PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 09-03

Mejoras a la función de adición de matrices dispersas en MatLab

Pablo Ezzatti **Gastón Simone**

2009

Mejoras a la función de adición de matrices dispersas en MatLab
Ezzatti, Pablo; Simone, Gastón
ISSN 0797-6410
Reporte Técnico RT 09-03
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, marzo de 2009

Mejoras a la función de adición de matrices dispersas en MatLab

Pablo Ezzatti
pezzatti@fing.edu.uy

Gastón Simone
gsimone@fing.edu.uy

Facultad de Ingeniería, Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay

marzo 2009

Palabras claves: MatLab, matrices dispersas, suma simbólica.

Resumen

La utilización de matrices dispersas se encuentra fuertemente difundida en la computación científica. Una de las herramientas más utilizadas para la resolución de problemas ingenieriles es MatLab.

MatLab dispone de opciones para trabajar con almacenamiento para matrices dispersas. Además, incluye diversos algoritmos diseñados particularmente para trabajar con dichas matrices. Sin embargo, MatLab posee una deficiencia en la forma que realiza la suma de matrices dispersas. El algoritmo disponible prevé la memoria para la matriz resultado en forma pesimista.

El trabajo presenta una propuesta de función de adición de matrices dispersas que permite mitigar los problemas de memoria de la función propietaria y además mostró un desempeño computacional superior.

1. Introducción

En las últimas décadas, la computación científica a sufrido un gran avance. Una de las líneas de trabajo de mayor expansión ha sido el manejo de matrices y en particular el estudio e investigación en los algoritmos y estrategias de almacenamiento para matrices en formatos dispersos o ralos.

La utilización de formatos de almacenamiento disperso para matrices permite modelar problemas de grandes dimensiones, los cuales serían imposible de tratar trabajando con matrices completas debido a limitantes de memoria. Algunas de las áreas de la computación científica donde se puede constatar el importante avance en la utilización de las matrices dispersas son los métodos de elementos finitos, la computación gráfica, los métodos de diferencias finitas y las estrategias de optimización.

El esfuerzo de la comunidad científica por el avance de las estrategias de trabajo con matrices dispersas no solo se focalizó en el ahorro de memoria para el almacenamiento de las matrices sino que también ha implicado la búsqueda y desarrollo de algoritmos particulares para explotar las características de las estructuras de las matrices. Algunos ejemplos de estas técnicas son los algoritmos de reordenamiento de filas/columnas [4], los métodos multifrontales para la factorización LU [7] y los algoritmos para multiplicar matrices dispersas [8].

Una de las herramientas más difundidas en el ámbito científico es MatLab [2]. MatLab es ampliamente utilizado tanto en el ámbito académico como para la resolución de problemas ingenieriles. Dicha herramienta dispone de opciones para trabajar con almacenamiento de matrices en formato disperso, en particular utiliza la estrategia de almacenamiento comprimido por columnas (CSC del ingles Compressed Sparse Column [5]). También dispone de implementaciones de diversos algoritmos particulares para explotar dicho formato [6].

Si bien MatLab posee implementación de diversas funciones para manejar almacenamiento disperso, posee una desventaja en la forma que realiza la suma de matrices dispersas. El algoritmo de suma estima la memoria a utilizar por la matriz resultado en forma pesimista y la reserva. Esto implica, que en pocas operaciones podamos estar frente al problema de grandes volúmenes de memoria mal utilizada, perdiendo así uno de las grandes beneficios de utilizar las matrices dispersas.

En este trabajo se presenta una función de adición de matrices dispersas para MatLab que mejora el uso de memoria y posee un desempeño computacional, desde el punto de vista de tiempo de ejecución, superior al del algoritmo propietario. La función está implementada en el lenguaje C y utiliza la interfaz que ofrece MatLab para el uso de funciones externas.

Lo que resta de este documento se estructura de la forma que se describe a continuación.

Primero se expone una introducción a las matrices dispersas en MatLab con particular atención en la función de adición. También, se describen los casos de prueba utilizados para evaluar la propuesta. Luego, en la Sección 3, se ofrece un resumen de la propuesta de función realizada. A continuación, se presentan los resultados obtenidos. Por último, se presentan las conclusiones y trabajos futuros planteados.

2. Planteo del Problema

Desde el trabajo de Gilbert, Moler y Schreiber [6] que data del año 1992 MatLab incluye de manera propietaria el almacenamiento y manejo de matrices en formato disperso. En particular el MatLab utiliza el formato de almacenamiento disperso comprimido por columnas (CSC - Compressed Sparse Column) [5].

El formato comprimido por columna utiliza para el almacenamiento de una matriz tres vectores. Más concretamente, para almacenar una matriz de m filas y n columnas con nnz coeficientes no cero se utiliza: un vector d de tamaño nnz y tipo de datos punto flotante en el que se almacenan los valores de los coeficientes; otro vector f de tamaño nnz y tipo de datos enteros en el que se almacenan los números de fila de los elementos distintos de cero; por último, un vector c de enteros de tamaño $n+1$ siendo n la cantidad de columnas de la matriz, en el cual se almacena la posición de la primera ocurrencia de cada columna.

En la Figura 1 se puede observar una matriz A de dimensiones 7×7 en formato completo y en la expresión 2 la misma matriz utilizando la estrategia CSC para almacenarla. En este ejemplo, se necesitan 392 bytes ($7 \times 7 \times 8$) para almacenar la matriz en formato completo trabajando con coeficientes en punto flotante doble precisión (8 bytes) mientras que para almacenarla en formato CSC se necesitan 248 bytes trabajando con índices de 4 bytes (18×8 de 18 coeficientes no cero de 8 bytes + 18×4 para los valores de las filas de los coeficientes no cero + 8×4 para los apuntadores de columnas).

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

$$\begin{aligned} d &= \begin{pmatrix} 1 & 1 & 2 & 5 & 7 & 3 & 8 & 6 & 6 & 9 & 2 & 3 & 4 & 7 & 4 & 5 & 8 & 9 \end{pmatrix} \\ f &= \begin{pmatrix} 1 & 4 & 1 & 2 & 3 & 5 & 3 & 6 & 2 & 3 & 4 & 1 & 5 & 6 & 1 & 5 & 6 & 7 \end{pmatrix} \\ c &= \begin{pmatrix} 1 & 3 & 7 & 9 & 12 & 15 & 18 & 19 \end{pmatrix} \end{aligned} \tag{2}$$

La forma de acceder al valor $A(i,j)$ de la matriz A almacenada en formato CSC es la siguiente, se obtienen primero los índice $p_1 = c(j)$ y $p_2 = c(j+1)$, luego se busca el índice p tal que $p_1 \leq p < p_2$ y $f(p) = i$, luego se puede obtener el valor buscado accediendo a $d(p)$.

Una desventaja del formato CSC es la necesidad de conocer todos las posiciones de los coeficientes no cero para generar la estructura en forma eficiente, o dicho de otra forma es muy difícil agregar un nuevo valor a la estructura. Este problema es compartido por todas las estrategias estáticas [5]. En contraposición, es fácil acceder a los elementos de una columna dada (pero no a los de una fila).

La especificación, según el trabajo de Gilbert et al. [6], de la operación de suma de matrices dispersas en MatLab indica que el algoritmo reserva memoria según la expresión 3, donde S y T son las matrices a sumar de dimensiones $m \times n$ y $nnz()$ la cantidad de elementos no-ceros de las matrices.

$$min(nnz(S) + nnz(T), mn) \tag{3}$$

2.1. Casos de prueba

La totalidad de los casos de prueba fueron generados con la función $sprandn$ de MatLab [1] que genera matrices dispersas en forma aleatoria en base a las dimensiones y la densidad deseada.

La Tabla 1 resume las principales características de las distintas matrices generadas para realizar los estudios, detallándose el rango de las matrices, la densidad con la que se invocó la función de generación, la cantidad real de coeficientes no cero y la memoria utilizada medida en bytes.

matriz	rango	densidad	nnz()	mem utilizada (bytes)
B	10×10	0.4	31	416
C	10×10	0.4	30	404
D	1000×1000	0.4	329636	3959636
E	1000×1000	0.4	329429	3957152
F	2000×2000	0.3	1318713	15832560
G	2000×2000	0.3	1319106	15837276

Tabla 1: Casos de prueba.

2.2. Memoria utilizada por la adición de matrices dispersas de MatLab

En la Tabla 2 se presentan algunos ejemplos de la memoria que se reserva para la matriz resultado de la suma matrices dispersas en MatLab, detallándose el rango de la matriz, la cantidad de coeficientes no nulos y la memoria utilizada medida en bytes.

matriz	rango	nnz()	mem teórica (bytes)	mem utilizada (bytes)
$B + C$	10×10	50	644	692
$D + E$	1000×1000	550613	6611360	7912784
$F + G$	2000×2000	2202709	26440512	31661832

Tabla 2: Memoria utilizada por la adición de matrices dispersas en MatLab.

Observando la Tabla 2 se puede corroborar que MatLab utiliza la expresión de la formula 3 para reservar la memoria del resultado de la suma de matrices dispersas.

En la Tabla 3 se ofrecen algunos ejemplos catastróficos de adición de matrices dispersas en MatLab. Estos ejemplos permiten reforzar la idea de los problemas de memoria al utilizar la operación de adición de MatLab. Primero se suma una matriz B y su opuesta. Luego se compara la memoria necesaria para realizar la operación $B+B$ y la operación $2*B$ en MatLab. Deduciendo que el resultado de ambas operaciones es el mismo y sin embargo las necesidades de memoria en MatLab son completamente distintas.

matriz	rango	nnz()	mem teórica (bytes)	memoria utilizada (bytes)
$B-B$	10×10	0	0	788
$2*B$	10×10	31	416	416
$B+B$	10×10	31	416	788

Tabla 3: Casos catastróficos de la función de adición de matrices dispersas en MatLab

Observando los ejemplos expuestos, se puede concluir que la función de adición de marices dispersas en MatLab posee un problema en la reserva de memoria para la matriz resultado que limita su utilización.

3. Propuesta

La propuesta se centra en una función invocable desde el entorno MatLab que permita realizar sumas de matrices dispersas en el formato CSC. El objetivo es que la función posea un costo computacional (tiempo de ejecución) similar al de la función de adición de matrices dispersas ofrecida por la herramienta pero mejorando el uso de memoria.

Al momento de diseñar la nueva función de adición de matrices dispersas se presentaron diferentes alternativas de trabajo. Una primera opción es realizar la suma utilizando la operación propietaria (con exceso de uso de memoria) y luego copiar el resultado a una estructura de memoria óptima. La segunda opción es realizar una etapa preliminar de suma “simbólica” en la cual se analice las posiciones donde habrá coeficientes no ceros y así estimar y reservar la memoria necesaria. Por último, otra opción es dejar en manos del usuario el tamaño de la matriz resultado, recibiendo dicho dato como parámetro.

La opción uno posee dos grandes desventajas. Primero tiene un *overhead* de tiempo de cómputo para copiar la memoria. Segundo, y más importante, necesita (temporalmente) más memoria que la rutina original ya que utiliza la memoria para el resultado temporal (la misma cantidad que la rutina original) más la memoria para el resultado final.

La principal desventaja de la segunda opción es el costo extra en tiempo de cálculo, ya que aún realizando las operaciones únicamente con índices, o sea valores enteros, el costo de cómputo es relativamente alto.

En cuanto a la tercera opción, posee como desventaja la necesidad de contar con un parámetro extra en la función. Sin embargo, cuando se trabaja con matrices dispersas no es raro conocer de antemano el patrón de la matriz resultado[1] y adicionalmente de no conocer el patrón se puede efectuar una adición para averiguar el tamaño de la matriz resultado con un costo marginal en el caso de estar trabajando con algún algoritmo repetitivo.

En base a lo expresado anteriormente y buscando respetar los requerimientos de mantener el desempeño computacional y mejorar el uso de memoria se escogió implementar la tercera opción.

La función se implementó en el lenguaje C y se utilizó la interfaz provista por MatLab para interactuar con dicho lenguaje, la API Mex-File [3]. Mex-File permite entre otra cosas el manejo y creación de matrices en formato disperso CSC compatibles con el entorno de MatLab.

La función propuesta tiene que ser invocada con tres parámetros, las dos matrices dispersas a sumar y la cantidad de coeficientes no ceros de la matriz resultado. La salida de la función se compone de dos valores, la matriz resultado de la suma de las matrices dispersas propiamente dicho y la cantidad real de coeficientes no ceros de la matriz resultado.

4. Resultados

Los casos de prueba utilizados para evaluar la propuesta son los presentados en la sección 2.1. La evaluación se realiza tanto desde el punto de vista de ahorro de memoria como de tiempo de ejecución. Para el caso del estudio de los tiempos de ejecución se presentan los promedios y desviaciones estándares correspondientes de 10 ejecuciones independientes.

4.1. Estudio de memoria utilizada

En las tablas 4 y 5 se describen los resultados de las pruebas presentadas en la sección 2.2 utilizando para efectuar la suma la función propuesta, se utiliza como parámetro el valor óptimo de cantidad de coeficientes no nulos. En las tablas se detalla rango de las matrices, cantidad de coeficientes no cero y memoria utilizada medida en bytes.

matriz	rango	nnz()	mem utilizada (bytes)
$B + C$	10×10	50	644 680
$D + E$	1000×1000	550613	6611360
$F + G$	2000×2000	2202709	26440512

Tabla 4: Memoria utilizada por la función propuesta.

En la tabla 6 se presentan los mismos ejemplos que en la Tabla 4 pero utilizando como parámetro de cantidad de valores no cero de la función propuesta un 5 % más que el valor óptimo.

[1] Un caso típico es la suma de dos matrices con el mismo patrón.

matriz	rango	nnz()	memoria utilizada (bytes)
$B - B$	10×10	0	0
$B + B$	10×10	31	416

Tabla 5: Memoria utilizada por la función propuesta.

matriz	rango	nnz()	mem utilizada (bytes)
$B + C$	10×10	50	680
$D + E$	1000×1000	550613	6941732
$F + G$	2000×2000	2202709	27762132

Tabla 6: Memoria utilizada por la función propuesta.

Observando los distintos ejemplos se puede deducir el importante ahorro de memoria al utilizar la función propuesta, incluso siendo invocada con un valor para el parámetro de coeficientes no cero un 5 % superior al óptimo.

4.2. Estudio de tiempos de ejecución

Todas las pruebas se efectuaron en el mismo entorno, un computador Pentium Dual Core de 1.6 GHz y 1 GB de memoria RAM utilizado en forma dedicada.

En la Tabla 7 se presentan los promedios de los tiempos de ejecución de la suma de matrices dispersas de la versión propietaria como de la función propuesta, brindándose coeficientes no nulos de la matriz resultados, tiempo de la función propietaria medida en segundos, desviación estándar, tiempo de la función propuesta y desviación estándar asociada.

matriz	nnz()	tiempo original (segundos)	desv. est.	tiempo propuesta (segundos)	desv. est.
$D + E$	550613	0.114	0.008	0.021	0.008
$F + G$	2202709	0.493	0.010	0.102	0.009

Tabla 7: Tiempos de ejecución de la función propietaria y propuesta.

Contemplando los tiempos de ejecución presentados se puede concluir que la versión propietaria de la suma de matrices dispersas insume aproximadamente cinco veces más de tiempo que la versión propuesta para efectuar la suma de matrices dispersas de tamaño medio.

5. Conclusiones y trabajo futuro

El reporte introduce en el manejo del formato de almacenamiento de matrices dispersas CSC y en particular presenta el estudio de la operación de adición de matrices dispersas en MatLab.

El resultado más importante del trabajo es la obtención de una función de suma de matrices dispersas para MatLab. La función demostró alcanzar un desempeño computacional superior que la operación propietaria, implicando la versión propietaria en el orden de 5 veces el tiempo de ejecución que el que insume la función propuesta. Además, la función propuesta hace un uso eficiente de memoria.

La función propuesta permite extender la aplicación de MatLab a diversos problemas en los que se utilizan matrices dispersas y las necesidades de memoria eran una limitante.

A continuación se describen las líneas de trabajo que se están llevando adelante en la actualidad o se plantean abordar en un futuro cercano.

Una primera línea de trabajo consiste en extender las pruebas realizadas, abarcando otros ejemplos. Incluso trabajando con matrices dispersas presentes en problemas ingenieriles reales.

Un camino no recorrido en el trabajo debido a limitantes de tiempo es el desarrollo de una función de adición simbólica de matrices dispersas trabajando únicamente con los índices (valores enteros) de forma de poder utilizarla en caso de no disponer de información del patrón de la matriz resultado de la adición.

Otra tarea interesante, es dotar de la funcionalidad de filtrado de coeficientes. La función de adición de matrices dispersa propuesta recorre todos los coeficientes del resultado, entonces no implicaría casi costo (en tiempo de ejecución) filtrar determinados coeficientes. En particular, es muy común querer filtrar los coeficientes menores a determinada cota.

También resulta importante extender el estudio realizado sobre la operación de adición para la multiplicación de matrices dispersas en MatLab.

Por último, parece interesante estudiar la aplicación de estrategias de paralelismo de memoria compartida sobre la operación de suma propuesta.

Referencias

[1] Página web de la ayuda de mathworks para la función *sparnd*,. http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/ref/sprand.html&http://www.google.com.uy/search?hl=es&q=matlab+sprandn&meta=. Consultada marzo 2009.

[2] Sitio web de matlab. http://www.mathworks.com/. Consultada marzo 2009.

[3] Sitio web del soporte de la api mex-file. http://www.mathworks.com/support/tech-notes/1600/1605.html. Consultada marzo 2009.

[4] Patrick Amestoy, Timothy A. Davis, Iain, and S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.

[5] James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[6] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, January 1992.

[7] Joseph W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Rev.*, 34(1):82–109, 1992.

[8] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005.