CNEA-NT 20/81

REPUBLICA ARGENTINA

COMISION NACIONAL DE ENERGIA ATOMICA
Dependiente de la Presidencia de la Nación

CENTRO ATOMICO BARILOCHE

INSTITUTO BALSEIRO

"COMPUTACION CON MATRICES RALAS"

Sergio Pissanetzky

BUENOS AIRES

1981

Sparse matrix computation

Summary

A matrix is sparse when most of its elements vanish. This property has been used to develop algorithms which perform matrix operations on sparse matrices. These algorithms are very efficient regarding both computer memory and time, and are thus convenient to solve large problems. We discuss the basic ideas and techniques used to develop the algorithms. A set of Fortran programs is presented, which can perform a variety of matrix operations on sparse matrices.

Resumen

Unas matriz se denomina rala cuando la mayor parte de sus elementos son nulos. Esta propiedad ha permitido desarrollar algoritmos para efectuar una gran variedad de operaciones con matrices ralas, que son muy eficientes tanto en memoria como en tiempo de computadora, y por lo tanto convenientes cuando el problema que se desea resolver es grande. En esta publicación se discuten las ideas y técnicas básicas que han permitido el desarrollo de los algoritmos, y se presenta un conjunto de programas en Fortran para realizar la mayoría de las operaciones matriciales con matrices ralas.

El término "Matriz Rala"

Parece apropiado iniciar este trabajo con algunas consideraciones sobre el término "matriz rala". Cuando en 1978 me propuse escribir un trabajo en castellano sobre matrices con muchos ceros y sobre un programa que aprovecha esta característica para resolver ecuaciones lineales, después de haber leído una cantidad de literatura en inglés en que una matriz con muchos ceros se llama "Sparse Matrix", y sin haber visto nunca un trabajo en castellano sobre el tema, me encontré con la dificultad de traducir ese término. La elección debi hacerla entre "dispersa", "esparcida" y "rala". Dejé de lado "dispersa", que tiene un cierto sentido dinámico ("scattered"), aunque ya lo había usado (1). Y si bien en ese momento no consideré el término "desparramada", podría haberlo considerado y haberlo descartado por la misma razón. Los dos términos restantes, "esparcida" y "rala", son sin duda muy semejantes. "Esparcida" parecía un vocablo mas apropiado para uso técnico, y además se usa como adjetivo con un significado similar tanto en Astronomía como en Botánica. Por eso lo elegí, y mi trabajo fue finalmente publicado en Mayo de 1978 como Nota Técnica de la Comisión Nacional de Energía Atómica (2) con el término "matriz esparcida".

El término "matriz rala" nunca dejó de preocuparme. Varias personas me llamaron la atención sobre su conveniencia. Después de todo podemos decir que un caballero tiene una

barba rala, o que un bosque es ralo, y el concepto que transmitimos es muy similar al que corresponde a una matriz rala. Siguiendo mis pensamientos en esta línea, ocurrió que en Julio de 1980 di un cursillo en el Centro de Cómputos del Centro Atómico Constituyentes, y uno de los temas tratados fue "Matrices Esparcidas o Ralas". Casi al mismo tiempo, en la correspondencia que mantuve con la Universidad Nacional del Centro de la Provincia de Buenos Aires a partir de Febrero de 1980, y más tarde, en Setiembre de 1980, con la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, se utilizaron ambos términos por mi parte, pero sólo "matrices ralas" por parte de la Universidad y de la Comisión.

La circunstancia de que un término sea apropiado o nó para uso técnico la determina el uso. Si usamos "matriz rala", esta expresión se convertirá en un término técnico. O tal vez ya se haya convertido. "Rala", como "esparcida", expresa correctamente el concepto de que se trata, pero es más breve, lo que es una ventaja nada despreciable para el uso técnico. Otra ventaja de "rala" es que en castellano tenemos "raleza", que podemos usar para traducir "sparseness". Claro que si la riqueza de nuestro idioma nos llevara a usar "matriz tupida" o "matriz espesa", los antónimos de "rala", para referirnos a una "full matrix" o "dense matrix", eso ya sería otra cuestión. Por el momento parece mejor usar "matriz llena" o "matriz densa". Los hechos son que tanto el presente trabajo

como el Seminario que tuve la satisfacción de exponer en Tandil en Noviembre de 1980, llevan por nombre "Operaciones con Matrices Ralas".

1. Introducción

Una matriz se llama Rala cuando muchos de sus elementos son iguales a cero. No existe una regla específica para determinar que fracción del total de elementos de la matriz se debe anular para que la matriz sea rala. La definición es simplemente una cuestión de conveniencia; los algoritmos de matrices ralas darán tanto mejores resultados cuanto menor sea el número de no-ceros. Una matriz rala puede ser procesada como si fuera densa, o de banda, pero el tiempo y la memoria de computadora requeridas pueden resultar muy grandes. También una matriz densa puede ser procesada por los algoritmos de matrices ralas: se obtendrían los mismos resultados numéricos pero el almacenamiento requerido y el tiempo serían algo mayores.

Los objetivos de los algoritmos de matrices ralas son:

i) Almacenar sólo los no-ceros en la memoria de la computadora. Además del valor numérico de cada no-cero hay que almacenar información que indique la ubicación del no-cero en la matriz. Esta información adicional es el precio que hay que pagar para evitar el almacenamiento de los ceros.

ii) Procesar sólo los no-ceros, de manera que el número de sumas, multiplicaciones, etc., realizadas por la computadora durante una operación matricial, sea proporcional a la cantidad de no-ceros y no al número de elementos en la matriz. En muchos casos prácticos, la cantidad de no-ceros por fila es constante, independiente del orden N de la matriz. En tales casos, la complejidad asintótica de un algoritmo de matrices ralas puede ser tan baja como N.

iii) Preservar la raleza. En otras palabras, realizar las operaciones matriciales de tal manera que la generación de nuevos no-ceros en el curso de las mismas se mantenga tan pequeña como sea posible.

Un algoritmo del que se espera que procese sólo los no-ceros de una matriz, requerirá habitualmente un conocimiento previo de las posiciones de esos no-ceros. Como consecuencia, muchos algoritmos de matrices ralas tienen una sección simbólica y una sección numérica. La sección simbólica del algoritmo determina las posiciones de los no-ceros o estructura de la matriz resultado. Los valores numéricos de los no-ceros se calculan durante la ejecución de la sección numérica del algoritmo. Una ventaja adicional de este procedimiento es que en muchos casos prácticos, por ejemplo en cálculos iterativos, la estructura es fija, mientras que se debe calcular diferentes conjuntos de valores numéricos de los no-ceros. En esos casos, la sección simbólica se ejecuta una

sola vez, y sólo la sección numérica debe ser ejecutada repetidamente, dentro del ciclo de iteración.

Hoy en día, las técnicas de matrices ralas son usadas para muchas aplicaciones con matrices grandes, típicamente 1000x1000 ó 10000x10000, y aún más. Algunas de esas aplicaciones son el Análisis Económico (3), simulación de yacimientos de petróleo (4, 5, 6), análisis de redes (7, 8, 9), análisis de sistemas eléctricos de potencia (10), programación lineal (11, 12), fotogrametría (13) y, obviamente, métodos numéricos tales como el método de Elementos Finitos y el método de Diferencias Finitas que se usan para la resolución de una variedad de problemas de física e ingeniería. Las técnicas de matrices ralas se usarían aún mucho más si fueran fáciles de usar, y es en éste sentido que esperamos que nuestra publicación resulte útil. El tamaño de los problemas a ser resueltos se hace cada día más y más grande. Las computadoras también se hacen más grandes, pero el usuario generalmente tiene una computadora que no puede cambiar y necesita procesar problemas cada vez mayores en su instalación. La mejor elección, y tal vez la única elección, puede ser el uso de formatos ralos y de matrices ralas.

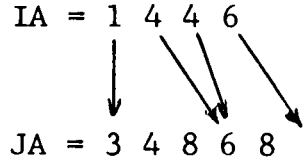
2. El formato ralo por filas.

Consideremos una matriz rala. Los valores numéricos de los elementos que son diferentes de cero están almacenados en la memoria de la computadora. También hay que almacenar información que indique a qué fila y a qué columna pertenece cada elemento. El formato de almacenamiento debe ser tal que permita construir algoritmos eficientes para las operaciones matriciales de interés. Aquí describiremos el formato ralo por filas, que tiene requerimientos mínimos de memoria y es al mismo tiempo muy conveniente para la mayoría de las operaciones.

Vamos a introducir el formato ralo por filas con la ayuda de un ejemplo. Consideremos la matriz:

$$A = \begin{vmatrix} 0 & 0 & 1 & 3 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 1 & 0 & 0 \end{vmatrix} \qquad (1)$$

que se representa de la siguiente manera:



R(C)FO (2)

Los valores numéricos de los no-ceros de A se almacenan en el vector AN y sus índices de columna se almacenan en

correspondencia en otro vector JA. El vector IA contiene punteros que señalan las posiciones en AN y JA donde comienza la descripción de cada fila. La descripción de la fila 1 de A empieza en la posición IA(1)=1 de AN y JA. Como la descripción de la fila 2 empieza en IA(2)=4, se deduce que la fila 1 de A está descripta en las posiciones 1, 2 y 3 de AN y JA. En el ejemplo:

IA(1)=1 la primera fila empieza en JA(1) y AN(1).

IA(2)=4 la segunda fila empieza en JA(4) y AN(4).

IA(3)=4 la tercera fila empieza en JA(4) y AN(4); puesto que esta es la misma posición donde empieza la fila 2, eso significa que la fila 2 está vacía.

IA(4)=6 señala el primer puesto vacío en JA y AN; este puntero se usa para especificar que la descripción de la fila 3 se extiende hasta la posición 6-1=5 de JA y AN.

En general, podemos decir que la fila f de A se describe en las posiciones IA(f) hasta IA(f+1)-1 de JA y AN, excepto cuando IA(f+1)=IA(f), en cuyo caso la fila f está vacía. Notar que si la matriz A tiene F filas, entonces IA tiene F+1 posiciones.

La representación (2) se llama completa porque toda la matriz A está representada, y ordenada porque los elementos de cada fila están dados en el orden ascendente de sus índices de columna. La representación (2) es por lo tanto una Repre-

sentación Completa por Filas Ordenada, o R(C)FO. Los vectores IA y JA representan la estructura de A. Si un algoritmo está dividido en una sección simbólica y una sección numérica, como se discutió en la Introducción, los vectores IA y JA son determinados por la sección simbólica, y AN por la numérica. Las representaciones de matrices ralas no necesitan estar obligatoriamente ordenadas, sino que los elementos de cada fila pueden ser dados en cualquier orden. El orden de las filas, por su parte, debe ser respetado. La matriz A de nuestro ejemplo, ecuación (1), puede ser especificada en una Representación Completa por Filas Desordenada:

```
IA= 1 4 4 6
JA= 8 3 4 8 6          R(C)FD                    (3)
AN= 5 1 3 1 7
```

Contrariamente a lo que pudiera parecer, las representaciones desordenadas son más convenientes que las ordenadas. Los resultados de muchas operaciones matriciales se obtienen en forma desordenada, y resultaría muy costoso ordenarlos. Por otra parte, con muy pocas excepciones, los algoritmos de matrices ralas pueden trabajar igualmente bien con representaciones ordenadas o desordenadas.

Las representaciones por columna también son usadas, pero se las puede considerar como representaciones por filas de las matrices traspuestas. Para nuestro ejemplo, ecuación (1), la

matriz A puede ser dada en una Representación Completa por Columnas Ordenada como sigue:

$$\begin{aligned} &IAT = 1\ 1\ 1\ 2\ 3\ 3\ 4\ 4\ 6\ 6\ 6 \\ &JAT = 1\ 1\ 3\ 1\ 3 \qquad\qquad R(C)CO \qquad\qquad (4) \\ &ANT = 1\ 3\ 7\ 5\ 1 \end{aligned}$$

que también se puede considerar una R(C)FO de A^T, la traspuesta de A. Notar que IAT tiene 11 posiciones, puesto que A tiene 10 columnas. Cuando la matriz dada es simétrica, es suficiente con representar sólo su diagonal y su triángulo superior. Consideremos el ejemplo siguiente:

$$B = \begin{vmatrix} 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 3 & 0 \\ 1 & 1 & 0 & 3 \end{vmatrix} \qquad (5)$$

cuya Representación Diagonal y Superior por Filas Ordenada es:

$$\begin{aligned} &IB = 1\ 3\ 5\ 6\ 7 \\ &JB = 1\ 4\ 2\ 4\ 3\ 4 \qquad\qquad R(DS)FO \qquad\qquad (6) \\ &BN = 2\ 1\ 1\ 1\ 3\ 3 \end{aligned}$$

Un caso muy importante es aquel en que la matriz dada, además de ser simétrica, tiene todos sus elementos diagonales diferentes de 0. Este caso es, por ejemplo, el de las matrices definidas positivas, que tienen todos sus elementos diagonales positivos. En ese caso, los elementos diagonales pueden ser almacenados en un vector AD, lleno, y sólo los elementos del triángulo superior son representados en formato ralo:

```
IB = 1 2 3 3 3
JB = 4 4                    R(S)FO                    (7)
BN = 1 1
BD = 2 1 3 3
```

Esta representación es muy densa y conveniente.

3. Algoritmos para matrices ralas

En el resto de esta publicación vamos a examinar algunos de los algoritmos que se usan para realizar las operaciones del álgebra matricial con matrices almacenadas en formato ralo por filas. En el texto, usaremos directamente los símbolos del Fortran a fin de evitar la introducción innecesaria de símbolos algebraicos, y daremos ejemplos de programas en Fortran al final de cada Sección. Usaremos también las convenciones usuales para los tipos de variable INTEGER y REAL. Al escribir los programas hubo que adoptar una decisión importante: se ha sacrificado la claridad en favor de la eficiencia y compatibilidad. Los programas de matrices ralas son inevitablemente difíciles de entender, pues los algoritmos de matrices ralas son complicados. Se han hecho todos los esfuerzos posibles para escribir los programas de la manera más simple posible, excepto cuando ello iba en detrimento de la calidad. Los siguientes dos ejemplos pueden ser útiles para ilustrar este punto. Primero, consideremos el siguiente ciclo DO:

```
      DO α K=IA(I),IA(I+1)-1
      ...
α     CONTINUE
      ...
```

El significado de la instrucción DO es muy claro: el ciclo debe ser ejecutado para valores de K en el rango IA(I) hasta IA(I+1)-1 en incrementos de una unidad. Si IA(I+1)-1< IA(I), una situación que se presenta cuando la fila I de la matriz está vacía, el ciclo no debe ser ejecutado en absoluto. Sin embargo, muchos compiladores rechazarían la instrucción DO, o ejecutarían el ciclo una vez aún si IA(I+1)-1< IA(I). Ambas dificultades se evitan y el programa se hace compatible con la mayoría de las instalaciones si lo escribimos:

```
      IAA = IA(I)
      IAB = IA(I+1)-1
      IF(IAB.LT.IAA) GO TO ß
      DO α K = IAA,IAB
      ...
α     CONTINUE
ß     ...
```

El segundo ejemplo es la optimización de los ciclos DO que llevan a cabo algunos compiladores modernos, sacando fuera del ciclo cualquier cálculo que sea independiente del parámetro del ciclo. El programador puede escribir expresiones completas

dentro del ciclo, aún si las expresiones son en parte o totalmente independientes del parámetro. El programa que resulta se parece a las fórmulas matemáticas y es fácil de comprender, pero no operaría con eficiencia en sistemas que no posean la facilidad de optimización. Por esa razón, en nuestros ejemplos usamos ciclos DO que ya están optimizados. Son un poco más difíciles de comprender, pero operan eficientemente en cualquier sistema.

Para evitar confusiones, se ha evitado el empleo de tarjetas de comentario dentro de los listados, y todos los comentarios se hacen por separado, con la ventaja adicional que así no están restringidos al empleo de los símbolos Fortran. Los programas ejemplo dados en las secciones siguientes son, por lo tanto:

- Eficientes en cualquier instalación.
- Compatibles con la mayoría de las instalaciones existentes.
- Difíciles de entender en algunos casos.

Los ejemplos pueden ser usados directamente en programas profesionales como secciones de programa o como subrutinas individuales. También existen otras publicaciones con programas de matrices ralas (15), y excelentes trabajos en que se describen las características que deben reunir tales programas (16, 17).

4. Producto de una matriz rala general por un vector lleno

La primera operación matricial que vamos a examinar es el producto de una matriz rala general IA, JA, AN en R(C)FD por un vector lleno almacenado en un vector B. El resultado es otro vector lleno que deberá ser almacenado en el vector C. Sea N el número de filas de la matriz. Puesto que los elementos de B pueden ser accedidos al azar, el orden en el cual se realizan los productos internos queda determinado simplemente por el orden arbitrario en que están almacenados los elementos de la matriz. Por lo tanto, para cada fila I de la matriz, usamos IA para obtener el puntero inicial IAA y el puntero final IAB a las posiciones de JA y AN donde se describe la fila I. Después, para calcular el producto interno de la fila I por el vector B, simplemente recorremos JA y AN desde IAA hasta IAB: cada valor almacenado en JA es un índice de columna que usamos para encontrar en el vector B el elemento que dede ser multiplicado por el correspondiente valor de AN. El resultado de cada multiplicación se acumula en C(I). El algoritmo se presenta completo en el ejemplo siguiente.

Ejemplo 1. Producto de una matriz rala general por un vector lleno.

Datos: IA, JA, AN matriz dada en R(C)FD.

B vector lleno dado.

N cantidad de filas de la matriz y de elementos de C.

Resultados: C vector resultado.

```
1.          DO 10 I=1,N
2.          U=0.
3.          IAA=IA(I)
4.          IAB=IA(I+1)-1
5.          IF(IAB.LT.IAA)GO TO 10
6.          DO 20 K=IAA,IAB
7.    20    U=U+AN(K)*B(JA(K))
8.    10    C(I)=U
```

La finalidad de la instrucción en la línea 5 es detectar filas vacías en la matriz. En la línea 6,K es el puntero para JA y AN, y JA(K) en la línea 7 es el índice de columna correspondiente al elemento AN(K). La variable U se usa para acumular los productos en la línea 7 en vez de C(I) porque algunos compiladores Fortran generan instruccioners de máquina más eficientes cuando se usan variables no subindicadas. Además, de esta manera se puede declarar U de doble precisión para obtener una mayor exactitud en la acumulación en la línea 7, y después redondear el resultado final en la línea 8.

5. Producto de una matriz rala simétrica por un vector lleno

En esta sección vamos a examinar el producto de una matriz rala simétrica IA, JA, AN, AD dada en R(S)FD, por un vector lleno B. Este algoritmo es muy similar al descripto en la sección anterior, excepto que, ahora, cada valor almacenado en AN describe dos elementos simétricos de la matriz, los que deben ser multiplicados por diferentes elementos del vector B.

Como consecuencia, la acumulación debe ser hecha directamente en el vector C. El algoritmo se presenta en el ejemplo 2.

Ejemplo 2. Producto de una matriz rala simétrica por un vector lleno.

Datos: IA,JA,AN,AD matriz dada en R(S)FD.

B vector lleno dado.

N cantidad de filas de la matriz y del vector C.

Resultados: C vector resultado.

```
 1.          DO 10 I=1,N
 2.          Z=B(I)
 3.          U=AD(I)*Z
 4.          IAA=IA(I)
 5.          IAB=IA(I+1)-1
 6.          IF(IAB.LT.IAA)GO TO 10
 7.          DO 20 K=IAA,IAB
 8.          J=JA(K)
 9.          U=U+AN(K)*B(J)
10.    20    C(J)=C(J)+AN(K)*Z
11.    10    C(I)=U
```

En las líneas 9 y 10,AN(K) es tanto el valor del elemento de la fila I, columna J, como del elemento de la fila J, columna I de la matriz. Por lo tanto, AN(K) debe ser multiplicado por B(J) y por B(I), y los resultados acumulados en C(I) y C(J), respectivamente. Notar también que J en la línea 8 es siempre mayor que I, lo que significa que cuando se realiza la acumulación en la línea 10, C(J) ya ha sido inicializado en la línea 11 durante alguna ejecución anterior del ciclo DO.

6. Trasposición de una matriz rala y ordenamiento de una representación rala.

Existe un algoritmo simple que transforma una representación por filas de una matriz en otra representación por columnas de la misma matriz, o viceversa (18). Puesto que una representación por columnas de la matriz es una representación por filas de la matriz traspuesta, el algoritmo efectivamente traspone la matriz. Una propiedad adicional del algoritmo es que la representación resultante está ordenada. Por lo tanto, si el algoritmo se usa para trasponer dos veces una matriz que estaba dada originalmente en una representación desordenada, resulta una representación ordenada de la misma matriz. Notar que cuando la matriz es simétrica, se puede ordenar la representación trasponiéndola una sola vez.

Sea IA, JA, AN cierta representación rala de una matriz con N filas y M columnas. Queremos obtener IAT, JAT, ANT, la representación rala de la matriz traspuesta. Un intento de buscar los elementos de cada columna de la matriz llevaría a una inspección muy ineficiente de IA y JA. En vez de ello, definimos M listas de enteros, inicialmente vacías, cada una con un puntero que señala el primer lugar vacío, inicialmente el primer lugar. También se definen M listas de números reales, en correspondencia con los números de enteros y asociadas con los mismos punteros. A continuación recorremos todos los no-ceros de la matriz. Por cada no-cero I, J, añadimos I a la

lista J de enteros y el valor del no-cero a la lista J de números reales, e incrementamos el puntero correspondiente. El ejemplo que sigue sirve para ilustrar el procedimiento.

Ejemplo 3. Trasposición simbólica de una matriz rala. Consideremos la siguiente matriz rala, donde hemos omitido los valores numéricos de los no-ceros para mayor simplicidad, y hemos numerado las filas y las columnas:

	1	2	3	4	5	6
1	0	0	X	0	X	X
2	X	0	0	X	0	0
3	0	0	X	X	0	0
4	X	0	X	X	0	0
5	0	X	0	0	X	X

Esta matriz se puede describir en R(C)FD como sigue:

IA = 1 4 6 8 11 14

JA = 5 6 3 , 4 1 , 3 4 , 4 3 1 , 2 6 5

fila 1 2 3 4 5

Para mayor claridad, hemos indicado a qué fila de la matriz corresponde cada grupo de índices de columna almacenados en JA, separando además los grupos con comas. También hemos omitido el vector AN. Ahora, como la matriz tiene 6 columnas, definimos 6 listas, inicialmente vacías. Enseguida recorremos la fila 1, que contiene los índices de columna 5, 6 y 3, y añadimos 1 a las listas 5, 6 y 3:

1:
2:
3: 1
4:
5: 1
6: 1

Enseguida, recorremos la fila 2, y añadimos 2 a las listas 4 y 1:

1: 2
2:
3: 1
4: 2
5: 1
6: 1

Cuando todas las filas han sido procesadas, tenemos:

1: 2 4
2: 5
3: 1 3 4
4: 2 3 4
5: 1 5
6: 1 5

o sea, escribiendo las listas una a continuación de la otra:

JAT = 2 4 , 5 , 1 3 4 , 2 3 4 , 1 5 , 1 5

que, dejando de lado por ahora IAT y ANT, es una representación por columnas de la matriz dada, o una R(C)FO de la matriz traspuesta. IAT se obtiene simplemente contando los elementos en cada lista, y ANT, si es necesario, puede también obtenerse sin más que añadir el valor numérico de cada no-cero, almacenado en AN, a las listas de números reales, enseguida que cada entero ha sido añadido a las listas de enteros.

En la práctica las listas se definen directamente en los vectores JAT y ANT, y los punteros que señalan el primer puesto libre en cada lista se almacenan en IAT. De esa manera se minimiza el movimiento de información dentro de la memoria de

la computadora. El algoritmo completo se presenta y se discute en detalle en el ejemplo 4.

Ejemplo 4. Trasposición de una matriz rala general.

Datos: IA, JA, AN matriz dada en R(C)FD.

N cantidad de filas de la matriz.

M cantidad de columnas de la matriz.

Resultados: IAT, JAT,ANT matriz traspuesta en R(C)FO.

```
 1.          MH=M+1
 2.          NH=N+1
 3.          DO 10 I=2,MH
 4.   10     IAT(I)=0
 5.          IAB=IA(NH)-1
 6.          DO 20 I=1,IAB
 7.          J=JA(I)+2
 8.          IF(J.LE.MH)IAT(J)=IAT(J)+1
 9.   20     CONTINUE
10.          IAT(1)=1
11.          IAT(2)=1
12.          IF(M.EQ.1)GO TO 40
13.          DO 30 I=3,MH
14.   30     IAT(I)=IAT(I)+IAT(I-1)
15.   40     DO 60 I=1,N
16.          IAA=IA(I)
17.          IAB=IA(I+1)-1
18.          IF(IAB.LT.IAA)GO TO 60
19.          DO 50 JP=IAA,IAB
20.          J=JA(JP)+1
21.          K=IAT(J)
22.          JAT(K)=I
23.          ANT(K)=AN(JP)
24.   50     IAT(J)=K+1
25.   60     CONTINUE
```

Este algoritmo traspone una matriz de NxM., Para ello se definen M listas en cada uno de los vectores JAT y ANT. En el vector IAT se inscriben los punteros para las listas, de la manera que se explica a continuación. Primero se cuenta la cantidad de elementos que van a ser inscriptos en cada lista,

en las líneas 6 a 9: enseguida después que la línea 9 ha sido procesada, IAT(J) contiene la cantidad de elementos en la columna J-2 de la matriz, para J en el rango 3 hasta M+1. Los punteros mismos son construidos en las líneas 13 y 14: cuando la línea 14 ha sido procesada, IAT(I) señala la posición en JAT y ANT donde empieza la lista I-1, para I-1 en el rango 1 a M. De esta manera, al añadir los elementos en la lista I-1 e incrementar su puntero IAT(I), éste se transforma automáticamente en el puntero para el primer lugar libre de la lista I, que sigue a continuación de la lista I-1 en los vectores JAT y ANT. Este resultado, que está de acuerdo con la definición de los punteros para representaciones ralas dada en la Sección 2, se consigue de esta manera con un mínimo de esfuerzo computacional.

El ciclo DO 60 recorre cada fila de la matriz y añade elementos a las listas definidas en los vectores JAT y ANT. En la línea 20, se ubica un no-cero en la fila I, columna J-1 de la matriz. En la fila 21, K señala la posición en JAT y ANT donde se encuentra el primer lugar disponible de la lista J-1. En las líneas 22 y 23, el índice de fila I y el valor AN(JP) del no-cero son añadidos a las listas JAT y ANT, respectivamente, y en la línea 24 el puntero IAT(J) es aumentado.

Cuando el algoritmo del ejemplo 4 se aplica dos veces sucesivamente, se vuelve a obtener una representación de la misma matriz, pero esta vez la representación queda ordenada. Las representaciones ordenadas son necesarias para algunas

aplicaciones, la más importante de las cuales es la eliminación de Gauss. Es importante notar que para ordenar una lista de n números se requieren n(n-1)/2 comparaciones, pues cada número debe ser comparado con cada uno de los que le siguen en la lista. Aquí, en cambio, el orden se consigue con sólo $k_1N+k_2M+k_3Z$ operaciones elementales, donde k_1, k_2 y k_3 son números pequeños y Z es la cantidad total de no-ceros. La complejidad asintótica del algoritmo de ordenamiento para matrices ralas es, por lo tanto, lineal.

En algunos casos, se requiere trasponer u ordenar sólo la estructura IA, JA de una matriz rala. Para ese fin se puede utilizar el mismo algoritmo del ejemplo 4, excepto que la instrucción en la línea 23 debe ser omitida, y el ejemplo se transforma en un algoritmo para la trasposición simbólica de una matriz rala general. Dos ejemplos importantes en que es necesario trasponer u ordenar simbólicamente una matriz rala son: la eliminación de Gauss (o la factorización triangular) y la conectividad de una red de Elementos Finitos. Ambos ejemplos serán discutidos cuidadosamente en las secciones siguientes. Por ahora baste decir que la matriz de conectividad es una matriz booleana, cuyos elementos valen 0 ó 1. En formato ralo, como todos los no-ceros son iguales a 1, no se los almacena. En consecuencia, la matriz queda completamente descrita por su estructura. Para la eliminación de Gauss se requiere una representación ordenada. Sin embargo, como la eliminación de Gauss se hace primero simbólicamente y después numéricamen-

te, y como la eliminación simbólica puede hacerse sobre una estructura desordenada, el procedimiento usual es como sigue: hacer la eliminación de Gauss simbólica sobre una estructura desordenada, trasponer la estructura desordenada resultante dos veces para ordenarla, y hacer la eliminación numérica sobre la estructura ordenada. La trasposición que requiere este procedimiento es sólo simbólica.

Las mismas ideas y métodos descritos en esta sección pueden ser usadas para permutar filas y columnas de una matriz rala (18).

7. Suma y resta de matrices ralas

En esta sección examinaremos el siguiente problema, dadas dos o más matrices ralas, calcular su suma. Puesto que la resta de una matriz es equivalente a la suma con los signos de los no-ceros cambiados, nos referimos a la sustracción como suma. La suma de matrices ralas se realiza por medio de dos algoritmos: un algoritmo simbólico, que determina la estructura de la matriz resultante, y un algoritmo numérico, que determina los valores de los no-ceros, sacando ventaja del conocimiento previo de sus posiciones en la matriz. Es posible realizar la suma completa en un solo paso, pero se gana poco haciéndolo de esa manera. Por otra parte, al dividir el proce-

dimiento de suma en dos partes se introduce un grado de libertad adicional en el programa, el que puede resultar muy conveniente, por ejemplo para cálculos iterativos en que las estructuras son fijas pero los valores numéricos cambian en cada iteración, o también cuando se necesita una representación ordenada y se la puede obtener ordenando la estructura sola antes de agregarle los valores numéricos. Es interesante notar que el resultado del algoritmo simbólico será una estructura desordenada aún cuando las estructuras de todos los sumandos estén ordenadas. Por otro lado, el algoritmo numérico requiere la estructura como dato de entrada y no la modifica.

Ahora vamos a introducir dos técnicas comunes, que se emplean muy comunmente en la computación con matrices ralas, pero que se pueden explicar mejor en conección con la operación de suma: la técnica de llave múltiple y el vector expandido para reunir dos o más conjuntos de números reales. Ambas técnicas serán descriptas con la ayuda del ejemplo siguiente.

Ejemplo 5. Suma de matrices ralas.

Consideremos la siguiente suma de dos matrices A y B, cuyo resultado es la matriz C:

$$\begin{vmatrix} 0 & 0 & 2 & 0 & -1 & 0 \\ 4 & 0 & 3 & 3 & 7 & 0 \\ -2 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{vmatrix} + \begin{vmatrix} 1 & 0 & -1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 4 & 6 & 0 & 2 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 & 0 & -1 & 5 \\ 4 & 0 & 3 & 3 & 5 & 0 \\ 2 & 6 & 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{vmatrix}$$

Las matrices A y B se dan en R(C)FO como sigue:

```
  JA = 5 3 , 4 3 1 5 , 1 6 , 4 2
  AN =-1 2 , 3 3 4 7 ,-2-1 , 1 1
fila    1        2        3     4

  JB = 1 6 3 , 5 , 4 2 1 , 2 3
  BN = 1 5-1 ,-2 , 2 6 4 ,-1 1
fila     1     2     3       4
```

donde, por simplicidad, hemos omitido los punteros de fila IA e IB, pero hemos indicado el índice de fila que corresponde a cada grupo de no-ceros, en forma explícita. Como la matriz C tiene 6 columnas, definimos el vector de llave múltiple IX como un vector entero de 6 posiciones, y lo inicializamos a 0:

IX = 0 0 0 0 0 0

Para obtener la matriz C, vamos a construir primeramente JC reuniendo JA y JB fila por fila. Para la primera fila, esto significa reunir las listas 5 3 y 1 6 3. Estos números son añadidos secuencialmente a la lista JC. Cuando el 5 es añadido en JC, también almacenamos el índice de fila 1 en IX(5). Luego añadimos 3 a JC y almacenamos 1 en IX(3). Para evitar la repetición de elementos en JC, antes de añadir cada elemento comprobamos si el valor 1 está almacenado en la posición correspondiente de IX. Por ejemplo, antes de inscribir el último número 3 en JC comprobamos que IX(3) es ya igual a 1, lo que significa que el 3 ya ha sido incluido en JC, y de esa manera evitamos incluirlo por segunda vez. Cuando la fila 1 ha sido procesada, tenemos:

JC = 5 3 1 6

IX = 1 0 1 0 1 1

Construir IC es fácil con la ayuda de un puntero que señala el primer lugar libre de JC. Por ejemplo, en este caso

IC(1)=1,IC(2)=5. Continuando con el proceso, ahora debemos reunir las segundas filas, que son 4 3 1 5 y 5. Como ahora el índice de fila es 2, almacenaremos 2 en IX y usaremos 2 como llave para decidir si un índice de columna ha sido anotado ya en JC o nó. De esta manera se hace claro el efecto de la técnica de llave múltiple: se evita la reinicialización de IX a 0 después de procesar cada fila, pues justo antes de procesar una fila i, en IX sólo hay números menores que i, los que no van a interferir con el uso de i como llave para la fila i. Por ejemplo, después de procesar la fila 2 tenemos:

JC = 5 3 1 6 , 4 3 1 5 2

IX = 2 0 2 2 2 1

e IX está listo para iniciar el proceso de la fila 3. Es útil señalar aquí, que hay una manera para reinicializar IX económicamente a 0 después de procesar cada fila, donde "económicamente" significa que el número de operaciones necesario es proporcional a la cantidad de no-ceros y no a la cantidad de elementos en la fila. Ello se logra usado la información almacenada en JC. Por ejemplo, cuando JC=5 3 1 6, reinicializamos sólo los lugares IX(5), IX(3), IX(1) e IX(6). Este procedimiento alternativo puede convenir para ahorrar memoria de computadora en aquellos casos en que IX puede ser almacenado en menos espacio declarándolo LOGICAL*1, o usando simplemente un vector de bits, cada uno de los cuales puede almacenar sólo 0 o 1, y entonces IX debe ser inevitablemente reinicializado a 0 después de procesar cada fila.

Ahora debemos construir CN, el vector que contiene los valores numéricos de los no-ceros de C. Esta es la parte numérica del algoritmo. Para hacerlo, usaremos un vector X en el cual vamos a reunir los valores de los no-ceros de cada fila, almacenados en AN y BN. Empezando por la fila 1, usamos primeramente JC para cargar ceros en las posiciones 5, 3, 1 y 6 de X. Después usamos JA=5 3 para almacenar los valores -1 y 2 de AN en las posiciones 5 y 3, respectivamente, de X. A continuación usamos JB y BN para acumular los valores 1, 5 y -1 en las posiciones 1,6 y 3 de X, respectivamente. Finalmente, usamos JC de nuevo para extraer de las posiciones 5, 3, 1 y 6 de X los valores finales de los no-ceros a ser almacenados en CN. Las filas restantes se procesan exactamente de la misma manera. Es interesante notar que resulta $C_{4,2}=0$. Sin embargo, como la parte simbólica del algoritmo no tiene conocimiento de los valores numéricos de los elementos, el elemento $C_{4,2}$ será tratado como un no-cero y su valor nulo será incluido en CN. La cancelación exacta de un elemento es algo que ocurre rara vez, y la presencia de unos pocos ceros en CN no produce ningún problema. En algunos casos especiales, sin embargo, la cancelación puede ser frecuente, por ejemplo para matrices que tienen todos sus elementos iguales a 1 o a -1. En tales casos puede ser conveniente generar JC y CN al mismo tiempo, fila por fila, y reprocesarlos inmediatamente para eliminar cualquier cero que pueda haber aparecido.

Las mismas ideas que acabamos de describir pueden ser extendidas facilmente para sumar más de dos matrices. También es facil sumar matrices que tengan diferentes números de filas y de columnas. Ahora procederemos a la descripción del algoritmo simbólico y del numérico.

Ejemplo 6. Algoritmo para la suma simbólica de dos matrices ralas con N filas y M columnas.

Datos: IA, JA estructura de la primera matriz en R(C)FD.

IB, JB estructura de la segunda matriz en R(C)FD.

N cantidad de filas de las matrices.

M cantidad de columnas de las matrices.

Resultados: IC, JC estructura de la matriz resultado en R(C)FD.

Vector auxiliar: IX llave múltiple.

```
 1.          IP=1
 2.          DO 10 I=1,M
 3.    10    IX(I)=0
 4.          DO 50 I=1,N
 5.          IC(I)=IP
 6.          IAA=IA(I)
 7.          IAB=IA(I+1)-1
 8.          IF(IAB.LT.IAA)GO TO 30
 9.          DO 20 JP=IAA,IAB
10.          J=JA(JP)
11.          JC(IP)=J
12.          IP=IP+1
13.    20    IX(J)=I
14.    30    IBA=IB(I)
15.          IBB=IB(I+1)-1
16.          IF(IBB.LT.IBA)GO TO 50
17.          DO 40 JP=IBA,IBB
18.          J=JB(JP)
19.          IF(IX(J).EQ.I)GO TO 40
20.          JC(IP)=J
21.          IP=IP+1
22.    40    CONTINUE
23.    50    CONTINUE
24.          IC(N+1)=IP
```

El puntero que recorre JC es IP, que se inicializa al valor 1 en la línea 1 y es incrementado cada vez que un elemento es añadido a la lista JC, en las líneas 12 y 21. También se usa IP para construir el vector de punteros IC en las líneas 5 y 24. El vector de llave múltiple IX se inicializa con el valor 0 en las líneas 2 y 3. I, que se define en la línea 4, identificada cada fila. El ciclo DO 20 recorre la fila I de la primera matriz dato. Los índices de columna son almacenados en JC en la línea 11 y el índice de fila I es almacenado en IX en la línea 13. El ciclo DO 40 recorre la fila I de la segunda matriz dato. Para cada índice de columna J, definido en la línea 18, se hace una comprobación de la llave múltiple en la línea 19: si el valor de IX(J) es I, entonces J ya ha sido añadido a la lista JC y no debe ser añadido nuevamente. De lo contrario, J es añadido a la lista JC en la línea 20. El lector podría esperar que la instrucción IX(J)=I debería aparecer entre las líneas 21 y 22 con el fin de registrar el hecho que el índice de columna J ha sido añadido a JC. Sin embargo, no es necesario registrar este hecho puesto que, durante el procesamiento del resto de la fila I, no se volverá a encontrar el mismo valor de J: no hay índices de columna repetidos en la representación de la fila I en el vector JB.

Ejemplo 7. Algoritmo para la suma numérica de dos matrices ralas con N filas.

Datos: IA, JA, AN primera matriz dada en R(C)FD.

IB, JB, BN segunda matriz dada en R(C)FD.

IC, JC estructura de la matriz resultado en R(C)FD.

N cantidad de filas de las matrices.

Resultado: CN valores numéricos de los no-ceros de la matriz resultado.

Vector auxiliar: X reunión expandida de no-ceros.

```
 1.          DO 70 I=1,N
 2.          IH=I+1
 3.          ICA=IC(I)
 4.          ICB=IC(IH)-1
 5.          IF(ICB.LT.ICA)GO TO 70
 6.          DO 10 IP=ICA,ICB
 7.    10    X(JC(IP))=0.
 8.          IAA=IA(I)
 9.          IAB=IA(IH)-1
10.          IF(IAB.LT.IAA)GO TO 30
11.          DO 20 IP=IAA,IAB
12.    20    X(JA(IP))=AN(IP)
13.    30    IBA=IB(I)
14.          IBB=IB(IH)-1
15.          IF(IBB.LT.IBA)GO TO 50
16.          DO 40 IP=IBA,IBB
17.          J=JB(IP)
18.    40    X(J)=X(J)+BN(IP)
19.    50    DO 60 IP=ICA,ICB
20.    60    CN(IP)=X(JC(IP))
21.    70    CONTINUE
```

Este algoritmo usa el vector expandido X para reunir los no-ceros de cada fila de las matrices dadas, tal como se ha explicado en el ejemplo 5. En las líneas 6 y 7 se cargan ceros en aquellas posiciones de X que corresponden a los no-ceros de la fila I de la matriz resultado. En las líneas 11 y 12, los no-ceros de la fila I de la primera matriz sumando, si hay alguno, se cargan en X. En la línea 18, los no-ceros de la fila I de la segunda matriz se acumulan en X. Finalmente, los

no-ceros resultantes se cargan en CN en la línea 20. Notar que este algoritmo aprovecha el conocimiento previo de las posiciones donde habrá no-ceros, dado por IC, JC, para realizar operaciones sólo con los no-ceros. Notar también que este algoritmo conserva el orden en que se da JC, y construye CN en correspondencia con este orden. En consecuencia, si IC, JC se dá en R(C)FO, lo que se puede conseguir usando dos veces el algoritmo para trasposición simbólica del ejemplo 3 después del algoritmo para suma simbólica del ejemplo 6, el resultado de la suma numérica de este ejemplo quedará también en R(C)FO.

8. Multiplicación de matrices ralas

En esta sección vamos a examinar los algoritmos que se usan para calcular el producto de dos matrices ralas. Dadas las matrices: A de p x q y B de q x r, la matriz producto C de p x r está dada por:

$$C_{ik} = \sum_{j=1}^{q} A_{ij} B_{jk} \qquad \text{para } \begin{matrix} i=1,\ldots,p \\ k=1,\ldots,r \end{matrix} \qquad (1)$$

Esta fórmula da el elemento C_{ik} como el producto interno de la fila i de A por la columna k de B. Las columnas de B,

sin embargo, no se pueden obtener directamenbte cuando B está dado en formato ralo por filas. Una solución posible a este problema es trasponer B, usando el algoritmo de trasposición descripto en la sección 6. Entonces, la ecuación (1) queda:

$$C_{ik} = \sum_{j=1}^{q} A_{ij} \; B^{T}{}_{kj} \qquad (2)$$

que requiere sólo filas tanto de A como de B^T. Aquí vamos a describir un método más eficiente y elegante, que no requiere la trasposición de B y que acepta que tanto A como B estén dadas en formato ralo por filas (19). La idea básica del método es cambiar el orden en que se realizan los productos en la ecuación (1): para i y j fijos, se forman los productos del elemento A_{ij} por todos los elementos B_{jk} de la fila j de B, y los resultados parciales se acumulan en las posiciones k del vector expandido X. Cuando todos los elementos de la fila i de A han sido procesados de esta manera, X contiene la fila i de C completa. En el ejemplo 8 se ilustra este concepto de una manera simple. El algoritmo simbólico se dá en el ejemplo 9, y el algoritmo numérico en el ejemplo 10. El procesamiento simbólico y el numérico se pueden llevar a cabo juntos, pero no se gana mucho al hacerlo. La estructura generada por el algoritmo simbólico queda desordenada, y se puede ordenar usando dos veces el algoritmo de trasposición simbólica. Cuando se hace la multiplicación numérica, se da la estructura

como dato y el resultado sale con el mismo orden que la estructura: o sea que, si está ordenada, el resultado final quedará también ordenado. Los algoritmos emplean la técnica de llave múltiple y el vector expandido para reunión de no-ceros, ya descriptos en la sección 7.

Ejemplo 8. Producto de dos matrices almacenadas por filas.

Consideremos el siguiente producto matricial:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \times \begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} = \begin{vmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{vmatrix}$$

Tenemos:

$$c_{11} = a_{11}\ b_{11} + a_{12}\ b_{21}$$
$$c_{12} = a_{11}\ b_{12} + a_{12}\ b_{22}$$
$$c_{21} = a_{21}\ b_{11} + a_{22}\ b_{21}$$
$$c_{22} = a_{21}\ b_{12} + a_{22}\ b_{22}$$

En lugar de formar los productos en este orden, los calculamos en el orden que sigue:

$$x_1 \leftarrow a_{11}\ b_{11}$$
$$x_2 \leftarrow a_{11}\ b_{12}$$
$$x_1 \leftarrow x_1 + a_{12}\ b_{21}$$
$$x_2 \leftarrow x_2 + a_{12}\ b_{22}$$

$c_{11} \leftarrow x_1$

$c_{12} \leftarrow x_2$

$x_1 \leftarrow a_{21}\ b_{11}$

$x_2 \leftarrow a_{21}\ b_{12}$

$x_1 \leftarrow x_1 + a_{22}\ b_{21}$

$x_2 \leftarrow x_2 + a_{22}\ b_{22}$

$c_{21} \leftarrow x_1$

$c_{22} \leftarrow x_2$

Notar que cada elemento de la primera matriz se multiplica secuencialmente por todos los elementos de una fila de la segunda matriz, los que son muy fáciles de encontrar cuando la segunda matriz está dada en un formato por filas.

Ejemplo 9. Algoritmo para la multiplicación simbólica de dos matrices ralas en NPxNQ y NQxNR, dadas en formato ralo por filas.

Datos: IA, JA estructura de la primera matriz en R(C)FD.
IB, JB estructura de la segunda matriz en R(C)FD.
NP cantidad de filas de la primera matriz.
NQ cantidad de columnas de la primera matriz y de filas de la segunda matriz.
NP cantidad de columnas de la segunda matriz.

Resultados: IC,JC estructura de la matriz resultado en R(C)FD.

Vector Auxiliar: IX de dimensión NR, llave múltiple.

```
 1.          IP=1
 2.          DO 10 I=1,NR
 3.    10    IX(I)=0
 4.          DO 40 I=1,NP
 5.          IC(I)=IP
 6.          IAA=IA(I)
 7.          IAB=IA(I+1)-1
 8.          IF(IAB.LT.IAA)GO TO 40
 9.          DO 30 JP=IAA,IAB
10.          J=JA(JP)
11.          IBA=IB(J)
12.          IBB=IB(J+1)-1
13.          IF(IBB.LT.IBA)GO TO 30
14.          DO 20 KP=IBA,IBB
15.          K=JB(KP)
16.          IF(IX(K).EQ.I)GO TO 20
17.          JC(IP)=K
18.          IP=IP+1
19.          IX(K)=I
20.    20    CONTINUE
21.    30    CONTINUE
22.    40    CONTINUE
23.          IC(NP+1)=IP
```

En este algoritmo, IP es el puntero que señala el primer puesto libre en el vector JC. IP se inicializa en la línea 1, y es incrementado en la línea 18, cada vez que se añade un nuevo elemento a la lista JC. IP se usa también para construir el vector IC en las líneas 5 y 23. El vector de llave múltiple IX se carga con ceros en las líneas 2 y 3. El ciclo DO 40 recorre las NP filas de la primera matriz dato. El ciclo DO 30, en la línea 9, recorre los no-ceros de la fila I, cuyos índices de columna J son definidos en la línea 10. Para cada uno de ellos, el ciclo DO 20 de la línea 14 recorre todos los no-ceros de la fila J de la segunda matriz dato, cuyos índices de columna K se definen en la línea 15. En la línea 16, se comprueba el estado de la llave múltiple IX para determinar si

el elemento K ha sido cargado ya o nó. Si no lo fue, K es añadido a JC en la línea 17 y este hecho se registra en IX en la línea 19.

Ejemplo 10. Algoritmo para la multiplicación numérica de dos matrices ralas dadas en formato ralo por filas.

Datos: IA,JA,AN primera matriz dato en R(C)FD.

IB,JB,BN segunda matriz dato en R(C)FD.

IC,JC estructura de la matriz resultado en R(C)FD.

NP cantidad de filas de la primera matriz dato y de la matriz resultado.

Resultado: CN valores numéricos de los no-ceros de la matriz resultado.

Vector auxiliar: X para reunión expandida de no-ceros.

```
 1.          DO 50 I=1,NP
 2.          ICA=IC(I)
 3.          ICB=IC(I+1)-1
 4.          IF(ICB.LT.ICA)GO TO 50
 5.          DO 10 J=ICA,ICB
 6.    10    X(JC(J))=0.
 7.          IAA=IA(I)
 8.          IAB=IA(I+1)-1
 9.          DO 30 JP=IAA,IAB
10.          J=JA(JP)
11.          A=AN(JP)
12.          IBA=IB(J)
13.          IBB=IB(J+1)-1
14.          IF(IBB.LT.IBA)GO TO 30
15.          DO 20 KP=IBA,IBB
16.          K=JB(KP)
17.    20    X(K)=X(K)+A*BN(KP)
18.    30    CONTINUE
19.          DO 40 J=ICA,ICB
20.    40    CN(J)=X(JC(J))
21.    50    CONTINUE
```

En este ejemplo I identifica una fila de la primera matriz dato y de la matriz resultado. Para cada fila I, en las líneas 5 y 6 se cargan ceros en aquellas posiciones del vector expandido X que corresponden a no-ceros en la matriz resultado. Notar que el programa alcanza la línea 9 sólo cuando la fila I de la matriz resultado no está vacía, lo que a su vez significa que la fila I de la primera matriz no está vacía, de manera que IAB≥IAA en este caso, y la instrucción IF(IAB.LT.IAA)GO TO α precediendo a la línea 9 no es necesaria en este caso. El ciclo DO 30 recorre los no-ceros de la fila I de la primera matriz. El valor del elemento I,J se almacena en la variable no subindicada A en la línea 11. Luego, el ciclo DO 20 recorre los no-ceros de la fila J de la segunda matriz, calculando para cada elemento J,K el producto parcial tal como se describió al principio de esta sección, y acumula el resultado en X(K). Cuando el ciclo DO 30 se completa, el vector X contiene la versión expandida de la fila I de la matriz resultado, la que se carga en CN en las líneas 19 y 20.

Este algoritmo no cambia el ordenamiento de JC. En consecuencia, si IC,JC se dan en R(C)FO como resultado de usar dos veces el algoritmo para trasposición simbólica del ejemplo 3 después del algoritmo de multiplicación simbólica del ejemplo 9, el resultado de la multiplicación numérica quedará también en R(C)FO. El orden es mantenido.

9. Matriz de conectividad de red y matriz de ensamblaje nodal

Existen importantes métodos numéricos, tales como el método de Elementos Finitos (20) y el de Elementos de Borde (21), usados para la resolución de ecuaciones diferenciales en dominios dados, que requieren que el dominio de interés o su frontera sean aproximados por una red de elementos. Los elementos y nodos de la red se numeran arbitrariamente, y la red se describe dando el conjunto de coordenadas de todos los nodos con referencia a algún sistema de coordenadas, y la lista de los números de los nodos que pertenecen a cada elemento. Para cada elemento, los números nodales deben ser dados en un cierto orden convencional. Por ejemplo, para elementos bidimensionales se puede adoptar el orden que resulta de recorrerlos en sentido antihorario. El conjunto de números nodales asociados con los elementos puede ser interpretado como una matriz rala E llamada la matriz de conectividad de la red, que es una matriz booleana. Se puede definir otra importante matriz booleana como la lista de los números de los elementos que comparten cada uno de los nodos de la red. Vamos a demostrar que esta matriz es E^T, la traspuesta de E. Ambas matrices, siendo booleanas, quedan definidas sólo por sus estructuras, IE, JE para E o bien IET, JET para E^T.

Los métodos de elementos requieren el ensamblaje y la resolución de un sistema de ecuaciones lineales. Para problemas escalares, hay una sola incógnita asociada con cada nodo,

y el sistema tiene tantas ecuaciones como nodos hay en la red. La matriz A del sistema es la matriz de ensamblaje nodal o matriz de "rigidez". La matriz A se calcula como una suma de matrices de elemento y tiene la siguiente propiedad: un elemento A_{ij} de la matriz A es diferente de cero si y sólo si los nodos i y j pertenecen al mismo elemento (decimos que los nodos i y j están conectados). Como un nodo i está conectado con sólo unos pocos nodos de la red, la fila i de A tendrá sólo unos pocos no-ceros y, aún más importanre, la cantidad de no-ceros en la fila i será independiente del tamaño de la red. La matriz de ensamblaje nodal A es, por lo tanto, rala, y la cantidad total de no-ceros es proporcional a la cantidad de nodos en la red.

En algunos casos, los valores de las incógnitas asociadas con algunos nodos de la red pueden ser conocidos. Este es el caso para nodos que pertenecen a fronteras o porciones de una frontera donde se dan condiciones de borde de Dirichlet. Nos referiremos a tales nodos como nodos de Dirichlet. La única ecuación asociada con un nodo de Dirichlet es por lo tanto la obvia: valor de la incógnita= valor prescripto, la que no necesita ser ensamblada en el sistema. Sin embargo, esta ecuación obvia es habitualmente ensamblada, dando origen a una fila en la matriz A que contiene un 1 en la diagonal como el único no-cero, y el valor prescripto para la incógnita como lado derecho. La razón para conservar los nodos de Dirichlet en la formulación es que otros nodos están conectados con los

nodos de Dirichlet y tienen ecuaciones en que hay términos conteniendo las incógnitas de Dirichlet. Puesto que las "incógnitas" de Dirichlet son en realidad conocidas, esos términos deben ser eliminados de las ecuaciones pasándolos al segundo miembro (20, pág.457), y esta operación resulta computacionalmente más simple cuando los nodos de Dirichlet están presentes en la formulación. Esta técnica resulta particularmente conveniente cuando se usan formatos ralos debido a la facilidad con que se pueden manejar las filas vacías en A, sin producir casi ningún aumento de esfuerzo.

Una práctica habitual es almacenar y procesar A como una matriz de banda. Como el ancho de banda aumenta cuando la cantidad de nodos en la red aumenta, para una red grande se hace necesario almacenar y procesar una gran cantidad de ceros. Por eso, para problemas grandes, las técnicas de matrices ralas son computacionalmente más convenientes. Una ventaja más de la notación empleada para matrices ralas es que no impone ninguna restricción a la numeración de los nodos y de los elementos, que es completamente arbitraria, no existiendo el concepto de ancho de banda. Una desventaja del uso de matrices de banda es que los nodos de Dirichlet pueden causar un aumento innecesario del ancho de banda.

En el ejemplo 11 damos la matriz de conectividad y la de ensamblaje nodal para una red simple. Los algoritmos para el ensamblaje simbólico y numérico de A se dan en los ejemplos 12 y 13. El problema de construir la matriz de conectividad E

pertenece a una categoría más general de problemas, los relacionados con la generación de redes, que se tratan en otras publicaciones (22).

Ejemplo 11. Matriz de conectividad y matriz de ensamblaje nodal para una red simple.

Consideremos la red bidimensional que se muestra en la Fig.1 con N=9 nodos y NEL=4 elementos, numerados de una manera arbitraria. Los números de los elementos aparecen dentro de círculos. La matriz de conectividad E de NELxN para este problema es:

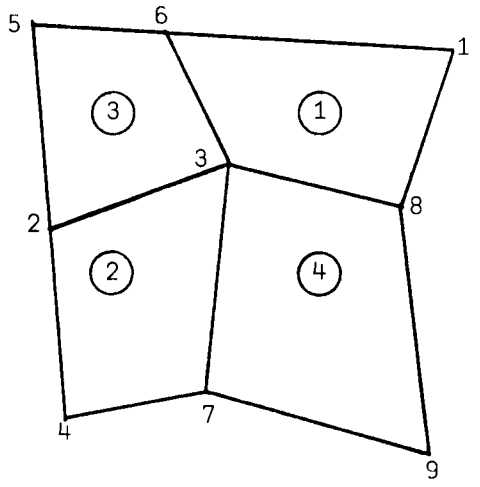


FIGURA 1

$$E = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Cada fila de esta matriz está asociada con un elemento, y cada columna con un nodo. La fila 1 tiene no-ceros en las posiciones 1, 3, 6 y 8, porque los nodos 1, 3, 6 y 8 pertenecen al elemento 1. Los no-ceros de la columna 7 están en las posiciones 2 y 4, pues el nodo 7 pertenece a los elementos 2 y 4. La matriz E expresada en esta forma no establece ningún orden para los nodos de cada elemento, excepto el orden natural, en que los números de los nodos están en orden creciente. Pero cuando E se dá en formato ralo por filas, se pueden dar los

números nodales en el orden convencional aceptado para elementos. Esta es otra ventaja más del formato ralo. Por ejemplo, para elementos bidimensionales se suelen dar los nodos en el orden en que aparecen cuando el elemento se recorre en sentido antihorario, en cuyo caso tendríamos:

IE = 1 5 9 13 17

JE = 3 8 1 6 , 7 3 2 4 , 5 2 3 6 , 7 9 8 3

en R(C)FD. Basta dar sólo la estructura de E pues los valores de todos los no-ceros son iguales a 1. Notar que esta representación es desordenada, según la definición de formato ralo, sin embargo, es justamente esta facilidad de poder dar los elementos en cualquier orden la que se aprovecha para darlos en el orden convencional. Sólo la selección del primer nodo de cada secuencia es arbitraria. Representemos también la traspuesta de E en formato ralo:

IET = 1 2 4 8 9 10 12 14 16 17

JET = 1 , 2 3 , 1 2 3 4 , 2 , 3 , 1 3 , 2 4 , 1 4 , 4

en R(C)FO. Esta representación se puede obtener fácilmente a partir de IE,JE usando el algoritmo de trasposición simbólica de la sección 6. En JET están listados los números de los elementos asociados con cada nodo. Por ejemplo, 2,3 en la fila 2 indica que el nodo 2 pertenece sólo a los elementos 2 y 3. Algunos programas pueden requerir ambas representaciones IE,JE y IET,JET simultaneamente. La matriz de ensamblaje nodal para la red de la figura 1 es:

$$A = \begin{array}{c|ccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & X & 0 & X & 0 & 0 & X & 0 & X & 0 \\ 2 & 0 & X & X & X & X & X & X & 0 & 0 \\ 3 & X & X & X & X & X & X & X & X & X \\ 4 & 0 & X & X & X & 0 & 0 & X & 0 & 0 \\ 5 & 0 & X & X & 0 & X & X & 0 & 0 & 0 \\ 6 & X & X & X & 0 & X & X & 0 & X & 0 \\ 7 & 0 & X & X & X & 0 & 0 & X & X & X \\ 8 & X & 0 & X & 0 & 0 & X & X & X & X \\ 9 & 0 & 0 & X & 0 & 0 & 0 & X & X & X \end{array}$$

donde sólo mostramos la estructura, pues los valores numéricos se pueden determinar solamente calculando y ensamblando las matrices de elemento asociadas con cada elemento y con el problema que se desea resolver. En muchos casos prácticos A es una matriz simétrica, y su estructura es siempre simétrica. Suponiendo qúe A es simétrica en este ejemplo, su estructura se puede representar como sigue, en R(S)FO:

IA = 1 4 9 15 16 17 18 20 21 21

JA = 3 6 8 , 3 4 5 6 7 , 4 5 6 7 8 9 , 7, 6, 8 , 8 9 , 9

Esta representación se puede obtener, desordenada, a partir de IE, JE y de IET,JET usando el algoritmo del ejemplo 12. Notar que hay sólo 20 no-ceros fuera de la diagonal. Si A fuera una matriz de banda, el semiancho de banda mínimo sería 5 para este ejemplo, y la cantidad de elementos fuera de la diagonal sería 26, lo que muestra la ventaja de las representaciones ralas aún para una red pequeña.

Tal como se explicó al comienzo de esta sección, es computacionalmente conveniente que las ecuaciones asociadas con los nodos de Dirichlet queden en el sistema, correspondiendo a filas de A con un 1 en la diagonal como el único no-cero. Tam-

bién se explicó que los términos que conectan otros nodos con nodos de Dirichlet deben ser removidos de las ecuaciones correspondientes, pasándolos a los segundos miembros. Supongamos que los nodos 2, 4 y 5 de la figura 1 son nodos de Dirichlet. La matriz de ensamblaje nodal es en este caso:

$$A = \begin{array}{c|ccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & X & 0 & X & 0 & 0 & X & 0 & X & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & X & 0 & X & 0 & 0 & X & X & X & X \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 6 & X & 0 & X & 0 & 0 & X & 0 & X & 0 \\ 7 & 0 & 0 & X & 0 & 0 & 0 & X & X & X \\ 8 & X & 0 & X & 0 & 0 & X & X & X & X \\ 9 & 0 & 0 & X & 0 & 0 & 0 & X & X & X \end{array}$$

donde los elementos diagonales 2, 4 y 5 son ahora iguales a 1. La estructura de esta matriz en formato ralo en R(S)FO es:

IA = 1 4 4 8 8 8 9 11 12 12

JA = 3 6 8 , 6 7 8 9 , 8 , 8 9 , 9

que sólo tiene 11 no-ceros fuera de la diagonal, mientras que la representación como matriz de banda tendría 26, como antes.

Ejemplo 12. Algoritmo para el ensamblaje simbólico de una matriz de ensamblaje nodal o de "rigidez" simétrica.

Datos: IE,JE matriz de conectividad nodal de la red en R(C)FD.

IET,JET traspuesta de IE,JE en R(C)FD.

N cantidad de nodos en la red.

IA vector de dimensión N que contiene el valor N en las posiciones correspondientes a nodos de Dirichlet, y 0 en las demás.

Resultados: IA,JA estructura de la matriz de ensamblaje nodal, de dimensión NxN, en R(S)FD.

```
 1.          JP=1
 2.          NM=N-1
 3.          DO 30 I=1,NM
 4.          JPI=JP
 5.          IF(IA(I).EQ.N)GO TO 30
 6.          IETA=IET(I)
 7.          IETB=IET(I+1)-1
 8.          DO 20 IP=IETA,IETB
 9.          J=JET(IP)
10.          IEA=IE(J)
11.          IEB=IE(J+1)-1
12.          DO 10 KP=IEA,IEB
13.          K=JE(KP)
14.          IF(K.LE.I)GO TO 10
15.          IF(IA(K).GE.I)GO TO 10
16.          JA(JP)=K
17.          JP=JP+1
18.          IA(K)=I
19.    10    CONTINUE
20.    20    CONTINUE
21.    30    IA(I)=JPI
22.          IA(N)=JP
23.          IA(N+1)=JP
```

Este algoritmo es en esencia similar a la multiplicación simbólica, dada en el ejemplo 9, donde los factores son E^T y E, y el resultado es la estructura de A (19). El vector IA desempeña tres papeles diferentes: como dato de entrada identifica a los nodos de Dirichlet, si hay alguno; durante la ejecución es usado como vector de llave múltiple (ver la sección 7); y a la salida es el vector de punteros para JA. Durante el procesamiento de una fila I, los punteros para las I-1 filas iniciales están almacenados en las posiciones 1 hasta I-1 de IA, mientras que las posiciones I hasta N de IA se usan para la llave múltiple. Esto se logra sin procesamiento adicional gracias a que para la fila I, como la representación es superior, sólo interesan los índices de columna mayores que I.

El puntero para la lista JA es JP, que es inicializado en la línea 1 e incrementado en la línea 17. El DO 30 procesa sólo N-1 filas porque la fila N está necesariamente vacía, ya que no tiene elementos a la derecha de la diagonal. La línea 5 tiene en cuenta las ecuaciones asociadas con nodos de Dirichlet: las filas correspondientes se dejan vacías. El ciclo DO 20 de la línea 8 recorre todos los elementos del nodo I: notar que IETB≥IETA, pues no hay nodos que no pertenezcan a ningún elemento. En la línea 9, J es el número de uno de los elementos del nodo I. El ciclo DO 10 de la línea 12 recorre todos los nodos del elemento J: notar que necesariamente IEB>IEA pues no hay elementos con menos de dos nodos. En la línea 13, K es el número de un nodo del elemento J. En la línea 14, se descartan todos los nodos que no están a la derecha del elemento diagonal. En la línea 15 se comprueba el estado de la llave múltiple: aquí hay que notar que IA(K)=I significaría que el nodo K ya ha sido inscripto en la lista JA y no debe ser inscripto una segunda vez, mientras que IA(K)=N significa que K es un nodo de Dirichlet que no debe ser inscripto en JA en absoluto; por estas razones se utiliza .GE.en la línea 15. La inscripción de los nodos restantes se realiza en la línea 16 y la llave múltiple es bloqueada en consecuencia en la línea 18. JP, el puntero de la lista JA, es utilizado para almacenar los valores de los punteros de fila en las líneas 21, 22 y 23.

Ejemplo 13. Algoritmo para el ensamblaje numérico de una matriz de elemento llena en la matriz de ensamblaje nodal A.

Datos: IA,JA estructura de la matriz A en R(S)FD.

IDIR vector que identifica los nodos de Dirichlet. Contiene 1 para un nodo de Dirichlet, y 0 para un nodo comun.

AE,BE matriz de elemento y vector de segundos miembros de elemento, respectivamente, a ser ensamblados en AN, AD y en B, respectivamente.

JEP números de los nodos asociados con las filas y columnas de AE.

N orden de A y de B.

NN orden de AE y BE.

Resultados: AN, AD valores numéricos de los no-ceros de A en R(S)FD.

B vector de segundos miembros del sistema de ecuaciones lineales.

Vector auxiliar: IP, de dimensión N, inicializado a 0. IP es usado y luego reinicializado a 0 por el algoritmo.

Nota: Los valores prescriptos para las incógnitas de Dirichlet deben estar almacenados en las posiciones correspondientes de B, las que no serán alteradas. En otras palabras, si i es un nodo de Dirichlet con condición en la frontera C_i, entonces, antes de usar este algoritmo, se debe definir IDIR(i)=1, B(i)=C_i y AD(i)=1.

```
 1.          DO 40 L=1,NN
 2.          LNN=L-NN
 3.          I=JEP(L)
 4.          IF(IDIR(I).NE.0)GO TO 40
 5.          AD(I)=AD(I)+AE(LNN+L*NN)
 6.          B(I)=B(I)+BE(L)
 7.          K=0
 8.          DO 20 LL=1,NN
 9.          IF(LL.EQ.L)GO TO 20
10.          J=JEP(LL)
11.          IF(IDIR(J).NE.0)GO TO 10
12.          IF(J.LT.I)GO TO 20
13.          IP(J)=LL
14.          K=1
15.          GO TO 20
16.   10     B(I)=B(I)-B(J)*AE(LNN+LL*NN)
17.   20     CONTINUE
18.          IF(K.EQ.0)GO TO 40
19.          IAA=IA(I)
20.          IAB=IA(I+1)-1
21.          DO 30 J=IAA,IAB
22.          LL=IP(JA(J))
23.          IF(LL.EQ.0)GO TO 30
24.          AN(J)=AN(J)+AE(LNN+LL*NN)
25.          IP(JA(J))=0
26.   30     CONTINUE
27.   40     CONTINUE
```

Este algoritmo acumula la matriz de elemento AE de NNxNN en las posiciones apropiadas de la matriz de ensamblaje nodal AN,AD de NxN, y el vector de elemento BE de largo NN en las posiciones correspondientes del vector B de segundos miembros, de largo N. Las ecuaciones asociadas con nodos de Dirichlet no son ensambladas (línea 4) y los términos que conectan nodos normales con nodos de Dirichlet son directamente sustraídos de B (línea 16). A pesar de que éste es, en el fondo, un algoritmo de adición, las técnicas empleadas son diferentes de las que se discutieron en el ejemplo 7. En vez de usar un vector expandido real X para reunir los no-ceros, usamos un vector entero IP en el que se construye un conjunto de punteros a

cada elemento de cada fila de AE que debe ser ensamblado. AE está almacenado por columnas, y el elemento (L,LL) de AE es por lo tanto AE(L-NN+LL*NN) (ver las líneas 5, 16 y 24). El ciclo DO 40 recorre las NN filas de AE. En la línea 3, I es el índice global de fila asociado con la fila L de AE. El elemento diagonal de esta fila es ensamblado en la línea 5 y el segundo miembro en la línea 6. El cilo DO 20 recorre los demás elementos de la fila L de AE. El índice de columna global J es calculado en la línea 10. En la línea 12 descartamos todos los elementos que están a la izquierda de la diagonal. Cuando la ejecución alcanza a la línea 13, ya se ha tomado la decisión de que el elemento (L,LL) de AE debe ser ensamblado en la posición (I,J) de la matriz global, y este hecho es registrado almacenando LL en IP(J). K se define en las líneas 7 y 14, y se usa en la línea 18 para saber si hay algún elemento que ensamblar o si no hay ninguno; de esta manera se ahorra tiempo de computadora cuando no hay nada que ensamblar. El ciclo DO 30 recorre la fila I de la matriz A. En la línea 22, los valores de LL son obtenidos desde IP y en la línea 24 el elemento correspondiente de AE es acumulado en la posición correcta de AN. IP se reinicializa a 0 en la línea 25.

Notar que los elementos de AE no pueden ser acumulados directamente en AN debido a la falta de orden de ambas listas de índices de columna JEP y JA. Este algoritmo recorre primero JEP, anotando los punteros expandidos IP, y después recorre JA usando los punteros expandidos para encontrar los elementos

requeridos en AE. De esta manera, se realiza un número fijo de operaciones por cada elemento de AE, y el número total de operaciones es por lo tanto linealmente proporcional a la cantidad total de no-ceros a ser ensamblados.

10. Factorización triangular de una matriz rala

En esta sección nos proponemos examinar el siguiente problema: dada una matriz simétrica definida positiva A en formato ralo, queremos determinar su factor triangular superior U y su factor diagonal, tales que:

$$A = U^T D U$$

La factorización triangular se obtiene realizando la eliminación de Gauss en la matriz A. Por simplicidad vamos a suponer que A ya está preparada para la eliminación de Gauss,en el sentido que se puede usar directamente como pivotes los elementos diagonales en el mismo orden en que están dados en el vector AD. También vamos a suponer que ha sido resuelto otro aspecto importante de la eliminación de Gauss en matrices ralas, del que no nos vamos a ocupar aquí: la selección de ordenamientos que minimicen la generación de nuevos no-ceros durante la eliminación (23-27).

La aplicación más importante de la factorización triangular es la resolución directa de sistemas de ecuaciones lineales por eliminación de Gauss, que será discutida en la sección

11. Otra aplicación posible es el cálculo de determinantes, ya que el determinanrte de A es igual al determinante de D, que es simplemente el producto de todos sus elementos diagonales.

La eliminación de Gauss puede realizarse por filas o por columnas. La versión por filas es más conveniente para una matriz dada en formato ralo por filas. En la versión por filas, se restan de cada fila de A, elemento por elemento, múltiplos de todas las filas precedentes, de tal manera que la nueva fila I contenga sólo ceros a la izquierda de la diagonal. Después, dividimos los elementos restantes de la fila I por el elemento diagonal para formar el factor triangular superior U (cuyos elementos diagonales son unitarios), y almacenamos separadamente el elemento diagonal para formar la matriz diagonal D.

La factorización triangular de una matriz rala se hace primero simbólicamente y después numéricamente. La factorización simbólica necesita la estructura IA,JA de la matriz A, que puede estar desordenada, y genera la estructura IU,JU de U, que resulta desordenada aún si JA estuviera ordenado. La factorización numérica, por su parte, requiere que IU,JU esté ordenado, mientras que IA,JA,AN pueden darse en una representación desordenada. Por lo tanto, hay que usar dos veces el algoritmo de trasposición simbólica descripto en la sección 6 para ordenar IU,JU, después de la factorización simbólica y antes de proceder a la factorización numérica. Las dificultades algorítmicas que se encuentran cuando una matriz rala

simétrica, dada en una representación superior, es factorizada, se entienden mejor con la ayuda de un ejemplo explícito, ver ejemplo 14. Los algoritmos para la factorización simbólica y la numérica se dan en los ejemplo 15 y 16.

Ejemplo 14. Factorizaicón triangular de una matriz rala simétrica representada en formato ralo por filas.
Consideremos la siguiente matriz simétrica, donde por simplicidad hemos omitido los valores numéricos de los no-ceros:

$$A = \begin{array}{c|ccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \hline 1 & X & 0 & 0 & 0 & 0 & X & 0 & 0 & 0 & X & 0 \\ 2 & 0 & X & 0 & X & 0 & 0 & 0 & 0 & X & 0 & 0 \\ 3 & 0 & 0 & X & 0 & X & 0 & X & 0 & 0 & 0 & X \\ 4 & 0 & X & 0 & X & 0 & 0 & X & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & X & 0 & X & 0 & 0 & X & 0 & 0 & 0 \\ 6 & X & 0 & 0 & 0 & 0 & X & 0 & 0 & X & 0 & 0 \\ 7 & 0 & 0 & X & X & 0 & 0 & X & 0 & 0 & 0 & X \\ 8 & 0 & 0 & 0 & 0 & X & 0 & 0 & X & 0 & 0 & X \\ 9 & 0 & X & 0 & 0 & 0 & X & 0 & 0 & X & 0 & X \\ 10 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & X \\ 11 & 0 & 0 & X & 0 & 0 & 0 & X & X & X & X & X \end{array}$$

La factorización triangular de esta matriz comienza por la cuarta fila, pues ninguna de las filas anteriores tiene no-ceros a la izquiera de la diagonal. De la fila 4 restamos la fila 2, elemento por elemento, multiplicada por una constante tal que el elemento A_{42} resulte cancelado. Al hacerlo, se introduce un nuevo no-cero en la fila 4: el elemento A_{49}. A continuación, de la fila 5 restamos un múltiplo de la fila 3 para eliminar A_{53}, introduciendo nuevos no-ceros en las posiciones A_{57} y $A_{5,11}$. Después eliminamos A_{61}, introduciendo un nuevo no-cero en $A_{6,10}$. Suponiendo ahora que la factorización

está completa hasta este punto, vamos a examinar en detalle como se procesa la fila 7 para eliminar A_{73} y A_{74}.

La primera dificultad es que sólo el triángulo superior y la diagonal de A están representadas en la computadora. No hay información que nos diga que A_{73} y A_{74} son diferentes de cero y deben ser eliminados. Tampoco existe ninguna información que nos diga que un nuevo no-cero será introducido en A_{75} cuando la fila 3 sea sustraida, y que A_{75} tendrá que ser eliminado a su vez. Lo que si sabemos es que los simétricos de los elementos mencionados, A_{37} y A_{47}, son no-ceros, y que un nuevo no-cero fue introduciddo en A_{57}, el simétrico de A_{75}, cuando la fila 5 fue procesada. Dicho en otras palabras, lo que tenemos que hacer es examinar los no-ceros de la columna 7 por encima de la diagonal para encontrar los no-ceros de la fila 7 a la izquierda de la diagonal que deben ser eliminados.

En segundo lugar, sabemos que los elementos de la fila 7 a la izquierda de la diagonal tomarán el valor cero, así que no necesitamos calcularlos. En otras palabras, basta con calcular los elementos que están sobre la diagonal o a su derecha, en la fila 7, y para hacerlo sólo se necesitan aquellos elementos de las filas 3, 4 y 5 que estén sobre la columna 7 o a su derecha. Esto parece requerir una representación ordenada a efectos de poder determinar fácilmente cuales son los elementos de cada fila cuyo índice de columna es 7 o más. Sin embargo, veremos más abajo, que este requerimiento sólo se aplica a la eliminación numérica y no a la simbólica.

Supongamos entonces haber realizado los pasos necesarios para ubicar los elementos que necesitamos. Tendremos así las siguientes listas de índices de columna, correspondientes a elementos en la columna 7 o a su derecha:

fila 3: 7 11

fila 4: 7 9

fila 5: 7 8 11

fila 7: 7 11

Estas listas de índices de columna deben ser reunidas para obtener la estructura de la nueva fila 7. Esto se hace usando la técnica de llave múltiple descrita en la sección 7, y se obtiene:

nueva fila 7: 7 11 9 8

en una representación desordenada. Hay sin embargo, una manera mejor de hacer este paso, aprovechando la siguiente propiedad de la eliminación de Gauss por filas (24): es necesario y suficiente considerar sólo aquellas filas cuyo no-cero en la columna 7 es el primer no-cero de la fila a la derecha de la diagonal (además de la fila 7 misma). En este caso, la fila 3 no es tenida en cuenta porque su primer no-cero a la derecha de la diagonal es A_{35}, que está a la izquierda de la columna 7. La propiedad que estamos usando se puede entender fácilmente con este ejemplo: puesto que $A_{35} \neq 0$, también por simetría $A_{53} \neq 0$; esto significa que A_{53} fue eliminado al procesar la fila 5 restando la fila 3, y al hacerlo todos los no-ceros de la fila 3 fueron reunidos con los de la fila 5; de tal manera que, cuan-

do tomamos los no-ceros de la fila 5, estamos de hecho tomando los de la fila 3 y no necesitamos considerar la fila 3 por separado. Por lo tanto nos basta considerar las estructuras siguientes:

fila 4: 7 9

fila 5: 7 8 11

fila 7: 7 11

las que una vez reunidas producen la estructura correcta:

nueva fila 7: 7 9 8 11

Es importante notar que estas consideraciones, que se refieren a la parte simbólica de la eliminación de Gauss, no requieren que las representaciones de las filas sean ordenadas. Todo lo que necesitamos saber es qué filas tienen su primer no-cero de la derecha de la diagonal en una columna dada. Esto significa que los restantes no-ceros de tales filas están a la derecha de la columna de interés y deben ser tenidos en cuenta todos ellos para generar la estructura de la nueva fila sin importar en que orden están, y con la ayuda de la técnica de llave múltiple. Por lo tanto, la parte simbólica de la eliminación de Gauss no requiere que la representación sea ordenada. Este es un resultado muy afortunado, ya que, como lo estamos viendo en este ejemplo, el propio algoritmo simbólico genera las estructuras en desorden.

Para saber cuales son las filas que tienen su primer no-cero en una columna dada, debemos asociar un conjunto de filas a cada columna. Por ejemplo, las filas 4 y 5 pertenecen

al conjunto asociado con la columna 7. En la práctica, lo que se hace es determinar el índice de columna más bajo de cada fila al mismo tiempo que se está generando la estructura de esa fila, y anotar de inmediato esa fila en el conjunto asociado con esa columna. Por ejemplo, al ensamblar la estructura de la fila 7 encontramos que 8 es el índice de columna más bajo a la derecha de la diagonal, e inscribimos inmediatamente la fila 7 en el conjunto asociado con la columna 8. Más tarde, cuando se procese la fila 8, encontraremos el índice de fila 7 en el conjunto correspondiente, y usaremos la estructura de la fila 7.

Es útil notar que los conjuntos asociados con cada columna son disjuntos, en el sentido de que cada fila pertenece a una y sólo una columna. Por lo tanto, todos los conjuntos pueden ser almacenados en un solo vector IP de largo N, donde N es el orden de la matriz, en la forma de listas circulares encadenadas. De hecho, en el momento que una fila I está siendo procesada, todos los conjuntos pueden ser almacenados en las I-1 posiciones iniciales de IP, pues sólo las filas 1 hasta I-1 podrán estar inscriptas en los conjuntos en ese momento. Las demás posiciones de IP se pueden usar para almacenar punteros por columna a todas las cadenas, pues solo las columnas I hasta N son de interés en ese momento. Más detalles se pueden encontrar en el ejemplo 16, donde presentamos y discutimos un algoritmo para la eliminación de Gauss simbólica.

El vector IU también puede ser usado para un doble propósito. En el momento que la fila I está siendo procesada, los punteros por fila a JU correspondientes a las filas precedentes están almacenados en las posiciones 1 hasta I-1 de IU. Las demás posiciones están libres. Como los índices de columna que se van a inscribir en JU son todos iguales o mayores que I, podemos aprovechar las posiciones desde I hasta N de IU para usarlas como vector de llave múltiple (ver la sección 7).

Ejemplo 15. Factorización triangular numérica de una matriz rala simétrica dada en formato ralo por filas.
En este ejemplo consideramos la misma matriz dada en el ejemplo 14. Suponemos que ya se ha completado la factorización y ordenamiento de la estructura y que tenemos IU,JU en formato ralo ordenado, y ahora estamos interesados en la parte numérica de la eliminación de Gauss. Igual como lo hicimos en el ejemplo anterior, vamos a suponer que la eliminación está completa hasta la fila 6 inclusive, y vamos a examinar de que manera se procesa la fila 7. Sabemos que la fila 7 tiene no-ceros en las posiciones 7, 8, 9 y 11.Para encontrar los valores de esos no-ceros, debemos examinar la columna 7, encontrar que las filas 3, 4 y 5 tienen no-ceros en ella, encontrar los elementos de esas filas que tengan índices de columna iguales o mayores que 7, y finalmente multiplicarlos por constantes convenientes y restarlos de la fila 7. Este procedimiento implica llevar a cabo varios pasos, tres de los cuales serán discutidos en detalle:

- Encontrar los elementos de una fila dada (digamos la fila 3) que tengan índices de columna iguales o mayores que el índice de la fila que está siendo procesada (7 en este ejemplo). Los punteros inicial y final a la descripción de la fila 3 en JU y UN son IU(3) y IU(4)-1, respectivamente. La fila 3 está ordenada y tiene no-ceros en las posiciones:

fila 3: 5 7 11

Pero estamos interesados sólo en la parte de la fila 3 que empieza en la columna 7. Esto requiere un puntero inicial diferente, al que llamaremos IUP, mientras que IU(4)-1 sigue siendo el puntero final. En el momento en que la fila 7 está siendo procesada, IUP(3) señala el no-cero de la fila 3 que está en la columna 7. Tan pronto como la fila 3 ha sido usada en el proceso de la fila 7, IUP(3) es aumentado. Puesto que la fila 3 está ordenada, IUP(3) señalará ahora el siguiente no-cero de la fila 3, y permanecerá allí hasta que la fila 11 sea procesada, pues la fila 3 no tiene ningún no-cero en las columnas 8, 9 ni 10 y por lo tanto la fila 3 no será usada e IUP(3) no será modificado cuando las filas 8, 9 y 10 sean procesadas.

- Encontrar qué filas tiene no-ceros en una columna dada (digamos en la columna 7). Ahora es necesario encontrar todas las filas que tengan un no-cero en la columna 7. La situación es diferente de la que se discutió en el ejemplo 14, donde sólo era necesario encontrar qué filas tenían su primer no-cero en la columna 7 para el proceso simbólico. En este caso, las

filas también se inscriben en conjuntos asociados con cada columna, pero, después que cada fila es usada, no se la descarta sino que se la inscribe en el conjunto correspondiente al siguiente no-cero de esa fila. Excepto por este detalle, todos los conjuntos se almacenan igual que antes en las primeras I-1 posiciones de un vector IP de dimensión N, mientras que las restantes posiciones de IP se usan para almacenar los punteros a cada uno de los conjuntos.

- Restar múltiplos de porciones de las filas seleccionadas de la fila bajo proceso (digamos la fila 7). Los elementos de cada fila, digamos la fila 3, tienen que ser multiplicados por un factor, que es la relación entre los elementos A_{37} (el simétrico de A_{73}) y A_{33}. Puesto que IUP(3) señala a A_{37}, y A_{33} es DI(3) (en realidad, DI(3) es la inversa de A_{33}, pues resulta conveniente almacenar en DI nó los elementos de la matriz diagonal D sino sus inversas), la relación mencionada se puede calcular fácilmente. Por supuesto, también se requiere un vector expandido para reunir las listas de números reales. Para este fin se usan las posiciones I hasta N del vector DI, las que están libres en el momento en que se está procesando la fila I. De esta manera se logra un uso muy eficiente del almacenamiento disponible. En los ejemplos 16 y 17 presentamos los algoritmos para la factorización simbólica y la numérica, respectivamente.

Ejemplo 16. Algoritmo para la factorización triangular simbólica de una matriz rala simétrica A.

Datos: IA,JA estructura de la matriz dada en R(S)FD.

N orden de la matriz A y de la matriz U.

Resultados: IU,JU estructura de la matriz resultante en R(S)FD.

Vector auxiliar: IP de dimensión N: listas encadenadas de filas asociadas con cada columna (ver ejemplo 14).

Nota: el vector IU se usa también como vector de llave múltiple (ver ejemplo 14).

```
1.          NM=N-1
2.          NH=N+1
3.          DO 10 I=1,N
4.          IU(I)=0
5.    10    IP(I)=0
6.          JP=1
7.          DO 90 I=1,NM
8.          JPI=JP
9.          JPP=N+JP-I
10.         MIN=NH
11.         IAA=IA(I)
12.         IAB=IA(I+1)-1
13.         IF(IAB.LT.IAA)GO TO 30
14.         DO 20 J=IAA,IAB
15.         JJ=JA(J)
16.         JU(JP)=JJ
17.         JP=JP+1
18.         IF(JJ.LT.MIN)MIN=JJ
19.   20    IU(JJ)=I
20.   30    LAST=IP(I)
21.         IF(LAST.EQ.0)GO TO 60
22.         L=LAST
23.   40    L=IP(L)
24.         LH=L+1
25.         IUA=IU(L)
26.         IUB=IU(LH)-1
27.         IF(LH.EQ.I)IUB=JPI-1
28.         IU(I)=I
29.         DO 50 J=IUA,IUB
30.         JJ=JU(J)
```

```
31.          IF(IU(JJ).EQ.I)GO TO 50
32.          JU(JP)=JJ
33.          JP=JP+1
34.          IU(JJ)=I
35.          IF(JJ.LT.MIN)MIN=JJ
36.    50    CONTINUE
37.          IF(JP.EQ.JPP)GO TO 70
38.          IF(L.NE.LAST)GO TO 40
39.    60    IF(MIN.EQ.NH)GO TO 90
40.    70    L=IP(MIN)
41.          IF(L.EQ.0)GO TO 80
42.          IP(I)=IP(L)
43.          IP(L)=I
44.          GO TO 90
45.    80    IP(MIN))=I
46.          IP(I)=I
47.    90    IU(I)=JPI
48.          IU(N)=JP
49.          IU(NH)=JP
```

Este algoritmo opera en base a los principios que han sido discutidos en el ejemplo 14. Aquí vamos a dar solamente algunas indicaciones que ayudan a entender mejor el programa. En las líneas 4 y 5, se inicializan a 0 el vector de llave múltiple IU y el vector de conjuntos encadenados IP. El puntero que señala la primera posición libre de JU es JP, que se inicializa al valor 1 en la línea 6 y es incrementado en las líneas 17 y 33, cada vez que un nuevo índice de columna se inscribe en la lista JU.

El ciclo DO 90 recorre las filas de la matriz. Para cada fila I, el valor de JP se almacena temporariamente en JPI en la línea 8, y definitivamente en IU(I) en la línea 47, cuando la posición I de IU ya no será necesaria para la llave múltiple. La finalidad del ciclo DO 20 es transferir los índices de columna JJ, correspondientes a la fila I de A, desde JA a JU; como es usual, el índice I es almacenado en el vector de llave

múltiple en la línea 19; en la línea 18 se obtiene en MIN el índice de columna más bajo. MIN fue inicializado ya en la línea 10.

El puntero a la cadena circular donde se listan las filas asociadas con la columna I, se saca de IP(I) en la línea 20 y se carga en LAST. LAST es en realidad el índice de una de esas filas, la "última" fila. La línea 21 separa el caso en que no hay filas asociadas con la columna I. En la línea 23, L es el "siguiente" elemento en la cadena. La línea 27 sirve para el caso I=LH, pues IU(I) no está definido todavía como un puntero, y en la línea 28 el índice de llave I es almacenado en la posición I de IU para impedir que I sea inscripto como índice de columna en JU. El ciclo DO 50 recorre la fila L de la matriz U. En la línea 31 se comprueba el estado del vector llave IU, en la línea 32 los índices de columna necesarios JJ son añadidos en JU, y cualquier intento futuro de volver a inscribir JJ en JU queda bloqueado al almacenar I en IU(JJ) en la línea 34. El índice de columna mínimo se encuentra en la línea 35.

JPP, que se calcula en la línea 9, es el valor que tendría JP si <u>todos</u> los no-ceros posibles existieran realmente en la fila I a la derecha de la diagonal. Esta condición se comprueba en la línea 37, y en el caso de que todos los no-ceros posibles hayan sido anotados ya se pasa directamente a la fila siguiente, ahorrando así tiempo de máquina. Esta situación se puede presentar con cierta frecuencia, particularmente para

las últimas filas de una matriz que tienen una gran probabilidad de quedar llenas.

Si L no es la "última" fila en la cadena asociada con la columna I, el programa se desvía desde la línea 38 a la 23, donde la siguiente fila en la cadena es seleccionada. Cuando se termina la cadena, el programa alcanza la línea 39, donde la condición MIN=NH significaría que la fila I de U está vacía a la derecha de la diagonal, en cuyo caso el programa se desvía a la línea 47. De lo contrario, hay uno o más no-ceros en la fila I, y la fila I debe ser inscripta en la cadena asociada con la columna MIN. En la línea 40 se detecta el puntero L para esta cadena, la cadena es cortada en la línea 42 y la fila I es inscripta en la línea 43. Si la cadena estuviera aún vacía, el algoritmo salta de la línea 41 a la 45, donde se define el puntero a la nueva cadena, y la nueva cadena es creada con el único elemento I en la línea 46.

Ejemplo 17. Algoritmo para la factorización triangular numérica de una matriz rala A simétrica y definida positiva.

Datos: IA,JA,AN,AD matriz dada A en R(S)FD.

IU,JU estructura de la matriz resultado U en R(S)FO.

N orden de las matrices A y U.

Resultados: UN valores numéricos de los no-ceros de la matriz U, en R(S)FO.

DI inversa de la matriz diagonal llena D.

Vectores auxiliares: IP de dimensión N; listas encadenadas de filas asociadas con cada columna (ver ejemplo 15).

IUP de dimensión N; punteros auxiliares a porciones de filas (ver ejemplo 15).

DI es usado también como vector expandido para reunión de no-ceros.

```
      DO 10 J=1,N
10    IP(J)=0
      DO 130 I=1,N
      IH=I+1
      IUA=IU(I)
      IUB=IU(IH)-1
      IF(IUB.LT.IUA)GO TO 40
      DO 20 J=IUA,IUB
20    DI(JU(J))=0.
      IAA=IA(I)
      IAB=IA(IH)-1
      IF(IAB.LT.IAA)GO TO 40
      DO 30 J=IAA,IAB
30    DI(JA(J))=AN(J)
40    DI(I)=AD(I)
      LAST=IP(I)
      IF(LAST.EQ.0)GO TO 90
      LN=IP(LAST)
50    L=LN
      LN=IP(L)
      IUC=IUP(L)
      IUD=IU(L+1)-1
      UM=UN(IUC)*DI(L)
      DO 60 J=IUC,IUD
      JJ=JU(J)
60    DI(JJ)=DI(JJ)-UN(J)*UM
      UN(IUC)=UM
      IUP(L)=IUC+1
      IF(IUC.EQ.IUD)GO TO 80
      J=JU(IUC+1)
      JJ=IP(J)
      IF(JJ.EQ.0)GO TO 70
      IP(L)=IP(JJ)
      IP(JJ)=L
      GO TO 80
70    IP(J)=L
      IP(L)=L
80    IF(L.NE.LAST)GO TO 50
```

```
39.  90    DI(I)=1./DI(I)
40.        IF(IUB.LT.IUA)GO TO 120
41.        DO 100 J=IUA,IUB
42. 100    UN(J)=DI(JU(J))
43.        J=JU(IUA)
44.        JJ=IP(J)
45.        IF(JJ.EQ.0)GO TO 110
46.        IP(I)=IP(JJ)
47.        IP(JJ)=I
48.        GO TO 120
49. 110    IP(J)=I
50.        IP(I)=I
51. 120    IUP(I)=IUA
52. 130    CONTINUE
```

En este algoritmo se emplea IP para anotar las listas encadenadas de filas asociadas con cada columna, según se explicó en el ejemplo 15. IP es inicializado a 0 en las líneas 1 y 2, pues todas las listas de filas están incialmente vacías. El ciclo DO 120 recorre las filas de las matrices A y U, las que son identificadas con el índice I. Si la fila I de U no tiene no-ceros fuera de la diagonal, el programa salta de la línea 7 a la 15, donde el elemento diagonal, que es siempre positivo, es procesado. De lo contrario, se inicializan las posiciones del vector expandido DI que corresponden a no-ceros en la línea 9, y en la línea 14 se cargan en DI los no-ceros de la fila I de A, si es que hay alguno. El elemento diagonal de la fila I es cargado en la línea 15.

El puntero para la lista circular donde se anotan las filas asociadas con la columna I se obtiene de IP(I) en la línea 16 y se almacena en LAST. Si la lista está vacía, LAST será igual a cero, y el algoritmo saltará de la línea 17 a la 39. De lo contrario, LAST es el índice de fila de una de las filas que tienen un no-cero en la columna I, la "última" en la

cadena, y L en la línea 19, es el índice de la "siguiente" fila, la "primera" en la lista. En la línea 21, IUC=IUP(L) señala precisamente el no-cero de la fila L sobre la columna I, mientras que IUD, en la línea 22, señala el final de la fila L. El factor por el cual los no-ceros de la fila L deben ser multiplicados antes de acumularlos, es calculado en la línea 23 y almacenado en UM. Finalmente, el ciclo DO 60 acumula los elementos de la fila L multiplicados por UM.

Al comienzo de esta sección hemos mencionado que los elementos de cada fila deben ser normalizados dividiéndolos por el elemento diagonal, para formar el factor triangular superior U. Esta operación se realiza en la línea 27, dónde UM fue determinado en la línea 23 y DI(L) es en realidad la inversa del elemento diagonal (ver la línea 39 y notar que L<I). Es importante señalar que cada fila I es generada y almacenada en una forma no normalizada. Cada no-cero de cada fila es normalizado más tarde, en el momento en que se lo usa como un multiplicador para la fila L, en las líneas 23 y 27. Procediendo de esta manera se evita una cantidad de multiplicaciones (9, pág.1805). Como las líneas 23 y 27 son ejecutadas una sola vez por cada no-cero, está garantizado que todo elemento será convenientemente normalizado.

El puntero auxiliar IUP, que señala el no-cero de la fila L sobre la columna I, es incrementado en la línea 28 del algoritmo para que señale hacia el próximo no-cero de la fila L, si es que hay alguno. Si no hubiera más no-ceros en la fila L,

el programa se desvía desde la línea 29 hacia la 38, donde se reinicia el recorrido de la cadena. De otra forma, en las líneas 30 a 37, se inscribe la fila L en la cadena asociada con la columna J, que es el índice de columna del siguiente no-cero de la fila L. Notar aquí la diferencia con el algoritmo simbólico del ejemplo 16, donde una fila L era usada una sola vez y después descartada.

La ejecución alcanza la línea 39 cuando todas las filas asociadas con la columna I han sido procesadas. Si la nueva fila I recién generada no está vacía, los no-ceros, todavía sin normalizar, son extraídos del vector expandido DI y almacenados en UN por el ciclo DO 100. A continuación la nueva fila I es inscripta en la cadena asociada con el índice de columna J de su primer no-cero. Esta operación es realizada en las líneas 43 a 50. Finalmente, antes de proceder a la fila siguiente, el puntero auxiliar IUP(I) es definido en la línea 51 para que señale el primer no-cero de la nueva fila I.

11. Resolución directa de sistemas de ecuaciones lineales con matriz simétrica definida positiva.

Consideremos el sistema de ecuaciones lineales:

$$A\,x = b \qquad (1)$$

Existen básicamente dos grupos de métodos para resolver (1): los métodos iterativos y los métodos directos. En esta sección

nos vamos a limitar a considerar un método directo: la factorización triangular de A seguida de sustitución hacia adelante y hacia atrás. Este método es computacionalmente equivalente a la eliminación de Gauss si el sistema (1) debe ser resuelto una sola vez, pero tiene ciertas ventajas cuando (1) tiene que ser resuelto varias veces con diferentes segundos miembros b. En esta sección también nos limitaremos al caso en que A es simétrica y definida positiva.

En el método de factorización triangular seguida de sustitución hacia adelante y luego hacia atrás, la matriz A es primero factorizada usando los algoritmos de la sección 10:

$$A = U_T\ D\ U \tag{2}$$

donde U es triangular superior con diagonal unitaria, y D es diagonal. El sistema (1) queda entonces:

$$U^T\ DUx = b \tag{3}$$

Llamando $z = DUx$ y $y = Ux$, podemos escribir:

$$\begin{aligned} U^T z &= b \\ Dy &= z \\ Ux &= y \end{aligned} \tag{4}$$

El sistema (4) se resuelve primero para z, después para y, y finalmente para x. Como tanto U^T y U son triangulares y D es diagonal, la resolución de (4) es inmediata. Los algoritmos para la factorización triangular ya fueron discutidos en la sección 10, y un ejemplo de sustitución y el correspondiente algoritmo se dan en los ejemplos 18 y 19, a continuación.

Ejemplo 18. Solución del sistema (4) en un caso simple. Supongamos que la matriz simétrica A de la ecuación (4) ha sido factorizada como sigue:

$$A = \begin{vmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ e & f & 1 \end{vmatrix} \begin{vmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{vmatrix} \begin{vmatrix} 1 & d & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{vmatrix} \qquad (5)$$

Entonces la primera ecuación del sistema (4) es:

$$\begin{vmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ e & f & 1 \end{vmatrix} \begin{vmatrix} z_1 \\ z_2 \\ z_3 \end{vmatrix} = \begin{vmatrix} b_1 \\ b_2 \\ b_3 \end{vmatrix} \qquad (6)$$

de donde obtenemos inmediatamente:

$$\begin{aligned} z_1 &= b_1 \\ z_2 &= b_2 - dz_1 \\ z_3 &= b_3 - ez_1 - fz_2 \end{aligned} \qquad (7)$$

Este paso es la <u>sustitución hacia adelante</u>. La segunda ecuación del sistema (4) es:

$$\begin{vmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{vmatrix} \begin{vmatrix} y_1 \\ y_2 \\ y_3 \end{vmatrix} = \begin{vmatrix} z_1 \\ z_2 \\ z_3 \end{vmatrix} \qquad (8)$$

Por lo tanto:

$$\begin{aligned} y_1 &= z_1/a \\ y_2 &= z_2/b \\ y_3 &= z_3/c \end{aligned} \qquad (9)$$

Finalmente, la última ecuación del sistema (4) es:

$$\begin{vmatrix} 1 & d & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \end{vmatrix} \qquad (10)$$

La solución final se obtiene ahora fácilmente:

$$\begin{aligned} x_3 &= y_3 \\ x_2 &= y_2 - fx_3 \\ x_1 &= y_1 - dx_2 - ex_3 \end{aligned} \qquad (11)$$

Este último paso, que se realiza hacia atrás, se llama sustitución hacia atrás.

Ejemplo 19. Algoritmo para la resolución del sistema $U^TDUx = b$.

Datos: IU,JU,UN matriz triangular superior con diagonal unitaria U dada en R(S)FD.

DI inversas de los elementos de la matriz diagonal D.

B vector de segundos miembros, lleno.

N orden del sistema.

Resultados: X vector de incógnitas x.

```
      NM = N-1
      DO 10 I = 1,N
10    X(I) = B(I)
      DO 40 K=1,NM
      IUA = IU(K)
      IUB = IU(K+1)-1
      XX = X(K)
      IF(IUB.LT.IUA)GO TO 30
      DO 20 I = IUA,IUB
20    X(JU(I)) = X(JU(I)) - UN(I)*XX
30    X(K) = XX*DI(K)
40    CONTINUE
      X(N) = X(N)*DI(N)
      K = NM
50    IUA = IU(K)
      IUB = IU(K+1)-1
```

```
17.          IF(IUB.LT.IUA)GO TO 70
18.          XX = X(K)
19.          DO 60 I = IUA,IUB
20.    60    XX = XX - UN(I)*X(JU(I))
21.          X(K) = XX
22.    70    K = K-1
23.          IF(K.GT.0)GO TO 50
```

La forma en que opera este algoritmo se comprende facilmente si se lo compara con el ejemplo 18. El algoritmo comienza construyendo el vector z de la ecuación (7) en X. Este vector es inicializado en las líneas 2 y 3. Después, el ciclo DO 40 recorre las filas de U y realiza por columnas las operaciones indicadas en la ecuación (7). Inmediatamente después de que cada fila de U ha sido procesada, X(K) es multiplicado por la inversa del elemento diagonal K de la matriz D, almacenada en DI(K), en la línea 11. Esta operación es requerida por la ecuación (9). Después que la línea 13 ha sido ejecutada, el vector y es construido en X.

La resolución de la ecuación (10) comienza en la línea 14. El vector X ya está inicializado con los elementos del vector y. El ciclo DO 60 realiza las operaciones indicadas por la ecuación (11), esta vez fila por fila, aprovechando que U está representado por filas. La solución final x es construida en X.

12. Producto de la inversa de una matriz rala triangular inferior por una matriz rala general.

Algunas aplicaciones (28) requieren el cálculo de productos de la forma:

$$X = (U^T)^{-1}B \qquad (1)$$

donde B es una matriz rala general, U es una matriz rala triangular superior con diagonal unitaria, y U^T es su traspuesta, que es por lo tanto rala triangular inferior con diagonal unitaria. Es posible obtener X sin calcular jamás la inversa de U^T, que sería una matriz densa. Con este fin, escribimos (1) de la siguiente manera:

$$U^T X = B \qquad (2)$$

que es ahora un sistema de ecuaciones lineales con los elementos de X como incógnitas. El sistema (2) se resuelve usando las técnicas usuales, muy similares a las que se explicaron en las secciones 10 y 11. Se usan dos algoritmos: uno simbólico y el otro numérico. Los algoritmos emplean la técnica de llave múltiple y la de reunión expandida de no-ceros descritas en la sección 7. Un caso simple se ilustra en el ejemplo 20, y los algoritmos se dan en los ejemplos 21 y 22.

Ejemplo 20. Producto de la inversa de una matriz triangular inferior por una matriz general.

Consideremos el siguiente sistema, del tipo de la ecuación (2):

$$\begin{vmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{vmatrix} \begin{vmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{vmatrix} = \begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{vmatrix} \qquad (3)$$

Para la primera fila tenemos:

$$(x_{11}\ x_{12}) = (b_{11}\ b_{12}) \qquad (3)$$

donde la notación significa:

$$x_{11} = b_{11}$$

$$x_{12} = b_{12}$$

Para la segunda fila:

$$a(x_{11}\ x_{12}) + (x_{21}\ x_{22}) = (b_{21}\ b_{22})$$

Por lo tanto:

$$(x_{21}\ x_{22}) = (b_{21}\ b_{22}) - a(x_{11}\ x_{12}) \qquad (5)$$

De una manera similar, para la tercera fila obtenemos:

$$(x_{31}\ x_{32}) = (b_{31}\ b_{32}) - b(x_{11}\ x_{12}) - c(x_{21}\ x_{22}) \qquad (6)$$

Las ecuaciones (4),(5) y (6) se pueden resolver una después de la otra, obteniéndose la matriz X buscada. Este algoritmo es apropiado para matrices ralas en formato por filas porque tanto U^T como B son accedidas sólo por filas.

Ejemplo 21. Algoritmo para la multiplicación simbólica de la inversa de una matriz triangular inferior $(U^T)^{-1}$ por una matriz rala general B.

Datos: IUT,JUT estructura de U^T en R(I)FD,donde I indica que el triángulo inferior está representado.

IB,JB estructura de la matriz B en R(C)FD.

N orden de U^T y cantidad de filas de B y de X.

NC cantidad de columnas de B y de X.

Resultados: IX,JX estructura de la matriz resultado en R(C)FD.

Vector auxiliar: IP de dimensión NC, llave múltiple.

```
 1.          DO 10 I=1,NC
 2.   10     IP(I)=0
 3.          JP=1
 4.          IX(1)=1
 5.          DO 70 I=1,N
 6.          IBA = IB(I)
 7.          IBB=IB(I+1)-1
 8.          IF(IBB.LT.IBA)GO TO 30
 9.          DO 20 JJ=IBA,IBB
10.          J=JB(JJ)
11.          IP(J)=I
12.          JX(JP)=J
13.   20     JP=JP+1
14.   30     IUA=IUT(I)
15.          IUB=IUT(I+1)-1
16.          IF(IUB.LT.IUA)GO TO 60
17.          DO 50 JJ=IUA,IUB
18.          J=JUT(JJ)
19.          IXA=IX(J)
20.          IXB=IX(J+1)-1
21.          IF(IXB.LT.IXA)GO TO 50
22.          DO 40 KP=IXA,IXB
23.          K=JX(KP)
24.          IF(IP(K).EQ.I)GO TO 40
25.          IP(K)=I
26.          JX(JP)=K
27.          JP=JP+1
28.   40     CONTINUE
29.   50     CONTINUE
30.   60     IX(I+1)=JP
31.   70     CONTINUE
```

En este algoritmo, el vector de llave múltiple IP es inicializado a 0 en las líneas 1 y 2. El puntero JP para IX se inicializa en la línea 3, se aumenta en las líneas 13 y 27 cada vez que un nuevo elemento se inscribe en la lista JX, y se usa para construir IX en la línea 30. El ciclo DO 70 recorre las N filas de las matrices. La fila I de B es cargada en JX por el ciclo DO 20, en la línea 12, e I se almacena en la posición J de la llave múltiple IP para impedir cualquier intento futuro de volver a inscribir el índice de columna J en JX.

A continuación comienza el algoritmo correspondiente a las ecuaciones (4),(5) y (6) del ejemplo 20. El ciclo DO 50 recorre la fila I de U^T, y para cada índice de columna J que se encuentra el ciclo DO 40 recorre la fila J de X e inscribe en JX cualquier índice de columna K que no haya sido inscripto antes. Notar que la posición K de la llave múltiple es comprobada en la línea 24, y bloqueada en la línea 25 cuando ello es necesario.

Ejemplo 22. Algoritmo para la multiplicación numérica de la inversa de una matriz triangular inferior $(U^T)^{-1}$ por una matriz rala general B.

Datos: IUT,JUT,UNT matriz triangular inferior dada U^T en R(I)FD.

IB,JB,BN matriz B dada en R(C)FD.

IX,JX estructura de la matriz resultado X en R(C)FD.

N orden de U^T y cantidad de filas de B y de X.

Resultado: XN valores numéricos para la matriz X, en R(C)FD.

Vector auxiliar: X de dimensión igual a la cantidad de columnas de B para reunión expandida de no-ceros.

```
 1.          DO 80 I=1,N
 2.          IH=I+1
 3.          IXA=IX(I)
 4.          IXB=IX(IH)-1
 5.          IF(IXB.LT.IXA)GO TO 80
 6.          DO 10 IP=IXA,IXB
 7.    10    X(JX(IP))=0.
 8.          IBA=IB(I)
 9.          IBB=IB(IH)-1
 0.          IF(IBB.LT.IBA)GO TO 30
11.          DO 20 IP=IBA,IBB
```

```
12.   20    X(JB(IP))=BN(IP)
13.   30    IUA=IUT(I)
14.         IUB=IUT(IH)-1
15.         IF(IUB.LT.IUA)GO TO 60
16.         DO 50 JP=IUA,IUB
17.         J=JUT(JP)
18.         IXC=IX(J)
19.         IXD=IX(J+1)-1
20.         IF(IXD.LT.IXC)GO TO 50
21.         A=UNT(JP)
22.         DO 40 KP=IXC,IXD
23.         K=JX(KP)
24.   40    X(K)=X(K)-A*XN(KP)
25.   50    CONTINUE
26.   60    CONTINUE
27.         DO 70 IP=IXA,IXB
28.   70    XN(IP)=X(JX(IP))
29.   80    CONTINUE
```

Dada una matriz triangular inferior con diagonal unitaria U^T de NxN y una matriz general B con N filas, este algoritmo determina en el vector XN los valores numéricos de los no-ceros de la matriz producto $X = (U^T)^{-1}B$. El ciclo DO 80 recorre las filas de las matrices. Para cada fila I, son inicializadas a cero en las líneas 6 y 7 aquellas posiciones del vector auxiliar X que corresponden a no-ceros de la matriz resultado en la fila I, si es que hay alguno. Los no-ceros de la fila I de B, si hay alguno, son obtenidos de su almacenamiento compacto en BN y almacenados en formato expandido en el vector X por el ciclo DO 20. El ciclo DO 50 recorre los no-ceros de la fila I de U^T y realiza las operaciones ilustradas por las ecuacioes (4),(5) y (6) del ejemplo 20. Para cada no-cero de la fila I de U^T con índice de columna J y valor A, el ciclo DO 40 recorre la fila J de la matriz X (ya completa a esta altura pues $J<I$), multiplica los no-ceros que haya por A, y acumula los resultados con los signos cambiados en las

posiciones correspondientes del vector X. Finalmente, el ciclo DO 70 saca los valores resultantes de X y los almacena en forma compacta en el vector XN, en correspondencia con los índices de columna JX de la fila I.

13. Algoritmos varios

Una colección de algoritmos útiles para matrices ralas es presentada y discutida en esta sección.

Ejemplo 23. Producto de la inversa de una matriz triangular superior de diagonal unitaria U por un vector lleno X.

Datos: IU,JU,UN matriz U en R(S)FD.

X vector lleno dado.

N orden de la matriz U.

Resultado: W vector resultado $W=U^{-1}X$.

```
      DO 10 I=1,N
10    W(I)=X(I)
      I=N-1
20    IUA=IU(I)
      IUB=IU(I+1)-1
      IF(IUB.LT.IUA)GO TO 40
      Z=W(I)
      DO 30 IP=IUA,IUB
30    Z=Z-UN(IP)*W(JU(IP))
      W(I)=Z
40    I=I-1
      IF(I.GT.0)GO TO 20
```

Este algoritmo es similar y cumple la misma función que una porción del algoritmo discutido en el ejemplo 19: la comprendida en las líneas 2,3 y 14 hasta 23.

Ejemplo 24. Producto de la inversa traspuesta de una matriz triangular superior de diagonal unitaria U por un vector lleno.

Datos: IU,JU,UN Matriz U dada en R(S)FD.

X vector lleno dado.

N orden de la matriz U.

Resultados: W vector lleno resultante $W=(U^T)^{-1}X$.

```
      NM=N-1
      DO 10 I=1,N
10    W(I)=X(I)
      DO 30 I=1,NM
      IUA=IU(I)
      IUB=IU(I+1)-1
      IF(IUB.LT.IUA)GO TO 30
      Z=W(I)
      DO 20 IP=IUA,IUB
      J=JU(IP)
20    W(J)=W(J)-Z*UN(IP)
30    CONTINUE
```

Este algoritmo es equivalente a la primera parte del algoritmo descrito en el ejemplo 19, y cumple la misma función.

Ejemplo 25. Copia de una matriz rala de IA,JA,AN a IB,JB,BN.

Datos: IA,JA,AN matriz dada en cualquier representación por filas.

N cantidad de filas de la matriz dada.

Resultados: IB,JB,BN matriz dada copiada a los vectores indicados.

```
      NH=N+1
      DO 10 I=1,NH
10    IB(I)=IA(I)
      IAA=IA(1)
      IAB=IA(NH)-1
      DO 20 I=IAA,IAB
      JB(I)=JA(I)
20    BN(I)=AN(I)
```

Este sencillo algoritmo puede ser facilmente modificado para cumplir otras funciones: cambiar de signo una matriz dada, multiplicar una matriz por una constante, etc.

Ejemplo 26. Premultiplicación de una matriz rala A por una matriz diagonal D.

Datos: IA,AN matriz A dada en R(C)FD. Notar que JA no es necesario.

D diagonal llena de la matriz diagonal dada.

N orden de D y cantidad de filas de A.

Resultados: IA,AN matriz resultado B=DA; notar que el resultado es devuelto en los mismos vectores IA, AN en que se dan los datos. El mismo JA es válido para B.

```
      DO 20 I=1,N
      IAA=IA(I)
      IAB=IA(I+1)-1
      IF(IAB.LT.IAA)GO TO 20
      DD=D(I)
      DO 10 J=IAA,IAB
10    AN(J)=AN(J)*DD
20    CONTINUE
```

En este algoritmo, el ciclo DO 20 recorre las N filas de la matriz A. Para cada fila I, el ciclo DO 10 recorre todos los no-ceros de la fila y los multiplica por el elemento diagonal D(I), previamente almacenado en DD.

14. Consideraciones finales

Algunas consideraciones pueden ser convenientes para la persona que se propone usar los algoritmos descritos en esta publicación. Los algoritmos son casi óptimos, en el sentido de que hacen una cantidad fija de operaciones y usan una cantidad fija de memoria de computadora por cada elemento de información útil presente en el problema, o sea, por cada no-cero. Los algoritmos también representan el estado del arte de la Tecnología de Matrices Ralas, en el sentido de que la cantidad de operaciones y de almacenamiento por no-cero son tan pequeñas como es posible lograrlo hoy en día, pero que puede haber desarrollos futuros que conduzcan a mejoras en algunos casos. Finalmente, los algoritmos han sido cuidadosamente probados, ya que, excepto algunos detalles, todos ellos están siendo usados en programas profesionales (29,30).

La experiencia enseña que la causa más frecuente de falla de los algoritmos de matrices ralas es la invasión de memoria. Cuando se construye una matriz en formato ralo en IA,JA,AN, en general no se sabe que largo final tendrán JA y AN. Una práctica recomendable es comparar el puntero a JA y AN con algún parámetro que indique el espacio de memoria reservado para ambos vectores, cada vez que un nuevo no-cero es inscripto. Otra fuente de errores frecuentes es el largo de IA, que debe ser N+1 para una matriz con N filas, y el elemento IA(N+1) debe ser definido inevitablemente de acuerdo con nuestras convenciones.

La invasión de memoria puede producir resultados impredecibles, los que en apariencia no tienen ninguna relación con la causa original del error. Por lo tanto es muy conveniente detectar los posibles errores en una etapa temprana. Como ejemplo, consideremos la siguiente secuencia de instrucciones, que ha sido usada en todos los algoritmos:

```
IAA=IA(I)
IAB=IA(I+1)-1
IF(IAB.LT.IAA)GO TO α
```

donde la condición IAB.LT.IAA se usa para detectar filas vacías en la matriz. Pero para una fila vacía debe valer IAB=IAA-1. Esta condición puede ser comprobada a bajo costo cada vez que se encuentra una fila vacía, pues las filas vacías son generalmente pocas, y puede servir muy bien el propósito de una detección temprana de errores. En particular, el olvido de definir IA(N+1) puede muchas veces ser detectado de esta manera. Por supuesto, hay muchas otras maneras de detectar errores en la etapa más temprana posible. Por ejemplo, se puede usar un algoritmo que examine una representación rala enseguida después de construirla y antes de usarla con algún otro propósito. El algoritmo podría comprobar si IA(I)≤IA(I+1) para todos los I, que JA no contiene índices de columna demasiado grandes ni demasiado chicos, que AN no contiene ceros, etc.

El mismo formato ralo se puede convertir en otra fuente de dificultades para el usuario. El usuario puede tener sus

matrices almacenadas como matrices llenas o de banda, y desear procesarlas como matrices ralas para ahorrar tiempo de máquina o aprovechar mejor la memoria disponible. Esta situación no debería ser la usual, porque conviene almacenar la información directamente en formato ralo tan pronto como es producida, para sacar provecho de los algoritmos ralos desde el mismo comienzo. Hay casos, sin embargo, en que un algoritmo que transforma una matriz llena o de banda a formato ralo puede ser muy conveniente. Uno de esos casos es el ensamblaje de matrices de elemento llenas, discutido en la sección 9. Otro caso se presenta cuando el usuario quiere comparar algoritmos para matrices llenas o de banda con algoritmos para matrices ralas, en lo referente a requerimientos de memoria, exactitud de los resultados, tiempo de ejecución, etc., resolviendo el mismo problema con diferentes algoritmos. Inversamente, también hay casos en que una matriz rala o una porción de la misma debe ser almacenada como matriz llena. Por ejemplo, el usuario puede querer revisar una porción de su matriz rala. Para ese fin, un listado de IA,JA AN resulta muy confuso, y es mejor diseñar un algoritmo que sea capaz de imprimir la matriz, o una porción de la misma definida por una "ventana", en formato lleno, o sea tabulada, rellenando con ceros donde sea necesario. Comentarios adicionales sobre esta cuestión y una solución interesante al problema se pueden ver en (31).

Hay algunos aspectos importantes de las matrices ralas que casi ni han sido mencionados en esta publicación para

evitar hacerla demasiado extensa. Entre tales aspectos están los siguientes: ordenamiento para reducir el llenado durante la eliminación de Gauss (23-27); procedimientos iterativos para resolver sistemas de ecuaciones lineales que solamente requieren el producto de la matriz de coeficientes por un vector lleno en cada iteración, y son por lo tanto muy convenientes cuando la matriz es llena (32,33); cálculo de autovalores de una matriz rala, usando ya sea procedimientos directos o iterativos que también requieren el producto de la matriz por un vector en cada iteración (34-36). El lector interesado debe consultar la literatura especializada en estos casos. Una confusión muy común es asociar las matrices ralas con la resolución de sistemas de ecuaciones lineales por eliminación de Gauss directa. Practicamente cualquier algoritmo que usa matrices con cualquier fin puede ser programado para usar matrices ralas. Las matrices ralas son por lo tanto una herramienta de trabajo general que se usará cada día con mayor intensidad.

Referencias

1. S.Pissanetzky. "Implementación de un conjunto de algoritmos que emplean la Tecnología de las Matrices Dispersas para resolver problemas de análisis de Elementos Finitos". XXVII Jornadas Nacionales de ASOVAC, 6 al 12 de Nov. de 1977. Valencia, Venezuela.

2. S.Pissanetzky. "Código de computación para la resolución directa de sistemas de ecuaciones algebraicas lineales con matriz esparcida, simétrica y definida positiva: código SPFACT". Nota Técnica CNEA-NT 15/78. Comisión Nacional de Energía Atómica, Arg. (1978).

3. F. Duchin y D.B.Szyld. "Aplication of Sparse Matrix Techniques to Inter-Regional Input-Output Analysis". Seventh International Conference on Input-Output Techniques, Innsbruck, Austria, 9-13 Abril 1979.

4. A.E.McDonald y R.H.Trimble. "A Sparse Matrix Factorization-Storage Management Algorithm for Direct Solution of Reservoir Simulation Equations". Fourth Symposium of Numerical Simulation of Reservoir Performance. Los Angeles, California, Feb.19-20 (1976).Paper nr.SPE 5731.

5. P.T.Woo, S.J.Roberts y F.G.Gustavson. "Application of Sparse Matrix Techniques in Reservoir Simulation". Soc.of Petroleum Engineers 48th.Annual Fall Meeting, Las Vegas, Paper nr.SPE 4544 (1973).

6. S.Pissanetzky. "Modelo de reservorio bidimensional formulado en base al análisis de elementos finitos". Primeras Jornadas Científico-Técnicas. Fac.de Ingeniería, Univ.del Zulia, Venezuela (1977).

7. G.D.Hachtel, R.K.Brayton y F.G.Gustavson. "The Sparse Tableau Approach to Network Analysis and Design". IEEE Trans. on Circuit Theory, Vol.CT-18, 101-113 (1971).

8. F.G.Gustavson y G.D.Hachtel. "Program Implementation of a Network Design Program based on the Sparse Tableau Adjoint Method". Proc.Sixteenth Midwest Symposium on Circuit Theory, Vol.1, Univ.of Waterloo, Ontario, Canadá, Abril 12-13 (1973).

9. W.F.Tinney. "Direct Solution of Sparse Network Equations by Optimally Ordered Triangular Factorization". Proc.of the IEEE 55, 1801-1809 (1967).

10. A.Chang. "Application of Sparse Matrix Mehods in Electric Power System Analysis". Sparse Matrix Proceedings, Willoughby ed.P.113-122.

11. E.Hellerman y D.C.Rarick. "The Partitioned Preassigned Pivot Procedure (P4)", en Sparse Matrices and their Applications, editado por D.J.Rose y R.A.Willoughby, Plenum Press (1972).P.67-76.

12. J.A.Tomlin. "Modifying Triangular Factors of the Basis in the Simplex Method", en Sparse Matrices and their Applications, editado por D.J.Rose y R.A.Willoughby, Plenum Press (1972).P.77-85.

13. G.H.Glaser y M.S.Saliba. "Application of Sparse Matrices to Analytical Photogrammetry", en Sparse Matrices and their Applications, editado por D.J.Rose y R.A.Willoughby, Plenum Press (1972) p.135-146.

14. F.G.Gustavson. "Some Basic Techniques for solving Sparse Systems of Linear Equations", en Sparse Matrices and their Applications, editado por D.J.Rose y R.A.Willoughby, Plenum Press, N.Y.(1972) p.41-52.

15. S.C.Eisenstat, M.C.Gursky, M.H.Schultz y A.H.Sherman. "Yale Sparse Matrix Package. I.The Symmetric Codes". Research Report 112, Yale University, Department of Computer Science.

16. W.M.Gentleman y A.George. "Sparse Matrix Software", en Sparse Matrix Computations", editado por J.R.Bunch y D.J. Rose, Academic Press (1976) p.243-261.

17. S.C.Eisenstat, M.H.Schultz y A.H.Sherman. "Considerations in the Design of Software for Sparse Gaussian Elimination", en Sparse Matrix Computations, editado por J.R. Bunch y D.J.Rose, Academic Press (1976) p.263-273.

18. F.G.Gustavson. "Permuting Matrices Stored in Sparse Format".IBM Research Report RC 6181 (26575) (1976).

19. F.G.Gustavson. "An efficient algorithm to perform sparse matrix multiplication". IBM Research Report RC 6176 (26569) (1976).

20. O.C.Zienkiewicz. "The Finite Element Method in Engineering Science" McGraw-Hill, London (1971).

21. C.A.Brebbia. "The Boundary Element Method for Engineers". J.Wiley & Sons, New York (1978).

22. S.Pissanetzky. "KUBIK: An Automatic Three-Dimensional Finite Element Mesh Generator". Int.J.for Num.Meth.in Engn. 17, 255-269 (1981).

23. A.George. "Nested Dissection of a Regular Finite Element Mesh". SIAM J.Numer.Anal.10, 345-363 (1973).

24. D.J.Rose, R.E.Tarjan y G.S.Lueker. "Algorithmic Aspects of Vertex Elimination on Graphs". SIAM J.on Computing 5, 266-283 (1976).

25. A.George, J.W.Liu. "An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems". Research Report CS-76-38. Dep.of Computer Science, University of Waterloo, Ontario, Canadá (1976).

26. A.George. "Numerical Experiments using Dissection Methods to solve n by n Grid Problems". Research Report CS-75-07, University of Waterloo, Waterloo, Ontario, Canadá (1976).

27. A.George y J.W.H.Liu. "A Minimal Storage Implementation of the Minimum Degree Algorithm". SIAM J.Numer.Anal.17, 282-299 (1980).

28. S.Pissanetzky. "Gauss Elimination with Supersparse Matrices". Informal Report BNL 26773, Brookhaven National Laboratory, New York (1979).

29. S.Pissanetzky. "Numerical Modelling of Electromagnets by the Finite Element Method". Informal Report BNL 26516, Brookhaven National Laboratory, New York (1979).

30. S.Pissanetzky. "MAGNUS: Computer-aided Design of Electromagnets. Implementation of the Two-Scalar-Potentials method at the Central Scientific Computing Facility of the Brookhaven National Laboratory". Informal Report BNL 28416, AMD 867, Brookhaven National Laboratory, New York (1980).

31. A.George y J.W.H.Liu. "The Design of a User Interface for a Sparse Matrix Package". ACM Transactions of Mathematical Software 5, 139-162 (1979).

32. O.Axelsson. "Solution of Linear Systems of Equations: Iterative Methods", en "Sparse Matrix Techniques", Lecture Notes in Mathematics nr.572, editado por V.A.Barker. Springer Verlag. Berlín (1977) p.1-51.

33. C.C.Paige y M.A.Saunders. "Solution of Sparse Indefinite Systems of Linear Equations". SIAM J.Numer.Anal.12, 617-629 (1975).

34. J.H.Wilkinson. "The Algebraic Eigenvalue Problem". Oxford University Press (1965).

35. G.W.Stewart. "A Bibliographical Tour of the Large, Sparse Generalized Eigenvalue Problem", en Sparse Matrix Computations, editado por J.R.Bunch y D.J.Rose, Academic Press (1976) p.113-130.

36. A.Ruhe. "Computation of Eigenvalues and Eigenvectors", en "Sparse Matrix Techniques", Lecture Notes in Mathematics, nr.572, editado por V.A.Barker. Springer Verlag, Berlín (1977) p.130-184.