XXVI Congreso Nacional de Estadística e Investigación Operativa
Úbeda, 6–9 de noviembre de 2001

RESOLVIENDO UNA SUCESIÓN DE SISTEMAS COMPATIBLES DISPERSOS

Ángel Santos Palomo[1], Pablo Guerrero García[1]

[1]Departamento de Matemática Aplicada
Universidad de Málaga

RESUMEN

Describimos cómo utilizar una factorización ortogonal dispersa actualizable explícita para resolver la sucesión de sistemas compatibles dispersos requerida para implementar un método non-simplex de programación lineal. Para ello adaptamos a matrices rectangulares las técnicas de Saunders para matrices cuadradas utilizando la estructura estática de datos de George-Heath pero permitiendo supresión de filas sobre ella; así explotamos los resultados intermedios obtenidos al abordar un problema mínimo-cuadrático procesando su matriz A^T por filas en relación con el programa lineal en forma estándar procesando su matriz A por columnas en el orden fijo dictado por un método non-simplex.

Palabras y frases clave: ortogonalización dispersa, rotaciones de Givens, métodos non-simplex, programación lineal.
Clasificación AMS: 90C05.

1. OBJETIVOS Y DIFICULTADES

En los métodos non-simplex (Gill & Murray, 1973; Gill, Murray & Wright, 1991; Santos, 1998) se requiere resolver una sucesión de sistemas compatibles de la forma:

$$\begin{bmatrix} A_k^T \\ c^T \end{bmatrix} d = \begin{bmatrix} O \\ -1 \end{bmatrix} \quad (c \notin \mathcal{R}(A_k)) \qquad ; \qquad A_k^T x = b_A \qquad ; \qquad A_k y = c$$

donde $A_k^T \in \mathbb{R}^{m_k \times n}$ con $m_k \leq n$ y rango$(A_k) = m_k$; la matriz A_{k+1} se obtiene añadiendo y/o eliminando columnas a/de A_k. La implementación non-simplex propuesta en (Gill & Murray, 1973) se basa en mantener la factorización QR de A_k de forma que las actualizaciones se hacen de una en una columna como se indica en (Golub & Van Loan, 1996, §12.5.2). Dicha implementación es recomendable para problemas densos por su estabilidad respecto a la propagación de los errores de redondeo (ver (Fletcher, 1987):172,234); aunque para problemas dispersos ha sido adaptada en (George, Heath & Ng, 1984), ellos mismos reconocen (ver (George,

Heath & Ng, 1984):995-996) que requiere más trabajo que la factorización QR de A_k^T cuando $m_k \ll n$ y que cuando basta cualquier solución particular (no la de mínima norma) no hay por qué restringir la utilización de la factorización QR de A_k^T al caso en el que $n - m_k$ sea relativamente pequeño (ver (Heath, 1982):229).

Como ya indicaban Gilbert y Peierls en (Gilbert & Peierls, 1988):862

> *Translating a matrix algorithm from a dense setting to a sparse setting may involve more than just generalizing indices and using lists instead of dense arrays.*

la implementación dispersa de un método non-simplex no es ni mucho menos trivial. Si bien hoy día existen "cajas negras"eficientes para el cálculo de la factorización QR dispersa usando técnicas multifrontales, dicha factorización sólo sirve cuando se quiere resolver un único problema mínimo-cuadrático, dado que dicha factorización no es aún actualizable como reconoce (Adlers, 1998):43

> *Multi-frontal techniques are more efficient for factorization of sparse matrices. However, there are no efficient ways yet, to update the QR factorization using multifrontal techniques.*

En este sentido, Mitra y Tamiz decían en (Mitra & Tamiz, 1991):280

> *Considerable progress continues to be made in sparse equation solving methods which may be looked upon "blackbox"procedures. Not all of these provide iterative update procedures. Hence methods for sparse and stable updates to work with sparse and stable solvers can be of considerable value.*

En este artículo se describe cómo resolver la *sucesión* de sistemas compatibles dispersos que requiere un método non-simplex de programación lineal (Santos, 1998; Santos & Guerrero, 1998a; Santos & Guerrero, 1998b) actualizando una factorización QR ortogonal dispersa de A_k^T por adición y supresión de filas como se indica para el caso denso en (Golub & Van Loan, 1996, §12.5.3) pero contemplando la permutación de columnas y su tratamiento disperso. Esta factorización es por naturaleza actualizable al añadirle filas (George & Heath, 1980), el orden de las filas no influye en la densidad del factor triangular y el trabajo disperso con $A_k A_k^T$ es más adecuado que con $A_k^T A_k$; de hecho en (Hager, Shih & Lundin, 1996):23 se propone trabajar con la factorización de $A_k A_k^T \in \mathbb{R}^{n \times n}$ aunque sea semidefinida positiva en vez de con la de $A_k^T A_k \in \mathbb{R}^{m_k \times m_k}$ que es definida positiva:

> *The matrix $A_F^T A_F$ is difficult to work with since its dimension grows and contracts during the iterations, and the sparsity structure is difficult to predict. The matrix $\sigma I_n + A_F A_F^T$ is quite amenable to numerical work since it has fixed dimensions and its sparsity is predictable.*

El primero que adoptó esta idea dispersa fue Michael Saunders en (Saunders, 1972) para el caso $m_k = n$; lo que hacemos en este artículo es adaptar su metodología al caso $m_k < n$ para poder trabajar incluso prescindiendo del factor ortogonal. En este sentido no recurriremos a la práctica habitual de utilizar factorizaciones LU dispersas a pesar de lo apuntado en (Gill & Murray, 1977):355

> *The effort involved in producing a linear-programming package capable of solving large problems on a day-to-day basis is so great that it has understably discouraged the implementation of new methods which depart radically from existing practice.*

dado que estamos interesados en resolver programas lineales dispersos con dificultades numéricas (Maros & Mészáros, 1995):4

> *Though the present day numerical tools in the simplex algorithm, first of all the LU factorizaction of the basis, are believed to be very safe, there can always be some newly arising problems that cause serious numerical troubles and prevent solution.*

De hecho, existe una clase de problemas degenerados que no pueden resolver las implementaciones de punto interior más robustas (e.g., (Stojković & Stanimirović, 2001, §5)). Luego hay razones más importantes que el mero hecho de apartarse de la práctica habitual para preferir una factorización QR frente a una LU: el hecho de que siempre exista, que sea robusta y numéricamente estable, que se pueda resolver fácilmente un sistema compatible a partir de ella y que se pueda estimar fácilmente el número de condición son conocidas ventajas a añadir a (Gill, Murray & Saunders, 1975):1057

> *For a general sparse matrix B of the type encountered in LP it is possible to compute an orthogonal factorization $B = LDV$ in product form whose density is only slightly greater than that of the triangular factorization $B = LU$. This is a surprising result.*

2. FACTORIZACIÓN INICIAL Y REINVERSIÓN

Siendo $A_k \in \mathbb{R}^{n\times k}$ una matriz dispersa tal que $\text{rango}(A_k) = k \leq n$, sabemos que existe una permutación Π_k^T de las columnas de A_k^T tal que

$$A_k^T \Pi_k^T = QR \doteq Q(R_1 \ R_2) \in \mathbb{R}^{k\times n} \qquad (Q, R_1 \in \mathbb{R}^{k\times k}) \wedge (R_2 \in \mathbb{R}^{k\times(n-k)})$$

siendo R trapezoidal superior, R_1 triangular superior y regular y Q ortogonal. (Para el número de columnas de A_k usaremos k en vez de un más acertado m_k por comodidad notacional.) Llamaremos factorización LV de $\Pi_k A_k$ a

$$\Pi_k A_k = \begin{bmatrix} R_1^T \\ R_2^T \end{bmatrix} Q^T \doteq \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} V \doteq LV$$

siendo L trapezoidal inferior, L_1 triangular inferior y regular y V ortogonal. Mediante `qr` en MATLAB 5 puede calcularse esta factorización (pero no siempre con los paquetes QR dispersos de Matstoms (Matstoms, 1994) y Adlers (Adlers, 1998), pues están pensados para el caso de más filas que columnas), y que se puede postprocesar la matriz R como se indica en (Heath, 1982) para obtener una factorización de rango explícito (*rank-revealing*). Nótese que ampliando R con $n-k$ filas de ceros por debajo

lo que obtenemos es el factor de Cholesky disperso de $A_k A_k^T \geq 0$ (ver (Higham, 1996, Teorema 10.9):210) calculado a partir de la factorización QR dispersa para evitar el pivotaje total que se necesitaría en el caso de usar Cholesky semidefinido denso.

Dado que L es base de $\mathcal{R}(\Pi_k A_k)$ catalogaremos esta metodología como espacio imagen, si bien a partir de dicha información podemos disponer de una base Z de $\mathcal{N}(A_k^T \Pi_k^T)$

$$Z = \Pi_k^T \begin{bmatrix} Z_1^T \\ I \end{bmatrix} \qquad \text{, siendo} \qquad Z_1 \doteq -L_2 L_1^{-1}$$

Obsérvese que los "ladrillos" (submatrices) con los que se construye Z se obtienen de L, con lo cual manteniendo L dispersa también se mantendrá Z dispersa si no calculamos Z_1 explícitamente. Supondremos $\Pi_k = I_n$ por simplicidad notacional y prescindiremos del subíndice k cuando sea claramente sobreentendido.

Se ha elegido esta factorización porque el orden de las filas de A_k^T no influye en la densidad del factor triangular. Si A_k es un subconjunto de las columnas de una matriz fija $A \in \mathbb{R}^{n \times m}$ con $m \geq n$ y rango$(A) = n$, se puede utilizar una estructura estática (i.e., determinable a priori) para intentar minimizar el llenado del factor triangular (ordenación previa de las columnas de A^T (i.e., de las filas de A) mediante por ejemplo `colperm(A')`) pues el esqueleto de $A_k A_k^T$ es un subconjunto del de AA^T. Además, el llenado queda confinado sólo a la fila respecto a la que se está rotando y a una fila de trabajo, y es por naturaleza actualizable al añadirle filas (George & Heath, 1980; Heath, 1984). Dado que lo que se pretendía en estos trabajos era $\min \|A^T x - b\|_2$ ninguno de ellos contemplaba el hecho de quitar una fila, pero también es lógico pensar que debe caer dentro de la estructura estática por la observación hecha por Golub en 1969 sobre la equivalencia formal existente entre quitar una fila a^T real y añadir una fila ia^T compleja (tienen el mismo esqueleto y las rotaciones hiperbólicas lo modifican de la misma manera que las rotaciones de Givens).

Aunque el orden de las filas de A^T no influye en la densidad sí lo hace en la cantidad de trabajo intermedio necesario para obtener la factorización; si suponemos que las filas de A^T no varían mucho en norma, la ordenación de filas no afectará a la estabilidad numérica y podrá elegirse basándose sólo en consideraciones de dispersión (ver (Björck, 1996):244–245). Si bien iremos añadiendo filas por el final para aprovechar el trabajo intermedio ya realizado, a la hora de reinvertir (o de calcular una factorización inicial para) un subconjunto de columnas $\mathcal{A}_0$ dado sí puede explotarse una ordenación previa de las filas de A^T (i.e., de las columnas de A) y utilizar la subordenación correspondiente a los índices incluídos en $\mathcal{A}_0$. La reinversión se dispara según una tolerancia prefijada (Murtagh, 1981), y se puede calcular mediante la caja negra anteriormente mencionada con la subordenación de columnas de A_k adecuada.

En programación lineal es habitual reordenar previamente la matriz A de restricciones de un PL en forma estándar para que esté en LBTF. Esto puede conseguirse mediante la primitiva MATLAB `dmperm` para calcular la UBTF de A^T; de esta manera obtendremos tanto el orden de filas como el orden de columnas adecuado, además de obtener un esqueleto disperso máximo necesario para R^T correctamente calculado en el sentido descrito en (Coleman, Edenbrandt & Gilbert, 1986). Tanto A como L utilizan un esquema de almacenamiento comprimido por columnas, pues el acceso a

las mismas se va a hacer por columnas.

Nótese cómo en todo momento explotamos la analogía entre abordar $\min \|A^T x - b\|_2$ con $A^T \in \mathbb{R}^{m\times n}$ procesando A^T fila a fila y el PL en forma estándar $\max b^T y$ sujeto a $Ay = c$, $y \geq 0$ con $A \in \mathbb{R}^{n\times m}$ procesando A columna a columna (en el orden fijo dictado por un método non-simplex). El tratamiento disperso eficiente (p.e., las técnicas multifrontales para el primero o los métodos de punto interior para el segundo) aprovecha en ambos casos el hecho de que A sea una matriz fija conocida de antemano pero no hace uso de los resultados intermedios para progresar (i.e., no aprovecha que A_k es un subconjunto de las columnas de A), como se dice p.e. en (Heath, 1984):507,

Direct methods generally do not produce meaningful intermediate results.

Nosotros sí los usaremos, contemplando que el número de filas de A^T rotadas a R (o equivalentemente, el número de columnas activadas de A) sea menor o igual que n. Esto nos permite relacionar *los resultados intermedios* obtenidos al aplicar dos grandes avances dispersos: la factorización LDV de A_k y la factorización QR de A_k^T.

3. RESOLUCIÓN DE LOS SISTEMAS

Cuando $c \notin \mathcal{R}(A_k)$, una primera posibilidad para obtener una solución del sistema compatible $(A_k^T d = 0) \wedge (c^T d = -1)$ se basa en proyectar $-c$ sobre $\mathcal{N}(A_k^T)$ calculando $-Z(Z^T Z)^{-1} Z^T c$, pero la expresión de $Z(Z^T Z)^{-1} Z^T$ se complica bastante. (Esta solución es una dirección espacio núcleo de descenso más pronunciado.) Dado que $-ZZ^T c$ también es solución aun cuando las columnas de Z no sean ortogonales

$$ZZ^T = \begin{bmatrix} Z_1^T \\ I \end{bmatrix} \begin{bmatrix} Z_1 & I \end{bmatrix} = \begin{bmatrix} Z_1^T Z_1 & Z_1^T \\ Z_1 & I \end{bmatrix}$$

(También es una dirección espacio núcleo de descenso más pronunciado, ver (Gill, Murray & Wright, 1991):378.) Nótese que esta forma sólo conlleva multiplicaciones por Z_1 y Z_1^T, lo cual implica a su vez multiplicaciones por las matrices dispersas L_2 y L_2^T y resoluciones de sistemas con matrices L_1 y L_1^T (que son triangulares inferiores dispersas y supondremos de momento bien condicionadas):

$$d = -ZZ^T c = \begin{bmatrix} -L_1^{-T} L_2^T w \\ w \end{bmatrix} \qquad , \qquad w \doteq c_2 - L_2 L_1^{-1} c_1 \quad \wedge \quad \Pi_k c \doteq \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

Una vez que se dispone de una matriz Z cuyas columnas forman base del $\mathcal{N}(A_k^T)$ se tiene

$$v \in \mathcal{R}(A_k) \quad \Leftrightarrow \quad \forall s \in \mathcal{N}(A_k^T)\ ,\ v^T s = 0 \quad \Leftrightarrow \quad Z^T v = O_{n-k,1}$$

con lo cual $v \notin \mathcal{R}(A_k)$ si y sólo si $Z^T v \neq O_{n-k,1}$. Nótese que si $v = (v_1; v_2)$ con $v_1 \in \mathbb{R}^k$ entonces $Z^T v = Z_1 v_1 + v_2 = -L_2 L_1^{-1} v_1 + v_2$, y que esto coincide con la forma en la que Björck y Duff en 1980 decidían si $v \in \mathcal{R}(A_k)$ sin necesidad de determinar la solución de $\min_y \|A_k y - v\|_2$ y analizar su residuo disponiendo de la

factorización LU dispersa de A_k (método de Peters-Wilkinson modificado, ver por ejemplo (Heath, 1984, Alg. 2)). Numéricamente comprobaremos si $\|Z^T v\| < \epsilon$ para un ϵ convenientemente prefijado.

Cuando $c \in \mathcal{R}(A_k)$ el sistema compatible $\Pi_k A_k y = \Pi_k c = (c_1; c_2)$ podrá resolverse mediante $L(Vy) = \Pi_k c$

$$\hat{y} = L_1^{-1} c_1 \quad \wedge \quad \hat{y} \doteq Vy \qquad \Rightarrow \qquad y = V^T (L_1^{-1} c_1)$$

Para encontrar una solución del sistema $A_k^T x = b_A$ compatible indeterminado, como $A_k^T = QR\Pi_k$ tendremos $R\hat{x} = Q^T b_A$, de donde

$$\hat{x} = \begin{bmatrix} L_1^{-T}(Vb_A) \\ O \end{bmatrix} \quad \wedge \quad x \doteq \Pi_k^T \hat{x} \qquad \Rightarrow \qquad x = \Pi_k^T \begin{bmatrix} L_1^{-T}(Vb_A) \\ O \end{bmatrix}$$

con lo cual se obtiene una solución básica que no será de mínima norma a menos que $L_2 = O$.

Si no se dispone de V podemos recurrir a una especie de ecuaciones seminormales, premultiplicando ambos lados de la ecuación por A_k

$$A_k A_k^T x = \Pi_k^T R^T R \Pi_k x = A_k b_A \qquad \Rightarrow \qquad R^T (R(\Pi_k x)) = \Pi_k (A_k b_A)$$

resolviendo primero el sistema compatible determinado que tiene por matriz R^T y después encontrando una solución básica para el sistema compatible indeterminado que tiene por matriz R trapezoidal superior. Si el que queremos resolver es $A_k y = c$ definimos $y \doteq A_k^T z$ y entonces procedemos igual que antes pero con $A_k A_k^T z = c$

$$A_k A_k^T z = \Pi_k^T R^T R \Pi_k z = c \qquad \Rightarrow \qquad R^T (R(\Pi_k z)) = \Pi_k c$$

En ambos casos el refinamiento iterativo en precisión fija nos permitirá mejorar los resultados obtenidos. Es de reseñar que los sistemas que estamos resolviendo son singulares pero compatibles con matriz de coeficientes semidefinida positiva; si bien el empleo de métodos iterativos para resolverlos es conocido (ver (Higham, 1996, §16.4) y las referencias allí mencionadas), la utilización de métodos directos como el propuesto anteriormente es por lo que nosotros sabemos novedosa.

4. ACTUALIZACIÓN POR ADICIÓN DE LA FACTORIZACIÓN

La adición de la p-ésima restricción al conjunto de trabajo añade una nueva columna al final de A_k, i.e., añade una nueva fila a_p^T debajo de A_k^T:

$$A_{k+1}^T = \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix}$$

Entonces, dada la factorización QR de $A_k^T \Pi_k^T$ se quiere determinar la nueva factorización QR tras la adición de una nueva observación a_p^T; este problema se conoce como actualización por adición (*updating*) de la factorización QR.

Como $A_k^T = Q_k R_k \Pi_k$ podemos escribir

$$A_{k+1}^T = \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix} = \begin{bmatrix} Q_k & O \\ O^T & 1 \end{bmatrix} \begin{bmatrix} R_k \Pi_k \\ a_p^T \end{bmatrix}$$

de donde se obtiene

$$\begin{bmatrix} Q_k^T & O \\ O^T & 1 \end{bmatrix} \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix} = \begin{bmatrix} R_k \Pi_k \\ a_p^T \end{bmatrix} \tag{1}$$

Usando una rotación (ver (Golub & Van Loan, 1996, §5.1.8)) de Givens $G(j, k+1)^T \in \mathbb{R}^{(k+1)\times(k+1)}$ con $j \in 1:k$ para anular el j-ésimo elemento de a_p^T, tras (como máximo) k rotaciones tendremos:

$$G^T \begin{bmatrix} R_k \Pi_k \\ a_p^T \end{bmatrix} \doteq G(k, k+1)^T \cdots G(2, k+1)^T G(1, k+1)^T \begin{bmatrix} R_k \Pi_k \\ a_p^T \end{bmatrix} = R_{k+1} \Pi_{k+1} \tag{2}$$

siendo Π_{k+1} una matriz de permutación que haga que R_{k+1} sea trapezoidal con elementos diagonales no nulos. Entonces, conforme a (1) y (2),

$$A_{k+1}^T \Pi_{k+1}^T = \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix} \Pi_{k+1}^T = \begin{bmatrix} Q_k & O \\ O^T & 1 \end{bmatrix} G G^T \begin{bmatrix} R_k \Pi_k \\ a_p^T \end{bmatrix} \Pi_{k+1}^T \doteq Q_{k+1} R_{k+1}$$

Sabido es que en algoritmos prácticos no es preciso disponer del factor ortogonal, pues se puede actualizar R_k aunque no se disponga de Q_k.

5. ACTUALIZACIÓN POR SUPRESIÓN DE LA FACTORIZACIÓN

Cuando se elimina del conjunto de trabajo la q-ésima restricción supondremos sin pérdida de generalidad que ésta es la primera columna de A_k, i.e., que se elimina la primera fila de A_k^T:

$$A_k^T = \begin{bmatrix} a_q^T \\ A_{k+1}^T \end{bmatrix}$$

ya que si no fuera la primera fila bastaría intercambiar previamente la fila correspondiente con la primera en A_k^T. Entonces, dada la factorización QR de $A_k^T \Pi_k^T$ se quiere determinar la nueva factorización QR tras la supresión de una antigua observación a_q^T; este problema es conocido como actualización por supresión (*downdating*) de la factorización QR.

Si añadimos una columna $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^k$ tendremos

$$Q_k^T \left[e_1 \; A_k^T \right] = \left[q \; R_k \Pi_k \right] \tag{3}$$

donde $q^T \in \mathbb{R}^k$ es la primera fila de Q_k (i.e., $q^T = e_1^T Q_k$). Ahora se usa una rotación de Givens $G(j, j+1)^T \in \mathbb{R}^{k\times k}$ con $j \in k-1:-1:1$ para anular el $(j+1)$-ésimo elemento de q; tras (como máximo) k rotaciones tendremos

$$G^T q \doteq G(1, 2)^T \cdots G(k-2, k-1)^T G(k-1, k)^T q = \alpha e_1 \tag{4}$$

con $\alpha = \pm 1$, de manera que entonces

$$G^T \left[q \; R_k \right] = \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ O & R_{k+1} \Pi_{k+1} \end{bmatrix} \tag{5}$$

donde $R_{k+1}\Pi_{k+1} \in \mathbb{R}^{(k-1)\times n}$ con R_{k+1} trapezoidal superior con elementos diagonales no nulos. De esta manera, dado que la primera fila de la matriz $Q_k G$ ortogonal es

$e_1^T Q_k G = \alpha e_1^T$ por (4), se obtiene introduciendo en (3) la matriz identidad GG^T y aplicando (5) que

$$\left[e_1 \; A_k^T\right] = Q_k G G^T \left[q \; R_k \Pi_k\right] = \begin{bmatrix} \alpha & O^T \\ O & Q_{k+1} \end{bmatrix} \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ O & R_{k+1}\Pi_{k+1} \end{bmatrix}$$

con lo cual ya podemos igualar los términos de ambos lados de la igualdad y obtener, con $v^T \doteq a_q^T \Pi_{k+1}^T$, que

$$\begin{bmatrix} 1 & a_q^T \\ O & A_{k+1}^T \end{bmatrix} = \left[e_1 \; A_k^T\right] = \begin{bmatrix} 1 & v^T \Pi_{k+1} \\ O & Q_{k+1} R_{k+1} \Pi_{k+1} \end{bmatrix}$$

es decir, $A_{k+1}^T \Pi_{k+1}^T = Q_{k+1} R_{k+1}$.

Hemos supuesto q denso en el peor caso; cuando q es disperso, en la rotación intervienen las dos últimas filas en la que q tenga elementos distintos de cero y al final se hace una rotación con respecto a la primera fila para obtener αe_1. El problema con este algoritmo en el caso disperso es la necesidad de efectuar el producto $Q_k G$ para obtener su submatriz Q_{k+1} y el mantenimiento del esqueleto estático, como veremos posteriormente.

En el caso disperso es habitual prescindir de Q_k; veamos que podemos trabajar de forma parecida al algoritmo LINPACK (Saunders, 1972) para actualizar por supresión el factor de Cholesky, puesto que $\Pi_k^T R_k^T q = a_q$ también es un sistema compatible determinado y resolviéndolo se obtiene q. Si ampliamos la matriz $R_k \Pi_k$ con un valor δ_n adecuado que determinaremos más adelante

$$\bar{R}_k \doteq \begin{bmatrix} \delta_n & O^T \\ q & R_k \Pi_k \end{bmatrix}$$

y le aplicamos a $\bar{R}_k$ una rotación de Givens $G(1,j)^T \in \mathbb{R}^{(k+1)\times(k+1)}$ con $j \in k+1 : -1 : 2$ para anular el j-ésimo elemento de su primera columna (i.e., el $(j-1)$-ésimo elemento de q), obtenemos

$$G^T \bar{R}_k \doteq G(1,2)^T \cdots G(1,k)^T G(1,k+1)^T \bar{R}_k = \bar{R}_{k+1}$$

donde $\bar{R}_{k+1}$ es la matriz ampliada correspondiente al nuevo factor con R_{k+1} trapezoidal con elementos diagonales no nulos:

$$\bar{R}_{k+1} \doteq \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ O & R_{k+1}\Pi_{k+1} \end{bmatrix}$$

siendo $\alpha = \pm 1$. Dado que $\bar{R}_{k+1}^T \bar{R}_{k+1} = \bar{R}_k^T \bar{R}_k$ por ser G ortogonal:

$$\begin{bmatrix} \alpha & O^T \\ \alpha \Pi_{k+1}^T v & \Pi_{k+1}^T R_{k+1}^T \end{bmatrix} \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ O & R_{k+1}\Pi_{k+1} \end{bmatrix} = \begin{bmatrix} \delta_n & q^T \\ O & \Pi_k^T R_k^T \end{bmatrix} \begin{bmatrix} \delta_n & O^T \\ q & R_k \Pi_k \end{bmatrix}$$

multiplicando e igualando bloques tendremos que $\Pi_{k+1}^T v = \Pi_k^T R_k^T q = a_q$ y que $\delta_n^2 = 1 - \|q\|_2^2 = 0$, pues

$$\begin{bmatrix} 1 & v^T \Pi_{k+1} \\ \Pi_{k+1}^T v & \Pi_{k+1}^T (R_{k+1}^T R_{k+1} + v v^T) \Pi_{k+1} \end{bmatrix} = \begin{bmatrix} \delta_n^2 + \|q\|_2^2 & q^T R_k \Pi_k \\ \Pi_k^T R_k^T q & \Pi_k^T R_k^T R_k \Pi_k \end{bmatrix}$$

con lo cual $\Pi_{k+1}^T R_{k+1}^T R_{k+1} \Pi_{k+1} = \Pi_k^T R_k^T R_k \Pi_k - a_q a_q^T$, que era lo que queríamos puesto que

$$A_{k+1} A_{k+1}^T = \Pi_{k+1}^T R_{k+1}^T R_{k+1} \Pi_{k+1} = \Pi_k^T R_k^T R_k \Pi_k - a_q a_q^T = A_k A_k^T - a_q a_q^T$$

Nótese que a diferencia del algoritmo anterior, ahora la matrices de Givens son $(k+1) \times (k+1)$. La utilización de la fila adicional permite recabar información útil para disparar una reinversión; además, la estructura de datos tiene en todo momento espacio asignado para trabajar, y el rellenado queda confinado al vector de trabajo, al igual que pasaba al actualizar por adición pero no en el de actualizar por supresión anteriormente descrito.

Ejemplo. Considérese la matriz cuyo esqueleto se presenta en (George, Liu & Ng, 1988):104. Mediante el comando MATLAB `symbfact(A','col')` podemos obtener el esqueleto máximo de R^T:

$$A^T = \begin{bmatrix} \times & & & & & \times \\ & \times & & \times & & \\ 1 & & -1 & & & \\ & \times & & \times & & \times \\ & & & & \times & \times \\ 1 & & & & & 2 \\ & & -1 & -1 & & \\ & \times & & \times & & \end{bmatrix} \quad ; \quad R^T = \begin{bmatrix} \times & & & & & \\ & \times & & & & \\ \times & & \times & & & \\ & \times & \times & \times & & \\ & & & & \times & \\ \times & \times & \times & \times & \times & \times \end{bmatrix}$$

Nótese que hemos considerado R^T en vez de R por cuestiones de localidad de memoria (MATLAB almacena las matrices dispersas por columnas). Tras rotar las filas a_6^T y a_7^T en R nos disponemos a rotar a_3^T:

$$[R^T | a_3] = \left[\begin{array}{cccccc|c} 1 & & & & & & 1 \\ & \times & & & & & \\ \times & & -1 & & & & -1 \\ & \times & -1 & \times & & & \\ & & & & \times & & \\ 2 & \times & \times & \times & \times & \times & \end{array}\right] \Rightarrow [\Pi_2^T R_2^T | a_3] = \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 0 & 0 \\ \mathbf{0} & -1 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ 2 & \mathbf{0} & 0 \end{array}\right]$$

donde el vector de trabajo queda a la derecha de la barra vertical; los elementos del vector de trabajo los vamos anulando de arriba a abajo mediante rotaciones de Givens, postmultiplicando las columnas 1 y 3 por $\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$ y luego las columas 2

y 3 por $\begin{bmatrix} \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{3} & \sqrt{6}/3 \end{bmatrix}$:

$$\left[\begin{array}{cc|c} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -1 & -1/\sqrt{2} \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ \sqrt{2} & \mathbf{0} & -\sqrt{2} \end{array}\right] \quad \Rightarrow \quad \Pi_3^T R_3^T = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{bmatrix}$$

Supongamos que ahora queremos eliminar a_7^T. En primer lugar resolvemos el sistema compatible $\Pi_3^T R_3^T q = a_7$ obteniendo $q = (0, \sqrt{6}/3, -1/\sqrt{3})^T$; nótese que la resolución de este sistema no requiere una sustitución progresiva especial pues puede hacerse superponiendo $\Pi_3^T R_3^T$ sobre I_6 para obtener $(0, 0, \sqrt{6}/3, -1/\sqrt{3}, 0, 0)^T$ y luego quedarse con las componentes primera, tercera y cuarta. Ahora ampliamos $\Pi_3^T R_3^T$ y rotamos respecto a la primera columna para ir eliminado los elementos de q^T de derecha a izquierda:

$$\bar{R}_3^T \doteq \begin{bmatrix} \delta_n & q^T \\ O & \Pi_3^T R_3^T \end{bmatrix} \qquad \Rightarrow \qquad \bar{R}_3^T \doteq \left[\begin{array}{c|ccc} 0 & 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ \hline 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{array}\right]$$

donde el vector de trabajo queda ahora a la izquierda de la barra vertical; primero postmultiplicamos las columnas 1 y 4 por $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ y luego las columas 1 y 3 por $\begin{bmatrix} -1/\sqrt{3} & \sqrt{6}/3 \\ -\sqrt{6}/3 & -1/\sqrt{3} \end{bmatrix}$ para obtener

$$\left[\begin{array}{c|ccc} 1/\sqrt{3} & 0 & \sqrt{6}/3 & 0 \\ \hline 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ -1/\sqrt{3} & 0 & -\sqrt{6}/3 & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ 2/\sqrt{3} & \sqrt{2} & -\sqrt{6}/3 & \mathbf{0} \end{array}\right] \quad \Rightarrow \quad \bar{R}_4^T \doteq \left[\begin{array}{c|ccc} -1 & 0 & 0 & 0 \\ \hline 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & 0 & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & \sqrt{2} & \mathbf{0} \end{array}\right]$$

Nótese que, salvo signos, se obtuvo lo mismo que tras anular el primer elemento al rotar a_3^T, que apareció en la columna adicional el vector a_7 y que en este caso $\alpha = -1$. Aunque aquí se han mostrado únicamente las columnas de R^T implicadas, para evitar la permutación Π_k realmente se trabaja con el esqueleto completo y habrá que poner a `NaN` los ceros señalados con negrita. En MATLAB puede comprobarse el

resultado obtenido haciendo `full(R')` tras `[Q,R]=qr(sparse([1 0 0 0 0 2; 1 0 -1 0 0 0]))`.

Es interesante comparar con el otro algoritmo de actualización por supresión. Partiremos de $A_3 = (\Pi_3^T R_3^T)Q_3^T$:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & -1 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{6} & -1/\sqrt{3} \\ 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix}^T$$

Dado que queremos eliminar a_7 primero habrá que intercambiar las filas primera y segunda de A_3^T y de Q_3:

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{bmatrix} \begin{bmatrix} 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{6} & -1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix}^T$$

Utilizando rotaciones de Givens, primero postmultiplicamos las columnas 2 y 3 por $\begin{bmatrix} \sqrt{6}/3 & 1/\sqrt{3} \\ -1/\sqrt{3} & \sqrt{6}/3 \end{bmatrix}$ y luego las columas 1 y 2 por $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$:

$$\begin{bmatrix} q^T \\ \Pi_3^T R_3^T \end{bmatrix} G(2,3)G(1,2) = \left[\begin{array}{ccc} 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ \hline \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{array}\right] G(2,3)G(1,2) =$$

$$= \left[\begin{array}{ccc} 0 & 1 & 0 \\ \hline \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -1 & -1/\sqrt{2} \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ \sqrt{2} & 0 & -\sqrt{2} \end{array}\right] G(1,2) = \left[\begin{array}{c|cc} -1 & 0 & 0 \\ \hline 0 & \sqrt{2} & 0 \\ 0 & 0 & 0 \\ 1 & -1/\sqrt{2} & -1/\sqrt{2} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \sqrt{2} & -\sqrt{2} \end{array}\right]$$

Nótese que tras la primera rotación apareció un rellenado no contemplado en el esqueleto estático. El nuevo factor ortogonal se obtiene de

$$\begin{bmatrix} 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{6} & -1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{6}/3 & 1/\sqrt{3} \\ 0 & -1/\sqrt{3} & \sqrt{6}/3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= Q_3 G(2,3) G(1,2) = \begin{bmatrix} \alpha & O^T \\ O & Q_4 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

6. CONCLUSIONES Y TRABAJO FUTURO

Hemos descrito cómo utilizar una factorización ortogonal dispersa actualizable explícita para resolver la sucesión de sistemas compatibles dispersos que se requiere para implementar un método non-simplex de programación lineal. Para ello hemos adaptado a matrices rectangulares ($m_k < n$) las técnicas de (Saunders, 1972) para matrices cuadradas ($m_k = n$) utilizando la estructura estática de datos de (George & Heath, 1980; Heath, 1982) y permitiendo supresión de filas sobre ella.

Estamos desarrollando una implementación sobre el paquete disperso de MATLAB 5, puesto que permite aprovechar el hecho de que la estructura sea estática (como indica (Davis & Hager, 1999, §7.1), en MATLAB todo cambio en el esqueleto de una matriz dispersa conlleva hacer una copia nueva de la matriz completa); dicha estructura se calcula aprovechando que A_k va a ser un subconjunto de las columnas de A, eligiendo una permutación P previa de las filas de A tal que el factor de Cholesky de PAA^TP^T sea disperso. En lugar de superponer una estructura de datos sobre el factor triangular (como se hace en (Vempati, Slutsker & Tinney, 1991)) marcamos con `NaN` los elementos no usados y los convertimos en ceros justo antes de operar, recuperando el esqueleto tras la operación en aquellos elementos donde aún no se produjo el llenado; esto conlleva cierta sobrecarga de trabajo que podría evitarse si esto se contemplara a bajo nivel. Aunque el orden de las columnas de A_k no influye en la densidad del factor triangular, sacamos partido de la subordenación inducida sobre una ordenación a "priori" de las columnas de A a la hora de refactorizar.

Como trabajo futuro tenemos la descripción en forma producto de dicha factorización ortogonal; está basada en la factorización LDV (Gill, Murray & Saunders, 1975) de la matriz

$$B_k \doteq \begin{bmatrix} A_1 & O \\ A_2 & I \end{bmatrix} = \begin{bmatrix} L_1 & O \\ L_2 & I \end{bmatrix} \begin{bmatrix} D_1 & O \\ O & I \end{bmatrix} \begin{bmatrix} V_1 & O \\ O & I \end{bmatrix}$$

puesto que la matriz L es la matriz que implícitamente se usa al resolver sistemas con matriz L_1 (ver (Heath, 1982):229)

> *It is unnecesary to extract R_1 from the data structure or write a special back substitution routine to skip over the zero rows, since the same effect may be obtained by simply setting the "zero" diagonal elements equal to 1 and noting that the corresponding components of the right-hand sides will already be zero.*

y dicha factorización LDV no requiere raíces cuadradas, es actualizable por adición y supresión de columnas y además el factor ortogonal es gratis (ver (Gill, Murray & Saunders, 1975):1055)

> *The important feature of the matrices L_j and V_j is that both can be constructed from a pair of vectors.*

lo cual nos permite utilizar la formulación dada en las secciones anteriores disponiendo del factor ortogonal.

Actualmente sólo contemplamos problemas en los que podemos elegir una permutación P previa de las filas de A tal que el factor de Cholesky de PAA^TP^T sea disperso. Cuando esto no sea así (e.g., cuando A tenga columnas densas) es más conveniente utilizar técnicas dinámicas como en (Davis & Hager, 1999), bien directamente trabajando sobre $B_kB_k^T$, bien adaptándolas al caso semidefinido positivo para trabajar con $A_kA_k^T$ o bien regularizando el problema para trabajar con $\sigma I_n + A_kA_k^T$.

El postprocesado de R para conseguir que L_1 sea bien condicionada también puede ser objeto de investigación futura.

7. REFERENCIAS

Adlers, M. (1998). *Sparse least squares problems with box constraints.* Licenciat thesis, Department of Mathematics, Linköping University.

Björck, A. (1996). *Numerical Methods for Least Squares Problems.* SIAM.

Coleman, T.F; Edenbrandt, A; Gilbert, J.R. (1986). *Predicting fill for sparse orthogonal factorization.* Journal of the Association of Computing Machinery 33:517–532.

Davis, T; Hager, W. (1999). *Modifying a sparse Cholesky factorization.* SIAM Journal on Matrix Analysis and Applications 20:606–627.

Fletcher, R. (1987). *Practical methods of optimization*, segunda edición; John Wiley and Sons.

George, A; Heath, M.T. (1980). *Solution of sparse linear least squares problems using Givens rotations.* Linear Algebra and Its Applications 34:69–83.

George, A; Heath, M.T; Ng, E. (1984). *Solution of sparse underdetermined systems of linear equations.* SIAM Journal on Scientific and Statistical Computing 5(4):988-997.

George, A; Liu, J; Ng, E. (1988). *A data structure for sparse QR and LU factorizations.* SIAM Journal on Scientific and Statistical Computing 9(1):100–121.

Gilbert, J.R; Peierls, T. (1988). *Sparse partial pivoting in time proportional to arithmetic operations.* SIAM Journal on Scientific and Statistical Computing 9(5):862–874.

Gill, P.E; Murray, W. (1973). *A numerically stable form of the simplex algorithm.* Linear Algebra and its Applications 7:99–138.

Gill, P.E; Murray, W; Saunders, M.A. (1975). *Methods for computing and modifying the LDV factors of a matrix.* Mathematics of Computations 29:1051–1077.

Gill, P.E; Murray, W. (1977). *Linearly-constrained problems including linear and quadratic programming.* En (Jacobs, 1977):313–363.

Gill, P.E; Murray, W; Wright, M.H. (1991). *Numerical Linear Algebra and Optimization, Vol. 1.* Addison-Wesley.

Golub, G.H; Van Loan, C.L. (1996). *Matrix computations.* North Oxford Academic Publishing.

Hager, W.W; Shih, C.-L; Lundin, E.O. (1996). *Active set strategies and the LP dual active set algorithm.* Technical report, Department of Mathematics, University of Florida.

Heath, M.T. (1982). *Some extensions of an algorithm for sparse linear least squares problems.* SIAM Journal on Scientific and Statistical Computing 3(2):223–237.

Heath, M.T. (1984). *Numerical methods for large sparse linear least squares problems.* SIAM Journal on Scientific and Statistical Computing 5(3):497–513.

Higham, N.J. (1996). *Accuracy and stability of numerical algorithms.* SIAM.

Jacobs, D.A.H. (editor) (1977). *The state of the art in numerical analysis.* Academic Press.

Kumar, S. (editor) (1991). *Recent developments in mathematical programming.* Gordon and Breach.

Maros, I; Mészáros, C. (1995). *A numerically exact implementation of the simplex method.* Annals of Operations Research 58:3–17.

Matstoms, P. (1994). *Sparse QR factorization with applications to linear least squares problems.* PhD thesis, Department of Mathematics, Linköping University.

Mitra, G; Tamiz, M. (1991). *Alternative methods for representing the inverse of linear programming basis matrices.* En (Kumar, 1991):273-302.

Murtagh, B.A. (1981). *Advanced linear programming: computation and practice.* McGraw-Hill.

Santos-Palomo, A. (1998). *The* Sagitta *method for solving linear programs.* Enviado para su publicación.

Santos-Palomo, A; Guerrero-García, P. (1998). *A feasible-point* Sagitta *approach in linear programming.* En las actas de Optimization 98, Coimbra. También enviado para su publicación.

Santos-Palomo, A; Guerrero-García, P. (1998). *The* Sagitta *method for solving linear programs as a feasible-point method.* En las actas de OR 98 International Conference on Operations Research, Zürich.

Saunders, M.A. (1972). *Large-scale linear programming using the Cholesky factorization.* Technical Report CS-252, Computer Science Department, Stanford University.

Stojković, N.V; Stanimirović, P.S. (2001). *Two direct methods in linear programming.* European Journal of Operational Research 131:417–439.

Vempati, N; Slutsker, I.W; Tinney, W.F. (1991). *Enhacements to Givens rotations for power system state estimation.* IEEE Transactions on Power Systems 6(2):842–849.