Mecánica Computacional Vol. XVI
compilado por G. Etse y B. Luccioni
Tucumán, Septiembre de 1996

MANIPULACION DE MATRICES RALAS

Paulo Porta
Centro de Informática, U.T.N., Facultad Regional Haedo
Paris 532 - 1706 - Haedo - Pcia. de Buenos Aires - Argentina
paulo@utnrha.edu.ar

RESUMEN

En este trabajo, se presenta la implementación de un paquete de rutinas para la manipulación de grandes matrices ralas. Se dan los algoritmos necesarios para representar las matrices como arreglos de tres vectores y para realizar las operaciones más usuales del álgebra lineal. Se ofrecen además índices comparativos de performance.

ABSTRACT

In this paper, we present the implementation of a library containing the routines for manipulating large sparse matrices. The algorithms for representing the matrices as an arrange of three vectors, and for performing the most usual operations of linear algebra are given. We also present a comparative performance of these.

INTRODUCCION

Los operadores lineales son los más simples de todos los operadores matemáticos y ofrecen además buenos modelos para representar muchos problemas y aplicaciones; es por esto que frecuentemente nos encontramos con problemas de álgebra lineal al trabajar en la Mecánica Computacional. Estos problemas pueden ser separados en tres categorías:

1. Álgebra elemental de matrices.
2. Resolución de sistemas lineales de ecuaciones algebraicas.
3. Problema de valores propios.

Existen en los textos infinidad de métodos y algoritmos para enfrentar este tipo de problemas, basados en la representación usual de las matrices como arreglos cuadrados o rectangulares de números. Muchos esquemas de integración numérica del problema de valor de contorno, dan lugar a los problemas de álgebra lineal antes enunciados, pero con la particularidad que las matrices que intervienen en ellos son grandes y ralas, es decir matrices de muchos elementos y con muchos de ellos nulos; un ejemplo arquetípico de esto son las matrices bandadas del Método de Elementos Finitos: en efecto, las matrices del sistema ensamblado tienen elementos no nulos sólo donde hay aporte de los nodos, dando lugar a grandes matrices ralas, con la estructura de banda.
Motivados por este tipo de problemas y por la complicación que requiere mantener en memoria grandes sistemas, es que presentamos una serie de algoritmos para resolver los problemas del álgebra

lineal antes enunciados, pero optimizando el tamaño de la memoria requerido, aprovechando la forma especial de estas matrices ralas.

GRANDES MATRICES RALAS

Consideremos una matriz A, grande y rala, es decir con una proporción de elementos nulos apreciable, de n filas y m columnas. La representación usual de una matriz como esta, requiere $n \times m$ variables, reales o complejas según sea A, ya sea en simple o doble precisión, para almacenar todos sus elementos. Sin embargo, si pensamos que muchos de los elementos de A son ceros, se puede idear una forma de almacenar la matriz que aproveche esta particularidad. En efecto, siguiendo a Gustavson [1], representaremos la matriz A con tres vectores AE, AJ y AI: el primero es el vector que contiene todos los elementos no nulos de A, guardados por filas; el segundo es un vector tal que el elemento AJ_k guarda el número de la columna de A que ocupa el k-ésimo elemento de AE. El vector AI es un vector de $n+1$ elementos, tal que su elemento AI_k da la posición que ocupa en AE el primer elemento de la k-ésima fila de A, siendo el AI_{m+1} igual a la cantidad de elementos no nulos de A, más uno. Este esquema suele denominarse CRS – *Compress Row Storage* (Cfr. [2]).
En ánimo de fijar ideas, propongamos el siguiente ejemplo: sea A la matriz dada por:

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

En la representación compacta de Gustavson, A queda representada por:

AE=(7, 3, 1, 2, 6, 1, 2, 1, 1)
AJ=(1, 4, 2, 3, 3, 1, 4, 4, 5)
AI=(1, 3, 5, 6, 8, 10)

A partir de esta representación de las matrices en la forma de tres vectores, es que construiremos los algoritmos para realizar las operaciones necesarias para resolver los problemas del álgebra lineal.

LOS ALGORITMOS

Presentaremos a continuación los algoritmos, dividiéndolos en la misma forma que dividimos los problemas del álgebra lineal. Los algoritmos se presentan en pseudocódigo, de manera que pueden ser implementados en cualquier lenguaje.

Álgebra elemental de matrices: En esta categoría de problemas, incluímos: producto de un escalar por una matriz, suma algebráica de matrices, multiplicación de una matriz por un vector, multiplicación de una matriz transpuesta por un vector y multiplicación de matrices. Estas operaciones, junto con la inversión de la matriz permiten construir todas las Subrutinas Básicas de Álgebra Lineal (BLAS) de nivel 2 y 3, según se especifican en [3] y [4], salvo las que involucran la solución de un sistema lineal; para este fin se presentan en el siguiente apartado algoritmos especiales para la estructura de datos bajo la que almacenamos la matriz. Las BLAS de nivel 2 y 3, junto a las BLAS de nivel 1 ([5] y [6]) conforman la especificación básica de cualquier bilioteca de rutinas para problemas de álgebra lineal.

- **Producto de una matriz por un escalar:** Dada una matriz A y un escalar γ, hallar la matriz SP que resulta de multiplicar el escalar por la matriz.

```
for K ← 1 step 1 to nonuloA do
    SPE[K] ← γAE[k];SPJ[K] ← AJ[K]
for K ← 1 step 1 to filasA + 1 do
    SPI[K] ← AI[K]
```

- **Suma de matrices:** Dadas dos matrices A y B, hallar la matriz S, suma de ambas.

```
I ← 0; L ← 1; M ← 1
for K ← 1 step 1 to nonuloS do
    S ← I + 1
    doloop
        case AJ[L] = BJ[M]
            I ← I + 1;
            SE[I] ← AE[L] + BE[M]
            if SE[I] ≠ 0 then
                SJ[I] ← AJ[L]; SI[K] ← S;
                L ← L + 1; M ← M + 1.
            else
                I ← I − 1
        case AJ[L] < BJ[M]
            I ← I + 1;
            SE[I] ← AE[L]; SJ[I] ← AJ[L]; SI[K] ← S;
            L ← L + 1.
        case AJ[L] > BJ[M]
            I ← I + 1;
            SE[I] ← BE[M]; SJ[I] ← BJ[M]; SI[K] ← S;
            M ← M + 1.
        if M ← BI[K + 1] then
            while L < AI[K + 1] do
            I ← I + 1;
            SE[I] ← AE[M]; SJ[I] ← AJ[L]; SI[K] ← S;
            L ← L + 1
        if L ← AI[K + 1] then
            while M < BI[K + 1] do
            I ← I + 1;
            SE[I] ← BE[M]; SJ[I] ← BJ[M]; SI[K] ← S;
            M ← M + 1
SI[N + 1] ← I + 1
```

Para la aplicación eficiente de métodos iterativos a la solución de sistemas algebraicos, es necesario contar con un algoritmo que permita realizar el producto de una matriz por un vector para realizar las operaciones elementales $y = Ax$ y $y = A^T x$. A continuación damos ambos algoritmos:

- `Producto de una matrix por un vector: Dada una matriz` A `y un vector` x`, representado como una matriz columna, hallar el vector` p `que resulta de` $Ax = p$.

  for $I \leftarrow 1$ **step** 1 **to** *columnasA* **do**
    for $J \leftarrow AI[I]$ **step** 1 **to** $AI[I+1]-1$ **do**
      $p[I] \leftarrow p[I] + AE[J] * x[AJ[J]]$

- `Producto de una matriz por un vector: Dada una matriz` A `y un vector` x`, representado como una matriz columna, hallar el vector` q `que resulta de` $A^T x = q$.

  for $J \leftarrow 1$ **step** 1 **to** *filasA* **do**
    for $I \leftarrow AI[J]$ **step** 1 **to** $AI[J+1]-1$ **do**
      $q[AJ[I]] \leftarrow q[AJ[I]] + AE[I] * x[J]$

- `Producto de dos matrices: Dadas dos matrices` A `y` B`, conformables, hallar la matriz` P`, producto de ambas.`

  $J \leftarrow 1$
  for $K \leftarrow 1$ **step** 1 **to** *filasA* **do**
    for $L \leftarrow 1$ **step** 1 **to** *columnasB* **do**
    $h \leftarrow 0$
      for $M \leftarrow AI[K]$ **step** 1 **to** $AI[K+1]-1$
        while $M <$ *nonuloB* $- 1$ **do**
          $S \leftarrow 0$; $N \leftarrow N + 1$;
          if $BJ[N] \leftarrow L$ **then**
            for $P \leftarrow 1$ **step** 1 **to** *filasB* $+ 1$ **do**
            if $BI[P] > M$ **and** $S \leftarrow 0$ **then**
              if $H \leftarrow 0$ **then**
                $T \leftarrow T + 1$;
                $PJ[T] \leftarrow L$;
                $PI[K] \leftarrow J$;
              else
                $PE[T] \leftarrow PE[T] + AE[M] * BE[N]$;
                $S \leftarrow 1$
      $M \leftarrow 0$
  $J \leftarrow T + 1$;
  $PI[filasA + 1] \leftarrow T + 1$

Los algoritmos antes enunciados, implementados en forma de subrutinas, permitirán resolver los otros dos problemas de álgebra lineal que presentamos en nuestra introducción. A continuación presentaremos los algoritmos sugeridos para resolver sistemas algebraicos de ecuaciones lineales y el problema de valores propios. En ellos utilizaremos los presentados en el apartado anterior como si ya fuesen rutinas implementadas en una biblioteca a las que se hacen llamadas.

Resolución de sistemas lineales de ecuaciones algebraicas: Este algoritmo está restringido a sistemas de ecuaciones determinados, de modo que no se provée validación alguna a este respecto. Dado el caracter ralo de la matriz de coeficiente y para preservarlo en los pasos intermedios de cálculo, se implementa un esquema iterativo de solución; como ejemplo, implementamos un esquema de Jacobi (Cfr. [7], [8] y [9]). Con esto presente, definimos el problema en los siguientes términos:

- `Hallar el vector` x, `que satisface el sistema` $A \cdot x = b$

  if $max < error_{tolerable}$ **do**

  $max \leftarrow 0$

  for $L \leftarrow 1$ **step** 1 **to** $filasA$ **do**

  for $M \leftarrow AI[L]$ **step** 1 **to** $AI[L+1]-1$ **do**

  if $AJ[M] \leftarrow L$ **then** $diag \leftarrow AE[M]$

  else $suma \leftarrow suma + AE[M] * x[AJ[M]]$

  $suma \leftarrow -suma + b(L)$

  $\epsilon(L) \leftarrow x(L) - suma/diag$

  $x(L) \leftarrow suma/diag$

  if $\mid \epsilon(L) \mid > max$ **then** $max \leftarrow \mid \epsilon(L) \mid$

  $suma \leftarrow 0$

En el apartado anterior comentamos algo sobre la necesidad de contar con un algoritmo para el cálculo de la inversa de una matriz. Podemos construir un algoritmo para el cálculo de lainversa de una matriz a partir de la siguiente observación: supongamos tener un conjunto de sistemas de ecuaciones algebraicas, donde lo único que cambia de sistema a sistema es el vector de términos independientes. Este puede escribirse en forma compacta del siguiente modo:

$$AX = B,$$

donde B es la matriz que se construye a partir de todos los vectores de términos independientes, que constituirán las distintas columnas de B y X es ahora una matriz de incógnitas. Si hacemos $B \leftarrow I$, es evidente que $X \leftarrow A^{-1}$, de modo que bastará con resolver simultáneamente los sistemas de ecuaciones dados por:

$$AX = I.$$

Si elegimos resolver este sistema por medio de un esquema iterativo, debemos tener en cuenta algunos detalles: en estos esquemas, la solución que se obtiene es una aproximación de A^{-1}. De acuerdo al criterio de corte del proceso iterativo y al tipo de computadora en que se realice la operación, existirá una familia de matrices $\mathcal{A}^{-1}$ numéricamente indistinguibles de A^{-1}. En este sentido, y antes de la implementación, deberán fijarse criterios para decidir si $\mathcal{A}^{-1}$ es una buena aproximación de A^{-1}.

Problema de valores propios: El problema de autovalores es un problema central en muchos modelos físicos, en particular en el estudio de estabilidad de los problemas de valor de contorno. Como se establece en [7], este es un problema particularmente delicado desde el punto de vista numérico y por consiguiente suele recomendarse el uso de paquetes de rutinas ya probados, como es el caso de EISPACK [10]. Los buenos paquetes proveen rutinas separadas para las siguientes tareas:

- Hallar todos los autovalores y ningún autovector,
- Hallar todos los autovalores y algunos autovectores asociados,
- Hallar todos los autovalores y todos los autovectores correspondientes.

Es evidente que la motivación para separar las tareas radica en la necesidad de optimizar los recursos computacionales. Además de esta separación en tareas, los paquetes proveen rutinas separadas para los distintos tipos de matrices: reales simétricas o no, ralas, con estructura de banda o sin ninguna estructura, complejas hermíticas o no.
De lo que expusimos más arriba es que decidimos presentar en esta apartado un algoritmo que permita transformar una matriz real genérica, almacenada bajo la estructura de datos del esquema CRS, a su forma de Hessenberg, para así aplicar a esta nueva matriz el algoritmo QR, como se encuentra por ejemplo en [10], o el método de Hyman [8]. La única precaución que debe tomarse es la siguiente: las matrices reales, no simétricas pueden tener autovalores complejos, por lo que debería contarse con la necesidad de utilizar aritmética compleja; esto sin embargo puede evitarse con una adecuada selección de traslaciones (*shifts*) de los valores propios [8,9].
Para reducir una matriz a su forma de Hessenberg, bastará o bien con una sucesión de transformaciones de Householder, o bien con un esquema de eliminación del tipo gaussiano; nosotros optaremos por este último aunque con algunas modificaciones, ya que la eliminación de Gauss con pivot parcial no aplica una transformación de semejanza sobre la matriz A. Para que esto ocurra es necesario además de intercambiar las filas, intercambiar las columnas correspondientes. Para implementar la estrategia de pivoteo a la matriz almacenada en forma compacta, podemos hacer la siguiente observación: construyamos una matriz $P_{r,s}$ como:

$$P_{r,s} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & & 1 & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & 1 & & & & 0 & \\ & & & & & & & & 1 \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix},$$

donde los elementos nulos de la diagonal, indican las filas r y s, la que tiene las siguientes propiedades:

- Es una matriz tal que $P_{r,s}^{-1} = P_{r,s}$.
- Hacer $P_{r,s}^{-1}A$ tiene el efecto de intercambiar las r–ésima y s–ésima filas.
- Hacer $AP_{r,s}$ tiene el efecto de intercambiar las r–ésima y s–ésima columnas.

Es evidente que si la s–ésima fila contiene el elemento de mayor valor absoluto de la r–ésima columna, la operación $P_{r,s}^{-1}A$ define la operación usual de pivoteo parcial en la que se intercambian filas. Si además hacemos una postmultiplicación por $P_{r,s}$, nos aseguramos que la transformación que aplicamos sea de semejanza, ya que por definición, $P_{r,s}^{-1}AP_{r,s}$ lo es.
Para la eliminación gaussiana, podemos definir una matriz de la forma:

$$G_j = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l_{j+1\,j} & 1 & \\ & & \vdots & & \ddots \\ & & l_{n\,j} & & 1 \end{pmatrix},$$

donde $|\, l_{ij} \,| \leq 1$, con las siguientes propiedades:

- $$G_j^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{j+1\,j} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n\,j} & & 1 \end{pmatrix}.$$

- Hacer $G_j^{-1}A$ tiene el efecto de restar l_{rj} veces la fila j de la fila r de A, para $r = j+1, j+2, \ldots, n$.
- Hacer AG_j tiene el efecto de sumar l_{rj} veces la columna j a la columna r de A, para $r = j+1, j+2, \ldots, n$.

Si construimos una matriz de transformación del tipo $T_i = P_{r,i+1}G_{i+1}$, y aplicamos sobre A la transformación:

$$T_i^{-1}A_{i-1}T_{i-1}, \qquad r \leq i+1,$$

esta será una transformación de semejanza que permite generar una sucesión de matrices

$$A := A_0 \to A_1 \to A_2 \ldots \to A_{n-2} := \mathcal{H},$$

donde $\mathcal{H}$ es la forma de Hessenberg de la matriz A. El algoritmo puede construirse ahora a partir de la rutina propuesta para multiplicar matrices almacenadas en forma compacta expuesta más arriba. Lo formalizaremos en los siguientes términos:

- **Dada una matriz** A **real, reducirla a la forma de Hessenberg.**

  for $I \leftarrow 2$ **step** 1 **to** $columnasA - 1$ **do**

  Hallar el elemento de mayor valor absoluto en la $(I-1)$**–ésima columna;**

  $m \leftarrow \max_{I \leq J \leq filaA} |\, a_{J\,I-1} \,|$;

  $r \leftarrow J$

  $A' \leftarrow P_{r,I}^{-1} A_{I-2} P_{r,I}$ 'Llamada al algoritmo antes definido

  for $J \leftarrow I+1$ **step** 1 **to** $filasA$ **do**

  if $a'_{I\,I-1} \neq 0$ **then**

  $l_{JI} \leftarrow \frac{a'_{J\,I-1}}{a'_{I\,I-1}}$

  $A_{I-1} \leftarrow G_I^{-1} A G_I$ 'Llamada al algoritmo antes definido

COMENTARIOS FINALES

1. En ánimos de estimar performances, podemos definir un *índice de nulidad* α, como:

$$\alpha = \frac{\textit{cantidad de elementos nulos}}{\textit{cantidad total de elementos}}$$

2. Se necesitarán almacenar $2\ p + n + 1$ variables, donde n es el número de filas o columnas de la matriz cuadrada y p la cantidad de elementos no nulos de la matriz. Un criterio conservativo nos indica que el índice de nulidad crítico para que este esquema de representación sea conveniente al almacenar matrices cuadradas es de 0.5. Esto sale de determinar cual es el índice de nulidad para el cual la cantidad de datos a almacenar en esta representación y en la usual es igual, para el límite de los grandes n.

3. Sólo el vector AE de los elementos no nulos de A requiere definir variables reales o complejas, en simple o doble precisión. Los vectores AJ y AI requieren variables enteras, ya que allí sólo se almacenan posiciones.

4. Un criterio un poco más preciso se obtiene a partir del volumen efectivo de memoria que se requiere para almacenar toda la información contenida en la matriz. Para aclarar un poco la idea, supongamos que almacenaramos los números en doble presición; así, cada elemento no nulo requerirá 8 bytes para ser almacenado, mientras que cada elemento de AJ y de AI requerirá 2 bytes. A continuación hacemos una comparación, para tres matrices y tres índices de nulidad; en la tabla, indicamos con n el tamaño de la matriz y a continuación el espacio requerido para almacenarla como un arreglo de $n \times n$:

α	$n = 1000$	$n = 3000$	$n = 5000$
	7.63 Mb	68.66 Mb	190.73
.5	4.77 Mb	42.92 Mb	119.73 Mb
.6	3.81 Mb	34.34 Mb	95.38 Mb
.7	2.86 Mb	25.75 Mb	71.53 Mb

5. Es evidente que el costo de almacenamiento se reduce si las matrices son simétricas, ya que basta con que se almacenen los elementos no nulos a_{ij} tales que $i \leq j$. Esto deberá evaluarse teniendo en cuenta la complicación adicional que sugiere esta estructura de datos a la hora de realizar una aplicación.

6. Cuando establecimos el algoritmo para la resolución de sistemas de ecuaciones algebraicas, nosotros optamos por un esquema iterativo que preserva el caracter ralo de la matriz. Sin embargo, existen técnicas de eliminación directa, especialmente aptos cuando se trata de matrices de coeficientes simétricas y definidas positivas, basadas en el método de Cholesky. Para una reseña detallada de estas, referirse a [8] y las referencias allí indicadas.

7. Los algoritmos para cálculo de valores propios exhiben ciertas características indeseadas debido a los efectos de los errores de redondeo, los que pueden verse sensiblemente reducidos con la utilización de *técnicas de balanceo*. Puede verse que el error en la solución del problema de valores propios es proporcional a la 2-norma de la matriz, esto es, proporcional a la raiz cuadrada de la suma de los cuadrados de los elementos de la matriz; la idea subyacente de estas técnicas de balanceo es usar una serie de transformaciones de semejanza de manera de que las normas de los correspondientes vectores fila y columna sean comparables, reduciendo de este modo la norma de la matriz pero manteniendo intactos los valores propios. Vale comentar que una matriz simétrica está totalmente balanceada. Otra aclaración interesante es que no es necesario que la norma sobre la que se realicen las operaciones de reducción y balanceo sea la euclídea: una reducción en una p-norma implica una reducción en cualquier otra.

8. Este trabajo se desarrolló en el marco del Proyecto de Tecnologías de Control Activo, Universidad Tecnológica Nacional, Facultad Regional Haedo.

REFERENCIAS

[1] **Gustavson, F.** citado en *Numerical methods* de Dahlquist G. y Björk Å, Prentice-Hall, 1969.

[2] **Barret, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijhout, V., Pozo, R., Romine, C. y van der Vost, H.** *Templates for the solution of linear systems: building blocks for iterative methods*, SIAM, 1994.

[3] **Dongarra, J., Du Croz, J., Hammarling, S. y Hanson, R. J.** *An extended set of* FORTRAN *basic linear algebra subprograms*, Tech. Mem. 41, Rev. 3, Argonne Nat. Lab., Sept. 1986.

[4] **Dongarra, J., Du Croz, J., Duff, I. y Hammarling, S.** *A set of level 3 basic linear algebra subprograms*, Preprint No. 1, Argonne Nat. Lab., Aug. 1988.

[5] **Lawson, C., Hanson, R., Kincaid, D. y Krogh, F.** *Basic linear algebra subprograms for* FORTRAN *usage*, ACM Trans. on Math. Soft. **5**, 308–325, 1979.

[6] *Basic linear algebra subprograms. A quick reference guide*, Univ. of Tennesse, Oak Ridge Nat. Lab. y N.A.G. Ltd., Nov. 1993. (Disponible via ftp de Netlib).

[7] **Press, W., Teukolsky, S., Vetterling, W. y Flannery, B.** *Numerical recipies in* FORTRAN. *The art of scientific computing*, 2nd. Edition, Cambridge Univ. Press, 1992.

[8] **Stoër, J. y Bullirsch, R.** *An introduction to numerical analysis*, 2nd. Edition, Springer-Verlag, 1991.

[9] **Golub, G. H. y Van Loan, C. F.** *Matrix computations*, The John Hopkins University Press, 1983.

[10] **Smith, B.T. et al.** *Matrix eigensystem routines – EISPACK guide*, 2nd. Edition, Lectures Notes in Computer Sciences, Vol 6, Springer-Verlag, 1976.