DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No: DSIC-II/05/06 **Pages: 28**

Title: Cálculo de Funciones de Matrices mediante la Descomposición Real de Schur de una Matriz.

Author (s): Vicente Hernández García y J. Javier Ibáñez González.

Date: 27/10/2006

Key Words: Ecuación Conmutante, Diagonalización a Bloques de una Matriz, Algoritmos Orientados a Columnas, Filas o Diagonales.

VºBº **Author (s):**

Leader of Research Group

Vicente Hernández García

Resumen

La necesidad de calcular funciones de matrices aparece en un gran número de aplicaciones, constituyendo una herramienta básica en el diseño de sistemas de control, problemas de convección difusión, estudio de fluidos, etc. En algunas de estas aplicaciones se requiere un tiempo de cálculo muy elevado, al aparecer en la resolución del problema matrices de gran dimensión, y en otras es necesario realizar muchos cálculos por unidad de tiempo para, por ejemplo, poder interactuar sobre un sistema físico en tiempo real.

Para la resolución rápida de estos problemas, resulta muy conveniente diseñar algoritmos eficientes y precisos, entre los que cabe destacar los basados en la forma real de Schur. Los algoritmos implementados en este informe técnico son:

- Algoritmo basado en la diagonalización por bloques de una matriz.
- Algoritmos basados en la resolución de la ecuación conmutante orientados a columnas, a diagonales y a bloques.
- Algoritmo basado en la resolución de la ecuación conmutante con agrupación de valores propios cercanos.
- Algoritmo basado en los aproximantes de Padé.

En cuanto al contenido del informe se comienza con una introducción en la que se define la forma real de Schur de una matriz y su aplicación al cálculo de funciones de matrices. En la segunda sección se describen los algoritmos referenciados anteriormente. En la tercera sección se describe el hardware utilizado en las pruebas y una descripción de los detalles de la implementación, mostrándose los resultados obtenidos. En la última sección se realizan las conclusiones de este informe técnico.

1 Introducción

La descomposición real de Schur de una matriz $A \in \Re^{nxn}$, consiste en encontrar una matriz ortogonal Q de orden n y una matriz casi triangular superior S de orden n, denominada forma real de Schur de A, tal que

$$A = QSQ^T = Q \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1m} \\ 0 & S_{22} & \cdots & S_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & S_{mm} \end{bmatrix} Q^T , \quad (1)$$

donde cada bloque diagonal S_{ii}, $i = 1, \cdots, m$, es de tamaño $1x1$ ó $2x2$. Los bloques diagonales de tamaño $1x1$ corresponden a valores propios reales, y los que son $2x2$ a un par de valores propios complejos conjugados de la matriz A.

Sea $f(z)$ una función analítica definida en un abierto del plano complejo que contiene al espectro de A, definida como

$$f(z) = \sum_{k=0}^{\infty} a_k z^k .$$

Conocida la descomposición real de Schur de la matriz A, entonces $f(A)$ se puede obtener fácilmente a partir de $f(S)$, como lo demuestran las siguientes igualdades

$$f(A) = \sum_{k=0}^{\infty} a_k A^k = \sum_{k=0}^{\infty} a_k \left(QSQ^T\right)^k = \sum_{k=0}^{\infty} a_k QS^k Q^t = Q\left(\sum_{k=0}^{\infty} a_k S^k\right)Q^T = Qf(S)Q^T .$$

El problema ahora es el cálculo de $f(S)$; es decir, el cálculo de la función de una matriz casi triangular superior. Diversos métodos se pueden utilizar para este cálculo ([Iba97], [TechRepPade], [High03]), y algunos otros que se introducen en este informe técnico.

El algoritmo completo para el cálculo de $f(A)$ es el siguiente.

Algoritmo 1: Algoritmo general para el cálculo de funciones de matrices, basado en la forma real de Schur de una matriz.

Entrada: Matriz $A \in \Re^{nxn}$.

Salida: Matriz $B = f(A) \in \Re^{nxn}$.

---

1 Obtener las matrices Q y S de la descomposición real de Schur de la matriz A.

2 Calcular $F = f(S)$.

3 Calcular $B = QFQ^T$.

El coste por etapas del Algoritmo 1 es:

- Primera etapa: El cálculo de la forma real de Schur está basado en el algoritmo iterativo QR de coste estimado de $25n^3$ flops (véase [BlDo99]), siendo n el orden de la matriz A. Este valor se ha determinado de forma heurística, puesto que al ser un método iterativo el número de flops puede variar, dependiendo de la velocidad de convergencia de la forma Hessenberg superior a la forma casi triangular superior de la matriz A.
- Segunda etapa : El coste computacional de $F = f(S)$ depende del método usado para su cálculo.
- Tercera etapa: Para obtener B es necesario realizar el producto de una matriz densa por una matriz casi triangular superior, $B = QF$, y el producto de dos matrices densas, $B = BQ^T$, por lo que el número necesario de flops para esos cálculos es aproximadamente igual a $3n^3$ flops.

2 Algoritmo Basado en la Reducción de una Matriz Casi Triangular Superior a la forma Diagonal por Bloques

Una primera estrategia para calcular $f(S)$ es reducir la matriz S a una matriz diagonal por bloques, D, utilizando para ello una transformación no ortogonal de semejanza definida por una matriz invertible $Y \in \Re^{nxn}$ bien condicionada, de manera que los valores propios cercanos se encuentren en el mismo bloque diagonal, y tal que la separación entre bloques diagonales distintos sea lo suficientemente grande. De esta forma, se tiene que

(2) $S = YDY^{-1},\ D = \text{diag}\left(D_1, D_2, \cdots, D_p\right),\ D_i \in \Re^{n_i x n_i},\ 1 \le i \le p,\ \sum_{i=1}^{p} n_i = n,$

y entonces $f(S)$ se puede calcular mediante la expresión

$$f(S) = Yf(D)Y^{-1} = Y\text{diag}\left(f(D_1), \cdots, f(D_p)\right)Y^{-1},$$

donde cada bloque diagonal $f(D_i)$, $i = 1,2,\cdots,p$, se puede obtener utilizando, por ejemplo, el algoritmo de Padé [TechReportPade].

Si la matriz S se puede dividir en bloques,

(3) $$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1p} \\ 0 & S_{22} & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & S_{pp} \end{bmatrix},\ S_{ij} \in \Re^{n_i x n_j},\ 1 \le i \le j \le p,\ \sum_{i=1}^{p} n_i = n,$$

de manera que los conjuntos de valores propios de los bloques diagonales, $\lambda(S_{11}), \lambda(S_{22}), \cdots, \lambda(S_{pp})$, sean disjuntos dos a dos, entonces existe una transformación de semejanza, definida mediante una matriz invertible $Y \in \Re^{nxn}$, de manera que

(4) $Y^{-1}SY = diag\left(S_{11}, S_{22}, \cdots, S_{pp}\right)$ ([GoVa96], Teorema 7.1.6).

La forma de obtener la matriz Y se describe a continuación.

En primer lugar, la matriz identidad de orden n se divide en bloques $I_n = \left[E_1, E_2, \cdots, E_p\right],\ E_i \in \Re^{nxn_i},\ i = 1, \cdots, p$, y se definen las matrices $Y_{ij} \in \Re^{nxn}$, como

$$Y_{ij} = I_n + E_i Z_{ij} E_j^T,\ 1 \le i < j \le p,$$

siendo $Z_{ij} \in \Re^{n_i x n_j}$ matrices a determinar.

Si se fijan los valores de i y de j, entonces la matriz Y_{ij} coincide con la matriz identidad excepto que su bloque (i, j) es igual a Z_{ij}; es decir,

$$Y_{ij} = \begin{bmatrix} I_{n_1} & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & I_{n_i} & \cdots & Z_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & I_{n_j} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & I_{n_p} \end{bmatrix}.$$

Si se define $\overline{S} = Y_{ij}^{-1} S Y_{ij}$, entonces se tiene que $Y_{ij}\overline{S} = SY_{ij}$; es decir

$$\begin{bmatrix} I_{n_1} & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & I_{n_i} & \cdots & Z_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & I_{n_j} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & I_{n_p} \end{bmatrix} \begin{bmatrix} \overline{S}_{11} & \cdots & \overline{S}_{1i} & \cdots & \overline{S}_{1j} & \cdots & \overline{S}_{1p} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \overline{S}_{ii} & \cdots & \overline{S}_{ij} & \cdots & \overline{S}_{ip} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \overline{S}_{jj} & \cdots & \overline{S}_{jp} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & \overline{S}_{pp} \end{bmatrix}$$

$$= \begin{bmatrix} S_{11} & \cdots & S_{1i} & \cdots & S_{1j} & \cdots & S_{1p} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & S_{ii} & \cdots & S_{ij} & \cdots & S_{ip} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & S_{jj} & \cdots & S_{jp} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & S_{pp} \end{bmatrix} \begin{bmatrix} I_{n_1} & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & I_{n_i} & \cdots & Z_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & I_{n_j} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & I_{n_p} \end{bmatrix}.$$

Por tanto, los bloques de las matrices S y $\overline{S}$, se relacionan entre sí mediante las siguientes igualdades.

- Bloques diagonales.

$$\overline{S}_{kk} = S_{kk},\ k = 1,2,\cdots,p.$$

- Bloques no diagonales.

$$\overline{S}_{ij} = S_{ii}Z_{ij} - Z_{ij}S_{jj} + S_{ij},\ 1 \le i < j \le p$$

$$\overline{S}_{kj} = S_{ki}Z_{ij} + S_{kj},\ k = 1,2,\cdots,i-1,$$

$$\overline{S}_{ik} = S_{ik} - Z_{ij}S_{jk},\ k = j+1, j+2,\cdots,p.$$

De este modo, el bloque $\overline{S}_{ij}$ se puede anular si Z_{ij} es solución de la ecuación de Sylvester

$$S_{ii}Z_{ij} - Z_{ij}S_{jj} = -S_{ij},$$

donde $S_{ii} \in \Re^{n_i x n_i}$ y $S_{jj} \in \Re^{n_j x n_j}$ son matrices casi triangulares superiores y $\overline{S}_{ij} \in \Re^{n_i x n_j}$.

Por lo tanto, definiendo la matriz Y como

$$Y = \prod_{i=1}^{p}\prod_{j>i}^{p} Y_{ij} = \begin{bmatrix} I_{n_1} & Z_{12} & \cdots & Z_{1p} \\ 0 & I_{n_2} & \cdots & Z_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_{n_p} \end{bmatrix},$$

se puede comprobar que $Y^{-1}SY = diag\left(S_{11}, S_{22}, \cdots, S_{pp}\right)$.

A continuación se detalla el algoritmo que permite obtener la transformación de semejanza definida por la matriz Y.

Algoritmo 2: Reducción a la forma diagonal por bloques de una matriz casi triangular superior.

Entradas: Matriz $S \in \Re^{nxn}$ casi triangular superior y matriz $Q \in \Re^{nxn}$ ortogonal, procedentes de la descomposición real de Schur de la matriz $A \in \Re^{nxn}$.

Salidas: Q es sobrescrita por la matriz QY y S es sobrescrita por la matriz $diag(S_{11}, S_{22}, \cdots, S_{pp})$, de manera que $Q^{-1}SQ = diag(S_{11}, S_{22}, \cdots, S_{pp})$.

1 Para $j = 2 : p$

1.1 Para $i = 1 : j-1$

1.1.1 Calcular Z_{ij} resolviendo la ecuación de Sylvester

$$S_{ii}Z_{ij} - Z_{ij}S_{jj} = -S_{ij}$$

1.1.2 Para $k = j+1 : p$

1.1.2.1 $S_{ik} = S_{ik} - ZS_{jk}$

1.1.2.2 Para $k = 1 : p$

1.1.2.2.1 $Q_{kj} = Q_{ki}Z + Q_{kj}$

El número de flops de este algoritmo depende de los tamaños de los bloques diagonales de la matriz S.

La elección de la división en bloques de la matriz S determina la sensibilidad de las ecuaciones de Sylvester que aparecen en el Algoritmo 2. Esto afecta al número de condición de la matriz Y y a la estabilidad numérica del proceso de diagonalización por bloques de la matriz S. Esto es debido a que el error relativo de la solución $\tilde{Z}$ obtenida al resolver la ecuación de Sylvester

(5) $$S_{ii}Z - ZS_{jj} = -S_{ij},$$

satisface

(6) $$\frac{\left\|Z - \tilde{Z}\right\|_F}{\left\|Z\right\|_F} \cong u \frac{\left\|S\right\|_F}{sep(S_{ii}, S_{jj})}$$ ([GoNV79]),

siendo

$$sep(S_{ii}, S_{jj}) = \min_{X \neq 0} \frac{\left\|S_{ii}X - XS_{jj}\right\|_F}{\left\|X\right\|_F}$$ ([Varah79]),

la separación que existe entre los bloques diagonales S_{ii} y S_{jj}, y u el error de redondeo unidad.

Si los conjuntos $\lambda(S_{ii})$ y $\lambda(S_{jj})$ están muy cercanos, entonces se pueden producir grandes errores en la obtención de la diagonalización, puesto que al cumplirse que

$$sep(S_{ii}, S_{jj}]) = \min_{X \neq 0} \frac{\|S_{ii}X - XS_{jj}\|_F}{\|X\|_F} \leq \min_{\substack{\lambda \in \lambda(T_{ii}) \\ \mu \in \lambda(T_{jj})}} |\lambda - \mu|,$$

se tiene que $sep(S_{ii}, S_{jj})$ puede ser pequeño, por lo que si Z_{ij} es la solución de la ecuación (5), entonces Z_{ij} tendrá una gran norma, con lo que la matriz Y estará mal condicionada por ser el producto de matrices de la forma

$$Y_{ij} = \begin{bmatrix} I & Z_{ij} \\ 0 & I \end{bmatrix}.$$

Puesto que en la forma real de Schur de una matriz A, los valores propios cercanos no están necesariamente en el mismo bloque diagonal, es necesario reordenar los bloques diagonales 1x1 o 2x2 mediante transformaciones de semejanza ([NgPa87]), de manera que los valores propios cercanos estén agrupados. De este modo, la separación entre bloques será suficientemente grande, con la consiguiente disminución del error cometido.

El algoritmo para el cálculo de funciones de matrices que se presenta a continuación, está basado en la diagonalización por bloques de una matriz triangular superior por bloques.

Algoritmo 3: Algoritmo para el cálculo de funciones de matrices, basado en la forma real de Schur y en la forma diagonal por bloques de una matriz casi triangular superior.

Entrada: Matriz $A \in \Re^{nxn}$.

Salida: Matriz $B = f(A) \in \Re^{nxn}$.

---

1 Obtener las matrices Q y S de la descomposición real de Schur (1) de la matriz A.

2 Aplicar el Algoritmo 2 sobre las matrices Q y S, para obtener las matrices Q (Q es sobrescrita por QY) y $\text{diag}(S_{11}, \cdots, S_{pp})$.

3 Calcular $F = \text{diag}(f(S_{11}), \cdots, f(S_{pp}))$.

4 Calcular $B = QFQ^{-1}$.

---

3 Algoritmos Basados en la Resolución de la Ecuación Conmutante

Sea $A \in \Re^{nxn}$ y $f(z)$ una función analítica definida en un abierto que contiene al espectro de A,

$$f(z) = \sum_{k=0}^{\infty} a_k z^k,$$

Si $B = f(A) \in \Re^{nxn}$, entonces

$$AB = Af(A) = A\left(\sum_{k=0}^{\infty} a_k A^k\right) = \sum_{k=0}^{\infty} a_k A^{k+1} = \left(\sum_{k=0}^{\infty} a_k A^k\right) A = f(A)A = BA,$$

por lo que B verifica la denominada ecuación conmutante

(7) $$AB = BA.$$

En esta sección se describen varios algoritmos basados en la forma real de Schur y en la ecuación conmutante. Dependiendo de si la forma real de Schur tiene valores propios cercanos o no, se pueden utilizar dos estrategias.

a) Sin agrupación de valores propios.

Sea

$$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1m} \\ 0 & S_{22} & \cdots & S_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & S_{mm} \end{bmatrix} \in \Re^{nxn},\ S_{ii} \in \Re \text{ ó } S_{ii} \in \Re^{2x2},\ i = 1,2,\cdots,m,$$

la forma real de Schur de la matriz A y

$$F = f(S) = \begin{bmatrix} F_{11} & F_{12} & \cdots & F_{1r} \\ 0 & F_{22} & \cdots & F_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F_{rr} \end{bmatrix} \in \Re^{nxn},\ F_{ii} \in \Re \text{ ó } F_{ii} \in \Re^{2x2},\ i = 1,2,\cdots,m,$$

con igual estructura por bloques que la matriz S.

Según la forma de obtener los bloques diagonales F_{ii}, se pueden diseñar los siguientes algoritmos:

- Algoritmo orientado a columnas: De izquierda a derecha, se obtienen sucesivamente las columnas de bloques de la matriz F. Para cada columna, se calcula en primer lugar el bloque diagonal F_{jj}, utilizando la expresión

(8) $$F_{jj} = f\left(S_{jj}\right),$$

  continuando con los bloques que se encuentran por encima de él, resolviendo para ello ecuaciones de Sylvester.

- Algoritmo orientado a filas: De abajo hacia arriba, se obtienen sucesivamente las filas de bloques de la matriz F. Para cada fila de bloques, se calcula en primer lugar el bloque diagonal F_{jj}, utilizando la expresión (8), continuando con los bloques que se encuentran a la derecha de él, resolviendo para ello ecuaciones de Sylvester.

- Algoritmo orientado a diagonales: En primer lugar, se determinan los bloques diagonales F_{jj} utilizando la expresión (8), calculando a continuación las superdiagonales de bloques de abajo hacia arriba, resolviendo para ello ecuaciones de Sylvester.

- Algoritmos orientados a bloques: A partir de un tamaño de bloque tb, las matrices S y F se dividen en sxs bloques de tamaño tb o $tb+1$ (salvo los bloques de filas y columnas de índice m), teniendo en cuenta que los valores

propios conjugados no se pueden encontrar en bloques distintos. A continuación se obtienen los bloques diagonales F_{jj}, bien calculándolos a partir de la expresión (8), utilizando para ello otros algoritmos para el cálculo de funciones de matrices, por ejemplo mediante los aproximantes de Padé, o bien resolviendo la ecuación conmutante

$$F_{jj}S_{jj} = S_{jj}F_{jj},$$

mediante cualquiera de los métodos descritos anteriormente. Según la forma de calcular los bloques de la matriz F, se pueden utilizar tres algoritmos: algoritmo orientado a columnas de bloques, algoritmo orientado a filas de bloques y algoritmo orientado a diagonales de bloques.

b) Con agrupación de valores propios.

En aquellas ocasiones en que la matriz S tiene valores propios cercanos entre sí, es conveniente agruparlos de manera que la resolución de la ecuación conmutante se realice con precisión. Para ello se deben realizar reordenaciones en la matriz S, utilizando transformaciones ortogonales de semejanza, de manera que los valores propios cercanos se encuentren en un mismo bloque diagonal. De este modo, la matriz S se puede estructurar por bloques, como se muestra en la siguiente expresión

$$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1p} \\ 0 & S_{22} & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & S_{pp} \end{bmatrix},\ S_{ij} \in \Re^{n_i x n_j},\ 1 \le i \le j \le p,\ \sum_{i=1}^{p} n_i = n, \qquad (9)$$

de manera los bloques diagonales S_{jj}, se encuentren suficientemente separados entre sí.

Estas agrupaciones se deben realizar teniendo en cuenta, además, que el maximizar la separación entre los bloques diagonales puede conducir a obtener grandes bloques con valores propios cercanos, con el consiguiente aumento de las reordenaciones de los valores propios sobre la diagonal de S.

Una vez se ha realizado la división en bloques de la matriz S, los bloques diagonales F_{jj} se pueden calcular utilizando, por ejemplo, algoritmos basados en la aproximación diagonal de Padé.

Según el orden utilizado en el cálculo de bloques, se pueden obtener tres algoritmos: algoritmo orientado a columnas de bloques, algoritmo orientado a filas de bloques o algoritmo orientado a diagonales de bloques.

Aunque los algoritmos orientados a filas y a columnas tienen una formulación similar, la forma de acceder a los datos es distinta. Como se puede ver en [Ibañ97], si se utiliza el lenguaje de programación Fortran, se obtienen mejores prestaciones al utilizar un algoritmo orientado a columnas, y si el lenguaje utilizado es C las mejores prestaciones se obtienen en los algoritmos orientados a filas. En este capítulo se describen únicamente los algoritmos orientados a columnas, puesto que en las implementaciones se ha utilizado el lenguaje Fortran.

En cuanto a la precisión de los algoritmos, se puede demostrar que algoritmos sin reordenación de bloques diagonales son estables siempre y cuando los valores propios de la matriz A estén suficientemente separados entre sí.

Los algoritmos orientados a bloques, como se verá en la sección de implementación, resultan ser más eficientes que los orientados a filas o columnas, pues hacen uso de la caché de los ordenadores; por lo que, si se elige convenientemente el tamaño de bloque, se produce una mayor localidad de datos y por tanto, un menor intercambio de datos entre la memoria principal y la memoria secundaria.

La ventaja de los algoritmos con agrupación de valores propios próximos, es que se pueden utilizar para cualquier matriz casi triangular superior, incluso si la matriz tiene valores propios muy próximos entre sí.

Para determinar los bloques diagonales en los algoritmos orientados a bloques, es necesario el cálculo de funciones de matrices de orden mayor que 2, utilizando para ello otro tipo de métodos, como los descritos en [TechRepPade].

En los algoritmos orientados a columnas o a filas es necesario determinar los bloques diagonales de orden $2x2$ de la matriz F, correspondientes a valores propios conjugados de la matriz S. A continuación se describe un método descrito en [Ibañ97] que permite calcular la matriz $F_{jj} = f(S_{jj})$, siendo S_{jj} un bloque diagonal de orden $2x2$.

Sea

$$S_{jj} = \begin{bmatrix} s_{k,k} & s_{k,k+1} \\ s_{k+1,k} & s_{k+1,k+1} \end{bmatrix},$$

y

$$F_{jj} = \begin{bmatrix} f_{k,k} & f_{k,k+1} \\ f_{k+1,k} & f_{k+1,k+1} \end{bmatrix} = f\left(\begin{bmatrix} s_{k,k} & s_{k,k+1} \\ s_{k+1,k} & s_{k+1,k+1} \end{bmatrix} \right)$$

el correspondiente bloque de la matriz F.

Sin pérdida de generalidad, se puede suponer que los bloques diagonales $2x2$ tienen los elementos diagonales iguales y los no diagonales de igual valor absoluto pero con distinto signo (ver rutina _gees de LAPACK); es decir,

$$s_{k+1,k+1} = s_{k,k}\,,\ |\, s_{k+1,k} \,|=|\, s_{k,k+1} \,|\,,\ s_{k+1,k} s_{k,k+1} < 0\ .$$

Llamando

$$a \equiv s_{k,k}\,,\ b \equiv s_{k,k+1}\ \text{y}\ c \equiv s_{k+1,k}\,,$$

y considerando la descomposición en valores propios

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix} = \begin{bmatrix} \sqrt{bc} & b \\ c & -\sqrt{bc} \end{bmatrix}^{-1} \begin{bmatrix} z & 0 \\ 0 & \bar{z} \end{bmatrix} \begin{bmatrix} \sqrt{bc} & b \\ c & -\sqrt{bc} \end{bmatrix},$$

siendo $z = a + \sqrt{bc}$ y $\bar{z}$ el complejo conjugado de z, entonces

$$\begin{bmatrix} f_{k,k} & f_{k,k+1} \\ f_{k+1,k} & f_{k+1,k+1} \end{bmatrix} = f\left(\begin{bmatrix} a & b \\ c & a \end{bmatrix} \right) = \begin{bmatrix} \sqrt{bc} & b \\ c & -\sqrt{bc} \end{bmatrix}^{-1} f\left(\begin{bmatrix} z & 0 \\ 0 & \bar{z} \end{bmatrix} \right) \begin{bmatrix} \sqrt{bc} & b \\ c & -\sqrt{bc} \end{bmatrix}$$

y

$$f\left(\begin{bmatrix} z & 0 \\ 0 & \bar{z} \end{bmatrix} \right) = \begin{bmatrix} f(z) & 0 \\ 0 & f(\bar{z}) \end{bmatrix} = \begin{bmatrix} f(z) & 0 \\ 0 & \overline{f(z)} \end{bmatrix}.$$

Luego

$$\begin{bmatrix} f_{k,k} & f_{k,k+1} \\ f_{k+1,k} & f_{k+1,k+1} \end{bmatrix} = \frac{1}{bc}\begin{bmatrix} \sqrt{bc} & b \\ c & -\sqrt{bc} \end{bmatrix}\begin{bmatrix} f(z) & 0 \\ 0 & \overline{f(z)} \end{bmatrix}\begin{bmatrix} \sqrt{bc} & b \\ c & -\sqrt{bc} \end{bmatrix},$$

(10)
$$\begin{bmatrix} f_{k,k} & f_{k,k+1} \\ f_{k+1,k} & f_{k+1,k+1} \end{bmatrix} = \begin{bmatrix} \mathrm{Re}(f(z)) & -\frac{1}{c}\sqrt{|bc|}\,\mathrm{Im}(f(z)) \\ -\frac{1}{b}\sqrt{|bc|}\,\mathrm{Im}(f(z)) & \mathrm{Re}(f(z)) \end{bmatrix}.$$

3.1 Algoritmos Orientados a Columnas y a Diagonales

En este apartado se describen dos algoritmos, uno orientado a columnas y otro orientado a diagonales, que permiten resolver la ecuación conmutante (7). Para ello se considera las matrices F y S divididas en mxm bloques $2x2$ o $1x1$ según sean los tamaños de los bloques diagonales.

Los bloques diagonales F_{jj}, $j = 1,2,\cdots,m$, se pueden calcular mediante la expresión

(11)
$$F_{jj} = f\left(S_{jj}\right).$$

Para el resto de los bloques F_{ij}, $i \neq j$, se tiene que

(12)
$$S_{ii}F_{ij} - F_{ij}S_{jj} = \sum_{k=i}^{j-1} F_{ik}S_{kj} - \sum_{k=i+1}^{j} S_{ik}F_{kj}\ .$$

Para calcular el bloque F_{ij} es necesario haber calculado previamente los bloques que se encuentran a su izquierda y debajo de él, tal como se indica en la siguiente figura.

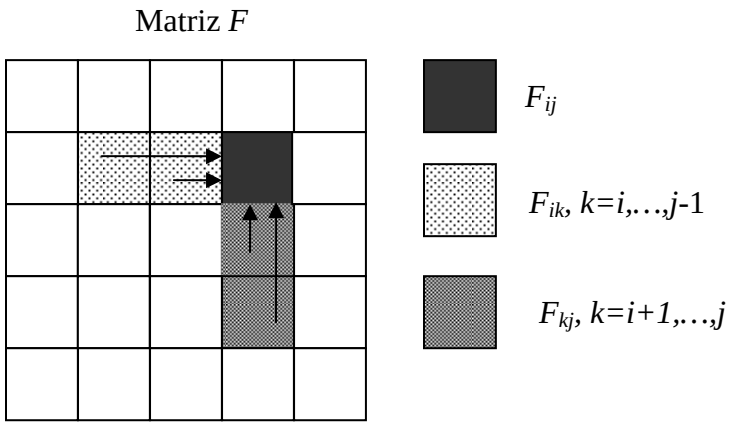

Figura 1.

La ecuación (12) corresponde a una ecuación del tipo Sylvester

$$AX - XB = C\ ,$$

siendo $A = S_{ii}$ y $B = S_{jj}$ matrices casi triangulares superiores y

$$C = \sum_{k=i}^{j-1} F_{ik}S_{kj} - \sum_{k=i+1}^{j} S_{ik}F_{kj}\ .$$

Este tipo de ecuaciones se pueden resolver mediante el algoritmo de Bartels y Stewart ([BaSt71]).

Según el orden utilizado en el cálculo de los bloques de la matriz F , se pueden diseñar diferentes algoritmos, tal como se muestra en los siguientes apartados.

3.1.1 Algoritmo Orientado a Columnas

En este algoritmo los bloques se calculan comenzando con el bloque diagonal de la primera columna, y continuando con los bloques que se encuentran en las sucesivas columnas. Para cada columna, se calcula en primer lugar el bloque diagonal, y a continuación los bloques que se encuentran por encima, en sentido ascendente tal como se indica en la siguiente figura.

Matriz F

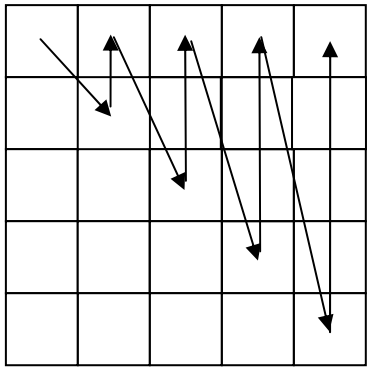

Figura 2.

Los bloques de la diagonal principal se obtienen a partir de la expresión (11), y el resto de los bloques resolviendo la ecuación de Sylvester (12). Para este algoritmo se puede, o bien calcular el miembro de la derecha en el momento en que se va a resolver la ecuación de Sylvester, o bien haberlo calculado previamente. En el algoritmo que se presenta a continuación se ha elegido la primera estrategia.

Algoritmo 4: Algoritmo para el cálculo de la función de una matriz casi triangular superior, basado en la resolución de la ecuación conmutante orientado a columnas.

Entradas: Matriz $S \in \Re^{nxn}$ casi triangular superior.

Salida: Matriz $F = f(S) \in \Re^{nxn}$ casi triangular superior

1 Dividir la matriz S en mxm bloques, de manera que los bloques diagonales sean $1x1$ o $2x2$, correspondientes a valores propios reales o complejos conjugados, respectivamente.

2 Para $j = 1:m$

2.1 $F_{jj} = f(S_{jj})$

2.2 Para $i = j-1:-1:1$

2.2.1 Calcular F_{ij} resolviendo la ecuación de Sylvester

$$S_{ii}F_{ij} - F_{ij}S_{jj} = \sum_{k=i}^{j-1} F_{ik}S_{kj} - \sum_{k=i+1}^{j} S_{ik}F_{kj} .$$

3.1.2 Algoritmo Orientado a Diagonales

En este caso los bloques se calculan por diagonales, comenzando con la diagonal principal y continuando con las diagonales que se encuentran encima de ella, tal como se indica en la siguiente figura. Para cada diagonal, los bloques se calculan de arriba hacia abajo.

Matriz *F*

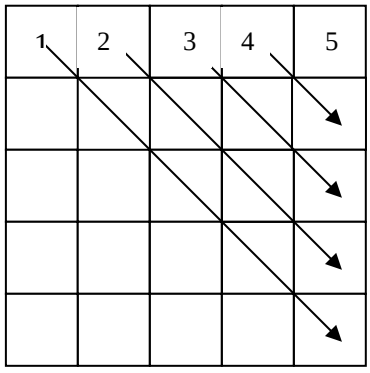

Figura 3.

Los bloques de la diagonal principal se calculan utilizando la expresión (11). Los bloques del resto de las diagonales se calculan mediante la resolución de la ecuación de Sylvester (12). Al igual que en el caso anterior, se puede, o bien calcular el miembro de la derecha en el momento en que se va a resolver la ecuación de Sylvester, o haberlo calculado previamente. En el algoritmo que se presenta a continuación se ha elegido esa segunda estrategia. En la siguiente figura se muestra el bloque que se ha calculado y los bloques que se actualizan a partir de él.

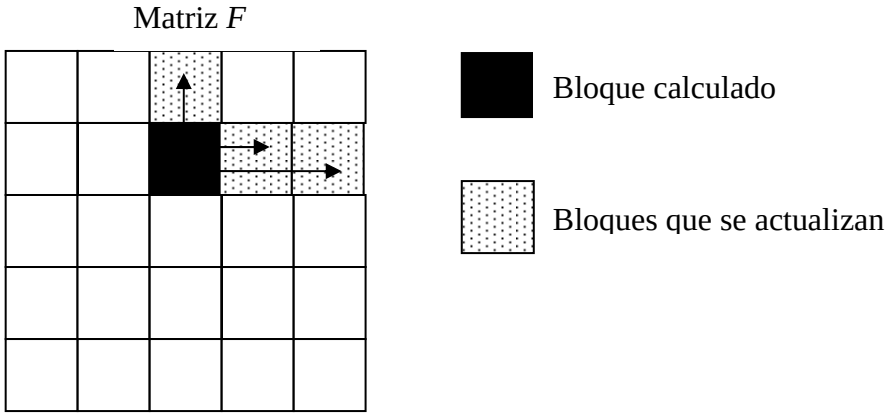

Figura 4.

A continuación se presenta el algoritmo correspondiente.

Algoritmo 5: Algoritmo para el cálculo de la función de una matriz casi triangular superior, basado en la resolución de la ecuación conmutante orientado a diagonales.

Entradas: Matriz $S \in \Re^{nxn}$, casi triangular superior.

Salida: Matriz $F = f(S) \in \Re^{nxn}$, casi triangular superior.

---

1 Dividir la matriz S en mxm bloques, de manera que los bloques diagonales sean $1x1$ o $2x2$, correspondientes a valores propios reales o complejos conjugados, respectivamente.

2 Para $k = 1:m$

2.1 Para $j = 1:m$

2.1.1 $i = j - k + 1$

2.1.2 Si $k > 1$

2.1.2.1 Calcular X resolviendo la ecuación de Sylvester

$$S_{ii}X - XS_{jj} = F_{ij}$$

2.1.2.2 $F_{ij} = X$

2.1.3 Si no

2.1.3.1 $F_{jj} = f(S_{jj})$

2.1.4 Actualizar los bloques $F_{1j}, \cdots, F_{i-1\,j}$

2.1.5 Actualizar los bloques $F_{ij+1}, \cdots, F_{im}$.

3.2 Algoritmos Orientados a Bloques

Sea tb el tamaño de bloque elegido. Debido a la presencia de elementos subdiagonales no nulos en la forma real de Schur de la matriz S, los tamaños de los bloques

diagonales no serán uniformes; es decir, los bloques diagonales de la matriz S podrán tener dimensiones:

- $tbxtb$ si $S_{tb+1,tb} = 0$,
- $(tb+1)x(tb+1)$ si $S_{tb+1,tb} \neq 0$,

salvo el bloque diagonal situado en la última fila y columna de bloques, que puede tener una dimensión distinta, como resultado del encaje final de la división por bloques con la dimensión de la matriz inicial. El resto de bloques de S se obtienen dividiendo la matriz en submatrices rectangulares, de acuerdo a los bloques diagonales; de este modo, la matriz S queda dividida pxp bloques. Para la matriz F se utiliza también la misma estructura de bloques.

Si se identifican los bloques diagonales en la ecuación $FS = SF$, se obtiene

$$F_{jj} = f\left(S_{jj}\right),\ j = 1,2,\cdots, p\,.$$

Esta ecuación puede resolverse utilizando cualquiera de los algoritmos comentados anteriormente (orientados a columnas u orientados a diagonales).

Si se consideran los índices i, j de manera que $i \neq j$, se obtiene la ecuación casi triangular superior de Sylvester

$$S_{ii}F_{ij} - S_{ij}T_{jj} = \sum_{k=i}^{j-1} F_{ik}S_{kj} - \sum_{k=i+1}^{j} S_{ik}F_{kj}\,.$$

Razonando como en el apartado anterior, se pueden deducir varios algoritmos: orientados a fila, orientados a columnas y orientados a diagonales. A continuación se presenta el algoritmo orientado a columnas de bloques.

Algoritmo 6: Algoritmo para el cálculo de la función de una matriz casi triangular superior, basado en la resolución de la ecuación conmutante orientado a columnas de bloques.

Entradas: Matriz $S \in \Re^{nxn}$ casi triangular superior, y tb el tamaño de bloque.

Salida: Matriz $F = f(S) \in \Re^{nxn}$ casi triangular superior.

---

1 Dividir S y F en pxp bloques de tamaños $tbxtb$ o $(tb+1)x(tb+1)$ (salvo los bloques S_{pp} y F_{pp} que tendrán las dimensiones adecuadas).

2 Para $j = 1:p$

2.1 Calcular F_{jj} resolviendo la ecuación conmutante $F_{jj}S_{jj} = S_{jj}F_{jj}$, mediante el Algoritmo 4 o el Algoritmo 5.

2.2 Para $i = j-1:-1:1$

2.2.1 Calcular F_{ij} resolviendo la ecuación de Sylvester

$$S_{ii}F_{ij} - F_{ij}S_{jj} = \sum_{k=i}^{j-1} F_{ik}S_{kj} - \sum_{k=i+1}^{j} S_{ik}F_{kj}\,.$$

El algoritmo orientado a diagonales de bloques se presenta a continuación.

Algoritmo 7: Algoritmo para el cálculo de la función de una matriz casi triangular superior, basado en la resolución de la ecuación conmutante orientado a diagonales de bloques.

Entradas: Matriz $S \in \Re^{nxn}$ casi triangular superior y tb el tamaño de bloque.

Salida: Matriz $F = f(S) \in \Re^{nxn}$ casi triangular superior.

1 Dividir S en pxp bloques de tamaños $tbxtb$ o $(tb+1)x(tb+1)$ (salvo los bloques S_{pp} y F_{pp} que tendrán las dimensiones adecuadas)

2 Para $k = 1 : p$

2.1 Para $j = k : p$

2.1.1 $i = j - k + 1$

2.1.2 Si $k > 1$

2.1.2.1 Calcular X resolviendo la ecuación de Sylvester

$$S_{ii}X - XS_{jj} = F_{ij}$$

2.1.2.2 $F_{ij} = X$

2.1.3 Si no

2.1.3.1 Calcular F_{jj} resolviendo la ecuación conmutante $F_{jj}S_{jj} = S_{jj}F_{jj}$, mediante el Algoritmo 4 o el Algoritmo 5.

2.1.4 Actualizar los bloques $F_{1j}, \cdots, F_{i-1j}$.

2.1.5 Actualizar los bloques $F_{ij+1}, \cdots, F_{ip}$.

3.3 Algoritmo con Agrupación de Valores Propios Cercanos

En esta sección se describe el algoritmo que consiste en agrupar en clusters los valores propios cercanos de la forma real de Schur de una matriz, para después realizar las reordenaciones necesarias, de manera que la matriz obtenida quede dividida en bloques, estando los valores del mismo cluster en el mismo bloque diagonal. A partir de ahí, se pueden utilizar estrategias similares a las utilizadas en los algoritmos a bloques descritos en el apartado 3.2.

El siguiente algoritmo determina un vector g que contiene los índices de comienzo de los bloques diagonales de la forma real de Schur S de la matriz $A \in \Re^{nxn}$.

Algoritmo 8.

Entradas: Vectores *vr* y *vi* de dimensión *n* que contienen, respectivamente, la parte real y la parte imaginaria de los valores propios de la forma real de Schur $S \in \Re^{nxn}$ de la matriz $A \in \Re^{nxn}$.

Salida: Vector *g* de dimensión *ng* que contiene los índices de los bloques 1x1 o 2x2 de la matriz S.

1 $ng = 0$

2 $i_1 = 1$

3 $i_2 = i_1 + 1$

4 Mientras $i_1 \leq n$

  4.1 $ng = ng + 1$

  4.2 $g(ng) = i_1$

  4.3 Si $vi(i_2) \neq 0$

    4.3.1 $i_1 = i_2 + 1$

  4.4 Si no

    4.4.1 $i_1 = i_2$

  4.5 $i_1 = i_2 + 1$

5 Si $i_1 \leq n$

  5.1 $ng = ng + 1$

  5.2 $g(ng) = i_1$.

Una vez se conoce el vector g, se deben determinar los clusters de valores propios. Estas agrupaciones en clusters se realizan de manera que dos valores propios se encuentran en el mismo cluster siempre que disten entre sí un valor menor que un valor de tolerancia predefinido, no necesariamente pequeño. En el siguiente algoritmo se determina un vector *c* de dimensión *ng* de manera que $c(k)$, $1 \leq k \leq ng$, indica a qué cluster pertenece el grupo de valores propios definidos por $g(k)$.

Algoritmo 9.

Entradas: Vectores vr y vi de dimensión n que contienen, respectivamente, la parte real y la parte imaginaria de los valores propios de la matriz casi triangular superior $S \in \Re^{nxn}$, vector g de dimensión ng que contiene los índices de los bloques de la expresión (1) y la tolerancia tol.

Salida: Vector c que contiene los índices de los clusters.

1 $nc = 0$

2 Para $k = 1 : ng$

2.1 Si $c(k) = 0$

2.1.1 $nc = nc + 1$

2.1.2 $c(k) = nc$

2.2 $vk = vr(g(k)) + \text{abs}(vi(g(k)))i$

2.3 Para $j = k + 1 : ng$

2.3.1 $vj = vr(g(j)) + \text{abs}(vi(g(j)))i$

2.3.2 Si $\text{abs}(vk - vj) < tol$

2.3.2.1 $c(j) = c(k)$.

Conocida la división en clusters de los valores propios de la matriz S, se realizan los intercambios necesarios de bloques diagonales, de manera que los valores propios con el mismo índice de cluster, se encuentren en el mismo bloque diagonal. Estos intercambios se realizan utilizando transformaciones de semejanza ortogonales. Al final de este proceso, se obtiene una matriz $\overline{S}$, divida en bloques conforme a la división en clusters de los valores propios de S, y una matriz ortogonal $\overline{Q}$, de manera que $S = \overline{Q}\overline{S}\overline{Q}^T$.

El algoritmo siguiente lleva a cabo el proceso descrito en el párrafo anterior.

Algoritmo 10.

Entradas: Matriz $S \in \Re^{nxn}$ casi triangular superior, matriz $Q \in \Re^{nxn}$ ortogonal, vector g, de dimensión ng, que contiene los índices de los bloques de la matriz S, vector c que contiene los índices de los clusters.

Salidas: Matriz $\overline{S} \in \Re^{nxn}$ casi triangular superior, con los bloques diagonales correspondientes a valores propios en el mismo cluster, matriz $\overline{Q} \in \Re^{nxn}$ ortogonal tal que $S = \overline{Q}\overline{S}\overline{Q}^T$, vector g que contiene los índices de los bloques diagonales correspondientes a los bloques de la matriz $\overline{S}$.

1. $\overline{S} = S$
2. $\overline{Q} = I_n$
3. $cont = true$
4. Mientras $cont$
   - 4.1 $cont = false$
   - 4.2 Para $k = 1 : ng - 1$
     - 4.2.1 Si $c(k) > c(k+1)$
       - 4.2.1.1 Intercambiar los bloques diagonales de la matriz $\overline{S}$ apuntados por $g(k)$ y $g(k+1)$, actualizando adecuadamente la matriz $\overline{Q}$ y el vector $\overline{g}$
       - 4.2.1.1 $caux = c(k)$
       - 4.2.1.2 $c(k) = c(k+1)$
       - 4.2.1.3 $c(k+1) = caux$
       - 4.2.1.4 $cont = true$
5. $k = 1$
6. Para $j = 2 : ng$
   - 6.1 Si $c(j) \neq c(k)$
     - 6.1.1 $k = k + 1$
     - 6.1.2 $g(k) = g(j)$
     - 6.1.3 $c(k) = c(j)$.

Una vez se han determinado $\overline{S}$ y $\overline{Q}$ se aplica un esquema semejante al utilizado en el Algoritmo 6 o en Algoritmo 7.

A continuación se presenta el algoritmo completo.

Algoritmo 11.

Entradas: Matrices $S \in \Re^{nxn}$ y $Q \in \Re^{nxn}$, procedentes de la forma real de Schur de la matriz $A \in \Re^{nxn}$, función $f(z)$, tolerancia tol.

Salida: Matriz $\overline{F} = f(S) \in \Re^{nxn}$ casi triangular superior y matriz $\overline{Q} \in \Re^{nxn}$ matriz ortogonal actualizada.

1 Utilizar el Algoritmo 8, el Algoritmo 9 y Algoritmo 10 para obtener las matrices $\overline{S}$ y $\overline{Q}$, y la división en bloques de la matriz $\overline{S}$.

2 Para $k = 1 : p$

  2.1 Para $j = k : p$

    2.1.1 $i = j - k + 1$

    2.1.2 Si $k > 1$

      2.1.2.1 Calcular X resolviendo la ecuación de Sylvester $\overline{S}_{ii} X - X \overline{S}_{jj} = F_{ij}$

      2.1.2.2 $F_{ij} = X$

    2.1.3 Si no

      2.1.2.1 Calcular $F_{jj} = f(S_{jj})$, mediante, por ejemplo, aproximantes diagonales de Padé

    2.1.4 Actualizar los bloques $F_{1j}, \cdots, F_{i-1\,j}$

    2.1.5 Actualizar los bloques $F_{ij+1}, \cdots, F_{ip}$.

4 Algoritmos Basados en la Forma Real de Schur y en los Aproximantes de Padé

Este método consiste en calcular $f(S)$ mediante una variante del método de los aproximantes de Padé para matrices casi triangulares superiores. Para ello, se han adaptado algunas de las rutinas que aparecen en [Tech_report_Pade]. El algoritmo resultante se presenta a continuación.

Algoritmo 12:	Algoritmo para el cálculo de funciones de matrices, basado en la forma real de Schur y en los aproximantes de Padé.
Entrada:	Matriz $A \in \Re^{nxn}$ y $f(.)$ función.
Salida:	Matriz $B = f(A) \in \Re^{nxn}$.

1 Obtener las matrices Q y S de la descomposición real de Schur de la matriz A.

2 Calcular $F = f(S)$, mediante aproximantes de Padé, basados en el método de Horner o en el método de de Paterson-Stockmeyer y Van Loan ([PaSt73],[VanL79]).

3 Calcular $B = QFQ^T$.

5 Implementaciones y Resultados

5.1 Software y Hardware Utilizados

5.2 Rutinas Implementadas

5.2.1 Rutinas Auxiliares

- Rutinas que calculan funciones de matrices de orden 2, correspondientes a los bloques diagonales $2x2$ de la forma real de Schur de una matriz, utilizando para ello la expresión (10).
  - $B = alfa^A$: *dlaax(alfa, n, A, lda, B, ldb).*
  - $B = \cos A$: *dlacs(alfa, n, A, lda, B, ldb).*
  - $B = e^A$: *dlaex(alfa, n, A, lda, B, ldb).*
  - $B = L_n A$: *dlaln(alfa, n, A, lda, B, ldb).*
  - $B = \log_{alfa} A$: *dlalg(alfa, n, A, lda, B, ldb).*
  - $B = A^{p/q}$: *dlapt(p,q, n, A, lda, B, ldb).*
  - $B = \sin A$: *dlasn(alfa, n, A, lda, B, ldb).*
- Rutinas que realizan operaciones básicas del tipo matriz-vector o matriz-matriz para matrices casi triangulares superiores. Los argumentos *A*, *B* y *C* que aparecen en la descripción de las rutinas, corresponden a matrices casi triangulares superiores, los vectores *h* y *g*, a vectores característicos de las estructura casi triangular superior de una matriz, y *x* e *y*, a vectores cualesquiera.
  - *dlaqt(n, A, lda, nh, h, g)*: obtiene los parámetros necesarios para realizar operaciones con matrices casi triangulares.
    - *g(1:n)*: *g(j)* indica el numero de elementos no nulos en la columna j-ésima de la matriz *A.*
    - *h(1:nh)*: contiene los índices de filas de los elementos no nulos situados debajo de la diagonal principal de la matriz *A.*
  - *dlaqi(g, alpha, beta, n, A, lda, B, ldb, info)*: calcula *B=alphaI+betaA.*

- *dlaqm(g, alpha, n, A, lda, B, ldb, info)*: calcula *A=alphaB+A*.
- *dqtmma(nh, h, g, n, A, lda, B, ldb, C, ldc, work, info)*: calcula *C=AB+C*.
- *dqtmmm(nh, h, g, n, A, lda, B, ldb, work, info)*: calcula *B=AB*.
- *dqtmv(nh, h, n, A, lda, x, work, info)*: calcula *x=Ax*.
- *dqtmva(nh, h, n, A, lda, x, y, work, info)*: calcula *y=Ax+y*.
- *dlalg(alfa, n, A, lda, B, ldb)*.

- Rutinas que calculan funciones racionales de matrices casi triangulares superiores.
  - *dqtrap(s, nh, h, g, gpq, p, q, n, A, lda, B, ldb, work, lwork, info)*: calcula *B=p(A)/q(A)*, siendo *p* y *q* polinomios de grado *gpq*, mediante el método de Paterson-Stockmeyer con factor de agrupamiento igual a *s*.
  - *dqtrah(n, A, lda, gpq, p, q, B, ldb, work, lwork, info)*: calcula *B=p(A)/q(A)*, siendo *p* y *q* polinomios de grado *gpq*, mediante el método de Horner.
- Otras rutinas.
  - *dlagb(tb, n, T, ldt, ng, g)*: determina, a partir de un tamaño de bloque tb, los índices de fila/columna de los bloques en la que queda dividida una matriz T, almacenándolos en el vector *g* (**¡Error! No se encuentra el origen de la referencia.**).
  - *dgetrd(n, A, lda, B, ldb, ipiv)*: resuelve la ecuación matricial $XA = B$, paso necesario para calcular T en la expresión (2).

5.2.2 Rutinas que Calculan Funciones de Matrices Casi Triangulares Superiores

- Las rutinas basadas en la resolución de la ecuación conmutante tienen la siguiente sintaxis:

  dtrfsxx(alfa, fun, n, T, ldt, F, ldf, iwork, liwork, info),

  siendo
  - xx: tipo de rutina,
  - *alfa*: parámetro característico de la función $f(.)$,
  - *fun*: función $f(.)$ considerada,
  - *n*: dimension de la matriz T,
  - *T*: matriz casi triangular superior,
  - $F = f(T)$.

  Las rutinas implementadas son:
  - *dtrfsc*: rutina basada en el algoritmo orientado a columnas.
  - *dtrfscb*: rutina basada en el algoritmo orientado a columnas de bloques.
  - *dtrfsd*: rutina basada en el algoritmo orientado a diagonales.
  - *dtrfsdb*: rutina basada en el algoritmo orientado a diagonales de bloques.
- Rutina *dtrfpp(s, gp, fun, n, A, lda, work, lwork, iwork, liwork, info),*

calcula $f(A)$, siendo $f(.)$ una función matricial definida en el argumento de entrada *fun*, mediante el aproximante diagonal de Padé de grado *gp* y el método de Paterson-Stockmeyer y Van Loan, con factor de agrupamiento igual a *s*.

- Rutina *dtrfph(gp, fun, n, A, lda, work, lwork, iwork, liwork, info)*
  calcula $f(A)$, siendo $f(.)$ una función matricial definida en el argumento de entrada *fun*, mediante el aproximante diagonal de Padé de grado *gp* y el método de Horner.

5.2.3 Rutinas que Calculan Funciones de Matrices mediante la Forma Real de Schur de una Matriz

- Rutinas basadas en la resolución de la ecuación conmutante:
  - orientadas a columnas y a diagonales:
    dgefsxx(alfa, fun, n, A, lda, iwork, liwork, info),
  - orientadas a columnas de bloques y a diagonales de bloques:
    dgefsxx(tb, alfa, fun, n, A, lda, iwork, liwork, info),

  siendo
  - xx: tipo de rutina,
  - *alfa*: parámetro característico de la función $f()$,
  - *fun*: función $f()$ considerada,
  - *n*: dimension de la matriz *A*,
  - *A*: matriz casi triangular superior (a la salida *A* es sobreescrita por $f(A)$.

  Las rutinas implementadas son:
  - *dgefsc*: rutina basada en el algoritmo orientado a columnas.
  - *dgefscb*: rutina basada en el algoritmo orientado a columnas de bloques.
  - *dgefsd*: rutina basada en el algoritmo orientado a diagonales.
  - *dtrfsdb*: rutina basada en el algoritmo orientado a diagonales de bloques.
- Rutina que calcula funciones de matrices mediante la forma real de Schur con diagonalización por bloques. Su sintaxis es
  - *dgefsbd(alfa, fun, n, A, lda, work, lwork, iwork, liwork,info),*
    con igual significado de los argumentos de la matriz.
- Rutinas basadas en la forma real de Schur de un matriz y en los aproximantes diagonales de Padé.
  - *dgefpp(s, gp, fun, n, A, lda, work, lwork, iwork, liwork, info)*: calcula $f(A)$, siendo $f(.)$ una función definida mediante el argumento de entrada *fun*, utilizando un aproximante diagonal de Padé de grado *gp* y el método de Paterson-Stockmeyer y Van Loan, con factor de agrupamiento igual a *s*.
  - *dtrfph(gp, fun, n, A, lda, work, lwork, iwork, liwork, info):* calcula $f(A)$, siendo $f(.)$ una función definida mediante el argumento de entrada *fun*, utilizando el aproximante diagonal de Padé de grado *gp* y el método de Horner.
- Rutina basada en la forma real de Schur con agrupación de valores propios cercanos.

– *dgefscc(s, gp, tol, alfa, fun, n, A, lda, work, lwork, iwork, liwork, info),*
siendo *gp* el grado del aproximante diagonal de Padé, *s* el factor de agrupamiento en el método de Paterson-Stockmeyer, *tol* la tolerancia utilizada para la formación de los clusters de valores propios.

5.3 Resultados y Conclusiones

En este apartado se describen los resultados obtenidos en las pruebas realizadas sobre las rutinas de la subapartado 5.2.3, salvo los correspondientes a la rutina basada en la agrupación de valores propios (rutina *dgefscc*), debido a que se ha comprobado que a partir de dimensiones de orden superior a 20, se comete un gran error a menos que se elija una agrupación mayor, coincidente en la mayoría de los casos con un único bloque. En tales casos es más eficiente utilizar directamente las rutinas basadas en los aproximantes diagonales de Padé (rutinas *dgefpp* y *dgefpp*), puesto que éstas no necesitan reordenaciones. La justificación de este comportamiento se encuentra en que el error cometido al resolver una ecuación

$$\overline{T}_{ii} X - X \overline{T}_{jj} = F_{ij},$$

satisface que

$$\frac{\left\| X - \tilde{X} \right\|_F}{\left\| X \right\|_F} \cong u \frac{\left\| F_{ij} \right\|_F}{sep\left(T_{ii}, T_{jj}\right)}.$$

Por lo tanto, para dimensiones grandes y en funciones del tipo exponencial, el error puede ser muy grande debido tanto al valor de $sep\left(T_{ii}, T_{jj}\right)$ como el de $\left\| F_{ij} \right\|_F$.

Todas las pruebas que se presentan se han realizado utilizando la función exponencial. El motivo es que la precisión y las prestaciones obtenidas para otras funciones serían semejantes, dadas las características de los algoritmos basados en la forma real de Schur, con la única excepción del basado en los aproximantes diagonales de Padé. En las pruebas realizadas se ha utilizado la rutina *dgefes* ([Tecreport]) basada en los aproximantes diagonales de Padé, para compararla con las rutinas anteriormente mencionadas.

Una de las medidas utilizadas para determinar la precisión de los algoritmos implementados es el error relativo cometido. Esto no siempre es posible puesto que la solución exacta únicamente se puede conocer en determinados casos y de pequeña dimensión, exceptuando, evidentemente, el caso de exponenciales de matrices diagonales. Por ello, en algunas de las pruebas se han utilizado matrices cuya exponencial es conocida.

Otra forma de medir el error en la solución, está basado en la ecuación conmutante. Como es sabido, las funciones de matrices cumplen la denominada ecuación conmutante; es decir, dada una matriz cuadrada A y $f(.)$ una función, entonces

$$Af(A) = f(A)A.$$

Se define el error cometido en la ecuación conmutante como

$$E_r = \frac{\left\| Af(A) - f(A)A \right\|}{\left\| Af(A) \right\|},$$

siendo $\|.\|$ una norma matricial. En los resultados que se presentan en esta sección se han considerado la 1-norma matricial.

Este error nos permite dar una estimación del error cometido, de hecho, como se evidencia en los resultados obtenidos, guarda una cercana relación con el error relativo en la solución.

En primer lugar presentamos los resultados obtenidos en cuanto a precisión, calculando el error relativo en el cálculo de la exponencial y el error relativo en la ecuación conmutante. Para ello se han generado matrices de la mayor dimensión posible que pudieran ser calculadas de manera exacta por Matlab, utilizando las funciones de Matlab *sym* y *double* que permiten pasar, respectivamente, datos de doble precisión a forma simbólica y datos de forma simbólica a doble precisión. Se han considerado las matrices

$$A_{40} \in \Re^{40x40} \begin{cases} a_{ij} = 0, & i > j \\ a_{ii} = i, & \\ a_{ij} = -1, & i < j \end{cases}$$

y

$$A_{70} \in \Re^{70x70} \begin{cases} a_{ij} = 0, & i > j \\ a_{ii} = 1, & \\ a_{ij} = -1, & i < j \end{cases} .$$

Aunque se han realizado pruebas en las que los parámetros característicos de cada una de ellas se han variado, únicamente se va a presentar los resultados correspondientes a un tamaño de bloque *s=4* (rutinas *dgefsdb* y *dgefscb*), por presentar tiempos de ejecución menores, un grado del aproximante diagonal de Padé (rutinas *dgefsph* y *dgefspp*) igual a 7, por presentar una precisión suficiente, y un factor de agrupamiento *s=4* (rutina *dgefspp*) por presentar un tiempo de ejecución menor.

Er	*Ecuación conmutante*	*Exponencial*
dgefes	0.31100E-15	0.22725E-13
dgefsd	0.47810E-16	0.10017E-15
dgefsc	0.33498E-16	0.50839E-16
dgefsbd	0.24563E-16	0.33250E-14
dgefsdb: tb= 6	0.44342E-16	0.93251E-16
dgefscb: tb= 6	0.35363E-16	0.48357E-16
dgefsph,gp=7	0.17226E-15	0.13560E-13
dgefspp,gp=7,s=4	0.50249E-15	0.22583E-13

Tabla 1: Errores relativos de las rutinas *dgefes*, *dgefsd*, *dgefsc*, *dgefsbd*, *dgefsph* y *dgefspp*, para la matriz A_{40}.

Er	*Ecuación conmutante*	*Exponencial*
dgefes	0.21248E-15	0.45270E-13
dgefsd	0	0.83864

dgefsc	0	0.83864
dgefsbd	0.33016E-15	0.10266E-13
dgefsdb: tb= 6	0	0.83864
dgefscb: tb= 6	0	0.83864
dgefsph,gp=7	0.33016E-15	0.10266E-13
dgefspp,gp=7,s=4	0.31550E-15	0.19864E-13

Tabla 2: Errores relativos de las rutinas *dgefes*, *dgefsd*, *dgefsc*, *dgefsbd*, *dgefsph* y *dgefspp*, para la matriz A_{70} .

A continuación se presentan los errores relativos cometidos en la ecuación conmutante y los tiempos de ejecución correspondientes a las rutinas que calculan la exponencial de matrices aleatorias de tamaños de 100, 500 y 1000.

Er	*100*	*500*
dgefes	0.12378E-13	0.80748E-13
dgefsd	0.59172E-14	0.98738E-14
dgefsc	0.59172E-14	0.98738E-14
dgefsbd	0.68640E-14	0.15640E-13
dgefsdb: tb= 6	0.59369E-14	0.98059E-14
dgefscb: tb= 6	0.59381E-14	0.98646E-14
dgefsph,gp=7	0.67320E-14	0.12915E-13
dgefspp,gp=7,s=4	0.75899E-14	0.15556E-13

Tabla 3: Error relativo cometido en la ecuación conmutante, por las rutinas *dgefes*, *dgefsd*, *dgefsc*, *dgefsbd*, *dgefsph* y *dgefspp*, para matrices de dimensiones 100, 500 y 1000.

Según la precisión obtenida, los algoritmos se pueden clasificar en tres grupos:

- Grupo 1: rutina *dgefes* (basada en Padé), *dgefsph* (basada en Schur y Padé) y *dgefspp* (basada en Schur y Padé).
- Grupo 2: rutinas *dgefsd*, *dgefsc*, *dgefsdb* y *dgefscb* (basadas en los aproximantes diagonales de Padé y en la resolución de la ecuación conmutante).
- Grupo 3: rutina *dgefsbd* (basada en la diagonalización por bloques de una matriz).

A partir de la Tabla 3, se pueden extraer las siguientes conclusiones:

- Los errores relativos cometidos en el cálculo de la función de una matriz y en la ecuación conmutante, son similares aunque este último es algo mayor que el primero. Esto es especialmente útil cuando no es conocido el valor exacto de la solución, pues permite conocer, aproximadamente, el error relativo cometido en el cálculo de funciones de matrices, a partir del error cometido en la ecuación conmutante.
- Las rutinas del grupo 2 son algo más precisas que las restantes, siempre y cuando las matrices no presenten valores propios cercanos.
- La rutina del grupo 3 es algo más precisa que las rutinas del grupo 1.

- Para valores propios cercanos, los errores relativos cometidos por las rutinas de los grupos 2 y 3 son de magnitudes semejantes, cosa que no ocurre con las rutinas del grupo 2, las cuales presentan un elevado error.

A continuación se muestran los tiempos de ejecución de las rutinas estudiadas.

Te	*100*	*500*	*1000*
dgefes	0.83570E-02	1.1626	8.6599
dgefsd	0.47339E-01	7.3742	69.256
dgefsc	0.32485E-01	6.7615	69.013
dgefsbd	0.61035E-01	15.539	199.85
dgefsdb: tb= 6	0.26432E-01	5.3908	58.146
dgefscb: tb= 6	0.26529E-01	5.4511	59.274
dgefsph,gp=7	0.37289E-01	6.7540	71.948
dgefspp,gp=7,s=4	0.34184E-01	5.7403	68.020

Tabla 4: Tiempos de ejecución, en segundos, de las rutinas *dgefes*, *dgefsd*, *dgefsc*, *dgefsbd*, *dgefsph* y *dgefspp*, para matrices de dimensiones 100, 500 y 1000.

De la tabla anterior se pueden extraer las siguientes conclusiones:

- La rutina *dgefes*, basada en los aproximantes diagonales de Padé, es la más rápida.
- Las rutinas basadas en la resolución de la ecuación conmutante por bloques (*dgefsdb*, *dgefscb*) se encuentran en segundo lugar.
- Las rutinas basadas en la resolución de la ecuación conmutante por diagonales y columnas (*dgefsd*, *dgefsc*) y las rutinas basadas en la forma real de Schur y Padé (*dgefsph*, *dgefspp*) se encuentran en tercer lugar.
- La rutina basada en la forma real de Schur y en la diagonalización por bloques (*dgefsbd*) es la más lenta.

La justificación de estos resultados es:

- Los algoritmos basados en los aproximantes de Padé son menos costosos que los basados en la forma real de Schur de una matriz.
- Los algoritmos orientados a bloques tienen tiempos de ejecución menores debido a la localidad de los datos.

6 Conclusiones

En este informe técnico se ha descrito un esquema general para el cálculo de funciones de matrices, basado en forma real de Schur de una matriz. Además, se han diseñado e implementado numerosos algoritmos basados en este esquema. En las pruebas realizadas se han variado los valores de tamaño de bloque *tb* (rutinas *dgefscb* y *dgefsdb*), grado de la aproximación diagonal de Padé *g* (rutinas *dgefpp* y dgefph) y el factor de agrupamiento *s* (rutina *dgefpp*), intentado determinar los valores óptimos de dichos parámetros en cuanto a precisión y prestaciones. Un resumen de estas conclusiones se presenta a continuación:

- Es fundamental disponer de rutinas que implementen, de una forma sencilla, el cálculo de un amplio espectro de funciones de matrices.
- Las rutinas basadas en la forma real de Schur tienen una mayor precisión que las rutinas basadas en los aproximantes diagonales de Padé, pero con un coste computacional mayor. Cuando la matriz considerada tiene valores propios múltiples o cercanos entre sí, las rutinas que tienen un mejor comportamiento son las basadas en la diagonalización por bloques y en los aproximantes diagonales de Padé. De la comparación de estos dos métodos se pueden extraer las siguientes conclusiones:
  - El método más preciso corresponde al de diagonalización por bloques, aunque su coste computacional es mayor que el correspondiente al otro método.
  - El algoritmo basado en los aproximantes diagonales de Padé tiene una precisión muy aceptable junto con un tiempo de ejecución menor que los otros dos.

- Si en el algoritmo basado en la agrupación de valores propios, se elige un valor pequeño de la tolerancia, entonces se cometen grandes errores, incluso para pequeños tamaños de matrices. Si se elije un tamaño de tolerancia mayor, el problema es que, además de tener que realizar muchas reordenaciones, el número de bloques es reducido (en la mayoría de las ocasiones un solo bloque), con lo cual es mejor utilizar directamente el algoritmo basado en los aproximantes diagonales de Padé, pues no se tienen que realizar reordenaciones, y el coste computacional es por tanto menor.

7 Bibliografía

[BaSt71] R. H. Bartels and G. W. Stewart, "Algorithm 432, solution of the matrix equation AX+XB=C", Communications of the ACM, Vol. 15, Nº 9, pp. 820-913, 1972.

[BlDo99] S. Blackford, J. Dongarra, "LAPACK Working Note 41 Installation Guide for LAPACK, Version 3.0", Department of Computer Science University of Tennessee Knoxville, Tennessee, 1999.

[GoNV79] G. H. Golub, S. Nash and C. F. Van Loan, "A Hessenberg-Schur method for the matrix problem AX+XB=C", IEEE Trans. Auto. Cont., AC-24, pag. 909-913, 1979.

[GoVa96] G. H. Golub and C. F. Van Loan, "Matrix Computations", Third Edition, The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[Grif81] D. H. Griffel, "Applied Functional Analysis". Ellis Hordwood Series, Mathematics and its applications, 1981.

[Hig103] N. J. Higham, "A schur-Parlett Algorithm for Computing Matrix Functions". SIAM J. Matrix Anal. Appl., Vol. 25, No. 2, pp. 464-485, 2003.

[Ibañ97] J. Javier Ibáñez González, "Algoritmos Paralelos para el Cálculo de la Exponencial de una Matriz sobre Multiprocesadores de Memoria Compartida". Proyecto Fin de Carrera UPV, 1997.

[IMKL03] Intel Math Kernel Library, Reference Manual. Intel Corporation, 2003.

[NgPa87] K.C. Ng and B. N. Parlett, "Programs to swap diagonal blocks", CPAM report, University of California, Bercleley, 1987.

[Parl74] B. N. Parlett, "Computation of matrix-valud functions". SIAM J. Matrix Anal., Vol 14, pp. 1061-1063, 1974.

[PaSt73] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials", SIAM J. Comput., Vol. 2, pp. 60–66, 1973.

[SGIA03] SGI Altix Applications Development an Optimization, Release Augost 1, 2003.

[VanL79] C. F. Van Loan, "A Note on the Evaluation of Matrix Polynomials", IEEE Transactions on Automatic Control, Vol. AC-24, 1979.

[Varah79] J. M. Varah, “On the separation of two matrices”. SIAM J. Numerical Analysis, Vol. 16, pp. 216-222, 1979.