                      */
/*    NVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO DE     */
/*          LOS RENGLONES APUNTADOS POR ADJNCYT.             */
/*************************************************************/
void pasar(int *xadjt,int *adjncyt,float *nval,int *ia,
        int *ja, float *an, int nmat, int nec)
{
 int i,j,k,l,m;
 for (i=1;i<=nmat;i++)
    nval[i]=0.0;
 for (i=1;i<=nec;i++)
    {
     for (j=ia[i];j<=ia[i+1]-1;j++)
     {
      l=ja[j];
      for(k=xadjt[i];k<=xadjt[i+1]-1;k++)
          {
           m=adjncyt[k];
           if(l!=m) continue;
           nval[k]=an[j];
           break;
          }
     }
    }
}
```

FRONTAL

Este procedimiento realiza la factorización numérica de Cholesky de una matriz simétrica, definida positiva y rala por el método frontal. La matriz frontal es reinicializada en cada paso, cambiando su tamaño de acuerdo al número de elementos distintos de cero en cada columna. Emplea tres subrutinas. COPIA que pasa los elementos de la estructura representada por XADJT, ADJNCYT, NVAL a FRONTVAL y FRONTREN para ser factorizados, el procedimiento CHOLE1 que realiza el primer paso de la factorización de Cholesky por submatrices, y el procedimiento REGCOPIA que regresa los valores a L.

```
/*************************************************************/
/*                        FRONTAL                            */
/*************************************************************/
/* OBTIENE LA FACTORIZACION NUMERICA POR CHOLESKY CON EL     */
/* METODO FRONTAL.                                           */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*          ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA      */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL      */
/*          CADA COLUMNA DE L.                               */
/*    NEC - NUMERO DE ECUACIONES                             */
/* PARAMETROS DE ACTUALIZACION                               */
/*    NVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO DE     */
/*          LOS RENGLONES APUNTADOS POR ADJNCYT.             */
/*************************************************************/
void frontal(int *xadjt,int *adjncyt, float *nval,int nec)
{
 int i,j,m,n;
 int *frontren;
 float *frontval;
 frontren=malloc(sizeof(int));
 frontval=malloc(sizeof(float));
 for(j=1;j<=nec;j++)
    {
     n=xadjt[j+1]-xadjt[j];
     m=n*(n+1)/2;
```

```
/*-------------------------------------------------------------*/
/* REASIGNA MEMORIA PARA LA MATRIZ FRONTAL Y LA INICIALIZA*/
/*-------------------------------------------------------------*/
     frontren=realloc(frontren,n*sizeof(int));
     if(!frontren)
       {
     printf("No hay memoria");
     break;
       }
     for(i=1;i<=n;i++)
     frontren[i]=0;
     frontval=realloc(frontval,m*sizeof(float));
     if(!frontval)
       {
     printf("No hay memoria");
     break;
       }
     for(i=1;i<=n;i++)
     frontval[i]=0.0;
     copia(xadjt,adjncyt,frontren,frontval,nval,j,n);
     chole1(frontval,n);
     regcopia(xadjt,adjncyt,frontren,frontval,nval,j,n);
     adjncyt[xadjt[j]]=j;
  }
}

/*************************************************************/
/*                        COPIA                              */
/*************************************************************/
/* PASA LOS ELEMENTOS CORRESPONDIENTES DE LA MATRIZ. A       */
/* LA MATRIZ FRONTAL.                                        */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*          ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA      */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL      */
/*          CADA COLUMNA DE L.                               */
/*    NVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO DE     */
/*          LOS RENGLONES APUNTADOS POR ADJNCYT.             */
/*    N - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN LA        */
/*          COLUMNA J.                                       */
/*    J - COLUMNA A PROCESAR.                                */
/* PARAMETROS DE ACTUALIZACION                               */
/*    FRONTREN - ARREGLO DE UNA DIMENSION QUE CONTIENE       */
/*          APUNTADORES A LOS ELEMENTOS DE LA MATRIZ         */
/*          FRONTAL.                                         */
/*    FRONTVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO    */
/*          APUNTADOS POR  FRONTREN.                         */
/*************************************************************/
void copia(int *xadjt,int *adjncyt,int *frontren,
```

```
        float *frontval,float *nval,int j, int n)
{
 int i,k,v,l,b;
 int a=1;
 for (i=xadjt[j];i<=xadjt[j+1]-1;i++)
   {
    frontren[a]=adjncyt[i];
    frontval[a]=nval[i];
    a++;
   }
 for(i=2;i<=n;i++)
   {
    v=i;
    l=frontren[i];
    for(k=xadjt[l];k<=xadjt[l+1]-1;k++)
      {
    b=adjncyt[k];
    if(b!=frontren[v]) continue;
    frontval[a]=nval[k];
    a++;
    v++;
    l=frontren[i];
      }
   }
}

/*************************************************************/
/*                      REGCOPIA                             */
/*************************************************************/
/* REGRESA LOS ELEMENTOS CORRESPONDIENTES DE LA MATRIZ       */
/* FRONTAL YA FACTORIZADA A LA MATRIZ                        */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*          ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA      */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL      */
/*          CADA COLUMNA DE L.                               */
/*    FRONTREN - ARREGLO DE UNA DIMENSION QUE CONTIENE       */
/*          APUNTADORES A LOS ELEMENTOS DE LA MATRIZ         */
/*          FRONTAL.                                         */
/*    FRONTVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO    */
/*          APUNTADOS POR  FRONTREN.                         */
/*    N - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN LA        */
/*        COLUMNA J.                                         */
/*    J - COLUMNA A PROCESAR.                                */
/* PARAMETROS DE ACTUALIZACION                               */
/*    NVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO DE     */
/*          LOS RENGLONES APUNTADOS POR ADJNCYT.             */
/*************************************************************/
void regcopia(int *xadjt,int *adjncyt,int *frontren,
```

```
          float *frontval,float *nval,int j,int n)
{
 int i,k,v,l,b;
 int a=1;
 for (i=xadjt[j];i<=xadjt[j+1]-1;i++)
    {
     nval[i]=frontval[a];
     a++;
    }
 for(i=2;i<=n;i++)
   {
    v=i;
    l=frontren[i];
    for(k=xadjt[l];k<=xadjt[l+1]-1;k++)
       {
     b=adjncyt[k];
     if(b!=frontren[v]) continue;
     nval[k]=frontval[a];
     a++;
     v++;
     l=frontren[i];
       }
   }
}


/*************************************************************/
/*                          CHOLE1                           */
/*************************************************************/
/* REALIZA EL PRIMER PASO DE LA FACTORIZACION DE CHOLESKY    */
/* POR SUBMATRICES                                           */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    N - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN LA        */
/*        COLUMNA J.                                         */
/* PARAMETROS DE ACTUALIZACION                               */
/*    FRONTVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO    */
/*        APUNTADOS POR  FRONTREN.                           */
/*************************************************************/
void chole1(float *frontval,int n)
{
 int j,k,a,b,l,v;
 int i=1;
 float q,p;
 p=frontval[i];
 if(p<=0)
   {
    printf("La matriz no es definida positiva");
    exit(1);
   }
 q=sqrt(p);
```

```
 frontval[i]=q;
 for (i=2;i<=n;i++)
    frontval[i]=frontval[i]/q;
 a=2;
 l=n+1;
 v=2*n-1;
 for(i=2;i<=n;i++)
    {
     b=a;
     for(k=l;k<=v;k++)
       {
     frontval[k]=frontval[k]-frontval[a]*frontval[b];
     b++;
       }
     a++;
     l=v+1;
     v=l+n-i;
    }
}
```

MFRONTAL

Este procedimiento realiza la factorización numérica de Cholesky de una matriz simétrica, definida positiva y rala por el método multifrontal[1]. La matriz frontal es obtenida en cada paso, cambiando su tamaño de acuerdo al número de elementos distintos de cero en cada columna. La matriz de actualización se definió como sigue:

```
struct actual {
 int ind;
 int num;
 int *ren;
 float *val;
 struct actual *sig;
 struct actual *ant;
};
```

Y se definieron dos apuntadores PPIO, FINAL, que apuntan hacia el principio y el final de la lista de matrices de actualización.

El procedimiento utiliza las siguiente subrutinas: SUMACT que suma la matriz de actualización a la matriz frontal, INSERDAT que inserta la matriz de actualización a la lista, y CHOLE1 visto anteriormente en el método frontal.

```
/*************************************************************/
/*                        MFRONTAL                           */
/*************************************************************/
/* OBTIENE LA FACTORIZACION NUMERICA POR CHOLESKY CON EL     */
/* METODO MULTIFRONTAL.                                      */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    ADJNCYT - ARREGLO DE UNA DIMENSION QUE CONTIENE LOS    */
/*          ELEMENTOS DISTINTOS DE CERO EN CADA COLUMNA      */
/*    XADJT - CONTIENE APUNTADORES AL PRINCIPIO Y FINAL      */
/*          CADA COLUMNA DE L.                               */
/*    PADRE - ARREGLO DE UNA DIMENSION QUE CONTIENE AL       */
/*      PADRE DE CADA NODO EN EL ARBOL DE ELIMINACION.       */
/*    NEC - NUMERO DE ECUACIONES                             */
/* PARAMETROS DE ACTUALIZACION                               */
/*    NVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO DE     */
/*          LOS RENGLONES APUNTADOS POR ADJNCYT.             */
/*************************************************************/
void  mfrontal(int *xadjt,int  *adjncyt, float *nval, int
*padre,
         int nec)
{
 int i,j,k,m,n;
 int *frontren;
 float *frontval;
 frontren=malloc(sizeof(int));
 if(!frontren)
   {
    printf("no hay memoria");
    exit(1);
   }
 frontval=malloc(sizeof(float));
 if(!frontval)
   {
    printf("no hay memoria");
    exit(1);
   }
 frontren[1]=1;
 frontval[1]=0.0;
 inserdat(frontren,frontval,1,j);
 for(j=1;j<=nec;j++)
```

```
    {
     n=xadjt[j+1]-xadjt[j];
     m=n*(n+1)/2;
/*-------------------------------------------------------------*/
/* REASIGNA MEMORIA PARA LA MATRIZ FRONTAL                     */
/*-------------------------------------------------------------*/
     frontren=realloc(frontren,n*sizeof(int));
     if(!frontren)
       {
     printf("No hay memoria");
     break;
       }
     for(i=1;i<=n;i++)
     frontren[i]=0;
     frontval=realloc(frontval,(m+1)*sizeof(float));
     if(!frontval)
       {
     printf("No hay memoria");
     break;
       }
     for(i=1;i<=m+1;i++)
     frontval[i]=0.0;
     k=1;
     for (i=xadjt[j];i<=xadjt[j+1]-1;i++)
     {
      frontren[k]=adjncyt[i];
      frontval[k]=nval[i];
      k++;
     }
/*-------------------------------------------------------------*/
/* BUSCA LOS HIJOS DEL NODO EN CURSO                           */
/*-------------------------------------------------------------*/
     for(k=1;k<j;k++)
       {
     i=padre[k];
     if(i!=j) continue;
     sumact(frontren,frontval,n,k);
       }
     chole1(frontval,n);
     k=1;
     for (i=xadjt[j];i<=xadjt[j+1]-1;i++)
     {
      nval[i]=frontval[k];
      k++;
     }
     if(j!=nec)
       inserdat(frontren,frontval,n,j);
     adjncyt[xadjt[j]]=j;
   }
 }
```

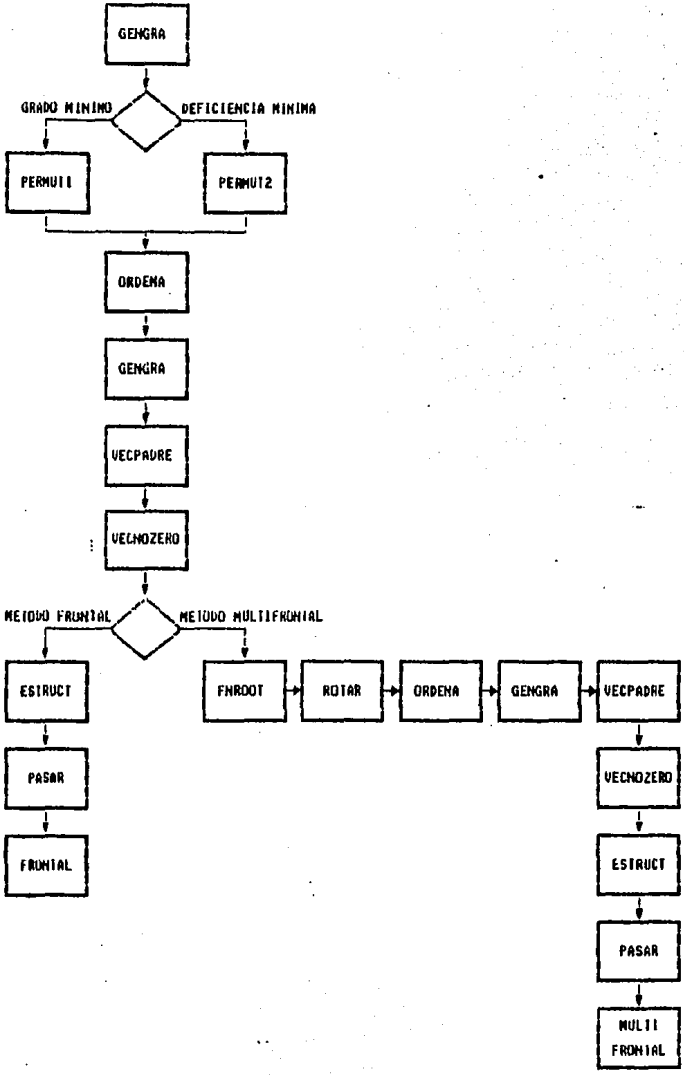
```
/*************************************************************/
/*                        INSERDAT                           */
/*************************************************************/
/* PASA LOS DATOS FACTORIZADOS A LA MATRIZ DE ACTUALIZA-     */
/* CION                                                      */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*    FRONTREN - ARREGLO DE UNA DIMENSION QUE CONTIENE       */
/*          APUNTADORES A LOS ELEMENTOS DE LA MATRIZ         */
/*          FRONTAL.                                         */
/*    FRONTVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO    */
/*          APUNTADOS POR  FRONTREN.                         */
/*    N - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN LA        */
/*          COLUMNA J.                                       */
/*    J - COLUMNA A PROCESAR,                                */
/*************************************************************/
void inserdat(int *frontren,float *frontval,int n,int j)
{
 int i,v,m;
 float p;
 struct actual *aux;
 m=n*(n+1)/2;
/*-----------------------------------------------------------*/
/* ASIGNA MEMORIA PARA LA MATRIZ DE ACTUALIZACION            */
/*-----------------------------------------------------------*/
 aux=malloc(sizeof(struct actual));
 if (!aux)
   {
    printf("No hay memoria");
    return;
   }
 aux->ind=j;
 aux->num=n-1;
 if(n==1) return;
 v=m-n;
 aux->ren=calloc(n-1,sizeof(int));
 if (!aux->ren)
   {
    printf("No hay memoria");
    return;
   }
 for (i=1;i<n;i++)
     aux->ren[i]=frontren[i+1];
 aux->val=calloc(v+1,sizeof(float));
 if (!aux->val)
   {
    printf("No hay memoria");
    return;
   }
 aux->val[v+1]=0.0;
 for (i=1;i<=v;i++)
```

```
    aux->val[i]=frontval[n+i];
 inserta(aux,&ppio,&final);
}

/*************************************************************/
/*                        INSERTA                            */
/*************************************************************/
/* INSERTA LA MATRIZ DE ACTUALIZACION EN LA LISTA            */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*     i - ESTRUCTURA QUE CONTIENE LA MATRIZ DE ACTUALIZA-   */
/*         CION A SER INSERTADA                              */
/*     PPIO - APUNTADOR AL PRIMER ELEMEMTO DE LA LISTA DE    */
/*         ESTRUCTURAS DE MATRICES DE ACTUALIZACION.         */
/*     FINAL - APUNTADOR AL PRIMER ELEMEMTO DE LA LISTA DE   */
/*         ESTRUCTURAS DE MATRICES DE ACTUALIZACION.         */
/*************************************************************/
void inserta(struct actual *i,struct actual **ppio,
          struct actual **final)
{
 if(!*final)
   {
     i->sig=NULL;
     i->ant=NULL;
     *final=i;
     *ppio=i;
     return;
    }
 (*final)->sig=i;
 i->sig=NULL;
 i->ant=*final;
 *final=i;
}

/*************************************************************/
/*                        SUMACT                             */
/*************************************************************/
/* SUMA LAS MATRICES DE ACTUALIZACION DE LOS HIKOS DE J      */
/* A LA MATRIZ FRONTAL DE J.                                 */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*     FRONTREN - ARREGLO DE UNA DIMENSION QUE CONTIENE      */
/*          APUNTADORES A LOS ELEMENTOS DE LA MATRIZ         */
/*          FRONTAL.                                         */
/*     FRONTVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO   */
/*          APUNTADOS POR  FRONTREN.                         */
/*     N - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN LA       */
/*         COLUMNA J.                                        */
/*     J - COLUMNA A PROCESAR.                               */
```

```
/*********************************************************/
void sumact(int *frontren,float *frontval, int n,int j)
{
 struct actual *aux;
 aux=busca(ppio,j);
 suma(frontren,frontval,aux,n);
 borra(aux,&ppio,&final);
 free(aux);
}


/*********************************************************/
/*                   FUNCION BUSCA                       */
/*********************************************************/
/* BUSCA LA MATRIZ DE ACTUALIZACION EN LA LISTA          */
/*********************************************************/
/* PARAMETROS DE ENTRADA                                 */
/*    PPIO - APUNTADOR AL PRIMER ELEMEMTO DE LA LISTA DE */
/*        ESTRUCTURAS DE MATRICES DE ACTUALIZACION.      */
/*    N - COLUMNA A PROCESAR.                            */
/*********************************************************/
struct actual *busca(struct actual *ppio, int n)
{
 while(ppio){
   if (n==ppio->ind) return ppio;
   ppio=ppio->sig;
    }
 return(NULL);
}


/*********************************************************/
/*                      BORRA                            */
/*********************************************************/
/* BORRA LA MATRIZ DE ACTUALIZACION DE LA LISTA          */
/*********************************************************/
/* PARAMETROS DE ENTRADA                                 */
/*    i - ESTRUCTURA QUE CONTIENE LA MATRIZ DE ACTUALIZA-*/
/*        CION A SER BORRADA                             */
/*    PPIO - APUNTADOR AL PRIMER ELEMEMTO DE LA LISTA DE */
/*        ESTRUCTURAS DE MATRICES DE ACTUALIZACION.      */
/*    FINAL - APUNTADOR AL PRIMER ELEMEMTO DE LA LISTA DE*/
/*        ESTRUCTURAS DE MATRICES DE ACTUALIZACION.      */
/*********************************************************/
void borra(struct actual *i,struct actual **ppio,
       struct actual **final)
{
 if(i->ant) i->ant->sig=i->sig;
 else
   {
    *ppio=i->sig;
```

```
    if(*ppio) (*ppio)->ant=NULL;
   }
 if(i->sig) i->sig->ant=i->ant;
 else *final=i->ant;
}


/*************************************************************/
/*                        SUMA                               */
/*************************************************************/
/* SUMA LA MATRIZ DE ACTUALIZACION A LA MATRIZ FRONTAL       */
/* DE J.                                                     */
/*************************************************************/
/* PARAMETROS DE ENTRADA                                     */
/*     FRONTREN - ARREGLO DE UNA DIMENSION QUE CONTIENE      */
/*          APUNTADORES A LOS ELEMENTOS DE LA MATRIZ         */
/*          FRONTAL.                                         */
/*     FRONTVAL - VALOR DE LOS ELEMENTOS DISTINTOS DE CERO   */
/*          APUNTADOS POR  FRONTREN.                         */
/*     AUX - ESTRUCTURA QUE CONTIENE LA MATRIZ DE ACTUALI-   */
/*          ZACION A SER SUMADA                              */
/*     N - NUMERO DE ELEMENTOS DISTINTOS DE CERO EN LA       */
/*          COLUMNA J.                                       */
/*************************************************************/
void suma(int *frontren,float *frontval,
       struct actual *aux, int n)
{
 int i,k,v,a,b,r,m,s,s1,ind1,ind2;
 float p;
 m=n*(n+1)/2;
 s=aux->num;
 if(s==n)
   {
    a=1;
    for(i=1;i<=m;i++)
      {
       p=aux->val[a];
       frontval[i]=frontval[i]+p;
       a++;
      }
    return;
   }
 s1=s*(s+1)/2;
 a=1;
 r=1;
 v=1;
 for(i=1;i<=n;i++)
    {
     ind1=aux->ren[a];
     ind2=frontren[i];
     if(ind1!=ind2)
```

```
      {
       r=m-(n-i)*(n+1-i)/2 +1;
       continue;
      }
     b=a;
     for(k=i;k<=n;k++)
       {
     ind1=aux->ren[b];
     ind2=frontren[k];
     if(ind1!=ind2)
      {
       r++;
       continue;
      }
     p=aux->val[v];
     frontval[r]=frontval[r]+p;
     b++;
     v++;
     r++;
     if(v>s1) return;
       }
     a++;
     r=m-(n-i)*(n+1-i)/2 +1;
    }
}
```


GENGRA
GRADO MINIMO
DEFICIENCIA MINIMA
PERMUT1
PERMUT2
ORDENA
GENGRA
VECPADRE
VECNOZERO
METODO FRONTAL
METODO MULTIFRONTAL
ESTRUCT
PASAR
FRONTAL
FNRDOT
ROTAR
ORDENA
GENGRA
VECPADRE
VECNOZERO
ESTRUCT
PASAR
MULTI FRONTAL

BIBLIOGRAFIA

ABRIN, V., Algoritmos para reordenar matrices ralas, Tesis de Maestria, UNAM, 1985.

ALLEN, R., PRUESS, S., & SHAMPINE, L., Fundamentals of numerical computing, Vinculos Matematicos Departamento de Matematicas Facultad de Ciencias UNAM, 1986.

BANK, R. E. & SMITH, R. K., "General sparse elimination requires no permanent integer storage". SIAM Journal of Scientific Statistic Computation, 8 (1987), pp. 574-584.

CURCO, M., Una introduccion a la teoria de graficas, Vinculos Matemáticos Departamento de Matematicas Facultad de Ciencias UNAM, 1991.

DUFF, I. S. & REID, J. K., "The multifrontal solution of indefinite sparse symmetric linear equations", ACM Transactions on Mathematical Software, 9 (1983), pp. 302-325.

DUFF, I. S., ERISMAN, A. M., & REID J. K., Direct methods for sparse matrices, Oxford University Press, 1987.

GEORGE, A. & LIU, J. W., Computer solution of large sparse positive definite systems, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

HEATH, M. T., NG, E., & PEYTON, B. W., "Parallel algorithms for sparse linear systems", SIAM Review, 33 (1991), pp. 420-460.

IRONS, M. Bruce, "A frontal solution program for finite element analysis", International Journal for Numerical Methods in Engineering, 2 (1970), pp. 5-32.

LIU, J. W. H., "A compact row storage scheme for Cholesky factors using eliminations trees", ACM Transactions on Mathematical Software, 12 (1986), pp. 127-148.

----, "On the storage requirement in the out-of-core multifrontal methods for sparse factorization", ACM Transactions on Mathematical Software, 12 (1986), pp. 249-264.

----, "Equivalent sparse matrix reordering by elimination tree rotations", SIAM Journal of Scientific Statistic Computation, 9 (1988), pp. 424-444.

----, "The role of elimination trees in sparse factorization",

SIAM Journal of Matrix Analysis and Applications, 11 (1990), pp. 134-172.

----, "The multifrontal method for sparse matrix solution: theory and practice", SIAM Review, 34 (1992), pp. 82-109.

PISSANETZKY, S., Sparse matrix technology, Academic Press, New York, 1984.

ROSE, D. J., "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations", en Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1972, pp. 183-217.

ROSE, D. J., TARJAN, R. E. & LUEKER, G. S., "Algorithmic aspects of vertex elimination on graphs", SIAM Journal of Computation, 5 (1976), pp. 266-283.

SCHREIBER, R., "A new implementation of sparse Gaussian elimination", ACM Transactions on Mathematical Software, 8 (1982), pp. 256-276.

TARJAN, R. E. & YANNAKAKIS M., "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs", SIAM, Journal of Computation, 13 (1984), pp. 566-579.