Universitat Politècnica de València

Departamento de Sistemas Informáticos y Computación

Algoritmos de Altas Prestaciones para el Cálculo de la Descomposición en Valores Singulares y su Aplicación a la Reducción de Modelos de Sistemas Lineales de Control

Tesis Doctoral

Requisito para la Obtención del Grado de Doctor en Informática
por la Universitat Politècnica de València

Presentada por
Carlos Alberto da Silva Sanches de Campos

Dirigida por
Dr. D. Rui Manuel da Silva Ralha
Dr. D. José E. Román Moltó

Valencia
Octubre de 2014

Esta hoja ha sido dejada en blanco de forma intencionada.

"O valor das coisas não está no tempo que elas duram,
mas na intensidade com que acontecem.
Por isso existem momentos inesquecíveis, coisas inexplicáveis
e pessoas incomparáveis."

Fernando Pessoa

Dedicado a mis padres, João y Maria

Esta hoja ha sido dejada en blanco de forma intencionada.

Agradecimientos

AL llegar al final de un largo viaje, uno contempla todo el camino recorrido y recuerda cada una de las etapas por las que ha ido pasando. Ha sido un largo camino para llegar hasta aquí. Un camino que empezó en una tarde de primavera de 1996, en las escaleras de la entrada principal del Departamento de Matemática de la Universidad de Coimbra con el Dr. José Vitória. Sin su consejo, creo que jamás habría descubierto el apasionante mundo del Álgebra Lineal Numérica y de la Computación Paralela. Por este motivo, mi primer agradecimiento es para el Dr. José Vitória, a quien debo que hoy pueda dedicarme a lo que más me apasiona y a quien tengo como mi punto de referencia en toda mi vida académica y profesional.

Por otra parte, esta tesis es producto no sólo de quien escribe, sino de todas las personas que de una manera u otra, voluntaria o involuntariamente, he involucrado en su desarrollo.

Todavía, en algunas etapas de mi vida, cuando llega la hora de los agradecimientos, me he planteado la opción de afirmar solamente *"Gracias a todos..."* y nada más. El motivo es por evitar resumir en una sola página la mención de tantas personas a las que debo mucho. Por otro lado, siempre me quedo con la sensación de que he olvidado alguien importante, o que mis palabras son insuficientes para transmitir mi profundo agradecimiento. Sin embargo, no quiero perder la oportunidad de expresar mi agradecimiento a todos aquellos que de alguna manera han contribuido a este feliz desenlace.

Quiero empezar por expresar mi profundo agradecimiento a los directores del trabajo, Dr. Vicente Hernández y Dr. Rui Ralha, por su constante apoyo, ayuda,

paciencia, fuente de motivación y a los que considero mis maestros y ejemplo de dedicación. A Dr. Rui Ralha quiero agradecerle la confianza que depositó en mí desde un principio y sin duda, el camino recorrido ha sido mucho más fácil gracias a su fuente inagotable de conocimientos. Además, la etapa final del trabajo fue también dirigida por el Dr. José Román. A él quiero igualmente expresar mi profundo agradecimiento por su incondicional ayuda, por su constante perseverancia en motivarme a terminar la tesis y por todo su apoyo a lo largo de este recorrido.

Quiero también dirigir mi agradecimiento, de manera especial, a David Guerrero, por su atención, apoyo, consejos y por su productiva colaboración en el desarrollo de la tesis.

Gracias a Germán que soportó en mis comienzos mis reiteradas preguntas y tonterías. Gracias por estar siempre ahí, por su compañerismo y amistad, y por haber sido el mejor compañero de viaje en esta jornada de nuestras vidas.

Gracias, también, a todos los miembros del Grupo de Redes y Computación de Altas Prestaciones, por el apoyo y por el excelente clima de trabajo del que he disfrutado, por su ejemplo como maestros del trabajo bien hecho y por mantener, día a día, al Grupo como un lugar de camaradería. Además, gracias por acogerme con tanto cariño donde siempre he encontrado una afectuosa ayuda y comprensión para mis desventuras en tierras de España. El mero hecho de saber que estáis siempre ahí me reconforta y eso no tiene nombre.

Gracias a Marisa, José, Fernando, Salva, Toni y Luís, por las noches inolvidables de cine que hemos pasado, a los que me une una gran amistad y que me han ayudado a descubrir las maravillas de Valencia, ciudad a la que siempre quiero volver. Gracias, además, por tanto apoyo que he recibido de manera tan natural, espontánea y fraternal.

Gracias a Sergio, el amigo extraordinario con quien he disfrutado de tantas horas divertidas, inolvidables y que llenan mi vida de perpetuos recuerdos.

A mis amigos de siempre: Fernando, Odete y Augusto, de los que siempre recibo mucho más de lo que doy, y con quienes he compartido y compartiré tantos momentos de mi vida.

A Pedro Matos, el compañero que desde un principio me motivó a hacer este camino, que siempre me ha dado su apoyo incondicional, su cariño y amistad, y que infelizmente ha tenido que partir durante esta jornada. Seguirás vivo en mi corazón por todos los días de mi vida.

Gracias a Luís, el siempre presente compañero de despacho, con quien he compartido tantos desahogos durante esta caminada, a quien me une un gran compañerismo y una especial amistad.

Gracias también a los restantes compañeros de Departamento, amigos y familiares, por estar siempre ahí con su apoyo y por aportarme tantas cosas buenas.

A Rui y a Dina, gracias por estar siempre junto a mí y a los que debo agradecer tantas cosas.

En último lugar quiero expresar con mayor énfasis un agradecimiento muy especial. Es para mis padres: João y Maria, y para Tomás y Luz, a quienes quiero como si lo fueran y a quienes quiero agradecer todo su cariño y apoyo incondicional, siempre preocupados por mi felicidad, para que viva cada día con una sonrisa en la cara y a quienes debo tanto en mi vida.

¡Gracias a Todos!
¡Os Quiero Mucho!

Esta hoja ha sido dejada en blanco de forma intencionada.

Resumen

PARA calcular la descomposición en valores singulares (DVS) de una matriz real densa, los métodos tradicionales empiezan por reducir la matriz a una forma bidiagonal y seguidamente calculan la DVS de esa matriz bidiagonal.

El proceso para reducir la matriz inicial a la forma bidiagonal es conocido como el método de la bidiagonalización, que en general consiste en la aplicación de sucesivas transformaciones de Householder, por la izquierda y por la derecha de la matriz. El hecho de que las transformaciones sean aplicadas por los dos lados de la matriz, repercute negativamente en los costes de comunicaciones de una implementación paralela destinada a sistemas de memoria distribuida.

Ralha y Barlow presentaron dos nuevos métodos para la bidiagonalización de matrices densas en los que las transformaciones de Householder son aplicadas solamente por el lado derecho de la matriz. Esto permite definir todas las operaciones en términos de las columnas de la matriz a transformar, facilitando así el desarrollo de implementaciones paralelas y que además reducen las comunicaciones necesarias.

En esta tesis se ha realizado un estudio comparativo entre las implementaciones secuenciales y paralelas de los métodos presentados por Ralha y por Barlow, desarrolladas en el entorno de las librerías LAPACK y SCALAPACK, y las correspondientes rutinas de estas librerías.

Como trabajo novedoso se han introducido algunas modificaciones en el método de Barlow con el objetivo de reducir el número de comunicaciones en la implementación paralela.

Tras estas líneas de investigación, el paso siguiente es calcular la DVS de

la matriz bidiagonal superior. Sin embargo, el problema de calcular la DVS de una matriz bidiagonal superior puede verse como el problema de calcular la descomposición en valores propios (DVP) de una matriz tridiagonal simétrica. Una vez calculada la DVP de la matriz tridiagonal simétrica es posible obtener la DVS de la matriz bidiagonal superior y con ella, la DVS de la matriz inicial.

La principal motivación para esta estrategia es el desarrollo de una implementación paralela, sin comunicaciones, del método `zeroinNR` propuesto por Ralha en su tesis doctoral, para el cálculo de la DVP de matrices tridiagonales simétricas y el correspondiente estudio comparativo con la implementación paralela estándar, la cual contiene comunicaciones.

Como ámbito de aplicación de la DVS se ha estudiado la reducción de modelos de sistemas lineales de control, basado en la diagonalización simultánea de los Gramianos de controlabilidad y de observabilidad, y nuestro enfoque va dirigido a la reducción a la forma bidiagonal superior del producto matricial sin calcular explícitamente ese producto y, para ello, se han desarrollado implementaciones secuenciales y paralelas del método propuesto por Golub, por Sølna y por van Dooren.

Los resultados presentados en esta tesis han sido obtenidos en los recursos computacionales ofrecidos por el Grupo de Redes y Computación de Altas Prestaciones (GRyCAP) de la Universitat Politècnica de València (UPV) y por el consorcio SEARCH[1] de la Universidad del Minho (UM).

[1] SEARCH: *Services and Advanced Research Computing with HTC/HPC clusters.*

Summary

Title

High Performance Algorithms for Computing the Singular Value Decomposition and Its Application to Model Reduction of Linear Control Systems

Abstract

TO calculate the singular value decomposition (SVD) of a real dense matrix, traditional methods start by reducing the initial matrix to a bidiagonal form, and then, calculate the SVD of the bidiagonal matrix.

The process of reducing the initial matrix to bidiagonal form is known as the bidiagonalization method, which generally consists of applying successive Householder transformations from the left and right of the matrix. Since the reflectors are applied on both sides of the matrix, this has a negative impact on the communication overheads of a parallel implementation for distributed memory systems.

Ralha and Barlow have proposed two new methods for the reduction of dense matrices to bidiagonal form in which the Householder transformations are applied only on the right side of the matrix. This allows to define all operations in terms of full columns of the matrix under transformation. Therefore, these methods are more attractive for distributed memory systems than the standard

methods and they reduce the communication overheads.

In this thesis we present a comparative study between our sequential and parallel implementations of Ralha's and Barlow's method, developed under LAPACK's and SCALAPACK's environment and its corresponding routines.

As a novel study, some changes were introduced in the Barlow's method with the aim of reducing the number of communications in the parallel implementation.

Following this line of research, the next step is to compute the SVD of the upper bidiagonal form. However, the standard problem of computing the SVD of an upper bidiagonal matrix can be seen as the problem of computing the eigenvalue decomposition (ED) of a symmetric tridiagonal matrix. Once we obtain the ED of a symmetric tridiagonal matrix, we can compute the SVD of the corresponding upper bidiagonal matrix, which is an intermediate step to obtain the SVD of the initial matrix.

The main motivation for this strategy is the development of a parallel implementation, without communication, of `zeroinNR` method proposed by Ralha in his doctoral thesis, for the resolution of the ED problem and its corresponding comparative study between our implementation and the standard one, with communications.

The model reduction of linear control systems has been studied as the scope of SVD application, and our approach was directed at the reduction to upper bidiagonal form the top of the matrix product without explicitly calculate this product, and to do this, sequential and parallel implementations of the method proposed by Golub, by Sølna and by van Dooren have been developed.

The results presented in this thesis have been obtained in the computational resources offered by the Networks and High Performance Computing Group (GRyCAP) from the Universitat Politècnica de València (UPV) and the consortium SEARCH[2] from the University of Minho (UM).

[2]SEARCH: *Services and Advanced Research Computing with HTC/HPC clusters.*

Resum

Títol

Algoritmes d'Altes Prestacions per al Càlcul de la Descomposició de Valors Singulars i la seua Aplicació a la Reducció de Models de Sistemes Lineals de Control

Resum

PER a calcular la descomposició en valors singulars (DVS) d'una matriu real densa, els mètodes tradicionals comencen per reduir la matriu a una forma bidiagonal i de seguida calculen la DVS d'eixa matriu bidiagonal.

El procés per a reduir la matriu inicial a la forma bidiagonal és conegut com el mètode de la bidiagonalització, que en general consistix en l'aplicació de successives transformacions de Householder, per l'esquerra i per la dreta de la matriu. El fet de que les transformacions siguen aplicades pels dos costats de la matriu, repercutix negativament en els costos de comunicacions d'una implementació paral·lela destinada a sistemes de memòria distribuïda.

Ralha i Barlow van presentar dos nous mètodes per a la bidiagonalització de matrius denses en els que les transformacions de Householder són aplicades només pel costat dret de la matriu. Açò, permet definir totes les operacions en termes de les columnes de la matriu a transformar, facilitant així el

desenrotllament d'implementacions paral·leles i a més, reduïxen significativament les comunicacions necessàries.

En esta tesi s'ha realitzat un estudi comparatiu entre les implementacions seqüencials i paral·leles dels mètodes presentats per Ralha i per Barlow, desenrotllades en l'entorn de les llibreries LAPACK i SCALAPACK, i les corresponents rutines d'estes llibreries.

Com a treball nou, s'han introduït algunes modificacions en el mètode de Barlow amb l'objectiu de reduir el nombre de comunicacions en la implementació paral·lela

Després d'estes línies d'investigació, el pas següent és calcular la DVS de la matriu bidiagonal superior. No obstant això, el problema de calcular la DVS d'una matriu bidiagonal superior pot veure's com el problema de calcular la descomposició en valors propis (DVP) d'una matriu tridiagonal simètrica. Una vegada calculada la DVP de la matriu tridiagonal simètrica és possible obtindre la DVS de la matriu bidiagonal superior i amb ella, la DVS de la matriu inicial.

La principal motivació per a esta estratègia és el desenrotllament d'una implementació paral·lela, sense comunicacions, del mètode `zeroinNR` proposat per Ralha en la seua tesi doctoral, per al càlcul de la DVP de matrius tridiagonals simètriques i el corresponent estudi comparatiu amb la implementació paral·lela estàndard, la qual conté comunicacions.

Com a àmbit d'aplicació de la DVS s'ha estudiat la reducció de models de sistemes lineals de control i el nostre enfocament va ser dirigit a la reducció a la forma bidiagonal superior del producte matricial sense calcular explícitament eixe producte i, per a això, s'han desenrotllat implementacions seqüencials i paral·leles del mètode proposat per Golub, per Sølna i per van Dooren.

Els resultats presentats en esta tesi han sigut obtinguts en els recursos computacionals oferits pel Grup de Xarxes i Computació d'Altes Prestacions (GRyCAP) de la Universitat Politècnica de València (UPV) i pel consorcio SEARCH[3] de la Universitat del Minho (UM).

[3] SEARCH: *Services and Advanced Research Computing with HTC/HPC clusters.*

Índice General

Esta hoja ha sido dejada en blanco de forma intencionada.

Índice de Figuras

Esta hoja ha sido dejada en blanco de forma intencionada.

Índice de Tablas

Capítulo 1

Introducción

En este capítulo introductorio se da una perspectiva general de la tesis con el título **"Algoritmos de Altas Prestaciones para el Cálculo de la Descomposición en Valores Singulares y su Aplicación a la Reducción de Modelos de Sistemas Lineales de Control"**.

El capítulo empieza por la descripción de la motivación para la tesis. A continuación expone los objetivos planteados para su desarrollo y describe las principales aportaciones que se han logrado. Seguidamente presenta los principales conceptos de la computación de altas prestaciones y termina con el resumen de la estructura de la memoria.

1.1. Motivación

El procedimiento más eficiente para calcular la **descomposición en valores singulares** (DVS) de una matriz real densa, es empezar por reducir la matriz a una forma bidiagonal, habitualmente a la forma bidiagonal superior, empleando un número finito de transformaciones ortogonales y posteriormente emplear un proceso iterativo para calcular la DVS de esa matriz bidiagonal.

El proceso para reducir la matriz inicial a la forma bidiagonal (superior) es conocido como el **método de la bidiagonalización** y en el cálculo de la DVS de una matriz real densa esta etapa tiene un coste computacional mucho más elevado que el proceso iterativo empleado posteriormente para calcular la DVS de la matriz bidiagonal.

El método de bidiagonalización más conocido es el método propuesto por Golub y por Kahan, y consiste en la aplicación de sucesivas transformaciones ortogonales, habitualmente **transformaciones de Householder**, por la izquierda y por la derecha de la matriz [Golub y Kahan, 1965].

El hecho de que las transformaciones de Householder sean aplicadas por los dos lados de la matriz repercute negativamente en las prestaciones de la implementación paralela del algoritmo sobre sistemas de memoria distribuida.

Intentando solucionar este problema, Ralha [Ralha, 1994], [Ralha y Mackiewicz, 1996], [Ralha, 2003] y Barlow [Barlow, 2003], [Barlow *et al.*, 2005], [Bosner y Drmač, 2005], [Bosner, 2006], presentaron **dos nuevos métodos para la bidiagonalización** de matrices reales densas en los que las transformaciones de Householder son aplicadas solamente por el lado derecho de la matriz. Esto permite definir todas las operaciones en términos de las columnas de la matriz a transformar, logrando así que el desarrollo de implementaciones paralelas sea más sencillo que con los métodos tradicionales, además de reducir significativamente las comunicaciones necesarias.

Terminado el método de bidiagonalización, la etapa siguiente es calcular la DVS de la matriz bidiagonal superior. Sin embargo, el problema de calcular la DVS de una matriz bidiagonal superior puede verse como el problema de calcular la **descomposición en valores propios** (DVP) de una matriz tridiagonal simétrica y esta metodología ha sido estudiada a lo largo del tiempo por distintos e importantes autores, pues una vez calculada la DVP de la matriz tridiagonal simétrica es posible obtener la DVS de la matriz bidiagonal superior y, con ella, la DVS de la matriz inicial [Demmel, 1997].

Las razones para esta estrategia son:

- El problema de calcular la DVP de una matriz tridiagonal simétrica es numéricamente tan estable como el problema de calcular la DVS de una matriz bidiagonal superior [Golub y van Loan, 1996].

- Para calcular los valores propios de la matriz tridiagonal simétrica es

habitual emplear el **método de bisección** [Wilkinson, 1965], [Golub y van Loan, 1996], [Parlett, 1998], el cual tiene excelentes propiedades para el desarrollo de una implementación paralela sobre sistemas de memoria distribuida y además, tiene características interesantes que se pueden explorar, en concreto, el desarrollo de una implementación paralela sin comunicaciones.

- Para acelerar la convergencia del cálculo de los valores propios de la matriz tridiagonal simétrica se puede emplear el **método de Newton--Raphson** [Wilkinson, 1965], [Parlett, 1998] y además, se puede emplear la modificación propuesta por Ralha en el **método** `zeroinNR` [Ralha, 1990].
- Para calcular los vectores propios de la matriz tridiagonal simétrica se puede emplear el **método MRRR** (*Multiple Relatively Robust Representations* o **Múltiples Representaciones Relativamente Robustas**), desarrollado por Parlett y por Dhillon [Dhillon, 1997], [Parlett y Dhillon, 2000] [Dhillon y Parlett, 2004a], [Dhillon y Parlett, 2004b], y cuya versión paralela está descrita, por ejemplo, en [Bientinesi *et al.*, 2005] y en [Vömel, 2010].

Como ámbito de aplicación de la DVS se ha estudiado la **reducción de modelos de sistemas lineales de control**.

Dado un sistema lineal de control de orden elevado, continuo e invariante en el tiempo, con la siguiente representación en el modelo de espacio de estados

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \qquad x(0) = x_0 \\ y(t) &= Cx(t) \end{aligned} \tag{1.1}$$

donde $x \in \mathbb{R}^n$ es el **vector de estados**, $u \in \mathbb{R}^m$ es el **vector de entradas** o **controles**, $y \in \mathbb{R}^p$ es el **vector de salidas**, $A \in \mathbb{R}^{n \times n}$ es la **matriz de estados**, $B \in \mathbb{R}^{n \times m}$ es la **matriz de entradas** o **controles** y $C \in \mathbb{R}^{p \times n}$ es la **matriz de salidas**, se puede encontrar un modelo de menor orden, con un comportamiento similar, que tendrá la representación

$$\begin{aligned} \dot{x_r}(t) &= A_r x_r(t) + B_r u(t) \qquad x_r(0) = \overline{x}_0 \\ y_r(t) &= C_r x_r(t) \end{aligned} \tag{1.2}$$

donde $x_r \in \mathbb{R}^r$, $y_r \in \mathbb{R}^p$, $A_r \in \mathbb{R}^{r \times r}$, $B_r \in \mathbb{R}^{r \times m}$ y $C_r \in \mathbb{R}^{p \times r}$, con $r \ll n$ [Petkov *et al.*, 1991].

Bajo ciertas condiciones, el vector de salidas y_r es una buena aproximación al vector de salidas y del sistema original, para todos los vectores de entradas u. Al proceso de obtener este modelo reducido, a partir del modelo original, se denomina **reducción de modelos de sistemas lineales de control**.

No hay un esquema universal para la reducción de modelos. Cuando se obtiene un modelo reducido, éste es más simple, pero a la vez, más inexacto. Así, siempre existirá un compromiso entre la precisión del modelo obtenido y su simplicidad.

De los distintos esquemas para la reducción de modelos, esta tesis se va a centrar en la reducción basada en **realizaciones balanceadas**, pues este tipo de aproximación permite obtener un modelo reducido que mantiene, en la medida de lo posible, las propiedades de respuesta temporal, controlabilidad y observabilidad, del modelo original. Además, la reducción basada en realizaciones balanceadas es un método muy general, que puede ser aplicado en una gran variedad de modelos.

En este tipo de esquema se intenta obtener una transformación de estados tal que, en la nueva representación, los **Gramianos de controlabilidad** (W_c) y **de observabilidad** (W_o), sean iguales y diagonales [Pernebo y Silverman, 1982], [Laub *et al.*, 1987]. Se puede así obtener un modelo de orden reducido, en el que se seleccionarán las r componentes que contribuyen más significativamente a estas importantes propiedades del sistema.

Para balancear el sistema original, el método descrito en [Laub *et al.*, 1987] está basado en la **diagonalización simultánea de los Gramianos de controlabilidad y de observabilidad**. En este método hay una etapa donde es necesario calcular la DVS del producto de los factores de Cholesky de los Gramianos de controlabilidad (W_c) y de observabilidad (W_o),

$$L_o^t L_c = U \Sigma V^t \tag{1.3}$$

donde $W_c = L_c L_c^t$ y $W_o = L_o L_o^t$.

Aunque el problema base sea el cálculo de la DVS del producto de dos matrices reales cuadradas, nuestro enfoque está dirigido a la reducción a la forma bidiagonal superior del producto matricial sin calcular explícitamente ese producto. En su tesis doctoral, Mollar [Mollar, 2003], desarrolló un estudio a través del cual se resuelve este problema mediante:

- El **método de Kogbetliantz implícito** [Kogbetliantz, 1955].

- El **método de Drmač** [Drmač, 1998].
- El **método de Golub-Sølna-van Dooren** [Golub *et al.*, 2000].

Como resultado de su investigación, Mollar concluye que las prestaciones obtenidas con el método de Drmač y con el método de Golub-Sølna-van Dooren son "muy similares" y además, mucho mejores que las prestaciones obtenidas con el método de Kogbetliantz implícito. Sin embargo, en el método de Drmač es calculado explícitamente el producto de las matrices que previamente son transformadas, pero el método no está descrito para su extensión para el cálculo de la matriz bidiagonal superior del producto de cualquier número de matrices reales cuadradas, cómo ocurre con el método de Golub-Sølna-van Dooren.

1.2. Objetivos

La tesis se centra en dos problemas fundamentales:

1. El cálculo de la DVS de una matriz real densa.
2. La aplicación de la DVS a la reducción de modelos de sistemas lineales de control, basada en realizaciones balanceadas.

Para el cálculo de la DVS de una matriz real densa, los objetivos perseguidos son:

- Realizar un estudio detallado de los métodos de Ralha [Ralha, 2003] y de Barlow [Barlow *et al.*, 2005], empleados en la reducción de una matriz real densa a la forma bidiagonal superior.
- Desarrollar implementaciones secuenciales y paralelas de los métodos anteriores, con altas prestaciones.
- Realizar un estudio de los métodos para calcular la DVS de una matriz bidiagonal superior, convirtiendo este problema en el problema de calcular la DVP de una matriz tridiagonal simétrica [Demmel, 1997].

- Desarrollar implementaciones secuenciales y paralelas para calcular los valores propios de la matriz tridiagonal simétrica, con altas prestaciones. Además, desarrollar un estudio comparativo entre una implementación paralela con comunicaciones basada en [Demmel *et al.*, 1995], y una implementación paralela sin comunicaciones basada en [Ralha, 2006].
- Emplear el método MRRR para calcular los vectores propios de la matriz tridiagonal simétrica.

Para la aplicación de la DVS a la reducción de modelos de sistemas lineales de control, basada en realizaciones balanceadas, los objetivos perseguidos son:

- Desarrollar implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren para el cálculo de la matriz bidiagonal superior del producto implícito de dos matrices reales cuadradas, con altas prestaciones.
- Desarrollar un estudio de las prestaciones obtenidas con las implementaciones anteriores y en concreto, calcular la matriz bidiagonal superior del producto de los factores de Cholesky (1.3) en [Laub *et al.*, 1987].
- Desarrollar implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren para el cálculo de la matriz bidiagonal superior del producto implícito de cualquier número de matrices reales cuadradas.
- Desarrollar un estudio de las prestaciones obtenidas con las implementaciones anteriores.

Los objetivos perseguidos con esta tesis están repartidos entre el desarrollo de implementaciones de métodos numéricos con altas prestaciones y el estudio comparativo entre las prestaciones obtenidas por las implementaciones de esos métodos y las prestaciones obtenidas por las correspondientes rutinas de librerías numéricas usadas habitualmente con la misma funcionalidad.

1.3. Principales aportaciones

Respecto al cálculo de la DVS de una matriz real densa, las principales aportaciones de la tesis son:

- Se ha realizado un estudio comparativo entre las implementaciones secuenciales y paralelas de los métodos de Ralha [Ralha, 2003] y de Barlow [Barlow *et al.*, 2005], desarrolladas en el entorno de las librerías numéricas LAPACK (*Linear Algebra Package*) [Anderson *et al.*, 1999], [webLAPACK, 2013] y SCALAPACK (*Scalable Linear Algebra Package*) [Blackford *et al.*, 1997], [webSCALAPACK, 2012], y las rutinas de estas librerías con la misma funcionalidad [Campos, 2004], [Campos *et al.*, 2004].

- Como trabajo más novedoso, se han propuesto modificaciones al método de Barlow [Barlow *et al.*, 2005], con el objetivo de reducir el número de comunicaciones en la implementación paralela destinada a sistemas de memoria distribuida [Campos *et al.*, 2007], [Campos *et al.*, 2008a].

- Se ha desarrollado una adaptación del método `zeroinNR` propuesto por Ralha en su tesis doctoral [Ralha, 1990], al cálculo de los valores propios de la matriz tridiagonal simétrica, y cuya implementación paralela tiene por objetivo no emplear comunicaciones [Ralha, 2006], [Campos *et al.*, 2008b], [Ralha y Campos, 2008].

- Se ha realizado un estudio comparativo entre las implementaciones secuenciales y paralelas del método anterior, y el método tradicional cuya implementación paralela está en SCALAPACK implementada según la metodología descrita en [Demmel *et al.*, 1995], y que además presenta comunicaciones.

Respecto a la aplicación de la DVS a la reducción de modelos de sistemas lineales de control, basada en realizaciones balanceadas, las principales aportaciones de la tesis son:

- Se han desarrollado implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren [Golub *et al.*, 2000], para el cálculo de la matriz bidiagonal superior del producto implícito de dos matrices reales, y se han estudiado sus prestaciones.

- Se han desarrollado las implementaciones secuenciales y paralelas del método anterior para el cálculo de la matriz bidiagonal superior del producto implícito de cualquier número de matrices reales cuadradas.

1.4. Computación de altas prestaciones

La **computación de altas prestaciones** (CAP) es el conjunto de técnicas que abarcan la obtención de las mejores prestaciones de las implementaciones algorítmicas según conceptos que sacan el máximo provecho del sistema computacional empleado y además, procuran aprovechar el uso colaborativo de los distintos procesadores en la resolución de un problema común [Gries, 1981], [Akl, 1989], [Kumar *et al.*, 1994], [Grama *et al.*, 2003].

Esencialmente, la CAP está basada en una gestión eficiente de la jerarquía de memorias. En efecto, el hecho de que hayan diferentes niveles de memoria implica un incremento del tiempo empleado en realizar una operación de acceso a la memoria, siempre que el nivel de la memoria esté lejos de los registros del procesador y, por tanto, una mala gestión de la memoria puede incrementar los fallos de caché e incluso de página, realizando accesos al disco duro y provocando peores prestaciones.

Además, para obtener buenas prestaciones cualquier implementación debe localizar los accesos a la memoria, tanto de los datos como de las instrucciones computacionales, en la medida de lo posible, en las memorias superiores de la jerarquía. Para lograrlo, se puede emplear un conjunto de técnicas como el desenrollado de bucles, la reescritura de los algoritmos a bloques, entre otros [Daydé y Duff, 1999].

Al mismo tiempo, para conseguir que una determinada implementación obtenga altas prestaciones hay que tener en cuenta otros aspectos, tales como:

- **Optimización del cálculo secuencial**: debe permitir la utilización de una adecuada estructura de datos para manejar la información del problema, así como la aplicación de los métodos numéricos más eficientes en su resolución.
- **Gestión eficiente de entrada/salida**: debe minimizar el impacto del acceso al disco duro, pues este tipo de acceso es mucho más lento que el acceso a la memoria principal del sistema.

- **Empleo de computación paralela**: debe permitir la utilización de múltiples procesadores de manera colaborativa en la resolución del problema.

La computación paralela está igualmente basada en dos aspectos muy importantes:

- **Particionamiento de tareas**: consiste en dividir una gran tarea en tareas más pequeñas y que puedan ser realizadas por distintos procesadores. En esta etapa es crucial identificar cuáles son las tareas que puedan ser agrupadas y que puedan ser ejecutadas al mismo tiempo por distintos procesadores. La **granularidad de las tareas** es un aspecto importante. Cuando las tareas contienen un número reducido de operaciones se denomina **granularidad fina**, en el caso contrario, se denomina **granularidad gruesa** [Kumar *et al.*, 1994], [Foster, 1995].

  Realizar una distribución equilibrada de las tareas por los distintos procesadores es otro requisito fundamental para obtener buenas prestaciones. A este procedimiento se denomina **balanceo de carga**. Garantizar una distribución de tareas y de datos, más o menos equilibrada, por los distintos procesadores, es un requisito para obtener una implementación con altas prestaciones.

- **Comunicación entre tareas**: aunque se pueda conseguir un buen particionamiento de tareas, no siempre es posible hacerlo de manera que los distintos procesadores trabajen de forma autónoma. En la mayoría de las implementaciones paralelas siempre llega un momento en que distintos procesadores tienen que cooperar en la resolución de una tarea común. En ese momento, los procesadores necesitan realizar operaciones de comunicación entre ellos. Una implementación paralela podrá obtener altas prestaciones siempre que consiga optimizar los costes de comunicaciones entre los procesadores, reduciendo el número y/o el volumen de las mismas.

1.4.1. Arquitecturas paralelas

En 1972, Flynn presentó una taxonomía para clasificar las arquitecturas computacionales [Flynn, 1972]. Aunque esta clasificación sea muy simple y haya

sido contestada por algunos autores (por ejemplo, por Hockney [Hockney y Jesshop, 1982] y por Chalmers [Chalmers y Tidmus, 1996]), pues ella no cubre todas las posibles arquitecturas, sigue siendo muy aplicada.

La taxonomía de Flynn se basa en el número de instrucciones concurrentes y en los flujos de datos disponibles en la arquitectura. Puede haber secuencias de instrucciones simples o múltiples, y secuencias de datos simples o múltiples. Esto da lugar a cuatro clasificaciones, de las cuales solamente dos son aplicables a las arquitecturas paralelas [Quinn, 2004]:

- **SISD** (*Single Instruction Single Data* o **Simple Instrucción Simple Dato**): arquitectura donde un solo procesador recibe una sola secuencia de instrucciones que operan en una secuencia de datos.
- **SIMD** (*Single Instruction Multiple Data* o **Simple Instrucción Múltiples Datos**): arquitectura donde múltiples procesadores ejecutan la misma secuencia de instrucciones, trabajando síncronamente sobre distintas secuencia de datos.
- **MISD** (*Multiple Instruction Single Data* o **Múltiples Instrucciones Simple Dato**): arquitectura donde múltiples procesadores ejecutan distintas secuencias de instrucciones sobre la misma secuencia de datos.
- **MIMD** (*Multiple Instruction Multiple Data* o **Múltiples Instrucciones Múltiples Datos**): arquitectura donde múltiples procesadores ejecutan distintas secuencias de instrucciones, de manera asíncrona, sobre distintas secuencia de datos.

La gran mayoría de las arquitecturas con más de un procesador pueden ser incluidas dentro de las clasificaciones SIMD y MIMD, de Flynn. Sin embargo, la gran diferencia entre los dos tipos de arquitecturas es la posibilidad que tienen los MIMD para ejecutaren distintas secuencias de instrucciones sobre distintas secuencia de datos, mientras que los SIMD ejecutan siempre la misma secuencia de instrucciones sobre los datos.

En los MIMD, ya que trabajan de manera asíncrona, en algún momento de sus ejecuciones distintos procesadores tendrán que trabajar de manera coordinada y, por lo tanto, tendrán que hacer tareas de comunicación y/o de sincronización.

Comúnmente, las arquitecturas MIMD son clasificadas según dos paradigmas y esa clasificación tiene por base el tipo de conexión y organización de la memoria:

- **Multiprocesadores de memoria compartida** (Figura 1.1): los procesadores comparten la misma unidad de memoria global, la cual es accesible por medio de la red de interconexión. En este tipo de sistema, la comunicación entre distintos procesadores se realiza mediante áreas de memoria común, a las que pueden acceder todos ellos, ya que todos los procesadores se encuentran igualmente comunicados con la memoria principal.

  En el modelo teórico, las lecturas y escrituras de todos los procesadores tienen exactamente las mismas latencias. Sin embargo, en computadores reales el tiempo de acceso suele ser variable en función de la distancia física desde el procesador que accede hasta el módulo de memoria accedido (arquitecturas NUMA, *Non-Uniform Memory Access* o de **acceso no uniforme a la memoria**).

  Es precisamente el acceso a esta unidad de memoria común uno de los puntos críticos de esta clase de sistemas. Habitualmente, los procesadores están conectados a la única unidad de memoria central a través de una antememoria para minimizar, en lo posible, el *cuello de botella* que supone el acceso a la memoria central.

  Las principales desventajas de este tipo de sistemas son el acceso simultáneo a memoria y la poca escalabilidad de procesadores, debido a que aumenta la complejidad y el coste económico de la red de interconexión al incrementar el número de procesadores. Sin embargo, la principal ventaja de este tipo de sistemas es la facilidad de la programación de implementaciones algorítmicas.

- **Multiprocesadores de memoria distribuida** (Figura 1.2): cada procesador dispone de una unidad de memoria local, no accesible a los demás procesadores y cuya comunicación se realiza por medio de una red de interconexión, mediante mecanismos de paso de mensajes entre los distintos procesadores involucrados en la computación. En este tipo de sistema la programación se realiza intercalando operaciones de

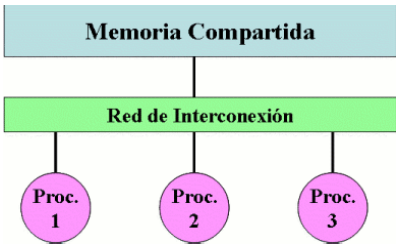

Figura 1.1: Memoria compartida.

computación con operaciones de comunicación, las cuales permiten la sincronización entre los diferentes procesadores empleados en la resolución del problema común. Además, en este tipo de sistema, los datos del problema están repartidos por las unidades de memoria local de los distintos procesadores.

La gran ventaja de los multiprocesadores de memoria distribuida es la escalabilidad, pues este tipo de sistemas son fáciles de escalar mientras que la demanda de los recursos crece, se puede agregar más memoria y procesadores. Sin embargo, las principales desventajas son que el acceso remoto a la memoria es lento y que la programación puede ser difícil.

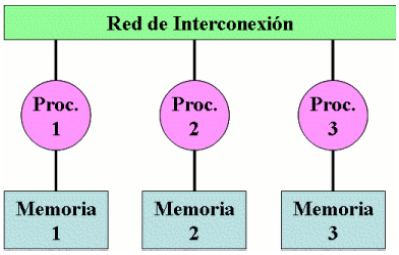

Figura 1.2: Memoria distribuida.

Aunque hayamos destacado dos tipos de clasificación para las arquitecturas MIMD, se pueden también encontrar arquitecturas que usan clasificación mixta de las dos anteriores. Son los llamados:

- **Multiprocesadores de memoria compartida distribuida** (Figura 1.3): cada procesador dispone de una unidad de memoria local, pero sin un canal compartido, o sea, los procesadores tienen acceso a una memoria compartida y se interconectan con otros procesadores por medio de un dispositivo de alta velocidad. Todos ven las memorias de cada uno como un espacio de direcciones globales.

  Las principales ventajas de este tipo de sistemas son el hecho de que presenten una escalabilidad como en los sistemas de memoria distribuida, su fácil programación como en los sistemas de memoria compartida y la no existencia del *cuello de botella* que se puede dar en arquitecturas de sólo memoria compartida.

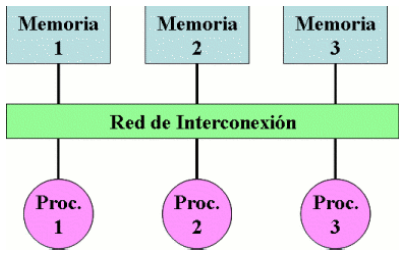

Figura 1.3: Memoria compartida distribuida.

Los resultados presentados en esta tesis han sido obtenidos en los recursos computacionales ofrecidos por el Grupo de Redes y Computación de Altas Prestaciones (GRyCAP, [webGRyCAP, 2014]) de la Universitat Politècnica de València (UPV, [webUPV, 2014]), y por el consorcio SEARCH[1] [webSearch, 2014] de la Universidad del Minho (UM, [webUM, 2014]):

[1] SEARCH: *Services and Advanced Research Computing with HTC/HPC clusters.*

- **Odin** (`odin.itaca.upv.es`): cluster de la UPV con 51 nodos biprocesadores Inter(R) Xeon(TM) CPU a 2.8GHz, interconectados mediante una red SCI (*Scalable Coherent Interface*) con topología de Toro 2D en malla de 10×5. 50 nodos están disponibles para cálculo científico y cada nodo consta de un disco duro IDE de 80GB y 2GB de memoria RAM. El nodo principal o *front-end* es el punto de acceso al cluster y consta de 4 discos duros Ultra160 SCSI de 20GB, en un total de 80GB de disco duro y 4GB de memoria RAM.

  En [webOdin, 2014] se puede encontrar una descripción más detallada del cluster donde están descritas sus prestaciones mediante los *benchmarks* [Petitet *et al.*, 2008] e [Intel, 2008], así como del software instalado y algunas directivas de compilación y de ejecución.

- **Kefren** (`kefren.dsic.upv.es`): cluster de la UPV con 19 nodos biprocesadores Pentium Xeon a 2GHz, interconectados mediante una red SCI con topología de Toro 2D en malla de 4×4. 15 nodos están disponibles para cálculo científico y constan de un disco duro IDE de 40GB a 7200rpm y 1GB de memoria RAM. El nodo principal consta de 4 discos duros Ultra160 SCSI de 36GB a 10000rpm y 1GB de memoria RAM. Los restantes 3 nodos para cálculo científico están configurados para la ejecución de trabajos secuenciales [webKefren, 2014].

- **Search** (`search1.di.uminho.pt`): cluster de la UM con 46 nodos dual Xeon a 3.2GHz, interconectados mediante una red Myrinet-10G [webMyrinet, 2009] y una red Gigabit ethernet [webGigabit, 2014]. Además, tiene 8 nodos GPU y 6 nodos entrada/salida interconectados a una red de fibra SAN a 3TB [webSearch, 2014].

1.4.2. Librerías de entornos paralelos

Para sacar el máximo rendimiento de las arquitecturas empleadas en el desarrollo de la tesis se han utilizado librerías específicas que explotan las características computacionales de las arquitecturas anteriormente nombradas.

En este contexto, las librerías empleadas son de dos tipos distintos: **librerías de soporte** y **librerías de cálculo científico** [Akl, 1989].

Librería de soporte

Para la programación en entornos paralelos existen distintas librerías de soporte. En el marco de la tesis se ha empleado el **MPI** (*Message Passing Interface* o **Interfaz de Paso de Mensajes**) [Pacheco, 1997], [Pacheco, 1998], [webMPI, 2014].

MPI es una interfaz de subrutinas que definen un estándar de comunicaciones mediante paso de mensajes [Gropp *et al.*, 1994], [Snir *et al.*, 1996]. En 1994, MPI fue formalmente establecido por el *MPI Forum*, cuyas características más importantes son:

1. Comunicaciones punto a punto y comunicaciones colectivas.
2. Las comunicaciones pueden ser síncronas y asíncronas, bloqueantes o no, con o sin búfer.
3. Posibilidad de definir contextos de comunicación para las operaciones colectivas.
4. Posibilidad de solapamiento entre comunicaciones y computaciones.
5. Posibilidad de definir nuevos tipos de datos.
6. Posibilidad de definir topologías de procesos virtuales.
7. Facilidad de utilización y fuerte robustez.

Sin embargo, hay que tener en cuenta que MPI es solamente una especificación y, por lo tanto, es necesario una implementación del estándar para emplear sus funcionalidades. Una de las implementaciones del MPI más difundidas es el MPICH [Gropp y Lusk, 1996], [Gropp *et al.*, 1996], desarrollado en los Estados Unidos en el Laboratorio Nacional Argonne.

Además, existen implementaciones del MPI más específicas para la optimización de las comunicaciones en plataformas computacionales con redes de interconexión con características propias. Por ejemplo, *Scali MPI Connect* es una implementación del MPI de la empresa Scali [webScali, 2010], optimizada para la red SCI empleada en los clusters de la UPV anteriormente nombrados.

Librerías de cálculo científico

Las **librerías de cálculo científico** (LCC) tienen por objetivo sacar el máximo rendimiento de las arquitecturas computacionales y suelen estar optimizadas para cada arquitectura concreta.

Las LCC ponen a disposición un conjunto de rutinas que suelen resolver problemas comunes, cuyas soluciones muchas veces se centran en problemas de Álgebra Lineal, aumentando la posibilidad de reutilizar implementaciones ya existentes que fueran testadas y verificadas.

El diseño de las LCC debe perseguir los siguientes objetivos:

- **Portabilidad**: es la facilidad con la que el código fuente se adapta para ser compilado y ejecutado en un nuevo entorno computacional. En ese sentido, está ligada al uso de librerías cuya implementación esté disponible en distintas plataformas, así como que la interfaz sea común con otras implementaciones.

- **Altas prestaciones**: para aprovechar al máximo los recursos de la arquitectura y aumentar la rapidez de las ejecuciones, el código fuente debe estar diseñado para la utilización de algoritmos orientados al uso de los núcleos básicos disponibles en la librería. Es habitual que el fabricante o implementador de la librería provea su optimización para cada arquitectura concreta.

- **Legibilidad** y **robustez**: la utilización de las rutinas de la librería logra un código fuente que permite sustituir múltiples líneas por llamadas a un número reducido de funciones, permitiendo una simplificación del código fuente y aumentando su legibilidad, ya que gran parte de las operaciones las realiza la librería de manera transparente al programador. Esto permite, a la vez, obtener un código fuente mucho más robusto (se delega el tratamiento de errores en la librería).

- **Flexibilidad** y **extensibilidad**: consiste en la capacidad del usuario para poder controlar parámetros que influyen en la ejecución o de sustituir partes del código fuente, además de permitir extender la funcionalidad de la rutina, manteniendo la interfaz de uso lo más intacta posible.

Seguidamente se describen de manera resumida las LCC que se han utilizado en el marco de la tesis:

- **BLAS** (*Basic Linear Algebra Subprograms*)[2]

  La librería BLAS está formada por un conjunto de implementaciones de referencia para realizar las operaciones básicas con vectores y con matrices, y proporciona una especificación para esas operaciones de Álgebra Lineal [Lawson *et al.*, 1979], [Dongarra *et al.*, 1988], [Dongarra *et al.*, 1990], [Blas Forum, 2001], [webBLAS, 2011].

- **LAPACK** (*Linear Algebra Package*)[2]

  La librería LAPACK proporciona un conjunto de rutinas, codificadas en Fortran 77, para resolver problemas de Álgebra Lineal. Las rutinas de LAPACK permiten resolver sistemas de ecuaciones lineales, problemas de mínimos cuadrados, problemas de valores propios y problemas de valores singulares. Además, LAPACK consta también de un conjunto de rutinas para el cálculo de algunas descomposiciones matriciales (LU, Cholesky, QR, Schur,...) y la estimación de números de condición [Anderson *et al.*, 1999], [webLAPACK, 2013].

- **SLICOT** (*Subroutine Library In Control Theory*)

  La librería SLICOT ofrece un conjunto de rutinas codificadas en Fortran 77, dedicadas a algoritmos numéricos de sistemas y teoría de control. Las rutinas de SLICOT están basadas en rutinas de BLAS y de LAPACK, y proporcionan un conjunto de métodos para el diseño y análisis de sistemas de control [Benner *et al.*, 1997], [van Huffel y Sima, 2002], [webSLICOT, 2014].

- **SCALAPACK** (*Scalable Linear Algebra Package*)[2]

  La librería SCALAPACK es una librería estándar dedicada a resolver problemas de Álgebra Lineal en multiprocesadores de memoria distribuida y en cualquier plataforma que permita mecanismos de paso de mensajes. La librería proporciona una gran parte de las funcionalidades de LAPACK y permite solucionar los mismos problemas que LAPACK, pero

[2] Para más información, consultar el Apéndice "BLAS, LAPACK y SCALAPACK".

en paralelo: resolución de sistemas de ecuaciones lineales, problemas de mínimos cuadrados, problemas de valores propios y problemas de valores singulares [Choi *et al.*, 1992], [Choi *et al.*, 1996b], [Blackford *et al.*, 1997], [webSCALAPACK, 2012].

- **BLACS** (*Basic Linear Algebra Communication Subprograms*)[3]

  La librería BLACS está compuesta por un conjunto de rutinas de paso de mensajes, diseñadas para Álgebra Lineal. Las rutinas están basadas en la utilización de mallas de procesadores unidimensionales o bidimensionales, en las cuales se usan múltiples operaciones de comunicación de matrices y/o vectores.

  La librería pretende proporcionar la misma facilidad de uso y de portabilidad que el BLAS ofrece para la computación en Álgebra Lineal, pero considerando las comunicaciones necesarias en arquitecturas con memoria distribuida [Dongarra, 1991], [Dongarra y van de Geijn, 1993], [Whaley, 1994], [Dongarra y Whaley, 1995], [webBLACS, 1997].

- **PBLAS** (*Parallel Basic Linear Algebra Subprograms*)

  La librería PBLAS proporciona las rutinas equivalentes a las de BLAS, pero para realizar las mismas operaciones en paralelo. La librería ofrece un estándar de operaciones matriciales para memoria distribuida, al igual que BLAS lo ha hecho para memoria compartida [Choi *et al.*, 1996c], [webPBLAS, 1995].

  PBLAS incluye solamente una rutina más que BLAS. Esa rutina está dedicada al cálculo de la transpuesta de una matriz, que es una operación matricial no trivial en un entorno de memoria distribuida [Choi *et al.*, 1995].

  En PBLAS todas las comunicaciones son dejadas a la responsabilidad de BLACS, permitiendo que todos los aspectos de comunicación queden ocultos dentro de la implementación de la librería, siendo por tanto, muy transparente su utilización como ocurre con BLAS.

Las librerías LAPACK y SCALAPACK, como es natural, son desarrolladas según los fundamentos de las LCC [Demmel *et al.*, 1991], [Demmel *et al.*, 1993], [Dongarra y Walker, 1993], [Choi *et al.*, 1996b].

[3] Para más información, consultar el Apéndice "BLAS, LAPACK y SCALAPACK".

Hay que señalar que todas las implementaciones desarrolladas en el marco de la tesis han sido codificadas en Fortran 90 [webFortran90, 2014], aunque todas las rutinas de BLAS, de LAPACK, de PBLAS, de SCALAPACK y de SLICOT empleadas sean rutinas codificadas en Fortran 77.

1.5. Indicadores de prestaciones

Para poder concluir que una implementación algorítmica es mejor que otra en la resolución del mismo problema, hay que valorar ambas implementaciones según los mismos criterios y hacer una comparación realista entre ellas.

Habitualmente los criterios de comparación incluyen referencias al tiempo de ejecución, a las propiedades numéricas del algoritmo, a los requisitos de memoria, entre otros. Sin embargo, hay un aspecto fundamental en el desarrollo de implementaciones algorítmicas: la implementación debe estar adecuada a la arquitectura física donde va a ser ejecutada. Si esto es esencial para las implementaciones secuenciales, entonces es aún más verdadero para las implementaciones paralelas.

Muchas veces en la valoración de las implementaciones paralelas no se tienen en consideración aspectos como el tiempo que lleva desarrollar la implementación paralela de manera eficiente o el tiempo invertido en proporcionar una versión paralela de un algoritmo secuencial eficiente.

Hay que tener en cuenta que ni todos los algoritmos secuenciales tienen una versión paralela de fácil implementación ni tampoco deben ser paralelizados, pues ya son implementaciones algorítmicas con altas prestaciones. La computación paralela sólo debe ser aplicada cuando hay claras ventajas en su aplicación.

Como fue descrito al inicio de esta sección, las implementaciones algorítmicas y en particular las implementaciones paralelas, deben ser valoradas mediante indicadores de prestaciones específicos.

A continuación se describen algunos de esos indicadores de prestaciones: **tiempo de ejecución**, **velocidad**, **aceleración de ejecución**, **eficiencia** y **escalabilidad** [Kumar *et al.*, 1994], [Foster, 1995], [Golub y van Loan, 1996]. Los indicadores de prestaciones anteriormente nombrados dependen de la dimensión del problema a resolver (m) y del número de procesadores empleados en su resolución (p).

1.5.1. Tiempo de ejecución

Para valorar una implementación algorítmica el **tiempo de ejecución** es el indicador más intuitivo.

En las implementaciones secuenciales, T_{sec} es el tiempo transcurrido desde que se inicia su ejecución hasta que termina. Así, para las implementaciones secuenciales que resuelven el mismo problema de manera eficiente y correcta, se puede decir que una implementación es mejor que otra en la medida que tenga un tiempo de ejecución menor.

Para las implementaciones paralelas, T_{par} es el tiempo transcurrido desde que el primero de los procesadores empieza a trabajar hasta que el último de los procesadores termina su ejecución.

Normalmente, en las implementaciones paralelas, los tiempos de ejecución no suelen incluir el tiempo empleado en la distribución de los datos por los distintos procesadores ni tampoco el tiempo empleado en la recogida de los resultados. Los tiempos anteriores pueden ser incluidos en la valoración del tiempo de ejecución de la implementación paralela con el objetivo de tener una valoración más realista del algoritmo, aunque en la mayoría de los casos se asume que los datos de entrada proceden de un algoritmo paralelo anterior y están distribuidos de la misma manera.

En el mejor de los casos, el tiempo de ejecución de una implementación paralela ejecutada con p procesadores es p veces inferior al tiempo de ejecución en un solo procesador, considerando que todos los procesadores tienen igual potencia de cálculo. Sin embargo, no siempre la reducción del tiempo de ejecución de la implementación paralela es de factor p, pues hay que tener en cuenta otros aspectos, como por ejemplo, los tiempos gastados en la sincronización de las comunicaciones, o los tiempos en que los procesadores están ociosos, entre otros.

En general, el tiempo de ejecución de una implementación paralela en memoria distribuida se puede expresar por [Kumar *et al.*, 1994], [Foster, 1995], [Golub y van Loan, 1996],

$$T_{par}\left(m,p\right) = T_{Comp}\left(m,p,t_{pf}\right) + T_{Comunic}\left(m,p\right) \tag{1.4}$$

donde:

- m − es la dimensión del problema a resolver;

- p – es el número de procesadores empleados en la resolución del problema;
- T_{Comp} – es el **tiempo de computación** y mide el tiempo en que los procesadores están ocupados en realizar cálculos computacionales;
- t_{pf} – es el tiempo de ejecución de una operación en punto flotante;
- $T_{Comunic}$ – es el **tiempo de comunicación** y mide el tiempo en que los procesadores están ocupados en realizar operaciones de envío y de recepción de mensajes.

Dependiendo de la arquitectura física empleada para ejecutar la implementación paralela, los tiempos T_{Comp} y $T_{Comunic}$ pueden solaparse o no, dependiendo de si las comunicaciones son asíncronas o no.

Para reducir el tiempo T_{par} es deseable que se puedan solapar las comunicaciones con los cálculos computacionales. Este solapamiento puede ser global, y en ese caso el tiempo T_{par} es el mayor de ellos, o parcial, y en ese caso el tiempo T_{par} es inferior a la suma (1.4).

En el modelo de paso de mensajes, el tiempo de comunicación ($T_{Comunic}$) de un mensaje de dimensión m entre dos procesadores vecinos se expresa como [Saad y Schultz, 1989], [Kumar *et al.*, 1994], [Foster, 1995], [Golub y van Loan, 1996], [El-Rewini y Lewis, 1998], [Dongarra *et al.*, 1998],

$$T_{Comunic}\,(m,p) = \alpha + m\tau \tag{1.5}$$

donde:

- α – es la **latencia** y mide el tiempo necesario para establecer la comunicación entre los procesadores;
- τ – es el **tiempo** de transferencia de un **mensaje de longitud 1**, desde el procesador emisor hasta el procesador receptor.

Además de la expresión (1.5), $T_{Comunic}$ también se puede expresar como

$$T_{Comunic}\,(m,p) = \alpha + \frac{m}{\beta} \tag{1.6}$$

donde β es el **ancho de banda**.

Como se observa, el tiempo de comunicación de un mensaje entre dos procesadores vecinos es una función lineal que depende de la longitud del mensaje.

Haciendo un análisis del tiempo (1.5), se observa que la latencia tiene una importancia elevada en la comunicación de mensajes de dimensión reducida. De la misma manera, la latencia va perdiendo importancia a medida que aumenta la dimensión de los mensajes a comunicar. En general, el valor de α es más significativo que el valor de τ y, por lo tanto, el número de comunicaciones empleadas en una implementación paralela debe ser lo más reducido posible [Kumar *et al.*, 1994], [Foster, 1995].

1.5.2. Velocidad

Para valorar el coste de un determinado algoritmo se suele calcular el número total de operaciones en coma flotante que necesita en su ejecución. A esta medida se denomina **flops** (*Floating Point Operations Per Second* u **Operaciones de Coma Flotante Por Segundo**) [Golub y van Loan, 1996, §1.2.4].

Otro indicador de prestaciones que permite valorar la potencia de cálculo de un sistema computacional concreto es la **velocidad** y esta medida se obtiene como el cociente entre el coste teórico del algoritmo (en flops) y el coste temporal del algoritmo (en segundos).

Cuanto mayor es la velocidad obtenida en la ejecución de la implementación de un algoritmo, mejores son sus prestaciones, pues se han realizado más operaciones en coma flotante por unidad de tiempo.

En algoritmos paralelos se suele distinguir entre la **velocidad total**, calculada entre el coste teórico y el coste temporal del algoritmo, y la **velocidad por nodo**, obtenida como el cociente entre los flops totales y el número de procesadores empleados en la ejecución del algoritmo paralelo. La velocidad por nodo permite establecer el número de operaciones en coma flotante por segundo que han sido realizadas en cada procesador, respecto a los flops totales del algoritmo paralelo.

1.5.3. Aceleración de ejecución o *speed-up*

La **aceleración de ejecución**, habitualmente conocida por *speed-up*, mide la ganancia de velocidad de ejecución del algoritmo paralelo respecto al mejor

algoritmo secuencial, que resuelve el mismo problema [Kumar *et al.*, 1994], [Foster, 1995], [Golub y van Loan, 1996].

El *speed-up* para p procesadores (S_p) se obtiene con el cociente entre el tiempo de ejecución del algoritmo secuencial (T_{sec}) y el tiempo de ejecución del algoritmo paralelo en p procesadores (T_{par}). Además, ambos tiempos de ejecución dependen de la dimensión del problema (m). Así, S_p se expresa como

$$S_p(m) = \frac{T_{sec}(m)}{T_{par}(m,p)}. \tag{1.7}$$

Sin embargo, para medir T_{sec}, el tiempo de ejecución de una implementación en un solo procesador, se puede optar por medir T_{sec} según dos perspectivas válidas:

1. T_{sec} es el tiempo de ejecución de la implementación del mejor algoritmo secuencial, ejecutada secuencialmente. En este caso, el *speed-up* es valorado según (1.7).

2. T_{sec} es el tiempo de ejecución de la implementación paralela, ejecutada en un solo procesador. En este caso, el *speed-up* es valorado según

$$S_p(m) = \frac{T_{par}(m,1)}{T_{par}(m,p)}. \tag{1.8}$$

La valoración (1.8) permite tener una idea más concreta respecto a la implementación paralela y cual es su aceleración de ejecución a medida que se incrementa el número de procesadores, considerando que el tiempo $T_{par}(m,1)$ es valorado en la misma configuración de procesadores que el tiempo $T_{par}(m,p)$.

En el caso en que la implementación paralela no es la versión paralela del mejor algoritmo secuencial, sigue teniendo utilidad la valoración de la aceleración de ejecución según (1.7). En este caso, al valor del *speed-up* se denomina *speed-up* **algorítmico**.

En el caso ideal, al implementar la versión paralela del mismo algoritmo secuencial, T_{par} debe ser una función que varia linealmente y en proporción con el incremento del número de procesadores. En este caso, al valor del *speed-up* se denomina *speed-up* **lineal**.

En la mayoría de las implementaciones paralelas, el valor del *speed-up* suele ser inferior al número de procesadores ($S_p(m) \leq p$). A este tipo de *speed-up* se denomina *speed-up* **sub-lineal**. Sin embargo, puede ocurrir que el valor del *speed-up* sea superior al número de procesadores. Es el llamado *speed-up* **súper--lineal**. En este caso, puede, por ejemplo, ocurrir que al incrementar el número de procesadores empleados en la resolución del problema, haya una distribución más favorable de los datos y/o del trabajo entre los distintos procesadores de la arquitectura. Según los autores Chalmers y Tidmus [Chalmers y Tidmus, 1996], por ejemplo, puede ocurrir que al repartir los datos del problema entre dos procesadores, estos sean almacenados totalmente en la memoria caché del procesador, mientras que al usar un solo procesador parte de los datos tienen que ser almacenados en alguna memoria externa.

1.5.4. Eficiencia

La **eficiencia** mide el grado de aprovechamiento de los procesadores y es el cociente entre el *speed-up* y el número de procesadores [Kumar *et al.*, 1994], [Foster, 1995],

$$E(m,p) = \frac{S_p(m)}{p}. \tag{1.9}$$

Según (1.7) y (1.8), la eficiencia se puede expresar por

$$E(m,p) = \frac{T_{sec}(m)}{pT_{par}(m,p)} \tag{1.10}$$

o por

$$E(m,p) = \frac{T_{par}(m,1)}{pT_{par}(m,p)}. \tag{1.11}$$

Idealmente, se espera que el valor del *speed-up* sea igual al número de procesadores, donde la eficiencia ideal es igual a 1.

1.5.5. Limitaciones del *speed-up*

El *speed-up* y la eficiencia son indicadores de prestaciones para valorar las implementaciones paralelas. Estos dos indicadores de prestaciones permiten únicamente valorar cuan bien el algoritmo secuencial fue paralelizado,

teniendo en cuenta que estamos hablando de la versión paralela del mismo algoritmo secuencial. Además, si la arquitectura física empleada tiene p procesadores, entonces

$$S_p(m) \leq p \qquad \text{y} \qquad E(m,p) \leq 1. \tag{1.12}$$

Muchas veces el valor del *speed-up* es limitado por la existencia de fracciones de la implementación secuencial que no pueden ser paralelizadas.

Si μ es la fracción de la implementación *naturalmente secuencial* y $1-\mu$ es la fracción de la implementación *naturalmente paralela*, entonces la parte *naturalmente secuencial* es ejecutada en un tiempo de ejecución igual a $\mu T_{par}(m,1)$, mientras que la parte *naturalmente paralela* es ejecutada en un tiempo de ejecución igual a

$$\frac{(1-\mu)\,T_{par}(m,1)}{p}. \tag{1.13}$$

En este caso, el tiempo de ejecución de la implementación paralela empleando p procesadores es

$$T_{par}(m,p) = \mu T_{par}(m,1) + \frac{(1-\mu)\,T_{par}(m,1)}{p} \tag{1.14}$$

y, por tanto, la **ley de Amdahl** afirma que el valor del *speed-up* está limitado por la existencia de partes de la implementación paralela que son *puramente secuenciales* [Amdahl, 1967], [Foster, 1995]:

- **Ley de Amdahl**: *si* $0 \leq \mu \leq 1$ *es la parte de la implementación paralela que es puramente secuencial, entonces el valor del speed-up obtenido por la implementación paralela con p procesadores es limitado por*

$$S_p(m) \leq \frac{p}{\mu p + (1-\mu)} \leq \frac{1}{\mu}. \tag{1.15}$$

La ley de Amdahl es bastante pesimista y pone un límite superior al valor del *speed-up*, sin embargo, transmite también una idea importante: el rendimiento no aumenta por el hecho de incrementarse indefinidamente el número de procesadores.

Sin embargo, la **ley de Gustafson** [Gustafson, 1988], viene a compensar el pesimismo dejado por la ley de Amdahl, pues en la práctica, la dimensión de un problema no es independente del número de procesadores, ya que con mayor número de procesadores se pueden resolver problemas de mayor dimensión. Por este motivo, la ley de Gustafson se refiere al crecimiento del volumen de cálculo necesario para resolver un problema dado, ya que, en la mayoría de los casos, cuando la dimensión de un problema crece, lo hace sólo en su parte paralelizable y no en su parte secuencial.

- **Ley de Gustafson**: *si* $0 \leq \mu \leq 1$ *es la parte de la implementación paralela que es puramente secuencial, entonces el valor del speed-up obtenido por la implementación paralela con p procesadores es limitado por*

$$S_p(m) \leq p + \mu(1-p). \tag{1.16}$$

Podría pensarse que hay una aparente contradicción entre las dos leyes anteriores, pero esto no es así debido al hecho de que las premisas de ambas leyes son distintas. La ley de Amdahl se refiere a procesos con un volumen de cálculo fijo mientras que la ley de Gustafson se refiere a problemas cuyo volumen de cálculo puede aumentar según el número de procesadores.

1.5.6. Escalabilidad

La **escalabilidad** es un indicador de prestaciones que permite valorar la capacidad de una implementación paralela de mantener constantes sus prestaciones al incrementar proporcionalmente el número de procesadores y la dimensión del problema [Kumar *et al.*, 1994], [Foster, 1995].

En otras palabras, la escalabilidad permite concluir si podremos aprovechar la potencia de cálculo aportada por el incremento del número de procesadores, aumentando de igual manera la dimensión del problema.

Aunque la escalabilidad se pueda valorar mediante métricas distintas [Kumar *et al.*, 1994], [Foster, 1995], es conveniente tener en cuenta las características de los problemas a resolver y elegir la métrica más adecuada [Martin y Tirado, 1997].

En esta tesis se ha utilizado la definición propuesta por [Kumar *et al.*, 1994]: *La escalabilidad de un sistema computacional paralelo es una medida de su*

capacidad para incrementar el speed-up proporcionalmente al número de procesadores (traducción nuestra).

1.6. Estructura de la memoria

A continuación se presenta una breve descripción de los restantes capítulos de la memoria.

En el capítulo 2 se introducen algunos conceptos básicos y propiedades elementales de la descomposición en valores singulares (DVS) de una matriz real. A continuación se describen algunos de los métodos tradicionales que reducen una matriz real a la forma bidiagonal (superior). Posteriormente se hace una descripción de los conceptos que permiten obtener la DVS de una matriz bidiagonal (superior) a partir del cálculo de la descomposición en valores propios (DVP) de una matriz tridiagonal simétrica. Seguidamente se introducen los conceptos del problema del cálculo de la DVS generalizada (DVSG) y, en particular, la descomposición producto (DVSP). La descripción del método de Golub-Sølna-van Dooren cierra el capítulo.

En el capítulo 3 se exponen los métodos de bidiagonalización propuestos por Ralha y por Barlow. A continuación se analiza el método que introduce modificaciones al método de Barlow con el objetivo de reducir el número de comunicaciones en la implementación paralela. Seguidamente se hace una descripción de las implementaciones secuenciales y paralelas de los métodos anteriores. A continuación se realiza un completo estudio comparativo entre las prestaciones obtenidas por las implementaciones secuenciales y paralelas desarrolladas en el entorno de las librerías numéricas LAPACK y SCALAPACK, y las correspondientes rutinas de estas librerías con la misma funcionalidad.

En el capítulo 4 se introducen algunos conceptos de la DVP de una matriz tridiagonal simétrica. Además, se describe el método de bisección, tradicionalmente empleado para calcular los valores propios de la matriz tridiagonal simétrica, y el método MRRR empleado para calcular los respectivos vectores propios. A continuación se describe el método `zeroinNR`, el cual es una variante del método `zeroin`, donde Ralha propone una nueva formulación al cálculo de la corrección del método de Newton-Raphson con el objetivo de garantizar que los valores propios de la matriz tridiagonal simétrica sean calculados con elevada precisión numérica. Seguidamente se hace una

descripción de las prestaciones obtenidas por algunas de las rutinas de LAPACK dedicadas al cálculo de la DVP de una matriz tridiagonal simétrica, y se hace un estudio comparativo entre las prestaciones obtenidas por una de las rutinas de LAPACK y la implementación secuencial del método `zeroinNR`.

En el capítulo 5 se hace una descripción del método alternativo para el cálculo de los valores propios de la matriz tridiagonal simétrica cuya implementación paralela no tiene comunicaciones. Este método es una variante del método `zeroinNR`, en la cual se empieza por emplear el método de bisección para calcular intervalos de números reales con un solo valor propio de la matriz y seguidamente se emplea una modificación del método de Newton-Raphson para calcular los valores propios con elevada precisión numérica. A continuación se realiza un estudio comparativo entre las prestaciones obtenidas por la implementación paralela del método anterior y la correspondiente rutina de SCALAPACK, en la cual, el método está implementado según [Demmel *et al.*, 1995], y además presenta comunicaciones.

En el capítulo 6 se introducen algunos conceptos básicos relacionados con la reducción de modelos de sistemas lineales de control y se hace una descripción del método para la reducción de modelos basada en realizaciones balanceadas. La descripción de algunos casos de prueba que fueron resueltos con la rutina de la librería SLICOT dedicada a la reducción de modelos de sistemas lineales de control cierra el capítulo.

En el capítulo 7 se describen las implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren, desarrolladas con las rutinas de LAPACK y de SCALAPACK. Seguidamente se analizan los resultados obtenidos, y a continuación se relatan los resultados obtenidos por las implementaciones del método, extensible al producto de cualquier número de matrices reales cuadradas.

Por último, en el capítulo 8 se presentan las conclusiones más importantes que se han logrado con todo el trabajo desarrollado. A continuación se ponen a la vista algunas de las líneas de investigación de trabajo futuro que proyecta la presente memoria. Finalmente, el capítulo cierra con la enumeración de los elementos de ámbito científico desarrollados en el marco de la tesis.

Capítulo **2**

La Descomposición en Valores Singulares

En este capítulo nuestro enfoque está dirigido a la **descomposición en valores singulares** (DVS) de matrices reales (densas).

El capítulo empieza con un resumen histórico de la DVS de una matriz. A continuación se exponen algunos conceptos y propiedades de esta importante descomposición matricial, así como las **transformaciones de Householder**, empleadas en el proceso de cálculo de la DVS.

Posteriormente se describen algunos de los métodos usuales para reducir una matriz real (densa) a la forma bidiagonal (superior), y como se puede obtener la DVS de esa matriz bidiagonal (superior), a partir del cálculo de la **descomposición en valores propios** (DVP) de una matriz tridiagonal simétrica.

Seguidamente se introducen los conceptos del problema del cálculo de la **DVS generalizada** (DVSG) y, en particular, de la **descomposición producto** (DVSP). Al terminar el capítulo se describe el **método de Golub--Sølna-van Dooren** [Golub *et al.*, 2000], para el cálculo de la matriz bidiagonal superior del producto implícito de dos matrices reales (cuadradas).

2.1. Un poco de la historia

El problema de calcular la DVS de una matriz ha sido uno de los temas importantes en la investigación del Álgebra Lineal Numérica debido a sus múltiples aplicaciones. De hecho, los métodos que calculan la DVS de una matriz son de los más antiguos del Álgebra Lineal y remontan al año 1873 [Stewart, 1993].

Además, debido a su estrecha relación, el cálculo de la DVS de una matriz ha acompañado a lo largo del tiempo el problema del cálculo de la DVP de matrices [van der Vorst y Golub, 1997].

En 1873, Beltrami fue el primero en presentar un método que permitía calcular la DVS de una matriz real cuadrada y no singular [Beltrami, 1873]. Según Stewart [Stewart, 1993], Beltrami trataba de motivar a sus alumnos en el estudio de las formas bilineales y fue a partir de esa definición que consiguió llegar al resultado que presentó.

En 1874, Jordan intentando solucionar el problema de reducir una forma bilineal a una forma diagonal, recurriendo a simples sustituciones ortogonales, consiguió llegar a un resultado semejante al de Beltrami.

Autonne, en 1913, realiza la adaptación del método existente al caso de las matrices cuadradas complejas [Autonne, 1913]. En 1936, Eckart y Young describen su extensión al caso de las matrices rectangulares [Eckart y Young, 1936], [Eckart y Young, 1939]. Sin embargo, es sólo a partir del año 1960 que el método empieza a ser utilizado como una herramienta de gran valor y de gran aplicación computacional.

Uno de los grandes pioneros de esta cruzada y que más ha contribuido para ese hecho, fue Golub, al cual se debe el esfuerzo inicial por intentar presentar un método que obtiene computacionalmente la DVS de una matriz real.

En 1965, Golub y Kahan presentaron por primera vez el **método de Golub--Kahan** para el cálculo de la DVS de una matriz real, donde describen de manera detallada el proceso para reducir la matriz inicial a la forma bidiagonal (superior) [Golub y Kahan, 1965].

El método de Golub-Kahan para el cálculo de la DVS de una matriz $A \in \mathbb{R}^{m\times n}$ consiste en dos etapas fundamentales:

1. La reducción de la matriz A a la forma bidiagonal (superior), empleando transformaciones de Householder.

2. El cálculo de la DVS de la matriz bidiagonal (superior).

Golub y Kahan demostraron que la etapa 1 tiene un coste computacional proporcional a mn^2. Además, en [Golub y Kahan, 1965] está demostrado la estabilidad numérica de la etapa 1, dado que los valores singulares de la matriz bidiagonal (superior) son distintos a los valores singulares de la matriz A, en cantidades no superiores a $p(n) \cdot \epsilon \cdot \|A\|_2$, donde $p(n)$ representa una función de crecimiento lento con n, ϵ representa la precisión de la máquina y $\|A\|_2$ representa la **norma espectral**[1] de la matriz A.

Aunque el método de Golub-Kahan haya sido presentado por primera vez en 1965, sólo en 1969 su primera implementación en Algol habría de ser desarrollada [Businger y Golub, 1965], [Businger y Golub, 1969]. Algol fue el lenguaje de programación empleado para desarrollar las implementaciones de las rutinas básicas de EISPACK[2], la cual es considerada como la primera librería que permitía calcular computacionalmente la DVS de una matriz [Smith *et al.*, 1976], [Garbow *et al.*, 1977].

La importancia del método de Golub-Kahan está en la etapa 1, donde por primera vez se presentó un método de bidiagonalización de matrices reales, que se suponen densas.

Para la etapa 2, en 1961, Francis [Francis, 1961] fue el primero en proponer que el **método QR** fuese utilizado con **traslaciones espectrales** o **desplazamientos de origen** con el objetivo de acelerar la convergencia. Wilkinson presentó una técnica de desplazamientos de origen aplicada al **método QR simétrico** para matrices tridiagonales simétricas que permitió convertirlo en un método mucho más eficiente [Wilkinson, 1968].

Sin embargo, en 1970, Golub y Reinsch emplearon la DVS en la resolución del problema de mínimos cuadrados, definiendo el **método de Golub-Reinsch**, lo que constituye una referencia importante en la historia de la DVS, pues por primera vez es expuesta una implementación del método de Golub-Kahan, el cual en la etapa 2 aplica implícitamente el método QR simétrico para calcular los valores propios de la matriz simétrica A^tA, que no es calculada de manera

[1] Se denomina **norma espectral** de una matriz A y se denota por $\|A\|_2$, al mayor valor singular de la misma.

[2] EISPACK (*Eigensystem Package*): considerada como la primera colección de rutinas computacionales, implementadas en Algol, descritas y publicadas por Wilkinson y por Reinsch en su libro *Handbook for Automatic Computation* ([Wilkinson y Reinsch, 1971]).

explícita, al cálculo de los valores singulares de una matriz A y explican cómo se aplica el proceso iterativo para calcular la DVS de la matriz bidiagonal, empleando los **desplazamientos de origen de Wilkinson** y las **rotaciones de Givens** [Golub y Reinsch, 1970].

Chan, en 1982, expone un método alternativo para el cálculo de la DVS de una matriz densa [Chan, 1982a], [Chan, 1982b]. Dada la matriz $A \in \mathbb{R}^{m\times n}$ (con $m \gg n$), el método empieza por reducir la matriz a la forma triangular superior $\begin{bmatrix} R^t & 0 \end{bmatrix}^t$, empleando una **descomposición QR**, y a continuación aplica el proceso de bidiagonalización a la matriz $R \in \mathbb{R}^{n\times n}$. De este modo, el **método de Chan** tiene un coste computacional más reducido que el método de Golub-Kahan, cuando es aplicado a matrices con $m \geq \frac{5}{3}n$.

En 1990, Demmel y Kahan demostraron por primera vez que los valores singulares de la matriz bidiagonal superior pueden ser calculados computacionalmente con errores relativos muy pequeños, independientemente de sus magnitudes [Demmel y Kahan, 1990].

A continuación fueron presentados muchos otros resultados para el cálculo de la DVS de una matriz: [Arbenz, 1989], [Fernando y Parlett, 1994], [Gu y Eisenstat, 1995a], [Li *et al.*, 1995], [Trefftz *et al.*, 1995], [Lang, 1996], [Demmel *et al.*, 1999b], [Fernando, 1998a], [Fernando, 1998b], [Großer y Lang, 1999], [Großer y Lang, 2003], [Drmač y Veselić, 2007a], [Drmač y Veselić, 2007b], [Willems *et al.*, 2006], [Howell *et al.*, 2008] y [Demmel *et al.*, 2009].

En 1987, Fernando y Hammarling presentaron por primera vez un estudio del problema de calcular la **DVS del producto** de dos matrices reales, basado en el producto AB^t de las matrices A y B [Fernando y Hammarling, 1987].

Sin embargo, la generalización de la DVS habría sido introducida en 1976, por van Loan, en su tesis doctoral [van Loan, 1976]. En este trabajo, van Loan llamó *B-singular value decomposition* a la DVS estándar de una matriz A asociada a la matriz B, donde B tiene el mismo número de columnas que A. Si acaso la matriz B es la matriz identidad, entonces la generalización de van Loan es la DVS de una matriz real.

En 1981, Paige y Saunders exponen una descripción más general para la formulación desarrollada por van Loan, la cual es actualmente designada como la **descomposición en valores singulares generalizada** (DVSG) [Paige y Saunders, 1981]. Además, en 1986, Paige y sus colaboradores emplearon el **método de Kogbetliantz** [Kogbetliantz, 1955], para calcular la **DVS del**

producto de dos matrices (DVSP), y en ese trabajo, Paige describe una propuesta para la generalización del método de Kogbetliantz para calcular directamente la DVSG [Heath *et al.*, 1986].

Más tarde, Bai y Demmel [Bai, 1992], [Bai y Demmel, 1993], Adams, Bojanczyk, Ewebring, Luk y van Dooren presentaron trabajos para mejorar el método de Paige [Bojanczyk *et al.*, 1991], [Adams *et al.*, 1994].

Posteriormente, Golub, Sølna y van Dooren presentaron un **método para la bidiagonalización implícita del producto/cociente** de dos o más matrices, obteniendo una matriz bidiagonal superior que posteriormente se diagonaliza empleando un proceso iterativo [Golub *et al.*, 2000].

En 1999, Claver, Mollar y Hernández presentaron una paralelización del método de Golub-Sølna-van Dooren [Claver *et al.*, 1999], la cual está descrita con detalle en la tesis doctoral de Mollar [Mollar, 2003].

2.2. Conceptos del problema estándar

Dada una matriz $A \in \mathbb{R}^{m\times n}$ (donde se puede asumir que $m \geq n$), los valores propios de la matriz cuadrada A^tA son todos no negativos y se denominan **valores singulares** de la matriz A a las raíces cuadradas de los valores propios de la matriz A^tA.

Teorema 1 (Descomposición en Valores Singulares) *Si* $A \in \mathbb{R}^{m\times n}$, *entonces existen dos matrices ortogonales*

$$U = \begin{bmatrix} u_1 & \dots & u_m \end{bmatrix} \in \mathbb{R}^{m\times m} \qquad y \qquad V = \begin{bmatrix} v_1 & \dots & v_n \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{2.1}$$

y una matriz "diagonal"

$$\Sigma = \begin{bmatrix} \widetilde{\Sigma} \\ 0 \end{bmatrix} \in \mathbb{R}^{m\times n} \qquad con \qquad \widetilde{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \sigma_p \end{bmatrix} \tag{2.2}$$

donde $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$ *y* $p = \min\{m, n\}$, *tales que*

$$A = U\Sigma V^t \tag{2.3}$$

expresión que se denomina ***descomposición en valores singulares*** *de la matriz A.*

DEMOSTRACIÓN: La demostración del teorema está en [Golub y van Loan, 1996]. ■

El teorema anterior garantiza que la DVS de una matriz existe siempre. Además, se puede asumir que la dimensión de la matriz es del tipo $m \times n$ con $m \geq n$, pues, en caso contrario, se aplica el teorema a la transpuesta de la matriz A. En la tesis vamos a asumir a partir de este momento que $m \geq n$ y, por tanto, $p = \text{mín}\{m, n\} = n$.

Al mismo tiempo, el teorema anterior también garantiza que los escalares

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0 \tag{2.4}$$

son únicos y son los **valores singulares** de la matriz A.

A las n primeras columnas de la matriz $U = \left[\begin{array}{ccc} u_1 & \ldots & u_m \end{array} \right] \in \mathbb{R}^{m \times m}$ se les denomina **vectores singulares por la izquierda** de la matriz A y a las n columnas de la matriz $V = \left[\begin{array}{ccc} v_1 & \ldots & v_n \end{array} \right] \in \mathbb{R}^{n \times n}$ se les denomina **vectores singulares por la derecha** de la matriz A. Estas dos clases de vectores verifican las condiciones

$$\left\{ \begin{array}{l} Av_j = \sigma_j u_j \\ A^t u_j = \sigma_j v_j \end{array} \right. \qquad \text{con} \qquad j = 1, \ldots, n. \tag{2.5}$$

A continuación se exponen algunas de las múltiples propiedades de la DVS, puntualizadas y demostradas en [Golub y van Loan, 1996].

Dada la matriz $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$, su DVS verifica las siguientes propiedades:

- $\|A\|_2 = \text{máx}\left\{\sqrt{\lambda}\text{: } \lambda \text{ es valor propio de } A^t A\right\} = \sigma_1$.
- $\|A\|_F = \sqrt{\sigma_1^2 + \ldots + \sigma_n^2}$, donde $\|A\|_F$ es la **norma matricial de Frobenius**[3].

[3] Se denomina **norma matricial de Frobenius** de una matriz A y se denota por $\|A\|_F$, al valor dado por $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$.

- Si A es una matriz invertible, entonces $\left\|A^{-1}\right\|_2 = \dfrac{1}{\sigma_n}$.
- El **número de condición**, $\kappa_2(A) = \left\|A^{-1}\right\|_2 \|A\|_2 = \dfrac{\sigma_1}{\sigma_n}$.
- Si $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_n = 0$, entonces:
  - $rango(A) = r$.
  - $Ker(A) = span\{v_{r+1}, v_{r+2}, \ldots, v_n\}$[4].
  - $Im(A) = span\{u_1, u_2, \ldots, u_r\}$[5].
  - $Ker(A^t) = span\{u_{r+1}, u_{r+2}, \ldots, u_m\}$.
  - $Im(A^t) = span\{v_1, v_2, \ldots, v_r\}$.
- Si $k < r = rango(A)$ y $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^t$, entonces:
  - $\min\limits_{rango(B)\leq k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$.
  - $\min\limits_{rango(B)\leq k} \|A - B\|_F = \|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}$.
- $AA^t = U\Sigma\Sigma^t U^t$, donde $\Sigma\Sigma^t = diag\left(\sigma_1^2, \ldots, \sigma_n^2, \underbrace{0, \ldots, 0}_{m-n}\right)$.

[4]Se denomina **subespacio vectorial real de $\mathbb{R}^m$ generado por los vectores** $v_1, \ldots, v_n$ y se denota por $span\{v_1, \ldots, v_n\}$, al subespacio vectorial real definido por

$$span\{v_1, \ldots, v_n\} = \left\{ \sum_{k=1}^{n} \beta_k v_k : \beta_k \in \mathbb{R} \right\}.$$

Se denomina **núcleo de una matriz** A de dimensión $m \times n$ y se denota por $Ker(A)$, al subespacio vectorial real definido por $Ker(A) = \{x \in R^n : Ax = 0\}$.

[5]Se denomina **imagen de una matriz** A de dimensión $m \times n$ y se denota por $Im(A)$, al subespacio vectorial real definido por $Im(A) = \{y \in R^m : y = Ax \text{ para algún } x \in R^n\}$.

- $A^tA = V\Sigma^t\Sigma V^t$, donde $\Sigma^t\Sigma = diag\left(\sigma_1^2, \ldots, \sigma_n^2\right)$.

- La **pseudo-inversa**, $A^+ = V\Sigma^+U^t$,

  donde $\Sigma^+ = diag\left(\dfrac{1}{\sigma_1}, \ldots, \dfrac{1}{\sigma_r}, 0, \ldots, 0\right) \in \mathbb{R}^{n\times m}$ y $r = rango\,(A)$.

- Si E es una matriz de dimensión $m \times n$ $(m \geq n)$, entonces:

  - $|\sigma_j\,(A+E) - \sigma_j\,(A)| \leq \|E\|_2 \qquad (j = 1, \ldots, n)$.

  - $\sum_{j=1}^{n} \left(\sigma_j\,(A+E) - \sigma_j\,(A)\right)^2 \leq \|E\|_F^2$.

Para calcular la DVS de una matriz real, los métodos tradicionales empiezan por reducir la matriz a una forma bidiagonal, habitualmente a la forma bidiagonal superior, cuando se asume que el número de filas de la matriz es mucho mayor que su número de columnas, empleando una secuencia de sucesivas transformaciones de Householder [Golub y Kahan, 1965], [Golub y Reinsch, 1970], [Chan, 1982a], [Chan, 1982b], [Demmel *et al.*, 1999b], y posteriormente emplean un proceso iterativo para calcular los valores singulares de la matriz bidiagonal superior que coinciden con los valores singulares de la matriz inicial [Golub y Reinsch, 1970],
[Fernando y Hammarling, 1987], [Arbenz, 1989], [Demmel y Kahan, 1990], [Gu y Eisenstat, 1995a].

Considerando que uno de los objetivos de la tesis es estudiar dos métodos alternativos para la reducción de matrices densas a la forma bidiagonal superior, vamos a dirigir nuestro interés a la primera etapa del método de Golub--Kahan.

Dada una matriz $A \in \mathbb{R}^{m\times n}$, su reducción a la forma bidiagonal superior consiste en el cálculo de la matriz $B \in \mathbb{R}^{n\times n}$ y de las dos matrices ortogonales

$$U_B \in \mathbb{R}^{m\times m} \qquad \text{y} \qquad V_B \in \mathbb{R}^{n\times n} \tag{2.6}$$

tales que

$$A = U_B \begin{bmatrix} B \\ 0 \end{bmatrix} V_B^t \qquad \text{con} \qquad B = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & 0 & 0 & \alpha_n \end{bmatrix} \tag{2.7}$$

donde

$$U_B = U_1 U_2 \dots U_n \qquad \text{y} \qquad V_B = V_1 V_2 \dots V_{n-2} \tag{2.8}$$

son productos de **matrices de Householder**.

A continuación se van a exponer algunos conceptos de las **transformaciones** o **reflexiones de Householder**.

2.2.1. Transformaciones de Householder

La sección anterior describe que las matrices ortogonales tienen un papel importante en el cálculo de la DVS de una matriz real. Estas matrices ortogonales son, en general, **matrices de rotaciones** o **matrices de reflexiones**, pues desde el punto de vista computacional, las rotaciones y las reflexiones son operaciones muy atractivas. Escogiendo muy bien los ángulos de rotación o los planos de reflexión, se pueden introducir ceros en las componentes de un determinado vector real no nulo.

Seguidamente se presentan las **reflexiones de Householder**, conocidas también por **transformaciones de Householder**.

Dado un vector no nulo $v \in \mathbb{R}^n$, se denomina **matriz de Householder** o **matriz de reflexión de Householder**, a la matriz $H \in \mathbb{R}^{n \times n}$ definida por

$$H = I - \frac{2}{v^t v} v v^t \tag{2.9}$$

donde I es la matriz identidad de orden n. Al vector v se le denomina **vector de Householder** y se obtiene a partir de un vector x en el que se quieren hacer ceros excepto en el primer elemento, como se explica a continuación.

Es fácil de demostrar que la matriz de Householder:

- Es simétrica.

- Es ortogonal.
- Preserva la 2-norma de los vectores.
- Refleja cualquier vector no nulo $x \in \mathbb{R}^n$ en el hiper-plano $span\{v\}^{\perp}$.

Además, las matrices de Householder ($H \in \mathbb{R}^{n\times n}$) son actualizaciones de rango uno de la matriz identidad y para cualquier vector no nulo $x \in \mathbb{R}^n$, H se construye de manera que Hx tiene la misma dirección que el vector $e_1 = (1, 0, \ldots, 0)^t \in \mathbb{R}^n$, pues para $v \in \mathbb{R}^n$, se tiene

$$Hx = \left(I - \frac{2}{v^t v} v v^t\right) x = x - \frac{2}{v^t v}\left(v v^t\right) x = x - \frac{2 v^t x}{v^t v} v \tag{2.10}$$

y, por lo tanto, $Hx \in span\{e_1\}$ implica que $v \in span\{x, e_1\}$.

La relación (2.10) permite concluir que el vector de Householder v se puede definir como

$$v = x \pm \|x\|_2 e_1. \tag{2.11}$$

En la expresión anterior, el vector de Householder puede ser calculado de dos maneras distintas. Para evitar la aparición de cancelaciones catastróficas (cuando $\|x\|_2 \approx x(1)$), es habitual definir el vector de Householder como

$$v = x + sign[x(1)] \|x\|_2 e_1 \tag{2.12}$$

donde $x(1)$ representa la primera componente del vector x.

Adicionalmente, (2.12) garantiza que $\|v\|_2 \geq \|x\|_2$ y asegura una muy buena ortogonalidad en el cálculo computacional de la matriz H. Por convención se admite que $sign[x(1)] = 1$ cuando $x(1) = 0$.

En 1971, Parlett demostró que definiendo la primera componente del vector de Householder como

$$v(1) = x(1) - \|x\|_2 = \frac{-\left[x(2)^2 + \ldots + x(n)^2\right]}{x(1) + \|x\|_2} \tag{2.13}$$

es posible evitar las cancelaciones catastróficas, cuando $x(1) > 0$ y además, el vector de Householder tiene la propiedad de que Hx es siempre un múltiplo positivo de e_1 [Parlett, 1971].

Dado un vector no nulo $x \in \mathbb{R}^n$, la rutina descrita a continuación (en código Matlab), calcula el vector de Householder $v \in \mathbb{R}^n$, donde $v(1) = 1$, $\beta = 2/v^t v$ y $H = I - \beta v v^t$, verificando $Hx = \|x\|_2 e_1$. Esta es la implementación del algoritmo 5.1.1 de [Golub y van Loan, 1996, pp. 210].

```
Function [v, beta] = HouseholderVector( x )
   n = length( x );
   sigma = x(2:n)' * x(2:n);
   v = [1.0; x(2:n)];
   if (sigma == 0.0)
      beta = 0.0;
   else
      neu = sqrt( x(1) * x(1) + sigma );
      if (x(1) <= 0.0)
         v(1) = x(1) - neu;
      else
         v(1) = -sigma / (x(1) + neu);
      end
      beta = 2.0 * v(1) * v(1) / (sigma + v(1) * v(1));
      v = v / v(1);
   end
```

Una de las aplicaciones más importantes de las reflexiones de Householder es la reducción de matrices densas a formas condensadas. En general, las transformaciones de Householder son empleadas para introducir ceros en las filas y/o columnas de una matriz real densa. Así, si acaso se pretende reducir a cero algunos de los elementos de las columnas de una matriz A, entonces se debe realizar un conjunto de pre-multiplicaciones de la forma HA, donde H representa las matrices de Householder adecuadas. De modo semejante, si acaso se pretende reducir a cero algunos de los elementos de las filas de la matriz A, entonces se debe realizar un conjunto de pos-multiplicaciones de la forma AH, donde H representa también las matrices de Householder adecuadas.

Dada una matriz $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$, se tiene:

- La matriz $H \in \mathbb{R}^{m \times m}$ es tal que

$$HA = \left(I - \frac{2}{v^t v} v v^t \right) A = A - \frac{2}{v^t v} v v^t A = A - v w^t \tag{2.14}$$

con $w = \beta A^t v$ y $\beta = \dfrac{2}{v^t v}$.

- La matriz $H \in \mathbb{R}^{n \times n}$ es tal que

$$AH = A\left(I - \frac{2}{v^t v} v v^t\right) = A - \frac{2}{v^t v} A v v^t = A - w v^t \tag{2.15}$$

con $w = \beta A v$ y $\beta = \dfrac{2}{v^t v}$.

Como se puede observar, la aplicación de la matriz de Householder H a una matriz real A, tanto por la izquierda como por la derecha, jamás necesita del cálculo explícito de la matriz H. Esa aplicación solamente involucra una multiplicación del tipo matriz-vector y un producto vectorial, representado por las relaciones $w = \beta A^t v$ y $v w^t$, $w = \beta A v$ y $w v^t$.

Además, las dos implementaciones pueden aprovechar el hecho de que el vector de Householder v verifica $v(1) = 1$, pues esto va a permitir que las restantes componentes del vector $(v(2:n))$, conocidas como la **parte esencial del vector de Householder**, puedan ser almacenadas en las entradas de la matriz A que van a ser transformadas en cero.

La normalización $v(1) = 1$ puede ser importante en el cálculo computacional del producto HA cuando m es pequeño y en el cálculo computacional del producto AH cuando n es pequeño, ya que permite reducir el número de operaciones de multiplicación a realizar.

2.3. Métodos tradicionales de bidiagonalización

A continuación se describen dos de los métodos tradicionales para la reducción de una matriz real (densa) a la forma bidiagonal (superior). Estos métodos son:

- El método propuesto por Golub y por Kahan [Golub y Kahan, 1965].
- El método propuesto por Lawson y por Hanson [Hanson y Lawson, 1969], [Lawson y Hanson, 1995], y posteriormente revisado por Chan [Chan, 1982a], [Chan, 1982b].

2.3.1. Método de Golub-Kahan

Dada una matriz $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$, el **método de Golub-Kahan** consiste en la construcción de dos secuencias finitas de transformaciones de Householder

$$U_B = U_1 U_2 \dots U_n \in \mathbb{R}^{m\times m} \qquad \text{y} \qquad V_B = V_1 V_2 \dots V_{n-2} \in \mathbb{R}^{n\times n} \tag{2.16}$$

tales que

$$A = U_B \begin{bmatrix} B \\ 0 \end{bmatrix} V_B^t \qquad \text{con} \qquad B = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & 0 & 0 & \alpha_n \end{bmatrix} \tag{2.17}$$

siendo U_B y V_B productos de matrices de Householder.

Por ejemplo, para una matriz de dimensión 5×4, el método es empleado de la siguiente manera

$$\begin{bmatrix} \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \end{bmatrix} \xrightarrow[U_1]{} \begin{bmatrix} \alpha_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 \end{bmatrix} \xrightarrow[V_1]{} \tag{2.18}$$

$$\xrightarrow[V_1]{} \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 \end{bmatrix} \xrightarrow[U_2]{} \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \times_3 & \times_3 \\ 0 & 0 & \times_3 & \times_3 \\ 0 & 0 & \times_3 & \times_3 \\ 0 & 0 & \times_3 & \times_3 \end{bmatrix} \xrightarrow[V_2]{}$$

$$\xrightarrow[V_2]{} \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 \end{bmatrix} \xrightarrow[U_3]{} \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \alpha_3 & \beta_3 \\ 0 & 0 & 0 & \times_5 \\ 0 & 0 & 0 & \times_5 \end{bmatrix} \xrightarrow[U_4]{}$$

$$\xrightarrow[U_4]{} \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \alpha_3 & \beta_3 \\ 0 & 0 & 0 & \alpha_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

con lo cual

$$U_4U_3U_2U_1 \begin{bmatrix} \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 \end{bmatrix} V_1V_2 = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \alpha_3 & \beta_3 \\ 0 & 0 & 0 & \alpha_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \qquad (2.19)$$

La secuencia de matrices de Householder U_j $(j = 1, \ldots, n)$ reducen a cero los últimos $m-j$ elementos de la columna j de la matriz, mientras que la secuencia de matrices de Householder V_k $(k = 1, \ldots, n-2)$ reducen a cero los últimos $n-k-1$ elementos de la fila k de la matriz. Hay que tener en cuenta que las matrices U_j y V_k anteriores son matrices que tienen la forma $\widetilde{H}_p = diag\,(I_r, H_q)$, donde $\widetilde{H}_p \in \mathbb{R}^{p\times p}$ (con $p = r+q$), $I_r \in \mathbb{R}^{r\times r}$ es la matriz identidad y $H_q \in \mathbb{R}^{q\times q}$ es la matriz de Householder definida en (2.9).

Como ya hemos descrito anteriormente, en las entradas de la matriz que van a ser reducidas a cero se pueden almacenar las componentes del vector de Householder, los cuales definen las matrices U y V. Al mismo tiempo, en cada etapa, no hace falta calcular explícitamente la matriz de Householder H para producir las actualizaciones HA o AH.

La rutina descrita seguidamente implementa (en código Matlab), el método de bidiagonalización propuesto por Golub y por Kahan, según el algoritmo 5.4.2 de [Golub y van Loan, 1996, pp. 252].

```
Function A = bidiag_Golub_Kahan( A )
   [m,n] = size( A );
   for j = 1 : n
      [v, beta] = HouseholderVector( A(j:m,j) );
      A(j:m,j:n) = PreHouseholderMult( A(j:m,j:n), v, beta );
      if (j <= n-2)
         [v, beta] = HouseholderVector( A(j,j+1:n)' );
         A(j:m,j+1:n) = PosHouseholderMult( A(j:m,j+1:n), v, beta );
      end
   end
```

2.3.2. Método de Lawson-Hanson-Chan

Cuando la matriz $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ tiene un número de filas mucho mayor que el número de columnas $(m \gg n)$, el método anterior presenta un coste computacional elevado debido al número de operaciones que es necesario realizar para anular los elementos de la matriz que están por debajo de la diagonal principal.

Por este motivo, el método de bidiagonalización de Lawson-Hanson-Chan consiste en reducir primero la matriz a la forma triangular superior, empleando la descomposición QR, y a continuación aplicar el método de Golub-Kahan a esa matriz triangular superior.

El **método de Lawson-Hanson-Chan** es entonces descrito por las siguientes etapas:

1. Aplicación de la descomposición QR a la matriz $A \in \mathbb{R}^{m\times n}$:

   - Cálculo de la matriz ortogonal $Q \in \mathbb{R}^{m\times m}$ y de la matriz triangular superior $R \in \mathbb{R}^{n\times n}$, donde

$$Q^t A = \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{2.20}$$

2. Aplicación del método de Golub-Kahan a la matriz $R \in \mathbb{R}^{n\times n}$:

   - Cálculo de las matrices ortogonales $U_R \in \mathbb{R}^{n\times n}$ y $V_R \in \mathbb{R}^{n\times n}$, tales que

$$U_R^t R V_R = B \qquad \text{con} \qquad B = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & 0 & 0 & \alpha_n \end{bmatrix}. \tag{2.21}$$

Concluidas las dos etapas del método, la matriz bidiagonal B es la matriz definida por

$$\begin{bmatrix} B \\ 0 \end{bmatrix} = U^t A V_R \tag{2.22}$$

con $U = Q \cdot diag\,(U_R, I)$, donde I es la matriz identidad de orden $m - n$.

En la literatura, el método de Lawson-Hanson-Chan es también conocido como el **método de la R-Bidiagonalización**. La rutina siguiente es la implementación (en código Matlab), del método de la R-Bidiagonalización.

```
1 Function [Q,A] = R_Bidiagonalizacion( A )
2    [Q,R] = qr( A, 0 );
3    A = bidiag_Golub_Kahan( R );
```

Dado que el método de Golub-Kahan tiene un coste computacional de $4mn^2 - \frac{4}{3}n^3$ flops en el cálculo de la matriz B (solamente), y en el mismo cálculo, el método de la R-bidiagonalización tiene un coste computacional de $2mn^2 + 2n^3$ flops, se puede concluir que el método de Lawson-Hanson-Chan resulta conveniente cuando $m \geq \frac{5}{3}n$.

Obtenida la matriz bidiagonal superior resultante de la bidiagonalización, la etapa siguiente es calcular su DVS y, con ella, obtener la DVS de la matriz inicial.

Sin embargo, el número de flops de la DVS de una matriz real (densa) depende directamente de las componentes de la descomposición $A = U\Sigma V^t$ que se pretenden calcular: hay aplicaciones donde solamente es necesario calcular los valores singulares (matriz Σ); hay aplicaciones donde se tiene

también que calcular algunos de los respectivos vectores singulares (matrices $\widetilde{U} = \begin{bmatrix} u_1 & \dots & u_n \end{bmatrix}$ o V); y finalmente, hay aplicaciones que necesitan de la descomposición completa. La Tabla 2.1, de [Golub y van Loan, 1996, pp. 254], expone el número de flops necesarios para calcular la DVS en cada una de las seis posibilidades, en función del método de Golub-Reinsch y del método de Lawson-Hanson-Chan.

Calcula	Golub-Kahan DVS	R-DVS
Σ	$4mn^2 - \frac{4}{3}n^3$	$2mn^2 + 2n^3$
Σ y $\widetilde{U}$	$14mn^2 - 2n^3$	$6mn^2 + 11n^3$
Σ y V	$4mn^2 + 8n^3$	$2mn^2 + 11n^3$
Σ y U	$4m^2n - 8mn^2$	$4m^2n + 13n^3$
Σ, $\widetilde{U}$ y V	$14mn^2 + 8n^3$	$6mn^2 + 20n^3$
Σ, U y V	$4m^2n + 8mn^2 + 9n^3$	$4m^2n + 22n^3$

Tabla 2.1: Número de flops de la DVS.

2.4. DVS de la matriz bidiagonal

Para obtener la DVS de la matriz bidiagonal superior $B \in \mathbb{R}^{n\times n}$,

$$B = U_B \Sigma V_B^t \tag{2.23}$$

y, con ella, la DVS de la matriz inicial $A \in \mathbb{R}^{m\times n}$,

$$A = (UU_B)\,\Sigma\,(VV_B)^t \tag{2.24}$$

donde U y V son las matrices ortogonales de la bidiagonalización, existen distintos métodos [Arbenz, 1989], [Gu y Eisenstat, 1995a], [Trefftz *et al.*, 1995],

[Fernando, 1998a], [Fernando, 1998b], [Großer y Lang, 2003], [Willems *et al.*, 2006] y entre ellos se van a describir los siguientes:

- El **método de Golub-Kahan "svd step"** [Golub y Kahan, 1965].
- El **método de Demmel-Kahan** [Demmel y Kahan, 1990].
- El **método de Fernando-Parlett** [Fernando y Parlett, 1994].

2.4.1. Método de Golub-Kahan "svd step"

Como ya fue expuesto, este método está basado en la diagonalización implícita de la matriz tridiagonal

$$T_0 = B_0^t B_0 \tag{2.25}$$

donde $B_0 = B$, empleando el método QR con los desplazamientos de origen de Wilkinson [Wilkinson, 1968],

$$\begin{cases} T_k - \mu_k I = UR & \text{(Descomposición QR)} \\ T_{k+1} = RU + \mu_k I \end{cases} \qquad (k = 0, 1, \ldots) \tag{2.26}$$

sin construir la matriz tridiagonal $T_{k+1} = U^t T_k U = B_{k+1}^t B_{k+1}$ de manera explícita. El desplazamiento de origen (μ_k) es escogido para mejorar la convergencia del método.

El método está compuesto por las siguientes etapas:

1. Calcular el desplazamiento de origen (μ_k), como el valor propio de

$$T_k\,(n-1:n, n-1:n) = \begin{bmatrix} \alpha_{n-1}^2 + \beta_{n-2}^2 & \alpha_{n-1}\beta_{n-1} \\ \alpha_{n-1}\beta_{n-1} & \alpha_n^2 + \beta_{n-1}^2 \end{bmatrix} \tag{2.27}$$

   más cercano a $\alpha_n^2 + \beta_{n-1}^2$.

   Esta manera de calcular μ_k fue descrita en [Wilkinson, 1968], donde

$$\mu_k = \left(\alpha_n^2 + \beta_{n-1}^2\right) + d - sign\,(d)\sqrt{d^2 + \left(\alpha_{n-1}\beta_{n-1}\right)^2} \tag{2.28}$$

   con

$$d = \frac{\left(\alpha_{n-1}^2 + \beta_{n-2}^2\right) - \left(\alpha_n^2 + \beta_{n-1}^2\right)}{2}. \tag{2.29}$$

2. Calcular $c_{\theta_1} = \cos(\theta_1)$ y $s_{\theta_1} = \sin(\theta_1)$, tales que

$$\begin{bmatrix} c_{\theta_1} & s_{\theta_1} \\ -s_{\theta_1} & c_{\theta_1} \end{bmatrix}^t \begin{bmatrix} \alpha_1^2 - \mu_k \\ \alpha_1\beta_1 \end{bmatrix} = \begin{bmatrix} \times \\ 0 \end{bmatrix} \tag{2.30}$$

   y definir la rotación de Givens $G_1 = G(1, 2, \theta_1)$.

3. Calcular las restantes rotaciones de Givens, G_2, ..., G_{n-1}, donde $G_j = G(j, j+1, \theta_j)$ $(j = 2, \ldots, n-1)$, tales que $G^{(k)} = G_1 G_2 \ldots G_{n-1}$ y entonces

$$T_{k+1} = \left(G^{(k)}\right)^t T_k G^{(k)} \tag{2.31}$$

   es tridiagonal y $G^{(k)} e_1 = G_1 e_1 = U e_1$.

   Esta etapa termina con una matriz bidiagonal B_{k+1}, la cual está relacionada con la matriz B_k, por

$$B_{k+1} = \left(U_{n-1}^t \ldots U_2^t U_1^t\right) B_k \left(G_1 G_2 \ldots G_{n-1}\right) = \left(U^{(k)}\right)^t B_k G^{(k)}. \tag{2.32}$$

El método de Golub-Kahan "svd step" converge a la matriz diagonal $\widetilde{\Sigma}$, es decir

$$\lim_{k\to\infty} B_k = \widetilde{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_n \end{bmatrix}. \tag{2.33}$$

2.4.2. Método de Demmel-Kahan

El **método de Demmel-Kahan** es una variación del método de Golub--Kahan "svd step", conocido en la literatura por *implicit zero-shift QR algorithm* y donde los autores han observado que el método anterior puede ser modificado para que, en aritmética de precisión finita, la matriz $\widetilde{B}_{k+1}$ (aproximación de la matriz B_{k+1} en precisión finita) puede ser calculada a partir de $\widetilde{B}_k$ con la máxima precisión numérica permitida [Demmel y Kahan, 1988].

Mientras que el método de Golub-Kahan "svd step" solamente garantiza que los valores singulares son calculados con pequeños errores absolutos

$$|\widetilde{\sigma}_j - \sigma_j| \leq \mathcal{O}(\epsilon) \|B\|_2 \qquad (j = 1, \ldots, n) \tag{2.34}$$

el método de Demmel-Kahan garantiza que los valores singulares son calculados con pequeños errores relativos

$$\frac{|\widetilde{\sigma}_j - \sigma_j|}{\sigma_j} \leq \mathcal{O}(\epsilon) \qquad (j = 1, \ldots, n). \tag{2.35}$$

La excelente precisión es obtenida debido al hecho de que los cálculos en el método son definidos de manera a eliminar las posibles cancelaciones catastróficas.

2.4.3. Método de Fernando-Parlett

En 1994, Fernando y Parlett empleando las **transformaciones de Cholesky**[6], han descrito el **método del cociente de diferencias diferenciales con desplazamientos de origen**, más conocido como el **método dqds** (*differential quotient difference with shifts*), lo cual permite calcular los valores singulares con errores relativos muy pequeños, del mismo modo que el método de Demmel--Kahan, pero siendo este método más rápido [Fernando y Parlett, 1994].

Al mismo tiempo, el método también permite desplazamientos de origen no nulos para incrementar la convergencia.

[6] Sea A una matriz simétrica definida positiva. El método de las transformaciones de Cholesky, empezando por la matriz $A_0 = A$, genera una secuencia de matrices semejantes

$$A_{k+1} = L_k^{-1} A_k L_k = L_k^t L_k \qquad (k = 0, 1, \ldots)$$

donde $L_k L_k^t = A_k$ es la **descomposición de Cholesky** de A_k, con L_k una matriz triangular inferior. El método es algorítmicamente definido por

$A_0 = A$
Para $k = 0, 1, \ldots$
  escoge el desplazamiento de origen μ_k
  $A_k - \mu_k I = L_k L_k^t$ (Cholesky)
  $A_{k+1} = L_k^t L_k + \mu_k I$

2.5. Bidiagonal-DVS *versus* tridiagonal-DVP

Dada la estrecha relación entre valores singulares y valores propios, se puede obtener la DVS de la matriz bidiagonal (superior) a partir de la DVP de una matriz tridiagonal simétrica.

Dada la matriz bidiagonal superior $B \in \mathbb{R}^{n\times n}$,

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & 0 & 0 & \alpha_n \end{bmatrix} \tag{2.36}$$

existen tres maneras de convertir el problema de calcular la DVS de B en calcular la DVP de una matriz tridiagonal simétrica T:

1. Definir la matriz $\widetilde{T} \in \mathbb{R}^{2n\times 2n}$ como

$$\widetilde{T} = \begin{bmatrix} 0 & B^t \\ B & 0 \end{bmatrix} \tag{2.37}$$

y la matriz de permutación $P \in \mathbb{R}^{2n\times 2n}$ por

$$P = \begin{bmatrix} e_1 & e_{n+1} & e_2 & e_{n+2} & \dots & e_n & e_{2n} \end{bmatrix} \tag{2.38}$$

donde e_k $(k = 1, \dots, 2n)$ es la columna k de la matriz identidad de orden $2n$, y seguidamente considerar que

$$T = P^t\widetilde{T}P = \begin{bmatrix} 0 & \alpha_1 & & & & & \\ \alpha_1 & 0 & \beta_1 & & & & \\ & \beta_1 & 0 & \alpha_2 & & & \\ & & \alpha_2 & 0 & \beta_2 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \beta_{n-1} & 0 & \alpha_n \\ & & & & & \alpha_n & 0 \end{bmatrix} \tag{2.39}$$

es la matriz tridiagonal simétrica con la característica de tener ceros en la diagonal principal y los elementos, α_1, β_1, $\ldots$, α_{n-1}, β_{n-1}, α_n, en las dos subdiagonales.

Fácilmente se demuestra que los valores propios positivos de la matriz simétrica $\widetilde{T}$ son los valores singulares de la matriz bidiagonal superior B. Además, como las matrices $\widetilde{T}$ y T son semejantes, los valores singulares de la matriz bidiagonal superior B son los valores propios positivos de la matriz tridiagonal simétrica T [Golub y Kahan, 1965], [Golub y van Loan, 1996].

En realidad, si (λ_k, x_k) es un **par propio** de la matriz T, es decir,

$$Tx_k = \lambda_k x_k \qquad (k = 1, \ldots, 2n) \tag{2.40}$$

donde x_k es un vector unitario, entonces

$$\lambda_k = \pm\sigma_k \tag{2.41}$$

siendo σ_k un valor singular de la matriz B y

$$Px_k = \frac{\sqrt{2}}{2} \begin{bmatrix} v_k \\ \pm u_k \end{bmatrix} \tag{2.42}$$

donde u_k y v_k son, respectivamente, un vector singular izquierdo y derecho de la matriz B.

2. Definir la matriz $T \in \mathbb{R}^{n\times n}$ como $T = BB^t$.

   La matriz T es tridiagonal simétrica y sus elementos de la diagonal principal son

   $$\alpha_1^2 + \beta_1^2,\ \alpha_2^2 + \beta_2^2,\ \ldots\ ,\ \alpha_{n-1}^2 + \beta_{n-1}^2,\ \alpha_n^2 \tag{2.43}$$

   y los elementos de las dos subdiagonales son

   $$\alpha_2\beta_1,\ \alpha_3\beta_2,\ \ldots\ ,\ \alpha_n\beta_{n-1}. \tag{2.44}$$

   Las raíces cuadradas de los valores propios de la matriz tridiagonal simétrica T son los valores singulares de la matriz bidiagonal superior B y los vectores propios de T son los vectores singulares izquierdos de B.

3. Definir la matriz $T \in \mathbb{R}^{n\times n}$ como $T = B^tB$.

   La matriz T es tridiagonal simétrica y sus elementos de la diagonal principal son

   $$\alpha_1^2,\ \alpha_2^2+\beta_1^2,\ \ \ldots\ ,\ \alpha_{n-1}^2+\beta_{n-2}^2,\ \alpha_n^2+\beta_{n-1}^2 \tag{2.45}$$

   y los elementos de las dos subdiagonales son

   $$\alpha_1\beta_1,\ \alpha_2\beta_2,\ \ \ldots\ ,\ \alpha_{n-1}\beta_{n-1}. \tag{2.46}$$

   Las raíces cuadradas de los valores propios de la matriz tridiagonal simétrica T son los valores singulares de la matriz bidiagonal superior B y los vectores propios de T son los vectores singulares derechos de B.

Las maneras 2 y 3 de convertir el problema de calcular la DVS de una matriz bidiagonal superior B en calcular la DVP de una matriz tridiagonal simétrica T, llevan a una pérdida de precisión numérica en el cálculo de los valores singulares más pequeños de B, resultante del cálculo explícito de las matrices BB^t y B^tB, teniendo en cuenta que el número de condición (κ_2) de las matrices BB^t y B^tB es el cuadrado del número de condición de la matriz B.

En [Demmel, 1997], el autor describe el siguiente ejemplo.

Sea $\eta = \frac{\varepsilon}{2}$, donde ε es la precisión de la máquina. En estas condiciones, $1+\eta$ es redondeado a 1. Sea la matriz $B = \begin{bmatrix} 1 & 1 \\ 0 & \sqrt{\eta} \end{bmatrix}$ que tiene como valores singulares valores próximos a $\sqrt{2}$ y $\sqrt{\frac{\eta}{2}}$. Entonces la matriz $B^tB = \begin{bmatrix} 1 & 1 \\ 1 & 1+\eta \end{bmatrix}$ es redondeada a la matriz $T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, la cual es singular. De esta manera, el redondeo de $1+\eta$ en 1, cambia el cálculo del valor singular más pequeño exacto en el valor aproximado de $\sqrt{\frac{\eta}{2}} = \frac{\sqrt{\varepsilon}}{2}$ en 0. Así, si $\varepsilon \approx 10^{-16}$ y $\frac{\sqrt{\varepsilon}}{2} \approx 10^{-8}$, el error introducido en el cálculo explícito de la matriz B^tB es 10^8 veces mayor que el redondeo realizado.

La manera 1 es numéricamente más estable y permite calcular todos los valores singulares de B con elevada precisión relativa [Demmel y Kahan, 1990].

Los métodos que calculan la DVS de una matriz trabajan, bien sobre la matriz bidiagonal superior, bien sobre la matriz tridiagonal simétrica (2.39), y

dada la naturaleza de las matrices, ellas son manejadas computacionalmente empleando solamente los vectores que las definen.

Llegados a este punto, hace falta presentar cómo se calcula la DVP de una matriz tridiagonal simétrica. Lo haremos en el capítulo 4.

2.6. DVS generalizada

La **descomposición en valores singulares generalizada** (DVSG), introducida por van Loan en su tesis doctoral [van Loan, 1976], y descrita igualmente por Paige y por Saunders en [Paige y Saunders, 1981], es también conocida cómo **descomposición en valores singulares cociente** (DVSC), según la propuesta de [de Moor y Golub, 1989].

Dadas las matrices $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{p\times n}$, donde por simplicidad se asume que la matriz $\begin{bmatrix} A^t & B^t \end{bmatrix}$ tiene rango n, existen tres matrices ortogonales, $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{p\times p}$ y $Q \in \mathbb{R}^{n\times n}$, y una matriz triangular superior invertible $R \in \mathbb{R}^{n\times n}$, tales que

$$A = U\Sigma_A RQ^t \qquad \text{y} \qquad B = V\Sigma_B RQ^t \tag{2.47}$$

donde

$$\Sigma_A = diag\,(I_A, D_A, O_A) \qquad \text{y} \qquad \Sigma_B = diag\,(O_B, D_B, I_B) \tag{2.48}$$

con $I_A \in \mathbb{R}^{l\times l}$ y $I_B \in \mathbb{R}^{(n-l-k)\times(n-l-k)}$ matrices identidad, $O_A \in \mathbb{R}^{(m-l-k)\times(n-l-k)}$ y $O_B \in \mathbb{R}^{(p-n+l)\times l}$ matrices nulas, y

$$D_A = \begin{bmatrix} \sigma_{l+1} & 0 & \dots & 0 \\ 0 & \sigma_{l+2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \sigma_{l+k} \end{bmatrix} \quad \text{y} \quad D_B = \begin{bmatrix} \varsigma_{l+1} & 0 & \dots & 0 \\ 0 & \varsigma_{l+2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \varsigma_{l+k} \end{bmatrix} \tag{2.49}$$

con $0 < \sigma_{l+k} \leq \ldots \leq \sigma_{l+1} < 1$, $0 < \varsigma_{l+k} \leq \ldots \leq \varsigma_{l+1} < 1$ y $\sigma_i^2 + \varsigma_i^2 = 1$ $(i = l+1, \ldots, l+k)$.

La DVSG es la generalización de la DVS en el sentido de que si B es la matriz identidad, entonces la DVSG es la DVS de la matriz A. Si B es una

matriz invertible, entonces la DVSG es la DVS de la matriz AB^{-1}. Además, si la matriz $\begin{bmatrix} A^t & B^t \end{bmatrix}^t$ tiene columnas ortonormales, entonces la DVSG es la **descomposición coseno-seno** descrita en [Stewart, 1982].

Los k pares (σ_i, ς_i) $(i = l+1, \ldots, l+k)$, se denominan **pares de valores singulares generalizados** y los cocientes $\frac{\sigma_i}{\varsigma_i}$ se denominan **valores singulares generalizados.**

En LAPACK, la DVSG es obtenida con la rutina `xGGSVD`, la cual implementa el método descrito en [Bai y Demmel, 1993], que está compuesto por dos etapas:

1. La primera etapa, desarrollada por la rutina `xGGSVP`, reduce las matrices A y B a la forma triangular superior

$$U_1^t A Q_1 = \begin{bmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{bmatrix} \quad \text{y} \quad V_1^t B Q_1 = \begin{bmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{bmatrix} \tag{2.50}$$

   donde $A_{12} \in \mathbb{R}^{k\times k}$ y $B_{13} \in \mathbb{R}^{l\times l}$ son matrices triangulares superiores invertibles, $A_{23} \in \mathbb{R}^{l\times l}$ es una matriz triangular superior, $A_{13} \in \mathbb{R}^{k\times l}$ es una matriz trapezoidal superior, U_1, V_1 y Q_1 son matrices ortogonales.

2. La segunda etapa, desarrollada por la rutina `xTGSJA`, calcula la DVSG de los bloques triangulares superiores A_{23} y B_{13}, con

$$A_{23} = U_2 C R Q_2^t \qquad \text{y} \qquad B_{13} = V_2 S R Q_2^t \tag{2.51}$$

   siendo U_2, V_2 y Q_2 matrices ortogonales, C y S matrices diagonales con elementos no negativos y tales que $C^2 + S^2 = I$, donde I es la matriz identidad.

Por ejemplo, en [Brent *et al.*, 1983] y en [Bai, 1994], se describe una implementación paralela de la DVSG.

2.7. DVS producto

El cálculo de la **descomposición en valores singulares producto** (DVSP) de dos matrices reales, fue estudiado por primera vez en 1987 por Fernando y por

Hammarling, como una generalización de la DVS basada en el producto AB^t de las matrices A y B [Fernando y Hammarling, 1987]. En [de Moor, 1989] está detallado un estudio completo de las propiedades de la DVSP. De la misma manera, en [Mollar y Hernández, 1996], en [Mollar y Hernández, 1998] y en [Claver *et al.*, 1999],están relatadas algunas de sus implementaciones paralelas.

Dadas las matrices $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times p}$, existen dos matrices $U \in \mathbb{R}^{m\times n}$ y $V \in \mathbb{R}^{n\times p}$, con las columnas ortogonales, una matriz ortogonal $Q \in \mathbb{R}^{n\times n}$ y una matriz triangular superior invertible $R \in \mathbb{R}^{n\times n}$, tales que

$$A = U\Sigma_A RQ^t \qquad \text{y} \qquad B = QR^{-1}\Sigma_B V^t \tag{2.52}$$

con Σ_A, $\Sigma_B \in \mathbb{R}^{n\times n}$, donde

$$\Sigma_A = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \sigma_n \end{bmatrix} \quad \text{y} \quad \Sigma_B = \begin{bmatrix} \varsigma_1 & 0 & \dots & 0 \\ 0 & \varsigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \varsigma_n \end{bmatrix} \tag{2.53}$$

y además, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ y $\varsigma_1 \geq \varsigma_2 \geq \ldots \geq \varsigma_n \geq 0$.

Los n pares (σ_j, ς_j) $(j = 1, \ldots, n)$, se denominan **valores singulares producto** de (A, B), y los **valores singulares del producto matricial** AB son los valores $\sigma_j\varsigma_j$.

En lo referente al cálculo de la DVSP de dos matrices, Golub, Sølna y van Dooren presentaron un método mucho más amplio, en el cual describe la bidiagonalización implícita del producto/cociente de dos o más matrices, obteniendo una matriz bidiagonal superior que posteriormente se diagonaliza empleando el proceso iterativo escogido [Golub *et al.*, 2000].

2.7.1. Método de Golub-Sølna-van Dooren

El método descrito por Golub, por Sølna y por van Dooren es un método mucho más amplio y permite calcular la DVS del producto y/o cociente de dos o más matrices reales. Sin embargo, dado que nuestro enfoque es el cálculo de la DVSP de dos matrices reales, vamos a exponer el método desde este caso particular.

Dadas las matrices $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times p}$, el **método de Golub-Sølna--van Dooren** calcula la DVS del producto AB sin calcular explícitamente este producto matricial [Golub *et al.*, 2000]. Este método presenta dos etapas distintas:

1. La bidiagonalización implícita del producto matricial AB.

2. El cálculo de la DVS de la matriz bidiagonal.

Dado que lo más novedoso del método está en la etapa 1, a continuación se describe el proceso de la bidiagonalización implícita del producto matricial AB.

Una vez concluida la etapa 1, para calcular la DVS de la matriz bidiagonal, se puede, por ejemplo, aplicar uno de los métodos descritos en la sección 2.4 [Golub y Reinsch, 1970], [Demmel y Kahan, 1990], [Fernando y Parlett, 1994].

Para llevar a cabo la bidiagonalización implícita del producto matricial AB, se realiza una secuencia de actualizaciones de las matrices A y B, mediante la aplicación de un número finito de transformaciones de Householder. Para ilustrar este proceso se describirá su evolución operando sobre un ejemplo con el producto matricial de dos matrices de dimensión 5×5.

El método empieza por aplicar una transformación de Householder H_1 sobre las filas de B y las columnas de A, considerando que $AB = AH_1^t H_1 B$.

La matriz H_1 es elegida para reducir a cero todos los elementos de la primera columna de B, con excepción del primero y de modo que A y B sean actualizadas según

$$A = AH_1^t \qquad \text{y} \qquad B = H_1 B \tag{2.54}$$

para obtener

$$A = \begin{bmatrix} \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \end{bmatrix} \qquad \text{y} \qquad B = \begin{bmatrix} \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \end{bmatrix}. \tag{2.55}$$

A continuación se aplica otra transformación de Householder $H_A^{(1)}$ sobre las filas de A, elegida para reducir a cero todos los elementos de la primera columna

de A, con excepción del primero

$$H_A^{(1)} A = \begin{bmatrix} \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \end{bmatrix}. \tag{2.56}$$

En este punto, la matriz resultante del producto matricial $H_A^{(1)} AB$ sigue teniendo la misma estructura

$$\begin{aligned} H_A^{(1)} AB &= \underbrace{\begin{bmatrix} \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \\ 0 & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} & \times_{1^{(1)}} \end{bmatrix}}_{H_A^{(1)} A} \underbrace{\begin{bmatrix} \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_1 & \times_1 & \times_1 & \times_1 \end{bmatrix}}_{B} \\ &= \begin{bmatrix} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & \times_2 \end{bmatrix}. \end{aligned} \tag{2.57}$$

La fase siguiente es reducir a cero los tres últimos elementos de la primera fila del producto (2.57). Nótese que esta fila se puede obtener a partir del producto de la primera fila de A con la matriz B. Una vez calculado ese producto, se define la transformación de Householder $H_B^{(1)}$ elegida para reducir a cero todos los elementos de la primera fila del producto, con excepción de los dos primeros

$$A\,(1,:)\,BH_B^{(1)} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.58}$$

Cuando la transformación $H_B^{(1)}$ es aplicada a la actual matriz B, la primera fase del proceso de bidiagonalización se completa, pues

$$H_A^{(1)} A B H_B^{(1)} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & 0 \\ 0 & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} \\ 0 & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} \\ 0 & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} \\ 0 & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} & \times_{2^{(1)}} \end{bmatrix}. \tag{2.59}$$

La segunda fase de la bidiagonalización es semejante a la primera. La diferencia es que ahora las transformaciones de Householder son aplicadas sobre las cuatro últimas filas y columnas de A y B. Al inicio de esta etapa, las matrices A y B son

$$A = H_A^{(1)} A \qquad \text{y} \qquad B = B H_B^{(1)}. \tag{2.60}$$

Se empieza aplicando la transformación de Householder H_2 sobre las filas de B y las columnas de A, donde H_2 es elegida para reducir a cero todos los elementos de la segunda columna de B, con excepción de los dos primeros. De esta manera, las matrices A y B se actualizan según

$$A = A H_2^t \qquad \text{y} \qquad B = H_2 B \tag{2.61}$$

y se obtiene

$$A = \begin{bmatrix} \times_{1^{(1)}} & \times_3 & \times_3 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & \times_3 \end{bmatrix} \qquad \text{y} \qquad B = \begin{bmatrix} \times_{1^{(1)}} & \times_3 & \times_3 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & \times_3 \\ 0 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & 0 & \times_3 & \times_3 & \times_3 \end{bmatrix}. \tag{2.62}$$

A continuación se aplica otra transformación de Householder $H_A^{(2)}$ sobre las filas de A, elegida para anular todos los elementos de la segunda columna, excepto los dos primeros

$$H_A^{(2)} A = \begin{bmatrix} \times_{1^{(1)}} & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} \\ 0 & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} \\ 0 & 0 & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} \\ 0 & 0 & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} \\ 0 & 0 & \times_{3^{(2)}} & \times_{3^{(2)}} & \times_{3^{(2)}} \end{bmatrix}. \tag{2.63}$$

De nuevo, la matriz resultante del producto $H_A^{(2)}AB$ sigue teniendo la misma estructura

$$H_A^{(2)}AB = \begin{bmatrix} \times_4 & \times_4 & \times_4 & \times_4 & \times_4 \\ 0 & \times_4 & \times_4 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 & \times_4 \end{bmatrix}. \tag{2.64}$$

La fase siguiente es reducir a cero los dos últimos elementos de la segunda fila del producto (2.64). Nótese que esta fila se puede obtener a partir del producto de la segunda fila de A con la matriz B. Una vez calculado ese producto, se define la transformación de Householder $H_B^{(2)}$ tal que

$$A\,(2,:)\,BH_B^{(2)} = \begin{bmatrix} 0 & \alpha_2 & \beta_2 & 0 & 0 \end{bmatrix}. \tag{2.65}$$

Cuando la transformación $H_B^{(2)}$ es aplicada a la actual matriz B, la segunda fase de la bidiagonalización se completa y se obtiene

$$H_A^{(2)}H_A^{(1)}ABH_B^{(1)}H_B^{(2)} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 & 0 \\ 0 & 0 & \times_{4^{(2)}} & \times_{4^{(2)}} & \times_{4^{(2)}} \\ 0 & 0 & \times_{4^{(2)}} & \times_{4^{(2)}} & \times_{4^{(2)}} \\ 0 & 0 & \times_{4^{(2)}} & \times_{4^{(2)}} & \times_{4^{(2)}} \end{bmatrix}. \tag{2.66}$$

Se puede proceder de esta manera sucesivamente, hasta que el producto esté completamente bidiagonalizado.

El algoritmo siguiente expone el método de Golub-Sølna-van Dooren, donde A y B son las matrices de entrada, D y E son los vectores que almacenan los elementos de la matriz bidiagonal.

Algoritmo 1 *Golub_Sølna_vanDooren* (A, B, D, E)

Para $i = 1, 2, \ldots, n-1$ **hacer:**
Calcular H_i que anule los elementos $B\,(i+1:n, i)$
Aplicar H_i a B por la izquierda $(B = H_i B)$
Aplicar H_i a A por la derecha $(A = AH_i^t)$
Calcular $H_A^{(i)}$ que anule los elementos $A\,(i+1:n, i)$
Aplicar $H_A^{(i)}$ a A por la izquierda $(A = H_A^{(i)} A)$
Calcular el producto implícito AB haciendo $A\,(i, i:n)\,B\,(i:n, i:n)$
Calcular $H_B^{(i)}$ que anule los elementos $AB\,(i, \text{mín}\,(i+2, n):n)$
Almacenar el elemento de la diagonal $(D\,(i) = AB\,(i))$
Almacenar el elemento de la super-diagonal $(E\,(i) = AB\,(i+1))$
Aplicar $H_B^{(i)}$ a B por la derecha $(B = BH_B^{(i)})$
Fin
Almacenar el elemento de la diagonal $(D\,(n) = A\,(n, n)\,B\,(n, n))$

Las implementaciones secuenciales y paralelas del método propuesto por Golub, por Sølna y por van Dooren, desarrolladas con las rutinas de LAPACK y de SCALAPACK, van a ser descritas en el capítulo 7. Al mismo tiempo, son relatados los resultados obtenidos por esas implementaciones, así como los resultados obtenidos por las nuevas implementaciones del método extensible al producto de cualquier número de matrices reales cuadradas.

Esta hoja ha sido dejada en blanco de forma intencionada.

Capítulo **3**

Métodos Alternativos de Bidiagonalización

EN este capítulo están descritos los **métodos alternativos de bidiagonalización** propuestos por **Ralha** [Ralha, 2003] y por **Barlow** [Barlow *et al.*, 2005], y algunas de las aportaciones de la tesis.

El capítulo empieza por una exposición del método propuesto por Ralha para la reducción de matrices reales densas a la forma bidiagonal superior, donde se aplican transformaciones de Householder unilaterales y a continuación se describen las modificaciones propuestas por Barlow para solucionar el problema de la precisión numérica del método original.

Seguidamente se presentan las **modificaciones** que se han propuesto al **método de Barlow**, en el marco de la tesis, con el objetivo de reducir el número de comunicaciones necesarias en la implementación paralela destinada a sistemas de memoria distribuida.

A continuación se exponen las implementaciones secuenciales y paralelas desarrolladas en el marco de tesis, para estos métodos alternativos de bidiagonalización.

Posteriormente se presenta el estudio comparativo de los resultados obtenidos por las implementaciones desarrolladas en el entorno de las librerías numéricas LAPACK y SCALAPACK, y las correspondientes rutinas de estas librerías con la misma funcionalidad.

La descripción de las conclusiones más importantes del trabajo desarrollado cierran el capítulo.

3.1. Método de Ralha

El método más eficiente para calcular la DVS de una matriz real densa es empezar por reducir la matriz a la forma bidiagonal (superior), y posteriormente aplicar un proceso iterativo para calcular la DVS de la matriz bidiagonal (superior).

Habitualmente este proceso de bidiagonalización consiste en la aplicación de sucesivas transformaciones de Householder, por la izquierda y por la derecha de la matriz. Además, el proceso de bidiagonalización tiene un coste computacional más elevado que el proceso iterativo empleado para calcular la DVS de la matriz bidiagonal (superior). Como fue descrito en el capítulo anterior, el método de bidiagonalización tiene un coste computacional que es proporcional a mn^2 flops, lo que significa tiempos de computación elevados, para matrices con dimensiones m y n grandes.

Al mismo tiempo, el hecho de que las transformaciones de Householder sean aplicadas por los dos lados de la matriz, repercute negativamente en los costes de comunicaciones de una implementación paralela destinada a sistemas de memoria distribuida.

Con el objetivo de presentar un método más sencillo de paralelizar que los métodos tradicionales, Ralha [Ralha, 1994], [Ralha, 2000], [Ralha, 2003], Ralha y Mackiewicz [Ralha y Mackiewicz, 1996], propusieron un nuevo método para la parte de la bidiagonalización en el que las transformaciones de Householder son aplicadas solamente por el lado derecho de la matriz. Esto permite presentar todos los cálculos computacionales en términos de las columnas de la matriz sobre la que se trabaja, permitiendo que el método sea más sencillo de paralelizar, comparado con los métodos tradicionales, y además, permite reducir significativamente el número de comunicaciones necesarias.

3.1.1. Descripción del método

Para una matriz densa $A \in \mathbb{R}^{m\times n}$ (con $m \geq n$), el **método de Ralha** está compuesto por dos etapas:

- La primera etapa consiste en la aplicación de una secuencia de $n-2$ transformaciones de Householder

$$A_r = A_{r-1}\widetilde{H}_r = A_{r-1} \cdot diag\,(I_r, H_r) \qquad (r = 1, \ldots, n-2) \tag{3.1}$$

  donde $\widetilde{H}_r \in \mathbb{R}^{n\times n}$, $I_r \in \mathbb{R}^{r\times r}$ es la matriz identidad, $H_r \in \mathbb{R}^{(n-r)\times(n-r)}$ es la matriz de Householder y A_0 es la matriz inicial A.

  Las matrices de Householder H_r en (3.1) son seleccionadas de manera que las columnas a_i y a_j de la matriz A_{n-2} verifiquen

$$a_i^t a_j = 0 \qquad \text{para} \qquad |i-j| > 1. \tag{3.2}$$

- La segunda etapa consiste en la aplicación de una variante del **método de ortogonalización de Gram-Schmidt**, con el objetivo de calcular la descomposición

$$A_{n-2} = QB \tag{3.3}$$

  donde $Q \in \mathbb{R}^{m\times n}$ es una matriz con las columnas ortogonales y $B \in \mathbb{R}^{n\times n}$ es la matriz bidiagonal superior buscada.

3.1.2. Tri-ortogonalización de Householder

A la primera etapa del método propuesto por Ralha, su autor la llamó **tri-ortogonalización de Householder**, pues afirmar que el hecho de que las columnas a_i y a_j de la matriz A_{n-2} verifiquen la condición expresada por (3.2) es equivalente a afirmar que la matriz simétrica $A_{n-2}^t A_{n-2}$ es tridiagonal y, por tanto, para la construcción de la matriz A_{n-2} en (3.1), las matrices de Householder H_r son las matrices necesarias para la reducción de la matriz simétrica A^tA a la forma tridiagonal T,

$$\begin{aligned} T &= \widetilde{H}_{n-2}^t \ldots \widetilde{H}_2^t \widetilde{H}_1^t \left(A^tA\right) \widetilde{H}_1 \widetilde{H}_2 \ldots \widetilde{H}_{n-2} = \\ &= \left(A\widetilde{H}_1 \widetilde{H}_2 \ldots \widetilde{H}_{n-2}\right)^t \left(A\widetilde{H}_1 \widetilde{H}_2 \ldots \widetilde{H}_{n-2}\right) = \\ &= A_{n-2}^t A_{n-2}. \end{aligned} \tag{3.4}$$

Suponiendo que se ha calculado explícitamente la matriz simétrica A^tA y que se ha empezado el proceso anterior, al final del primer paso se tendría

$$\widetilde{H}_1^t \left(A^tA\right) \widetilde{H}_1 = \begin{bmatrix} \times & \times & 0 & 0 & \dots & 0 \\ \times & \times & \times & \times & \dots & \times \\ 0 & \times & \times & \times & \dots & \times \\ 0 & \times & \times & \times & \dots & \times \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \times & \times & \times & \dots & \times \end{bmatrix} \tag{3.5}$$

con

$$\widetilde{H}_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} \tag{3.6}$$

donde H_1 es la matriz de Householder construida a partir de los $n-1$ últimos elementos de la primera fila (o columna) de la matriz simétrica A^tA.

Sin embargo, en el método de Ralha, la matriz simétrica A^tA no se calcula explícitamente y tampoco se aplica la pre-multiplicación de Householder. En el paso r del método, la construcción de la matriz de Householder

$$H_r = I_{n-r} - \frac{2}{v_r^t v_r} v_r v_r^t \tag{3.7}$$

necesita del vector v_r, el cual es calculado a partir de los $n-r$ productos internos involucrando las convenientes columnas de la matriz A_{r-1}. Además, una vez que las columnas a_i y a_j de la matriz A_{r-1} verifican

$$a_i^t a_j = 0 \qquad \text{para} \qquad i = 1, \dots, r-1; j = i+2, \dots, n \tag{3.8}$$

en cada paso r del método, la transformación (3.1) cambia solamente las $n-r$ últimas columnas de la matriz A_{r-1}.

A continuación se expone el algoritmo que implementa la tri-ortogonalización de Householder, donde A es la matriz de entrada/salida. En el algoritmo, las etapas 4 y 6 son responsables por la mayoría del esfuerzo computacional, sin embargo, en una implementación orientada por columnas, esas dos etapas pueden ser realizadas en términos de operaciones del tipo *axpy*[1].

[1] En BLAS, dados x, $y \in \mathbb{R}^n$ y $a \in \mathbb{R}$, se llama *axpy* al conjunto de operaciones que actualizan y mediante la relación $y = ax + y$.

Algoritmo 2 *RalhaTriortogonalizacion* (A)

Para $r = 1, 2, \ldots, n-2$ **calcular:**

1. el vector x_r con los $n - r$ productos internos

2. el vector de Householder v_r

3. el parámetro β_r

4. el producto matriz-vector $A_{r-1}v_r$

5. el vector $w_r = \beta_r \left(A_{r-1}v_r\right)$

6. la matriz $A_r = A_{r-1} - w_r v_r^t$

Fin

En [Ralha y Mackiewicz, 1996], los autores demostraron también las dos siguientes propiedades de la tri-ortogonalización de Householder:

1. Todas las columnas de la matriz A_{n-2} son *esencialmente independientes* (solamente cambian los signos de las columnas), respecto al orden de las columnas a_2, a_3, ... , a_n, de la matriz inicial A_0.

2. Si la matriz inicial A_0 tiene rango k (con $k \ll n$), entonces las columnas a_{k+2}, a_{k+3}, ... , a_n, de la matriz A_{n-2} son nulas (en precisión exacta).

3.1.3. Bidiagonalización de la matriz A_{n-2}

Una vez calculada la matriz A_{n-2}, la segunda etapa del método de Ralha consiste en la aplicación de una variante del **método de ortogonalización de Gram-Schmidt** con el objetivo de calcular la descomposición

$$A_{n-2} = QB \tag{3.9}$$

donde $Q \in \mathbb{R}^{m\times n}$ es una matriz con las columnas ortogonales y $B \in \mathbb{R}^{n\times n}$ es la matriz bidiagonal superior buscada.

Representando por a_j las columnas de la matriz A_{n-2} y por q_j las columnas de la matriz Q, tenemos

$$\begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} = \begin{bmatrix} q_1 & q_2 & \dots & q_{n-1} & q_n \end{bmatrix} \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \dots & 0 & 0 & \alpha_n \end{bmatrix} \tag{3.10}$$

con lo cual,

$$\begin{aligned} a_1 &= \alpha_1 q_1; \\ a_2 &= \beta_1 q_1 + \alpha_2 q_2; \\ &\vdots \\ a_n &= \beta_{n-1} q_{n-1} + \alpha_n q_n; \end{aligned} \tag{3.11}$$

y consecuentemente,

$$\begin{aligned} q_1 &= \frac{a_1}{\alpha_1}; \\ q_j &= \frac{a_j - \beta_{j-1} q_{j-1}}{\alpha_j} \qquad (j = 2, \dots, n)\,. \end{aligned} \tag{3.12}$$

Una vez que cada β_{j-1} $(j = 2, \dots, n)$ es escogido de manera que las columnas q_j y q_{j-1} sean ortogonales, y que cada α_j $(j = 1, \dots, n)$ es tal que $\|q_j\|_2 = 1$, a partir de las condiciones anteriores, tenemos:

- Para $j = 1$:

$$\|q_1\|_2 = 1 \Leftrightarrow \frac{\|a_1\|_2}{\alpha_1} = 1 \Leftrightarrow \alpha_1 = \|a_1\|_2\,. \tag{3.13}$$

- Para $j = 2, \dots, n$:

$$q_j^t q_{j-1} = 0 \Leftrightarrow \left(a_j - \beta_{j-1} q_{j-1}\right)^t q_{j-1} = 0 \Leftrightarrow \beta_{j-1} = a_j^t q_{j-1}; \tag{3.14}$$

$$\|q_j\|_2 = 1 \Leftrightarrow \frac{\left\|a_j - \beta_{j-1} q_{j-1}\right\|_2}{\alpha_j} = 1 \Leftrightarrow \alpha_j = \left\|a_j - \beta_{j-1} q_{j-1}\right\|_2\,. \tag{3.15}$$

3.1.4. Algoritmo del método de Ralha

El método de Ralha se puede así expresar, en forma de algoritmo, de la siguiente manera.

Algoritmo 3 *RalhaBidiagonalizacion* (A, α, β, Q)

Para $r = 1, 2, \ldots, n-2$ **calcular:**

$x_r = A(:, r+1:n)^t A(:, r)$

H_r tal que $H_r^t x_r = \phi_r e_1$

$A(:, r+1:n) = A(:, r+1:n) H_r$

Fin

Calcular $\alpha_1 = \|A(:,1)\|_2$

Calcular $q_1 = \dfrac{A(:,1)}{\alpha_1}$

Para $r = 2, 3, \ldots, n$ **calcular:**

$\beta_{r-1} = q_{r-1}^t A(:, r)$

$A(:, r) = A(:, r) - \beta_{r-1} q_{r-1}$

$\alpha_r = \|A(:, r)\|_2$

$q_r = \dfrac{A(:, r)}{\alpha_r}$

Fin

Hay que señalar que en el algoritmo anterior y en los que van a ser expuestos a continuación, los valores reales α_j $(j = 1, \ldots, n)$ y β_{j-1} $(j = 2, \ldots, n)$, representan de manera compacta la matriz bidiagonal superior B: el vector α es la diagonal principal y el vector β es la primera superdiagonal.

En el método de Ralha, la tri-ortogonalización de Householder tiene un coste computacional de $3mn^2 - 3mn - 6m$ flops, mientras que la bidiagonalización de la matriz A_{n-2} tiene un coste computacional de $4mn - 4m$ flops. La variante del método de ortogonalización de Gram-Schmidt aplicada a la matriz A_{n-2} necesita $\mathcal{O}(mn)$ flops, respecto a los $\mathcal{O}(mn^2)$ flops habituales. Además, en los problemas en los cuales solo se necesita calcular los valores singulares de la matriz A, la matriz Q no necesita ser calculada y, por tanto, en el algoritmo anterior, un solo vector q es suficiente pues todos los vectores q_j $(j = 1, \ldots, n)$

son almacenados en q. Sin embargo, esto no significa que haya un ahorro de memoria, pues cuando se tenga que calcular la matriz Q, como fue descrito en el capítulo anterior, cada vector q_j $(j = 1, \ldots, n)$ puede ser almacenado en las entradas de las correspondientes columnas de la matriz A, teniendo en cuenta la parte esencial del vector de Householder.

3.2. Método de Barlow

El método de Ralha, sin embargo, presenta problemas de precisión numérica para algunos casos de matrices, o sea, para los casos descritos en [Ralha, 2003] no es posible garantizar que los valores singulares de la matriz bidiagonal obtenida correspondan a los valores singulares de la matriz perturbada $A+E$, donde $\|E\|$ se limita como en los métodos tradicionales (*backward stability*) [Parlett, 1998, pp. 94–96].

Basados en la investigación de Ralha, Barlow [Barlow, 2003], y más tarde en colaboración con Bosner y con Drmač [Barlow *et al.*, 2005], han propuesto dos modificaciones al método de Ralha con el objetivo de solucionar el problema de la precisión numérica descrito en [Ralha, 2003]. En el **método de Barlow**, las dos modificaciones propuestas y presentadas en [Barlow *et al.*, 2005], para cada paso r $(r = 1, \ldots, n-2)$ del método de Ralha consisten en:

1. El vector q_r es calculado inmediatamente después de aplicar la transformación de Householder H_r.

2. La transformación de Householder H_{r+1} es calculada a partir de

$$x_r = A\left(:, r+1:n\right)^t q_r. \tag{3.16}$$

Estos autores también han señalado que el cálculo del producto interno $\beta_{r-1} = q_{r-1}^t A\left(:, r\right)$ no es necesario (tal como acontece en el método de Ralha), pues

$$\beta_{r-1} = \frac{\phi_r}{\alpha_{r-1}} \tag{3.17}$$

donde ϕ_r es tal que $H_r^t x_r = \phi_r e_1$, con e_1 la primera columna de la matriz identidad.

En [Barlow *et al.*, 2005] está expuesto un extenso estudio respecto al análisis de los errores y está demostrado que las dos modificaciones introducidas en el método propuesto por Ralha tienen las siguientes consecuencias:

- El método es capaz de calcular una matriz bidiagonal B de una manera estable y con un pequeño *backward error*, o sea, para cada $j = 1, \ldots, n$,

$$|\sigma_j(B) - \sigma_j(A)| \leq f(m,n)\varepsilon_M \|A\|_2 + \mathcal{O}\left(\varepsilon_M^2\right) \tag{3.18}$$

  donde $f(m,n)$ es una función de crecimiento moderado y ε_M es el *epsilon* de la máquina.

- La ortogonalidad de las columnas de la matriz U es semejante a la de la matriz Q en la descomposición QR del método de Gram-Schmidt modificado [Björck, 1967], [Björck y Paige, 1992], [Björck, 1994], o sea, la matriz U puede estar lejos de ser una matriz ortogonal [Barlow *et al.*, 2005, Ejemplo 3.1], pero los principales vectores singulares izquierdos de la matriz A pueden ser recalculados con elevada ortogonalidad [Barlow *et al.*, 2005, Corolario 3.20].

Las dos modificaciones propuestas al método de Ralha representan una composición de las dos etapas de ese método. Como fue descrito anteriormente, en el método de Ralha existen dos etapas, las cuales son realizadas de manera independiente y una después de la otra. En este método, esas dos etapas son realizadas en el mismo paso. Así, el Algoritmo 4 expresa el método propuesto por Barlow, por Bosner y por Drmač.

3.3. Método de Barlow modificado

Al desarrollar las implementaciones paralelas del método de Barlow se ha observado que en cada paso r $(r = 1, \ldots, n-2)$, el método necesita de una operación de comunicación involucrando a todos los procesadores empleados para calcular α_r. Seguidamente se aplica una normalización para calcular el vector q_r, y posteriormente se necesita de una segunda operación de comunicación involucrando de nuevo a todos los procesadores empleados para calcular los $n - r$ productos internos

$$x_r = A\left(:, r+1 : n\right)^t q_r. \tag{3.19}$$

Algoritmo 4 $BarlowBidiagonalizacion\,(A, \alpha, \beta, Q)$

Para $r = 1, 2, \ldots, n-2$ **calcular:**

$\alpha_r = \|A\,(:, r)\|_2$

$q_r = \dfrac{A\,(:, r)}{\alpha_r}$

$x_r = A\,(:, r+1:n)^t\, q_r$

H_r tal que $H_r^t x_r = \beta_r e_1$

$A\,(:, r+1:n) = A\,(:, r+1:n)\, H_r$

$A\,(:, r+1) = A\,(:, r+1) - \beta_r q_r$

Fin

Calcular $\alpha_{n-1} = \|A\,(:, n-1)\|_2$

Calcular $q_{n-1} = \dfrac{A\,(:, n-1)}{\alpha_{n-1}}$

Calcular $\beta_{n-1} = q_{n-1}^t A\,(:, n)$

Calcular $A\,(:, n) = A\,(:, n) - \beta_{n-1} q_{n-1}$

Calcular $\alpha_n = \|A\,(:, n)\|_2$

Calcular $q_n = \dfrac{A\,(:, n)}{\alpha_n}$

Estas dos operaciones de comunicación pueden ser agrupadas si hacemos la normalización del vector q_r más tarde. Estas ideas son aportaciones originales en el marco de la tesis y fueron descritas por primera vez en [Campos *et al.*, 2007], dando origen a un nuevo método alternativo de bidiagonalización, el cual se nombró **método de Barlow modificado**.

El método de Barlow modificado consiste en:

1. Sustituir el cálculo del vector x_r

$$x_r = A\,(:, r+1:n)^t\, q_r \tag{3.20}$$

por

$$x_r = A\,(:, r:n)^t\, A\,(:, r) \tag{3.21}$$

y, por tanto, hacer el cálculo de $n-r+1$ productos internos en cada paso r del método.

2. Sustituir el cálculo del valor α_r

$$\alpha_r = \|A(:,r)\|_2 \tag{3.22}$$

por

$$\alpha_r = \sqrt{x_r(1)} \tag{3.23}$$

donde $x_r(1)$ es la primera componente del vector x_r.

Se puede observar que el segmento del vector x_r definido por $x_r(2:n-r+1)$ es distinto del vector x_r calculado en el método de Barlow, en un factor de α_r. En este nuevo método no hay necesidad de hacer esta actualización pues la transformación de Householder resultante es invariante respecto a ese escalado. Sin embargo, hay que tener en cuenta que en el cálculo de los valores β_r, la relación

$$\beta_r = \frac{\|x_r(2:n-r+1)\|_2}{\alpha_r} \tag{3.24}$$

se tiene que preservar en cada paso r del método.

Las dos modificaciones propuestas al método de Barlow permiten aportar las siguientes observaciones:

- En el cálculo del vector x_r el vector q_r es sustituido por la columna r de la matriz A

$$q_r = A(:,r) \tag{3.25}$$

y, por lo tanto, hay una optimización del espacio de memoria empleado por el nuevo método.

- El nuevo vector x_r tiene una componente más, la cual almacena el producto interno $A(:,r)^t A(:,r)$

$$x_r(1) = A(:,r)^t A(:,r). \tag{3.26}$$

Así, hay una optimización del coste de comunicaciones a la hora de calcular los valores de α_r ya que cada procesador puede calcular localmente α_r según (3.23).

- La normalización del vector q_r puede ser efectuada localmente, pues cada procesador actualiza su segmento local de q_r con

$$A(:,r) = \frac{A(:,r)}{\alpha_r}. \tag{3.27}$$

Además, el cálculo de los valores α_{n-1} y β_{n-1} puede también ser agrupado para reducir el número de comunicaciones, pues se puede emplear una sola operación de comunicación en el cálculo de

$$x_{n-1} = A(:,n-1:n)^t A(:,n-1) \tag{3.28}$$

en cada procesador, y a continuación cada procesador calcula localmente α_{n-1} y β_{n-1} según

$$\alpha_{n-1} = \sqrt{x_{n-1}(1)} \qquad \text{y} \qquad \beta_{n-1} = \frac{x_{n-1}(2)}{\alpha_{n-1}}. \tag{3.29}$$

El Algoritmo 5 expresa el método de Barlow modificado.

Cabe destacar que las modificaciones introducidas al método de Barlow no cambian su complejidad computacional y tampoco sus propiedades de precisión numérica, pues en el método de Barlow modificado se pueden aplicar los mismos argumentos del análisis de errores descritos en [Barlow *et al.*, 2005] para el método de Barlow.

En cada paso r ($r = 1, \ldots, n-2$) de este nuevo método se calcula una matriz A_r la cual es el producto exacto

$$A_r = (A_{r-1} + E_r) \cdot diag(I_r, H_r) \tag{3.30}$$

donde H_r es la transformación de Householder definida exactamente por el vector x_r de los productos internos y E_r es una matriz de perturbación con

$$\|E_r\|_2 \leq \mathcal{O}(\varepsilon_M) \|A_{r-1}\|_2. \tag{3.31}$$

La matriz E_r agrupa los errores producidos por la aproximación $\widehat{v}_r$ en el cálculo del vector exacto de Householder v_r y los errores producidos en la aplicación de la transformación de Householder $A_{r-1} \cdot diag\left(I_r, \widehat{H}_r\right)$.

Algoritmo 5 $Barlow Modificado\,(A, \alpha, \beta)$

Para $r = 1, 2, \ldots, n-2$ **calcular:**

$x_r = A\,(:, r:n)^t\, A\,(:, r)$

$\alpha_r = \sqrt{x_r\,(1)}$

$A\,(:, r) = \dfrac{A\,(:, r)}{\alpha_r}$

H_r tal que $H_r^t x_r\,(2 : n-r+1) = \phi_r e_1$

$A\,(:, r+1:n) = A\,(:, r+1:n)\, H_r$

$\beta_r = \dfrac{\phi_r}{\alpha_r}$

$A\,(:, r+1) = A\,(:, r+1) - \beta_r A\,(:, r)$

Fin

Calcular $x_{n-1} = A\,(:, n-1:n)^t\, A\,(:, n-1)$

Calcular $\alpha_{n-1} = \sqrt{x_{n-1}\,(1)}$

Calcular $\beta_{n-1} = \dfrac{x_{n-1}\,(2)}{\alpha_{n-1}}$

Calcular $A\,(:, n-1) = \dfrac{A\,(:, n-1)}{\alpha_{n-1}}$

Calcular $A\,(:, n) = A\,(:, n) - \beta_{n-1} A\,(:, n-1)$

Calcular $\alpha_n = \|A\,(:, n)\|_2$

Calcular $A\,(:, n) = \dfrac{A\,(:, n)}{\alpha_n}$

En el método de Barlow modificado se calcula una aproximación $\widetilde{v}_r$ con pequeñas diferencias respecto al vector $\widehat{v}_r$ (para un análisis más detallado, consultar [Wilkinson, 1965, pp. 152–157]), pero aún así, la transformación de Householder producida $\widetilde{H}_r$ está muy cerca de la transformación de Householder exacta pues

$$\left\| \widetilde{H}_r - H_r \right\|_2 = \mathcal{O}\,(\varepsilon_M)\,. \tag{3.32}$$

En este método alternativo para la bidiagonalización de matrices reales densas, lo único que se ha hecho es reducir el número de comunicaciones necesarias para una implementación paralela destinada a sistemas de memoria distribuida.

3.4. Comparativa de costes computacionales

Los dos métodos alternativos de bidiagonalización tienen el mismo orden de complejidad, pues el número de operaciones aritméticas es similar. Además, dependiendo del tipo de DVS deseada, el número de operaciones aritméticas es distinto, tal como ocurre en los métodos tradicionales de bidiagonalización de matrices reales densas.

Como fue descrito en el capítulo anterior, cuando la dimensión de la matriz real densa es tal que $m \geq \frac{5}{3}n$, el método tradicional de bidiagonalización empleado e implementado en la librería numérica LAPACK es el método de la R-bidiagonalización, en el cual se empieza la reducción de la matriz inicial a la forma bidiagonal superior aplicando una descomposición QR.

La Tabla 3.1 expone el número de flops del método de la R-bidiagonalización, respecto a los métodos alternativos de bidiagonalización, en el cálculo de la matriz bidiagonal superior, para los casos más frecuentes del cálculo de la DVS: calcular solamente los valores singulares; calcular los valores singulares y los vectores singulares por la derecha; y calcular la DVS completa.

Calcula	R-bidiagonalización	Ralha & Barlow
Σ	$2mn^2 + 2n^3$	$3mn^2$
Σ y V_B	$2mn^2 + \frac{10}{3}n^3$	$3mn^2 + \frac{4}{3}n^3$
Σ, U_B y V_B	$4mn^2 + \frac{8}{3}n^3$	$3mn^2 + \frac{4}{3}n^3$

Tabla 3.1: Número de flops de la bidiagonalización con descomposición QR.

Comparando el número de flops, se puede observar la clara ventaja de los métodos alternativos de bidiagonalización cuando son aplicados en el cálculo de la DVS completa de una matriz real densa.

Al mismo tiempo, se puede observar que cuando $m \in \left[\frac{5}{3}n, 2n\right]$, los métodos alternativos de bidiagonalización tienen un coste computacional menor que el método de la R-bidiagonalización, en los otros dos casos expuestos en la Tabla 3.1.

Hay que señalar que cuando se necesita calcular vectores singulares, los métodos alternativos de bidiagonalización tienen el mismo orden de complejidad, independientemente del tipo de DVS deseada, pues en estos métodos, la matriz U_B siempre se calcula una vez que se hace la acumulación de las transformaciones de Householder.

Sin embargo, cuando se hace un análisis entre en número de flops del método de Golub-Kahan y los métodos alternativos de bidiagonalización (Tabla 3.2), se puede concluir que los métodos de Ralha y de Barlow tienen un coste computacional menor cuando $m \geq \frac{4}{3}n$.

Calcula	Golub-Kahan	Ralha & Barlow
Σ	$4mn^2 - \frac{4}{3}n^3$	$3mn^2$
Σ y V_B	$4mn^2$	$3mn^2 + \frac{4}{3}n^3$
Σ, U_B y V_B	$4m^2n + 4mn^2 - \frac{4}{3}n^3$	$3mn^2 + \frac{4}{3}n^3$

Tabla 3.2: Número de flops de la bidiagonalización sin descomposición QR.

La Tabla 3.2 tiene especial importancia pues en la librería SCALAPACK el método de bidiagonalización implementado es el método de Golub-Kahan y, por tanto, en esta librería, cuando la dimensión de la matriz es tal que $m \geq \frac{5}{3}n$ no se aplica la descomposición QR inicial como en LAPACK. Muchas veces en la computación de altas prestaciones, la implementación paralela no es la versión exactamente paralela del mejor algoritmo secuencial. En [Choi *et al.*, 1996a] se describe la implementación paralela de la descomposición QR puesta a disposición por SCALAPACK.

La Tabla 3.3 particulariza la Tabla 3.2 para el caso de las matrices reales cuadradas. Se observa que el número de flops de los métodos alternativos de bidiagonalización está muy cerca del número de flops del método de Golub--Kahan, teniendo clara ventaja en el caso de la DVS completa.

Aunque se haya observado que hay un espectro pequeño de casos donde los métodos alternativos de bidiagonalización tienen un menor número de flops, respecto al método de la R-bidiagonalización, hay otro aspecto que conviene

Calcula	Golub-Kahan	Ralha & Barlow
Σ	$\frac{8}{3}n^3$	$3n^3$
Σ y V_B	$4n^3$	$\frac{13}{3}n^3$
Σ, U_B y V_B	$\frac{20}{3}n^3$	$\frac{13}{3}n^3$

Tabla 3.3: Número de flops de la bidiagonalización en matrices cuadradas.

señalar y que es la aplicación unilateral de las transformaciones de Householder permitiendo definir todas las operaciones en términos de las columnas de la matriz con la cual se trabaja. Esto tiene particular importancia en las implementaciones codificadas en lenguajes en las cuales se almacenan las matrices por columnas, como es en particular el Fortran empleado en el marco de la tesis.

La aplicación de las transformaciones de semejanza unilaterales es una característica explotada por el método de Jacobi, expuesto por primera vez en 1848 [Jacobi, 1848]. Además, el método de Jacobi permite calcular los valores singulares de una matriz real con una precisión numérica mucho más elevada que con cualquier algoritmo numérico que empiece por reducir la matriz inicial a la forma bidiagonal superior [Demmel y Veselić, 1992]. El mayor problema del método de Jacobi es su lentitud, pues sólo suele ser un método competitivo cuando permite calcular los valores singulares, más o menos, en tres pasos, cosa que raramente ocurre. Sin embargo, hay estudios que tienen por objetivo mejorar las prestaciones del método de Jacobi [Drmač, 1999], [Drmač y Veselić, 2007a], [Drmač y Veselić, 2007b].

Otra observación que conviene señalar es que el método de Barlow puede producir una matriz bidiagonal superior de manera estable, pues este método es *backward stable*. Esta propiedad no puede ser totalmente garantizada para el método de Ralha y las razones de esto se pueden encontrar en [Ralha, 2003]. Sin embargo, hay ejemplos de matrices en que los dos métodos se comportan de modo semejante y existen matrices mal condicionadas en las cuales los dos métodos calculan todos los valores singulares con pequeños errores relativos

[Ralha, 2003], [Bosner y Drmač, 2005].

Para cerrar esta sección hay que resaltar que el análisis descrito para el método de Barlow es igualmente válido para el método de Barlow modificado.

3.5. Implementaciones desarrolladas

Todas las implementaciones desarrolladas en el marco de la tesis han sido codificadas en lenguaje Fortran 90 y se han empleado datos en doble precisión, según el estándar IEEE [IEEE, 1985]. Además, para obtener implementaciones de altas prestaciones, con un elevado grado de portabilidad, de legibilidad y de robustez, se ha empleado el mayor número posible de rutinas puestas a disposición por las librerías BLAS, LAPACK, BLACS, PBLAS y SCALAPACK.

A continuación son descritas las implementaciones secuenciales y paralelas que se han desarrollado para los métodos alternativos de bidiagonalización, y con las cuales se ha hecho el estudio comparativo entre las prestaciones obtenidas respecto a las rutinas de LAPACK y de SCALAPACK con la misma funcionalidad.

Hay que señalar que en los clusters Kefren y Odin de la UPV se han empleado las versiones 3.1.1 y 3.2 de LAPACK, y la versión 1.8.0 de SCALAPACK. En cambio, en el cluster Search de la UM se ha empleado la implementación de LAPACK y de SCALAPACK disponibles en la versión 8.1 de la librería MKL (*Math Kernel Library*) de Intel [webIntel®MKL, 2013].

Como es natural, las nuevas versiones de LAPACK y de MKL incorporan nuevas funcionalidades, resuelven errores encontrados en las versiones anteriores y presentan mejores prestaciones. En nuestro caso esto no es un problema adicional, pues debido al hecho de que todas nuestras implementaciones emplean módulos computacionales básicos de estas librerías, el hecho de que presenten mejores prestaciones también repercutirá en nuestras implementaciones.

3.5.1. Implementaciones secuenciales

La implementación secuencial del método de Ralha fue desarrollada según el esquema que se expone a continuación.

Implementación Secuencial 1 *RalhaBidiagonalizacion* (A, α, β, Q)

Para $r = 1, 2, \ldots, n-2$ **calcular:**
x_r usando DGEMV de BLAS
H_r usando DLARFG de LAPACK
$A_r = A_{r-1}H_r$ usando DLARF de LAPACK
Fin

Calcular α_1 usando DNRM2 de BLAS
Calcular q_1 usando DCOPY y DSCAL de BLAS

Para $r = 2, 3, \ldots, n$ **calcular:**
β_{r-1} usando DDOT de BLAS
$A_r = A_r - \beta_{r-1}q_{r-1}$ usando DAXPY de BLAS
α_r usando DNRM2 de BLAS
q_r usando DCOPY y DSCAL de BLAS
Fin

De modo similar fueron desarrolladas las implementaciones secuenciales del método de Barlow y del método de Barlow modificado. Como se puede observar en el ejemplo anterior, los métodos alternativos de bidiagonalización son, típicamente, métodos del tipo BLAS de nivel 2, pues algunas de las computaciones involucran operaciones del tipo matriz-vector.

Para obtener altas prestaciones, los métodos implementados en la librería LAPACK aplican rutinas de BLAS de nivel 3 y, por tanto, sacan el máximo provecho de las operaciones del tipo matriz-matriz. En particular, en el método de Golub-Kahan y en el método de la R-bidiagonalización, las transformaciones de Householder a bloques son aplicadas según la representación WY [Bischof y van Loan, 1987], [Schreiber y Parlett, 1988], [Schreiber y van Loan, 1989], y descrita en [Golub y van Loan, 1996, pp. 213–214, pp. 225–22].

Aunque nuestras implementaciones trabajen con rutinas de BLAS de nivel 2, en la tesis doctoral de Bosner se describe una implementación secuencial del método de Barlow orientada a bloques [Bosner, 2006]. En esa implementación, Bosner recurre a las ideas presentadas en [Howell *et al.*, 2008], donde los autores describen cómo usar de manera eficiente los datos almacenados en la memoria caché durante la bidiagonalización de matrices. Howell *et al.* describen cómo usar

las operaciones de tipo BLAS de nivel 2.5 en la aplicación de las transformaciones de Householder. En concreto, describen cómo aplicar las operaciones

$$\left\{ \begin{array}{l} x \leftarrow \beta A^t y + z \\ w \leftarrow \alpha A x \end{array} \right. \qquad \text{y} \qquad \left\{ \begin{array}{l} \widehat{A} \leftarrow A + u_1 v_1^t + u_2 v_2^t \\ x \leftarrow \beta \widehat{A}^t y + z \\ w \leftarrow \alpha \widehat{A} x \end{array} \right. \tag{3.33}$$

al método de Golub-Kahan, basado en [Dongarra *et al.*, 1989]. Como se observa, las operaciones del tipo BLAS de nivel 2.5 son las operaciones que pueden combinar dos o más operaciones del tipo BLAS de nivel 2 en la misma operación.

Bosner, en [Bosner, 2006] expone también un estudio de las propiedades numéricas de su implementación secuencial del método de Barlow orientada a bloques empleando las rutinas de BLAS de nivel 2.5 y demuestra que la nueva implementación es *backward stable* (teorema 3.6.4, pp. 127)[2], tal como en el método de Barlow. En sus trabajos, Bosner recurre a los conceptos descritos por Higham en [Higham, 2002a] para el análisis numérico de sus propuestas.

Además, en [Bosner, 2006] y en [Bosner y Barlow, 2007] se describe que dependiendo de la dimensión de las matrices y del tamaño del bloque matricial empleado, los tiempos de ejecución de la implementación secuencial orientada a

[2]

Teorema 2 (Bosner, T. 3.6.4) *Si $\widetilde{B}$ es la matriz bidiagonal superior calculada por el algoritmo secuencial orientado a bloques, entonces existen, una matriz ortogonal $\widehat{P} \in \mathbb{R}^{(m+n)\times(m+n)}$, una matriz ortogonal $\widehat{V} \in \mathbb{R}^{n\times n}$, y perturbaciones, ΔA y δA tales que*

$$\begin{bmatrix} \widetilde{B} \\ 0 \end{bmatrix} = \widehat{P}^t \begin{bmatrix} \Delta A \\ A + \delta A \end{bmatrix} \widehat{V} \quad y \quad \left\| \begin{bmatrix} \Delta A \\ A + \delta A \end{bmatrix} \right\|_F \leq \xi \left\| A \right\|_F$$

donde $0 \leq \xi \leq \mathcal{O}\left(b\left(mn + n^3\right)\right)\epsilon$ y b es el número de columnas del bloque matricial.

Además, la aproximación $\widetilde{V}$ de la matriz $\widehat{V}$ verifica

$$\left\| \widetilde{V} - \widehat{V} \right\|_F \leq \mathcal{O}\left(n^2\right)\epsilon$$

y existe una matriz ortogonal $\widehat{U}$ y una perturbación $\delta\widehat{A}$ tal que

$$A + \delta\widehat{A} = \widehat{U}\widetilde{B}\widehat{V}^t$$

con

$$\left\| \delta\widehat{A} \right\|_F \leq \sqrt{2}\xi \left\| A \right\|_F .$$

bloques son menores que los tiempos de ejecución de las correspondientes rutinas de LAPACK, trabajando en simple precisión.

Al mismo tiempo, en los mismos documentos, se llega a la conclusión que para una implementación paralela destinada a sistemas de memoria distribuida, el método no orientado a bloques es mejor que el método orientado a bloques, pues en el método orientado a bloques los tiempos de ejecución obtenidos son más elevados debido al mayor número de comunicaciones.

3.5.2. Implementaciones paralelas

Las primeras implementaciones paralelas desarrolladas en el marco de la tesis han sido obtenidas "convirtiendo" las llamadas de las rutinas de BLAS y de LAPACK, en las correspondientes rutinas de PBLAS y de SCALAPACK.

En este proceso de conversión se han aplicado las ideas expuestas en el apéndice B de [Blackford *et al.*, 1997], donde se describe cómo trasladar una implementación secuencial en LAPACK a una implementación paralela en SCALAPACK. Sin embargo, hay que tener en cuenta que esta metodología obliga a trabajar con la distribución de datos empleada por SCALAPACK, o sea, una distribución cíclica de bloques de datos, sobre una malla bidimensional de procesadores[3].

En la Implementación Paralela 1, que se presenta a continuación, está señalado el esquema de la implementación paralela del método de Barlow empleando esta metodología, donde se puede observar las rutinas empleadas, y además, al final de su ejecución, cada procesador almacena una copia de los vectores con los elementos de la matriz bidiagonal superior. Para ello, es necesario emplear rutinas de BLACS [Dongarra, 1991], [Dongarra y van de Geijn, 1993], [Dongarra y Whaley, 1995], [webBLACS, 1997], para la difusión de los elementos locales a cada procesador.

Aunque las rutinas de PBLAS y de SCALAPACK presentan altas prestaciones, Ralha señala que las transformaciones unilaterales siguen siendo ventajosas cuando son aplicadas en la bidiagonalización de matrices reales densas y en concreto, en implementaciones paralelas destinadas a sistemas de memoria distribuida.

[3] Para más información, consultar el Apéndice "BLAS, LAPACK y SCALAPACK".

Implementación Paralela 1 *BarlowBidiagonalizacion* (A, α, β, Q)

Obtiene información respecto a los procesadores usando BLACS_PINFO

Para $r = 1, 2, \ldots, n-2$ **calcular:**
- α_r usando PDNRM2 de PBLAS
- Broadcast de α_r usando DGEBS2D/DGEBR2D de BLACS
- q_r usando PDCOPY y PDSCAL de PBLAS
- x_r usando PDGEMV de PBLAS
- H_r usando PDLARFG de SCALAPACK
- $A(:, r+1:n) = A(:, r+1:n) H_r$ usando PDLARF de SCALAPACK
- Broadcast de β_r usando DGEBS2D/DGEBR2D de BLACS
- $A(:, r+1) = A(:, r+1) - \beta_r q_r$ usando PDAXPY de PBLAS

Fin

Calcular α_{n-1} usando PDNRM2 de PBLAS
Broadcast de α_{n-1} usando DGEBS2D/DGEBR2D de BLACS
Calcular q_{n-1} usando PDCOPY y PDSCAL de PBLAS
Calcular β_{n-1} usando PDDOT de PBLAS
Broadcast de β_{n-1} usando DGEBS2D/DGEBR2D de BLACS
Calcular $A(:, n) = A(:, n) - \beta_{n-1} q_{n-1}$; usando PDAXPY de PBLAS
Calcular α_n usando PDNRM2 de PBLAS
Broadcast de α_n usando DGEBS2D/DGEBR2D de BLACS
Calcular q_n usando PDCOPY y PDSCAL de PBLAS

Según [Ralha, 1994], la mejor descomposición de los datos consiste en atribuir a cada uno de los p procesadores empleados en la ejecución, un número de filas de la matriz a transformar y, por tanto, cada procesador almacena un segmento de dimensión m/p de cada una de las columnas (por simplicidad, asumimos que p es un divisor de m; si eso no es así, entonces cada procesador almacena $\lfloor m/p \rfloor$ ó $\lfloor m/p \rfloor + 1$ filas).

Como ha expuesto Ralha, en cada paso r del método, cada procesador recibe una copia completa del vector x_r y a continuación calcula su propio vector de Householder. Es evidente que con esta estrategia hay una redundancia en las computaciones, pues todos los procesadores tienen que calcular el mismo vector

v_r, pero la influencia negativa de estas computaciones no es dramática en el coste total del método siempre que la razón m/p sea grande.

Al mismo tiempo, las ventajas de esta estrategia son evidentes: hay un buen balance de la carga y los procesadores solamente tienen que hacer comunicaciones a la hora de calcular los $n-r$ productos internos necesarios en el cálculo de la norma α_r y del vector x_r.

Esta fue también la distribución de datos que empleó Bosner en sus trabajos.

Considerando la distribución de datos descrita en [Ralha, 1994], se han desarrollado experimentos computacionales ejecutando las implementaciones paralelas sobre mallas bidimensionales de procesadores del tipo $p \times 1$. En este tipo de mallas, ya que los procesadores están organizados según una única columna de procesadores, al aplicar la distribución cíclica de bloques de datos, cada procesador va a recibir las m/p filas.

Seguidamente fueron desarrolladas nuevas implementaciones paralelas para los métodos alternativos de bidiagonalización, con el objetivo de evitar la directa dependencia de las rutinas de SCALAPACK. Como fue expuesto en la sección 3.3, en el método de Barlow modificado, todos los procesadores empleados están involucrados en el cálculo del vector x_r con los $n-r+1$ productos internos, y posteriormente, cada procesador calcula localmente α_r y actualiza su segmento local de q_r. Además, el cálculo de α_{n-1} y β_{n-1} es igualmente local. Así, la implementación paralela del método de Barlow modificado es esencialmente la implementación secuencial con un único mecanismo de comunicación necesario para el cálculo del vector x_r, el cual contiene los productos internos realizados entre las columnas apropiadas de la matriz A. Lo mismo ocurre con los otros métodos alternativos de bidiagonalización.

En ese sentido, la Implementación Paralela 2 describe la implementación paralela del método de Barlow modificado, en la cual se observa que para las computaciones se han empleado rutinas de BLAS y de LAPACK, y que para las comunicaciones se han empleado rutinas de BLACS. Así, en esta implementación no se utilizan rutinas de SCALAPACK.

Implementación Paralela 2 $Barlow Modificado\,(A, \alpha, \beta)$

Obtiene información respecto a los procesadores usando BLACS_PINFO

Para $r = 1, 2, \ldots, n-2$ **calcular:**
x_r local usando DGEMV de BLAS
x_r global en P_0 usando DGSUM2D de BLACS
Broadcast de x_r global usando DGEBS2D/DGEBR2D de BLACS
$\alpha_r = \sqrt{x_r\,(1)}$ local
$A\,(:, r) = \dfrac{A\,(:, r)}{\alpha_r}$ local usando DSCAL de BLAS
H_r local usando DLARFG de LAPACK
$A\,(:, r+1 : n) = A\,(:, r+1 : n)\,H_r$ local usando DLARF de LAPACK
$\beta_r = \dfrac{x_r\,(2)}{\alpha_r}$ local
$A\,(:, r+1) = A\,(:, r+1) - \beta_r A\,(:, r)$ local usando DAXPY de BLAS
Fin

Calcular x_{n-1} local usando DGEMV de BLAS
Calcular x_{n-1} global en P_0 usando DGSUM2D de BLACS
Broadcast de x_{n-1} global usando DGEBS2D/DGEBR2D de BLACS
Calcular α_{n-1} local
Calcular β_{n-1} local
Calcular $A\,(:, n-1)$ local usando DSCAL de BLAS
Calcular $A\,(:, n) = A\,(:, n) - \beta_{n-1} A\,(:, n-1)$ local usando DAXPY de BLAS
Calcular x_n local usando DDOT de BLAS
Calcular x_n global en P_0 usando DGSUM2D de BLACS
Broadcast de x_n global usando DGEBS2D/DGEBR2D de BLACS
Calcular α_n local

Finalmente, las últimas implementaciones paralelas desarrolladas para los métodos alternativos de bidiagonalización se han obtenido sustituyendo en las implementaciones anteriores las llamadas de las rutinas de BLACS por las correspondientes rutinas del MPI [webMPI, 2014], [Pacheco, 1997], [Pacheco, 1998] y, por lo tanto, estas implementaciones no tienen dependencia con SCALAPACK ni emplean su distribución de bloques cíclica.

3.6. Resultados experimentales

Seguidamente se describe el estudio comparativo entre las prestaciones obtenidas por las implementaciones secuenciales y paralelas desarrolladas en el marco de la tesis y las correspondientes rutinas de LAPACK y de SCALAPACK con la misma funcionalidad.

Los resultados obtenidos resultan de un elevado número de experimentos computacionales, empleando un gran número de casos de prueba. Además, al emplear las rutinas de SCALAPACK, siempre se ha procurado obtener sus mejores prestaciones y, para ello, se ha estudiado cual es la mejor configuración de la malla bidimensional de procesadores y cual es el mejor tamaño del bloque matricial a emplear. Por ejemplo, al emplear 8 procesadores, se ha ejecutado la rutina de SCALAPACK empleando las cuatro configuraciones posibles para la malla bidimensional de procesadores: 1×8, 2×4, 4×2 y 8×1. Al mismo tiempo, para cada configuración, se han empleado las dimensiones 8×8, 16×16, 32×32, 64×64 y 128×128 para el tamaño del bloque matricial.

Aunque SCALAPACK no imponga ninguna restricción a los dos parámetros descritos anteriormente, las prestaciones obtenidas con sus rutinas son mejores cuando se definen mallas cuadradas de procesadores y en el caso de las mallas rectangulares, la configuración debe emplear un mayor número de columnas que de filas. Así, en el caso anterior, las mejores prestaciones fueron obtenidas en la malla del tipo 2×4 y empleando el bloque matricial de dimensión 32×32.

Para testar todas las implementaciones desarrolladas respecto a los métodos alternativos de bidiagonalización se ha escogido un vasto conjunto de matrices de prueba. En los primeros experimentos computacionales, las matrices empleadas fueron generadas recurriendo a la rutina `DLATMS`[4] de LAPACK [Demmel y McKenney, 1989]. Seguidamente fue desarrollado un módulo auxiliar con las matrices de prueba descritas por Higham en [Higham, 1991] y en [Higham, 2002b]. Además, en éste módulo se ha incluido también la matriz de Kahan empleada en los casos de prueba de [Barlow *et al.*, 2005, Ejemplo 6.1].

Dado que las prestaciones obtenidas, empleando los distintos tipos de matrices, permiten describir las mismas conclusiones, hemos decidido hacer el estudio comparativo relativo a las prestaciones obtenidas empleando matrices aleatorias, y en cuanto a la dimensión de las matrices, se han empleado dos tipos

[4] Para más información, consultar el Apéndice "Rutinas de LAPACK y SCALAPACK".

de casos:

- Matrices cuadradas cuya dimensión varía de 1000 hasta 4500.
- Matrices rectangulares con un número fijo de 10000 filas y donde el número de columnas varía de 1000 hasta 4500.

En el caso de las matrices rectangulares se ha escogido un tipo de matrices donde el número de filas es mucho mayor que 5/3 el número de columnas. En este tipo de matrices, como ya fue descrito, el algoritmo secuencial empieza por aplicar una descomposición QR inicial.

Además, todos los tiempos de ejecución están en segundos y cumplen con los indicadores de prestaciones descritos en la sección 1.5. Sin embargo, aunque los sistemas computacionales presenten nodos que son biprocesadores, esta característica no ha sido empleada en los experimentos computacionales realizados y así, todos los tiempos de ejecución obtenidos resultan de la ejecución de un único proceso en cada nodo, con lo cual, los sobrecostes de comunicación se deben en gran parte a latencias y a tiempos de transferencia entre los nodos de la red.

3.6.1. Prestaciones de DGEBRD y de PDGEBRD

En el caso secuencial, la rutina `DGEBRD`[5] de LAPACK reduce una matriz real densa a la forma bidiagonal (superior), empleando el método de Golub-Kahan o el método de la R-Bidiagonalización, dependiendo de la dimensión de la matriz inicial.

Considerando que la bidiagonalización es la primera etapa del cálculo de la DVS de una matriz real densa, la Tabla 3.4 compara los tiempos de ejecución obtenidos por `DGEBRD` (casos con B) y por `DGESVD`[5] (casos con Σ) para matrices cuadradas.

Como se puede observar, la bidiagonalización tiene un coste computacional importante en el cálculo de la DVS de una matriz real densa. Para matrices pequeñas casi todo el tiempo es dedicado en la bidiagonalización, mientras que en los casos con mayor dimensión, esta etapa supera el 60 % del tiempo total.

[5] Para más información, consultar el Apéndice "Rutinas de LAPACK y SCALAPACK".

Calcula	**2000**	**2500**	**3000**	**3500**	**4000**
Σ	23.73	45.79	73.23	129.38	183.43
B	23.43	45.34	72.56	128.48	182.28
$\Sigma + U$	40.13	77.98	125.89	211.36	303.76
$B + U$	27.00	52.34	84.75	147.42	210.70
$\Sigma + V^t$	40.85	77.83	126.77	212.51	305.52
$B + V^t$	27.46	52.84	85.46	148.32	212.01
$\Sigma + U + V^t$	56.80	108.90	178.95	293.57	424.66
$B + U + V^t$	31.05	59.72	97.65	167.26	240.43

Tabla 3.4: Tiempos de ejecución de `DGESVD` y de `DGEBRD` en Odin.

La Tabla 3.5 describe en detalle algunos de los tiempos de ejecución de la Tabla 3.4. Una vez concluida la ejecución de la rutina `DGEBRD`, se aplica la rutina `DORGBR` para el cálculo de los vectores ortogonales que reducen la matriz inicial a la forma bidiagonal superior, los cuales son almacenados en las matrices U y V^t, y construidos a partir de la parte esencial de los vectores de Householder almacenada en la matriz de entrada.

Calcula	**3000**	**3500**	**4000**
B	72.56	128.48	182.28
$B + U$	72.56 + 12.19	128.48 + 18.94	182.28 + 28.42
$B + V^t$	72.56 + 12.90	128.48 + 19.84	182.28 + 29.78
$B + U + V^t$	72.56 + 12.19 + 12.90	128.48 + 18.94 + 19.84	182.28 + 28.42 + 29.73

Tabla 3.5: Tiempos de ejecución de `DGEBRD` para matrices cuadradas en Odin.

La rutina `DGEBRD` implementa el método de Golub-Kahan orientado a bloques matriciales basado en la representación WY [Bischof y van Loan, 1987], [Schreiber y van Loan, 1989], y así puede sacar el máximo provecho de las rutinas de BLAS de nivel 3. En cambio, la rutina `DGEBD2` es la implementación del mismo método empleando solamente rutinas de BLAS de nivel 2. Esta rutina es utilizada por la rutina `DGEBRD` para reducir a la forma bidiagonal superior el conjunto de columnas de la matriz inicial que no entran en la representación

WY, es decir, que se quedan fuera de la distribución general a bloques que requiere `DGEBRD`[6].

Como es natural, la rutina `DGEBRD` presenta mejores prestaciones respecto a la rutina `DGEBD2`. Eso puede ser confirmado en la Figura 3.1 para el caso cuadrado y para el caso rectangular.

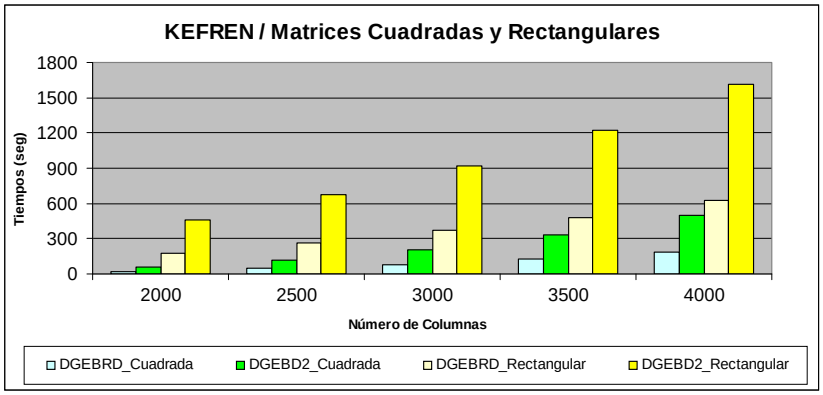

Figura 3.1: Tiempos de ejecución de `DGEBRD` y de `DGEBD2`.

Para terminar el estudio secuencial, la Figura 3.2 ilustra los tiempos de ejecución obtenidos en el cálculo de los valores singulares de matrices rectangulares con $m \geq \frac{5}{3}n$. En este problema concreto, la rutina `DGESVD` pasa por tres etapas fundamentales: la aplicación de la descomposición QR inicial, empleando la rutina `DGEQRF`; la bidiagonalización de la matriz triangular superior R, empleando la rutina `DGEBRD`; el cálculo de los valores singulares de la matriz bidiagonal superior, empleando la rutina `DBDSQR`. Como se observa, la bidiagonalización de la matriz triangular superior R es la etapa que necesita de más recursos en la rutina `DGESVD`. La Tabla 3.6 describe los tiempos de ejecución de la Figura 3.2.

[6]Para todas las rutinas de LAPACK que implementan métodos orientados a bloques matriciales, la rutina `ILAENV` define el tamaño del bloque matricial a emplear y este parámetro suele ser afinado durante el proceso de instalación de LAPACK.

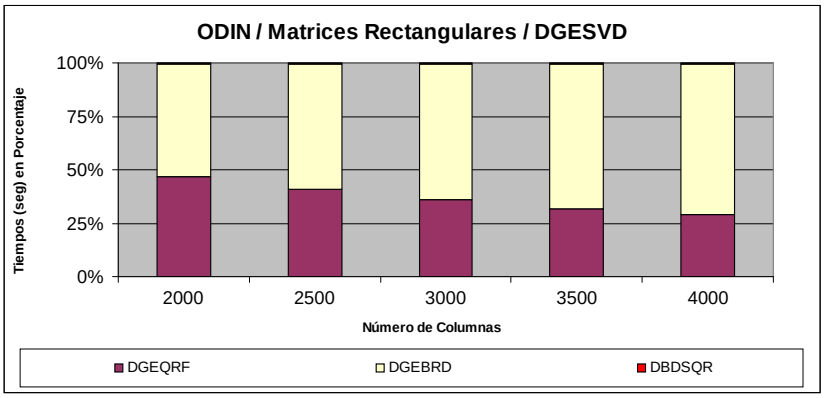

Figura 3.2: Tiempos de ejecución de `DGESVD` para matrices rectangulares.

	2000	**2500**	**3000**	**3500**	**4000**
`DGEQRF`	21.94	32.68	44.94	59.14	76.39
`DGEBRD`	24.57	46.62	79.86	125.38	185.92
`DBDSQR`	0.28	0.52	0.87	0.96	1.09

Tabla 3.6: Tiempos de ejecución de `DGESVD` para matrices rectangulares.

Dado que uno de los objetivos principales de la investigación desarrollada en el marco de la tesis es encontrar los mejores indicadores de prestaciones de las implementaciones desarrolladas o empleadas, los primeros experimentos prácticos desarrollados con las rutinas de SCALAPACK fueron dirigidos a estudiar las prestaciones de la rutina `PDGEBRD`[7], la cual es la correspondiente implementación paralela de la rutina `DGEBRD` de LAPACK.

En ese sentido, se ha testado la rutina `PDGEBRD` con diferentes tamaños de bloques en la distribución cíclica de los datos y para las distintas posibilidades de definir la malla de procesadores con p procesadores fijos. En estos experimentos

[7] Para más información, consultar el Apéndice "Rutinas de LAPACK y SCALAPACK".

computacionales se ha empleado un máximo de 10 procesadores, pues se han usado los recursos necesarios para demostrar las tendencias en las prestaciones.

Aunque SCALAPACK no imponga ninguna restricción en la dimensión del bloque matricial con el cual se realiza la distribución cíclica de datos, la guía de usuario aconseja emplear tamaños de bloque suficientemente grandes, y habitualmente, por simplicidad, se suele emplear bloques cuadrados [Blackford *et al.*, 1997, §5].

Las tablas que se encuentran en el apéndice B (Tablas B.1−B.5) exponen los tiempos de ejecución obtenidos por `PDGEBRD` en el caso cuadrado, empleando bloques matriciales de tamaño 32×32, 64×64 y 128×128, en la distribución cíclica de los datos y para todas las configuraciones posibles con 2, 4, 6, 8 y 10 procesadores. Los tiempos de ejecución descritos fueron obtenidos en el cluster Kefren y en el cluster Odin. Estos tiempos de ejecución permiten obtener las mismas conclusiones.

Respecto al tamaño del bloque matricial, los mejores tiempos de ejecución de la rutina `PDGEBRD` son obtenidos con el bloque de tamaño 32×32 y además, el tamaño del bloque matricial óptimo es independiente de la dimensión de la matriz y de la distribución de la malla de procesadores. En cambio, la mejor configuración para la malla de procesadores es la que está descrita en la Tabla 3.7, y los mejores tiempos de ejecución son obtenidos en las mallas de procesadores cuadradas. En las mallas de procesadores rectangulares los mejores tiempos de ejecución son conseguidos cuando el número de procesadores por filas es menor que por columnas. Además, el estudio desarrollado respecto a la rutina `PDGEBRD` conduce a las mismas conclusiones en el caso de las matrices rectangulares, aunque los tiempos de ejecución sean más elevados.

Número de Procesadores	**Configuración**
2	1×2
4	2×2
6	2×3
8	2×4
10	2×5

Tabla 3.7: Mejor configuración de la malla de procesadores.

La Figura 3.3 ilustra los tiempos de ejecución obtenidos por la rutina `DGEBRD` y por la rutina `PDGEBRD`, en el caso de las matrices cuadradas y empleando el bloque matricial de tamaño 32×32, hasta 10 procesadores. Como se observa, la rutina `PDGEBRD` reduce el tiempo de ejecución respecto a la rutina secuencial al usar más procesadores e incluso la implementación paralela ejecutada en un solo procesador permite obtener tiempos de ejecución menores que con la rutina `DGEBRD`. Así, y debido a ello, a la hora de valorar la aceleración de ejecución (*speed-up*) y la eficiencia, se ha considerado como tiempo secuencial el tiempo de ejecución de la implementación paralela en un solo procesador.

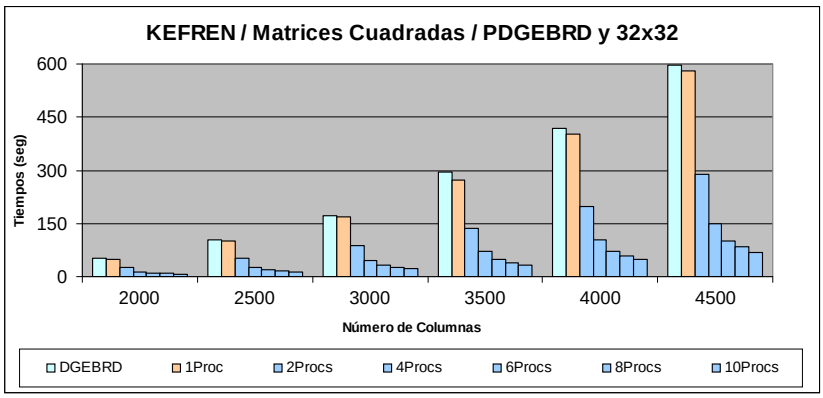

Figura 3.3: Tiempos de ejecución de `DGEBRD` y de `PDGEBRD`.

Seguidamente las implementaciones que permiten testar las rutinas `DGEBRD` y `PDGEBRD` fueron ejecutadas en el cluster Odin. La Tabla 3.8 presenta los correspondientes tiempos de ejecución, en el caso de las matrices rectangulares, empleando el bloque matricial de tamaño 32×32 y hasta 40 procesadores. En estos experimentos computacionales no se ha realizado la descomposición QR inicial propuesta por el método de Lawson-Hanson-Chan y, por lo tanto, los tiempos de ejecución obtenidos por las rutinas `DGEBRD` y `PDGEBRD` hacen referencia al mismo algoritmo. Además, los tiempos de ejecución obtenidos por la rutina `PDGEBRD` en un solo procesador son menores que con la rutina `DGEBRD`.

	2000	**2500**	**3000**	**3500**	**4000**
DGEBRD	179.16	264.43	373.57	482.99	628.77
1 Proc (1×1)	136.08	207.32	291.89	388.56	496.08
2 Procs (1×2)	75.84	113.36	157.09	206.97	262.92
4 Procs (2×2)	43.47	64.02	88.48	116.45	148.40
8 Procs (2×4)	27.38	38.71	52.22	67.07	83.54
9 Procs (3×3)	23.27	31.74	42.02	53.55	66.65
12 Procs (3×4)	19.61	27.21	36.13	45.23	55.27
16 Procs (4×4)	14.62	20.96	28.41	36.09	43.98
20 Procs (4×5)	14.10	18.80	24.90	31.74	38.70
25 Procs (5×5)	12.44	15.98	20.87	26.54	32.84
30 Procs (5×6)	11.25	15.72	20.03	25.35	30.32
32 Procs (4×8)	11.89	16.21	20.75	25.69	30.92
36 Procs (6×6)	9.78	13.35	17.19	21.48	26.14
40 Procs (5×8)	9.80	13.26	16.82	20.79	24.98

Tabla 3.8: Tiempos de ejecución de `DGEBRD` y de `PDGEBRD` para matrices rectangulares.

La Figura 3.4 ilustra los valores del *speed-up* obtenido según (1.8), teniendo en cuenta los tiempos de ejecución de la Tabla 3.8. Los valores mostrados no son para valorar, en realidad, la escalabilidad de la implementación paralela, pues lo que se ha hecho fue mantener fija la dimensión del problema y medir los tiempos de ejecución que se obtienen con el incremento del número de procesadores. En este tipo de prueba se puede observar que la aportación de más recursos computacionales permite calcular la matriz bidiagonal superior de manera más rápida pero perdiendo eficiencia.

Para valorar la escalabilidad de la rutina `PDGEBRD` se han empleado matrices rectangulares con un número fijo de 4000 columnas y donde se incrementa proporcionalmente el número de filas con el número de procesadores. Como fue comentado en el capítulo 2, la bidiagonalización tiene un coste computacional proporcional a $\mathcal{O}\left(mn^2\right)$ y, por tanto, el coste computacional varía linealmente con el número de filas de la matriz, y varía cuadráticamente con el número de columnas de la matriz.

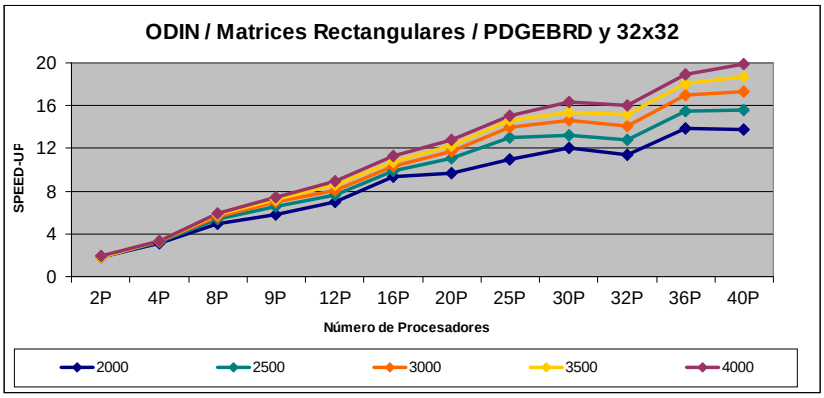

Figura 3.4: Aceleración de la ejecución para matrices rectangulares.

La Figura 3.5 muestra que la escalabilidad de `PDGEBRD` es buena hasta 32 procesadores.

3.6.2. Prestaciones en secuencial

En este apartado se exponen los tiempos de ejecución obtenidos por las implementaciones secuenciales de los métodos alternativos de bidiagonalización y las rutinas de LAPACK.

Así, la Figura 3.6 ilustra los tiempos de ejecución obtenidos por la rutina `DGEBRD` y por las implementaciones secuenciales de los métodos de Ralha, de Barlow y de Barlow modificado, para matrices rectangulares y donde no se ha empleado la descomposición QR inicial por el motivo que ya fue descrito. En la figura, los tiempos de ejecución se refieren al cálculo de la matriz bidiagonal superior y, por lo tanto, las matrices ortogonales U_B y V_B no son calculadas. Además, se puede observar que los tiempos de ejecución de las implementaciones secuenciales de los métodos alternativos de bidiagonalización son semejantes y al mismo tiempo, para algunos casos, los tiempos de ejecución son menores que los tiempos de ejecución de `DGEBRD`. Sin embargo, eso no puede ser garantizado para

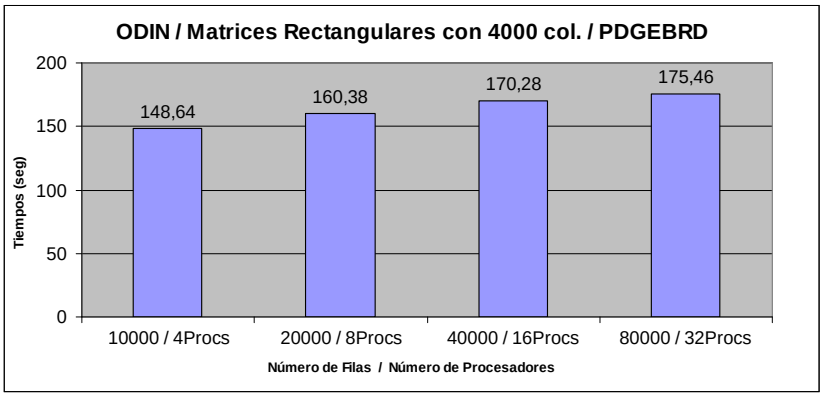

Figura 3.5: Escalabilidad de `PDGEBRD` para matrices rectangulares.

todos los casos, pues para las matrices de mayor dimensión, la rutina `DGEBRD` presenta mejores prestaciones, aunque la diferencia no sea muy significativa.

La Figura 3.6 permite concluir que las prestaciones de las implementaciones secuenciales de los métodos alternativos de bidiagonalización son competitivas respecto a la rutina de LAPACK. Además, los métodos alternativos de bidiagonalización tienen un coste computacional de $3mn^2$ flops, mientras que la rutina `DGEBRD` tiene un coste computacional de $4mn^2 - \frac{4}{3}n^3$ y, por tanto, los métodos alternativos de bidiagonalización tienen menos operaciones en punto flotante por segundo siempre que $m \geq \frac{4}{3}n$. Sin embargo, para un m fijo, los métodos que estamos estudiando son menos competitivos respecto a `DGEBRD` a la medida que n aumenta.

La Figura 3.7 ilustra los tiempos de ejecución del problema similar al anterior, pero ahora empleando la descomposición QR inicial, tal como debe ocurrir en este tipo de matrices ya que $m \geq \frac{5}{3}n$. En todos los tiempos de ejecución expuestos, el cálculo de la descomposición QR inicial (realizado por la rutina `DGEQRF`) tiene el mismo tiempo de ejecución y, por tanto, la diferencia entre los tiempos de ejecución obtenidos resultan de los diferentes métodos usados en la bidiagonalización de la matriz triangular superior R. Además,

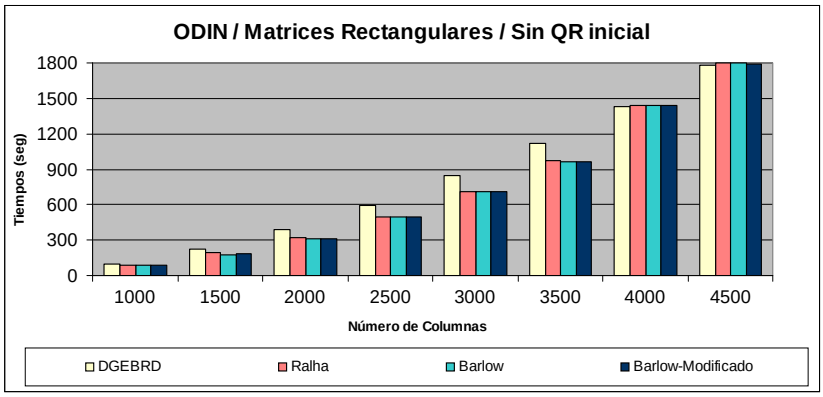

Figura 3.6: Tiempos de ejecución secuenciales para matrices rectangulares sin la descomposición QR inicial.

se puede observar que los tiempos de ejecución de las implementaciones secuenciales de los métodos alternativos de bidiagonalización son semejantes y el incremento del tiempo de ejecución respecto a `DGEBRD` no es muy significativo, aunque para las matrices de mayor dimensión esa diferencia sea más elevada. Como se esperaba, los tiempos de ejecución ilustrados en la Figura 3.7 son bastante menores a los tiempos de ejecución ilustrados en la Figura 3.6.

Conclusiones semejantes se desprenden en el caso de las matrices cuadradas y, por tanto, las implementaciones secuenciales de los métodos alternativos de bidiagonalización obtienen tiempos de ejecución que, casi siempre, son competitivas respecto a la rutina de LAPACK. Además, como ya fue comentado en este capítulo, hay que considerar que la rutina `DGEBRD` implementa un método orientado a bloques matriciales, empleando la representación WY en las transformaciones de Householder, para que se puedan agrupar y aplicar de una sola vez, realizando la actualización de la matriz según $A - UW^t - YV^t$, y para ello, emplea dos llamadas de la rutina `DGEMM` de BLAS de nivel 3, mientras que las implementaciones secuenciales de los métodos alternativos de bidiagonalización solamente emplean rutinas de BLAS de nivel 2.

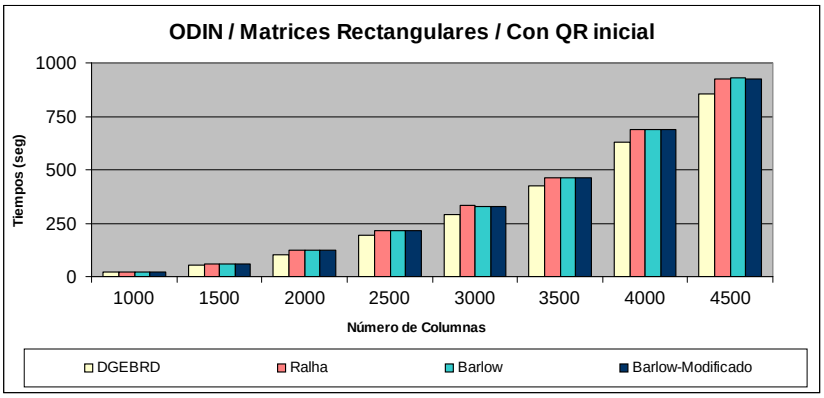

Figura 3.7: Tiempos de ejecución secuenciales para matrices rectangulares con la descomposición QR inicial.

Como también ya fue comentado, Bosner desarrolló una implementación secuencial del método de Barlow "orientada a bloques" y describe que consigue obtener tiempos de ejecución de su implementación secuencial que son bastante mejores que las prestaciones de la rutina `SGEBRD`, pues en sus experimentos computacionales todos los datos están en simple precisión.

En esta implementación secuencial "orientada a bloques", Bosner emplea las rutinas de BLAS de nivel 2.5 y, por tanto, tampoco consigue desarrollar una implementación secuencial del método de Barlow en la cual se emplean rutinas de BLAS de nivel 3. Además, Bosner también afirma que aunque las prestaciones de su implementación secuencial del método de Barlow con rutinas de BLAS de nivel 2.5 sean competitivas respecto a LAPACK, esta metodología no es eficiente para el desarrollo de la correspondiente implementación paralela, dado que los tiempos de ejecución obtenidos son mucho peores que la implementación paralela que se obtiene adaptando el código secuencial en LAPACK a un código paralelo en SCALAPACK.

Volviendo a las implementaciones secuenciales desarrolladas en el marco de la tesis, en la exposición del método de Ralha y de los demás métodos alternativos

de bidiagonalización, se ha descrito que en estos métodos siempre se hace la acumulación de las transformaciones de Householder y, por tanto, siempre se calcula la matriz ortogonal U_B, independientemente del problema que se necesita resolver. En ese sentido, para hacer una comparativa más justa con la rutina `DGEBRD` hay que añadir, por lo menos, el cálculo de la matriz U_B a la ejecución de la rutina de LAPACK. Así, la Figura 3.8 y la Figura 3.9 ilustran los tiempos de ejecución de `DGEBRD` y de la implementación secuencial del método de Barlow modificado, obtenidos en los clusters Search y Kefren, para el caso de las matrices rectangulares, sin emplear la descomposición QR inicial y donde se calculan las matrices B y U_B. Como se observa, los tiempos de ejecución de la implementación secuencial del método de Barlow modificado son menores que los de `DGEBRD`, a la cual se añade el cálculo de la matriz U_B. Además, los tiempos de ejecución obtenidos en Odin son semejantes y refuerzan nuestras conclusiones.

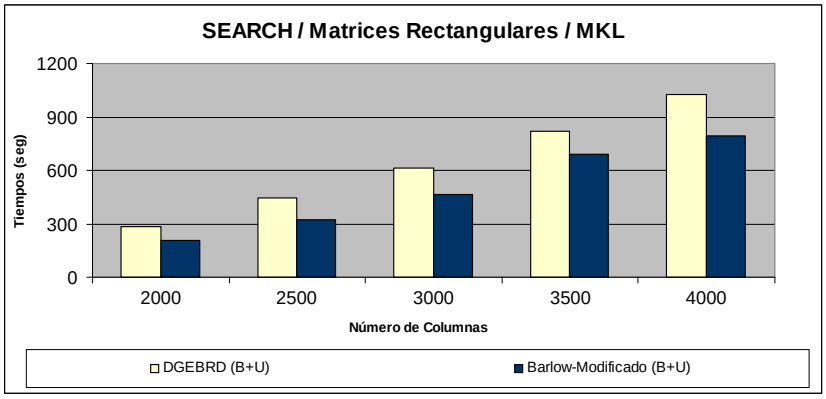

Figura 3.8: Tiempos de ejecución para matrices rectangulares en Search.

Siguiendo en la misma línea de experimentos computacionales, la Figura 3.10 ilustra los tiempos de ejecución obtenidos por las rutinas `DGEBRD` y `DGEBD2` de LAPACK, y la implementación secuencial del método de Barlow modificado. En las rutinas de LAPACK solo se calcula la matriz bidiagonal superior, mientras

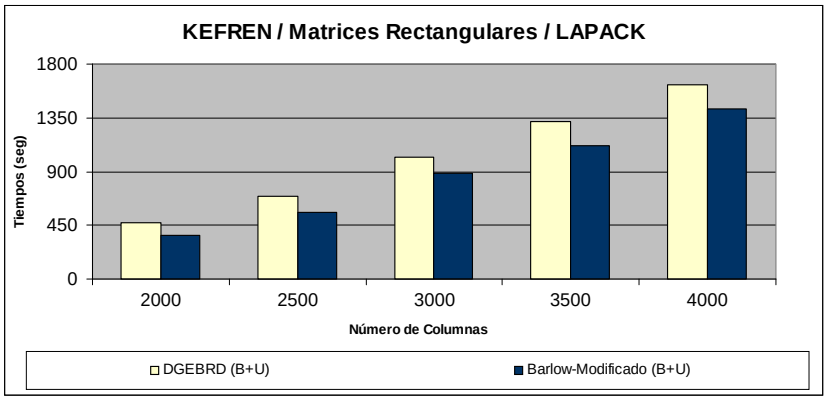

Figura 3.9: Tiempos de ejecución para matrices rectangulares en Kefren.

que en el método de Barlow modificado también se hace la acumulación de las transformaciones de Householder. Además, como fue comentado en la sección anterior, la rutina `DGEBD2` implementa el método de Golub-Kahan empleando solamente rutinas de BLAS de nivel 2.

Al mismo tiempo, la Figura 3.10 tiene importancia pues permite observar la diferencia entre los tiempos de ejecución que hay entre las dos rutinas de LAPACK y la diferencia que hay entre los tiempos de ejecución de la implementación secuencial del método de Barlow modificado respecto a la rutina `DGEBD2`, donde ahora se está haciendo una comparativa entre los tiempos de ejecución obtenidos por dos implementaciones distintas para reducir matrices densas a la forma bidiagonal superior e implementadas de manera similar.

3.6.3. Prestaciones en paralelo

El estudio que se describe a continuación empieza por comparar los tiempos de ejecución obtenidos con la rutina `PDGEBRD` de SCALAPACK y los tiempos de ejecución obtenidos con las implementaciones paralelas de los métodos alternativos de bidiagonalización que fueron desarrolladas trasladando el código

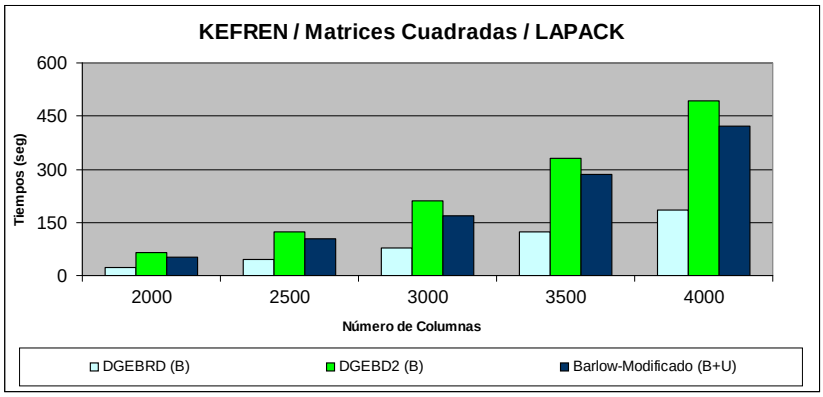

Figura 3.10: Tiempos de ejecución para matrices cuadradas en Kefren.

secuencial en LAPACK a un código paralelo en SCALAPACK.

Las implementaciones paralelas fueron ejecutadas empleando bloques matriciales de tamaño 32×32 para la distribución cíclica de los datos y las configuraciones de las mallas de procesadores descritas en la Tabla 3.7, que son en las que mejores tiempos de ejecución obtuvieron la rutina de SCALAPACK. En este estudio se está favoreciendo la rutina `PDGEBRD` con respecto a los métodos alternativos de bidiagonalización, pues ellos obtienen mejores tiempos de ejecución en mallas de procesadores del tipo $p \times 1$, ya que en este tipo de configuración cada procesador recibe un bloque de m/p filas de la matriz, obteniéndose así una distribución de datos más cercana a la descrita en [Ralha, 1994].

La Figura 3.11 ilustra los tiempos de ejecución obtenidos con un solo procesador, con las matrices rectangulares y donde no se ha aplicado la descomposición QR inicial debido al hecho de que `PDGEBRD` no la aplica. Se observa que las implementaciones paralelas de los métodos alternativos de bidiagonalización obtienen tiempos de ejecución semejantes y además, son mejores que los tiempos de ejecución de `PDGEBRD`. Aunque no sea muy significativo, los mejores tiempos de ejecución son los del método de Barlow

modificado, el cual ha sido desarrollado con el objetivo de reducir los costes de comunicaciones.

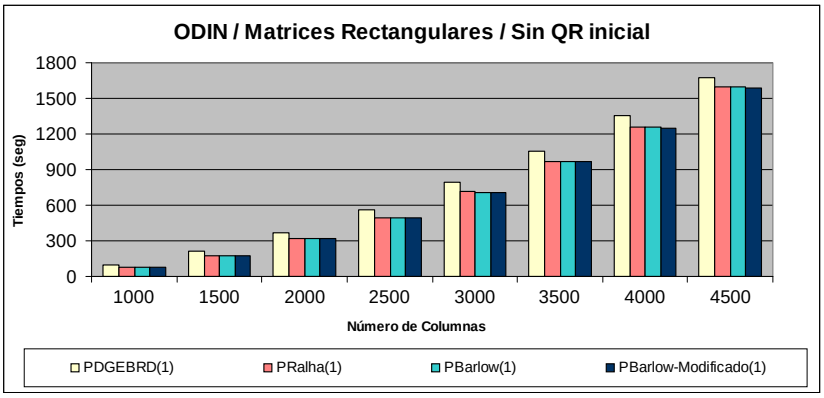

Figura 3.11: Tiempos de ejecución con un solo procesador sin QR inicial.

A partir de este punto, los tiempos de ejecución obtenidos con las implementaciones del método de Ralha no van a estar ilustradas en las figuras debido al hecho de que son muy semejantes a los tiempos de ejecución obtenidos con las implementaciones del método de Barlow.

La Figura 3.12 ilustra los tiempos de ejecución de la Figura 3.11, respecto a las correspondientes rutinas o implementaciones secuenciales. Se puede observar que los tiempos de ejecución de las implementaciones paralelas de los métodos alternativos de bidiagonalización ejecutadas en un solo procesador son mejores que los tiempos de ejecución obtenidos por las correspondientes implementaciones secuenciales. Eso es igualmente verdadero para la rutina de SCALAPACK respecto a `DGEBRD`, como lo había evidenciado los resultados de la Tabla 3.8. En estas condiciones, para valorar la aceleración de ejecución y la eficiencia, se han considerado los tiempos de ejecución de las implementaciones paralelas ejecutadas en un solo procesador como los tiempos del algoritmo secuencial y, por lo tanto, estos indicadores de prestaciones son valorados según las fórmulas (1.8) y (1.11).

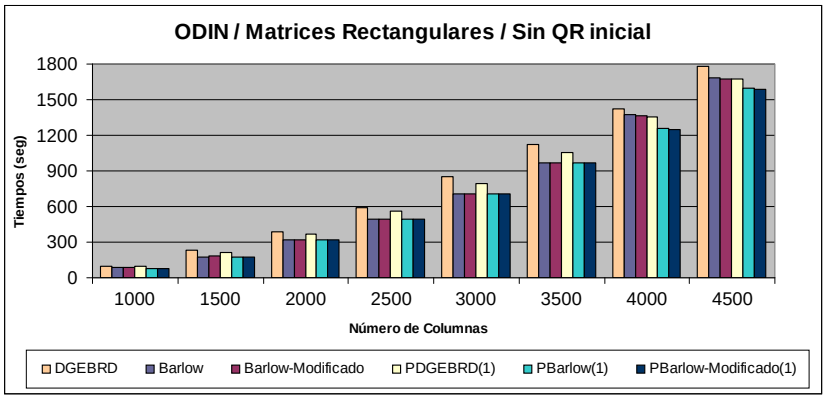

Figura 3.12: Tiempos de ejecución secuenciales *versus* tiempos de ejecución con un solo procesador.

La Figura 3.13 ilustra los tiempos de ejecución obtenidos por la rutina `PDGEBRD` respecto a la implementación paralela del método de Barlow con 2, 4, 6 y 8 procesadores empleando matrices cuadradas. La implementación paralela del método de Barlow presenta un "buen" comportamiento en el sentido de que la aportación de más procesadores reduce de manera más o menos proporcional el tiempo de ejecución y de manera muy similar a la rutina de SCALAPACK. El problema es que los tiempos de ejecución obtenidos están por encima de los tiempos de ejecución de la rutina `PDGEBRD` y hay casos donde el incremento del tiempo de ejecución es un poco significativo.

Conclusiones semejantes se pueden presentar para las implementaciones paralelas de los otros dos métodos alternativos de bidiagonalización, aunque la implementación paralela del método de Barlow modificado sea la que obtiene siempre mejores tiempos de ejecución.

Sin embargo, la Figura 3.14 ilustra los tiempos de ejecución obtenidos empleando también matrices cuadradas, pero ejecutando las implementaciones en mallas de procesadores del tipo $p \times 1$. En éste tipo de configuración de la malla de procesadores, por ejemplo, si la matriz tiene un tamaño de 9×9 y los

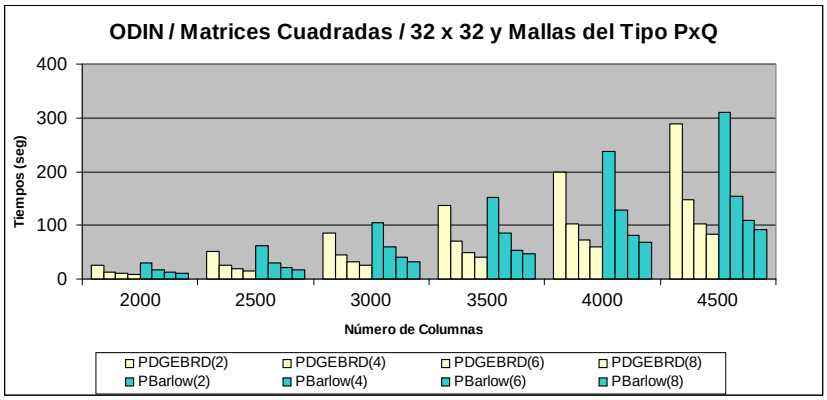

Figura 3.13: Tiempos de ejecución de `PDGEBRD` y del método de Barlow con matrices cuadradas en mallas del tipo $p \times q$.

bloques matriciales para la distribución cíclica de los datos son de tamaño 2×2, al emplear 2 procesadores organizados en una malla del tipo 2×1, se obtiene una distribución unidimensional cíclica de bloques de filas contiguas, resultando en la siguiente partición de la matriz.

$$\left[\begin{array}{ccccccccc}
a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & a_{19} \\
a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} & a_{29} \\
a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} & a_{59} \\
a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & a_{69} \\
a_{91} & a_{92} & a_{93} & a_{94} & a_{95} & a_{96} & a_{97} & a_{98} & a_{99} \\
\hline
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} & a_{39} \\
a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} & a_{49} \\
a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} \\
a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} & a_{87} & a_{88} & a_{89}
\end{array}\right]$$

Esta no es la distribución de datos propuesta por Ralha en [Ralha, 1994], que consiste en una distribución de bloques de filas contiguas por los distintos procesadores. Sin embargo, la distribución de datos obtenida en SCALAPACK,

empleando mallas de procesadores con la configuración $p \times 1$, permite que cada procesador tenga filas completas de la matriz, hecho que favorece a las implementaciones paralelas de los métodos alternativos de bidiagonalización. Como se observa en la Figura 3.14, las implementaciones paralelas de los métodos alternativos de bidiagonalización consiguen obtener mejores tiempos de ejecución, mientras que la rutina `PDGEBRD` obtiene peores tiempos de ejecución, cuando se compara con los tiempos de ejecución ilustrados en la Figura 3.13.

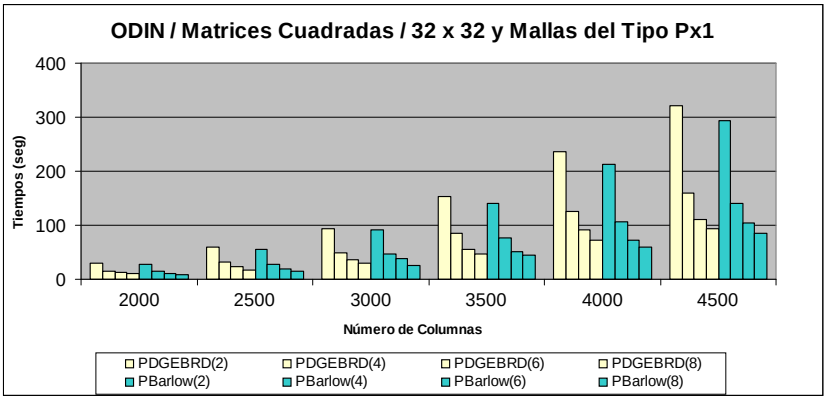

Figura 3.14: Tiempos de ejecución de `PDGEBRD` y del método de Barlow con matrices cuadradas en mallas del tipo $p \times 1$.

En los experimentos computacionales desarrollados con matrices rectangulares, los tiempos de ejecución obtenidos permiten presentar las mismas conclusiones, es decir, las implementaciones paralelas de los métodos alternativos de bidiagonalización presentan mejores tiempos de ejecución en las mallas del tipo $p \times 1$, que en las mallas de procesadores con las configuraciones descritas en la Tabla 3.7, mientras que la rutina `PDGEBRD` presenta un comportamiento contrario. Sin embargo, los mejores tiempos de ejecución son siempre obtenidos con la implementación paralela del método de Barlow modificado. La Figura 3.15 ilustra los tiempos de ejecución obtenidos por la

implementación paralela del método de Barlow modificado con matrices rectangulares, empleando las mallas de procesadores de la Tabla 3.7, respecto a los tiempos de ejecución obtenidos en las mallas del tipo $p \times 1$.

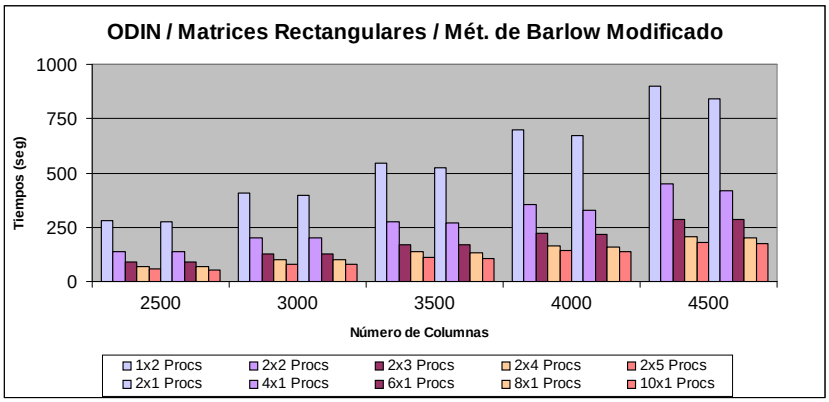

Figura 3.15: Tiempos de ejecución del método de Barlow modificado con matrices rectangulares.

La Figura 3.16 ilustra los mejores tiempos de ejecución obtenidos con la rutina `PDGEBRD`, respecto a los mejores tiempos de ejecución obtenidos con la implementación paralela del método de Barlow modificado, empleando matrices rectangulares. En la gran mayoría de los casos, los tiempos de ejecución obtenidos con la implementación paralela del método de Barlow modificado son muy semejantes a los de la rutina `PDGEBRD` o incluso mejores. Además, esta diferencia entre los tiempos de ejecución es más evidente en matrices rectangulares que en matrices cuadradas. En estas condiciones, se puede asumir que la implementación paralela del método de Barlow modificado y ejecutada en mallas de procesadores del tipo $p \times 1$, obtiene tiempos de ejecución que son comparables a los tiempos de ejecución de la rutina de SCALAPACK y además, competitivas. Sin embargo, hay que tener en cuenta que los métodos alternativos de bidiagonalización fueron desarrollados con el objetivo de reducir el número de comunicaciones necesarias en sistemas computacionales de

memoria distribuida. Sin embargo, los sistemas computacionales utilizados en el marco de la tesis tienen sistemas de comunicaciones muy eficientes y, por tanto, el tiempo total empleado en comunicaciones es poco significativo respecto al tiempo total de ejecución. Además, se puede prever que en un sistema computacional de memoria distribuida que disponga de un mecanismo de comunicaciones menos eficiente, todas las implementaciones paralelas de los métodos alternativos de bidiagonalización obtendrán tiempos de ejecución bastante mejores que la rutina `PDGEBRD` de SCALAPACK.

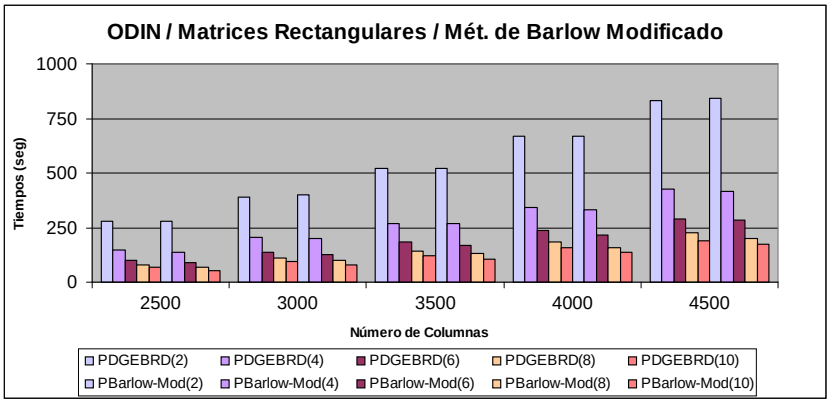

Figura 3.16: Tiempos de ejecución de `PDGEBRD` y del método de Barlow modificado con matrices rectangulares.

A continuación se presenta el estudio comparativo entre la rutina `PDGEBRD` y la implementación paralela del método de Barlow modificado, respecto a la aceleración de ejecución y a la eficiencia.

La Figura 3.17 ilustra los valores del *speed-up* de la implementación paralela del método de Barlow modificado con matrices rectangulares y empleando 2, 4, 6, 8 y 10 procesadores. Como se observa, los valores ilustrados son cercanos a los valores ideales. Además, esta implementación presenta también elevados valores de eficiencia, ilustrados en la Figura 3.18.

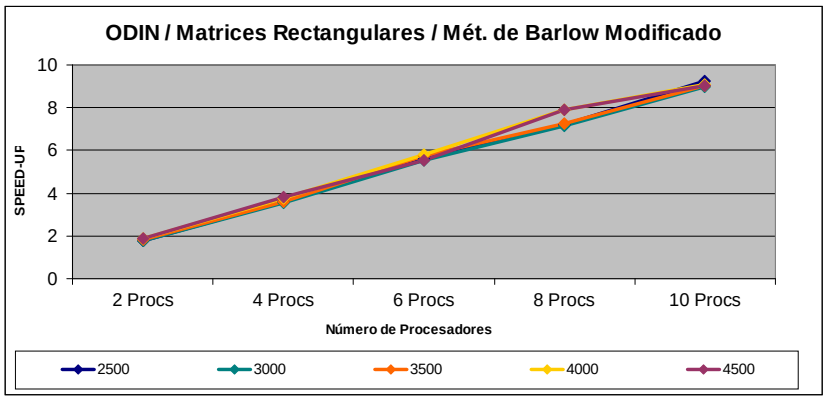

Figura 3.17: *Speed-up* del método de Barlow modificado con matrices rectangulares.

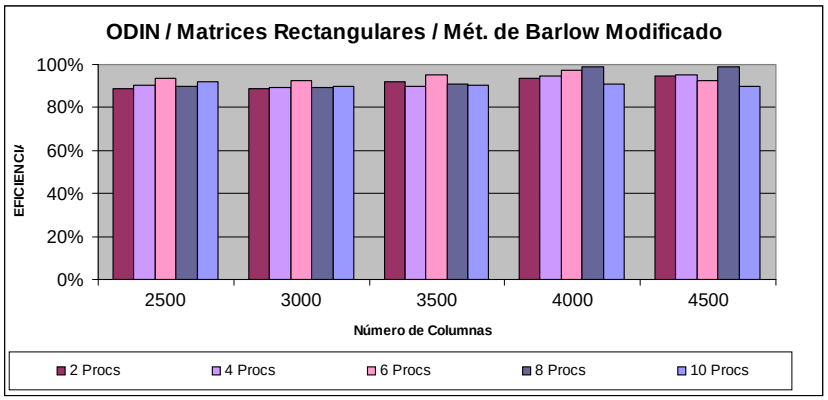

Figura 3.18: Eficiencia del método de Barlow modificado con matrices rectangulares.

La Tabla 3.9 ilustra los valores del *speed-up* de la rutina `PDGEBRD` y de la implementación paralela del método de Barlow modificado con matrices rectangulares. Como se observa, la implementación paralela del método de Barlow modificado obtiene valores de aceleración de ejecución semejantes a los de la rutina `PDGEBRD` y hay casos en los que incluso los valores son mejores y, por lo tanto, eso permite obtener valores de eficiencia mejores que los obtenidos con la rutina de SCALAPACK.

Implementación	**2500**	**3000**	**3500**	**4000**	**4500**
`PDGEBRD`(2)	1.8	1.9	1.9	1.9	1.9
`PDGEBRD`(4)	3.9	3.9	3.9	3.9	3.9
`PDGEBRD`(6)	5.7	5.7	5.8	5.8	5.8
`PDGEBRD`(8)	7.1	7.3	7.3	7.4	7.5
`PDGEBRD`(10)	8.1	8.4	8.6	8.6	8.8
`PBarlow-Modificado`(2)	1.8	1.8	1.8	1.9	1.9
`PBarlow-Modificado`(4)	3.6	3.6	3.6	3.8	3.8
`PBarlow-Modificado`(6)	5.6	5.6	5.7	5.8	5.8
`PBarlow-Modificado`(8)	7.2	7.3	7.3	7.6	7.8
`PBarlow-Modificado`(10)	9.2	9.1	9.1	9.1	9.0

Tabla 3.9: *Speed-up* de `PDGEBRD` y del método de Barlow modificado con matrices rectangulares.

La Figura 3.19 contrasta los valores de la eficiencia de la rutina `PDGEBRD` con los valores expuestos en la Figura 3.18, obtenidos con la implementación paralela del método de Barlow modificado. Como se puede observar, hay casos en que la implementación paralela del método de Barlow modificado es más eficiente que la rutina `PDGEBRD` y al contrario. Por tanto, se puede concluir que la implementación paralela del método de Barlow modificado es tan eficiente como la rutina `PDGEBRD`. Hay que tener en cuenta que en la Figura 3.19 son ilustrados los valores de la eficiencia obtenidos con los mejores tiempos de ejecución de la rutina `PDGEBRD`, respecto a los mejores tiempos de ejecución obtenidos con la implementación paralela del método de Barlow modificado. Sin embargo, si la comparativa es desarrollada con los tiempos de ejecución obtenidos en las mallas del tipo $p \times 1$, como la rutina `PDGEBRD` obtiene peores tiempos de ejecución, su

eficiencia es menor y, por lo tanto, la implementación paralela del método de Barlow modificado es más eficiente que la rutina `PDGEBRD`.

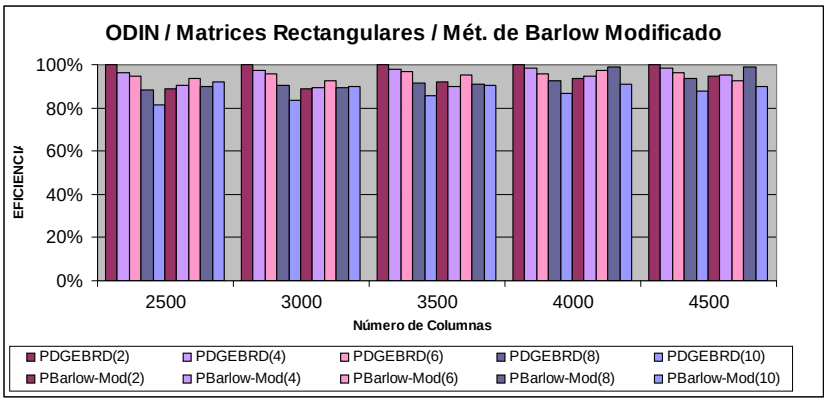

Figura 3.19: Eficiencia de `PDGEBRD` y del método de Barlow modificado con matrices rectangulares.

Para las matrices cuadradas, los resultados obtenidos permiten sacar las mismas conclusiones que para las matrices rectangulares, aunque los tiempos de ejecución obtenidos por la implementación paralela del método de Barlow modificado sean peores.

A continuación, la Figura 3.20 ilustra los valores del *speed-up* de la implementación paralela del método de Barlow modificado que contrastan con los valores del *speed-up* de la rutina `PDGEBRD`, obtenidos con los tiempos de ejecución descritos en la Tabla 3.8, empleando las mismas configuraciones de la malla de procesadores de la Tabla 3.8.

Como se observa, aunque se hayan empleado las configuraciones donde la rutina `PDGEBRD` obtiene mejores tiempos de ejecución, la implementación paralela del método de Barlow modificado presenta valores del *speed-up* que son mejores a los de la rutina `PDGEBRD` y, por lo tanto, esta implementación es más eficiente. Sin embargo, esos valores del *speed-up* pueden ser mejorados considerando los tiempos de ejecución obtenidos por la implementación paralela del método de

Barlow modificado ejecutada en las mallas de procesadores del tipo $p \times 1$.

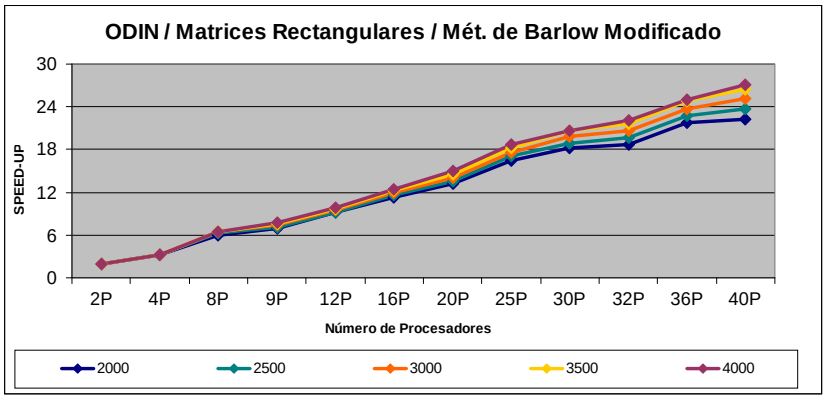

Figura 3.20: *Speed-up* del método de Barlow modificado con matrices rectangulares.

Para terminar esta sección y para promover la utilización de los métodos alternativos de bidiagonalización en la reducción de matrices reales densas a la forma bidiagonal superior, cabe destacar que si no se emplean las rutinas de SCALAPACK y de BLACS en la resolución paralela del problema, entonces la mejor metodología sería la de emplear la distribución de datos descrita por Ralha en [Ralha, 1994], usando rutinas de LAPACK o de BLAS en las computaciones y rutinas del MPI (o PVM) en las comunicaciones.

En este caso, las implementaciones paralelas son esencialmente la implementación secuencial, en la cual se usan comunicaciones para calcular el vector con los productos internos. En esta metodología hay un buen balanceo de la carga, aunque haya una pequeña redundancia en las computaciones, con el objetivo de que todos los procesadores tengan que calcular el mismo vector de Householder. Sin embargo, el impacto de este sobrecoste no es dramático en el coste total de los tiempos de ejecución, siempre que la razón m/p sea grande, donde m es el número de filas de la matriz y p es el número de procesadores empleados.

En el marco de la tesis se han desarrollado implementaciones paralelas de los métodos alternativos de bidiagonalización empleando las rutinas de LAPACK o de BLAS para las computaciones y las rutinas del MPI para las comunicaciones. Estas son las implementaciones paralelas de los métodos alternativos de bidiagonalización que obtienen los mejores tiempos de ejecución, cuando se comparan con los tiempos de ejecución obtenidos con las correspondientes implementaciones paralelas ejecutadas en el entorno de SCALAPACK. Además, hay casos en que la reducción del tiempo de ejecución llega a 20 %.

3.7. Conclusiones

En este capítulo 3 se ha empezado por describir los métodos propuestos por Ralha y por Barlow para la reducción de matrices reales densas a la forma bidiagonal superior, donde se aplican transformaciones de Householder por el lado derecho de la matriz. Esto permite definir todos los cálculos computacionales en términos de las columnas de la matriz a transformar, facilitando así el desarrollo de implementaciones paralelas que resultan más sencillas que con los métodos tradicionales. Al mismo tiempo, se reduce significativamente el número de comunicaciones necesarias.

Como aportación novedosa de la tesis se han introducido modificaciones en el método de Barlow con el objetivo de reducir el número de comunicaciones necesarias en la implementación paralela y en ese sentido, fue descrito el método de Barlow modificado.

A continuación fueron presentadas algunas comparativas de los costes computacionales y las metodologías empleadas en el desarrollo de las implementaciones secuenciales y paralelas de los métodos alternativos de bidiagonalización, cumpliendo con uno de los objetivos de la tesis. Además, para alcanzar altas prestaciones y un elevado grado de portabilidad, se ha procurado sacar el máximo provecho de las altas prestaciones de las rutinas de LAPACK, de BLAS, de SCALAPACK y de PBLAS.

Posteriormente se ha desarrollado un elevado conjunto de experimentos prácticos con el objetivo de hacer un completo estudio comparativo entre los tiempos de ejecución obtenidos con las implementaciones desarrolladas en el marco de la tesis y las correspondientes rutinas de LAPACK y de SCALAPACK con la misma funcionalidad.

Basados en todo el trabajo desarrollado en este capítulo, se pueden presentar las siguientes conclusiones:

- Las implementaciones secuenciales y paralelas de los métodos alternativos de bidiagonalización obtienen tiempos de ejecución semejantes, aunque los mejores tiempos de ejecución sean los del método de Barlow modificado.
- Los métodos alternativos de bidiagonalización tienen el mismo orden de complejidad computacional y además, su mayor ventaja está en el cálculo de la DVS completa de una matriz real densa, dado que presentan un coste computacional mucho menor que los métodos tradicionales, debido al hecho de que siempre se hace la acumulación de las transformaciones de Householder empleadas.
- En los problemas numéricos donde solo se necesita calcular los valores singulares de una matriz real densa, se ha comprobado que las implementaciones paralelas de los métodos alternativos de bidiagonalización presentan altas prestaciones y elevadas eficiencias, con valores que son comparables y competitivos con los valores obtenidos con la rutina `PDGEBRD` de SCALAPACK.
- Se ha comprobado que las configuraciones de la malla de procesadores óptimas para la rutina `PDGEBRD` son las que están más cercanas a la configuración cuadrada, mientras que para los métodos alternativos de bidiagonalización son las mallas de procesadores del tipo $p \times 1$, en el entorno de SCALAPACK, aunque en este tipo de malla de procesadores no se obtenga la distribución de datos descrita por Ralha en [Ralha, 1994].
- Para obtener implementaciones paralelas de los métodos alternativos de bidiagonalización con la distribución de datos descrita en [Ralha, 1994], la mejor metodología es la de emplear rutinas de LAPACK y de BLAS para las computaciones y rutinas de MPI o PVM para las comunicaciones. En este tipo de implementación, la implementación paralela es esencialmente la implementación secuencial, en la cual se usan comunicaciones para calcular el vector con los productos internos, originando una pequeña redundancia en las computaciones, debido al hecho de que todos los procesadores tienen que calcular el mismo vector de Householder. Sin

embargo, el impacto de este sobrecoste no es dramático en el coste total de los tiempos de ejecución, siempre que la razón m/p sea grande, donde m es el número de filas de la matriz y p es el número de procesadores empleados.

- A pesar de que los métodos alternativos de bidiagonalización reducen significativamente el número de comunicaciones, esta diferencia no se manifiesta notablemente en los tiempos de ejecución ilustrados. Cabe señalar que la red de interconexión usada en los sistemas computacionales empleados en el marco de la tesis era muy rápida. Por tanto, cabe esperar que en sistemas computacionales con redes menos rápidas o en matrices de mayor dimensión, la ventaja de los métodos propuestos sea más notable y así, la diferencia entre los tiempos de ejecución obtenidos con los métodos alternativos de bidiagonalización y los tiempos de ejecución obtenidos con `PDGEBRD` sea más significativa.

- Fue posible comprobar que los tiempos de ejecución obtenidos por las implementaciones del método de Barlow modificado (desarrollado en el marco de la tesis), son mejores que los tiempos de ejecución obtenidos por las implementaciones del método de Ralha y del método de Barlow.

- La gran desventaja de los métodos alternativos de bidiagonalización está en el hecho de que no permitiesen el desarrollo de implementaciones secuenciales y paralelas orientadas a bloques matriciales y, por lo tanto, no permitiesen obtener las ventajas de la utilización de las rutinas de BLAS de nivel 3. En su tesis doctoral y en sus publicaciones, Bosner describe una implementación secuencial y paralela del método de Barlow en las que emplea una metodología "orientada a bloques" con rutinas de BLAS de nivel 2.5. Como resultado de sus experimentos prácticos, defiende que dependiendo de la dimensión de la matriz y del tamaño del bloque matricial empleado, las prestaciones de su implementación secuencial son mejores que las de la rutina de LAPACK, pero que su implementación paralela es mucho peor que la rutina de SCALAPACK. Como consecuencia, Bosner aboga que para obtener una implementación paralela de los métodos alternativos de bidiagonalización que consiga altas prestaciones en sistemas de memoria distribuida, la mejor metodología es trasladar las llamadas de las rutinas de LAPACK y de BLAS de la

implementación secuencial a las correspondientes rutinas paralelas de SCALAPACK y de PBLAS, tal como hemos hecho en esta tesis.

Como línea de trabajo futuro se plantea la posibilidad de adaptar y de emplear el método de Barlow modificado en la reducción de matrices reales densas a la forma de matriz banda, siguiendo la misma línea de investigación desarrollada por Lang y sus colaboradores [Lang, 1997], [Großer y Lang, 1999], [Bischof *et al.*, 2000], [Großer y Lang, 2003], [Großer y Lang, 2005].

Capítulo 4

La Descomposición en Valores Propios

ESTE capítulo está dirigido al cálculo de la **descomposición en valores propios** (DVP) de matrices reales y en particular, de matrices tridiagonales simétricas.

El capítulo empieza con una breve descripción del estado del arte y después se describen los conceptos del problema estándar del cálculo de la DVP de una matriz tridiagonal simétrica, así como los métodos numéricos para el cálculo de los valores y de los vectores propios de la matriz. En concreto, se exponen los fundamentos del **método de bisección**, empleado en el cálculo de los valores propios de la matriz tridiagonal simétrica y del **método MRRR** (*Multiple Relatively Robust Representations* o **Múltiples Representaciones Relativamente Robustas**), empleado en el cálculo de sus respectivos vectores propios.

Posteriormente se describe el **método** `zeroinNR` propuesto por Ralha en [Ralha, 1990], donde el autor presenta una variante del **método** `zeroin` [Philippe *et al.*, 1987], en la cual introduce una nueva formulación al cálculo de

la corrección del método de Newton-Raphson con el objetivo de garantizar que los valores propios de la matriz sean calculados con elevada precisión numérica.

A continuación se hace un análisis de las prestaciones obtenidas por algunas de las rutinas de LAPACK dedicadas a la DVP y seguidamente son descritos algunos resultados obtenidos por la implementación secuencial del método `zeroinNR`, desarrollada en el marco de la tesis, exponiendo el estudio entre los tiempos de ejecución obtenidos por esa implementación secuencial y los tiempos de ejecución obtenidos por la rutina `DSTEBZ` de LAPACK.

Al terminar el capítulo se describen las conclusiones más importantes que la investigación desarrollada ha permitido lograr y que son resultado de los objetivos planteados para la tesis:

- Calcular la DVS de una matriz bidiagonal superior, convirtiendo este problema en el problema de calcular la DVP de una matriz tridiagonal simétrica.
- Desarrollar implementaciones secuenciales de altas prestaciones para calcular los valores propios de una matriz tridiagonal simétrica.
- Emplear el método MRRR para calcular los respectivos vectores propios de la matriz tridiagonal simétrica.

4.1. Breve estado del arte

Dada su elevada importancia, tanto a nivel teórico, como a nivel práctico, y en particular, en la computación numérica, el problema algebraico del cálculo de los valores propios, y en ocasiones, el cálculo de los respectivos vectores propios, de matrices reales o complejas, simétricas o no, ha sido un asunto estudiado por muchos y distintos autores, aportando una lista de referencias bibliográficas a este tema muy diversa y bastante extensa.

Para esta sección no es nuestro objetivo hacer un listado de toda la bibliografía especializada en el cálculo de la DVP de matrices, sin embargo, vamos a empezar la sección por hacer constancia de los libros: *The Algebraic Eigenvalue Problem* [Wilkinson, 1965], *Matrix Computations* [Golub y van Loan, 1996], *Applied Numerical Linear Algebra* [Demmel, 1997], *The Symmetric Eigenvalue Problem* [Parlett, 1998], *Numerical Linear Algebra*

for High-Performance Computers [Dongarra *et al.*, 1998] y *Templates for the Solution of Algebraic Eigenvalue Problems: a Pratical Guide* [Bai *et al.*, 2000], como documentos recomendables para su estudio y como una excelente introducción al problema del cálculo de la DVP de matrices.

En general, los métodos numéricos para el cálculo de la DVP de matrices se pueden dividir en dos grupos: los **métodos iterativos** y los **métodos de transformación**.

Los **métodos iterativos** son métodos numéricos que habitualmente calculan aproximaciones para un subconjunto de valores y vectores propios de una matriz, empleando una secuencia de iteraciones que parten de una aproximación inicial. Estos métodos suelen ser empleados en matrices dispersas y a esta clase pertenecen métodos como el **método de la potencia**, el **método de la iteración inversa**, el **método del cociente de Rayleigh**, el **método de Lanczos** y el **método de Jacobi-Davidson** [Saad, 2011].

Los **métodos de transformación** son aquellos que trabajan con transformaciones de semejanza y que habitualmente calculan todos los valores propios, y en ocasiones, los respectivos vectores propios. Además, estos métodos suelen ser empleados en matrices densas.

Dado que más adelante se estudian ciertos métodos de transformación, pasamos a describir esta clase y nos centramos en el caso simétrico. En primer lugar suelen transformar la matriz inicial densa en una matriz tridiagonal simétrica, empleando transformaciones de semejanza y a continuación calculan la DVP de la matriz tridiagonal simétrica y, con ella, obtienen la DVP de la matriz inicial. Por ejemplo, el **método de Jacobi** [Wilkinson, 1965], [Golub y van Loan, 1996], no reduce la matriz inicial a la forma tridiagonal simétrica, sino que el método trabaja directamente sobre la matriz inicial, empleando una secuencia de rotaciones ortogonales que llevan la matriz a la forma diagonal, donde los elementos de la diagonal son los valores propios de la matriz inicial.

Para calcular la DVP de una matriz tridiagonal simétrica, podemos distinguir, básicamente, tres clases de métodos: el **método iterativo QR** en sus distintas variantes, los **métodos del tipo divide-y-vencerás** y los **métodos de bisección/multisección**.

El **método iterativo QR**, desarrollado de forma independiente por Francis [Francis, 1961], [Francis, 1962] y por Kublanovskaya [Kublanovskaya, 1962],

fue considerado durante muchos años como el método numérico más rápido para el cálculo secuencial de todos los valores propios de una matriz tridiagonal [Golub y van Loan, 1996], [Demmel, 1997]. En LAPACK, su implementación está desarrollada en la rutina `xSTERF`, la cual calcula todos los valores propios de una matriz tridiagonal simétrica empleando la variante Pal-Walker-Kahan del método QR [Parlett, 1998], y en la rutina `xSTEQR`, que opcionalmente permite calcular también los vectores propios de la matriz. Actualmente el mejor método numérico para el cálculo de los valores propios de matrices tridiagonales simétricas definidas positivas es el **método dqds** (*differential quotient diference with shifts* o **cociente de diferencias diferenciales con desplazamientos de origen**), implementado en LAPACK en la rutina `xSTEMR` [Parlett y Marques, 2002]. Esta rutina permite, opcionalmente, calcular los vectores propios de la matriz.

El **método divide-y-vencerás** fue presentado por primera vez en 1981 por Cuppen [Cuppen, 1981], pero fueron necesarios casi 15 años para que una implementación secuencial estable del método fuese desarrollada [Rutter, 1994], [Gu y Eisenstat, 1995b]. Este método numérico es mucho más rápido que los demás y, de hecho, en [Demmel, 1997, pp. 211], el autor aboga que este método es el más rápido de todos a la hora de calcular todos los valores y todos los vectores propios de una matriz tridiagonal simétrica cuya dimensión es superior a 25. Como describe Willems en su tesis doctoral [Willems, 2010], aunque la complejidad teórica del método sea $\mathcal{O}\left(n^3\right)$, en realidad, debido al número de operaciones que pueden ser desarrolladas empleando operaciones de BLAS de nivel 3, se puede conseguir una complejidad práctica del tipo $\mathcal{O}\left(n^{2,5}\right)$. En LAPACK, el método está implementado en la rutina `xSTEDC`, que también opcionalmente permite calcular los vectores propios de la matriz. Una implementación paralela del método es descrita, por ejemplo, en [Tisseur y Dongarra, 1998].

El **método de bisección** fue desarrollado inicialmente por Givens [Givens, 1953], [Givens, 1954], y está basado en la aplicación de una técnica de bisección en el cálculo de la **secuencia de Sturm** (la cual definiremos en la próxima sección).

En LAPACK, el método está implementado en la rutina `xSTEBZ` y además de permitir calcular todos los valores propios de la matriz tridiagonal simétrica, permite también calcular solamente los valores propios que están contenidos en

un intervalo de números reales predefinido o que están definidos por un subconjunto de índices. En [Kahan, 1966] es descrita una de las primeras implementaciones secuenciales del método de bisección.

De la misma manera que en otras implementaciones, en la rutina `xSTEBZ`, el método de bisección está implementado empleando dos etapas distintas: la **etapa de aislamiento**, en la cual se calculan los intervalos de números reales disjuntos y que contienen un solo valor propio de la matriz, y la **etapa de extracción**, en la cual se calculan cada uno de los valores propios que están contenidos en los intervalos anteriores, con la precisión numérica deseada.

Cabe señalar que la descripción del método de bisección como un método numérico que emplea la etapa de aislamiento y a continuación emplea la etapa de extracción, fue por primera vez presentada en 1987 en [Lo *et al.*, 1987]. Sin embargo, anteriormente, en [Wilkinson, 1965], el autor había expuesto que el método es muy bueno en la etapa de aislamiento pero que aun puede ser acelerado más cuando se combina con una técnica de aproximación de raíces de polinomios, como es, por ejemplo, el método de Newton-Raphson o el método de Laguerre.

En [Parlett, 1964], en [Bernstein, 1984] y en [Li y Zeng, 1992a], se pueden encontrar, por ejemplo, algunas maneras de acelerar el método de bisección y además, en [Barth *et al.*, 1967] se describe una implementación secuencial propia.

La rutina `xSTEBZ` sólo calcula los valores propios de la matriz tridiagonal simétrica. En cambio, la rutina `xSTEIN`, por ejemplo, sólo calcula los respectivos vectores propios, implementando el método de la iteración inversa con un máximo de cinco iteraciones. Una descripción de la implementación del método de la iteración inversa se puede encontrar, por ejemplo, en [Jessup y Ipsen, 1992].

En [Demmel *et al.*, 1988] y en [Li y Zeng, 1992a], se describen estudios comparativos entre los métodos de transformación anteriormente nombrados, teniendo en cuenta sus implementaciones secuenciales. Sin embargo, en [Lo *et al.*, 1987], en [Simon, 1989] y en [Ipsen y Jessup, 1990], se describen estudios comparativos entre sus correspondientes implementaciones paralelas. Además, por ejemplo, en [Barlow y Evans, 1977], en [Dongarra y Sorensen, 1987], en [Baserman y Weidner, 1992], en [Ralha, 1993], en [Badía y Vidal, 1998] y en [Forjaz y Ralha, 2001], se pueden encontrar descripciones de la implementación paralela del método de bisección.

Aunque no sea de nuestro interés, hay que señalar que existe un conjunto

de autores que se han dedicado a estudiar la aplicación de un **método de homotopía** en el cálculo de la DVP de matrices. Son ejemplo de ello las siguientes referencias de la bibliografía: [Li y Rhee, 1989], [Li *et al.*, 1991], [Zeng, 1991], [Li y Zeng, 1992b], [Li *et al.*, 1992], [Oettli, 1995], [Lui *et al.*, 1997] y [Brockman *et al.*, 2013].

Hay igualmente otro conjunto de autores que han dedicado sus esfuerzos al estudio de la precisión numérica con la cual son calculados los valores y los valores singulares de las matrices, presentando algunos resultados sobre cotas del error, en especial, del error relativo. Algunos de esos resultados son, por ejemplo, las siguientes referencias: [Barlow y Demmel, 1990], [Demmel y Kahan, 1990], [Demmel y Veselić, 1992], [Demmel y Gragg, 1993], [Eisenstat y Ipsen, 1995], [Demmel, 1999a] [Demmel *et al.*, 1999b], [Eisenstat y Ipsen, 1998], [Ipsen, 1998], [Koev, 2005], [Dopico y Koev, 2006] y [Ralha, 2009].

Para terminar la sección deseamos hacer constar que las líneas de investigación más recientes están relacionadas con la aplicación de sistemas computacionales con **procesadores multinúcleo** (*multi-core processors*) o sistemas computacionales con **GPU**s (*Graphics Processing Unit* o **Unidad de Procesamiento Gráfico**). En los *working notes* de LAPACK, por ejemplo, se pueden encontrar algunos documentos que están en estas líneas de investigación y en particular, en el cálculo de las habituales descomposiciones matriciales, como lo son la DVS y la DVP [Volkov y Demmel, 2008a], [Volkov y Demmel, 2008b], [Ballard *et al.*, 2010], [Gates *et al.*, 2014].

4.2. Conceptos del problema estándar

Dada una matriz $A \in \mathbb{C}^{n\times n}$, se denominan **valores propios** de A a las n raíces del **polinomio característico** $p(\lambda) = \det(A - \lambda I)$, donde I representa la matriz identidad de orden n. Al conjunto de todos los valores propios de A se denomina **espectro** de la matriz y se suele denotar por $\sigma(A)$.

Si $\lambda \in \sigma(A)$, entonces todo vector no nulo x que verifique $Ax = \lambda x$ se denomina **vector propio de A por la derecha**, asociado al valor propio λ. En cambio, todo vector no nulo y que verifique $y^*A = \lambda y^*$, donde $y^* = \overline{y^t} = \overline{y}^t$, se denomina **vector propio de A por la izquierda**, asociado al valor propio λ. Cuando se afirma solamente "vector propio" se está hablando del vector propio por la derecha.

Aunque las definiciones de valor propio y de vector propio sean simples, cuando se pasa al entorno de la computación numérica y se intenta emplear un método numérico en su cálculo, la cuestión es más difícil. Habitualmente, lo que hacen la mayoría de los métodos numéricos dedicados al cálculo de los valores propios, y en ocasiones, de los respectivos vectores propios, es transformar la matriz inicial en una matriz más simple, a la cual se suele denominar **forma canónica** y para la cual es más fácil calcular sus valores y vectores propios.

Las transformaciones empleadas que llevan la matriz inicial a una forma canónica son **transformaciones de semejanza**. Así, dadas las matrices A y B ($A, B \in \mathbb{C}^{n\times n}$), si existe una matriz $S \in \mathbb{C}^{n\times n}$, no singular y tal que $B = S^{-1}AS$, entonces se dice que las matrices A y B son **semejantes** y que S es una **transformación de semejanza**.

Las formas canónicas más conocidas son la **forma canónica de Jordan** y la **forma canónica de Schur**, que se exponen a continuación.

Teorema 3 (Forma canónica de Jordan) *Dada una matriz $A \in \mathbb{C}^{n\times n}$, existe una matriz no singular $S \in \mathbb{C}^{n\times n}$, tal que $S^{-1}AS = J$, donde $J \in \mathbb{C}^{n\times n}$ es la llamada **forma canónica de Jordan**, definida por*

$$J = diag\left[J_{n_1}(\lambda_1), J_{n_2}(\lambda_2), \ldots, J_{n_k}(\lambda_k)\right] \tag{4.1}$$

con $n_1 + n_2 + \ldots + n_k = n$ y donde cada $J_{n_i}(\lambda_i)$ es un bloque matricial de dimensión $n_i \times n_i$, definido por

$$J_{n_i}(\lambda_i) = \begin{bmatrix} \lambda_i & 1 & & 0 \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_i \end{bmatrix}. \tag{4.2}$$

DEMOSTRACIÓN: La demostración del teorema está en [Halmos, 1958] y en [Horn y Johnson, 1990]. ■

En el teorema anterior, salvo para el orden de los bloques matriciales de la diagonal, la matriz J es única.

Teorema 4 (Forma canónica de Schur) *Dada una matriz $A \in \mathbb{C}^{n\times n}$, existe una matriz unitaria $Q \in \mathbb{C}^{n\times n}$ y una matriz triangular superior $T \in \mathbb{C}^{n\times n}$, tal que $Q^*AQ = T$, donde T es la llamada* ***forma canónica de Schur****.*

DEMOSTRACIÓN: La demostración del teorema está en la sección 7.1 de [Golub y van Loan, 1996] y en la sección 4.2 de [Demmel, 1997]. ■

Del teorema anterior se puede deducir que los valores propios de la matriz A son los elementos de la diagonal principal de la matriz T y además, la forma canónica de Schur no es única, ya que los valores propios de la matriz A se pueden distribuir de distintas maneras a lo largo de la diagonal principal de la matriz T.

Una vez conocida la forma canónica de Schur de la matriz A ($Q^*AQ = T$), y conocidos los vectores propios de la matriz T ($Tx = \lambda x$), se tiene $AQx = QTx = \lambda Qx$ y, por tanto, los vectores propios de la matriz A son los vectores Qx. Así, de esta manera sencilla, para calcular los vectores propios de la matriz inicial basta conocer los vectores propios de su forma canónica de Schur. En la sección 4.2.1 de [Demmel, 1997], por ejemplo, es descrito cómo se calculan los vectores propios de la matriz T.

En el marco de la tesis siempre que se nombra la DVP de matrices se está pensando en la aplicación del teorema que se expone a continuación y que define la **forma canónica real de Schur**.

Teorema 5 (Forma canónica real de Schur) *Dada una matriz $A \in \mathbb{R}^{n\times n}$, existe una matriz ortogonal $Q \in \mathbb{R}^{n\times n}$ y una matriz cuasi-triangular superior $T \in \mathbb{R}^{n\times n}$, tal que*

$$Q^tAQ = T = \begin{bmatrix} R_{11} & R_{12} & \dots & R_{1k} \\ & R_{22} & \dots & R_{2k} \\ & & \ddots & \vdots \\ & & & R_{kk} \end{bmatrix} \tag{4.3}$$

donde T es la llamada ***forma canónica real de Schur*** *y donde cada bloque matricial R_{ii} $(i = 1, \dots, k)$, o es un bloque matricial de dimensión 1×1 en el caso de los valores propios reales, o es un bloque matricial de dimensión 2×2 en el caso de los valores propios complejos conjugados.*

DEMOSTRACIÓN: La demostración del teorema está en la sección 7.4.1 de [Golub y van Loan, 1996] y en la sección 4.2 de [Demmel, 1997]. ■

En el teorema anterior, cuando la matriz A, además de ser real, es también una matriz simétrica, entonces la matriz T es una matriz diagonal con valores propios reales. En la sección 8.1.1 de [Golub y van Loan, 1996], por ejemplo, está el teorema y la correspondiente demostración de la **forma canónica real simétrica de Schur**, la cual se va a emplear en el marco de la tesis cuando nos centremos en el cálculo de los valores y de los vectores propios de matrices tridiagonales simétricas:

- *Si $A \in \mathbb{R}^{n\times n}$ es simétrica, entonces existe una matriz ortogonal $Q \in \mathbb{R}^{n\times n}$ tal que*

$$Q^t AQ = diag\left(\lambda_1, \ldots, \lambda_n\right) \tag{4.4}$$

y además, para $j = 1, \ldots, n$, se tiene

$$AQ\left(:, j\right) = \lambda_j Q\left(:, j\right). \tag{4.5}$$

Para terminar esta sección se presenta a continuación el **teorema de los círculos de Gershgorin**, el cual garantiza que en el cálculo de los valores propios de una matriz se puede restringir el cálculo a un subintervalo de la recta real.

Teorema 6 (Círculos de Gershgorin) *Dada una matriz $A \in \mathbb{C}^{n\times n}$, sus valores propios están dentro de la unión de los n* ***círculos de Gershgorin*** *definidos por*

$$|\lambda - a_{ii}| \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \qquad (i = 1, \ldots, n). \tag{4.6}$$

DEMOSTRACIÓN: La demostración del teorema está en la sección 2.7.3 de [Demmel, 1997]. ■

4.2.1. DVP de matrices reales simétricas

En esta sección se exponen algunos conceptos fundamentales para el cálculo de la DVP de matrices reales simétricas, y en particular, para matrices tridiagonales simétricas.

Definición 7 (Inercia) *Dada una matriz $A \in \mathbb{R}^{n\times n}$ simétrica, se llama **inercia** de A y vamos a denotarla por $Inercia\,(A)$, a la terna de números enteros no negativos definida por*

$$Inercia\,(A) = (\nu, \zeta, \pi) \tag{4.7}$$

donde ν es el número de valores propios negativos de A, ζ es el número de valores propios nulos de A y π es el número de valores propios positivos de A.

Si X es una matriz ortogonal, entonces las matrices X^tAX y A son semejantes y, por tanto, tienen los mismos valores propios. Si X sólo es una matriz no singular, entonces las matrices X^tAX y A son **congruentes** y, en este caso, en general, no tienen los mismos valores propios, pero, sin embargo, el teorema expuesto a continuación garantiza que, por lo menos, los valores propios tienen el mismo signo.

Teorema 8 (Ley de inercia de Sylvester) *Dos matrices $A \in \mathbb{R}^{n\times n}$ y $B \in \mathbb{R}^{n\times n}$, simétricas y congruentes, tienen la misma inercia.*

DEMOSTRACIÓN: La demostración del teorema está en la sección 8.1.5 de [Golub y van Loan, 1996] y en la sección 5.2 de [Demmel, 1997]. ■

Una vez que la ley de inercia de Sylvester garantiza que dos matrices simétricas y congruentes tienen la misma inercia, el objetivo ahora es aprovechar esta propiedad de las matrices para dividir el espectro de una matriz simétrica en distintos subconjuntos. Para ello, se va a emplear el teorema descrito a continuación, el cual garantiza que existe una relación directa entre los valores propios de una matriz y los de esta misma matriz desplazada.

Así, para calcular el número de valores propios de una matriz A que son más pequeños que un valor μ dado, se emplea un corolario del teorema de la ley de inercia de Sylvester, que garantiza la invarianza del número de valores propios negativos respecto a transformaciones congruentes.

Teorema 9 *Dada una matriz $A \in \mathbb{R}^{n\times n}$, supongamos que la matriz $A - \mu M$ admite la descomposición triangular*

$$A - \mu M = L_\mu D_\mu L_\mu^t \tag{4.8}$$

donde L_μ es una matriz triangular inferior no singular y D_μ es una matriz diagonal. Si el par (A, M) tiene un conjunto completo de valores propios reales, entonces

$$\nu(\Lambda - \mu I) = \nu(A - \mu M) = \nu(D_\mu) \tag{4.9}$$

donde $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$, siendo λ_i $(i = 1, \ldots, n)$ los valores propios del par (A, M).

DEMOSTRACIÓN: La demostración del teorema está en la sección 3.3 de [Parlett, 1998]. ■

El teorema anterior tiene especial importancia en el contexto del cálculo de los valores propios de matrices tridiagonales simétricas, pues la ley de inercia de Sylvester garantiza que $Inercia(A) = Inercia(X^t AX)$, donde X es una matriz no singular, y además, si $A - \mu I = L_\mu D_\mu L_\mu^t$, entonces $Inercia(A - \mu I) = Inercia(D_\mu)$. Además, del teorema anterior se puede concluir que, por un lado, $\nu(D_\mu)$ es el número de elementos negativos de la diagonal de D_μ y, por otro lado, $\nu(A - \mu M)$ es el número de valores propios del par de matrices (A, M) que son más pequeños que μ.

El método de bisección explota las propiedades anteriores, pero primero vamos a definir la **secuencia de Sturm** y una de sus propiedades.

Secuencia de Sturm

Dada una matriz tridiagonal simétrica

$$T = T_n = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix} \tag{4.10}$$

donde, sin pérdida de generalidad, se asume que $\beta_i \neq 0$ $(i = 1, 2, \ldots, n-1)$, y se denomina **matriz irreducida**. En el caso en el que alguno de los β_i sea nulo, se tiene

$$T = \begin{bmatrix} \widehat{T} & 0 \\ 0 & \overline{T} \end{bmatrix} \tag{4.11}$$

donde $\widehat{T}$ y $\overline{T}$ son bloques matriciales tridiagonales simétricos irreducidos y, por lo tanto, los valores propios de T son los valores propios de $\widehat{T}$ y de $\overline{T}$. Sea

$$P_n(\lambda) = \det(T_n - \lambda I_n) \tag{4.12}$$

el polinomio característico de la matriz T_n.

En estas condiciones, existe una recurrencia de segundo orden que permite el cálculo del polinomio característico en cualquier punto y que se denomina **secuencia de Sturm**. Dada T_r la submatriz principal de orden r de la matriz T_n, consideramos la secuencia de polinomios característicos P_r de grado r definidos por

$$P_r(\lambda) = \det(T_r - \lambda I_r) \qquad (r = 1, \ldots, n) \tag{4.13}$$

los cuales se pueden construir de forma recursiva como

$$\begin{aligned} P_0(\lambda) &= 1; \\ P_1(\lambda) &= \alpha_1 - \lambda; \\ P_r(\lambda) &= (\alpha_r - \lambda) P_{r-1}(\lambda) - \beta_{r-1}^2 P_{r-2}(\lambda) \qquad (r = 2, \ldots, n). \end{aligned} \tag{4.14}$$

Como se observa, en la secuencia de polinomios (4.14), para calcular el polinomio característico de T_n hay que calcular el polinomio característico de todas las submatrices principales de T_n. Además, fácilmente se demuestra que los ceros de dos polinomios consecutivos de una secuencia de Sturm se intercalan y, al mismo tiempo el número de signos que son iguales en los términos consecutivos en la secuencia (4.14) es igual al número de ceros del polinomio $P_n(\mu)$ que son más pequeños a μ, garantizado por el conjunto de siguientes teoremas.

Teorema 10 (Propiedad de entrelazado no estricto) *Sean A_{r-1} y A_r las submatrices principales de orden $r-1$ y r, respectivamente, de una matriz $A \in \mathbb{R}^{n \times n}$ simétrica.*

Si $\lambda'_1 \leq \lambda'_2 \leq \ldots \leq \lambda'_{r-1}$ *son los valores propios de la matriz* A_{r-1} *y* $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_r$ *son los valores propios de la matriz* A_r*, entonces se cumple*

$$\lambda_1 \leq \lambda'_1 \leq \lambda_2 \leq \lambda'_2 \leq \ldots \leq \lambda'_{r-1} \leq \lambda_r \qquad (r = 1, \ldots, n)\,. \tag{4.15}$$

DEMOSTRACIÓN: La demostración del teorema está en la sección 47 de [Wilkinson, 1965] y en la sección 10.1 de [Parlett, 1998]. ■

La propiedad anterior garantiza que los valores propios de una submatriz principal de una matriz real simétrica sirven como separadores de los valores propios de la submatriz principal de orden inmediatamente inferior. Además, la propiedad es igualmente verdadera en sentido inverso, o sea, los valores propios de la submatriz principal de orden $r-1$ sirven como separadores de los valores propios de la submatriz principal de orden r.

Cabe señalar que en el caso simétrico general, este entrelazado de los valores propios no es estricto, sin embargo, en el caso de matrices tridiagonales simétricas irreducidas, sí que lo es.

Teorema 11 (Propiedad de entrelazado estricto) *Sean* T_{r-1} *y* T_r *las submatrices principales de orden* $r-1$ *y* r*, respectivamente, de una matriz* $T \in \mathbb{R}^{n\times n}$ *tridiagonal simétrica irreducida.*

Si $\lambda'_1 < \lambda'_2 < \ldots < \lambda'_{r-1}$ *son los valores propios de la matriz* T_{r-1} *y* $\lambda_1 < \lambda_2 < \ldots < \lambda_r$ *son los valores propios de la matriz* T_r*, entonces se cumple*

$$\lambda_1 < \lambda'_1 < \lambda_2 < \lambda'_2 < \ldots < \lambda'_{r-1} < \lambda_r \qquad (r = 1, \ldots, n)\,. \tag{4.16}$$

DEMOSTRACIÓN: La demostración del teorema está en la sección 37 de [Wilkinson, 1965] y en la sección 8.5.2 de [Golub y van Loan, 1996]. ■

A partir de la propiedad anterior se puede presentar la siguiente propiedad de la secuencia de Sturm.

Teorema 12 (Propiedad de la secuencia de Sturm) *Si* $T \in \mathbb{R}^{n\times n}$ *es una matriz tridiagonal simétrica irreducida y* λ *es un número real, entonces el número de cambios de signo en la secuencia de Sturm*

$$\{P_0(\lambda), P_1(\lambda), \ldots, P_n(\lambda)\} \tag{4.17}$$

es igual al número de valores propios de T *que son menores que* λ*.*

DEMOSTRACIÓN: La demostración del teorema está en la sección 8.5.2 de [Golub y van Loan, 1996]. ■

En el teorema anterior, si para algún r $(r = 1, \ldots, n)$, se tiene $P_r(\lambda) = 0$, entonce se admite que el signo de P_r es contrario al signo de P_{r-1}.

Además, es evidente que cuando n es grande, la secuencia de Sturm definida en (4.14) puede tener problemas de desbordamiento, hecho estudiado por primera vez en [Barth *et al.*, 1967], en el cual los autores han propuesto el empleo de una **secuencia de Sturm modificada** en la cual cada término de la nueva secuencia es obtenido como el cociente entre dos términos consecutivos de la secuencia anterior, o sea, se define

$$Q_r(\lambda) = \frac{P_r(\lambda)}{P_{r-1}(\lambda)} \qquad (r = 1, \ldots, n) \tag{4.18}$$

y, por lo tanto, la secuencia de Sturm modificada es definida como

$$\begin{aligned} Q_0(\lambda) &= 1; \\ Q_1(\lambda) &= \alpha_1 - \lambda; \\ Q_r(\lambda) &= \alpha_r - \lambda - \frac{\beta_{r-1}^2}{Q_{r-1}(\lambda)} \qquad (r = 2, \ldots, n). \end{aligned} \tag{4.19}$$

En la secuencia anterior, el hecho de que algún Q_{r-1} $(r = 1, \ldots, n)$, sea cero, en sistemas computacionales con aritmética en coma flotante IEEE, no es problemático ya que el estándar IEEE consigue hacer divisiones por cero. Como describe Demmel en [Demmel, 1997], si $Q_{r-1}(\lambda) = 0$, entonces $Q_r(\lambda) = -\infty$ y $Q_{r+1}(\lambda) = \alpha_{r+1} - \lambda$, y el cálculo de los restantes términos de la secuencia pueden seguir sin problema. En [Demmel y Li, 1993] son descritos algunos casos de excepciones que el estándar IEEE resuelve. Además, en [Marques *et al.*, 2005b] los autores exploran los beneficios del empleo del estándar IEEE-754 en el cálculo de la DVP de matrices tridiagonales simétricas.

Es evidente que el número de valores propios de una matriz tridiagonal simétrica irreducida T y que son menores que el número real λ es igual al número de elementos negativos de la secuencia de Sturm modificada obtenida a partir de la descomposición triangular de la matriz tridiagonal desplazada $T - \lambda I$.

4.3. Método de bisección

Dada una matriz $T \in \mathbb{R}^{n\times n}$, tridiagonal simétrica irreducida, supongamos que la matriz tridiagonal simétrica desplazada $T - \lambda I \in \mathbb{R}^{n\times n}$, admite la descomposición triangular $L_\lambda D_\lambda L_\lambda^t$, o sea,

$$
\begin{aligned}
T - \lambda I &= \begin{bmatrix} \alpha_1 - \lambda & \beta_1 & & & & \\ \beta_1 & \alpha_2 - \lambda & \beta_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \beta_{n-2} & \alpha_{n-1} - \lambda & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n - \lambda \end{bmatrix} = \qquad (4.20)\\
&= \begin{bmatrix} 1 & & & \\ l_1 & 1 & & \\ & \ddots & \ddots & \\ & & l_{n-1} & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \begin{bmatrix} 1 & l_1 & & \\ & 1 & \ddots & \\ & & \ddots & l_{n-1} \\ & & & 1 \end{bmatrix} = \\
&= L_\lambda D_\lambda L_\lambda^t
\end{aligned}
$$

y, por tanto,

$$
\begin{aligned}
\alpha_1 - \lambda &= d_1; \qquad (4.21)\\
\beta_1 &= l_1 d_1;\\
\alpha_r - \lambda &= l_{r-1}^2 d_{r-1} + d_r;\\
\beta_r &= l_r d_r.
\end{aligned}
$$

Sustituyendo $l_r = \frac{\beta_r}{d_r}$ en $l_{r-1}^2 d_{r-1} + d_r = \alpha_r - \lambda$, se obtiene la fórmula de recurrencia definida por

$$
\begin{aligned}
d_1 &= \alpha_1 - \lambda; \qquad (4.22)\\
d_r &= \alpha_r - \lambda - \frac{\beta_{r-1}^2}{d_{r-1}} \qquad (r = 2, \ldots, n)
\end{aligned}
$$

y, por lo tanto, la recurrencia anterior permite calcular los elementos de la diagonal de la matriz diagonal de la descomposición triangular de la matriz tridiagonal simétrica desplazada $T - \lambda I$.

Algoritmo 6 Count$(n, \alpha, \beta, \lambda)$

$d = \alpha_1 - \lambda$;
Si $d < 0$ **entonces** $Count = 1$; **si no** $Count = 0$;**Fin**
Para $r = 2, 3, \ldots, n$ **calcular:**
    $d = \alpha_r - \lambda - \frac{\beta_{r-1}^2}{d}$;
    Si $d < 0$ **entonces** $Count = Count + 1$;**Fin**
Fin

En estas condiciones, el Algoritmo 6 permite calcular el número de valores propios de una matriz tridiagonal simétrica irreducida que son menores que λ, donde α y β son los vectores con los elementos de la matriz tridiagonal T de orden n, y λ es el valor del desplazamiento. Además, por ejemplo, en [Volkov y Demmel, 2008a] se presenta un estudio de la implementación anterior en sistemas computacionales con GPUs.

El Algoritmo 6 es la base del **método de bisección** para el cálculo de los valores propios de una matriz tridiagonal simétrica irreducida T. De hecho, dados τ_1 y τ_2, tales que $\tau_1 < \tau_2$, el número de valores propios de la matriz T que están en el intervalo $[\tau_1, \tau_2[$ es igual a

$$Count\,(n, \alpha, \beta, \tau_2) - Count\,(n, \alpha, \beta, \tau_1)\,. \tag{4.23}$$

Empleando el método de bisección se puede obtener una secuencia de subintervalos, contenida en $[\tau_1, \tau_2[$, donde cada uno de los subintervalos tiene una amplitud que es siempre la mitad de la amplitud del intervalo anterior. El proceso termina cuando se encuentre un subintervalo con una amplitud inferior a una tolerancia predefinida y con esto se concluye la llamada etapa de aislamiento del método de bisección.

El Algoritmo 7, que describe este método, calcula todos los valores propios de la matriz tridiagonal simétrica irreducida T existentes en el intervalo $[\tau_1, \tau_2[$, para una cierta cota de error δ. Como se observa, el método de bisección permite calcular un valor propio usando alrededor de $2\varepsilon n$ sumas y εn divisiones, donde ε representa el número de bits de la precisión con la cual se desea calcular el valor propio. Todos los valores propios de la matriz T pueden ser calculados en $\mathcal{O}\left(\varepsilon n^2\right)$ operaciones en coma flotante. En LAPACK, el método de bisección

está implementado en la rutina `xSTEBZ`[1].

Algoritmo 7 *Bisección* $(n, \alpha, \beta, \tau_1, \tau_2, \delta)$

$n_{\tau_1} = Count\,(n, \alpha, \beta, \tau_1)$;
$n_{\tau_2} = Count\,(n, \alpha, \beta, \tau_2)$;
Si $n_{\tau_1} = n_{\tau_2}$ **entonces** $EXIT$ **Fin** /*No hay valores propios en $[\tau_1, \tau_2[$*/
Colocar $[\tau_1, n_{\tau_1}, \tau_2, n_{\tau_2}]$ en una $Worklist$
    /* $Worklist$ contiene la lista de subintervalos de $[\tau_1, \tau_2[$
    que contiene los valores propios de $n - n_{\tau_1}$ hasta $n - n_{\tau_2} + 1$ */
Mientras $Worklist$ no está vacia **hacer**
    Extraer $[low, n_{low}, up, n_{up}]$ de la $Worklist$
    Si $(up - low < \delta)$ **entonces**
        Existen $n_{up} - n_{low}$ valores propios en $[low, up[$
    si no
        $mid = \frac{low+up}{2}$;
        $n_{mid} = Count\,(n, \alpha, \beta, mid)$;
        Si $n_{mid} > n_{low}$ **entonces**
            Colocar $[low, n_{low}, mid, n_{mid}]$ en la $Worklist$
        Fin
        Si $n_{up} > n_{mid}$ **entonces**
            Colocar $[mid, n_{mid}, up, n_{up}]$ en la $Worklist$
        Fin
    Fin
Fin

Una de las propiedades fundamentales del cálculo de los valores propios de matrices reales simétricas es su buen condicionamiento en el sentido de que pequeñas perturbaciones en las entradas de la matriz dan lugar a pequeñas modificaciones en los valores propios. Esta propiedad es verificada por el siguiente lema.

Lemma 13 *Los valores d_r de la recurrencia (4.22), calculados en aritmética de coma flotante, tienen los mismos signos (y, por tanto, la misma inercia), que los valores $\widetilde{d}_r$ calculados exactamente a partir de $\widetilde{T}$, donde $\widetilde{T}$ está muy cercana*

[1] Para más información consultar el apéndice "Rutinas de LAPACK y de SCALAPACK".

a T*:*

$$\widetilde{\alpha}_r = \alpha_r \qquad y \qquad \widetilde{\beta}_r = \beta_r \left(1 + \varepsilon_r\right) \tag{4.24}$$

con $|\varepsilon_r| \leq 2{,}5\varepsilon + O\left(\varepsilon^2\right)$*, donde* ε *es el epsilon de la máquina.*

DEMOSTRACIÓN: La demostración del lema está en la sección 5.3.4 de [Demmel, 1997]. ■

Otra cuestión que cabe señalar es que el método de bisección para ser empleado necesita de un intervalo inicial que contenga todos los valores propios de la matriz tridiagonal simétrica irreducida T. Para ello, se aplica el teorema de los círculos de Gershgorin, adaptado a las entradas de la matriz tridiagonal simétrica y, por lo tanto, todos los valores propios de la matriz T están en el intervalo definido por

$$\left[\min_{1\leq i\leq n}\left\{\alpha_i - \left|\beta_{i-1}\right| - \left|\beta_i\right|\right\}, \max_{1\leq i\leq n}\left\{\alpha_i + \left|\beta_{i-1}\right| + \left|\beta_i\right|\right\}\right] \tag{4.25}$$

con $\beta_0 = \beta_n = 0$.

Para terminar la sección hay que tener en cuenta que una vez calculado un valor propio de la matriz T con la precisión numérica deseada $(\widetilde{\lambda})$, el **método de la iteración inversa** puede ser empleado para calcular su correspondiente vector propio [Peters y Wilkinson, 1971], [Jessup y Ipsen, 1992], [Ipsen, 1997]

$$\begin{aligned} x^{(0)} &= v \\ \left(T - \widetilde{\lambda} I\right) x^{(i+1)} &= \kappa^{(i)} x^{(i)} \qquad (i = 0, 1, \ldots) \end{aligned} \tag{4.26}$$

donde v es un vector inicial y $\kappa^{(i)}$ es un número real positivo que normaliza el vector $x^{(i+1)}$.

Sin embargo, puede que el método de iteración inversa calcule vectores propios, asociados a los distintos valores propios de T, que no sean ortogonales. Para evitar este problema, se empleará el **método MRRR** para calcular los vectores propios de la matriz tridiagonal simétrica irreducida T.

4.4. Método MRRR

El **método MRRR** (*Multiple Relatively Robust Representations* o **Múltiples Representaciones Relativamente Robustas**), desarrollado por Parlett y por Dhillon, es un método numérico al cual se han dedicado muchos autores y, por ejemplo, en los *working notes* de LAPACK se pueden encontrar algunos de esos trabajos, y algunos de los cuales se han empleado como referencia bibliográfica para esta sección: [Dhillon, 1997], [Parlett y Dhillon, 2000], [Dhillon y Parlett, 2004a], [Dhillon y Parlett, 2004b], [Dhillon *et al.*, 2006].

Aunque en esta sección solamente se describan las ideas principales del método, por ejemplo, en [Bientinesi *et al.*, 2005], en [Antonelli y Vömel, 2005] y en [Vömel, 2010], se puede encontrar una exposición de la implementación paralela del método.

El método MRRR está basado en tres principios fundamentales:

1. Las representaciones relativamente robustas.
2. El cálculo de un vector propio asociado a un valor propio aislado.
3. El cálculo de vectores propios ortogonales en clusters de valores propios, empleando múltiples representaciones relativamente robustas.

4.4.1. Representaciones relativamente robustas

Una **representación relativamente robusta** (RRR), es una representación en la cual los valores y vectores propios de una matriz están determinados con elevada precisión relativa, o sea, pequeñas perturbaciones en las componentes individuales de los elementos de la matriz producen pequeñas perturbaciones relativas en los valores propios y pequeñas perturbaciones en los respectivos vectores propios.

En general, la descomposición bidiagonal LDL^t de una matriz tridiagonal definida positiva es una RRR, pero las condiciones siguientes determinan cuándo la descomposición LDL^t es una RRR.

Sean l_i $(i = 1, \ldots, n-1)$ los elementos $L(i+1, i)$ de la matriz L, y d_i $(i = 1, \ldots, n)$ los elementos diagonales de la matriz D. Se define **gap relativo** de $\widetilde{\lambda}$ ($relgap$), donde $\widetilde{\lambda}$ está más cerca de λ que de cualquier otro valor

propio de LDL^t, a

$$relgap\left(\widetilde{\lambda}\right) = \frac{\text{mín}\left\{\left|\tau - \widetilde{\lambda}\right| : \tau \neq \lambda, \tau \in \sigma\left(LDL^t\right)\right\}}{\left|\widetilde{\lambda}\right|}. \tag{4.27}$$

Sean λ un valor propio, x su correspondiente vector propio, y $\lambda + \delta\lambda$ y $x + \delta x$ los correspondientes valor propio y vector propio perturbados. Se dice que el par (λ, x) está determinado con elevada precisión relativa por las matrices L y D, si pequeñas perturbaciones relativas

$$l_i \rightsquigarrow l_i\left(1 + \eta_i\right) \qquad |\eta_i| < \xi \ll 1 \tag{4.28}$$

$$d_i \rightsquigarrow d_i\left(1 + \delta_i\right) \qquad |\delta_i| < \xi \ll 1 \tag{4.29}$$

producen perturbaciones $\delta\lambda$ y δx, que verifican

$$|\delta\lambda| \leq \kappa_1 n \xi \left|\lambda\right| \qquad \lambda \neq 0 \tag{4.30}$$

$$\sin\angle\left(x, x + \delta x\right) \leq \frac{\kappa_2 n \xi}{relgap\left(\lambda\right)} \tag{4.31}$$

para valores de las constantes κ_1 y κ_2 relativamente pequeños (habitualmente más pequeños que 10).

En las condiciones anteriores, se dice que la descomposición LDL^t es una RRR de (λ, x) en la cual los valores y vectores propios son calculados con elevada precisión relativa, verificando (4.30) y (4.31).

4.4.2. Cálculo de un vector propio asociado a un valor propio aislado

Una vez calculado un valor propio $\widetilde{\lambda}$ con elevada precisión, su correspondiente vector propio puede ser calculado resolviendo aproximadamente el sistema

$$\left(LDL^t - \widetilde{\lambda} I\right) x \approx 0. \tag{4.32}$$

Sin embargo, el sistema anterior, en muchos casos, puede ser cuasi-singular o singular, ocasionando problemas en el cálculo del vector propio.

En [Dhillon y Parlett, 2004b] fue expuesto por primera vez el método `Getvec`, el cual calcula el vector propio asociado a un valor propio aislado usando una combinación de la descomposición *top-down* LDL^t, con la descomposición *bottom-up* UDU^t.

A continuación se presenta el algoritmo que implementa el método `Getvec`, basado en las descripciones expuestas en [Dhillon y Parlett, 2004b] y en [Dhillon y Parlett, 2004a], donde l_i $(i = 1, \ldots, n-1)$ son los elementos $L(i+1, i)$ de la matriz L, d_i $(i = 1, \ldots, n-1)$ los elementos diagonales de la matriz D y $\widetilde{\lambda}$ una aproximación al valor propio exacto.

Algoritmo 8 *Getvec* $\left(L, D, \widetilde{\lambda}\right)$

Calcular $LDL^t - \widetilde{\lambda}I = L_+ D_+ L_+^t$ empleando la transformación *dstqds*
Calcular $LDL^t - \widetilde{\lambda}I = U_- D_- U_-^t$ empleando la transformación *dqds*
Para $i = 1, \ldots, n$ **hacer:**
$\gamma_i = s_i + \frac{d_i}{D_-(i+1)} p_{i+1}$; /* s_i y p_{i+1} son calculados por las transformaciones *dstqds* y *dqds* */
Escoger un r tal que $|\gamma_r| = \text{mín}_i |\gamma_i|$;
Calcular N_r y Δ_r, tales que $N_r \Delta_r N_r^t = LDL^t - \widetilde{\lambda}I$;
Fin
Calcula el vector propio x resolviendo $N_r^t x = e_r : x(r) = 1$

Para $i = r-1, \ldots, 1$ **hacer:** $x(i) = \begin{cases} -L_+(i)\, x(i+1) & x(i+1) \neq 0 \\ -\left(\frac{d_{i+1} l_{i+1}}{d_i l_i}\right) x(i+2) & \textit{otro caso} \end{cases}$

Para j=r,. . . ,n-1 **hacer:** x$(j+1) = \begin{cases} -U_-(j)\, x(j) & x(j) \neq 0 \\ -\left(\frac{d_{j-1} l_{j-1}}{d_j l_j}\right) x(j-1) & \textit{otro caso} \end{cases}$

Definir $x = \frac{x}{\|x\|}$;

El método `Getvec` calcula un solo vector propio de una RRR. En [Dhillon y Parlett, 2004a] está demostrado que el vector propio calculado tiene una precisión elevada y además, es numéricamente ortogonal a los restantes vectores propios, si los correspondientes valores propios tienen un gap relativo

grande. Sin embargo, en el caso en que los valores propios estén dentro de un cluster, cuando se emplea este método, el vector propio calculado no es, en general, ortogonal a los demás vectores propios asociados a los valores propios del cluster. La razón para esta pérdida de ortogonalidad es que los vectores propios y los valores propios con pequeños gaps relativos son muy sensibles a pequeñas perturbaciones en las entradas de las matrices L y D.

4.4.3. Cálculo de vectores propios ortogonales en clusters de valores propios

Empleando RRR y el método `Getvec` para calcular vectores propios, en [Dhillon y Parlett, 2004a] fue demostrado que el vector propio calculado es numéricamente ortogonal a los otros vectores propios, cuando los valores propios tienen un gap relativo grande. Sin embargo, cuando los valores propios están dentro de un cluster, la estrategia anterior es inadecuada y hay que emplear el **método MRRR**, el cual utiliza múltiples RRR.

El método MRRR emplea múltiples descomposiciones

$$L_c D_c L_c^t = LDL^t - \tau_c I \tag{4.33}$$

donde τ_c está cerca de un cluster. Las traslaciones τ_c son escogidas de modo adecuado y para incrementar el gap relativo entre los valores propios, o sea, los valores τ_c de (4.33) son escogidos de modo que la nueva descomposición sea RRR y además, por lo menos uno de los valores propios del cluster esté aislado de los restantes valores propios.

Una vez calculada la nueva descomposición $L_c D_c L_c^t$, los valores propios dentro del cluster son refinados de manera que tengan una elevada precisión relativa respecto a $L_c D_c L_c^t$. Finalmente, los vectores propios de los valores propios que están relativamente aislados pueden ser calculados por el método `Getvec` empleando $L_c D_c L_c^t$. El proceso es iterativo para valores propios que aún tienen gaps relativos muy pequeños. En el método MRRR, la ortogonalidad de los vectores propios calculados está garantizada sin el empleo del método de ortogonalización de Gram-Schmidt.

Seguidamente se expone el algoritmo que implementa el método MRRR, donde T es la matriz tridiagonal simétrica irreducida, Γ_0 es el conjunto de índices de los pares propios deseados y *tol* es la cota de error para los gaps relativos. El

algoritmo del método MRRR emplea una llamada a la rutina $MRRR\_Vec$ la cual se encuentra descrita en Algoritmo 10.

Algoritmo 9 $MRRR\,(T, \Gamma_0, tol)$

Subdividir T en bloques $T_1, T_2, \ldots, T_l$
Para cada bloque T_i, $i = 1, 2, \ldots, l$ **hacer:**
A. Escoger μ_i. Calcular L_0 y D_0, tales que
$L_0 D_0 L_0^t = T_i + \mu_i I$ es la descomposición que
determina los valores y vectores propios deseados,
λ_j y x_j $j \in \Gamma_0$, con elevada precisión relativa.
En general, la traslación μ_i puede ser escogida
dentro del espectro de T_i, sin embargo, la mejor
selección es hacer $T_i + \mu_i I$ definida positiva o negativa
B. Calcular los valores propios deseados de $L_0 D_0 L_0^t$,
con elevada precisión relativa, empleando *dqds*
C. Construir la $Worklist$ Q e inicializar $Q = \{(L_0, D_0, \Gamma_0)\}$.
Llamar a $MRRR\_Vec\,(Q, tol)$
Fin

4.5. Método *zeroinNR*

En el cálculo de los valores propios de una matriz tridiagonal simétrica, el método de bisección, aunque garantice el cálculo de los valores propios con la precisión numérica deseada debido a su convergencia global, la velocidad de convergencia es lineal. Para mejorar la velocidad de convergencia es aconsejable combinar el método de bisección con otro método numérico de aproximación de raíces de polinomios, como es, por ejemplo, el método de Newton-Raphson o el método de Laguerre. Así, muchos autores han propuesto métodos "híbridos", en los cuales el método de bisección es empleado en la etapa de aislamiento y el otro método numérico es empleado en la etapa de extracción.

Una de las distintas propuestas de combinar el método de bisección con otro método numérico fue presentada por Ralha en su tesis doctoral [Ralha, 1990], donde el autor describe por primera vez el método `zeroinNR`, el cual es una variante del método `zeroin` [Philippe *et al.*, 1987], en el que se combina el

Algoritmo 10 $MRRR\_Vec(Q, tol)$

Mientras la lista Q no está vacía **hacer:**
Extraer un elemento (L, D, Γ) de la lista Q
/* Particionar los valores propios calculados $\widetilde{\lambda}_j$, $j \in \Gamma$
en clusters $\Gamma_1, \ldots, \Gamma_h$ según sus gaps relativos y según tol.
Si $relgap\left(\widetilde{\lambda}_j\right) = \frac{\min_{i \neq j} |\widetilde{\lambda}_j - \widetilde{\lambda}_i|}{|\widetilde{\lambda}_j|} \geq tol$, entonces $\widetilde{\lambda}_j$ es aislado
Si no, todos los valores propios consecutivos $\widetilde{\lambda}_{j-1}$, $\widetilde{\lambda}_j$
verifican $\frac{|\widetilde{\lambda}_j - \widetilde{\lambda}_{j-1}|}{|\widetilde{\lambda}_j|} < tol$ */
Para cada cluster Γ_c, $c = 1, \ldots, h$ **hacer:**
Si $|\Gamma_c| = 1$ con valor propio $\widetilde{\lambda}_j$
entonces
Llamar $Getvec\left(L, D, \widetilde{\lambda}_j\right)$ para calcular $\widetilde{x}_j$
si no
a. Escoger τ_c muy cerca del cluster y calcular
$LDL^t - \tau_c I = L_c D_c L_c^t$ usando $dstqds$
b. Refinar los valores propios $\widetilde{\lambda} - \tau_c$ del cluster,
de modo que tengan elevada precisión relativa
respecto a $L_c D_c L_c^t$.
Definir $\widetilde{\lambda} \leftarrow \left(\widetilde{\lambda} - \tau_c\right)_{refined}$ para todos los valores
propios dentro del cluster
c. Añadir (L_c, D_c, Γ_c) a la lista Q
Fin
Fin
Fin

método de bisección con el método de Newton-Raphson, empleando una nueva formulación para el cálculo de la corrección del método de Newton-Raphson.

El método de Newton-Raphson emplea la fórmula iterativa

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{4.34}$$

en el cálculo de las sucesivas aproximaciones a la solución de la ecuación no lineal

$f(x) = 0$, partiendo de una aproximación inicial x_0 cercana a la solución deseada.

Concluida la etapa de aislamiento del método de bisección, el método `zeroinNR` calcula el valor propio existente en cada uno de los respectivos subintervalos, con velocidad de convergencia cuadrática (siempre que el valor propio tenga multiplicidad algébrica igual a uno), donde cada valor propio es una raíz del polinomio característico P_n de la matriz tridiagonal simétrica irreducida T.

En este caso, la corrección del método de Newton-Raphson es la razón

$$\frac{P_n(x_k)}{P'_n(x_k)} \tag{4.35}$$

pero el método `zeroinNR` intenta calcular (4.35) sin calcular explícitamente los valores de $P_n(x_k)$ y de $P'_n(x_k)$.

Como ya fue descrito, $P_n(x_k)$, puede ser calculado empleando la secuencia de Sturm modificada (4.19) y, por lo tanto, para $r = 1, \ldots, n$, se tiene

$$P_r(x_k) = Q_r(x_k) P_{r-1}(x_k) \tag{4.36}$$

$$P'_r(x_k) = Q'_r(x_k) P_{r-1}(x_k) + Q_r(x_k) P'_{r-1}(x_k) \tag{4.37}$$

y, por tanto,

$$\frac{P'_r(x_k)}{P_r(x_k)} = \frac{Q'_r(x_k)}{Q_r(x_k)} + \frac{P'_{r-1}(x_k)}{P_{r-1}(x_k)}. \tag{4.38}$$

La relación anterior tiene especial interés pues describe una razón que es la inversa a la corrección del método de Newton-Raphson para los polinomios P_r y P_{r-1}, y para su cociente Q_r. En estas condiciones, (4.35) puede ser calculada de manera recursiva empleando (4.38), siempre que el término $\frac{Q'_r}{Q_r}$ pueda ser fácil de calcular.

A partir de (4.19), para $r = 2, \ldots, n$, se tiene

$$Q'_r(x_k) = -1 + \frac{\beta^2_{r-1} Q'_{r-1}(x_k)}{Q^2_{r-1}(x_k)} \tag{4.39}$$

y

$$\frac{Q'_r(x_k)}{Q_r(x_k)} = \frac{1}{Q_r(x_k)} \left[-1 + \frac{\beta^2_{r-1}}{Q_{r-1}(x_k)} \frac{Q'_{r-1}(x_k)}{Q_{r-1}(x_k)} \right]. \tag{4.40}$$

Según Ralha, empleando la notación

$$\Delta Q_r(x_k) = \frac{Q_r'(x_k)}{Q_r(x_k)} \qquad \text{y} \qquad \Delta P_r(x_k) = \frac{P_r'(x_k)}{P_r(x_k)} \tag{4.41}$$

el cálculo completo de

$$\Delta P_n(x_k) = \frac{P_n'(x_k)}{P_n(x_k)} \tag{4.42}$$

es expresado en términos de las siguientes relaciones

$$\begin{aligned} Q_1(x_k) &= \alpha_1 - x_k; \qquad (4.43)\\ \Delta Q_1(x_k) &= \Delta P_1(x_k) = -\frac{1}{Q_1(x_k)};\\ &\left.\begin{aligned} m_r(x_k) &= \frac{\beta_{r-1}^2}{Q_{r-1}(x_k)};\\ Q_r(x_k) &= \alpha_r - x_k - m_r(x_k);\\ \Delta Q_r(x_k) &= \frac{-1 + m_r(x_k)\,\Delta Q_{r-1}(x_k)}{Q_r(x_k)};\\ \Delta P_r(x_k) &= \Delta Q_r(x_k) + \Delta P_{r-1}(x_k); \end{aligned}\right\} \quad (r = 2, \ldots, n). \end{aligned}$$

Es importante observar que en (4.43), los signos de los valores Q_r pueden ser empleados en la descripción de un método "híbrido", en el cual se emplea el método de bisección en la etapa de aislamiento y el método de Newton-Rapshon con la nueva formulación del cálculo de su corrección en la etapa de extracción. Este es el método `zeroinNR`.

Dado el intervalo $[a, b]$ con un solo valor propio de la matriz tridiagonal simétrica irreducida T y una aproximación x_k, el método `zeroinNR` calcula la aproximación x_{k+1} empleando (4.42) y (4.43). Puede ocurrir que en la etapa de extracción el método de Newton-Raphson calcule una aproximación x_{k+1} que no pertenezca al intervalo que contiene el valor propio. En ese caso, la aproximación x_{k+1} es actualizada con el punto medio de ese intervalo.

4.6. Resultados experimentales

Tal como hemos hecho en el capítulo anterior, para hacer un análisis de la implementación secuencial del método `zeroinNR`, hay que definir algunos casos

de prueba y hay que probar la implementación desarrollada con distintas matrices tridiagonales simétricas.

Es natural que las prestaciones obtenidas por las implementaciones secuenciales empleadas y desarrolladas en el marco de la tesis para la DVP de matrices tridiagonales simétricas, dependan del tipo de matriz con la que se está trabajando. De echo, el comportamiento de las prestaciones obtenidas suele ser distinto dependiendo de la distribución de los valores propios a lo largo del espectro de la matriz, es decir, de lo homogéneamente que se repartan los valores propios y si algunos se encuentran agrupados en *clusters*.

Para valorar los resultados obtenidos es fundamental emplear un conjunto de matrices de prueba adecuado. Tal como en la sección 3.6, en los primeros experimentos computacionales se han empleado las matrices de prueba generadas por la rutina `DLATMS`[2] de LAPACK [Demmel y McKenney, 1989], descritas en [Demmel *et al.*, 2008a], y cuyos valores propios λ_i $(i = 1, \ldots, n)$ se distribuyen a lo largo de su espectro según la información expuesta en la Tabla 4.1, donde k puede ser definido por $k = ulp^{-1}$, donde *ulp* significa *unit in the last place*.

Tipo	**Descripción**
1	$\lambda_1 = 1 \wedge \lambda_i = \frac{1}{k}$ $(i = 2, \ldots, n)$
2	$\lambda_i = 1$ $(i = 1, \ldots, n-1) \wedge \lambda_n = \frac{1}{k}$
3	$\lambda_i = k^{-\frac{i-1}{n-1}}$
4	$\lambda_i = 1 - \frac{i-1}{n-1}\left(1 - \frac{1}{k}\right)$
5	n números aleatorios en $\left]\frac{1}{k}, 1\right[$ con logaritmos homog. dist.
6	n números aleatorios con una distribución específica
7	$\lambda_i = ulp \times i$ $(i = 1, \ldots, n-1) \wedge \lambda_n = 1$
8	$\lambda_1 = ulp \wedge \lambda_i = 1 + \sqrt{ulp} \times i$ $(i = 2, \ldots, n-1) \wedge \lambda_n = 2$
9	$\lambda_1 = 1 \wedge \lambda_i = \lambda_{i-1} + 100 \times ulp$ $(i = 2, \ldots, n)$

Tabla 4.1: Distribución de los valores propios de las matrices de LAPACK.

[2] Para más información, consultar el Apéndice "Rutinas de LAPACK y SCALAPACK".

Posteriormente hemos desarrollado un módulo auxiliar con un conjunto de matrices de prueba, algunas de las cuales han sido expuestas y estudiadas en la sección 4.1 de la tesis doctoral [Badía, 1996]. La Tabla 4.2 hace una descripción de esas matrices, donde α_i $(i = 1, \ldots, n)$ son los elementos de la diagonal y β_j $(j = 1, \ldots, n-1)$ son los elementos fuera de la diagonal.

Tipo	**Elementos**	**Valores Propios**
1	$\alpha_i = a$ $\beta_j = b$	$\lambda_i = a + 2b\cos\left(\dfrac{i\pi}{n+1}\right)$
2	$\alpha_i = \begin{cases} a-b & si \quad i = 1 \\ a & si \quad i = 2, \ldots, n-1 \\ a+b & si \quad i = n \end{cases}$ $\beta_j = b$	$\lambda_i = a + 2b\cos\left(\dfrac{(2i-1)\,\pi}{2n}\right)$
3	$\alpha_i = 0$ $\beta_j = \sqrt{j\,(n-j)}$	$\lambda_i = -n + 2i - 1$
4	$\alpha_i = -\left[(2i-1)\,(n-1) - 2\,(i-1)^2\right]$ $\beta_j = j\,(n-j)$	$\lambda_i = -i\,(i-1)$
5	$\alpha_i = \begin{cases} \dfrac{m}{2} - i + 1 & si \quad i = 1, \ldots, \dfrac{m}{2} \\ i - \dfrac{m}{2} & si \quad i = \dfrac{m}{2} + 1, \ldots, m \end{cases}$ $m = \begin{cases} n & si \quad n \text{ es par} \\ n+1 & si \quad n \text{ es impar} \end{cases}$ $\beta_j = 1$	Matrices de Wilkinson

Tabla 4.2: Matrices de prueba tridiagonales simétricas.

Respecto a las matrices de la Tabla 4.2, se ha empleado, en particular, el siguiente conjunto de matrices tridiagonales simétricas irreducidas:

- **Caso 1**: matriz del tipo 1 con $a = 2$ y $b = 1$. En este caso, para n grande, el valor propio más pequeño es mucho más pequeño que el redondeo unidad del sistema computacional empleado y además, los valores propios siguen una distribución de Chebyshev en el intervalo $[0, 4]$, cuyos valores son

$$\lambda_i = 4\sin^2\left(\frac{i\pi}{2\,(n+1)}\right). \tag{4.44}$$

- **Caso 2**: matriz del tipo 1 con $a = 4$ y $b = 1$. Este caso es semejante al caso anterior, pero ahora los valores propios pertenecen al intervalo $[2, 6]$, resultantes de un desplazamiento de origen de valor 2.
- **Caso 3**: matriz del tipo 3 con $\beta_j = j$. En este caso, las matrices definen bien sus valores propios, en el sentido de que la introducción de pequeñas perturbaciones relativas en las entradas de la matriz provoca pequeñas perturbaciones en los valores propios. Las matrices tridiagonales con ceros en la diagonal verifican esta propiedad. Los valores propios de las matrices de este caso pertenecen al intervalo $[-2n + 3, 2n - 3]$.
- **Caso 4**: matriz del tipo 5 con n impar, y en particular con $n = 21$. En este caso, la matriz de Wilkinson tiene valores propios, la mayoría formando *clusters* por pares.

Tal y como describe Badía en su tesis doctoral [Badía, 1996], del conjunto anterior de matrices tridiagonales se pueden distinguir grupos de matrices en cuanto a la distribución de sus valores propios a lo largo del espectro. Así, se debe señalar que:

- Los valores propios de las matrices del tipo 1 y del tipo 2, "tienden a acumularse de modo simétrico en los extremos del espectro y su cantidad disminuye paulatinamente hacia el centro del mismo".
- Los valores propios de las matrices del tipo 3 y del tipo 5, "se distribuyen en cantidades iguales a lo largo de los distintos subintervalos del espectro".
- Los valores propios de las matrices del tipo 4, "tienden a acumularse progresivamente hacia el extremo superior del espectro. Además, todos los valores propios son muy grandes".

4.6.1. El método de bisección y la precisión relativa

En esta sección se describen dos ejemplos numéricos que permiten señalar el empleo del método de bisección en el cálculo de los valores propios de matrices tridiagonales simétricas.

En general, se considera que el método de bisección empleado en el cálculo de todo el espectro de la matriz tridiagonal tiene tiempos de ejecución secuenciales

que son más elevados que, por ejemplo, el método iterativo QR, sin embargo, el método de bisección es naturalmente paralelo mientras que el método iterativo QR no.

En el primer ejemplo se considera la matriz tridiagonal simétrica definida por

$$T = \begin{bmatrix} 1 & a & 0 \\ a & b & a \\ 0 & a & 1 \end{bmatrix} \tag{4.45}$$

cuyos valores propios son

$$\begin{aligned} \lambda_1 &= \frac{1}{2}(b+1) - \frac{1}{2}\sqrt{8a^2 + (b-1)^2}; \\ \lambda_2 &= 1; \\ \lambda_3 &= \frac{1}{2}(b+1) + \frac{1}{2}\sqrt{8a^2 + (b-1)^2}. \end{aligned} \tag{4.46}$$

Para evitar la cancelación de dígitos significativos en el cálculo de λ_1, cuando $|a|$ y $|b|$ son valores muy pequeños, se pueden emplear las relaciones

$$\lambda_1 + \lambda_3 = b + 1 \qquad \text{y} \qquad \lambda_1 \times \lambda_3 = b - 2a^2. \tag{4.47}$$

La matriz descrita en [Demmel, 1992] y definida por

$$T = \begin{bmatrix} 1 & 0{,}15 \times 10^{-16} & 0 \\ 0{,}15 \times 10^{-16} & 10^{-32} & 0{,}15 \times 10^{-16} \\ 0 & 0{,}15 \times 10^{-16} & 1 \end{bmatrix} \tag{4.48}$$

permite calcular sus valores propios con elevada precisión relativa, pues es una matriz *scaled diagonally dominant* ya que

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10^{-16} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0{,}15 & 0 \\ 0{,}15 & 1 & 0{,}15 \\ 0 & 0{,}15 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10^{-16} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.49}$$

y sus valores propios son $\lambda_1 = 1$, $\lambda_2 = 1$ y $\lambda_3 = 0{,}955 \times 10^{-32}$.

Para este ejemplo de matriz, las distintas rutinas de LAPACK calculan los valores propios expuestos en la Tabla 4.3. Como se puede observar, las rutinas `DSTEMR` y `DSTEQR` calculan un valor incorrecto de λ_3.

Valor	DSTEBZ	DSTEDC	DSTEMR	DSTEQR	DSTERF
λ_1	λ_1	λ_1	λ_1	λ_1	λ_1
λ_2	λ_2	λ_2	λ_2	λ_2	λ_2
λ_3	λ_3	λ_3	$1{,}0 \times 10^{-32}$	$-2{,}25 \times 10^{-34}$	λ_3

Tabla 4.3: Resultados de las rutinas de LAPACK con la matriz T.

El segundo ejemplo es el cálculo de los valores propios de la matriz Golub-Kahan definida por

$$GK = \begin{bmatrix} 0 & 10^{100} & & & & & & \\ 10^{100} & 0 & 10^{20} & & & & & \\ & 10^{20} & 0 & 10 & & & & \\ & & 10 & 0 & 10 & & & \\ & & & 10 & 0 & 10^{20} & & \\ & & & & 10^{20} & 0 & 10^{100} & \\ & & & & & 10^{100} & 0 & \end{bmatrix} \tag{4.50}$$

cuyos valores propios positivos son los valores singulares de la matriz bidiagonal superior

$$B = \begin{bmatrix} 10^{100} & 10^{20} & & & \\ & 10 & 10 & & \\ & & 10^{20} & 10^{100} & \\ & & & 0 & \end{bmatrix} \tag{4.51}$$

los cuales pueden ser calculados con elevada precisión relativa, pues Demmel y Kahan demostraron que todas las matrices bidiagonales definen muy bien sus valores singulares [Demmel y Kahan, 1988].

Los valores propios de la matriz (4.50) son $\lambda_1 = -10^{100}$, $\lambda_2 = -10^{100}$, $\lambda_3 = -14{,}1421356237310$, $\lambda_4 = 0$, $\lambda_5 = -\lambda_3$, $\lambda_6 = -\lambda_2$ y $\lambda_7 = -\lambda_1$. Todas las rutinas de LAPACK expuestas en la Tabla 4.3 calculan bien los valores propios λ_1, λ_2, λ_6 y λ_7. La diferencia está en el cálculo de los otros valores propios. La Tabla 4.4 presenta los valores obtenidos.

Como se observa en la Tabla 4.4, la rutina que implementa en LAPACK el método de bisección es la única que calcula bien todos los valores propios de la matriz GK.

Valor	DSTEBZ	DSTEDC	DSTEMR	DSTEQR	DSTERF
λ_3	λ_3	$-5{,}7 \times 10^{45}$	0,0	$-2{,}0 \times 10^{-82}$	$-5{,}7 \times 10^{45}$
λ_4	$-2{,}9 \times 10^{-108}$	$-5{,}5 \times 10^{-83}$	0,0	$1{,}3 \times 10^{-29}$	$-4{,}1 \times 10^{-76}$
λ_5	λ_5	$3{,}5 \times 10^{-44}$	0,0	$2{,}4 \times 10^{84}$	$3{,}5 \times 10^{-44}$

Tabla 4.4: Resultados de las rutinas de LAPACK con la matriz GK.

La misma situación ocurre si a la matriz GK añadimos la matriz identidad de orden 7. En este caso, los valores propios de la matriz $GK + I$ son $\lambda_1 = -10^{100}$, $\lambda_2 = -10^{100}$, $\lambda_3 = -13{,}1421356237310$, $\lambda_4 = 1{,}0$, $\lambda_5 = 15{,}1421356237310$, $\lambda_6 = -\lambda_2$ y $\lambda_7 = -\lambda_1$. En esta situación tenemos la Tabla 4.5.

Valor	DSTEBZ	DSTEDC	DSTEMR	DSTEQR	DSTERF
λ_3	λ_3	1,0	1,0	λ_3	1,0
λ_4	λ_4	1,0	1,0	λ_4	1,0
λ_5	λ_5	$2{,}0 \times 10^{62}$	1,0	λ_5	$2{,}0 \times 10^{62}$

Tabla 4.5: Resultados de las rutinas de LAPACK con la matriz $GK + I$.

Los dos ejemplos anteriores demuestran la ventaja en emplear el método de bisección en el cálculo de los valores propios de matrices tridiagonales simétricas, y fueron descritos en [Campos *et al.*, 2008b] y en [Ralha y Campos, 2008].

4.6.2. Prestaciones en secuencial

En la sección anterior fue expuesto que el método de bisección, implementado en LAPACK en la rutina `DSTEBZ`, devuelve aproximaciones a los valores propios de la matriz tridiagonal simétrica con elevada precisión numérica. Además, uno de los parámetros de la rutina `DSTEBZ` es el valor del `ABSTOL`[3], el cual define la cota del error absoluto permitido en el cálculo de los valores propios de la matriz tridiagonal simétrica.

La Figura 4.1 ilustra los tiempos de ejecución obtenidos por la rutina `DSTEBZ` en el cálculo de todos los valores propios de matrices tridiagonales simétricas

[3] Para más información, consultar el Apéndice "Rutinas de LAPACK y SCALAPACK".

del caso 1 del tipo 1, para distintos valores del parámetro `ABSTOL`. Dado que los valores propios de este tipo de matrices pertenecen al intervalo $[0, 4]$, el cálculo de los valores propios más pequeños y más cercanos a cero depende del valor de la cota del error absoluto y, por lo tanto, distintos valores del `ABSTOL` conducen a distintos tiempos de ejecución, pues el número de iteraciones del método de bisección empleadas es distinto. Los tiempos de ejecución de la Figura 4.1 permiten comprobar ese hecho y además, para valorar el comportamiento de la rutina `DSTEBZ` en casos de prueba "limite", en nuestros experimentos computacionales se han empleado valores de `ABSTOL` muy pequeños.

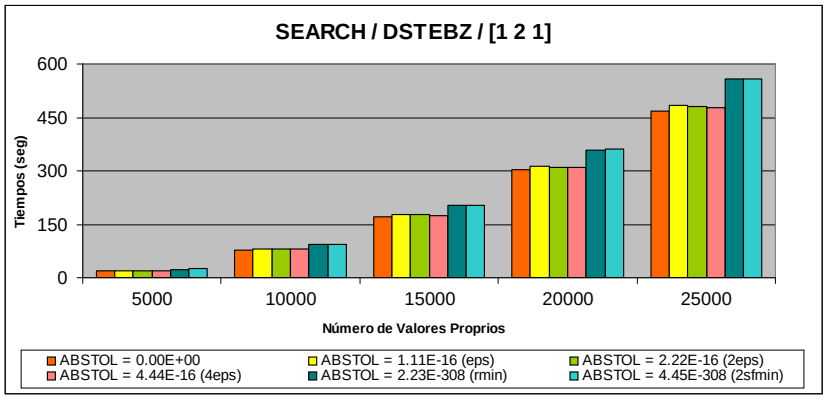

Figura 4.1: Resultados del parámetro `ABSTOL` con la matriz [1 2 1].

Hay que señalar que en los casos de matrices en los cuales los valores propios son de la misma orden de grandeza que la unidad, se ha ejecutado la rutina `DSTEBZ` con `ABSTOL` igual a cero (`0.00E+00`), pues siempre que el parámetro `ABSTOL` asume ese valor, la rutina `DSTEBZ` redefine su valor para $ulp \cdot \|T\|_2$. En el caso contrario, se ha empleado parámetro `ABSTOL` igual a $2sfmin$ (`4.45E-308`).

Al mismo tiempo, la rutina `DSTEBZ` trabaja de manera consistente en el caso de matrices tridiagonales simétricas reducidas, es decir, en el caso en que la matriz tridiagonal simétrica T es una matriz *glued* definida como

$$T = \begin{bmatrix} \widehat{T} & \delta \\ \delta & \overline{T} \end{bmatrix} \tag{4.52}$$

donde $\widehat{T}$ y $\overline{T}$ son bloques matriciales tridiagonales simétricos irreducidos y δ es un número real de unión entre los bloques matriciales.

La Figura 4.2 ilustra los tiempos de ejecución obtenidos con dos bloques matriciales distintos (con la misma dimensión) y empleando $\delta = 0$. Se puede observar que el tiempo de ejecución obtenido en el cálculo de todos los valores propios de la matriz inicial es distinto de la suma de los tiempos de ejecución obtenidos en el cálculo de los valores propios de cada uno de los bloques matriciales. Además, se han desarrollado más experimentos computacionales con distintos bloques matriciales tridiagonales simétricos irreducidos con $\delta = 0$ y los resultados obtenidos son semejantes.

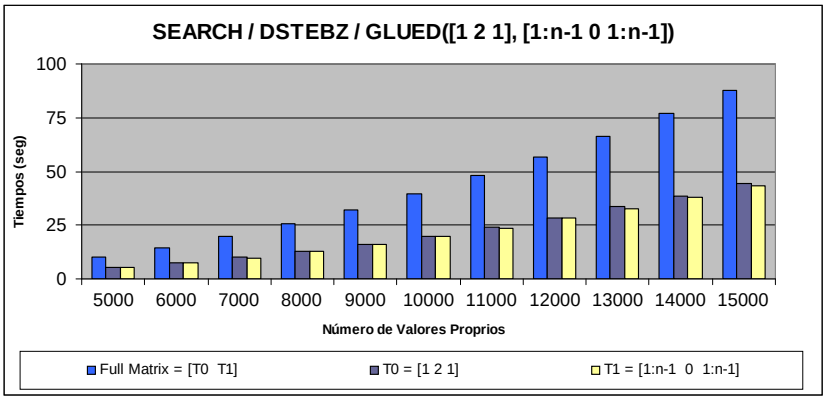

Figura 4.2: Tiempos de ejecución con matrices *glued*.

En cambio, cuando los bloques matriciales son iguales, hay que destacar que la variación del valor de δ en (4.52) hace cambiar los tiempos de ejecución obtenidos con la rutina `DSTEBZ`. La Figura 4.3 y la Figura 4.4 son ejemplos de ello.

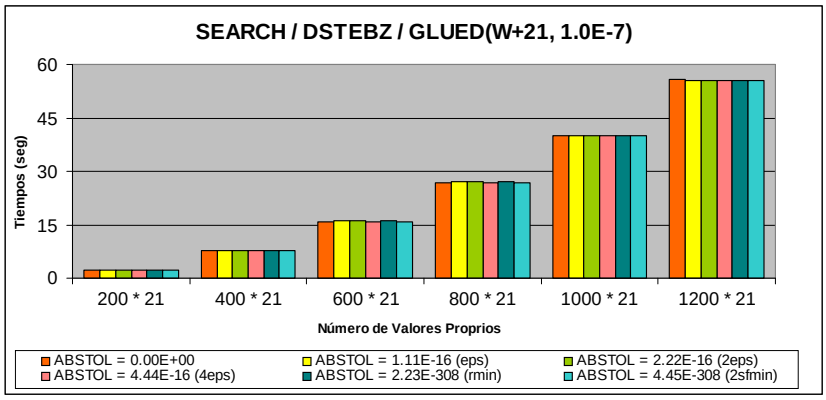

Figura 4.3: Tiempos de ejecución con matrices *glued* con $\delta = 10^{-7}$.

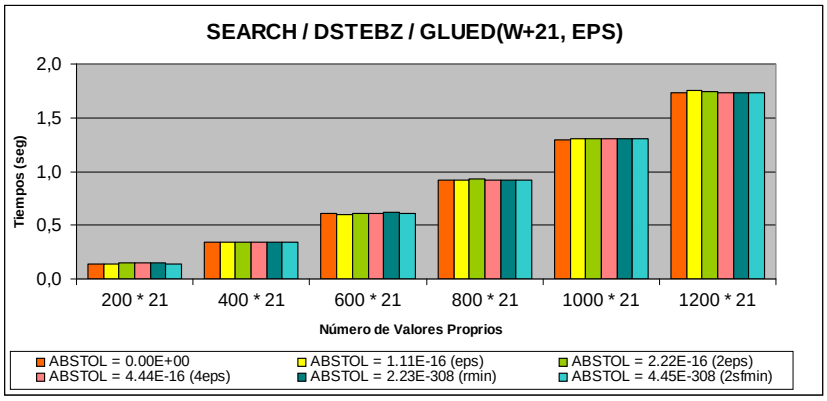

Figura 4.4: Tiempos de ejecución con matrices *glued* con $\delta = eps$.

Como fue ilustrado en la Figura 4.1, debido al hecho de que las matrices tienen sus valores propios en el intervalo $[0, 4]$, hay una dependencia entre el tiempo de ejecución y el valor del parámetro `ABSTOL`.

Al mismo tiempo, la fórmula (4.44) permite calcular el valor exacto de todos los valores propios de cada una de las matrices y, por tanto, la Figura 4.5 y la Figura 4.6 ilustran los valores del error relativo cometido en los 100 primeros valores propios de la matriz de dimensión 1000, con `ABSTOL` igual a cero e igual a $2sfmin$.

Como es natural, los tiempos de ejecución obtenidos con `ABSTOL` igual a $2sfmin$ son mayores que con cero, pero los errores relativos cometidos son más pequeños.

En el caso de la matriz de dimensión 2000, los resultados obtenidos son semejantes (Figura 4.7 y Figura 4.8).

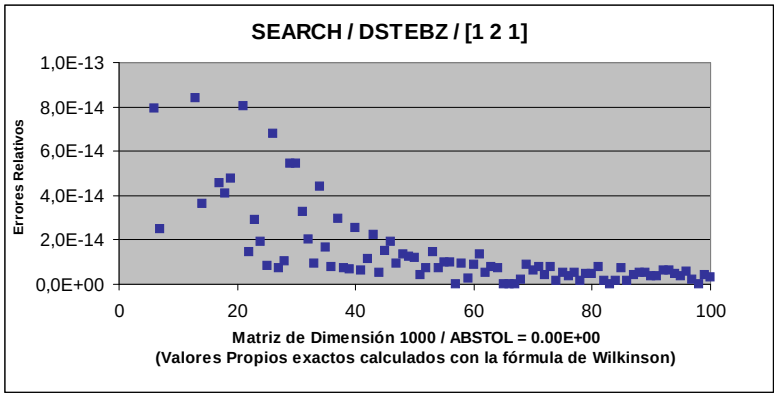

Figura 4.5: Errores relativos de la matriz 1000 con `ABSTOL=cero`.

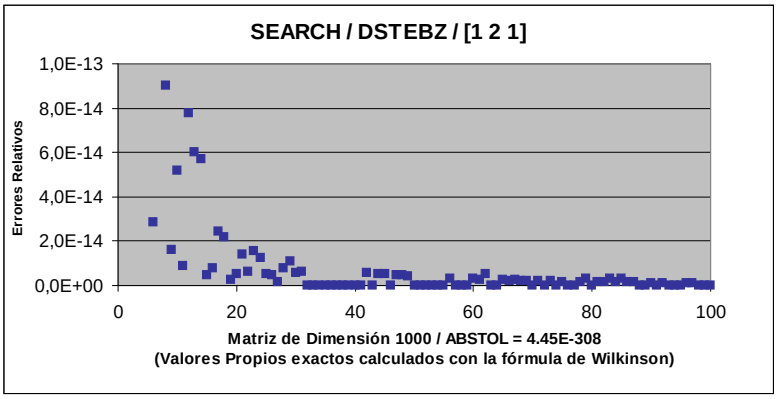

Figura 4.6: Errores relativos de la matriz 1000 con `ABSTOL=2sfmin`.

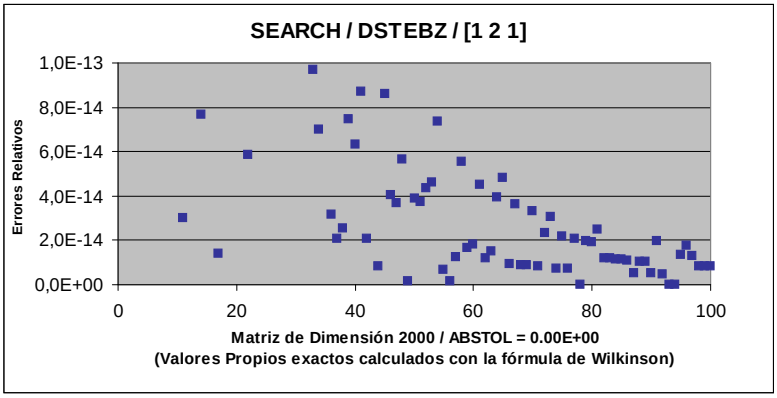

Figura 4.7: Errores relativos de la matriz 2000 con `ABSTOL=cero`.

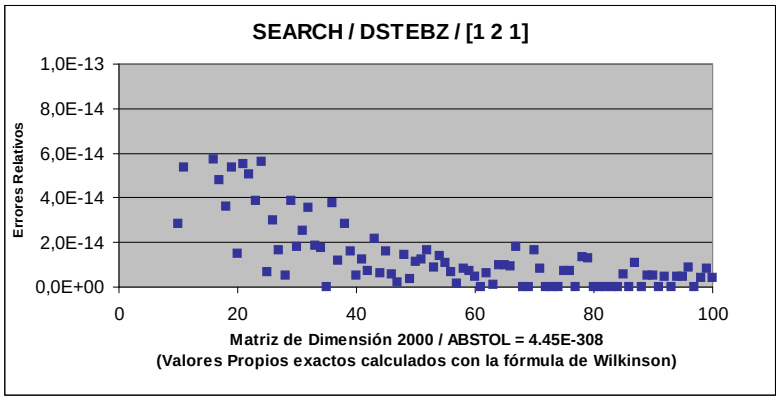

Figura 4.8: Errores relativos de la matriz 2000 con `ABSTOL=2sfmin`.

Otro conjunto de experimentos computacionales que se han desarrollado con la rutina `DSTEBZ` fueron simulaciones "paralelas", en el sentido de que conocidos los limites del intervalo $[GL, GU]$[4], que contiene todos los valores propios de la matriz, los cuales se calculan según (4.25) aplicando el teorema de los círculos de Gershgorin, se puede emplear la metodología de dividir el intervalo $[GL, GU]$ en subintervalos de amplitud $\frac{GU-GL}{p}$, donde p representa el número de procesadores empleados en la ejecución paralela y, seguidamente, cada uno de los p procesadores calcula todos los valores propios contenidos en el subintervalo que le corresponde. Esta metodología no es novedosa y fue empleada, por ejemplo, por Badía en su tesis doctoral [Badía, 1996].

Es evidente que esta metodología es relevante cuando es empleada en matrices tridiagonales simétricas cuyos valores propios se distribuyen homogéneamente a lo largo del intervalo $[GL, GU]$. Sin embargo, en el caso en que eso no sea verdadero, como son, por ejemplo, las matrices del tipo 1 donde los valores propios "tienden a acumularse de modo simétrico en los extremos del espectro y su cantidad disminuye paulatinamente hacia el centro

[4] GL representa el limite *lower* (inferior) del teorema de los círculos de Gershgorin y GU representa el limite *upper* (superior) del mismo teorema.

del mismo", no hay un buen balance de la carga y, por lo tanto, hay procesadores que tienen que trabajar más que otros.

La Figura 4.9 ilustra los tiempos de ejecución obtenidos con la rutina `DSTEBZ` en la simulación "paralela" con 4 procesadores, mientras que la Figura 4.10 es la misma simulación pero con 8 procesadores.

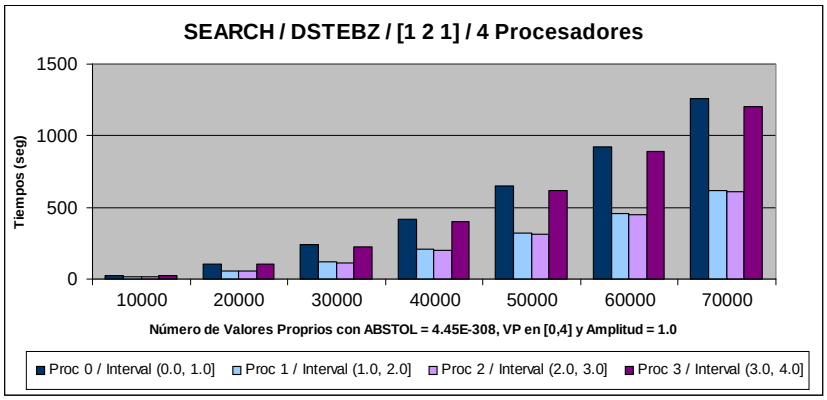

Figura 4.9: Simulación "paralela" con 4 procesadores.

La situación anterior puede ser más problemática a medida que se aumenta el número de procesadores.

La Figura 4.11 ilustra los tiempos de ejecución de los dos primeros procesadores empleados en una simulación "paralela" con 100 procesadores. En este caso, por ejemplo, para la matriz tridiagonal simétrica de dimensión 200000, hay un 6,4 % de valores propios en intervalo $[0{,}00; 0{,}04]$, mientras que la dimensión del intervalo es el 1 % de la amplitud del intervalo inicial. Es evidente que el primer procesador tiene que hacer muchos más cálculos que el segundo y, por tanto, el balanceo de la carga no está equilibrado.

Los ejemplos anteriores dan fuerza al empleo de otra metodología en el cálculo de los valores propios de matrices tridiagonales simétricas de gran dimensión en paralelo. Esa otra metodología es estudiada en el próximo capítulo de la tesis.

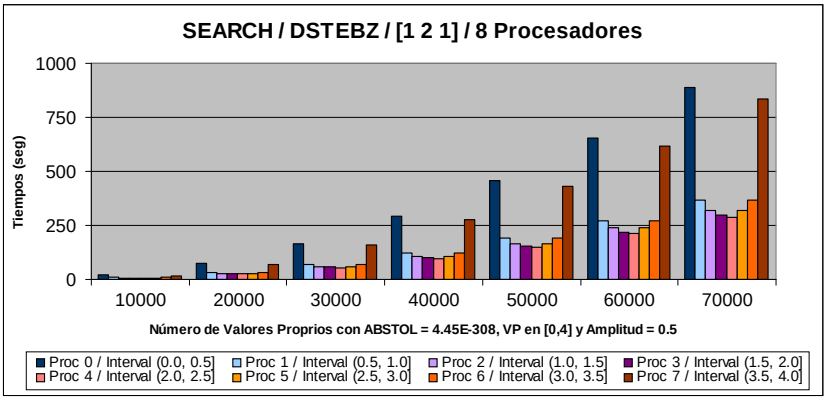

Figura 4.10: Simulación "paralela" con 8 procesadores.

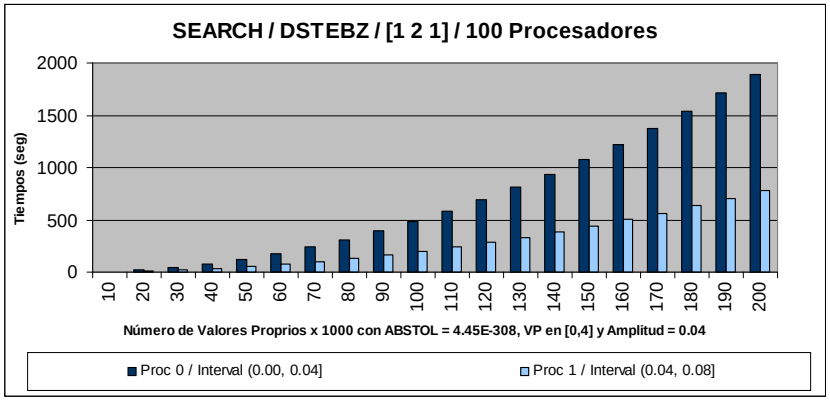

Figura 4.11: Simulación "paralela" con 100 procesadores.

Terminada la presentación de algunos de los resultados obtenidos con la rutina `DSTEBZ`, a continuación se presentan algunos de los resultados obtenidos con la implementación secuencial del método `zeroinNR`, desarrollada en el marco de la tesis. Para ello, vamos a empezar por presentar algunos de los resultados obtenidos referentes a la precisión numérica con que son calculados los valores propios de las matrices de prueba.

Respecto al primer ejemplo de la sección anterior, la Tabla 4.6 presenta los resultados obtenidos con el método `zeroinNR`. Se observa que el método calcula bien los valores propios de la matriz y además, se puede observar el número de iteraciones empleadas por el método de bisección (`Bis`) y por la variante del método de Newton-Raphson (`NR`). La Tabla 4.7 presenta el mismo tipo de resultados, pero para la matriz GK.

DSTEBZ	zeroinNR	Bis	NR
$1{,}00E+00$	$1{,}00E+00$	49	0
$1{,}00E+00$	$1{,}00E+00$	0	0
$9{,}55E-033$	$9{,}55000000000058E-033$	2	3

Tabla 4.6: Resultados del método `zeroinNR` con la matriz T.

DSTEBZ	zeroinNR	Bis	NR
$-1{,}00E+100$	$-1{,}00E+100$	51	0
$-1{,}00E+100$	$-1{,}00E+100$	0	0
$-14{,}1421356237310$	$-14{,}1421356237310$	329	5
$-2{,}9200766458294E-108$	$-1{,}0151767349263E-015$	1	2
$14{,}1421356237310$	$14{,}1421356237310$	2	8
$1{,}00E+100$	$1{,}00E+100$	49	0
$1{,}00E+100$	$1{,}00E+100$	0	0

Tabla 4.7: Resultados del método `zeroinNR` con la matriz GK.

Como se observa en las tablas anteriores, el método `zeroinNR` calcula los valores propios de las matrices de ejemplo con la misma precisión numérica

que la rutina `DSTEBZ`. Eso es igualmente verdadero para los otros ejemplos de matrices de prueba.

Seguidamente se presentan algunos de los resultados obtenidos:

- La Tabla 4.8 presenta los resultados obtenidos con la matriz tridiagonal simétrica del caso 1 del tipo 1 de dimensión 10.

`DSTEBZ`	`zeroinNR`	`Bis`	`NR`
$8{,}10140577100517E-002$	$8{,}10140577100517E-002$	5	8
0,317492934337638	0,317492934337638	1	6
0,690278532109430	0,690278532109430	1	5
1,16916997399623	1,16916997399623	2	5
1,71537032345343	1,71537032345343	1	4
2,28462967654657	2,28462967654657	3	4
2,83083002600377	2,83083002600377	1	5
3,30972146789057	3,30972146789057	2	5
3,68250706566236	3,68250706566236	2	6
3,91898594722899	3,91898594722899	1	7

Tabla 4.8: Resultados del método `zeroinNR` con la matriz del tipo 1.

- La Tabla 4.9 es la situación semejante, pero con la matriz tridiagonal simétrica del tipo 3 con $\beta_j = j$ $(j = 1, \ldots, n-1)$ de dimensión 10.

- La Tabla 4.10 presenta los resultados obtenidos con la matriz de Wilkinson W_{21}^{+} ([Wilkinson, 1965, pp. 309]), y su interés está en la diferencia que existe entre el número de iteraciones empleadas en el cálculo de cada uno de los valores propios de la matriz.

Los resultados anteriores permiten concluir que la implementación secuencial del método `zeroinNR` calcula, en general, los valores propios de las matrices tridiagonales simétricas con la misma precisión numérica que la rutina `DSTEBZ`. Sin embargo, no hay una uniformidad en el número de iteraciones empleadas, tanto por el método de bisección, como por la variante del método de Newton--Raphson, en el cálculo de cada uno de los valores propios de la matriz.

DSTEBZ	zeroinNR	Bis	NR
−13,4279971611570	−13,4279971611570	4	8
−8,51896315328023	−8,51896315328023	1	8
−5,05347662061546	−5,05347662061546	2	7
−2,47861957250507	−2,47861957250507	2	6
−0,659528689823754	−0,659528689823754	1	5
0,659528689823754	0,659528689823754	4	5
2,47861957250507	2,47861957250507	1	6
5,05347662061546	5,05347662061546	1	7
8,51896315328023	8,51896315328023	2	8
13,4279971611570	13,4279971611570	1	8

Tabla 4.9: Resultados del método `zeroinNR` con la matriz del tipo 3.

Aunque numéricamente la implementación secuencial del método `zeroinNR` sea correcta, una de sus mayores ventajas está en los tiempos de ejecución obtenidos cuando son contrastados con los tiempos de ejecución obtenidos por la rutina `DSTEBZ`. La Figura 4.12 y la Figura 4.13 exhiben ese hecho.

4.7. Conclusiones

En este capítulo se han descrito los conceptos del problema estándar del cálculo de la DVP de una matriz tridiagonal simétrica, así cómo del método de bisección empleado en el cálculo de los valores propios de la matriz y del método MRRR empleado en el cálculo de los respectivos vectores propios.

A continuación se ha presentado el método `zeroinNR` desarrollado por Ralha en su tesis doctoral [Ralha, 1990], donde se combina el método de bisección en la etapa de aislamiento con la nueva formulación para el cálculo de la corrección del método de Newton-Raphson en la etapa de extracción.

Posteriormente se han expuesto algunos casos de prueba y los resultados obtenidos con las rutinas de LAPACK y, en especial, con la rutina `DSTEBZ`, la cual es la implementación secuencial del método de bisección.

Para terminar se ha desarrollado un estudio comparativo entre los resultados

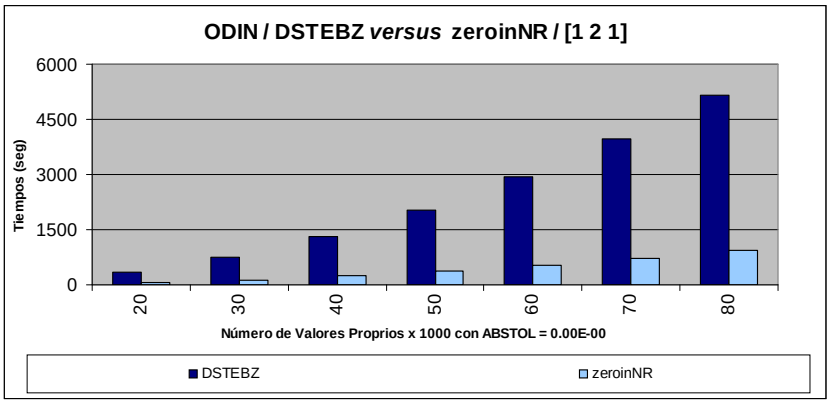

Figura 4.12: Tiempos de ejecución con matrices del tipo 1.

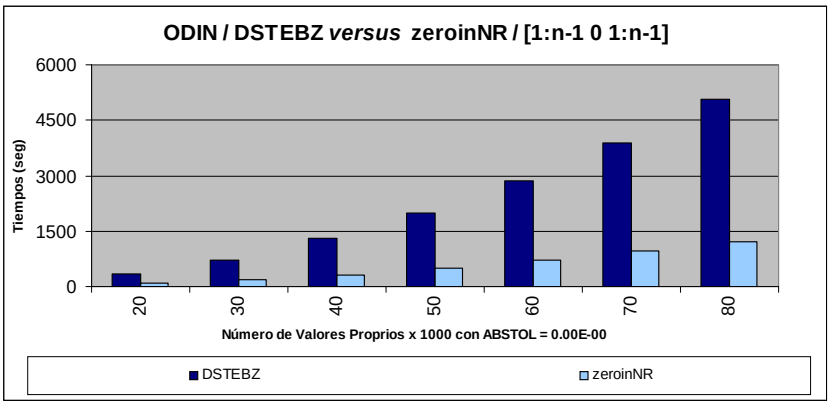

Figura 4.13: Tiempos de ejecución con matrices del tipo 3.

DSTEBZ	zeroinNR	Bis	NR
10,7461941829034	10,7461941829034	4	6
10,7461941829033	10,7461941829033	2	8
9,21067864736133	9,21067864736133	1	6
9,21067864730492	9,21067864730492	3	7
8,03994112282902	8,03994112282902	1	7
8,03994111581427	8,03994111581427	5	6
7,00395220952868	7,00395220952868	1	6
7,00395179861638	7,00395179861638	8	5
6,00023403158417	6,00023403158417	1	6
6,00021752225710	6,00021752225710	13	7
5,00024442500191	5,00024442500191	1	7
4,99978247774290	4,99978247774290	13	8
4,00435402344086	4,00435402344086	1	9
3,99604820138363	3,99604820138363	14	15
3,04309929257882	3,04309929257882	1	15
2,96105888418573	2,96105888418573	30	5
2,13020921936251	2,13020921936251	1	4
1,78932135269508	1,78932135269508	35	4
0,947534367529293	0,947534367529293	1	3
0,253805817096678	0,253805817096678	44	2
−1,12544152211998	−1,12544152211998	1	3

Tabla 4.10: Resultados del método `zeroinNR` con la matriz W_{21}^{+}.

obtenidos con la rutina `DSTEBZ` y la implementación secuencial del método `zeroinNR`, desarrollada en el marco de la tesis.

Del trabajo publicado en este capítulo, las conclusiones más importantes que se logran son:

- Se ha desarrollado un estudio exhaustivo con las rutinas de LAPACK dedicadas a la DVP de matrices tridiagonales simétricas y se han exhibido casos de prueba en los cuales las rutinas de LAPACK calculan valores propios incorrectos.

- Los experimentos computacionales fueron extensos, empleando un elevado conjunto de matrices de prueba conocidas del estado del arte.
- La implementación secuencial del método `zeroinNR` garantiza el cálculo de todos los valores propios de la matriz tridiagonal simétrica con elevada precisión.
- Los tiempos de ejecución de la implementación secuencial del método `zeroinNR` son extraordinariamente menores que los tiempos de ejecución obtenidos con la rutina `DSTEBZ`.
- Una implementación paralela del método de bisección que haga una distribución uniforme del intervalo que contiene todos los valores propios de la matriz por los distintos procesadores, puede conducir a un mal balanceo de la carga, alcanzando tiempos de ejecución elevados.

Capítulo 5

Métodos Paralelos para la DVP Tridiagonal

En este capítulo se describe el método implementado en SCALAPACK para el cálculo de los valores propios de una matriz tridiagonal simétrica, el cual es una implementación del **método de bisección con comunicaciones**.

A continuación se exponen las ideas novedosas presentadas por Ralha en [Ralha, 2006], para el progreso de una implementación paralela destinada a sistemas de memoria distribuida, en la cual los valores propios de la matriz tridiagonal simétrica son calculados con el método `zeroinNR` y empleando una metodología **sin comunicaciones**.

Como trabajo novedoso en el marco de la tesis se ha desarrollado la implementación paralela del método descrito y se ha estudiado comparativamente esa implementación con la rutina `PDSTEBZ` de SCALAPACK. Así, seguidamente se presenta el estudio comparativo entre los tiempos de ejecución obtenidos con la rutina `PDSTEBZ` y los tiempos de ejecución obtenidos con la implementación paralela desarrollada, cumpliendo con otro de los objetivos de la tesis:

- Exhibir un estudio comparativo entre una implementación paralela con

comunicaciones y una implementación paralela sin comunicaciones, para el cálculo de los valores propios de una matriz tridiagonal simétrica irreducida.

La presentación de las conclusiones logradas con la investigación desarrollada cierra el capítulo.

5.1. Implementación paralela con comunicaciones

En [Demmel *et al.*, 1995] se describen distintas implementaciones paralelas del método de bisección para el cálculo de los valores propios de una matriz tridiagonal simétrica irreducida T, donde todas las implementaciones están basadas en el empleo de la función $Count(x)$, la cual obtiene el número de términos negativos de la secuencia (4.19) y que es igual al número de valores propios de T más pequeños que x.

En aritmética exacta, la función $Count$ es una función continua, monótona creciente y, por lo tanto, en el intervalo $[\sigma_1, \sigma_2[$ hay exactamente $Count(\sigma_2) - Count(\sigma_1)$ valores propios de la matriz. Sin embargo, en sistemas computacionales, puede ocurrir que

$$\sigma_1 < \sigma_2 \qquad \text{con} \qquad FloatingCount\,(\sigma_1) > FloatingCount\,(\sigma_2) \tag{5.1}$$

donde $FloatingCount$ es la designación para la implementación computacional de la función $Count$.

En estas condiciones, la implementación computacional puede llegar a la conclusión de que en el intervalo $[\sigma_1, \sigma_2[$ hay un número negativo de valores propios de la matriz, pues $FloatingCount\,(\sigma_2) - FloatingCount\,(\sigma_1) < 0$. Para evitar este problema, en [Demmel *et al.*, 1995], los autores han sugerido el concepto de **tarea válida** definida por

$$(\alpha, \beta, n_\alpha, n_\beta, O) \tag{5.2}$$

donde:

- $[\alpha, \beta]$ es un intervalo no vacío;
- n_α y n_β son, respectivamente, los $FloatingCount$ de α y de β;

- O es el conjunto de los índices de los valores propios que se desean calcular en $[\alpha, \beta]$. El conjunto O es definido como

$$O \subseteq I_{n_\alpha}^{n_\beta} = \{n_\alpha + 1, \ldots, n_\beta\} \tag{5.3}$$

  y $I_{n_\alpha}^{n_\beta} = \varnothing$ siempre que $n_\alpha \geq n_\beta$.

Además, en [Demmel *et al.*, 1995], los autores describen la rutina `Ser_Bracket`, la cual garantiza la monotonicidad de la función $FloatingCount$, empleando las sentencias

$$\begin{aligned} mid &= inside\,(\alpha, \beta, T)\,; \\ n_{mid} &= \text{mín}\,(\text{máx}\,(FloatingCount\,(mid)\,, n_\alpha)\,, n_\beta)\,; \end{aligned} \tag{5.4}$$

y que calcula los valores propios $\lambda_{n_\alpha+1} \leq \ldots \leq \lambda_{n_\beta}$ con el método de bisección.

Hay que señalar que la segunda sentencia de (5.4) no tiene ningún efecto en el caso de que $FloatingCount$ sea monótona. En el caso contrario, garantiza que n_{mid} está entre n_α y n_β.

En [Demmel *et al.*, 1995] se considera también que un método de división (*bracket*) para el cálculo de los valores propios de la matriz tridiagonal simétrica es correcto si verifica las siguientes condiciones:

1. Todos los valores propios en $[\alpha, \beta]$ son calculados una única vez.
2. Todos los valores propios son calculados con la cota de error definida por el usuario.
3. Los valores propios calculados están expuestos según una cierta ordenación.

Para el cálculo de todos los valores propios de la matriz tridiagonal simétrica irreducida T, la primera implementación paralela propuesta en [Demmel *et al.*, 1995] es la de dividir el intervalo $[gl, gu]$ que contiene todos los valores propios de la matriz T y cuyos limites son obtenidos con el teorema de los círculos de Gerschgorin, en subintervalos de amplitud $\frac{gu-gl}{p} = h$, donde p es el número de procesadores empleados y que cada uno de los procesadores calcule los valores propios que están dentro del intervalo $[gl + ih, gl + (i+1)\,h]$, siendo i $(i = 0, \ldots, p-1)$ el identificador del procesador.

Esta metodología es empleada, por ejemplo, por Badía en su tesis doctoral [Badía, 1996] pero, tal y como refieren los autores, esta metodología no es la más adecuada para las matrices tridiagonales simétricas cuyos valores propios no están homogéneamente distribuidos: *Aunque correcto, el algoritmo Par_Alleig2 es muy sensible a la distribución de valores propios en el intervalo de Gerschgorin, y no da lugar a altas aceleraciones en los equipos masivamente paralelos cuando los valores propios no se distribuyen de manera uniforme* (traducción nuestra).

En estas condiciones, en [Demmel *et al.*, 1995] se ha propuesto una técnica alternativa y que es la misma que sigue Ralha en [Ralha, 2006], la cual está orientada a valores propios y no a intervalos con valores propios, o sea, cada uno de los p procesadores va a tener que calcular más o menos el mismo número de valores propios, en concreto, $\frac{n}{p}$ valores propios, donde por simplicidad se asume que $\frac{n}{p}$ es un número entero.

Así, el procesador i $(i = 0, \ldots, p-1)$ va a calcular los valores propios

$$\lambda_{i\left(\frac{n}{p}\right)+1} \leq \lambda_{i\left(\frac{n}{p}\right)+2} \leq \ldots \leq \lambda_{(i+1)\frac{n}{p}} \tag{5.5}$$

y, para ello, va a tener que llamar la rutina `Find_Init_Task`, la cual tiene por objetivo calcular la tarea válida

$$\left(\alpha^{(i)}, \beta^{(i)}, n_{\alpha^{(i)}}, n_{\beta^{(i)}}, I_{n_{\alpha^{(i)}}}^{n_{\beta^{(i)}}}\right) \tag{5.6}$$

en la cual el intervalo $\left[\alpha^{(i)}, \beta^{(i)}\right]$ contiene exactamente $\frac{n}{p}$ valores propios, pues $\alpha^{(i)}$ y $\beta^{(i)}$ son tales que

$$FloatingCount\left(\beta^{(i)}\right) - FloatingCount\left(\alpha^{(i)}\right) = \frac{n}{p}. \tag{5.7}$$

Una vez obtenida la tarea válida (5.6), los valores propios (5.5) son calculados haciendo llamada a la rutina `Ser_Bracket` con esa tarea válida.

En el cálculo del intervalo $\left[\alpha^{(i)}, \beta^{(i)}\right]$, cada procesador tiene que determinar el valor $\beta^{(i)}$ tal que

$$FloatingCount\left(\beta^{(i)}\right) = (i+1)\,\frac{n}{p} \tag{5.8}$$

a menos que el valor propio $\lambda_{(i+1)\frac{n}{p}}$ pertenezca a un *cluster* de valores propios y en ese caso, se asume que los valores propios $\lambda_{(i+1)\frac{n}{p}}, \ldots, \lambda_{FloatingCount\left(\beta^{(i)}\right)}$, con

$$FloatingCount\left(\beta^{(i)}\right) > (i+1)\,\frac{n}{p} \tag{5.9}$$

son valores propios de un *cluster* relativamente a la cota de error definida por el usuario.

En estas condiciones, cada uno de los p procesadores sólo tiene que calcular el valor $\beta^{(i)}$, mientras que el valor $\alpha^{(i)}$ es obtenido por comunicación a partir del procesador $i-1$, es decir, cada procesador, una vez calculado el valor $\beta^{(i)}$ envía ese valor al procesador $i+1$, el cual le asigna el valor $\alpha^{(i+1)}$. Así, todos los procesadores, con excepción del primero y del último, envían $\beta^{(i)}$ al procesador $i+1$ y reciben $\alpha^{(i)}$ del procesador $i-1$.

Sin embargo, y principalmente cuando los procesadores tienen aritméticas distintas, puede ocurrir que $\beta^{(i)} > \beta^{(i+1)}$ y, por lo tanto, para garantizar que

$$\alpha^{(i)} \leq \beta^{(i)} \qquad \text{y} \qquad n_{\alpha^{(i)}} \leq n_{\beta^{(i)}} \tag{5.10}$$

al final de la rutina `Find_Init_Task` ([Demmel *et al.*, 1995, pp. 130]), los autores han incluido las sentencias

$$\begin{aligned}
\alpha &= \text{mín}\,(\alpha,\beta)\,; \quad \beta = \text{máx}\,(\alpha,\beta)\,; \\
\alpha &= max_scan\,(\alpha)\,; \quad \beta = max_scan\,(\beta)\,; \\
n_\alpha &= \text{mín}\,(n_\alpha, n_\beta)\,; \quad n_\beta = \text{máx}\,(n_\alpha, n_\beta)\,; \\
n_\alpha &= max_scan\,(n_\alpha)\,; \quad n_\beta = max_scan\,(n_\beta)\,;
\end{aligned} \tag{5.11}$$

donde la rutina max_scan es tal que

$$x = max\_scan\,(x) \qquad \text{con} \qquad x^{(i)} = \underset{1\leq j\leq i}{\text{máx}}\left(x^{(j)}\right) \tag{5.12}$$

para $i = 1, \ldots, p-1$.

Tal y como ya fue referido anteriormente, una vez obtenida la tarea válida (5.6), los valores propios (5.5) son calculados llamando a la rutina `Ser_Bracket` y así, la rutina `PDSTEBZ`[1] de SCALAPACK calcula los valores propios de la matriz tridiagonal simétrica con el método de bisección y con comunicaciones.

[1] Para más información, consultar el Apéndice "Rutinas de LAPACK y SCALAPACK".

5.2. Implementación paralela sin comunicaciones

En [Demmel *et al.*, 1995, pp. 117], los autores han escrito: *Idealmente, nos gustaría que hubiera un algoritmo bracketing que fuese al mismo tiempo, paralelo, con un buen balanceo de la carga, desprovisto de comunicaciones, y correcto ante la no monotonicidad. Todavía no sabemos cómo lograrlo por completo; en el caso más general, cuando diferentes procesadores paralelos ni siquiera poseen el mismo formato de coma flotante, no sabemos cómo implementar un algoritmo correcto y razonablemente rápido en absoluto. Incluso cuando los formatos de coma flotante son los mismos, no sabemos cómo evitar algunas comunicaciones globales* (traducción nuestra).

Intentando resolver los retos planteados por Demmel, Dhillon y Ren, pero manteniendo los mismos objetivos, en 2006, Ralha presentó las ideas que se describen a continuación, donde la idea fundamental es, en lugar de cada procesador calcular el valor $\beta^{(i)}$ y recibir, por comunicación, el valor $\alpha^{(i)}$, cada procesador va a tener que calcular $\alpha^{(i)}$ y $\beta^{(i)}$. En ese sentido, la idea base propuesta en [Ralha, 2006] es la de suprimir todos los mecanismos de comunicación que están descritos en [Demmel *et al.*, 1995], aunque para ello se obligue a que los procesadores realicen algunos cálculos computacionales redundantes.

Así, dada una matriz tridiagonal simétrica irreducida T, cuyos valores propios verifican $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, Ralha propone en [Ralha, 2006] que todos los valores propios sean calculados una sola vez, con la cota de error definida por el usuario y además, que verifiquen $\widetilde{\lambda}_1 \leq \widetilde{\lambda}_2 \leq \ldots \leq \widetilde{\lambda}_n$, donde $\widetilde{\lambda}$ representa la aproximación al valor propio λ.

Para ello, a cada procesador i $(i = 0, \ldots, p-1)$ es asignada la tarea de calcular los $\frac{n}{p}$ valores propios de la matriz T,

$$\lambda_{i\left(\frac{n}{p}\right)+1} \leq \lambda_{i\left(\frac{n}{p}\right)+2} \leq \ldots \leq \lambda_{(i+1)\frac{n}{p}} \tag{5.13}$$

donde su primer objetivo es el de calcular los valores

$$\lambda_{min}^{(i)} = \lambda_{i\left(\frac{n}{p}\right)+1} \qquad \text{y} \qquad \lambda_{max}^{(i)} = \lambda_{(i+1)\frac{n}{p}} \tag{5.14}$$

con $\lambda_{min}^{(i)} \leq \lambda_{max}^{(i)}$ verificando

$$FloatingCount\left(\lambda_{max}^{(i)}\right) - FloatingCount\left(\lambda_{min}^{(i)}\right) = \frac{n}{p}. \tag{5.15}$$

Por lo tanto, $\lambda_{min}^{(i)}$ y $\lambda_{max}^{(i)}$ son los valores que definen el intervalo $\left[\lambda_{min}^{(i)}, \lambda_{max}^{(i)}\right]$ en el cual están contenidos todos los $\frac{n}{p}$ valores propios que el procesador i tiene que calcular.

En estas circunstancias, todos los procesadores empiezan por calcular el intervalo inicial $[gl, gu]$, que como ya fue referido, se obtiene con el teorema de los círculos de Gerschgorin y, a continuación, van en busca del intervalo $\left[\lambda_{min}^{(i)}, \lambda_{max}^{(i)}\right]$.

Como fácilmente se comprende, hasta que cada procesador consiga calcular el intervalo $\left[\lambda_{min}^{(i)}, \lambda_{max}^{(i)}\right]$, a partir del intervalo inicial $[gl, gu]$, va a hacer un conjunto de cálculos computacionales que son iguales en todos los procesadores y, por lo tanto, la idea fundamental en este método es que distintos procesadores puedan calcular el mismo valor del $FloatingCount$ para el mismo valor x, hasta que consigan llegar a un punto al que Ralha nombró **punto de separación** que permite aislar los valores propios del procesador i de los valores propios de los procesadores vecinos, con excepción del primer procesador y del último procesador.

Tal y como ocurre en [Demmel *et al.*, 1995], una vez obtenido el intervalo $\left[\lambda_{min}^{(i)}, \lambda_{max}^{(i)}\right]$, el procesador i puede definir la tarea válida

$$\left(\lambda_{min}^{(i)}, \lambda_{max}^{(i)}, i\left(\frac{n}{p}\right), (i+1)\frac{n}{p}, I_{i\left(\frac{n}{p}\right)}^{(i+1)\frac{n}{p}}\right) \tag{5.16}$$

y calcular los valores propios (5.13) empleando el método `zeroinNR`, que como ya fue descrito, empieza por aplicar el método de bisección para aislar cada valor propio en un subintervalo de $\left[\lambda_{min}^{(i)}, \lambda_{max}^{(i)}\right]$ y, a continuación, aplica el método de Newton-Raphson con la formulación propuesta por Ralha en su tesis doctoral [Ralha, 1990] para extraer el valor propio con la cota de error definida por el usuario.

Para evitar la posibilidad de $\lambda_{max}^{(i)} > \lambda_{min}^{(i+1)}$ hay que garantizar que el procesador i y el procesador $i+1$ obtienen el mismo valor del $FloatingCount$ para los mismos valores de x, hasta que lleguen al punto de separación ps en el cual

$$FloatingCount\,(ps) = i\left(\frac{n}{p}\right). \tag{5.17}$$

Esto garantiza que $\lambda_{max}^{(i)} \leq \lambda_{min}^{(i+1)}$, pues el punto de separación ps es el limite superior para el cálculo del valor propio $\lambda_{(i+1)\frac{n}{p}}$ y el limite inferior para el cálculo del valor propio $\lambda_{(i+1)\frac{n}{p}+1}$.

Sin embargo, si estos valores propios son muy cercanos, puede ocurrir que el procesador i y el procesador $i+1$ no consigan llegar al punto de separación ps y, en estas condiciones, ambos procesadores devuelven el mismo intervalo $[\alpha, \beta]$, el cual se asume que constituye una representación de más o menos $FloatingCount\,(\beta) - FloatingCount\,(\alpha)$ valores propios, que son los valores propios que pertenecen al intervalo $[\alpha, \beta]$. Esta situación puede ocurrir cuando los valores propios están dentro de *clusters* de valores propios.

Como ya fue observado, el punto base del método propuesto por Ralha es que hay que garantizar que distintos procesadores puedan calcular el mismo valor del $FloatingCount$ para los mismos valores de x, hasta el punto de separación ps.

La situación más sencilla ocurre cuando todos los procesadores tienen la misma aritmética y, por lo tanto, el sistema computacional empleado es homogéneo. En este caso, el método propuesto por Ralha en [Ralha, 2006] resuelve los retos planteados por Demmel, Dhillon y Ren, y además, cumple con las tres condiciones descritas para que el método sea correcto. Al mismo tiempo, hay que tener en cuenta que en esta situación no hace falta que la función $Count$ sea monótona.

En el caso de que el sistema computacional empleado sea heterogéneo, dos problemas inmediatos salen a la luz: el primero es que no se puede garantizar que distintos procesadores con aritméticas diferentes calculen los mismos puntos de bisección para el mismo intervalo inicial y el segundo es que para el mismo valor x, distintos procesadores pueden obtener diferentes valores del $FloatingCount$.

Para resolver el primer problema, Ralha propone la idea extraordinaria de, a la vez de cada procesador empezar sus cálculos computacionales a partir del intervalo inicial $[gl, gu]$, ampliar un poco ese intervalo de tal manera que los limites del intervalo sean potencias de 2. Además, en [Ralha, 2006], el autor propone que haya una separación entre los valores propios positivos y los valores propios negativos y, por tanto, que el intervalo inicial a partir del cual cada procesador va a calcular sus valores propios sea el intervalo $\left[0, 2^j\right]$ para los valores propios positivos y el intervalo $\left[-2^l, 0\right]$ para los valores propios negativos, siendo el valor del $FloatingCount$ de cero el valor que permite identificar cuantos

valores propios positivos y cuantos valores propios negativos tiene la matriz. Está claro que en esta situación, el autor asume una representación binaria en coma flotante en todos los procesadores de sistema computacional.

En cuanto al segundo problema, tal y como describe Ralha, no tiene una solución inmediata, dado que dos procesadores vecinos pueden calcular diferentes valores del $FloatingCount$ para el mismo valor x. Por ejemplo, puede ocurrir

$$FloatingCount^{(i)}(x) < i\left(\frac{n}{p}\right) \quad \text{y} \quad FloatingCount^{(i+1)}(x) \geq i\left(\frac{n}{p}\right) + 1 \tag{5.18}$$

y, por tanto, $\widetilde{\lambda}_{i\left(\frac{n}{p}\right)} > \widetilde{\lambda}_{i\left(\frac{n}{p}\right)+1}$.

En estas condiciones, aunque los valores propios sean calculados una sola vez, y cumpliendo con las dos primeras condiciones para que el método sea correcto, no están ordenados de menor a mayor. Sin embargo, y tal como cuestiona Ralha, "*¿en sistemas heterogéneos es fundamental que los valores propios sean calculados de forma ordenada?*". Una vez que todos los procesadores calculan todos los valores propios que tienen que calcular una única vez, al final de la ejecución puede haber un procesador que almacena todos los valores propios calculados y se encarga de hacer su ordenación, por ejemplo, de menor a mayor.

El método propuesto en [Ralha, 2006], garantiza que en sistemas computacionales homogéneos los valores propios de la matriz tridiagonal simétrica irreducida son calculados una sola vez, con la precisión numérica definida por la cota de error impuesta por el usuario y además, están ordenados. En sistemas computacionales heterogéneos solamente no es posible garantizar el cálculo ordenado de los valores propios. Sin embargo, en ambos sistemas computacionales, la implementación paralela hace un buen balanceo de la carga, pues cada procesador calcula $\frac{n}{p}$ valores propios de la matriz y sin comunicaciones.

La única desventaja del método está en la redundancia que hay en las computaciones iniciales, dado que todos los procesadores tienen que calcular el mismo intervalo inicial, a partir del cual empiezan las iteraciones del método de bisección. Sin embargo, la influencia negativa de estas computaciones no es problemática ya que el número de iteraciones que cada procesador tiene que calcular hasta llegar a su punto de separación ps es pequeño cuando se compara con el número total de iteraciones empleadas.

En el marco de la tesis y como aportación novedosa, se ha desarrollado la implementación paralela del método descrito. Al mismo tiempo, como todos los sistemas computacionales utilizados en la investigación desarrollada son sistemas homogéneos, se han comprobado las afirmaciones anteriores respecto a este tipo de sistemas computacionales.

Queda abierto y como línea de investigación futura, hacer pruebas en sistemas computacionales heterogéneos, en los que, además, aún se pueden explorar las ideas propuestas por Ralha en la sección 7 de [Ralha, 2009], para el desarrollo de implementaciones secuenciales y paralelas del método de bisección empleando el paradigma de la precisión mixta y comparar los resultados según la perspectiva de [Langou *et al.*, 2006].

5.3. Resultados experimentales

A continuación vamos a exponer los tiempos de ejecución obtenidos por la implementación paralela desarrollada para el método descrito en [Ralha, 2006], la cual no necesita comunicaciones en el cálculo de todos los valores propios de una matriz tridiagonal simétrica irreducida, y los tiempos de ejecución de la rutina `PDSTEBZ` de SCALAPACK, que implementa el método de bisección descrito en [Demmel *et al.*, 1995] con comunicaciones.

Para el estudio comparativo entre los tiempos de ejecución obtenidos, se ha desarrollado un elevado número de experimentos computacionales con las matrices de prueba descritas en la Tabla 4.2.

Tras un análisis exhaustivo a los resultados obtenidos y una vez que ellos permiten lograr las mismas conclusiones para todos los casos de matrices de prueba utilizados, en la sección se va a presentar solamente los resultados obtenidos con las matrices del tipo 1 con $a = 2$ y $b = 1$. Sin embargo, hay que tener en cuenta que pueden existir matrices tridiagonales simétricas irreducidas con distribuciones espectrales distintas de las exhibidas por las matrices de prueba descritas en la Tabla 4.2 y para las cuales nuestras conclusiones tienen que ser reformuladas o incluso, pueden no ser verdaderas.

Para los experimentos computacionales se han empleado matrices de dimensión 20000, 30000, 40000, 50000 y 60000, y las configuraciones 1×1, 1×2, 1×4, 1×8, 1×16 y 1×32 para las mallas de los procesadores, pues en su implementación, la rutina `PDSTEBZ` redefine la configuración de la malla

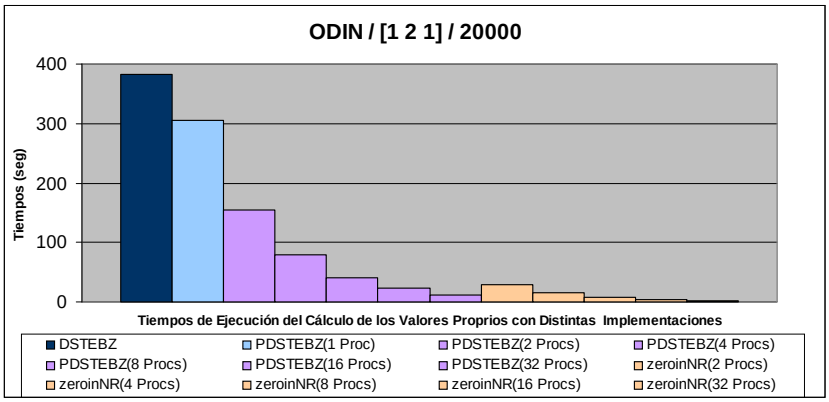

Figura 5.1: Tiempos de ejecución con la matriz de dimensión 20000.

de procesadores definida por el usuario en una malla de procesadores del tipo $1 \times p$.

Hay que señalar que en la rutina `PDSTEBZ` y en la implementación paralela desarrollada, todos los procesadores empleados almacenan en sus memorias locales los dos vectores que definen la matriz tridiagonal simétrica y aprovechan uno de esos vectores para almacenaren los valores propios calculados. En la rutina `PDSTEBZ` no hace falta hacer una distribución cíclica de los datos, tal como se ha hecho, por ejemplo, en la rutina `PDGEBRD`, para la cual se ha estudiado cual era la dimensión del bloque matricial y cual era la configuración de la malla de procesadores que permitía lograr las mejores prestaciones de la rutina.

Las figuras, Figura 5.1 – Figura 5.5, ilustran los tiempos de ejecución obtenidos con la rutina `DSTEBZ` de LAPACK, con la rutina `PDSTEBZ` de SCALAPACK empleando 1, 2, 4, 8, 16 y 32 procesadores, y con la implementación paralela del método descrito en [Ralha, 2006].

En todas las figuras se puede observar que la rutina `PDSTEBZ` ejecutada en un solo procesador obtiene mejores prestaciones que la correspondiente rutina de LAPACK. Debido a eso y tal como fue descrito en la sección 1.5, la aceleración

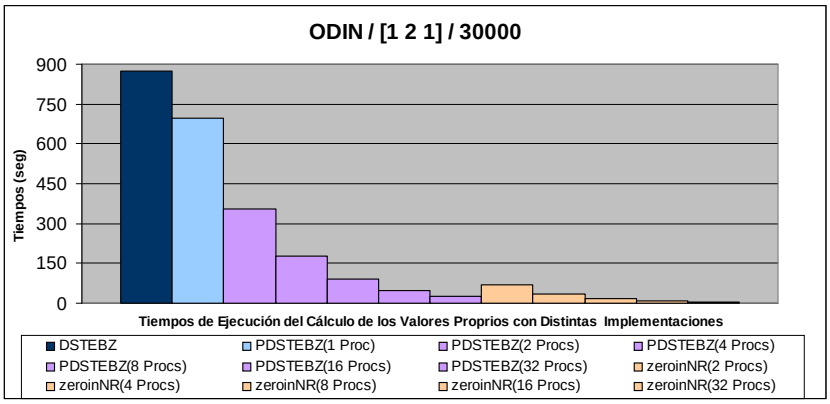

Figura 5.2: Tiempos de ejecución con la matriz de dimensión 30000.

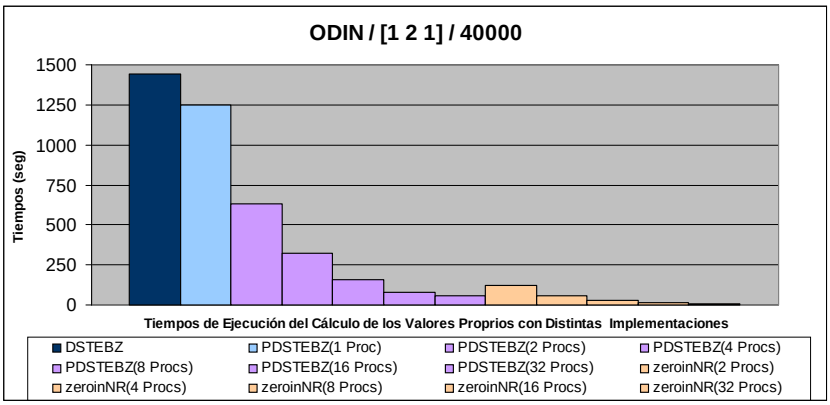

Figura 5.3: Tiempos de ejecución con la matriz de dimensión 40000.

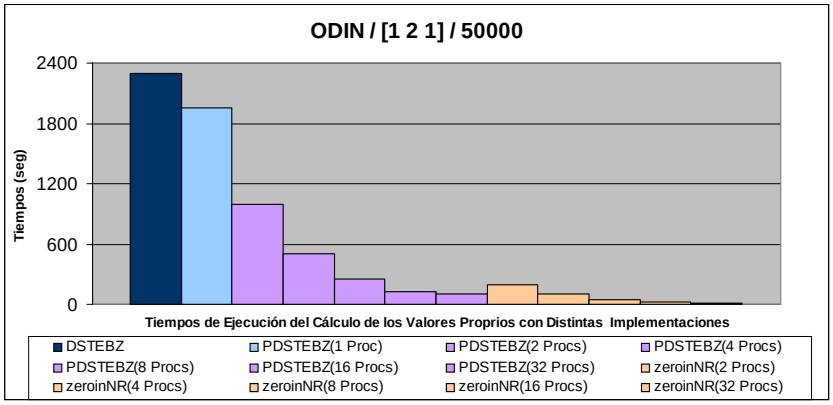

Figura 5.4: Tiempos de ejecución con la matriz de dimensión 50000.

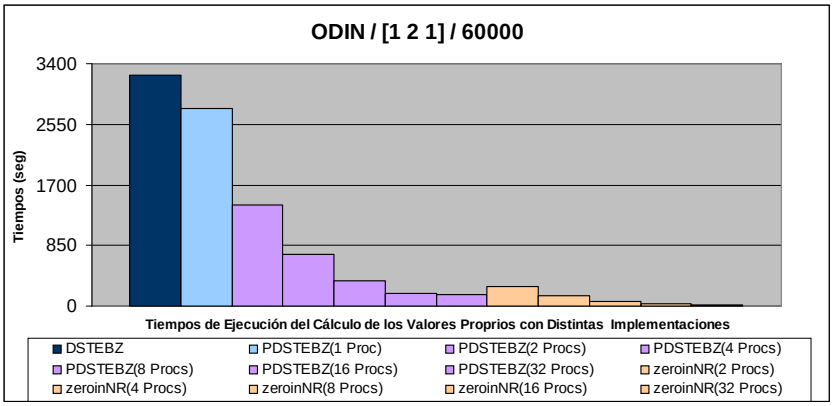

Figura 5.5: Tiempos de ejecución con la matriz de dimensión 60000.

de ejecución es valorada según (1.8) y la eficiencia es valorada según (1.11).

Debido al hecho de que la implementación secuencial del método `zeroinNR` obtiene tiempos de ejecución extraordinariamente más pequeños que la rutina `DSTEBZ` (Figura 4.12), se observa que la implementación paralela del método descrito en [Ralha, 2006], ejecutada con 2 procesadores obtiene tiempos de ejecución que son sencillamente menores que los tiempos de ejecución obtenidos por la rutina `PDSTEBZ` ejecutada con 8 procesadores. Además, en todas las figuras se observa que la implementación paralela permite calcular todos los valores propios de la matriz tridiagonal simétrica irreducida utilizando menos recursos computacionales que con la rutina de SCALAPACK.

La Figura 5.6 ilustra los valores del *speed-up* de la rutina `PDSTEBZ`. Como se observa, hasta 16 procesadores, los valores obtenidos son los valores ideales. Sin embargo, para 32 procesadores, el mayor valor del *speed-up* que se obtiene es de 27.

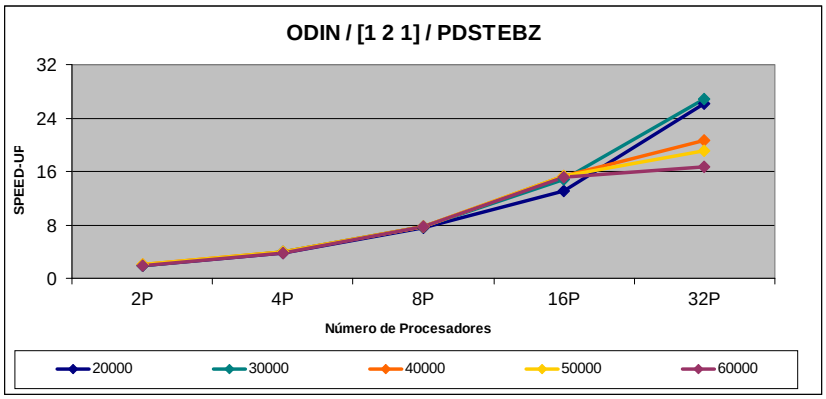

Figura 5.6: Valores del *speed-up* de `PDSTEBZ`.

En cambio, la implementación paralela del método descrito en [Ralha, 2006] y desarrollada en el marco de la tesis, obtiene valores del *speed-up* que son sencillamente los mismos para todas las dimensiones de las matrices utilizadas y además, hasta 16 procesadores, los valores obtenidos son los valores ideales

y para 32 procesadores, el valor obtenido es de 30. La Figura 5.7 ilustra esos valores del *speed-up* y hay que tener en cuenta que aunque cada uno de los p procesadores tenga que calcular $\frac{n}{p}$ valores propios de la matriz tridiagonal simétrica irreducida, el balanceo de la carga no es el ideal debido al hecho de que cada procesador va a tener que aplicar un número distinto de iteraciones en el cálculo de sus valores propios.

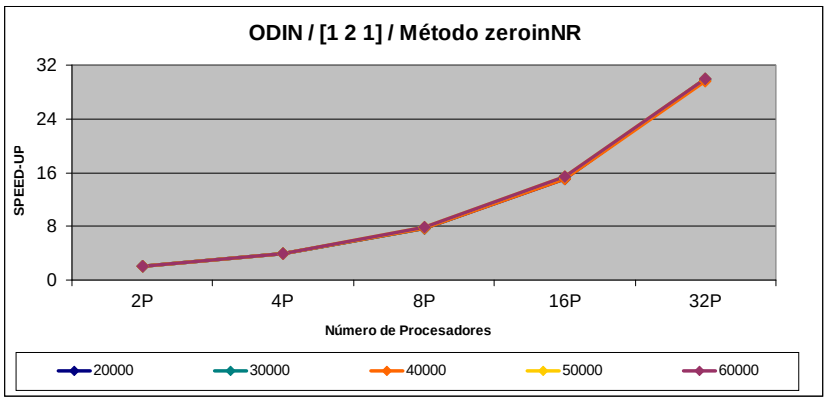

Figura 5.7: Valores del *speed-up* del método `zeroinNR`.

La Tabla 5.1 presenta el número total de iteraciones que cada uno de los 8 procesadores ha tenido que aplicar en el cálculo de sus $\frac{n}{p}$ valores propios. Como se observa, cada procesador aplica un número distinto de iteraciones y, por lo tanto, aunque cada procesador tenga que calcular el mismo número de valores propios, el balanceo de la carga no es ideal y eso tiene importancia en los tiempos de ejecución obtenidos y en los valores del *speed-up*.

Nº Procs	Procs	20000	30000	40000	50000	60000
1	Proc0	124843	188264	248326	315015	374753
2	Proc0	62616	94427	124550	158004	188021
	Proc1	62227	93837	123776	157011	186732
	total	124843	188264	248326	315015	374753
4	Proc0	31476	47103	62586	77828	93825
	Proc1	31149	47339	61979	80191	94211
	Proc2	31069	47188	61808	79939	93859
	Proc3	31167	46664	61983	77087	92888
	total	124861	188294	248356	315045	374783
8	Proc0	15711	23417	31247	38762	46669
	Proc1	15775	23702	31355	39082	47173
	Proc2	15913	23174	31648	39945	46136
	Proc3	15250	24179	30345	40260	48089
	Proc4	15224	24137	30307	40203	47983
	Proc5	15859	23065	31515	39750	45890
	Proc6	15662	23531	31094	38761	46745
	Proc7	15515	23149	30905	38342	46160
	total	124909	188354	248416	315105	374845

Tabla 5.1: Número total de iteraciones por procesador.

La Tabla 5.2 presenta el número de iteraciones del método de bisección (`Bis`) y el número de iteraciones del método de Newton-Raphson (`NR`) que, por ejemplo, cada uno de los 8 procesadores empleados en la ejecución ha tenido que aplicar en el cálculo de sus valores propios y para las matrices de mayor dimensión.

Como se observa, el número de iteraciones empleadas con el método de Newton-Rapshon es siempre mayor que el doble de iteraciones empleadas con el método de bisección y esto tiene su efecto en las prestaciones obtenidas con el método `zeroinNR`, dado que el método de Newton-Raphson tiene convergencia cuadrática, mientras que el método de bisección tiene convergencia lineal. Hay que tener en cuenta que una iteración del método de Newton-Rapshon es equivalente a ocho iteraciones del método de bisección, tal y como se describe

en [Parlett y Nour-Omid, 1985] y, por lo tanto, tanto en la rutina de LAPACK como en la rutina de SCALAPACK, el número de iteraciones empleadas en el cálculo de todos los valores propios de la matriz tridiagonal es muy grande, ya que en estas rutinas solo se emplea el método de bisección.

Procs	30000	40000	50000	60000
Proc0	7507 + 15910	10007 + 21240	12507 + 26255	15008 + 31661
Proc1	7515 + 16187	10015 + 21340	12515 + 26567	15015 + 32158
Proc2	7517 + 15657	10017 + 21631	12517 + 27428	15017 + 31119
Proc3	7506 + 16673	10006 + 20339	12506 + 27754	15006 + 33083
Proc4	7506 + 16631	10006 + 20301	12506 + 27697	15006 + 32977
Proc5	7517 + 15548	10017 + 21498	12517 + 27233	15017 + 30873
Proc6	7515 + 16016	10015 + 21079	12515 + 26246	15015 + 31730
Proc7	7507 + 15642	10007 + 20898	12507 + 25835	15008 + 31152

Tabla 5.2: Número de iteraciones `Bis+NR` por procesador.

5.4. Conclusiones

En este capítulo se ha empezado por describir la implementación paralela del método de bisección existente en SCALAPACK, para el cálculo de todos los valores propios de una matriz tridiagonal simétrica irreducida y la cual emplea comunicaciones.

Seguidamente se ha descrito el método propuesto por Ralha en [Ralha, 2006], para el desarrollo de una implementación paralela del método `zeroinNR`, destinada a sistemas de memoria distribuida y cuya implementación no necesita comunicaciones.

Tal y como fue descrito por Ralha, con la implementación paralela desarrollada en el marco de la tesis y que es una aportación novedosa, debido al hecho de haber sido ejecutada en sistemas computacionales homogéneos, ha dado la posibilidad de comprobar todas las condiciones definidas por Demmel, Dhillon y Ren en [Demmel *et al.*, 1995], para la existencia de una correcta implementación de un método del tipo bisección: todos los valores propios son

calculados una única vez, con la precisión numérica impuesta por la cota de error definida por el usuario y están expuestos según una cierta ordenación.

De los tiempos de ejecución obtenidos y resultantes de un vasto conjunto de experimentos computacionales, las conclusiones que la investigación desarrollada permite lograr son:

- La rutina `PDSTEBZ` de SCALAPACK, ejecutada en un solo procesador, obtiene tiempos de ejecución que son significativamente menores que los tiempos de ejecución de la correspondiente rutina secuencial de LAPACK.

- La rutina `PDSTEBZ` presenta valores del *speed-up* que son los valores ideales hasta 16 procesadores, bajando al valor de 17 cuando se emplean 32 procesadores con matrices de mayor dimensión. En cambio, para matrices de menor dimensión, ese valor es 27.

- La implementación paralela desarrollada según las ideas expuesta en [Ralha, 2006] y adaptadas al método `zeroinNR`, permite lograr tiempos de ejecución que son extraordinariamente menores que la rutina `PDSTEBZ`, pues con solo 2 procesadores se han obtenido tiempos de ejecución que están un poco por debajo de los tiempos de ejecución de la rutina de SCALAPACK con 8 procesadores.

- La implementación paralela desarrollada permite calcular todos los valores propios de la matriz tridiagonal simétrica irreducida con menos recursos que la rutina `PDSTEBZ`.

- Aunque todos los procesadores tengan que calcular el mismo número de valores propios de la matriz tridiagonal simétrica irreducida, se ha comprobado que el número de iteraciones empleadas es distinto de procesador a procesador y, por lo tanto, no hay un balanceo de la carga ideal. Sin embargo, cada procesador calcula su conjunto de $\frac{n}{p}$ valores propios de manera independiente y los tiempos de ejecución permiten obtener valores del *speed-up* que hasta 16 procesadores son los valores ideales, bajando a 30 cuando se emplean 32 procesadores. Además, el comportamiento del *speed-up* es semejante para todas las dimensiones de las matrices de prueba expuestas anteriormente.

- La implementación paralela del método descrito en [Ralha, 2006] es muy competitiva respecto a la rutina de SCALAPACK, de la misma manera que la implementación secuencial del método `zeroinNR` lo era respecto a la rutina `DSTEBZ` de LAPACK.
- Mientras que el método descrito en [Demmel *et al.*, 1995] está basado en la división equitativa del intervalo inicial por todos los procesadores, el método propuesto en [Ralha, 2006] sigue una idea alternativa y que está basada en la división equitativa del número de valores propios por todos los procesadores. Se ha comprobado que la metodología propuesta por Demmel, Dhillon y Ren no es la más adecuada para las matrices tridiagonales simétricas cuyos valores propios no están homogéneamente distribuidos, mientras que la metodología propuesta por Ralha sí que lo es. Sin embargo, pueden existir matrices tridiagonales simétricas irreducidas con distribuciones espectrales distintas de las exhibidas por las matrices de prueba y para las cuales esta conclusión tenga que ser reformulada o incluso pueda no ser verdadera.
- La implementación paralela desarrollada tiene la desventaja en la redundancia que ocurre en las computaciones iniciales ya que todos los procesadores tienen que calcular el mismo intervalo inicial, a partir del cual empiezan las iteraciones del método de bisección, hasta que lleguen a su correspondiente punto de separación. Sin embargo, la influencia negativa de estas computaciones no es problemática, pues el número de iteraciones que cada procesador tiene que calcular hasta llegar a su punto de separación es pequeño, cuando se compara con el número total de iteraciones utilizadas en el cálculo de sus $\frac{n}{p}$ valores propios.

Como ya fue referido anteriormente, una línea de investigación futura es la posibilidad de hacer pruebas en sistemas computacionales heterogéneos y comprobar si la implementación paralela del método descrito en [Ralha, 2006] verifica las ideas expuestas por Ralha para este tipo de sistemas computacionales. Además, se pueden explorar también las ideas propuestas en la sección 7 de [Ralha, 2009], para el desarrollo de implementaciones secuenciales y paralelas del método de bisección, empleando el paradigma de la precisión mixta y comparar los tiempos de ejecución obtenidos según la perspectiva presentada en [Langou *et al.*, 2006].

Esta hoja ha sido dejada en blanco de forma intencionada.

Capítulo **6**

Teoría de Control y Reducción de Modelos

EN este capítulo están descritos algunos conceptos de la **teoría de control** y en particular, de la **reducción de modelos de sistemas lineales**.

El capítulo empieza con una introducción respecto a los **sistemas lineales de control** y sus propiedades. A continuación describe el planteamiento de la reducción de modelos y su importancia en la teoría de control. Después se clasifican los métodos numéricos empleados en la reducción de modelos de sistemas lineales y en particular, los métodos numéricos basados en las **realizaciones balanceadas**.

Seguidamente se describe de manera resumida el método expuesto en [Laub *et al.*, 1987], el cual está basado en la diagonalización simultánea de los **Gramianos de controlabilidad** y **de observabilidad**, y donde hay una etapa en la cual es necesario calcular la DVS del producto de los factores de Cholesky de los Gramianos de controlabilidad y de observabilidad. Es en este punto que en el marco de la tesis se ha desarrollado un estudio respecto al método de Golub-Sølna-van Dooren, que va a ser redactado en el capítulo 7.

La presentación de algunos casos de prueba de sistemas lineales de control que fueron resueltos mediante la llamada a la rutina `AB09AD` de la librería SLICOT [webSLICOT, 2014], dedicada a la reducción de modelos de sistemas lineales de control, cierra el capítulo.

Hay que señalar que en este capítulo no se hace una introducción sobre el estado del arte de la Teoría de Control y de la Reducción de Modelos, pues nuestro principal enfoque son los métodos numéricos del Álgebra Lineal y la Teoría de Control es solamente el área de aplicación elegida.

6.1. Conceptos de sistemas lineales de control

Un **sistema lineal de control**, continuo e invariante en el tiempo, es un sistema lineal con la siguiente representación en el **modelo de espacio de estados**

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \qquad x(0) = x_0 \\ y(t) &= Cx(t) \end{aligned} \tag{6.1}$$

donde $x \in \mathbb{R}^n$ es el **vector de estados**, $u \in \mathbb{R}^m$ es el **vector de entradas** o **controles**, $y \in \mathbb{R}^p$ es el **vector de salidas**, $A \in \mathbb{R}^{n\times n}$ es la **matriz de estados**, $B \in \mathbb{R}^{n\times m}$ es la **matriz de entradas** o **controles** y $C \in \mathbb{R}^{p\times n}$ es la **matriz de salidas** [Petkov *et al.*, 1991].

Las matrices anteriores son constantes de números reales independientes del tiempo, el **orden** del sistema lineal es definido por n, donde normalmente se cumple $m \leq n$ y $p \leq n$, y x_0 es el **estado inicial** del sistema lineal.

Habitualmente, la representación del sistema lineal de control invariante en el tiempo en el modelo de espacio de estados no es única. En realidad, **un sistema lineal de control tiene distintas representaciones**. Para ello, se puede obtener un sistema lineal de control, **algebraicamente equivalente** definido por

$$\begin{aligned} \dot{\widetilde{x}}(t) &= \widetilde{A}\widetilde{x}(t) + \widetilde{B}u(t) \qquad \widetilde{x}(0) = \widetilde{x}_0 \\ y(t) &= \widetilde{C}\widetilde{x}(t) \end{aligned} \tag{6.2}$$

donde

$$\widetilde{A} = S^{-1}AS, \qquad \widetilde{B} = S^{-1}B \qquad \text{y} \qquad \widetilde{C} = CS \tag{6.3}$$

mediante la transformación no singular

$$x(t) = S\widetilde{x}(t). \tag{6.4}$$

Se puede asumir que el modelo (6.1) se ha obtenido mediante un procedimiento de linearización de un sistema no lineal, o bien de un sistema lineal distribuido mediante técnicas de aproximación, como lo son, por ejemplo, la técnica de Galerkin o la técnica de Padé [Barnett, 1985], [Fortuna *et al.*, 1992].

En general, los sistemas lineales de control invariantes en el tiempo suelen ser sistemas de gran dimensión, los cuales poseen un elevado número de variables de estado [Skelton, 1988]. Debido a eso, es del todo necesario buscar modelos más simples que aproximen al máximo el comportamiento del sistema original [Fortuna *et al.*, 1992].

De esta manera, dado el sistema lineal de control (6.1), se puede encontrar un **modelo de menor orden**, con un comportamiento similar, el cual tendrá la representación

$$\begin{aligned} \dot{x_r}(t) &= A_r x_r(t) + B_r u(t) \qquad x_r(0) = \overline{x}_0 \\ y_r(t) &= C_r x_r(t) \end{aligned} \tag{6.5}$$

donde $x_r \in \mathbb{R}^r$, $y_r \in \mathbb{R}^p$, $A_r \in \mathbb{R}^{r\times r}$, $B_r \in \mathbb{R}^{r\times m}$ y $C_r \in \mathbb{R}^{p\times r}$, con $r \ll n$ [Petkov *et al.*, 1991].

Bajo ciertas condiciones, el vector de salidas y_r es una buena aproximación al vector de salidas y del sistema original, para todos los vectores de entradas u. Al proceso de obtener este modelo reducido, a partir del modelo original, se denomina **reducción de modelos de sistemas lineales de control** [Fortuna *et al.*, 1992].

Cabe señalar que la reducción de modelos suele ser la misma para **sistemas lineales en tiempo continuo** y para **sistemas lineales en tiempo discreto**, pues las conclusiones que se pueden referir para el caso del tiempo continuo se pueden trasladar al caso del tiempo discreto.

No hay un esquema universal para la reducción de modelos. Cuando se obtiene un modelo reducido, éste es más simple, pero a la vez, más inexacto. Hay que tener en cuenta que siempre va a existir un compromiso entre la precisión del modelo obtenido y su simplicidad.

Según [Fortuna *et al.*, 1992], las principales razones para obtener sistemas lineales de control de orden más reducido son:

- Simplifica la comprensión del sistema.
- Reduce el coste computacional en los problemas de simulación.
- Permite el diseño de controladores numéricamente más eficientes con un menor esfuerzo.
- Define leyes de control más simples.

Habitualmente, el problema de control de un sistema lineal consiste en determinar las entradas adecuadas que provoquen en las salidas el comportamiento deseado. Esta tarea es desarrollada por el **controlador del sistema lineal**.

Si para calcular el valor de las entradas, el controlador del sistema lineal dispone únicamente de una función que sólo depende del tiempo, entonces se dice que el controlador del sistema lineal hace un **control en bucle abierto**. En cambio, en los sistemas lineales que están sometidos a perturbaciones y, por tanto, en los sistemas lineales donde el controlador tiene que observar las salidas para ver como se van cambiando según las entradas, se dice que el controlador del sistema lineal hace un **control en bucle cerrado**.

6.2. Propiedades de los sistemas lineales

En esta sección se describen algunas de las propiedades más importantes de los sistemas lineales de control: **estabilidad**, **controlabilidad**, **estabilizabilidad**, **observabilidad** y **detectabilidad**.

6.2.1. Estabilidad

Un sistema lineal de control, continuo e invariante en el tiempo, definido por

$$\dot{x}(t) = Ax(t) \qquad x(0) = x_0 \tag{6.6}$$

es **asintóticamente estable**, si x tiende a cero cuando t tiende a infinito, cualquiera que sea el estado inicial x_0,

$$\lim_{t \longrightarrow \infty} x(t) = 0 \qquad \forall x_0. \tag{6.7}$$

Se dice que el sistema lineal es **estable**, si para cada valor de x_0 existe una constante $c < \infty$, tal que $\|x(t)\| < c$, cuando t tiende a infinito,

$$\forall x_0 \in \mathbb{R}^n, \exists c \in \mathbb{R}^+, \text{ con } c < \infty : \qquad \lim_{t \longrightarrow \infty} \|x(t)\| < c. \tag{6.8}$$

En cambio, si existe un estado inicial x_0 tal que $\|x(t)\|$ tiende a infinito, cuando t tiende a infinito,

$$\exists x_0 \in \mathbb{R}^n : \qquad \lim_{t \longrightarrow \infty} \|x(t)\| = \infty \tag{6.9}$$

entonces, se dice que el sistema lineal es **inestable**.

La propiedad de la **estabilidad asintótica** es fundamental para el correcto funcionamiento del sistema lineal descrito por (6.1), pues garantiza que **entradas** u **acotadas proporcionan salidas** y **acotadas**.

Para estudiar la estabilidad de un sistema lineal de control importa analizar si la matriz de estados del sistema lineal cumple con los teoremas que se describen a continuación.

Teorema 14 *Un sistema lineal de control de tiempo continuo (6.6) es asintóticamente estable si y sólo si todos los valores propios λ_k de A tienen parte real negativa,*

$$\mathrm{Re}(\lambda_k) < 0 \qquad \forall \lambda_k \in \sigma(A), \quad k = 1, \ldots, n. \tag{6.10}$$

DEMOSTRACIÓN: La demostración del teorema está en la sección 2.3 de [Petkov *et al.*, 1991]. ■

En el caso de que la matriz A verifique el teorema anterior, se dice que la matriz A es una **matriz estable**.

Para sistemas lineales de control de tiempo discreto se aplica el siguiente teorema.

Teorema 15 *Un sistema lineal de control de tiempo discreto es asintóticamente estable si y sólo si todos los valores propios λ_k de A tienen módulo menor que uno,*

$$|\lambda_k| < 1 \qquad \forall \lambda_k \in \sigma(A), \quad k = 1, \ldots, n. \tag{6.11}$$

DEMOSTRACIÓN: La demostración del teorema está en la sección 2.3 de [Petkov *et al.*, 1991]. ■

En el caso de que la matriz A verifique el teorema anterior, se dice que la matriz A es una **matriz convergente**.

6.2.2. Controlabilidad y estabilizabilidad

La **controlabilidad** y la **estabilizabilidad** son dos propiedades de los sistemas lineales de control y que relacionan las entradas con las salidas.

Un sistema lineal de control, continuo y definido por

$$\dot{x}(t) = Ax(t) + Bu(t) \qquad x(0) = x_0 \tag{6.12}$$

es (**completamente**) **controlable**, si para cualquier estado inicial x_0 y para cualquier estado final x_f, existe un tiempo finito t_f y un control u, con $0 \leq t \leq t_f$, tal que $x(t_f) = x_f$.

La propiedad de la controlabilidad de un sistema lineal de control supone la **posibilidad de llevarlo de un estado inicial cualquiera, a un estado final cualquiera, en tiempo finito, mediante una adecuada elección del control**. Para sistemas lineales de control, invariantes en el tiempo, se suele asumir que el estado final es el vector nulo.

Si el sistema lineal de control (6.12) es controlable, entonces se dice que el par (A, B) es **controlable**.

Para estudiar la controlabilidad de un sistema lineal de control se puede aplicar el siguiente teorema, debido a Kalman.

Teorema 16 (Teorema de Kalman) *Un sistema lineal de control (6.12) es completamente controlable si y sólo si la* ***matriz de controlabilidad*** *de orden $n \times nm$*

$$P = \left[B, AB, A^2B, \ldots, A^{n-1}B\right] \tag{6.13}$$

tiene rango n (rango completo).

Si el rango de la matriz B es l, entonces esta condición se reduce a

$$rango\,(P) = rango\left(\left[B, AB, A^2B, \ldots, A^{l-1}B\right]\right) = n. \tag{6.14}$$

DEMOSTRACIÓN: La demostración del teorema está en el capítulo 2 de [Kalman *et al.*, 1969]. ■

Además, el sistema lineal de control (6.12) es **estabilizable**, si existe un control en lazo cerrado

$$u\,(t) = -Kx\,(t) + \overline{u}\,(t) \tag{6.15}$$

con $K \in \mathbb{R}^{m\times n}$, que transforma el sistema en

$$\dot{x}\,(t) = (A - BK)\,x\,(t) + B\overline{u}\,(t) \qquad\qquad x\,(0) = x_0 \tag{6.16}$$

tal que $A - BK$ es una **matriz estable**.

Hay que señalar que si el sistema lineal de control (6.12) es controlable, entonces es estabilizable.

6.2.3. Observabilidad y detectabilidad

La **observabilidad** y la **detectabilidad** son dos propiedades de los sistemas lineales de control y que relacionan los estados con las salidas, en ausencia de controles.

Un sistema lineal de control, continuo y definido por

$$\begin{aligned} \dot{x}\,(t) &= Ax\,(t) \qquad\qquad x\,(0) = x_0 \\ y\,(t) &= Cx\,(t) \end{aligned} \tag{6.17}$$

es (**completamente**) **observable**, si para cualquier estado inicial x_0, existe un tiempo finito t_f para el cual el conocimiento de y, con $0 \leq t \leq t_f$, es suficiente para determinar x_0.

La propiedad de la observabilidad de un sistema lineal de control supone la **posibilidad de conocer el estado inicial del sistema, con sólo conocer sus salidas**.

Si el sistema lineal de control (6.17) es observable, entonces se dice que el par (A, C) es **observable**.

Para estudiar la observabilidad de un sistema lineal de control se puede aplicar el teorema descrito a continuación.

Teorema 17 *Un sistema lineal de control (6.17) es completamente observable si y sólo si la* ***matriz de observabilidad*** *de orden* $pn \times n$

$$Q = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{6.18}$$

tiene rango n *(rango completo).*

DEMOSTRACIÓN: La demostración del teorema está en el capítulo 2 de [Kalman *et al.*, 1969]. ■

De manera semejante, el sistema lineal de control (6.17) es **detectable**, si existe una matriz $K \in \mathbb{R}^{n \times p}$, tal que $A - KC$ es una **matriz estable**. Además, si el sistema lineal de control (6.17) es observable, entonces es detectable.

La controlabilidad y la observabilidad, de la misma manera que la estabilizabilidad y la detectabilidad, son **propiedades duales de un sistema lineal de control**, en el sentido de que **el par** (A, B) **es controlable (estabilizable) si y sólo si el par** (A^t, B^t) **es observable (detectable)**.

6.3. Reducción de modelos

En general, un sistema dinámico complejo suele ser un sistema no lineal, distribuido y variable en el tiempo. Por tanto, uno de los requisitos iniciales es transformar el sistema lineal de control inicial en un sistema lineal e invariante en el tiempo.

En este planteamiento, se suelen aplicar procedimientos de aproximación como son, por ejemplo, los de Galerkin o los de Padé [Barnett, 1985], [Fortuna *et al.*, 1992], pero el problema es que los sistemas lineales de control invariantes en el tiempo suelen ser sistemas lineales de gran dimensión y suelen poseer un gran número de variables de estado [Skelton, 1988]. Por ese motivo, es del todo necesario buscar modelos más simples que aproximen al máximo el comportamiento del sistema lineal original. A esta tarea de obtener este nuevo modelo, a partir del modelo original, se denomina **reducción de**

modelos de sistemas lineales de control y al modelo obtenido se denomina **modelo reducido** o **modelo de orden reducido** [Fortuna *et al.*, 1992].

Las técnicas de reducción de modelos suelen ser las mismas tanto para sistemas lineales de control en tiempo continuo como para sistemas lineales de control en tiempo discreto. Además, la reducción de modelos puede ocurrir tanto en el **dominio del tiempo**, como en el **dominio de la frecuencia**. Nuestro enfoque va dirigido a la reducción de modelos en el dominio del tiempo.

Por tanto, la reducción de modelos de un sistema lineal consiste en determinar las entradas adecuadas que provoquen en las salidas el comportamiento deseado y, por lo tanto, está claro que cuando se obtiene un sistema lineal de control más reducido, éste es más simple, pero a la vez, más inexacto.

Bajo estas condiciones, el diseñador tiene que conocer muy bien el impacto que la reducción puede provocar sobre el comportamiento del sistema lineal de control, para poder evaluar el tipo y la cantidad de reducción posible, en cada caso, atendiendo a los límites del error que se quieren mantener y permitir. En cualquier caso, siempre existirá un compromiso entre la precisión del modelo obtenido y su simplicidad.

6.3.1. Métodos para la reducción de modelos

Fortuna *et al.* proponen una clasificación de los métodos numéricos utilizados para la reducción de modelos, basada en el ámbito de aplicación o dominio donde intervienen. Así, hay métodos numéricos para la reducción de modelos que actúan en el dominio del tiempo; hay métodos numéricos que actúan en el dominio de la frecuencia y hay métodos numéricos que actúan en ambos dominios [Fortuna *et al.*, 1992].

Sin embargo, la clasificación de los métodos numéricos para la reducción de modelos propuesta por Skelton es más operativa y está agrupada en tres categorías [Skelton, 1988]:

1. **Métodos basados en aproximaciones polinomiales** (aplicables en el dominio de la frecuencia).

2. **Procedimientos de truncamiento del espacio de estados** (aplicables en el dominio del tiempo).

3. **Técnicas de optimizaciones paramétricas** (aplicables en ambos dominios).

Los **métodos basados en aproximaciones polinomiales** son aplicables en el dominio de la frecuencia y normalmente no necesitan de un cálculo computacional intensivo. Son empleados para obtener funciones de transferencia de orden reducido y el modelo de coeficientes utilizado se escoge tratando de relacionar los momentos y los parámetros de Markov, entre el modelo original y el modelo reducido. Estos métodos suelen producir sistemas lineales de control reducidos poco precisos.

Los **procedimientos de truncamiento** son aplicables en el dominio del tiempo y están basados en una transformación de la representación en coordenadas de estados del modelo del sistema lineal de control original que permite obtener un modelo reducido, que mantenga, en la medida de lo posible, las propiedades de respuesta temporal, controlabilidad, observabilidad, etc. Estos tipos de procedimientos permiten una mayor precisión en la representación del sistema lineal de control reducido que los obtenidos con aproximaciones polinómicas y un menor coste computacional respecto a los métodos basados en las técnicas de optimizaciones paramétricas.

Las **técnicas de optimizaciones paramétricas** son técnicas secuenciales, basadas en la minimización de algunos índices definidos apropiadamente que miden el error o diferencia de comportamiento entre el modelo original y el modelo de orden reducido [Wilson y Mishra, 1979]. Habitualmente, estas técnicas requieren un alto coste computacional que las hace, en muchos casos, prohibitivas, sobre todo si el sistema lineal de control original es de gran dimensión. Son de especial interés las técnicas basadas en la minimización de la norma de Hankel, definida como la norma de la diferencia entre las matrices de transferencia de los dos sistemas lineales de control [Glover, 1984].

Así, los procedimientos de truncamiento son los que permiten obtener sistemas lineales de control más reducidos y en los cuales es posible mantener algunas de las propiedades del sistema lineal original. En general, los procedimientos de truncamiento están basados en transformaciones en el espacio de estados y suelen ser agrupados en dos clases de métodos:

- Métodos basados en **realizaciones balanceadas**.
- Métodos basados en **la función de coste y su descomposición**.

La tesis centraliza su estudio en la **reducción de modelos basada en las realizaciones balanceadas**, pues este tipo de aproximación permite obtener un modelo reducido, con las propiedades del modelo inicial y además, es un método muy general, pudiendo ser aplicado en una gran variedad de modelos. Al mismo tiempo, la reducción de modelos basada en las realizaciones balanceadas emplea transformaciones ortogonales, lo que le otorga una importante estabilidad numérica. Sin embargo, como destaca Moore, no existe ningún método de reducción de modelos que sirva para todos los sistemas lineales, dado que muchas de las características del sistema lineal dependen, en gran medida, de la aplicación [Moore, 1981].

6.3.2. Realizaciones balanceadas

Las **realizaciones balanceadas** son técnicas de reducción de modelos que están dentro del grupo de los procedimientos de truncamiento del espacio de estados. Están basadas en el hecho de que la transformación de estados es tal que en la nueva representación los **Gramianos de controlabilidad** ($W_c \in \mathbb{R}^{n\times n}$) y **de observabilidad** ($W_o \in \mathbb{R}^{n\times n}$), son iguales y diagonales [Pernebo y Silverman, 1982], [Laub *et al.*, 1987],

$$W_c = W_o = \Sigma = diag\left(\sigma_1, \ldots, \sigma_n\right). \tag{6.19}$$

Seguidamente se definen los **Gramianos de controlabilidad** y **de observabilidad**.

Los Gramianos o matrices de Gram

Dado el sistema lineal de control, continuo e invariante en el tiempo con la siguiente representación en el modelo de espacio de estados

$$\begin{aligned} \dot{x}\left(t\right) &= Ax\left(t\right) + Bu\left(t\right) \qquad x\left(0\right) = x_0 \\ y\left(t\right) &= Cx\left(t\right) \end{aligned} \tag{6.20}$$

donde se asume que el par (A, B) es controlable y el par (A, C) es observable, el **Gramiano de controlabilidad** W_c y el **Gramiano de observabilidad** W_o,

son las respectivas soluciones de las **ecuaciones matriciales de Lyapunov en tiempo continuo** definidas por

$$\begin{aligned} AW_c + W_c A^t + BB^t &= 0 \\ A^t W_o + W_o A + C^t C &= 0. \end{aligned} \tag{6.21}$$

A los Gramianos también se les suele llamar **matrices de Gram**. Como consecuencia del carácter observable y controlable del sistema lineal de control, son matrices simétricas y definidas positivas. Además, si los valores propios de la matriz A están en el semi-plano izquierdo, entonces los Gramianos de controlabilidad y de observabilidad son definidos por

$$W_c = \int_0^{+\infty} e^{At} B B^t e^{A^t t} dt \tag{6.22}$$

$$W_o = \int_0^{+\infty} e^{A^t t} C^t C e^{At} dt. \tag{6.23}$$

Los siguientes teoremas relacionan las propiedades de controlabilidad y de observabilidad de un sistema lineal de control con las matrices de Gram.

Teorema 18 *Un sistema lineal de control definido por*

$$\dot{x}(t) = Ax(t) + Bu(t) \qquad x(0) = x_0 \tag{6.24}$$

es completamente controlable si y sólo si el Gramiano de controlabilidad

$$W_c(t) = \int_0^t e^{As} B B^t e^{A^t s} ds \tag{6.25}$$

es definido positivo, para $t > 0$.

DEMOSTRACIÓN: La demostración del teorema está en el capítulo 2 de [Kalman, 1963]. ■

Teorema 19 *Un sistema lineal de control definido por*

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \qquad x(0) = x_0 \\ y(t) &= Cx(t) \end{aligned} \tag{6.26}$$

es completamente observable si y sólo si el Gramiano de observabilidad

$$W_o(t) = \int_0^t e^{A^t s} C^t C e^{As} ds \tag{6.27}$$

es definido positivo, para $t > 0$.

DEMOSTRACIÓN: La demostración del teorema está en el capítulo 2 de [Kalman, 1963]. ■

Para balancear el sistema lineal de control nos basaremos en la diagonalización simultánea de ambos Gramianos. Esta diagonalización se logrará mediante una transformación de estado de semejanza que se conoce con el nombre de **transformación contragradiente**.

6.3.3. Transformación contragradiente

Dado el sistema lineal de control, continuo e invariante en el tiempo con la siguiente representación en el modelo de espacio de estados

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \qquad x(0) = x_0 \\ y(t) &= Cx(t) \end{aligned} \tag{6.28}$$

donde se asume que el par (A, B) es controlable y el par (A, C) es observable, si se aplica una transformación no singular

$$x(t) = S\widetilde{x}(t) \tag{6.29}$$

el sistema lineal de control (6.28) se transforma en el sistema lineal de control algebraicamente equivalente

$$\begin{aligned} \dot{\widetilde{x}}(t) &= \widetilde{A}\widetilde{x}(t) + \widetilde{B}u(t) \qquad \widetilde{x}(0) = \widetilde{x}_0 \\ y(t) &= \widetilde{C}\widetilde{x}(t) \end{aligned} \tag{6.30}$$

donde

$$\widetilde{A} = S^{-1}AS, \qquad \widetilde{B} = S^{-1}B \qquad \text{y} \qquad \widetilde{C} = CS. \tag{6.31}$$

Bajo estas condiciones, los Gramianos de controlabilidad y de observabilidad del nuevo sistema lineal de control son

$$\widetilde{W_c} = S^{-1}W_cS^{-t} \qquad \text{y} \qquad \widetilde{W_o} = S^tW_oS. \tag{6.32}$$

Si la transformación (6.29) hace que los nuevos Gramianos $\widetilde{W_c}$ y $\widetilde{W_o}$ sean diagonales, se le denomina **transformación contragradiente**.

Puesto que la matriz de Gram W_c es simétrica y definida positiva, existe una matriz ortogonal $V_c \in \mathbb{R}^{n\times n}$ que permite reducir W_c a la forma diagonal, mediante una transformación de semejanza

$$V_c^tW_cV_c = \Sigma_c^2 \tag{6.33}$$

donde Σ_c es una matriz diagonal y definida positiva.

De la misma manera, dado que la matriz de Gram W_o es simétrica y definida positiva, existe una matriz ortogonal $U \in \mathbb{R}^{n\times n}$ y existe una matriz diagonal y definida positiva Σ, tales que

$$U^t\left(V_c\Sigma_c\right)^tW_o\left(V_c\Sigma_c\right)U = \Sigma^2 \tag{6.34}$$

donde la matriz $\left(V_c\Sigma_c\right)^tW_o\left(V_c\Sigma_c\right)$ es simétrica y definida positiva.

Consideremos a continuación la familia de transformaciones que se puede deducir de la expresión anterior

$$S_k = V_c\Sigma_cU\Sigma^{-k} \qquad\qquad -\infty < k < +\infty \tag{6.35}$$

y que define una familia de transformaciones contragradientes con

$$S_k^{-1}W_cS_k^{-t} = \Sigma^{2k} \qquad \text{y} \qquad S_k^tW_oS_k = \Sigma^{2-2k}. \tag{6.36}$$

Son de especial interés las transformaciones contragradientes para los siguientes valores de k:

- $k = 0$:
  - $\widetilde{W_c} = I$ y $\widetilde{W_o} = \Sigma^2$ (**coordenadas de entrada normal**).

- $k = \frac{1}{2}$:
  - $\widetilde{W}_c = \Sigma$ y $\widetilde{W}_o = \Sigma$ (**coordenadas balanceadas internamente**).
- $k = 1$:
  - $\widetilde{W}_c = \Sigma^2$ y $\widetilde{W}_o = I$ (**coordenadas de salida normal**).

En la tesis el enfoque es dirigido a la transformación contragradiente con $k = \frac{1}{2}$, a la que se denomina **transformación balanceada** y la cual se define por

$$S = V_c \Sigma_c U \Sigma^{-\frac{1}{2}}. \tag{6.37}$$

Utilizando esta transformación de coordenadas balanceadas internamente, se llega a una representación del sistema lineal de control en la que los Gramianos de controlabilidad y de observabilidad son iguales y diagonales, con sus elementos ordenados de mayor a menor. Se consigue así una representación en la que se tiene las variables de estado ordenadas según su importancia sobre las propiedades de controlabilidad y de observabilidad del sistema lineal. A continuación se puede aplicar una técnica de truncamiento, obteniendo así un sistema lineal de control de orden reducido que conserva en gran medida estas propiedades del sistema lineal original.

6.4. Algoritmo de reducción de modelos

Para reducir el sistema lineal de control original, el método numérico descrito en [Laub *et al.*, 1987] y relatado a continuación, está basado en la transformación balanceada y proporciona la diagonalización simultánea de los Gramianos de controlabilidad y de observabilidad.

El método numérico descrito en [Laub *et al.*, 1987] empieza por calcular los factores de Cholesky de las matrices de Gram W_c y W_o,

$$W_c = L_c L_c^t \qquad \text{y} \qquad W_o = L_o L_o^t \tag{6.38}$$

resolviendo con el **método de Hammarling** ([Hammarling, 1982]) las **ecuaciones de Lyapunov**

$$AW_c + W_c A^t + BB^t = 0 \tag{6.39}$$

$$A^t W_o + W_o A + C^t C = 0. \tag{6.40}$$

Posteriormente calcula la DVS del producto de los factores de Cholesky

$$L_o^t L_c = U \Sigma V^t \tag{6.41}$$

con el objetivo de obtener la matriz de transformación balanceada

$$S = L_c V \Sigma^{-\frac{1}{2}} \tag{6.42}$$

y su inversa

$$S^{-1} = \Sigma^{-\frac{1}{2}} U^t L_o^t. \tag{6.43}$$

Finalmente obtiene las matrices del nuevo sistema balanceado definidas por

$$\widetilde{A} = S^{-1} A S = \Sigma^{-\frac{1}{2}} U^t L_o^t A L_c V \Sigma^{-\frac{1}{2}} \tag{6.44}$$

$$\widetilde{B} = S^{-1} B = \Sigma^{-\frac{1}{2}} U^t L_o^t B \tag{6.45}$$

$$\widetilde{C} = C S = C L_c V \Sigma^{-\frac{1}{2}}. \tag{6.46}$$

Por lo tanto, las principales etapas del método numérico descrito en [Laub *et al.*, 1987, §II] son las siguientes:

1. Calcular los factores de Cholesky (L_c y L_o) de los Gramianos (W_c y W_o).
2. Calcular la DVS del producto de los factores de Cholesky (6.41).
3. Calcular la matriz de transformación balanceada (6.42).
4. Calcular las matrices que definen el nuevo sistema balanceado (6.44), (6.45) y (6.46).

Al final de la aplicación del método se obtiene un sistema lineal de control equivalente al sistema lineal original pero con las variables de estado ordenadas según su importancia sobre las propiedades de controlabilidad y de observabilidad del sistema lineal.

Queda tan sólo determinar en qué punto de esta nueva representación se va a truncar para obtener un modelo de orden reducido del sistema original.

Está claro que cuanto más se trunque mayor será la reducción conseguida pero más se perderá de estas propiedades.

Como ya fue anteriormente mencionado, la etapa 2 del método calcula la DVS del producto de dos matrices y es en esta etapa que en el marco de la tesis se han desarrollado implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren, cuyos resultados van a ser descritos en el próximo capítulo.

6.5. Casos de prueba

En el ámbito de la teoría de control, la librería SLICOT pone a disposición un conjunto de implementaciones que proporcionan métodos numéricos para el diseño y análisis de sistemas de control [Benner *et al.*, 1997], [van Huffel y Sima, 2002].

Para estudiar la reducción de modelos de sistemas lineales de control se han elegido algunos casos de prueba, los cuales han sido empleados en los tests computacionales llevados a cabo con la rutina `AB09AD` de SLICOT[1].

En cada caso son definidas las matrices A, B y C del sistema lineal de control definido en (6.1):

- **Caso de prueba 1**: este caso es una adaptación del caso descrito en [Benner *et al.*, 1999a], donde

$$\begin{aligned} A &= V_n \cdot diag\,(\alpha_1, \ldots, \alpha_n) \cdot W_n, \\ B &= [b_{ij}]_{n\times m} = \begin{cases} 1 & si \quad i = j \\ 0 & si \quad i \neq j \end{cases} \quad \text{y} \quad C = [c_{ki}]_{p\times n} = \begin{cases} 1 & si \quad k = i \\ 0 & si \quad k \neq i \end{cases}, \end{aligned} \tag{6.47}$$

  con los números reales α_i $(1 \leq i \leq n)$ uniformemente distribuidos en el intervalo $[-10, 0)$, W_n es una matriz triangular inferior con todas las entradas iguales a uno, V_n es una matriz cuyas entradas de la diagonal secundaria y por debajo de ella son todos iguales a uno, y las demás entradas son cero.

[1] Para más información consultar el apéndice "Resultados con SLICOT".

- **Caso de prueba 2**: este caso es una adaptación del ejemplo 3 descrito en [Penzl, 1999], donde

$$A = diag\left(A_1, A_2, A_3, A_4\right), \qquad B = \begin{bmatrix} 10v_6 \\ v_{n-6} \end{bmatrix} \qquad \text{y} \qquad C = B^t, \tag{6.48}$$

con

$$\begin{aligned} A_1 &= \begin{bmatrix} -1 & 100 \\ -100 & -1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -1 & 200 \\ -200 & -1 \end{bmatrix}, \\ A_3 &= \begin{bmatrix} -1 & 400 \\ -400 & -1 \end{bmatrix} \quad \text{y} \quad A_4 = -diag\left(1, 2, \ldots, n-6\right), \end{aligned} \tag{6.49}$$

y $v_i \in \mathbb{R}^{i\times 1}$ es el vector con todas las entradas iguales a uno.

- **Caso de prueba 3**: este caso es el ejemplo 9.4.1 de la página 331 de [Green y Limebeer, 2000], donde

$$A = -diag\left(\alpha, \alpha^2, \ldots, \alpha^n\right), \quad B = \begin{bmatrix} \sqrt{\alpha} \\ \sqrt{\alpha^2} \\ \vdots \\ \sqrt{\alpha^n} \end{bmatrix} \quad \text{y} \quad C = B^t, \tag{6.50}$$

con $\alpha > 0$ y $\alpha \neq 0$.

- **Caso de prueba 4**: este caso es una adaptación del caso expuesto en 6.2 de [Guerrero *et al.*, 2002], donde

$$\begin{aligned} A &= W_n^{-1} \cdot diag\left(A_1, \ldots, A_q\right) \cdot W_n, \\ B &= \left[b_{ij}\right]_{n\times m} = n - |i-j| \quad \text{y} \quad C = \left[c_{ki}\right]_{p\times n} = n - |k-i|, \end{aligned} \tag{6.51}$$

con $n = 3q$,

$$A_i = \begin{bmatrix} s_i & 0 & 0 \\ 0 & t_i & t_i \\ 0 & -t_i & t_i \end{bmatrix} \qquad \text{y} \qquad s_i = t_i = t^i \tag{6.52}$$

donde se ha empleado $t = 1{,}01$, y W_n es la matriz con todas las entradas iguales a uno, excepto en la diagonal que son cero.

- **Caso de prueba 5**: este caso es el ejemplo de la página 14 de [Benner *et al.*, 1999b], donde se han construido las matrices del sistema lineal de control realizando las siguientes etapas:

  1. $W_c = diag\,(\Sigma_{q1}, \Sigma_{q2}, 0_{q3}, 0_{q4})$, es una matriz diagonal aleatoria semi definida positiva donde $\Sigma_{q1} \in \mathbb{R}^{q1\times q1}$ contiene los valores propios de la matriz W_cW_o del sistema y $\Sigma_{q2} \in \mathbb{R}^{q2\times q2}$ es una matriz diagonal positiva.
  2. $W_o = diag\,(\Sigma_{q1}, 0_{q2}, \Sigma_{q3}, 0_{q4})$, es una matriz diagonal aleatoria semi definida positiva donde $\Sigma_{q3} \in \mathbb{R}^{q2\times q2}$ es una matriz diagonal positiva.
  3. A es una matriz diagonal aleatoria estable.
  4. $F = -\left(AW_c + W_cA^t\right)$.
  5. $G = -\left(A^tW_o + W_oA\right)$.
  6. $B \in \mathbb{R}^{n\times(q1+q2)}$ es el factor de Cholesky de la matriz F, o sea, $F = BB^t$.
  7. $C \in \mathbb{R}^{p\times n}$ es el factor de Cholesky de la matriz G, o sea, $G = CC^t$.
  8. Finalmente,

  $$A = U^tAU, \qquad B = U^tB \qquad \text{y} \qquad C = CU, \tag{6.53}$$

  donde $U \in \mathbb{R}^{n\times n}$ es una matriz ortogonal aleatoria.

- **Caso de prueba 6**: este caso ha sido tomado de una hoja de ejercicios, donde

$$A = [a_{ij}]_{n\times n} = \begin{cases} -1 & si \quad j = i \\ 1 & si \quad j = i+1 \\ 0 & si \quad (j \neq i) \wedge (j \neq i+1) \end{cases}, \tag{6.54}$$

$$B = [b_{ij}]_{n\times m} = \begin{cases} 1 & si \quad i = j \\ 0 & si \quad i \neq j \end{cases} \quad \text{y} \quad C = [c_{ki}]_{p\times n} = \begin{cases} 1 & si \quad k = i \\ 0 & si \quad k \neq i \end{cases}.$$

- **Caso de prueba 7**: este caso es otro caso de la misma hoja de ejercicios, donde

$$
\begin{aligned}
A &= [a_{ij}]_{n\times n} = \begin{cases} -i & si \quad i = j \\ 0 & si \quad i \neq j \end{cases}, \qquad (6.55)\\
B &= [b_{ij}]_{n\times m} = \begin{cases} 1 & si \quad (i = j) = 1 \\ 0{,}1 & si \quad j = m \\ 0 & si \quad ((i \wedge j) \neq 1) \wedge (j \neq m) \end{cases} \quad \text{y}\\
C &= [c_{ki}]_{p\times n} = \begin{cases} 1 & si \quad (k = p) \wedge (i \neq n) \\ 0{,}1 & si \quad (k = p) \wedge (i = n) \\ 0 & si \quad k \neq p \end{cases}.
\end{aligned}
$$

En el apéndice "Resultados con SLICOT" son expuestos los resultados obtenidos por la rutina `AB09AD` en la resolución de ejemplos simples de los casos de prueba descritos anteriormente.

Capítulo 7

La Bidiagonalización del Producto Matricial Implícito

En este capítulo nuestro enfoque está dirigido al **método de Golub-Sølna-van Dooren**, descrito por los autores en [Golub *et al.*, 2000] y que permite calcular la DVS del producto y/o el cociente de dos o más matrices reales. El aspecto más novedoso del método está en la **bidiagonalización del producto matricial implícito** de las matrices factor, es decir, en el cálculo de la matriz bidiagonal superior del producto matricial, sin que el producto matricial sea calculado de manera explícita.

En este contexto, el capítulo empieza por exponer las implementaciones secuenciales y paralelas del método descrito en la sección 2.7.1, para el cálculo de la matriz bidiagonal superior del producto matricial implícito de dos matrices reales cuadradas, desarrolladas con rutinas del entorno de LAPACK y del entorno de SCALAPACK.

A continuación son estudiadas las implementaciones secuenciales y paralelas del método, desarrolladas para el cálculo de la matriz bidiagonal superior del producto matricial implícito de cualquier número de matrices reales cuadradas.

Estas son las implementaciones más generales del método de Golub-Sølna-van Dooren para la bidiagonalización del producto matricial implícito y además, dan continuidad a la investigación desarrollada por Mollar en su tesis doctoral [Mollar, 2003].

Seguidamente se analizan los resultados obtenidos con las implementaciones logradas en el marco de la tesis y para terminar el capítulo se presentan las conclusiones del trabajo elaborado.

7.1. Producto de dos matrices

Como fue estudiado en el capítulo anterior, el método para la reducción de modelos de sistemas lineales de control, descrito en la sección 6.4, está basado en la diagonalización simultánea de los Gramianos de controlabilidad y de observabilidad. Para lograrlo, es necesario obtener la transformación balanceada que se define empleando la DVS del producto de los dos factores de Cholesky de los respectivos Gramianos.

En su tesis doctoral, Mollar [Mollar, 2003] estudió el cálculo de la DVS del producto de dos matrices reales cuadradas (DVSP). Sin embargo, su enfoque fue dirigido a la bidiagonalización del producto matricial y lo hizo desarrollando implementaciones secuenciales y paralelas de tres métodos distintos:

1. El **método de Kogbetliantz implícito** [Kogbetliantz, 1955].
2. El **método de Golub-Sølna-van Dooren** [Golub *et al.*, 2000].
3. El **método de Drmač** [Drmač, 1998].

Mientras que los dos primeros métodos están basados en la bidiagonalización implícita del producto matricial, aunque en el método de Kogbetliantz implícito sea asumido que las matrices del producto son matrices triangulares superiores, en el método propuesto por Drmač es calculado explícitamente ese producto matricial, en el cual las matrices son previamente transformadas de manera que eviten la pérdida de precisión en el producto de las mismas.

En la Figura 145 de su tesis doctoral [Mollar, 2003, pp. 167], Mollar ilustra los tiempos de ejecución obtenidos por sus implementaciones paralelas en el cálculo de los valores singulares de la DVSP y escribe: "*Los métodos de Golub*

y Drmac son claramente mejores, lo que se evidencia especialmente a medida que crece el tamaño de las matrices. Además, en ambos métodos se parte de una matriz completa, no triangular, mientras que en Kogbetliantz partimos de unas matrices previamente triangulares."

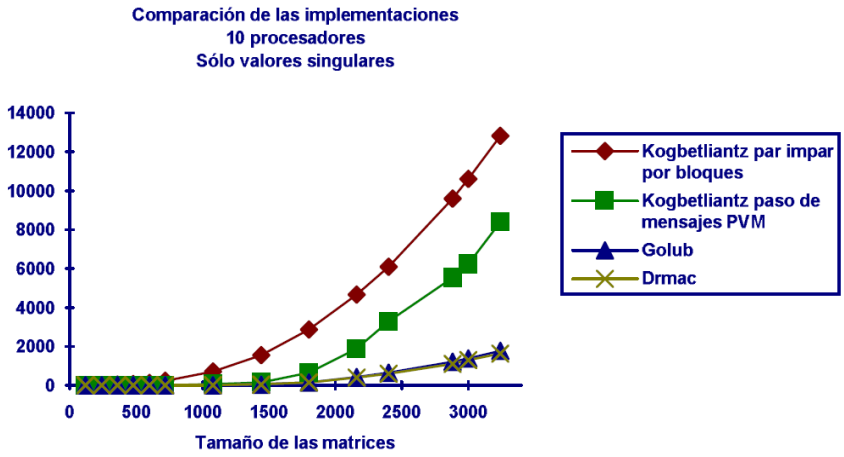

Figura 7.1: Figura 145 de la tesis doctoral de Mollar.

Además, en sus conclusiones, Mollar escribe también: "*Frente al método de Kogbetliantz, el método de Golub es matemática y algorítmicamente más complejo y más costoso de implementar. No obstante la posibilidad de usar BLAS 2 en el proceso de bidiagonalización implícita convierte las implementaciones en más eficientes y no tan complejas como lo serían a priori, al disponer de librerías que realizan gran parte de las operaciones implicadas. Una vez realizada la bidiagonalización, realizamos la SVD de la bidiagonal mediante cualquiera de los complejos métodos disponibles (Golub, Demel, Fernando) que también están implementados en librerías, debido a su gran rendimiento. Como conclusión, si se dispone de LAPACK, o en el caso paralelo de ScaLAPACK, la implementación es mucho menos costosa y los rendimientos que se obtienen son excelentes. En la misma línea, el algoritmo de Drmac es sencillo de realizar*

empleando las librerías estándar y ofrece unos resultados excelentes. La única particularidad es su difícil extensión al caso de más de dos matrices."

Así, y dado que la mayoría de las implementaciones desarrolladas en el marco de la tesis están basadas en la aplicación de las rutinas puestas a disposición por el entorno de LAPACK y por el entorno de SCALAPACK, las implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren, expuesto en la sección 2.7.1 para el cálculo de la DVS del producto matricial implícito de dos matrices reales cuadradas han sido desarrolladas empleando rutinas de BLAS, de LAPACK, de PBLAS y de SCALAPACK.

Tal y como fue igualmente referido por Mollar, una vez calculados los elementos de la matriz bidiagonal superior del producto matricial implícito de las matrices reales, su DVS puede ser obtenida aplicando uno de los métodos iterativos descritos en la sección 2.4.

Al final de la sección 2.7.1 está descrito el algoritmo que implementa el método propuesto por Golub, por Sølna y por van Dooren para el cálculo de la matriz bidiagonal superior del producto matricial implícito de dos matrices reales cuadradas. Su implementación secuencial fue desarrollada empleando principalmente las rutinas de LAPACK que permiten calcular y aplicar las transformaciones de Householder. En particular, la rutina `DLARFG` que calcula la transformación de Householder y la rutina `DLARF` que la aplica a una matriz real de dimensión $m \times n$ de datos en doble precisión.

En ese sentido, la Implementación Secuencial 2 es la implementación secuencial del método de Golub-Sølna-van Dooren descrito en la sección 2.7.1, la cual es la correspondiente implementación del Algoritmo 1, empleando rutinas de BLAS y de LAPACK.

Implementación Secuencial 2 *Golub_Sølna_vanDooren* (A, B, D, E)

Para $i = 1, 2, \ldots, n-1$ **hacer:**
Calcular H_i usando DLARFG de LAPACK
Aplicar H_i a B por la izquierda usando DLARF de LAPACK
Aplicar H_i a A por la derecha usando DLARF de LAPACK
Calcular $H_A^{(i)}$ usando DLARFG de LAPACK
Aplicar $H_A^{(i)}$ a A por la izquierda usando DLARF de LAPACK
Calcular el producto implícito AB usando DGEMV de BLAS
Calcular $H_B^{(i)}$ usando DLARFG de LAPACK
Almacenar el elemento de la diagonal $D(i)$
Almacenar el elemento de la super-diagonal $E(i)$
Aplicar $H_B^{(i)}$ a B por la derecha usando DLARF de LAPACK
Fin
Almacenar el elemento de la diagonal $D(n)$

Siguiendo la misma línea de investigación que hemos empleado en el capítulo 3, una vez obtenida la implementación secuencial del método de Golub-Sølna-van Dooren descrito en la sección 2.7.1, su correspondiente implementación paralela fue desarrollada trasladando las rutinas de BLAS y de LAPACK, a las correspondientes rutinas de PBLAS y de SCALAPACK y, por tanto, se ha aplicado exactamente el mismo proceso de trasladar una implementación secuencial en LAPACK a una implementación paralela en SCALAPACK.

Además, y basándonos en las indicaciones de Mollar en su tesis doctoral, se puede prever que los tiempos de ejecución obtenidos por estas implementaciones secuenciales y paralelas sean "excelentes". Al mismo tiempo, las implementaciones logradas presentan un elevado grado de portabilidad, mientras sean utilizadas en sistemas computacionales que dispongan de las librerías LAPACK y SCALAPACK.

También hay que tener en cuenta que en la implementación secuencial desarrollada, cada una de las operaciones implicadas es paralelizable y ese hecho es comprobado por la existencia de esas mismas funcionalidades en SCALAPACK.

De esta manera y teniendo en cuenta las consideraciones anteriores, la

Implementación Paralela 3 es la implementación paralela del método de Golub-Sølna-van Dooren, descrito en la sección 2.7.1, empleando rutinas de PBLAS y de SCALAPACK, asumiendo que las matrices A y B se encuentran distribuidas al estilo de SCALAPACK y que los vectores D y E son vectores locales de cada procesador (no distribuidos).

Implementación Paralela 3 $Golub\_Sølna\_vanDooren\,(A, B, D, E)$

Obtiene información respecto a los procesadores usando BLACS_PINFO

Para $i = 1, 2, \ldots, n-1$ **hacer:**
Calcular H_i usando PDLARFG de SCALAPACK
Aplicar H_i a B por la izquierda usando PDLARF de SCALAPACK
Aplicar H_i a A por la derecha usando PDLARF de SCALAPACK
Calcular $H_A^{(i)}$ usando PDLARFG de SCALAPACK
Aplicar $H_A^{(i)}$ a A por la izquierda usando PDLARF de SCALAPACK
Calcular el producto implícito AB usando PDGEMV de PBLAS
Calcular $H_B^{(i)}$ usando PDLARFG de SCALAPACK
Almacenar el elemento de la diagonal $D\,(i)$
Almacenar el elemento de la super-diagonal $E\,(i)$
Aplicar $H_B^{(i)}$ a B por la derecha usando PDLARF de SCALAPACK
Fin
Almacenar el elemento de la diagonal $D\,(n)$

Las implementaciones desarrolladas permiten el empleo de la matriz A de dimensión $m \times n$ y de la matriz B de dimensión $n \times n$. Sin embargo, en los experimentos computaciones desarrollados se han empleado solamente matrices reales cuadradas debido al hecho de que los factores de Cholesky de los Gramianos de controlabilidad y de observabilidad son matrices reales cuadradas.

Dado que en el marco de la tesis no se han desarrollado implementaciones secuenciales y paralelas de otro método numérico para la resolución del mismo problema, en el apartado de los resultados obtenidos solamente haremos la exposición y los comentarios de los tiempos de ejecución obtenidos con la ejecución de las implementaciones logradas. Además, tampoco hemos planteado hacer un estudio comparativo con la rutina `DGGSVD` de LAPACK, pues el cálculo

de la DVSG es distinto del cálculo de la DVSP. Y además, SCALAPACK no tiene la funcionalidad del cálculo de la DVSG en paralelo.

7.2. Producto de matrices cuadradas

Aunque Golub, Sølna y van Dooren hayan desarrollado un método general para el cálculo de la DVS de la matriz

$$A = A_k^{s_k} \dots A_2^{s_2} A_1^{s_1} \tag{7.1}$$

donde $s_i = \pm 1$ $(i = 1, ..., k)$ y donde por simplicidad se asume que cada matriz A_i es una matriz real de dimensión $n \times n$ que puede ser invertible, en la sección 2 de [Golub *et al.*, 2000] los autores ejemplifican el cálculo de la matriz bidiagonal superior del producto matricial implícito de tres matrices de dimensión 5×5 cuya ejemplificación es fácilmente extensible al cálculo de los elementos de la matriz bidiagonal superior del producto matricial implícito de cualquier número de matrices reales cuadradas.

Como ya fue referido, el aspecto más novedoso del método de Golub-Sølna--van Dooren está en la bidiagonalización de la matriz A en (7.1), sin calcular explícitamente el producto matricial de las matrices factor. Además, y tal como escriben los autores al final de la sección 2 de [Golub *et al.*, 2000], si en algún momento hay que calcular un cociente de matrices, es decir, si en (7.1) algún s_i es igual a -1, entonces hay que dejar de emplear las transformaciones de Householder y pasar a emplear las rotaciones de Givens, teniendo el cuidado de transformar la matriz factor con $s_i = -1$ en una matriz triangular superior, empleando, por ejemplo, una descomposición QR inicial. En esas condiciones, según Golub, Sølna y van Dooren, el método deja de tener $2n^3$ flops por cada matriz actualizada y pasa a tener $4n^3$ flops por cada matriz actualizada.

En este contexto y dando continuidad a la investigación desarrollada por Mollar en su tesis doctoral, como también fue expuesto, cumpliendo con uno de los objetivos de la tesis y como aportación novedosa, fueron desarrolladas implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren para el cálculo de los elementos de la matriz bidiagonal superior del producto matricial implícito de cualquier número de matrices reales cuadradas.

Así, supongamos que el método de Golub-Sølna-van Dooren es empleado para calcular los elementos de la matriz bidiagonal superior del producto

matricial de tres matrices reales de dimensión 4×4, o sea, el método es empleado en la bidiagonalización de la matriz A con

$$A = A_3 A_2 A_1 \tag{7.2}$$

donde A_1, A_2 y A_3 son matrices reales de dimensión 4×4 no invertibles.

En las implementaciones desarrolladas en el marco de la tesis, la matriz A, matriz de entrada en el método de Golub-Sølna-van Dooren, es una matriz de dimensión $n\times rn$, donde $n\times n$ es la dimensión de cada una de las matrices factor y r es el número de matrices en el producto matricial. En nuestro ejemplo, la matriz A tiene dimensión 4×12

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 \end{array}\right]. \tag{7.3}$$

En estas condiciones, el método de Golub-Sølna-van Dooren es desarrollado según tres pasos y el primer paso del método consiste en las etapas que se describen a continuación.

La primera etapa consiste en aplicar las matrices de Householder H_1 y H_2 de manera que la matriz H_1 es elegida para reducir a cero los 3 últimos elementos de la columna 9 de la matriz A y es aplicada dos veces:

1. H_1 es aplicada sobre las filas de las 4 últimas columnas de la matriz A ($A = H_1 A$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & 0 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & 0 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & \times_0 & 0 & \times_1 & \times_1 & \times_1 \end{array}\right]. \tag{7.4}$$

2. H_1 es aplicada sobre las columnas $5-8$ de la matriz A ($A = AH_1^t$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_0 & \times_0 & \times_0 & \times_0 & \times_1 & \times_1 & \times_1 & \times_1 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_1 & \times_1 & \times_1 & \times_1 & 0 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_1 & \times_1 & \times_1 & \times_1 & 0 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & \times_1 & \times_1 & \times_1 & \times_1 & 0 & \times_1 & \times_1 & \times_1 \end{array}\right]. \tag{7.5}$$

La matriz de Householder H_2 es elegida para reducir a cero los 3 últimos elementos de la columna 5 de la matriz A y también es aplicada dos veces:

1. H_2 es aplicada sobre las filas de las columnas $5-8$ de la matriz A ($A = H_2A$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_0 & \times_0 & \times_0 & \times_0 & \times_2 & \times_2 & \times_2 & \times_2 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \\ \times_0 & \times_0 & \times_0 & \times_0 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \end{array}\right]. \tag{7.6}$$

2. H_2 es aplicada sobre las 4 primeras columnas de la matriz A ($A = AH_2^t$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_1 & \times_1 & \times_1 & \times_1 & \times_2 & \times_2 & \times_2 & \times_2 & \times_1 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \\ \times_1 & \times_1 & \times_1 & \times_1 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \end{array}\right]. \tag{7.7}$$

La segunda etapa consiste en aplicar la matriz de Householder $H_A^{(1)}$ sobre las filas de las primeras 4 columnas de la matriz A para reducir a cero los 3 últimos elementos de la primera columna de la matriz A ($A = H_A^{(1)}A$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_1 & \times_1 & \times_1 & \times_1 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_1 & \times_1 & \times_1 \end{array}\right]. \tag{7.8}$$

Con la aplicación de la matriz de Householder $H_A^{(1)}$, la matriz A en (7.8) tiene las matrices A_1, A_2 y A_3 con la misma estructura, es decir, en cada una de las matrices factor fueron reducidos a cero los 3 últimos elementos de su primera columna.

Terminada la etapa anterior, la etapa siguiente es para reducir a cero los 2 últimos elementos de la primera fila de la matriz producto definida en (7.2), pero sin calcular explícitamente ese producto matricial. Sin embargo, la primera fila del producto matricial (7.2) puede ser calculado haciendo el producto de los 4 primeros elementos de la primera fila de (7.8) con las matrices factor que están almacenadas en las columnas $5-12$ de la matriz A. En estas condiciones, se puede construir el vector auxiliar v definido por

$$v = A\,(1, 1:4) \cdot A\,(1:4, 5:8) \cdot A\,(1:4, 9:12) \tag{7.9}$$

que permite construir la matriz de Householder $H_B^{(1)}$ que reduce a cero los 2 últimos elementos de la primera fila de la matriz (7.2).

Obtenida la matriz de Householder $H_B^{(1)}$ y aplicada por la derecha a las 4 últimas columnas de la matriz A, tenemos

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 \end{array}\right] \tag{7.10}$$

con la que se obtiene la matriz

$$H_A^{(1)} A_3 H_2^t H_2 A_2 H_1^t H_1 A_1 H_B^{(1)} = \left[\begin{array}{cccc} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{array}\right] \tag{7.11}$$

que termina el primer paso del método de Golub-Sølna-van Dooren.

Hay que señalar que en el ejemplo se está empleando la misma notación que en la sección 2.7.1.

El segundo paso del método de Golub-Sølna-van Dooren consiste en volver a aplicar las mismas etapas del primer paso.

Así, se empieza por aplicar las matrices de Householder H_3 y H_4 donde la matriz H_3 es elegida para reducir a cero los 2 últimos elementos de la columna 10 de la matriz A e igualmente es aplicada dos veces:

1. H_3 es aplicada sobre las filas de las 3 últimas columnas de la matriz A ($A = H_3A$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & 0 & \times_3 & \times_3 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_2 & \times_2 & \times_2 & 0 & 0 & \times_3 & \times_3 \end{array}\right]. \tag{7.12}$$

2. H_3 es aplicada sobre las columnas $6-8$ de la matriz A ($A = AH_3^t$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_3 & \times_3 & \times_3 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_3 & \times_3 & \times_3 & 0 & 0 & \times_3 & \times_3 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_3 & \times_3 & \times_3 & 0 & 0 & \times_3 & \times_3 \end{array}\right]. \tag{7.13}$$

La matriz de Householder H_4 es elegida para reducir a cero los 2 últimos elementos de la columna 6 de la matriz A y es aplicada de nuevo dos veces:

1. H_4 es aplicada sobre las filas de las columnas $6-8$ de la matriz A ($A = H_4A$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & \times_4 & \times_4 & \times_4 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_3 & \times_3 \\ 0 & \times_2 & \times_2 & \times_2 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_3 & \times_3 \end{array}\right]. \tag{7.14}$$

2. H_4 es aplicada sobre las columnas $2-4$ de la matriz A ($A = AH_4^t$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_3 & \times_3 & \times_3 & 0 & \times_4 & \times_4 & \times_4 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_3 & \times_3 \\ 0 & \times_3 & \times_3 & \times_3 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_3 & \times_3 \end{array}\right]. \tag{7.15}$$

A continuación es aplicada la matriz de Householder $H_A^{(2)}$ sobre las filas de las columnas $2-4$ de la matriz A para reducir a cero los 2 últimos elementos de la columna 2 de la matriz A ($A = H_A^{(2)} A$):

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_4 & \times_4 & \times_4 & 0 & \times_4 & \times_4 & \times_4 & 0 & \times_3 & \times_3 & \times_3 \\ 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_3 & \times_3 \\ 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_3 & \times_3 \end{array}\right]. \tag{7.16}$$

De nuevo, con la aplicación de la matriz $H_A^{(2)}$, la matriz A en (7.16) tiene las matrices A_1, A_2 y A_3 con la misma estructura.

Lograda la etapa anterior, la etapa siguiente consiste en reducir a cero el último elemento de la segunda fila de la matriz producto definida en (7.2). Una vez más, el producto matricial no es calculado explícitamente y el vector auxiliar v que almacena el resultado del producto matricial, definido por

$$v = A\,(2, 2:4) \cdot A\,(2:4, 6:8) \cdot A\,(2:4, 10:12) \tag{7.17}$$

que permite construir la matriz de Householder $H_B^{(2)}$ que reduce a cero el último elemento de la segunda fila de la matriz (7.2).

La matriz de Householder $H_B^{(2)}$ es aplicada por la derecha a las 3 últimas columnas de la matriz A, obteniéndose la matriz

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_4 & \times_4 & \times_4 & 0 & \times_4 & \times_4 & \times_4 & 0 & \times_4 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_4 & \times_4 \\ 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_4 & \times_4 & 0 & 0 & \times_4 & \times_4 \end{array}\right] \tag{7.18}$$

con la cual termina el segundo paso del método de Golub-Sølna-van Dooren pues

$$H_A^{(2)} A_3 H_4^t H_4 A_2 H_3^t H_3 A_1 H_B^{(2)} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}. \tag{7.19}$$

A partir de este momento es evidente cuál es el tercer paso del método de Golub-Sølna-van Dooren y logradas todas sus etapas, la matriz A tiene la

estructura

$$A = \left[\begin{array}{cccc|cccc|cccc} \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 & \times_2 \\ 0 & \times_4 & \times_4 & \times_4 & 0 & \times_4 & \times_4 & \times_4 & 0 & \times_4 & \times_4 & \times_4 \\ 0 & 0 & \times_6 & \times_6 & 0 & 0 & \times_6 & \times_6 & 0 & 0 & \times_5 & \times_5 \\ 0 & 0 & 0 & \times_6 & 0 & 0 & 0 & \times_6 & 0 & 0 & 0 & \times_5 \end{array}\right] \tag{7.20}$$

donde

$$H_A^{(3)} A_3 H_6^t H_6 A_2 H_5^t H_5 A_1 = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \alpha_3 & \beta_3 \\ 0 & 0 & 0 & \alpha_4 \end{bmatrix} \tag{7.21}$$

y, por tanto, el método de Golub-Sølna-van Dooren completa sus operaciones y, además, fácilmente se extiende al cálculo de la matriz bidiagonal superior del producto matricial implícito de cualquier número de matrices reales cuadradas

$$A_k A_{k-1} \ldots A_2 A_1 = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \ldots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \ldots & 0 & 0 & \alpha_n \end{bmatrix}. \tag{7.22}$$

donde cada matriz A_i $(i = 1, ..., k)$ es una matriz real de dimensión $n \times n$ no invertible.

El Algoritmo 11 describe el método de Golub-Sølna-van Dooren para el cálculo de la matriz bidiagonal superior de producto matricial implícito de cualquier número de matrices reales cuadradas, donde la matriz de entrada A tiene dimensión $n \times rn$ y los vectores de salida D y E son los vectores que almacenan los elementos de la matriz bidiagonal superior.

Algoritmo 11 *Golub_Sølna_vanDooren* (A, D, E)

Para $i = 1, 2, \ldots, n-1$ **hacer:**
  Para $j = r, r-1, \ldots, 2$ **hacer:**
    Calcular H_j que anule los elementos $A\,(i+1:n, (j-1)n+i)$
    Aplicar H_j a A por la izquierda
      $(A = H_j A\,(i:n, (j-1)n+i:jn))$
    Aplicar H_j a A por la derecha
      $(A = A\,(i:n, (j-2)\,n+i:(j-1)\,n)\,H_j^t)$
  Fin
  Calcular $H_A^{(i)}$ que anule los elementos $A\,(i+1:n, i)$
  Aplicar $H_A^{(i)}$ a A por la izquierda $(A = H_A^{(i)} A\,(i:n, i:n))$
  Si $i \neq n-1$ **hacer:**
    Inicializar el vector v_{aux} con $v_{aux} = A\,(i, i:n)$
    Para $j = 2, 3, \ldots, r$ **hacer:**
      Calcular el producto matricial implícito
        $(v\,(i:n) = A\,(i:n, (j-1)n+i:jn)\,v_{aux})$
      Actualizar v_{aux} con $v\,(i:n)$
    Fin
    Calcular $H_B^{(i)}$ que anule los elementos $v\,(i+2:n)$
    Almacenar el elemento de la diagonal $(D\,(i) = v\,(i))$
    Almacenar el elemento de la super-diagonal $(E\,(i) = v\,(i+1))$
    Aplicar $H_B^{(i)}$ a A por la derecha
      $(A = A\,(i:n, (r-1)n+i:rn)\,H_B^{(i)})$
  Fin
Fin
Almacenar el elemento de la diagonal $(D\,(n-1) = v\,(n-1))$
Almacenar el elemento de la super-diagonal $(E\,(n-1) = v\,(n))$
Inicializar el elemento de la diagonal $(D\,(n) = A\,(n, n))$
Para $j = 2, 3, \ldots, r$ **hacer:**
  Actualizar el elemento de la diagonal $(D\,(n) = D\,(n)\,A\,(n, jn))$
Fin

Descrito el algoritmo del método de Golub-Sølna-van Dooren, su implementación secuencial fue desarrollada empleando rutinas de BLAS y de LAPACK y, por tanto, la Implementación Secuencial 3 es la correspondiente implementación

secuencial del Algoritmo 11.

Implementación Secuencial 3 *Golub_Sølna_vanDooren* (A, D, E)

Para $i = 1, 2, \ldots, n-1$ **hacer:**
  Para $j = r, r-1, \ldots, 2$ **hacer:**
    Calcular H_j usando DLARFG de LAPACK
    Aplicar H_j a A por la izquierda usando DLARF de LAPACK
    Aplicar H_j a A por la derecha usando DLARF de LAPACK
  Fin
  Calcular $H_A^{(i)}$ usando DLARFG de LAPACK
  Aplicar $H_A^{(i)}$ a A por la izquierda usando DLARF de LAPACK
  Si $i \neq n-1$ **hacer:**
    Inicializar el vector v_{aux} usando DCOPY de BLAS
    Para $j = 2, 3, \ldots, r$ **hacer:**
      Calcular el producto implícito usando DGEMV de BLAS
      Actualizar v_{aux} con v usando DCOPY de BLAS
    Fin
    Calcular $H_B^{(i)}$ usando DLARFG de LAPACK
    Almacenar el elemento de la diagonal $D(i)$
    Almacenar el elemento de la super-diagonal $E(i)$
    Aplicar $H_B^{(i)}$ a A por la derecha usando DLARF de LAPACK
  Fin
Fin
Almacenar el elemento de la diagonal $D(n-1)$
Almacenar el elemento de la super-diagonal $E(n-1)$
Inicializar el elemento de la diagonal $D(n)$
Para $j = 2, 3, \ldots, r$ **hacer:**
  Actualizar el elemento de la diagonal $D(n)$
Fin

A partir de la Implementación Secuencial 3 fue desarrollada la implementación paralela del método de Golub-Sølna-van Dooren empleando las correspondientes rutinas de PBLAS y de SCALAPACK, teniendo en cuenta que la matriz A de dimensión $n \times rn$ está distribuida al estilo de SCALAPACK y que los vectores D y E son vectores no distribuidos.

7.3. Resultados experimentales

Para testar todas las implementaciones desarrolladas en el marco de la tesis para el método de Golub-Sølna-van Dooren, se ha realizado igualmente un gran número de experimentos computacionales.

En los primeros experimentos computacionales se han empleado matrices de prueba haciendo adaptaciones y/o combinaciones de las matrices definidas en los casos de prueba expuestos en la sección 6.5, aunque esos casos de prueba hayan sido empleados para estudiar la rutina `AB09AD` de SLICOT, dedicada a la reducción de modelos de sistemas lineales de control.

A continuación, para testar todas las implementaciones del método de Golub--Sølna-van Dooren se han empleado matrices de prueba generadas de manera aleatoria y para testar las implementaciones paralelas se han empleado hasta 10 procesadores, organizados según las configuraciones descritas en la Tabla 3.7 para la malla de procesadores y, de nuevo, fue empleado el bloque matricial de dimensión 32×32 para la distribución cíclica de los datos.

7.3.1. Prestaciones del producto de dos matrices

En este apartado se exponen los tiempos de ejecución obtenidos con las implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren desarrolladas para el cálculo de la matriz bidiagonal superior del producto matricial implícito de dos matrices reales.

Como fue descrito en la sección 7.1, las implementaciones desarrolladas permiten el cálculo de la matriz bidiagonal superior del producto matricial AB, siempre que el producto matricial esté bien definido. Sin embargo, en todos los experimentos computacionales desarrollados se han empleado matrices reales cuadradas, cuya dimensión varía de 100×100 hasta 1000×1000 y generadas de manera aleatoria.

Hay que señalar que no se han empleado matrices de prueba con mayores dimensiones debido al incremento de los tiempos de ejecución a obtener con la implementación secuencial.

En este contexto, la Figura 7.2 ilustra los tiempos de ejecución obtenidos con la implementación secuencial del método de Golub-Sølna-van Dooren y la correspondiente implementación paralela ejecutada en un solo procesador. Como

se puede observar, en los dos conjuntos de tiempos de ejecución, estos aumentan considerablemente a medida que aumenta la dimensión de las matrices en el producto matricial AB y, además, son tiempos de ejecución muy semejantes, aunque los tiempos de ejecución obtenidos con la implementación secuencial sean un poco mejores.

Así, y al contrario de lo que se ha hecho en los otros capítulos de la tesis, el *speed-up* y la eficiencia han sido valorados, en este capítulo, según (1.7) y (1.10), es decir, relativos al tiempo de la implementación secuencial.

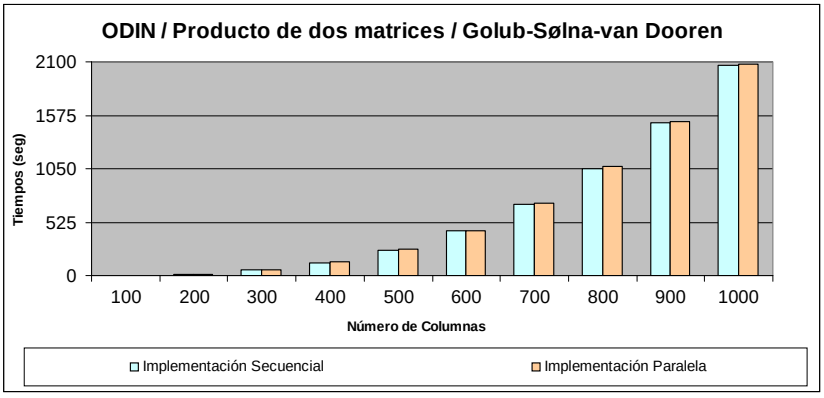

Figura 7.2: Tiempos de ejecución secuenciales *versus* tiempos de ejecución paralelos con un solo procesador para el producto de dos matrices.

Dado que los tiempos de ejecución más expresivos son los tiempos de ejecución obtenidos empleando matrices de mayor dimensión, la Figura 7.3 ilustra los tiempos de ejecución de la Figura 7.2 e incluye los tiempos de ejecución obtenidos con la implementación paralela con 2, 4, 6, 8 y 10 procesadores, para los seis últimos casos de matrices de prueba empleados.

Se puede observar que hasta 6 procesadores la implementación paralela desarrollada presenta un comportamiento "esperado", es decir, la aportación de más procesadores reduce más o menos proporcionalmente el tiempo de ejecución y además, esto puede ser observado en la Figura 7.4, donde se ilustran

los valores del *speed-up* de la implementación paralela y que están cercanos a los valores ideales.

Para 8 procesadores y principalmente para 10 procesadores, ya se empieza a notar que la aportación de más procesadores no reduce en proporción el tiempo de ejecución y, por tanto, el valor del *speed-up* con 8 procesadores es 7 y con 10 procesadores es 8.

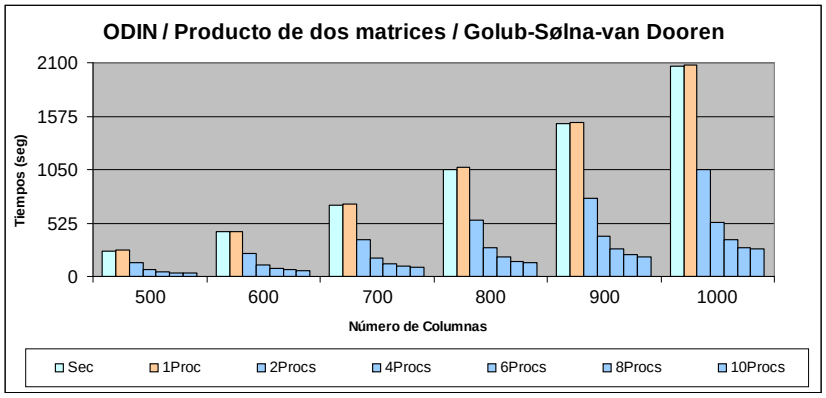

Figura 7.3: Tiempos de ejecución para el producto de dos matrices.

Como fue expuesto, no se han desarrollado más experimentos computacionales para obtener tiempos de ejecución de la implementación paralela empleando matrices de mayor dimensión, debido al gran incremento de tiempo que es necesario esperar para obtener los tiempos de ejecución de la implementación secuencial con esas matrices.

Es evidente que en este tipo de problema se puede asumir que el tiempo de ejecución de la implementación secuencial puede ser aproximado a partir de los tiempos de ejecución obtenidos con la implementación paralela ejecutada con 2 y/o 4 procesadores, y a partir de esa aproximación, estudiar el comportamiento de la implementación paralela con más procesadores y con matrices de mayor dimensión. Sin embargo, en toda la tesis no se ha empleado esa metodología.

Además, los tiempos de ejecución obtenidos con las implementaciones

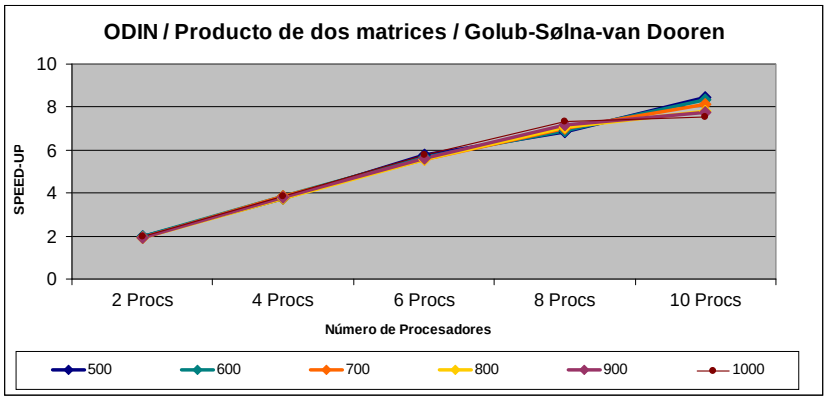

Figura 7.4: *Speed-up* de la implementación paralela para el producto de dos matrices.

desarrolladas para el método de Golub-Sølna-van Dooren permiten concluir que la implementación paralela es eficiente, dentro del contexto donde se han realizado los experimentos computacionales, y su eficiencia varía entre 75 % y 98 %, como se puede observar por los valores ilustrados en la Tabla 7.1.

Implementación	**500**	**600**	**700**	**800**	**900**	**1000**
GSvD(2)	98 %	98 %	97 %	96 %	97 %	98 %
GSvD(4)	94 %	96 %	96 %	94 %	96 %	97 %
GSvD(6)	94 %	94 %	93 %	93 %	93 %	96 %
GSvD(8)	86 %	86 %	87 %	88 %	90 %	91 %
GSvD(10)	85 %	83 %	83 %	78 %	78 %	75 %

Tabla 7.1: Eficiencia de la implementación paralela para el producto de dos matrices.

7.3.2. Prestaciones del producto de matrices cuadradas

Este es el apartado de la tesis donde es más difícil ilustrar los tiempos de ejecución obtenidos con las implementaciones desarrolladas. Es evidente que las implementaciones más generales del método de Golub-Sølna-van Dooren abren un gran abanico de casos de prueba en los cuales se puede variar tanto la dimensión de las matrices factor en el producto matricial, como también se puede variar el número de factores en el producto matricial.

Así, para los experimentos computacionales se ha empleado el mismo tipo de matrices de prueba y el mismo entorno paralelo que en el apartado anterior. Dado que en el apartado anterior el producto matricial es el producto de dos matrices, en este apartado se han empleado productos matriciales implícitos con 3, 4, 5, 6, 8 y 10 factores. Como es natural, a medida que aumenta el número de factores en el producto matricial y a medida que aumenta la dimensión de cada matriz factor, los tiempos de ejecución pasan a ser muy elevados y, por tanto, hay algunos casos de prueba que no fueron desarrollados.

En ese sentido, la Tabla 7.2 ilustra los tiempos de ejecución obtenidos con la Implementación Secuencial 3. Hay que señalar que los tiempos de ejecución obtenidos con la correspondiente implementación paralela ejecutada con un solo procesador están un poco por encima de los valores de la Tabla 7.2.

	Número de Factores					
Dimensión	**3**	**4**	**5**	**6**	**8**	**10**
100	2.4	3.5	4.2	5.4	7.0	8.6
200	24.1	33.6	44.3	50.0	69.2	85.4
300	84.2	119.0	145.2	172.2	228.1	281.4
400	220.0	300.2	382.3	423.3	560.1	684.8
500	402.5	525.6	672.4	795.9	1052.5	1321.3
600	683.0	921.1	1140.0	1357.2	1791.3	2251.2
700	1070.2	1452.5	1825.2	–	–	–
800	1601.2	2196.3	–	–	–	–
900	2273.8	–	–	–	–	–
1000	–	–	–	–	–	–

Tabla 7.2: Tiempos de ejecución secuenciales para el producto de matrices.

La Figura 7.5 ilustra algunos de los tiempos de ejecución expuestos en la Tabla 7.2. Es evidente que los tiempos de ejecución aumentan considerablemente con el número de factores en el producto matricial y con la dimensión de las matrices factor.

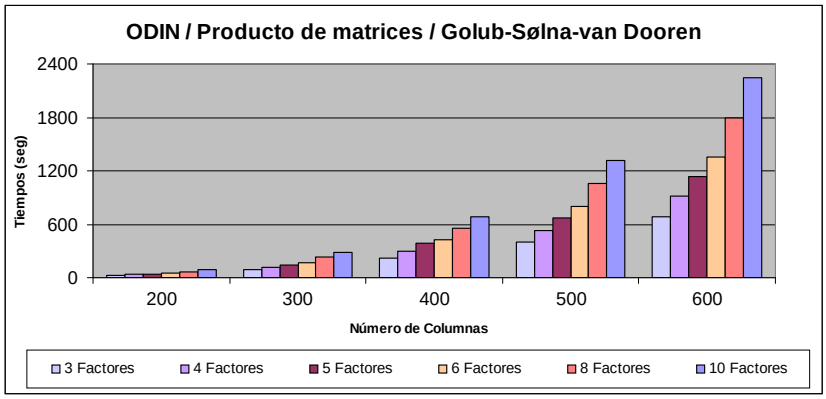

Figura 7.5: Tiempos de ejecución secuenciales para el producto de matrices.

Para valorar la implementación paralela desarrollada, se han considerado de nuevo los tiempos de ejecución más expresivos obtenidos con la implementación secuencial y esos tiempos son los tiempos de ejecución obtenidos con las matrices factor de dimensión 400×400, 500×500 y 600×600.

Así, las Figuras 7.6–7.8 ilustran los tiempos de ejecución obtenidos con la implementación paralela con 2, 4, 6, 8 y 10 procesadores.

Al mismo tiempo, las Figuras 7.9–7.11 ilustran los valores del *speed-up* de la implementación paralela, calculados con los tiempos de ejecución ilustrados en las Figuras 7.6–7.8. Se puede observar que hasta 6 procesadores, los valores del *speed-up* obtenidos están cercanos a los valores ideales, pero, de nuevo, con 8 y 10 procesadores, esos valores vuelven a bajar para valores que están entre 7 y 8. Aún así, y para los experimentos computacionales desarrollados, se puede asumir que esta implementación paralela del método de Golub-Sølna-van Dooren es eficiente.

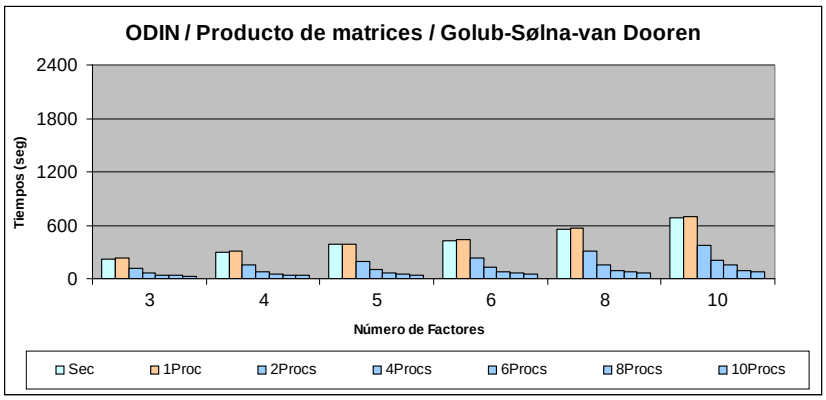

Figura 7.6: Tiempos de ejecución para el producto de matrices de dimensión 400×400.

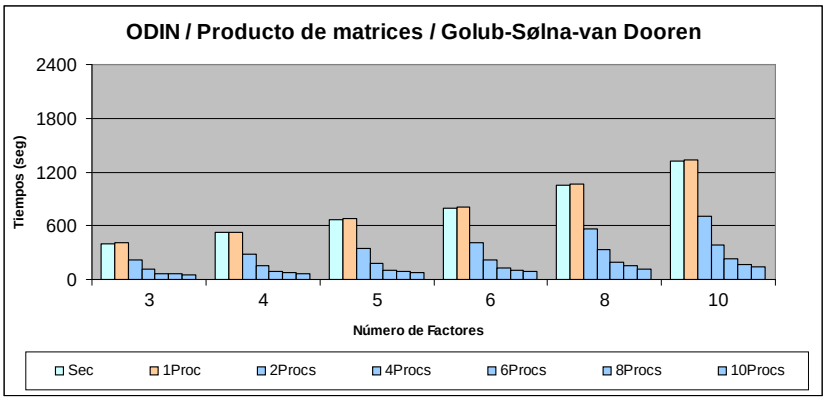

Figura 7.7: Tiempos de ejecución para el producto de matrices de dimensión 500×500.

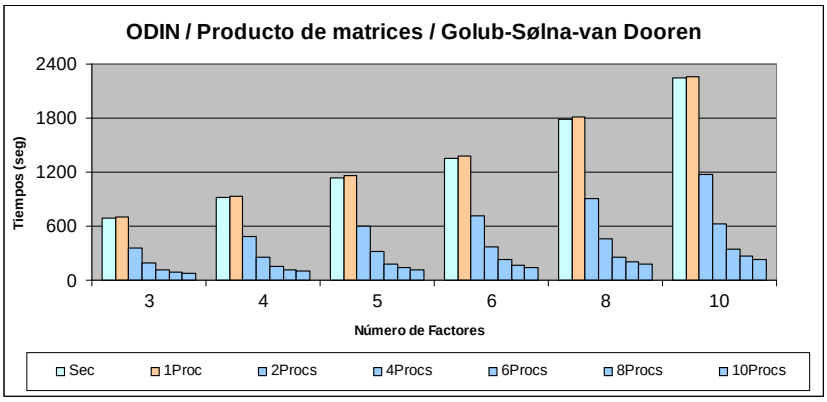

Figura 7.8: Tiempos de ejecución para el producto de matrices de dimensión 600×600.

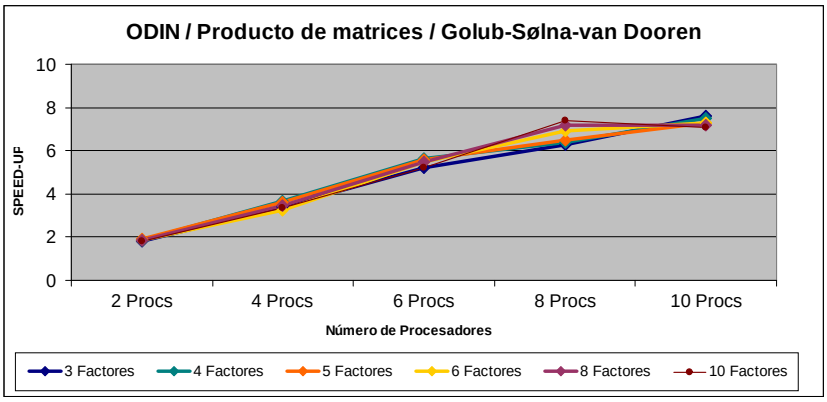

Figura 7.9: *Speed-up* de la implementación paralela para el producto de matrices de dimensión 400×400.

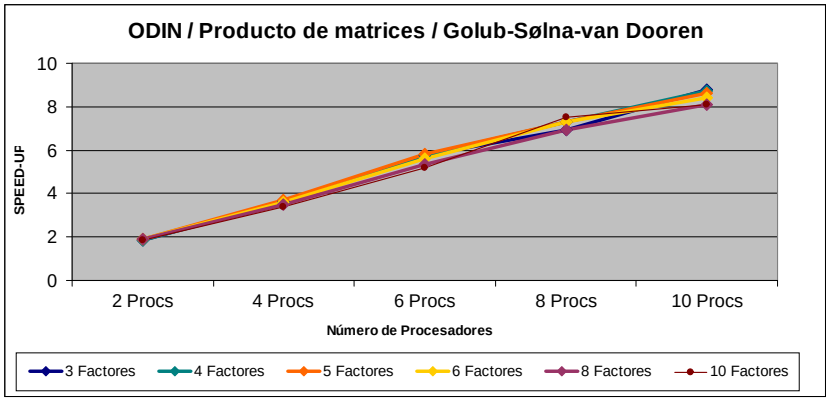

Figura 7.10: *Speed-up* de la implementación paralela para el producto de matrices de dimensión 500×500.

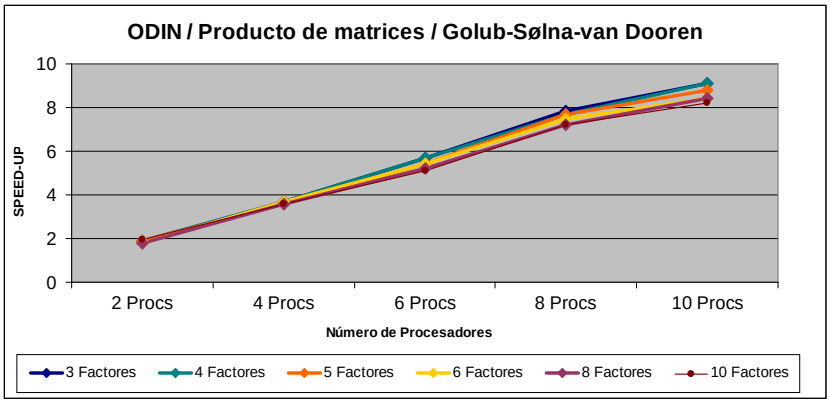

Figura 7.11: *Speed-up* de la implementación paralela para el producto de matrices de dimensión 600×600.

7.4. Conclusiones

En este séptimo capítulo se ha empezado por describir la implementación secuencial y paralela del método de Golub-Sølna-van Dooren descrito en la sección 2.7.1.

A continuación y como aportación novedosa fue expuesta la versión general del método de Golub-Sølna-van Dooren para el cálculo de la matriz bidiagonal superior del producto matricial implícito de cualquier número de matrices reales cuadradas. Además de describir su algoritmo, fue también expuesta su implementación secuencial, desarrollada en el entorno de LAPACK.

Del trabajo desarrollado, se pueden presentar las siguientes conclusiones:

- Todas las implementaciones desarrolladas presentan tiempos de ejecución que se consideran "buenos", teniendo en cuenta que no fueron contrastados con los tiempos de ejecución obtenidos por otro método numérico.

- Es evidente que a medida que aumenta el número de matrices en el producto matricial o a medida que aumenta la dimensión de las matrices, los tiempos de ejecución obtenidos son muy elevados y, por tanto, en los experimentos computacionales se han empleado matrices de prueba de pequeñas dimensiones. Por este motivo, existe la necesidad de estudiar el comportamiento de todas las implementaciones desarrolladas, empleando matrices de mayor dimensión, y principalmente, de buscar aplicaciones donde sea evidente la necesidad de aplicar la versión general del método de Golub-Sølna-van Dooren.

- Las implementaciones paralelas desarrolladas en el entorno de SCALAPACK presentan buenos comportamientos, en el sentido de que los valores del *speed-up* hasta 6 procesadores son valores muy cercanos a los valores ideales. Sin embargo, para 8 y para 10 procesadores, los valores del *speed-up* empiezan a bajar y, por tanto, es necesario testar las implementaciones con más procesadores, pero lo más deseable sería que los nuevos experimentos computacionales fuesen desarrollados en aplicaciones reales.

Para el capítulo 7 se pueden plantear tres líneas de trabajo futuro:

1. Desarrollar los experimentos computacionales anteriormente nombrados, empleando más procesadores y en aplicaciones reales.

2. Intentar mejorar los tiempos de ejecución obtenidos en todas las implementaciones desarrolladas para el método de Golub-Sølna-van Dooren, estudiando su adaptación a un método orientado a bloques matriciales y, por tanto, trasladar todas las implementaciones con rutinas de BLAS de nivel 2 a rutinas de BLAS de nivel 3.

3. Desarrollar un estudio comparativo entre todas las implementaciones desarrolladas para el método de Golub-Sølna-van Dooren y las correspondientes implementaciones de un nuevo método numérico, donde una posibilidad es el método descrito en [Heath *et al.*, 1986] y otra posibilidad es el método descrito en [Stewart, 1997].

Capítulo 8

Conclusiones y Futuras Perspectivas

EN este último capítulo se presentan las conclusiones más relevantes del trabajo desarrollado en el marco de la tesis. A continuación se describen algunas líneas de investigación de trabajo futuro que proyecta la presente memoria de la tesis. El conjunto de elementos de carácter científico a las que ha dado lugar la investigación desarrollada cierra el capítulo.

8.1. Conclusiones del trabajo desarrollado

Los objetivos de la tesis están repartidos entre el desarrollo de implementaciones secuenciales y paralelas de métodos numéricos para el cálculo de la DVS de matrices reales densas y su aplicación a la reducción de modelos de sistemas lineales de control. Además, las implementaciones desarrolladas en el marco de la tesis deben presentar altas prestaciones y un elevado grado de portabilidad. Al mismo tiempo, se debe hacer un estudio del comportamiento de las implementaciones desarrolladas y se debe contrastar los tiempos de ejecución obtenidos por sus ejecuciones respecto a los tiempos de ejecución obtenidos por las correspondientes rutinas de las librerías numéricas estándar con la misma

funcionalidad.

En el capítulo 2 se han estudiado los conceptos del problema estándar del cálculo de la DVS de una matriz real densa y los métodos numéricos que le están asociados. Como fue descrito, el procedimiento más eficiente para calcular la DVS de una matriz real densa es empezar por reducir la matriz a una forma bidiagonal, habitualmente a la forma bidiagonal superior, empleando un número finito de transformaciones ortogonales y posteriormente emplear un proceso iterativo para calcular la DVS de esa matriz bidiagonal.

El proceso para reducir la matriz inicial a la forma bidiagonal (superior) es conocido como el método de la bidiagonalización y en el cálculo de la DVS de una matriz real densa esta etapa tiene un coste computacional mucho más elevado que el proceso iterativo empleado posteriormente para calcular la DVS de la matriz bidiagonal. Además, como fue estudiado, el método de bidiagonalización más conocido es el método propuesto por Golub y por Kahan, y consiste en la aplicación de sucesivas transformaciones de Householder por la izquierda y por la derecha de la matriz, repercutiendo negativamente en las prestaciones de la implementación paralela del algoritmo sobre sistemas de memoria distribuida.

Debido a ese problema, en el capítulo 3 se han estudiado dos métodos alternativos de bidiagonalización, uno propuesto por Ralha y más tarde, otro propuesto por Barlow, en los que las transformaciones de Householder son aplicadas solamente por el lado derecho de la matriz. Esto permite definir todas las operaciones en términos de las columnas de la matriz a transformar, logrando así que el desarrollo de implementaciones paralelas sea más sencillo que con los métodos tradicionales, además de reducir significativamente las comunicaciones necesarias.

Como fue expuesto en el capítulo 3, se ha realizado un estudio comparativo entre las implementaciones secuenciales y paralelas de los dos métodos alternativos de bidiagonalización, desarrolladas en el entorno de las librerías numéricas LAPACK y SCALAPACK, y las rutinas de estas librerías con la misma funcionalidad. Además, como trabajo más novedoso, se han presentado modificaciones al método propuesto por Barlow con el objetivo de reducir el número de comunicaciones necesarias en la implementación paralela destinada a sistemas de memoria distribuida.

Las conclusiones más relevantes que se logran del trabajo desarrollado respecto a los métodos alternativos de bidiagonalización son:

1. Los métodos tienen el mismo orden de complejidad y su mayor ventaja está en el cálculo de la DVS completa de una matriz real densa, dado que presentan un coste computacional mucho menor que los métodos tradicionales.

2. Sus implementaciones secuenciales y paralelas obtienen tiempos de ejecución semejantes, aunque los mejores tiempos de ejecución sean los del método de Barlow modificado.

3. Sus implementaciones secuenciales y paralelas presentan altas prestaciones, con valores que son competitivos con los tiempos de ejecución obtenidos con la rutina `DGEBRD` de LAPACK y con la rutina `PDGEBRD` de SCALAPACK.

4. Sus implementaciones paralelas obtienen mejores tiempos de ejecución en mallas de procesadores del tipo $p \times 1$, en el entorno de SCALAPACK, aunque en este tipo de malla de procesadores no se obtenga la distribución de datos descrita por Ralha, mientras que la rutina `PDGEBRD` obtiene mejores tiempos de ejecución en mallas de procesadores que están más cercanas a la configuración cuadrada.

5. Para desarrollar implementaciones paralelas con la distribución de datos descrita por Ralha, la mejor metodología es la de emplear rutinas de LAPACK y de BLAS para las computaciones, y rutinas de MPI o PVM para las comunicaciones. En este tipo de implementación, la implementación paralela es esencialmente la implementación secuencial, en la cual se usan comunicaciones para calcular el vector con los productos internos, donde hay una pequeña redundancia en las computaciones debido al hecho de que todos los procesadores tienen que calcular el mismo vector de Householder. Sin embargo, el impacto de este sobrecoste no es dramático en el coste total de los tiempos de ejecución, siempre que la razón m/p sea grande, donde m es el número de filas de la matriz y p es el número de procesadores empleados.

6. Aunque los métodos reducen significativamente el número de comunicaciones, esta diferencia no se manifiesta notablemente en los tiempos de ejecución ilustrados, pues la red de interconexión usada en los sistemas

computacionales empleados en el marco de la tesis era muy rápida. Por tanto, cabe esperar que en sistemas computacionales con redes menos rápidas o en matrices de mayor dimensión, la ventaja de los métodos propuestos sea más notable y así, la diferencia entre sus tiempos de ejecución y los tiempos de ejecución obtenidos con `PDGEBRD` sea más expresiva.

7. La gran desventaja de los métodos está en el hecho de que no permitiesen el desarrollo de implementaciones secuenciales y paralelas orientadas a bloques matriciales y, por lo tanto, no permitiesen obtener las ventajas de la utilización de las rutinas de BLAS de nivel 3.

Una vez obtenida la matriz bidiagonal superior, el paso siguiente es calcular su DVS. Pero, como fue descrito en el capítulo 2, el problema de calcular la DVS de una matriz bidiagonal superior puede verse como el problema de calcular la DVP de una matriz tridiagonal simétrica. En ese sentido, en el capítulo 4 se han estudiado los conceptos del problema estándar del cálculo de la DVP de una matriz tridiagonal simétrica irreducida y los métodos numéricos que le están asociados. Además, se ha estudiado el método `zeroinNR` propuesto por Ralha en su tesis doctoral y se ha desarrollado su implementación secuencial en el entorno de LAPACK.

Al mismo tiempo, en el capítulo 4, se han descrito los resultados obtenidos con la implementación secuencial del método `zeroinNR` y se han contrastado esos resultados con los resultados obtenidos por la rutina `DSTEBZ` de LAPACK, la cual es la implementación secuencial del método de bisección, empleado en el cálculo de los valores propios de una matriz tridiagonal simétrica irreducida.

Las conclusiones más relevantes que se logran con el trabajo desarrollado respecto a la implementación secuencial del método `zeroinNR` son:

1. La implementación secuencial garantiza el cálculo de todos los valores propios de la matriz tridiagonal simétrica con elevada precisión. Además, fueron desarrollados extensos experimentos computacionales, empleando un elevado conjunto de matrices de prueba conocidas del estado del arte.

2. Los tiempos de ejecución de la implementación secuencial son extraordinariamente menores que los tiempos de ejecución obtenidos con la rutina `DSTEBZ`.

3. Dentro del estudio descrito en 1, se ha desarrollado un estudio exhaustivo con las rutinas de LAPACK dedicadas a la DVP de matrices tridiagonales simétricas y se han exhibido casos de prueba en los cuales las rutinas de LAPACK calculan valores propios incorrectos.

4. Además, se ha desarrollado una simulación de una implementación paralela del método de bisección con una distribución uniforme del intervalo que contiene todos los valores propios de la matriz por los distintos procesadores, y se puede concluir que este tipo de implementación paralela puede conducir a un mal balanceo de la carga, alcanzando tiempos de ejecución elevados.

5. El punto anterior pone en evidencia la necesidad de desarrollar una implementación paralela del método de bisección con una metodología distinta a la que está implementada en SCALAPACK.

Descritas las conclusiones del capítulo 4, el punto 5 anterior da paso al capítulo 5, donde se han descrito las metodologías empleadas en el desarrollo de implementaciones paralelas del método de bisección, con y sin comunicaciones.

Como fue estudiado en el capítulo 5, la implementación paralela del método de bisección puesta a disposición por SCALAPACK en la rutina `PDSTEBZ` necesita de comunicaciones. Además, en la referencia bibliográfica que soporta esa implementación son definidas tres condiciones obligatorias para que un método de división sea considerado correcto: todos los valores propios tienen que ser calculados una única vez; todos los valores propios son calculados con la cota de error definida por el usuario; y los valores propios calculados están expuestos según una cierta ordenación.

Intentando cumplir con las condiciones anteriores, Ralha propuso una metodología alternativa para el desarrollo de una implementación paralela del método de bisección, en la cual es empleado el método `zeroinNR`, adaptado a un proceso de división sin comunicaciones. En el capítulo 5 fueron estudiadas esas ideas y como aportación novedosa en el marco de la tesis, fue desarrollada, testada y estudiada esa implementación paralela.

Las conclusiones más relevantes que se logran con el trabajo desarrollado respecto a la implementación paralela del método `zeroinNR` sin comunicaciones son:

1. La implementación paralela desarrollada permite lograr tiempos de ejecución que son extraordinariamente menores que la rutina `PDSTEBZ`, pues con solo 2 procesadores se han obtenidos tiempos de ejecución que están un poco por debajo de los tiempos de ejecución de la rutina de SCALAPACK con 8 procesadores.

2. La implementación paralela desarrollada permite calcular todos los valores propios de la matriz tridiagonal simétrica irreducida con menos recursos que la rutina `PDSTEBZ`, cumpliendo con las tres condiciones para que el método sea correcto en sistemas computacionales cuyos procesadores tienen todos las mismas características.

3. Aunque todos los procesadores tengan que calcular el mismo número de valores propios de la matriz tridiagonal simétrica irreducida, se ha comprobado que el número de iteraciones empleadas es distinto de procesador a procesador y, por lo tanto, no hay un balanceo de la carga ideal. Sin embargo, cada procesador calcula su conjunto de $\frac{n}{p}$ valores propios de manera independiente y los tiempos de ejecución permiten obtener valores del *speed-up* que hasta 16 procesadores son los valores ideales, bajando a 30 cuando se emplean 32 procesadores. Además, el comportamiento del *speed-up* es semejante para todas las dimensiones de las matrices de prueba empleadas.

4. La implementación paralela del método es muy competitiva respecto a la rutina de SCALAPACK, de la misma manera que la implementación secuencial del método `zeroinNR` lo era respecto a la rutina `DSTEBZ` de LAPACK.

5. Mientras que el método de bisección implementado en SCALAPACK está basado en la división equitativa del intervalo inicial por todos los procesadores, el método propuesto por Ralha está basado en la división equitativa del número de valores propios por todos los procesadores. Se ha comprobado que la metodología propuesta en SCALAPACK no es la más adecuada para las matrices tridiagonales simétricas cuyos valores propios no están homogéneamente distribuidos, mientras que la metodología propuesta por Ralha sí que lo es. Sin embargo, pueden existir matrices tridiagonales simétricas irreducidas con distribuciones espectrales

distintas de las exhibidas por las matrices de prueba y para las cuales esta conclusión tenga que ser reformulada o incluso pueda no ser verdadera.

6. La implementación paralela desarrollada tiene la desventaja en la redundancia que ocurre en las computaciones iniciales dado que todos los procesadores tienen que calcular el mismo intervalo inicial, a partir del cual empiezan las iteraciones del método de bisección, hasta que lleguen a su correspondiente punto de separación. Sin embargo, la influencia negativa de estas computaciones no es problemática, pues el número de iteraciones que cada procesador tiene que calcular hasta llegar a su punto de separación es pequeño, cuando se compara con el número total de iteraciones utilizadas en el cálculo de sus $\frac{n}{p}$ valores propios.

Logrado el estudio del cálculo de la DVS de una matriz real densa, el paso siguiente en el trabajo desarrollado en el marco de la tesis fue pasar a su aplicación en la reducción de modelos de sistemas lineales de control. Para ello, el capítulo 6 describe algunos conceptos de la teoría de control y en particular, de la reducción de modelos de sistemas lineales.

Fueron estudiados los sistemas lineales de control y sus propiedades. Además, fue estudiado igualmente su importancia en la teoría de control y casi al terminar el capítulo 6 fue estudiado un método numérico para la reducción de modelos, el cual está basado en la transformación balanceada y que proporciona la diagonalización simultánea de los Gramianos de controlabilidad y de observabilidad. En su momento, el método estudiado necesita calcular la DVS del producto de dos matrices cuadradas, los factores de Cholesky de los dos Gramianos, pero donde su producto no debe ser calculado de manera explícita para no introducir cambios en el condicionamiento del problema. Por ese motivo, durante el cálculo de la DVS del producto de los factores de Cholesky puede ser aplicado el método propuesto por Golub, por Sølna y por van Dooren, para el cálculo de la matriz bidiagonal superior sin calcular explícitamente ese producto matricial. Las razones del empleo del método de Golub-Sølna-van Dooren fueron descritas y estudiadas por Mollar en su tesis doctoral.

Respecto al capítulo 6 se pueden presentar las dos siguientes observaciones:

A. Durante el estudio de la teoría de control y de la reducción de modelos fueron estudiadas algunas rutinas de la librería SLICOT y fueron

desarrollados algunos experimentos computacionales con esas rutinas, principalmente con la rutina `AB09AD` dedicada a la reducción de sistemas lineales de control y cuyos resultados son descritos en el apéndice "Resultados con SLICOT".

B. De la diversa bibliografía estudiada fueron descritos algunos casos de prueba para desarrollar futuros experimentos computacionales, pero que en el marco de la tesis no fueron tenidos en cuenta a la hora de estudiar las implementaciones del método de Golub-Sølna-van Dooren.

En el capítulo 7 fueron descritas las implementaciones secuenciales y paralelas del método de Golub-Sølna-van Dooren, dedicadas al cálculo de la bidiagonalización implícita del producto matricial. Como trabajo novedoso en el marco de la tesis fueron desarrolladas las implementaciones generales del método propuesto por Golub, por Sølna y por van Dooren, dando continuidad a la investigación desarrollada por Mollar en su tesis doctoral.

Las conclusiones más relevantes que se logran del trabajo desarrollado respecto a la bidiagonalización implícita del producto matricial son:

1. Todas las implementaciones desarrolladas presentan tiempos de ejecución que se consideran "buenos", teniendo en cuenta que no fueron contrastados con los tiempos de ejecución obtenidos por otro método numérico.

2. A medida que aumenta el número de matrices en el producto matricial o a medida que aumenta la dimensión de las matrices, los tiempos de ejecución obtenidos son muy elevados y, por tanto, en los experimentos computacionales realizados se han empleado matrices de prueba de pequeñas dimensiones. Así, existe la necesidad de estudiar el comportamiento de todas las implementaciones desarrolladas, empleando matrices de mayor dimensión y existe la necesidad de buscar aplicaciones donde sea evidente la ventaja de aplicar la versión general del método de Golub-Sølna-van Dooren.

3. Las implementaciones paralelas desarrolladas en el entorno de SCALAPACK presentan valores del *speed-up* que hasta 6 procesadores son valores cercanos a los valores ideales. Sin embargo, existe la necesidad de testar las implementaciones paralelas con más procesadores, pero en aplicaciones reales.

8.2. Futuras líneas de investigación

Tras el trabajo desarrollado en el marco de la tesis, se ha apreciado que algunos aspectos pueden dar origen a futuras líneas de investigación.

Respecto a la bidiagonalización de matrices reales densas, se plantea la posibilidad de adaptar el método de Barlow modificado en la reducción de matrices a la forma de matriz banda, siguiendo la misma línea de investigación desarrollada por Lang y sus colaboradores [Lang, 1997], [Großer y Lang, 1999], [Bischof *et al.*, 2000], [Großer y Lang, 2003], [Großer y Lang, 2005]. En este sentido, se puede desarrollar un estudio comparativo entre los tiempos de ejecución obtenidos por las implementaciones secuenciales y paralelas del nuevo método, y los correspondientes tiempos de ejecución obtenidos por las implementaciones del método propuesto por Lang y sus colaboradores.

Además, se puede intentar desarrollar una implementación secuencial y paralela del método de Barlow modificado, siguiendo las ideas de investigación desarrolladas por Bosner en su tesis doctoral, empleando rutinas de BLAS de nivel 2.5 y estudiar la posibilidad de introducir optimizaciones que permitan obtener tiempos de ejecución más competitivos con los tiempos de ejecución obtenidos con las rutinas de LAPACK y de SCALAPACK.

Dado que la mayor ventaja de los métodos alternativos de bidiagonalización estudiados en el marco de la tesis está en el cálculo de la DVS completa de matrices reales densas, se deben procurar encontrar aplicaciones en proyectos de investigación en los cuales se puedan aplicar los métodos estudiados, obteniendo resultados que refuerzan y comprueban su importancia en ese tipo de aplicaciones.

Respecto al método `zeroinNR` para el cálculo de los valores propios de matrices tridiagonales simétricas irreducidas, se puede desarrollar una nueva adaptación de este método, empleando las ideas expuestas en [Ralha y Campos, 2012], donde se produce el cambio entre el método de bisección y el método de Newton-Raphson cuando en la etapa de aislamiento se hayan encontrado intervalos que contienen un solo valor propio de la matriz y donde no hayan polos del polinomio Q_n calculado según (4.19).

Una vez desarrollada la nueva implementación del método `zeroinNR`, se puede hacer un estudio comparativo entre las dos implementaciones y estudiar en detalle las aportaciones de la nueva metodología en el método `zeroinNR`.

Al mismo tiempo, seguir con más experimentos computacionales empleando otros tipos de matrices de prueba y que puedan reforzar las conclusiones descritas en el capítulo 4. Además, con este conjunto de matrices de prueba se puede estudiar cual es el mejor momento para cambiar del método de bisección al método de Newton-Raphson.

Como fue observado en el capítulo 5, la implementación paralela desarrollada para el cálculo de los valores propios de la matriz tridiagonal simétrica irreducida sin comunicaciones fue testeada en sistemas computacionales homogéneos y, por lo tanto, queda la posibilidad de hacer pruebas en sistemas computacionales heterogéneos y además, se pueden explorar las ideas propuestas por Ralha en [Ralha, 2009], para el desarrollo de implementaciones secuenciales y paralelas del método de bisección, empleando el paradigma de la precisión mixta y comparar los resultados según la perspectiva presentada en [Langou *et al.*, 2006].

En el capítulo 7 se han estudiado los resultados obtenidos por las implementaciones secuenciales y paralelas del método de Golub-Sølna--van Dooren [Golub *et al.*, 2000], empleado en la bidiagonalización implícita del producto de matrices reales cuadradas. En cuanto al trabajo desarrollado, existe la necesidad de desarrollar nuevos experimentos computacionales con más procesadores, con matrices de prueba de mayor dimensión y en aplicaciones reales. Además, existe la necesidad de contrastar los resultados obtenidos con el método de Golub-Sølna-van Dooren con las correspondientes implementaciones secuenciales y paralelas de otro método, donde dos posibilidades son: emplear el método descrito en [Heath *et al.*, 1986] o emplear el método descrito en [Stewart, 1997].

Además, se puede desarrollar un estudio comparativo entre los resultados obtenidos por las implementaciones desarrolladas en el marco de la tesis y las implementaciones existentes en la librería SLICOT dedicadas a la reducción de modelos de sistemas lineales de control. Al mismo tiempo, queda abierta la posibilidad de hacer este estudio comparativo en aplicaciones dedicadas a la teoría de control.

Aunque todas las implementaciones y sus respectivos tiempos de ejecución obtenidos en el marco de la tesis hayan sido logrados empleando sistemas computacionales homogéneos y con procesadores del tipo CPU, no cabe la menor duda que para que toda la investigación desarrollada pueda surtir un mayor

impacto, hay que plantear su traslado a sistemas computacionales con procesadores gráficos y beneficiarse de las nuevas líneas de investigación que conllevan el empleo de procesadores del tipo GPU.

8.3. Elementos de carácter científico

Durante el desarrollo de la tesis se han obtenidos los siguientes elementos de carácter científico:

- *Parallel bidiagonalization of a dense matrix*, C. Campos, D. Guerrero, V. Hernández, R. Ralha, V International Workshop on Accurate Solution of Eigenvalue Problems, Hagen, Abstract Book, (June, 2004).
- *Algoritmos de altas prestaciones para la bidiagonalización de matrices densas*, C. Campos, D. Guerrero, V. Hernández, R. Ralha, XV Jornadas de Paralelismo, Almería, Libro de Actas, (Septiembre, 2004), pp. 78–83.
- *Uma experiência com o cluster SEARCH*, C. Campos, R. Ralha, Workshop para o Centro de Matemática da Universidade do Minho, Braga, (Abril, 2006).
- *A new bidiagonalization method*, C. Campos, D. Guerrero, V. Hernández, R. Ralha, Jornada INGRID'06, Braga, (November, 2006).
- *O cálculo de valores próprios de matrizes tridiagonais simétricas*, C. Campos, R. Ralha, 1.º Encontro de Investigadores do SEARCH, Universidade do Minho, Braga, (Fevereiro, 2007).
- *Parallel bidiagonalization of a dense matrix*, C. Campos, D. Guerrero, V. Hernández, R. Ralha, SIAM Journal on Matrix Analysis and Applications, 29 (July, 2007), pp. 826–837.
- *Towards a parallel code without communication for the eigenvalues of symmetric tridiagonals*, C. Campos, R. Ralha, V. Hernández, D. Guerrero, Parallel Matrix Algorithms and Applications, Neuchatel, Abstract Book, (June, 2008), pp. 14.

- *Do we need yet another code for symmetric tridiagonals?*, R. Ralha, C. Campos, VII International Workshop on Accurate Solution of Eigenvalue Problems, Dubrovnik, Abstract Book, (June, 2008), pp. 22.
- *Nuevos resultados en la bidiagonalización de matrices densas*, C. Campos, R. Ralha, V. Hernández, D. Guerrero, XIX Jornadas de Paralelismo, Castellón, (Septiembre, 2008).
- *O algoritmo da bisecção em aritmética de precisão variável*, R. Ralha, C. Campos, Seminário do Centro de Matemática da Universidade do Minho, (Fevereiro, 2009).
- *On the convergence of Newton's method for eigenvalues of symmetric tridiagonal matrices*, R. Ralha, C. Campos, artículo enviado (SIAM Journal on Matrix Analysis and Applications, July, 2012).
- *Algoritmos de alto rendimento para o cálculo da decomposição em valores singulares e a sua aplicação à redução de modelos de sistemas lineares de controlo*, C. Campos, 1.º Encontro de Doutorandos do Centro de Matemática da Universidade do Minho, Braga, (Maio, 2014).

Bibliografía

[Adams *et al.*, 1994] G. Adams, A. Bojanczyk, F. Luk, *Computing the PSVD of two 2x2 triangular matrices*, SIAM Journal on Matrix Analysis and Applications, 15 (1994), pp. 366–382.

[Akl, 1989] S. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall, 1989.

[Amdahl, 1967] G. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, AIFPS Conference Proceedings, 30 (1967), pp. 483–485.

[Anderson *et al.*, 1999] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, *LAPACK User's Guide*, SIAM, 1999.
`http://www.netlib.org/lapack/lug`
(accedido el 1/9/2014)

[Antonelli y Vömel, 2005] D. Antonelli, C. Vömel, *PDSYEVR. SCALAPACK's parallel MRRR algorithm for the symmetric eigenvalue problem*, Technical Report, University of California (Berkeley), Computer Science Division, UCB/CSD-05--1399, 2005.
(LAPACK Working Note #168)

[Arbenz, 1989] P. Arbenz, *Divide-and-conquer algorithms for the computation of the SVD of bidiagonal matrices*, Vector and Parallel Computing, (1989), pp. 1–10.

[Autonne, 1913] L. Autonne, *Sur les matrices hypohermitiennes et les unitairs*, Comptes Rendus de l'Academie des Sciences, 156 (1913), pp. 858–860.

[Badía y Vidal, 1998] J. Badía, A. Vidal, *Parallel bisection algorithms for solving the symmetric tridiagonal eigenproblem*, High Performance Algorithms for Structured Matrix Problems, Nova Science Publishers, (1998), pp. 91–107.

[Bai, 1992] Z. Bai, *The CSD, GSVD, Their Applications and Computations*, University of Minnesota, Institute for Mathematics and Its Applications, IMA Preprint Series 958, 1992.

[Bai y Demmel, 1993] Z. Bai, J. Demmel, *Computing the generalized singular value decomposition*, SIAM Journal on Scientific Computing, 14 (1993), pp. 1464–1486.
(LAPACK Working Note #46)

[Bai, 1994] Z. Bai, *A parallel algorithm for computing the generalized singular value decomposition*, Journal of Parallel and Distributed Computing, 20 (1994), pp. 280–288.

[Bai *et al.*, 2000] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: a Pratical Guide*, SIAM, 2000.

[Badía, 1996] J. Badía, *Algoritmos Paralelos para el Cálculo de los Valores Propios de Matrices Estructuradas*, Tesis Doctoral, Universidad Politécnica de Valencia, 1996.

[Ballard *et al.*, 2010] G. Ballard, J. Demmel, I. Dumitriu, *Minimizing Communication for Eigenproblems and the Singular Value Decomposition*, Technical Report, University

of California (Berkeley), Computer Science Division, UCB/EECS-2010-139, 2010.
(LAPACK Working Note #237)

[Barlow y Evans, 1977] R. Barlow, D. Evans, *A parallel organisation of the bisection algorithm*, The Computer Journal, 2 (1977), pp. 267–269.

[Barlow y Demmel, 1990] J. Barlow, J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, SIAM Journal on Numerical Analysis, 3 (1990), pp. 762–791.

[Barlow, 2001] J. Barlow, *More accurate bidiagonal reduction for computing the singular value decomposition*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 761–798.

[Barlow, 2003] J. Barlow, *A faster backward stable bidiagonalization procedure*, ETNA: Following the Flows of Numerical Analysis, Abstract, (2003).

[Barlow *et al.*, 2005] J. Barlow, N. Bosner, Z. Drmač, *A new stable bidiagonal reduction algorithm*, Linear Algebra and Its Applications, 397 (2005), pp. 35–84.

[Barnett, 1985] S. Barnett, *Matrices in Control Theory*, Krieger Pub Co, 1985.

[Barth *et al.*, 1967] W. Barth, R. Martin, J. Wilkinson, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, Numerische Mathematik, 9 (1967), pp. 386–393.

[Baserman y Weidner, 1992] A. Baserman, P. Weidner, *A parallel algorithm for determining all eigenvalues of large real symmetric tridiagonal matrices*, Parallel Computing, 18 (1992), pp. 1129–1141.

[Beltrami, 1873] E. Beltrami, *Sulle funzioni bilineari*, Giornale di Matematiche ud uso Degli Studenti Delle Universita, 11 (1873), pp. 98–106.

[Benner *et al.*, 1997] P. Benner, V. Mehrmann, V. Sima, S. van Huffel, A. Varga, *SLICOT - a subroutine library in systems and control theory*, Applied and Computational Control, Signals and Circuits, 1 (1997), pp. 499–539.

[Benner *et al.*, 1999a] P. Benner, J. Claver, E. Quintana-Ortí, *Parallel distributed solvers for large stable generalized Lyapunov equations*, Parallel Processing Letters, 9 (1999), pp. 147–158.

[Benner *et al.*, 1999b] P. Benner, E. Quintana-Ortí, G. Quintana-Ortí, *Balanced Truncation Model Reduction of Large-Scale Dense Systems on Parallel Computers*, Technical Report, Universität Bremen, 99-07, 1999.

[Bernstein, 1984] H. Bernstein, *An accelerated bisection method for the calculation of eigenvalue of a symmetric tridiagonal matrix*, Numerische Mathematik, 43 (1984), pp. 153–160.

[Bientinesi *et al.*, 2005] P. Bientinesi, I. Dhillon, R. van de Geijn, *A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations*, SIAM Journal on Scientific Computing, 27 (2005), pp. 43–66.

[Bischof y van Loan, 1987] C. Bischof, C. van Loan, *The WY representation of products of Householder matrices*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. 2–13.

[Bischof *et al.*, 2000] C. Bischof, B. Lang, X. Sun, *A framework for symmetric band reduction*, ACM Transactions on Mathematical Software, 26 (2000), pp. 581–601.

[Björck, 1967] A. Björck, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, BIT, 7 (1967), pp. 1–21.

[Björck y Paige, 1992] A. Björck, C. Paige, *Loss and recapture of orthogonality in the modified Gram-Schmidt algorithm*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 176–190.

[Björck, 1994] A. Björck, *Numerics of Gram-Schmidt orthogonalization*, Linear Algebra and Its Applications, 197/198 (1994), pp. 297–316.

[Blackford *et al.*, 1997] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley, *SCALAPACK User's Guide*, SIAM, 1997.
`http://www.netlib.org/scalapack/slug`
(accedido el 1/9/2014)

[Blas Forum, 2001] *BLAS Technical Forum*, 2001.
`http://www.netlib.org/blas/blast-forum`
(accedido el 1/9/2014)

[Bojanczyk *et al.*, 1991] A. Bojanczyk, M. Ewerbring, F. Luk, P. van Dooren, *An accurate product SVD algorithm,* Signal Processing, 25 (1991), pp. 189–201.

[Bosner y Drmač, 2005] N. Bosner, Z. Drmač, *On accuracy properties of one--sided bidiagonalization algorithm and its applications*, Proceedings of the Conference on Applied Mathematics and Scientific Computing, Abstract Book, (2005), pp. 141–150.

[Bosner, 2006] N. Bosner, *Fast Methods for Large Scale Singular Value Decomposition*, PhD Thesis, University of Zagreb, 2006.

[Bosner y Barlow, 2007] N. Bosner, J. Barlow, *Block and parallel versions of one-sided bidiagonalization*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 927–953.

[Brent *et al.*, 1983] R. Brent, F. Luk, C. van Loan, *Computation of the generalized singular value decomposition using mesh connected processors*, Real Time Signal Processing, VI (1983), pp. 66–71.

[Brockman *et al.*, 2013] P. Brockman, T. Carson, Y. Cheng, T. Elgindi, K. Jensen, X. Zhoun, M. Elgindi, *Homotopy Method for the Eigenvalues of Symmetric Tridiagonal Matrices*, Journal of Computational and Applied Mathematics, 237 (2013), pp. 644–653.

[Businger y Golub, 1965] P. Businger, G. Golub, *Linear least squares solutions by Householder transformations*, Numerische Mathematik, 7 (1965), pp. 269–276.

[Businger y Golub, 1969] P. Businger, G. Golub, *Algorithm 358: singular value decomposition of a complex matrix*, Communications of the ACM, 12 (1969), pp. 564–565.

[Campos, 2004] C. Campos, *Algoritmos de Altas Prestaciones para la Reducción de Matrices Densas a la Forma Bidiagonal Superior*, Trabajo de Investigación para la Obtención del Diploma de Estudios Avanzados, Universidad Politécnica de Valencia, 2004.

[Campos *et al.*, 2004] C. Campos, D. Guerrero, V. Hernández, R. Ralha, *Algoritmos de altas prestaciones para la bidiagonalización de matrices densas*, XV Jornadas de Paralelismo, Almería, Libro de Actas, (2004), pp. 78–83.

[Campos *et al.*, 2007] C. Campos, D. Guerrero, V. Hernández, R. Ralha, *Parallel bidiagonalization of a dense matrix*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 826–837.

[Campos *et al.*, 2008a] C. Campos, R. Ralha, V. Hernández, D. Guerrero, *Nuevos resultados en la bidiagonalización de matrices densas*, XIX Jornadas de Paralelismo, Castellón, 2008.

[Campos *et al.*, 2008b] C. Campos, R. Ralha, V. Hernández, D. Guerrero, *Towards a parallel code without communication for the eigenvalues of symmetric tridiagonals*, Parallel Matrix Algorithms and Applications, Neuchatel, Abstract Book, (2008), pp. 14.

[Chalmers y Tidmus, 1996] A. Chalmers, J. Tidmus, *Practical Parallel Processing: an Introduction to Problem Solving in Parallel*, International Thomson Computer Press, 1996.

[Chan, 1982a] T. Chan, *An improved algorithm for computing the singular value decomposition*, ACM Transactions on Mathematical Software, 8 (1982), pp. 72–83.

[Chan, 1982b] T. Chan, *Algorithm 581: an improved algorithm for computing the singular value decomposition*, ACM Transactions on Mathematical Software, 8 (1982), pp. 84–88.

[Choi *et al.*, 1992] J. Choi, J. Dongarra, R. Pozo, D. Walker, *SCALAPACK: a scalable linear algebra library for distributed memory concurrent computers*, Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Society Press, (1992), pp. 120–127.
(LAPACK Working Note #55)

[Choi *et al.*, 1995] J. Choi, J. Dongarra, D. Walker, *Parallel matrix transpose algorithms on distributed memory concurrent computers*, Parallel Computing, 21 (1995), pp. 1387–1405.
(LAPACK Working Note #65)

[Choi *et al.*, 1996a] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, R. Whaley, *The design and implementation of*

the SCALAPACK LU, QR, and Cholesky factorization routines, Scientific Programming, 5 (1996), pp. 173–184.
(LAPACK Working Note #80)

[Choi *et al.*, 1996b] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, R. Whaley, *SCALAPACK: a portable linear algebra library for distributed memory computers - design issues and performance*, Computer Physics Communications, 97 (1996), pp. 1–15.
(LAPACK Working Note #95)

[Choi *et al.*, 1996c] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, R. Whaley, *A proposal for a set of parallel basic linear algebra subprograms*, Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science, Lecture Notes in Computer Science, 1041 (1996), pp. 107–114.
(LAPACK Working Note #100)

[Claver *et al.*, 1999] J. Claver, M. Mollar, V. Hernández, *Parallel computation of the SVD of a matrix product*, Lecture Notes in Computer Science, 1697 (1999), pp. 388–395.

[Cuppen, 1981] J. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numerische Mathematik, 36 (1981), pp. 177–195.

[Daydé y Duff, 1999] M. Daydé, I. Duff, *The RISC BLAS: a blocked implementation of level 3 BLAS for RISC processors*, ACM Transactions on Mathematical Software, 25 (1999), pp. 316–340.

[de Moor, 1989] B. de Moor, *On the Structure and Geometry of the PSVD*, Manuscript NA-89-05, Stanford University, 1989.

[de Moor y Golub, 1989] B. de Moor, G. Golub, *Generalized Singular Value Decompositions: a Proposal for a Standardized Nomenclature*, Manuscript NA-89-04, Stanford University, 1989.

[Demmel *et al.*, 1988] J. Demmel, J. Croz, S. Hammarling, D. Sorensen, *Guidelines for the design of symmetric eigenroutines, SVD, and iterative refinement and condition estimation for linear systems*, Technical Report, Argonne National Laboratory, Mathematics and Computer Science, TM-111, 1988.
(LAPACK Working Note #4)

[Demmel y Kahan, 1988] J. Demmel, W. Kahan, *Computing small singular values of bidiagonal matrices with guaranteed high relative accuracy*, Technical Report, Argonne National Laboratory, Mathematics and Computer Science, TM-110, 1988.
(LAPACK Working Note #3)

[Demmel y McKenney, 1989] J. Demmel, A. McKenney, *A test matrix generation suite*, Technical Report, Argonne National Laboratory, Mathematics and Computer Science, P69-0389, 1989.
(LAPACK Working Note #9)

[Demmel y Kahan, 1990] J. Demmel, W. Kahan, *Accurate singular values of bidiagonal matrices*, SIAM Journal on Scientific and Statistical Computing, 11 (1990), pp. 873–912.

[Demmel *et al.*, 1991] J. Demmel, J. Dongarra, W. Kahan, *On designing portable high performance numerical libraries*, Technical Report, Center for Research on Parallel Computation, 91433, 1991.
(LAPACK Working Note #39)

[Demmel, 1992] J. Demmel, *The inherent inaccuracy of implicit tridiagonal QR*, Technical Report, Center for Research on

Parallel Computation, 92413, 1992.
(LAPACK Working Note #45)

[Demmel y Veselić, 1992] J. Demmel, K. Veselić, *Jacobi's method is more accurate than QR*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 1204–1243.

[Demmel y Gragg, 1993] J. Demmel, W. Gragg, *On computing accurate singular values and eigenvalues of matrices with acyclic graphs*, Linear Algebra and Its Applications, 185 (1993), pp. 203–217.

[Demmel y Li, 1993] J. Demmel, X. Li, *Faster numerical algorithms via exception handling*, IEEE Transactions on Computers, 43 (1993), pp. 983–992.
(LAPACK Working Note #59)

[Demmel *et al.*, 1993] J. Demmel, M. Heath, H. van der Vorst, *Parallel numerical linear algebra*, Technical Report, Center for Research on Parallel Computation, 93424, 1993.
(LAPACK Working Note #60)

[Demmel *et al.*, 1995] J. Demmel, I. Dhillon, H. Ren, *On the correctness of some bisection-like parallel eigenvalue algorithms in floating point arithmetic*, Electronic Transactions on Numerical Analysis, 3 (1995), pp. 116–149.

[Demmel, 1997] J. Demmel, *Applied Numerical Linear Algebra*, SIAM, 1997.

[Demmel, 1999a] J. Demmel, *Accurate SVDs of structured matrices*, SIAM Journal on Matrix Analysis and Applications, 21 (1999), pp. 562–580.
(LAPACK Working Note #130)

[Demmel *et al.*, 1999b] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, Z. Drmač, *Computing the singular value decomposition with high relative accuracy*, Linear

Algebra and Its Applications, 299 (1999), pp. 21–80.
(LAPACK Working Note #119)

[Demmel *et al.*, 2008a] J. Demmel, O. Marques, B. Parlett, C. Vömel, *A testing infrastructure for LAPACK's symmetric eigensolvers*, ACM Transactions on Mathematical Software, 35 (2008), pp. 1–13.
(LAPACK Working Note #182)

[Demmel *et al.*, 2008b] J. Demmel, O. Marques, B. Parlett, C. Vömel, *Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers*, SIAM Journal on Scientific Computing, 30 (2008), pp. 1508–1526.
(LAPACK Working Note #183)

[Demmel *et al.*, 2009] J. Demmel, M. Hoemmen, Y. Hida, E. Riedy, *Non--negative diagonals and high performance on low-profile matrices from Householder QR*, SIAM Journal on Scientific Computing, 31(2009), pp. 2832–2841.
(LAPACK Working Note #203)

[Dhillon, 1997] I. Dhillon, *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*, PhD Thesis, University of California, 1997.

[Dhillon y Parlett, 2004a] I. Dhillon, B. Parlett, *Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices*, Linear Algebra and Its Applications, 387 (2004), pp. 1–28.

[Dhillon y Parlett, 2004b] I. Dhillon, B. Parlett, *Orthogonal eigenvectors and relative gaps*, SIAM Journal on Matrix Analysis and Applications, 25 (2004), pp. 858–899.

[Dhillon *et al.*, 2006] I. Dhillon, B. Parlett, C. Vömel, *The design and implementation of the MRRR algorithm*, ACM Transactions on Mathematical Software, 32 (2006), pp. 533–560.
(LAPACK Working Note #162)

[Dongarra y Sorensen, 1987] J. Dongarra, D. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. 139–154.

[Dongarra *et al.*, 1988] J. Dongarra, J. Croz, S. Hammarling, R. Hanson, *An extended set of Fortran basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 14 (1988), pp. 1–17.

[Dongarra *et al.*, 1989] J. Dongarra, S. Hammarling, D. Sorensen, *Block reduction of matrices to condensed forms for eigenvalue computations*, Journal of Computational and Applied Mathematics, 27 (1989), pp. 215–227.

[Dongarra *et al.*, 1990] J. Dongarra, J. Croz, I. Duff, S. Hammarling, *A set of level 3 basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[Dongarra, 1991] J. Dongarra, *Workshop on the BLACS*, Technical Report, University of Tennessee, Computer Science Department, UT-CS-91-134, 1991.
(LAPACK Working Note #34)

[Dongarra y van de Geijn, 1993] J. Dongarra, R. van de Geijn, *Two dimensional basic linear algebra communication subprograms*, Environments and Tools for Parallel Scientific Computing, (1993), pp. 31–40.
(LAPACK Working Note #37)

[Dongarra y Walker, 1993] J. Dongarra, D. Walker, *The design of linear algebra libraries for high performance computer*, Technical Report, University of Tennessee, Computer Science Department, UT-CS-93-188, 1993.
(LAPACK Working Note #58)

[Dongarra y Whaley, 1995] J. Dongarra, R. Whaley, *A user's guide to the BLACS v1.1*, Technical Report, University of Tennessee, Computer Science Department, UT-CS-95-281,

1995.
(LAPACK Working Note #94)

[Dongarra *et al.*, 1998] J. Dongarra, I. Duff, D. Sorensen, H. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, SIAM, 1998.

[Dopico y Koev, 2006] F. Dopico, P. Koev, *Accurate symmetric rank revealing and eigendecompositions of symmetric structured matrices*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 1126–1156.

[Drmač, 1998] Z. Drmač, *Accurate computation of the product-induced singular value decomposition with applications*, SIAM Journal on Numerical Analysis, 35 (1998), pp. 1969–1994.

[Drmač, 1999] Z. Drmač, *A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm*, IMA Journal of Numerical Analysis, 19 (1999), pp. 191–213.

[Drmač y Veselić, 2007a] Z. Drmač, K. Veselić, *New fast and accurate Jacobi SVD algorithm: I*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 1322–1342.
(LAPACK Working Note #169)

[Drmač y Veselić, 2007b] Z. Drmač, K. Veselić, *New fast and accurate Jacobi SVD algorithm: II*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 1343–1362.
(LAPACK Working Note #170)

[Eckart y Young, 1936] C. Eckart, G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[Eckart y Young, 1939] C. Eckart, G. Young, *A principal axis transformation for non-Hermitian matrices*, Bulletin of American Mathematics Society, 45 (1939), pp. 118–121.

[Eisenstat y Ipsen, 1995] S. Eisenstat, I. Ipsen, *Relative perturbation techniques for singular value problems*, SIAM Journal on Numerical Analysis, 32 (1995), pp. 1972–1988.

[Eisenstat y Ipsen, 1998] S. Eisenstat, I. Ipsen, *Three absolute perturbation bounds for matrix eigenvalues imply relative bounds*, SIAM Journal on Matrix Analysis and Applications, 20 (1998), pp. 149–158.

[El-Rewini y Lewis, 1998] H. El-Rewini, T. Lewis, *Distributed and Parallel Computing*, Manning Pubns Co, 1998.

[Fernando y Hammarling, 1987] K. Fernando, S. Hammarling, *A Generalized Singular Value Decomposition for a Product of Two Matrices and Balanced Realization*, Technical Report, Numerical Algorithms Group, TR1/87, 1987.

[Fernando y Parlett, 1994] K. Fernando, B. Parlett, *Accurate singular values and differential QD algorithms*, Numerische Mathematik, 67 (1994), pp. 191–229.

[Fernando, 1998a] K. Fernando, *Accurately Counting Singular Values of Bidiagonal Matrices*, Technical Report, Numerical Algorithms Group, TR4/96 (NP3007), 1998.

[Fernando, 1998b] K. Fernando, *Accurately counting singular values of bidiagonal matrices and eigenvalues of skew-symmetric tridiagonal matrices*, SIAM Journal on Matrix Analysis and Applications, 20 (1998), pp. 373–399.

[Flynn, 1972] M. Flynn, *Some computer organizations and their effectiveness*, IEEE Transactions on Computers, 21 (1972), pp. 948–960.

[Forjaz y Ralha, 2001] M. Forjaz, R. Ralha, *An efficient parallel algorithm for the symmetric tridiagonal eigenvalue problem*, Lecture Notes in Computer Science, Springer-Verlag, (2001), pp. 369–379.

[Fortuna *et al.*, 1992] L. Fortuna, G. Nunnari, A. Gallo, *Model Order Reduction Techniques With Applications in Electrical Engineering*, Springer-Verlag, 1992.

[Foster, 1995] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995.

[Francis, 1961] J. Francis, *The QR transformation: a unitary analogue to the LR transformation - Part 1*, Computer Journal, 4 (1961), pp. 265–271.

[Francis, 1962] J. Francis, *The QR transformation - Part 2*, Computer Journal, 4 (1962), pp. 332–345.

[Garbow *et al.*, 1977] B. Garbow, J. Boyle, J. Dongarra, C. Moler, *Matrix eigensystem routines - EISPACK guide extension*, Lecture Notes in Computer Science, 51, 1977.

[Gates *et al.*, 2014] M. Gates, A. Haidar, J. Dongarra, *Accelerating computation of eigenvectors in the nonsymmetric eigenvalue problem*, Technical Report, University of Tennessee, Computer Science Division, UT-EECS-14-724, 2014.
(LAPACK Working Note #286)

[Givens, 1953] J. Givens, *A method of computing eigenvalues and eigenvectors suggested by classical results on symmetric matrices*, National Bureau of Standards Applied Mathematics Series, 29 (1953), pp. 117–122.

[Givens, 1954] J. Givens, *Numerical Computations of the Characteristic Values of a Real Symmetric Matrix*, Technical Report, Oak Ridge National Laboratory, ORNL-1574, 1954.

[Glover, 1984] K. Glover, *All optimal Hankel-norm approximations of linear multivariable systems and their L^∞-error bounds*, International Journal of Control, 39 (1984), pp. 1115–1193.

[Golub y Kahan, 1965] G. Golub, W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM Journal on Numerical Analysis, 2 (1965), pp. 205–224.

[Golub y Reinsch, 1970] G. Golub, C. Reinsch, *Singular value decomposition and least squares solution*, Numerische Mathematik, 14 (1970), pp. 403–420.

[Golub y van Loan, 1996] G. Golub, C. van Loan, *Matrix Computations*, The John Hopkins University Press, 3rd Edition, 1996.

[Golub *et al.*, 2000] G. Golub, K. Sølna, P. van Dooren, *Computing the SVD of a general matrix product/quotient*, SIAM Journal on Matrix Analysis and Applications, 22 (2000), pp. 1–19.

[Grama *et al.*, 2003] A. Grama, A. Gupta, G. Karypis, V. Kumar, *Introduction to Parallel Computing*, Addison Wesley, 2003.

[Green y Limebeer, 2000] M. Green, D. Limebeer, *Linear Robust Control*, Pearson Education Inc, 2000.

[Gries, 1981] D. Gries, *The Science of Programming*, Springer-Verlag, 1981.

[Gropp *et al.*, 1994] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming With the Message Passing Interface*, The MIT Press, 1994.

[Gropp y Lusk, 1996] W. Gropp, E. Lusk, *User's Guide for MPICH, a Portable Implementation of MPI*, Technical Report, Argonne National Laboratory, ANL-96/6, 1996.

[Gropp *et al.*, 1996] W. Gropp, E. Lusk, N. Doss, A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel Computing, 22 (1996), pp. 789–828.

[Großer y Lang, 1999] B. Großer, B. Lang, *Efficient parallel reduction to bidiagonal form*, Parallel Computing, 25 (1999), pp. 969–986.

[Großer y Lang, 2003] B. Großer, B. Lang, *An $O(n^2)$ algorithm for the bidiagonal SVD*, Linear Algebra and Its Applications, 358 (2003), pp. 45–70.

[Großer y Lang, 2005] B. Großer, B. Lang, *On symmetric eigenproblems for the bidiagonal SVD*, SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 599–620.

[Gu y Eisenstat, 1995a] M. Gu, S. Eisenstat, *A divide-and-conquer algorithm for the bidiagonal SVD*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 79–92.

[Gu y Eisenstat, 1995b] M. Gu, S. Eisenstat, *A divide-and-conquer algorithm for the symmetric eigenproblem*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 172–191.

[Guerrero *et al.*, 2002] D. Guerrero, V. Hernández, J. Román, *Parallel SLICOT model reduction routines: the Cholesky factor of Grammians*, 15th Triennial World Congress of the International Federation of Automatic Control, 15 (2002), pp. 1–6.

[Gustafson, 1988] J. Gustafson, *Reevaluating Amdahl's law*, Communications of the ACM, 31 (1988), pp. 532–533.

[Halmos, 1958] P. Halmos, *Finite Dimensional Vector Spaces*, Van Nostrand, New York, 1958.

[Hammarling, 1982] S. Hammarling, *Numerical solution of the stable, non--negative definite Lyapunov equation*, IMA Journal of Numerical Analysis, 2 (1982), pp. 303–323.

[Hanson y Lawson, 1969] R. Hanson, C. Lawson, *Extensions and applications of the Householder algorithm for solving linear least squares problems*, Mathematics of Computation, 23 (1969), pp. 787–812.

[Heath *et al.*, 1986] M. Heath, J. Laub, C. Paige, R. Ward, *Computing the singular value decomposition of a product of two*

matrices, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 1147–1159.

[Higham, 1991] N. Higham, *Algorithm 694: a collection of test matrices in Matlab*, ACM Transactions on Mathematical Software, 17 (1991), pp. 289–305.

[Higham, 2002a] N. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, 2002.

[Higham, 2002b] N. Higham, *The Matrix Computation Toolbox*, 2002. `http://www.ma.man.ac.uk/~higham/mctoolbox` (accedido el 1/9/2014)

[Hockney y Jesshop, 1982] R. Hockey, C. Jesshop, *Parallel Computers*, Adam Hilger Ltd, 1982.

[Horn y Johnson, 1990] R. Horn, C. Johnson, *Matrix Analysis*, Cambridge University Press, 1990.

[Howell *et al.*, 2008] G. Howell, J. Demmel, C. Fulton, S. Hammarling, K. Marmol, *Cache efficient bidiagonalization using BLAS 2.5 operators*, ACM Transactions on Mathematical Software, 34 (2008), pp. 1–33. (LAPACK Working Note #174)

[IEEE, 1985] IEEE, *IEEE Standard for Binary Floating Point Arithmetic*, ANSI/IEEE Std 754/1985, IEEE Computer Society, Los Alamitos, 1985.

[Intel, 2008] Intel, *Intel® MPI Benchmarks 3.2*, 2008.

[Ipsen y Jessup, 1990] I. Ipsen, E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM Journal Scientific and Statistical Computing, 11 (1990), pp. 203–229.

[Ipsen, 1997] I. Ipsen, *Computing an eigenvector with inverse iteration*, SIAM Review, 39 (1997), pp. 254–291.

[Ipsen, 1998] I. Ipsen, *Relative perturbation for matrix eigenvalues and singular values*, Acta Numerica, 7 (1998), pp. 151–201.

[Jacobi, 1848] C. Jacobi, *Über ein leichtes verfahren die in der theorie der sacularstorungen vorkommendern gleichungen numerisch aufzulosen*, Crelle's J., 30 (1848), pp. 51–94.

[Jessup y Ipsen, 1992] E. Jessup, I. Ipsen, *Improving the accuracy of inverse iteration*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 550–572.

[Kahan, 1966] W. Kahan, *Accurate Eigenvalues of a Symmetric Tridiagonal Matrix*, Technical Report, Stanford University, Computer Science Department, CS41, 1966.

[Kalman, 1963] R. Kalman, *Mathematical description of linear dynamical systems*, SIAM Journal on Control and Optimization, 1 (1963), pp. 152–192.

[Kalman *et al.*, 1969] R. Kalman, P. Falb, M. Arbib, *Topics in Mathematical System Theory*, McGraw-Hill, 1969.

[Koev, 2005] P. Koev, *Accurate eigenvalues and SVDs of totally nonnegative matrices*, SIAM Journal on Matrix Analysis and Applications, 27 (2005), pp. 1–23.

[Kogbetliantz, 1955] E. Kogbetliantz, *Solution of linear equations by diagonalization of coefficients matrix*, Quarterly of Applied Mathematics, 13 (1955), pp. 123–132.

[Kublanovskaya, 1962] V. Kublanovskaya, *On some algorithms for the solution of the complete eigenvalue problem*, USSR Computational Mathematics and Mathematical Physics, 3 (1962), pp. 637–657.

[Kumar *et al.*, 1994] V. Kumar, A. Grama, A. Gupta, G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, 1994.

[Lang, 1996] B. Lang, *Parallel reduction of banded matrices to bidiagonal form*, Parallel Computing, 22 (1996), pp. 1–18.

[Lang, 1997] B. Lang, *Efficient Algorithms for Reducing Banded Matrices to Bidiagonal and Tridiagonal Form*, Preprint BUGHW-SC 97/1, Universität GH Wuppertal, 1997.

[Langou *et al.*, 2006] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, J. Dongarra, *Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy*, Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, (2006), pp. 11–17. (LAPACK Working Note #175)

[Laub *et al.*, 1987] A. Laub, M. Heath, C. Paige, R. Ward, *Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms*, IEEE Transactions on Automatic Control, AC-32 (1987), pp. 115–122.

[Lawson *et al.*, 1979] C. Lawson, R. Hanson, D. Kincaid, F. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Transactions on Mathematical Software, 5 (1979), pp. 308–325.

[Lawson y Hanson, 1995] C. Lawson, R. Hanson, *Solving Least Squares Problems*, SIAM, 1995.

[Li y Rhee, 1989] T. Li, N. Rhee, *Homotopy algorithm for symmetric eigenvalue problems*, Numerische Mathematik, 55 (1989), pp. 265–280.

[Li *et al.*, 1991] T. Li, H. Zhang, H. Sun, *Parallel homotopy algorithm for symmetric tridiagonal eigenvalue problems*, SIAM Journal Scientific and Statistical Computing, 12 (1991), pp. 464–485.

[Li y Zeng, 1992a] T. Li, Z. Zeng, *Laguerre's iteration in solving the symmetric tridiagonal eigenproblem - revisited*, SIAM

Journal Scientific and Statistical Computing, 15 (1992), pp. 1145–1173.

[Li y Zeng, 1992b] T. Li, Z. Zeng, *Homotopy-determinant algorithm for solving nonsymmetric eigenvalue problems*, Mathematics of Computation, 59 (1992), pp. 483–502.

[Li *et al.*, 1992] T. Li, Z. Zeng, L. Cong, *Solving eigenvalue problems of real nonsymmetric matrices with real homotopies*, SIAM Journal on Numerical Analysis, 29 (1992), pp. 229–248.

[Li *et al.*, 1995] T. Li, N. Rhee, Z. Zeng, *An efficient and accurate parallel algorithm for the singular value problem of bidiagonal matrices*, Numerische Mathematik, 69 (1995), pp. 283–301.

[Lo *et al.*, 1987] S. Lo, B. Philippe, A. Sameh, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM Journal Scientific and Statistical Computing, 8 (1987), pp. 155–165.

[Lui *et al.*, 1997] S. Lui, H. Keller, T. Kwok, *Homotopy method for the large, sparse, real nonsymmetric eigenvalue problem*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 312–333.

[Oettli, 1995] M. Oettli, *The Homotopy Method Applied to the Symmetric Eigenproblem*, PhD Thesis, Swiss Federal Institute of Technology Zurich, 1995.

[Marques *et al.*, 2005a] O. Marques, B. Parlett, C. Vömel, *Subset computations with the MRRR algorithm*, Technical Report, University of California (Berkeley), Computer Science Division, UCB/CSD-05-1392, 2005.
(LAPACK Working Note #167)

[Marques *et al.*, 2005b] O. Marques, J. Riedy, C. Vömel, *Benefits of IEEE-754 features in modern symmetric tridiagonal eigensolvers*,

Technical Report, University of California (Berkeley), Computer Science Division, UCB/CSD-05-1414, 2005. (LAPACK Working Note #172)

[Martin y Tirado, 1997] I. Martin, F. Tirado, *Relationships between efficiency and execution time of full multigrid methods on parallel computers*, IEEE Transactions on Parallel and Distributed Systems, 8 (1997), pp. 562–573.

[Mollar y Hernández, 1996] M. Mollar, V. Hernández, *Computing the singular values of the product of two matrices in distributed memory processors*, Proceedings of 4th Euromicro Workshop on Parallel and Distributed Computation, Braga, (1996).

[Mollar y Hernández, 1998] M. Mollar, V. Hernández, *A parallel implementation of the singular value decomposition of the product of triangular matrices*, 1st NICONET Workshop, Valencia, (1998).

[Mollar, 2003] M. Mollar, *Algoritmos Paralelos en Problemas de Control y Tratamiento de la Señal: Computación de los Valores Singulares del Producto de Dos Matrices y Sus Aplicaciones*, Tesis Doctoral, Universidad Politécnica de Valencia, 2003.

[Moore, 1981] B. Moore, *Principal component analysis in linear systems: controlability, observability, and model reduction*, IEEE Transactions on Automatic Control, 26 (1981), pp. 17–32.

[Pacheco, 1997] P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann, 1997.

[Pacheco, 1998] P. Pacheco, *A User's Guide to MPI*, University of San Francisco, 1998.

[Paige y Saunders, 1981] C. Paige, M. Saunders, *Towards a generalized singular value decomposition*, SIAM Journal on Numerical Analysis, 18 (1981), pp. 398–405.

[Parlett, 1964] B. Parlett, *Laguerre's method applied to the matrix eigenvalue problem*, Mathematics of Computation, 18 (1964), pp. 464–485.

[Parlett, 1971] B. Parlett, *Analysis of algorithms for reflections in bisectors*, SIAM Review, 13 (1971), pp. 197–208.

[Parlett y Nour-Omid, 1985] B. Parlett, B. Nour-Omid, *The use of a refined error bound when updating eigenvalues of tridiagonals*, Linear Algebra and Its Applications, 68 (1985), pp. 179–219.

[Parlett y Dhillon, 1997] B. Parlett, I. Dhillon, *Fernando's solution to Wilkinson's problem: an application of double factorization*, Linear Algebra and Its Applications, 267 (1997), pp. 247–279.

[Parlett, 1998] B. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, 1998.

[Parlett y Dhillon, 2000] B. Parlett, I. Dhillon, *Relatively robust representations for symmetric tridiagonals*, Linear Algebra and Its Applications, 309 (2000), pp. 121–151.

[Parlett y Marques, 2002] B. Parlett, O. Marques, *An implementation of the dqds algorithm (positive case)*, Linear Algebra and Its Applications, 309 (2002), pp. 217–259.
(LAPACK Working Note #155)

[Parlett y Dhillon, 2004a] B. Parlett, I. Dhillon, *Orthogonal eigenvectors and relative gaps*, SIAM Journal on Matrix Analysis and Applications, 25 (2004), pp. 858–899.

[Parlett y Dhillon, 2004b] B. Parlett, I. Dhillon, *Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices*, Linear Algebra and Its Applications, 387 (2004), pp. 1–28.

[Penzl, 1999] T. Penzl, *Algorithms for Model Reduction of Large Dynamical Systems*, Preprint SFB393/99-40, Technische Universität Chemnitz, 1999.

[Pernebo y Silverman, 1982] L. Pernebo, L. Silverman, *Model reduction via balanced state space representations*, IEEE Transactions on Automatic Control, AC-27 (1982), pp. 382–387.

[Peters y Wilkinson, 1971] G. Peters, J. Wilkinson, *The calculation of specified eigenvectors by inverse iteration, contribution II/18*, Volume II of Handbook of Automatic Computation, (1971), pp. 418–439.

[Petitet *et al.*, 2008] A. Petitet, R. Whaley, J. Dongarra, A. Cleary, *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*, Innovative Computing Laboratory, 2008. `http://www.netlib.org/benchmark/hpl/index.html` (accedido el 1/9/2014)

[Petkov *et al.*, 1991] P. Petkov, N. Christov, M. Konstantinov, *Computational Methods for Linear Control Systems*, Prentice-Hall, 1991.

[Philippe *et al.*, 1987] B. Philippe, S. Lo, A. Sameh, *A multiprocessor algorithm for the symmetric eigenproblem*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. 155–165.

[Quinn, 2004] M. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw Hill, 2004.

[Ralha, 1990] R. Ralha, *Parallel Computation of Eigenvalues and Eigenvectors Using Occam and Transputers*, PhD Thesis, University of Southampton, 1990.

[Ralha, 1993] R. Ralha, *Parallel solution of the symmetric tridiagonal eigenvalue problem on a transputer network*, SEMNI93, 2 (1993), pp. 1026–1033.

[Ralha, 1994] R. Ralha, *A new algorithm for singular value decompositions*, Proceedings of 3rd Euromicro Workshop on Parallel and Distributed Processing, IEEE Computer Society Press, (1994), pp. 240–244.

[Ralha y Mackiewicz, 1996] R. Ralha, A. Mackiewicz, *An efficient algorithm for the computation of singular values*, Proceedings of 3rd International Congress of Numerical Methods in Engineering, (1996), pp. 1371–1380.

[Ralha, 2000] R. Ralha, *An accurate bidiagonal reduction for the computation of singular values*, III International Workshop on Accurate Solution of Eigenvalue Problems, Hagen, Best Poster, (2000).

[Ralha, 2003] R. Ralha, *One-sided reduction to bidiagonal form*, Linear Algebra and Its Applications, 358 (2003), pp. 221–238.

[Ralha, 2006] R. Ralha, *A parallel algorithm, without communication, for the computation of the eigenvalues of symmetric tridiagonal matrices*, VI International Workshop on Accurate Solution of Eigenvalue Problems, Pennsylvania, Abstract Book, (2006), pp. 20–22.

[Ralha y Campos, 2008] R. Ralha, C. Campos, *Do we need yet another code for symmetric tridiagonals?*, VII International Workshop on Accurate Solution of Eigenvalue Problems, Dubrovnik, Abstract Book, (2008), pp. 22.

[Ralha, 2009] R. Ralha, *Perturbation splitting for more accurate eigenvalues*, SIAM Journal on Matrix Analysis and Applications, 31 (2009), pp. 75–91.

[Ralha, 2011] R. Ralha, *Reliable eigenvalues of symmetric tridiagonals*, SIAM Journal on Matrix Analysis and Applications, 32 (2011), pp. 1524–1536.

[Ralha y Campos, 2012] R. Ralha, C. Campos, *On the convergence of Newton's method for eigenvalues of symmetric tridiagonal matrices*, artículo enviado (SIAM Journal on Matrix Analysis and Applications, 2012).

[Rutter, 1994] J. Rutter, *A Serial Implementation of Cuppen's Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem*, Technical Report, Rice University, Center for Research on Parallel Computation, CRPC–TR94443, 1994.

[Saad y Schultz, 1989] Y. Saad, M. Schultz, *Data communication in parallel architectures*, Parallel Computing, 11 (1989), pp. 131–150.

[Saad, 2011] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, SIAM, 2nd Edition, 2011.

[Schreiber y Parlett, 1988] R. Schreiber, B. Parlett, *Block reflectors: theory and computation*, SIAM Journal on Numerical Analysis, 25 (1988), pp. 189–205.

[Schreiber y van Loan, 1989] R. Schreiber, C. van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM Journal Scientific and Statistical Computing, 10 (1989), pp. 53–57.

[Simon, 1989] H. Simon, *Bisection in not optimal on vector processors*, SIAM Journal Scientific and Statistical Computing, 10 (1989), pp. 205–209.

[Skelton, 1988] R. Skelton, *Dynamic Systems Control*, John Wiley and Sons, 1988.

[Smith *et al.*, 1976] B. Smith, J. Boyle, J. Dongarra, B. Garbow, Y. Ikebe, V. Klema, C. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, 6, 1976.

[Snir *et al.*, 1996] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI: The Complete Reference*, The MIT Press, 1996.

[Stanley, 1997] K. Stanley, *Execution Time of Symmetric Eigensolvers*, PhD Thesis, University of California, 1997.

[Stewart, 1982] G. Stewart, *Computing the CS-decomposition of a partitioned orthonormal matrix*, Numerische Mathematik, 40 (1982), pp. 297–306.

[Stewart, 1993] G. Stewart, *On the early history of the singular value decomposition*, SIAM Review, 35 (1993), pp. 551–566.

[Stewart, 1997] D. Stewart, *A new algorithm for the SVD of a long product of matrices and the stability of products*, Electronic Transactions on Numerical Analysis, 5 (1997), pp. 29–47.

[Tisseur y Dongarra, 1998] F. Tisseur, J. Dongarra, *Parallelizing the divide and conquer algorithm for the symmetric tridiagonal eigenvalue problem on distributed memory architectures*, SIAM Journal on Scientific Computing, 20 (1998), pp. 2223–2236.
(LAPACK Working Note #132)

[Trefftz *et al.*, 1995] C. Trefftz, P. McKinley, T. Li, Z. Zeng, *A parallel algorithm for the singular value problem in bidiagonal matrices*, Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing, (1995), pp. 62–67.

[van der Vorst y Golub, 1997] H. van der Vorst, G. Golub, *150 Years Old and Still Alive: Eigenproblems*, The State of the Art in Numerical Analysis, Claredon Press, 1997.

[van Huffel y Sima, 2002] S. van Huffel, V. Sima, *SLICOT and control systems numerical software packages*, Proceedings of IEEE International Conference Control Applications and International Symposium on Computer Aided Control Systems Design, (2002), pp. 39–44.

[van Loan, 1976] C. van Loan, *Generalizing the singular value decomposition*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 76–83.

[Volkov y Demmel, 2008a] V. Volkov, J. Demmel, *Using GPUs to accelerate the bisection algorithm for finding eigenvalues of symmetric tridiagonal matrices*, Technical Report, University of California (Berkeley), Computer Science Division, UCB/EECS-2007-179, 2008.
(LAPACK Working Note #197)

[Volkov y Demmel, 2008b] V. Volkov, J. Demmel, *LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs*, Technical Report, University of California (Berkeley), Computer Science Division, UCB/EECS-2008-49, 2008.
(LAPACK Working Note #202)

[Vömel, 2010] C. Vömel, *SCALAPACK's MRRR algorithm*, ACM Transactions on Mathematical Software, 37 (2010), pp. 1–35.
(LAPACK Working Note #195)

[webBLACS, 1997] `http://www.netlib.org/blacs`, 1997.
(accedido el 1/9/2014)

[webBLAS, 2011] `http://www.netlib.org/blas`, 2011.
(accedido el 1/9/2014)

[webDMat, 2014] http://www.math.uminho.pt, 2014.
(accedido el 1/9/2014)

[webFortran90, 2014] http://www.nag.co.uk/sc22wg5, 2014.
(accedido el 1/9/2014)

[webGigabit, 2014] http://www.ieee802.org/3/, 2014.
(accedido el 1/9/2014)

[webGRyCAP, 2014] http://www.grycap.upv.es, 2014.
(accedido el 1/9/2014)

[webIntel®MKL, 2013] https://software.intel.com/en-us/intel-mkl, 2013.
(accedido el 1/9/2014)

[webKefren, 2014] http://www.grycap.upv.es/view.php/GRyCAP/ /Infraestructura/Equipos/kefren, 2014.
(accedido el 1/9/2014)

[webLAPACK, 2013] http://www.netlib.org/lapack, 2013.
(accedido el 1/9/2014)

[webMPI, 2014] http://www.mcs.anl.gov/research/projects/mpi, 2014.
(accedido el 1/9/2014)

[webMyrinet, 2009] http://www.myri.com/myrinet/overview, 2009.
(accedido el 1/7/2009)

[webOdin, 2014] http://www.grycap.upv.es/view.php/GRyCAP/ /Infraestructura/Equipos/odin, 2014.
(accedido el 1/9/2014)

[webPBLAS, 1995] http://www.netlib.org/scalapack/pblas_qref.html, 1995.
(accedido el 1/9/2014)

[webSCALAPACK, 2012] http://www.netlib.org/scalapack, 2012.
(accedido el 1/9/2014)

[webScali, 2010] `http://www.scali.com`, 2010.
(accedido el 1/7/2010)

[webSearch, 2014] `http://search.di.uminho.pt`, 2014.
(accedido el 1/9/2014)

[webSLICOT, 2014] `http://www.slicot.org`, 2014.
(accedido el 1/9/2014)

[webUM, 2014] `http://www.uminho.pt`, 2014.
(accedido el 1/9/2014)

[webUPV, 2014] `http://www.upv.es`, 2014.
(accedido el 1/9/2014)

[Whaley, 1994] R. Whaley, *Basic linear algebra communication subprograms: analysis and implementation across multiple parallel architectures*, Technical Report, University of Tennessee, Computer Science Department, UT-CS-94--234, 1994.
(LAPACK Working Note #73)

[Wilkinson, 1965] J. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

[Wilkinson, 1968] J. Wilkinson, *Global convergence of the tridiagonal QR algorithm with origin shifts*, Linear Algebra and Its Applications, 1 (1968), pp. 409–420.

[Wilkinson y Reinsch, 1971] J. Wilkinson, C. Reinsch, *Handbook for Automatic Computation, Vol. II, Linear Algebra*, Springer-Verlag, 1971.

[Wilson y Mishra, 1979] D. Wilson, R. Mishra, *Optimal reduction of multivariable systems*, International Journal of Control, 29 (1979), pp. 267–278.

[Willems *et al.*, 2006] P. Willems, B. Lang, C. Vömel, *Computing the bidiagonal SVD using multiple relatively robust*

representations, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 907–926.
(LAPACK Working Note #166)

[Willems, 2010] P. Willems, *On MR^3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and the Bidiagonal SVD*, PhD Thesis, University of Wuppertal, 2010.

[Zeng, 1991] Z. Zeng, *Homotopy-determinant Method for Solving Matrix Eigenvalue Problems and Its Parallelizations*, PhD Thesis, Michigan State University, 1991.

Esta hoja ha sido dejada en blanco de forma intencionada.

Apéndice **A**

BLAS, LAPACK y SCALAPACK

EN este apéndice se hace una breve descripción de las librerías numéricas BLAS, LAPACK y SCALAPACK, y dado que la librería SCALAPACK necesita de rutinas de la librería BLACS para las comunicaciones, el apéndice cierra con un resumen de la librería BLACS.

A.1. BLAS

BLAS (*Basic Linear Algebra Subprograms*) constituye una especificación para las operaciones básicas de Álgebra Lineal con vectores y con matrices [Lawson *et al.*, 1979], [Dongarra *et al.*, 1988], [webBLAS, 2011].

Las rutinas de BLAS están organizadas según tres niveles, los cuales se caracterizan según el tipo de objetos con los que se trabaja:

- **Nivel 1**: incluye las rutinas que realizan operaciones del tipo **vector--vector**. Son rutinas que realizan $\mathcal{O}(n)$ operaciones sobre $\mathcal{O}(n)$ datos.
- **Nivel 2**: incluye las rutinas que realizan operaciones del tipo **matriz--vector**. Son rutinas que realizan $\mathcal{O}(n^2)$ operaciones sobre $\mathcal{O}(n^2)$ datos.

- **Nivel 3**: incluye las rutinas que realizan operaciones del tipo **matriz--matriz**. Son rutinas que realizan $\mathcal{O}\left(n^3\right)$ operaciones sobre $\mathcal{O}\left(n^2\right)$ datos.

Las rutinas de BLAS de nivel 3 son las que permiten lograr mejores prestaciones, ya que se realizan más operaciones que accesos a la memoria. Sin embargo, en las rutinas de nivel 1 y de nivel 2, el número de operaciones es sencillamente igual al número de accesos a la memoria y, por tanto, no se puede superar la velocidad de acceso a la memoria. Además, la jerarquía de memorias supone la existencia de un *cuello de botella* entre la memoria y la CPU.

A.1.1. Nomenclatura

Las rutinas de BLAS tienen una nomenclatura propia, la cual intenta ofrecer la mayor información posible con la limitación de los 6 caracteres impuesta en la identificación de rutinas codificadas en Fortran 77.

Las rutinas de BLAS de nivel 1, con la excepción de la rutina `IxAMAX` , tienen la grafía

XZZZZZ

donde:

- X − es el identificador del tipo de datos con los cuales trabaja la rutina y sigue la nomenclatura:
  - S − para números reales en simple precisión.
  - D − para números reales en doble precisión.
  - C − para números complejos en simple precisión.
  - Z − para números complejos en doble precisión.
- ZZZZZ − es el identificador del tipo de operación que realiza la rutina.

Las rutinas de BLAS de nivel 2 y de nivel 3 tienen la grafía

XYYZZZ

donde:

- X − sigue siendo el identificador del tipo de datos con los cuales trabaja la rutina.
- YY − es el identificador del tipo de matrices con las cuales trabaja la rutina y sigue la nomenclatura:
  - GE − matriz general.
  - GB − matriz general banda.
  - SY − matriz simétrica.
  - SB − matriz simétrica banda.
  - SP − matriz simétrica empaquetada.
  - HE − matriz hermítica (compleja).
  - HB − matriz hermítica banda (compleja).
  - HP − matriz hermítica empaquetada (compleja).
  - TR − matriz triangular.
  - TB − matriz triangular banda.
  - TP − matriz triangular empaquetada.
- ZZZ − es el identificador del tipo de operación que realiza la rutina.

A.2. LAPACK

LAPACK (*Linear Algebra Package*) es una librería estándar dedicada a resolver problemas de Álgebra Lineal [Anderson *et al.*, 1999], [webLAPACK, 2013].

LAPACK permite resolver **sistemas de ecuaciones lineales**, **problemas de mínimos cuadrados**, **problemas de valores propios** y **problemas de valores singulares**. También permite calcular algunas **descomposiciones matriciales** (LU, Cholesky, QR, Schur,...) y estimar **números de condición**.

La librería está estructurada en tres tipos de rutinas:

- **Rutinas driver**: permiten resolver problemas completos de Álgebra Lineal, como por ejemplo, la resolución de sistemas de ecuaciones lineales.

- **Rutinas computacionales**: permiten realizar distintas tareas computacionales, como por ejemplo, el cálculo de la descomposición LU de una matriz.
- **Rutinas auxiliares**: permiten realizar subtareas y/o computaciones de bajo nivel, como por ejemplo, el cálculo de una norma matricial.

Las rutinas driver resuelven problemas completos de Álgebra Lineal recurriendo a llamadas de rutinas computacionales y las rutinas auxiliares implementan pequeñas tareas que son necesarias para las otras rutinas y que no están en la librería BLAS.

Hay que añadir que LAPACK ha sido diseñada para obtener altas prestaciones y sus rutinas confían a BLAS el peso de la portabilidad y de las buenas prestaciones. Así, es del todo conveniente tener una versión de BLAS optimizada para la arquitectura en que se vaya a utilizar LAPACK, pues un BLAS mal instalado puede provocar malas prestaciones en LAPACK, dado que las rutinas de LAPACK están por encima de las rutinas de BLAS.

LAPACK utiliza todos los niveles de BLAS, pero para aprovechar mejor la localidad de los datos y así conseguir mejores prestaciones, sus rutinas implementan algoritmos orientados a bloques matriciales con el objetivo de realizar $\mathcal{O}\left(n^3\right)$ operaciones sobre $\mathcal{O}\left(n^2\right)$ datos.

Al mismo tiempo, la gran mayoría de las rutinas de LAPACK tienen uno o más parámetros de afinado, lo que permite optimizar LAPACK para arquitecturas con muy diferentes tamaños de memoria y de caché. Estos parámetros dependen directamente de la arquitectura y por ejemplo, uno de ellos es el tamaño del bloque especificado por la rutina `ILAENV`, el cual suele ser definido durante la instalación de LAPACK.

A.2.1. Nomenclatura

Como en BLAS, las rutinas de LAPACK tienen una nomenclatura propia con la grafía

XYYZZZ

donde:

- X − sigue siendo el identificador del tipo de datos con los cuales trabaja la rutina, con la nomenclatura:

  - S − para números reales en simple precisión.
  - D − para números reales en doble precisión.
  - C − para números complejos en simple precisión.
  - Z − para números complejos en doble precisión.

- YY − sigue siendo el identificador del tipo de matrices con las cuales trabaja la rutina, con la nomenclatura:

  - BD − matriz bidiagonal.
  - GB − matriz general banda.
  - GE − matriz general.
  - GG − matriz general (para problemas generalizados).
  - GT − matriz general tridiagonal.
  - HB − matriz hermítica banda (compleja).
  - HE − matriz hermítica (compleja).
  - HG − matriz Hessenberg superior (para problemas generalizados).
  - HP − matriz hermítica empaquetada (compleja).
  - HS − matriz Hessenberg superior.
  - OP − matriz ortogonal empaquetada (real).
  - OR − matriz ortogonal (real).
  - PB − matriz simétrica o hermítica banda positiva definida.
  - PO − matriz simétrica o hermítica positiva definida.
  - PP − matriz simétrica o hermítica positiva definida empaquetada.
  - PT − matriz simétrica o hermítica tridiagonal positiva definida.
  - SB − matriz simétrica banda (real).
  - SP − matriz simétrica empaquetada.
  - ST − matriz simétrica tridiagonal (real).
  - SY − matriz simétrica.
  - TB − matriz triangular banda.

  - TG − matriz triangular (para problemas generalizados).
  - TP − matriz triangular empaquetada.
  - TR − matriz triangular.
  - TZ − matriz trapezoidal.
  - UN − matriz unitaria (compleja).
  - UP − matriz unitaria empaquetada (compleja).

- ZZZ − sigue siendo el identificador del tipo de operación que realiza la rutina.

De acuerdo con la nomenclatura, la rutina `DGEBRD`, por ejemplo, es la rutina que reduce una matriz general (`GE`) en doble precisión (`D`) a la forma bidiagonal (`BRD`).

En las rutinas auxiliares los caracteres YY son `LA`.

A.2.2. Esquemas de almacenamiento

Además de permitir el uso de muchos y variados tipos de matrices, LAPACK permite también utilizar distintos esquemas de almacenamiento para conseguir un ahorro en cuanto a espacio de memoria y a coste de las operaciones. Sin embargo, LAPACK no permite el uso de matrices dispersas y sus correspondientes esquemas de almacenamiento.

Seguidamente se describen los esquemas de almacenamiento permitidos por LAPACK.

Almacenamiento convencional

El almacenamiento usual de LAPACK es el almacenamiento convencional y sigue el típico almacenamiento implementado por Fortran, o sea, las matrices o los vectores son almacenados en memoria por columnas (Tabla A.1).

En el caso de las matrices triangulares superiores o inferiores, solamente son almacenados en memoria los elementos de la parte triangular, procediendo a su almacenamiento por columnas consecutivas (Tabla A.2).

Los valores indicados con '$*$' son los elementos que no necesitan ser inicializados, pues las rutinas de LAPACK no acceden a ellos.

Matriz general A	**Almacenamiento en memoria**
$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

Tabla A.1: Almacenamiento de una matriz general.

Matriz triangular A	**Almacenamiento en memoria**
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{bmatrix}$
$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$	$\begin{bmatrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$

Tabla A.2: Almacenamiento de matrices triangulares.

El esquema de almacenamiento anterior sólo tiene ventajas para las matrices simétricas o hermíticas, ya que no es necesario tener almacenada toda la matriz. En estos dos casos, sólo hay que almacenar los elementos de la parte triangular pues los restantes elementos son accesibles de manera implícita (Tabla A.3).

Almacenamiento empaquetado

El almacenamiento convencional, sin embargo, para las matrices triangulares simétricas o hermíticas supone un gasto de memoria adicional, ya que, como se puede observar en las Tablas A.2 y A.3, hay elementos que no son utilizados por las rutinas de LAPACK. En estos casos, para conseguir un ahorro de memoria, se puede almacenar la matriz de forma empaquetada, almacenando todos sus elementos en un vector unidimensional, recorriendo las sucesivas columnas de la matriz (Tabla A.4).

Matriz hermítica A	**Almacenamiento en memoria**
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \overline{a_{12}} & a_{22} & a_{23} & a_{24} \\ \overline{a_{13}} & \overline{a_{23}} & a_{33} & a_{34} \\ \overline{a_{14}} & \overline{a_{24}} & \overline{a_{34}} & a_{44} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{bmatrix}$
$\begin{bmatrix} a_{11} & \overline{a_{21}} & \overline{a_{31}} & \overline{a_{41}} \\ a_{21} & a_{22} & \overline{a_{32}} & \overline{a_{42}} \\ a_{31} & a_{32} & a_{33} & \overline{a_{43}} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$	$\begin{bmatrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$

Tabla A.3: Almacenamiento de matrices simétricas o hermíticas.

Matriz triangular A	**Almacenamiento en memoria**
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & a_{22} & a_{13} & a_{23} & a_{33} & a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix}$
$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{41} & a_{22} & a_{32} & a_{42} & a_{33} & a_{43} & a_{44} \end{bmatrix}$

Tabla A.4: Almacenamiento de matrices triangulares empaquetadas.

Almacenamiento de matrices banda

Las matrices banda son las matrices en las que los elementos fuera de ciertas diagonales son todos nulos.

Una matriz banda con kl diagonales por debajo de la diagonal principal (subdiagonales) y con ku diagonales por encima de la diagonal principal (superdiagonales), puede ser almacenada de forma compacta en una matriz con $kl + ku + 1$ filas y n columnas (orden de la matriz). De hecho, las columnas de la matriz banda se almacenan en columnas de la matriz empaquetada, mientras que las diagonales de la matriz banda se almacenan en las filas de la matriz

empaquetada.

Hay que tener en cuenta que este esquema de almacenamiento se debe utilizar solamente en los casos en que kl y ku son mucho menores que n. Sin embargo, las rutinas de LAPACK que utilizan este tipo de matrices permiten cualquier valor de kl y ku (Tabla A.5).

Matriz banda A	**Almacenamiento en memoria**
$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} \\ 0 & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix}$	$\begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{bmatrix}$

Tabla A.5: Almacenamiento de matrices banda.

Las matrices triangulares banda son almacenadas de la misma manera con:

- $kl = 0$ para matrices triangulares superiores.
- $ku = 0$ para matrices triangulares inferiores.

En el caso de las matrices simétricas o hermíticas, solo se almacenan los elementos de la parte triangular (Tabla A.6).

Almacenamiento de matrices tridiagonales y bidiagonales

Las matrices tridiagonales no simétricas son las matrices en las que los elementos de la diagonal principal, de la primera subdiagonal y de la primera superdiagonal, pueden ser no nulos. En el caso de las matrices tridiagonales simétricas, la subdiagonal y la superdiagonal son iguales.

Las matrices bidiagonales son las matrices en las que los elementos de la diagonal principal y de una de las primeras diagonales (superior o inferior), pueden ser no nulos.

Para almacenar matrices tridiagonales no simétricas, LAPACK almacena la diagonal principal en un vector de n (orden de la matriz) elementos y las otras diagonales en dos vectores de $n - 1$ elementos.

Matriz hermítica banda A	**Almacenamiento en memoria**
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ \overline{a_{12}} & a_{22} & a_{23} & a_{24} & 0 \\ \overline{a_{13}} & \overline{a_{23}} & a_{33} & a_{34} & a_{35} \\ 0 & \overline{a_{24}} & \overline{a_{34}} & a_{44} & a_{45} \\ 0 & 0 & \overline{a_{35}} & \overline{a_{45}} & a_{55} \end{bmatrix}$	$\begin{bmatrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{bmatrix}$
$\begin{bmatrix} a_{11} & \overline{a_{21}} & \overline{a_{31}} & 0 & 0 \\ a_{21} & a_{22} & \overline{a_{32}} & \overline{a_{42}} & 0 \\ a_{31} & a_{32} & a_{33} & \overline{a_{43}} & \overline{a_{53}} \\ 0 & a_{42} & a_{43} & a_{44} & \overline{a_{54}} \\ 0 & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{bmatrix}$

Tabla A.6: Almacenamiento de matrices simétricas o hermíticas banda.

En el caso de las matrices tridiagonales simétricas o bidiagonales, LAPACK almacena la diagonal principal en un vector de n elementos y la otra diagonal en un vector de $n-1$ elementos.

A.3. SCALAPACK

SCALAPACK (*Scalable Linear Algebra Package*) es una librería estándar dedicada a solucionar problemas de Álgebra Lineal, en multiprocesadores de memoria distribuida y en cualquier plataforma que permita mecanismos de paso de mensajes [Blackford *et al.*, 1997], [webSCALAPACK, 2012].

SCALAPACK ofrece gran parte de las funcionalidades de la librería LAPACK y permite solucionar los mismos problemas, pero en paralelo: resolución de **sistemas de ecuaciones lineales**, **problemas de mínimos cuadrados**, **problemas de valores propios** y **problemas de valores singulares**.

Además, las rutinas de SCALAPACK tienen una interfaz semejante a las rutinas de LAPACK y eso permite una fácil adaptación para los usuarios acostumbrados a usar la librería LAPACK.

De la misma manera que en LAPACK, SCALAPACK emplea algoritmos orientados a bloques matriciales, permitiendo minimizar la frecuencia de la

transferencia de datos y reducir el número de comunicaciones.

Al mismo tiempo, SCALAPACK logra un elevado grado de portabilidad y ofrece altas prestaciones debido al hecho de utilizar la librería PBLAS para las computaciones en paralelo, igual que LAPACK al utilizar BLAS. El uso de PBLAS permite ocultar las llamadas de las rutinas internas de comunicación, permitiendo que SCALAPACK pueda ser empleada de manera transparente y de modo muy semejante a LAPACK.

A.3.1. Componentes

Para permitir las mejores prestaciones y un elevado grado de portabilidad, SCALAPACK está diseñada de manera que todas sus operaciones son ejecutadas mediante llamadas a rutinas de las librerías BLAS, LAPACK, BLACS y PBLAS.

La Figura A.1 ilustra el esquema de organización de SCALAPACK y cuál es la relación existente entre las distintas librerías, donde:

- **BLAS** (*Basic Linear Algebra Subprograms*): librería estructurada en tres niveles distintos y compuesta por las rutinas necesarias para realizar, de modo secuencial, operaciones básicas del tipo vector-vector, matriz-vector y matriz-matriz.
- **LAPACK** (*Linear Algebra Package*): librería compuesta por las rutinas necesarias para resolver problemas completos de Álgebra Lineal. Son un conjunto de rutinas que solucionan problemas de ecuaciones lineales, problemas de mínimos cuadrados, problemas de valores propios y problemas de valores singulares. Las altas prestaciones de las rutinas de LAPACK son obtenidas recurriendo a métodos que realizan llamadas a rutinas de BLAS, en especial, a las rutinas del tipo matriz-matriz.
- **BLACS** (*Basic Linear Algebra Communication Subprograms*): librería compuesta por las rutinas de paso de mensajes diseñada para el Álgebra Lineal y basada en la utilización de mallas de procesadores unidimensionales o bidimensionales, en las cuales se usan múltiples operaciones de comunicación de matrices y/o vectores.
- **PBLAS** (*Parallel BLAS*): librería compuesta por las rutinas equivalentes a las de BLAS, pero para realizar las mismas operaciones en paralelo.

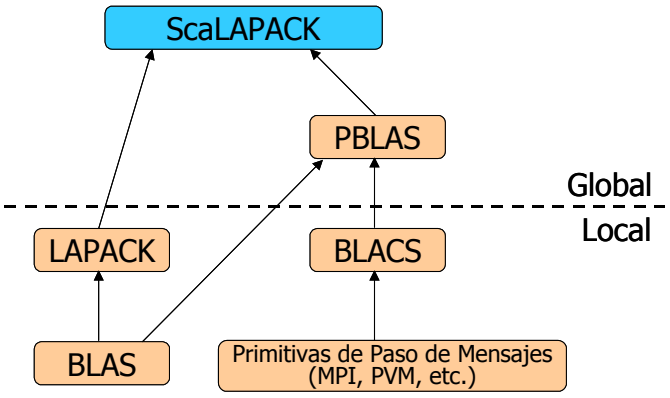

Figura A.1: Componentes de SCALAPACK.

Los diferentes componentes de SCALAPACK pertenecen a dos grupos distintos denominados **global** y **local**. La clasificación depende de si el componente realiza operaciones que deben ser llamadas por todos o por algunos de los procesadores. Al grupo **local** pertenecen las librerías que realizan operaciones que son llamadas por algunos procesadores. En cambio, al grupo **global** pertenecen las librerías que realizan operaciones que son llamadas por todos los procesadores.

Por ejemplo, en el cálculo del producto de dos matrices en paralelo, todos los procesos llamarán a la rutina de PBLAS que realiza esta operación, aunque dentro de esta rutina sólo algunos procesos llamarán a las rutinas de BLACS para hacer las comunicaciones y/o llamarán a las rutinas de BLAS necesarias para operar con las matrices.

A.3.2. Estructura

SCALAPACK sigue la misma estructura que LAPACK y así, sus rutinas están divididas en tres tipos:

- **Rutinas driver**: permiten resolver problemas completos de Álgebra

Lineal.

- **Rutinas computacionales**: permiten realizar distintas tareas computacionales.
- **Rutinas auxiliares**: permiten realizar subtareas y/o computaciones de bajo nivel.

Una vez más, las rutinas driver resuelven problemas completos de Álgebra Lineal, recurriendo a llamadas de rutinas computacionales, y las rutinas auxiliares implementan pequeñas tareas que son necesarias para las otras rutinas.

Hay que tener en cuenta que se ha intentado que las rutinas de SCALAPACK sean muy similares a las rutinas de LAPACK, es decir, las rutinas de SCALAPACK siguen el mismo formato de llamadas. Además, siguen el mismo esquema de algoritmos orientados a bloques matriciales para sacar el máximo provecho de la localidad de los datos y minimizar la frecuencia de movimiento de datos entre los diferentes niveles de la memoria. También, las rutinas de la librería confían a PBLAS las computaciones en paralelo, del mismo modo que en LAPACK se utiliza BLAS.

El uso de PBLAS oculta las llamadas a rutinas de comunicación, logrando así que SCALAPACK sea muy parecida a LAPACK. Sin embargo, no todas las funcionalidades disponibles en LAPACK están cubiertas por las rutinas de SCALAPACK, pero aquellas funcionalidades que sí lo están, tienen en SCALAPACK la misma nomenclatura que tenían en LAPACK, precedido por el carácter `P`. Por ejemplo, la rutina `DGEBRD` de LAPACK, en SCALAPACK tiene el nombre `PDGEBRD`.

A.3.3. Distribución de datos

Los datos en SCALAPACK son distribuidos según una distribución cíclica por bloques bidimensionales de datos, la cual se obtiene tras particionar los datos en bloques de tamaño fijo y a continuación realizar una distribución cíclica de los bloques por todos los procesadores disponibles, los cuales están organizados según una malla de procesadores unidimensional o bidimensional.

Tras la distribución de datos, todos los bloques de una misma fila (columna) de bloques de la matriz estarán en una misma fila (columna) de la malla de procesadores.

Por ejemplo, consideremos que la matriz de orden 9

$$\left[\begin{array}{cc|cc|cc|cc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & a_{19} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} & a_{29} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} & a_{39} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} & a_{49} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} & a_{59} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & a_{69} \\ \hline a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} & a_{87} & a_{88} & a_{89} \\ \hline a_{91} & a_{92} & a_{93} & a_{94} & a_{95} & a_{96} & a_{97} & a_{98} & a_{99} \end{array}\right]$$

ha sido distribuida en un conjunto de 6 procesadores, organizados en una malla del tipo 2×3, en la cual han sido considerados bloques matriciales de dimensión 2×2.

Tras la distribución, la matriz está distribuida por los procesadores según la ilustración de la Figura A.2.

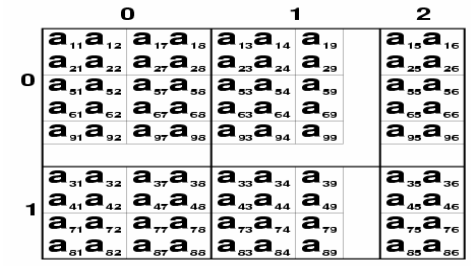

Figura A.2: Ejemplo de la distribución cíclica por bloques.

De acuerdo con la distribución de datos empleada por las rutinas de SCALAPACK, el elemento a_{ij} de la matriz A, de dimensión $m \times n$ ($i = 1, \ldots, m$; $j = 1, \ldots, n$), estará en la posición (i_p, j_p) de la submatriz que corresponde al procesador (p, q) en la malla, según las siguientes fórmulas, donde $mb \times nb$ representa la dimensión del bloque matricial y $P \times Q$ representa el tipo de la malla de procesadores:

- $Malla_{P\times Q} = \{(p,q) : 0 \leq p \leq P-1 \wedge 0 \leq q \leq Q-1\}$.
- $(p,q) = \left(\left\lfloor \frac{i}{mb} \right\rfloor \bmod P, \left\lfloor \frac{j}{nb} \right\rfloor \bmod Q\right)$.
- $(i_p, j_p) = \left(mb \left\lfloor \frac{i}{P\,mb} \right\rfloor + i \bmod mb, nb \left\lfloor \frac{j}{Q\,nb} \right\rfloor + j \bmod nb\right)$.

Además, cualquier procesador de la malla del tipo $P \times Q$ es correctamente identificado por las siguientes funciones.

$$\begin{array}{rcl} \varphi : \{0,\ldots,P\times Q-1\} & \rightarrow & Malla_{P\times Q} \\ t & \mapsto & \left(\left\lfloor \frac{t}{Q} \right\rfloor, t \bmod Q\right) \end{array}$$

$$\begin{array}{rcl} \varphi^{-1} : Malla_{P\times Q} & \rightarrow & \{0,\ldots,P\times Q-1\} \\ (p,q) & \mapsto & pQ+q \end{array}$$

Todos los datos empleados en SCALAPACK están asociados a un vector que se conoce como el **descriptor** y en el cual están descritos todos los parámetros necesarios para identificar el tipo de distribución que se ha empleado. Este **descriptor** está compuesto por 9 componentes, entre ellos, el tamaño de la matriz a distribuir (parámetros `M` y `N`), el tamaño de los bloques en que se particiona la matriz (parámetros `MB` y `NB`), los índices de la malla del procesador en que se empieza la distribución (parámetros `RSRC` y `CSRC`), el contexto de la malla de procesadores sobre la que se ha distribuido la matriz (parámetro `CTXT`) y la dimensión principal de la submatriz local en cada procesador (parámetro `LLD`).

Para construir el vector descriptor hay que utilizar la rutina `DESCINIT` de SCALAPACK.

Aunque SCALAPACK utilice una distribución bidimensional cíclica de datos, existen otras formas de distribución de datos que pueden ser unidimensionales o bidimensionales, cíclicas o no. Sin embargo, todas ellas intentan:

- Distribuir equitativamente la carga computacional por los distintos procesadores, pues, en general, el tiempo total de un algoritmo paralelo es igual

al tiempo gastado por el procesador que más tarda en completar su ejecución y por este motivo es aconsejable que todos los procesadores realicen aproximadamente el mismo volumen de trabajo.

- Minimizar las comunicaciones necesarias, pues, en general, las comunicaciones tienden a ser síncronas y, por tanto, los procesadores tienen que establecer puntos de sincronización entre procesos y, por lo tanto, demoras y esperas disminuyen el grado de aprovechamiento de los recursos.

- Reducir las necesidades de memoria, distribuyendo los datos por los diferentes procesadores disponibles y teniendo en consideración sus capacidades de memoria, sin olvidar lograr una distribución equilibrada del trabajo.

A.3.4. Implementaciones

Como ya fue descrito, SCALAPACK funciona sobre BLACS y esto supone que antes de llamar a alguna rutina de SCALAPACK el usuario tenga que configurar y distribuir los datos entre los distintos procesadores.

En cualquier implementación desarrollada en el entorno de SCALAPACK, antes de poder llamar a alguna rutina de la librería, hay que inicializar la librería BLACS. Del mismo modo, al final de la implementación es necesario finalizar BLACS. Además, SCALAPACK supone que los procesadores están agrupados según una malla, la cual tiene que ser inicializada por BLACS.

En general, cualquier implementación que llame a las rutinas de SCALAPACK tiene obligatoriamente las cuatro etapas siguientes:

1. **Inicialización de BLACS**: etapa en la cual se realiza la inicialización y la configuración de la malla de procesadores, mediante llamadas a las rutinas `BLACS_GET`, `BLACS_GRIDINIT` y `BLACS_GRIDINFO`.

2. **Distribución de datos por los procesadores**: etapa en la cual se realiza la inicialización y la descripción de la forma en que los datos son distribuidos por los distintos procesadores de la malla, recurriendo a llamadas a la rutina `DESCINIT`. Una vez terminada esta etapa se procede a la distribución de los datos por los distintos procesadores.

3. **Llamadas a las rutinas de SCALAPACK**: etapa en la cual se realizan las llamadas a las rutinas de SCALAPACK, necesarias para la resolución del problema planteado por el usuario.

4. **Finalización de BLACS**: etapa en la cual se realiza la finalización de BLACS, mediante llamadas a las rutinas `BLACS_GRIDEXIT` y `BLACS_EXIT`.

A continuación se presenta un pequeño ejemplo de una implementación en SCALAPACK, en la cual se destacan las cuatro etapas anteriores.

```
1  program EJEMPLO
2  ...
3  call BLACS_GET( -1, 0, ICTXT )
4  call BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
5  call BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
6  ...
7  call DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA, INFO)
8  ...
9  call RUTINAS_DE_SCALAPACK
10 ...
11 call BLACS_GRIDEXIT( ICTXT )
12 call BLACS_EXIT( 0 )
13 end
```

A.4. BLACS

BLACS (*Basic Linear Algebra Communication Subprograms*) es una librería compuesta por un conjunto de rutinas de paso de mensajes, diseñadas para Álgebra Lineal [Dongarra, 1991], [Dongarra y van de Geijn, 1993], [Whaley, 1994], [Dongarra y Whaley, 1995], [webBLACS, 1997].

Las rutinas están basadas en la utilización de mallas de procesadores unidimensionales o bidimensionales, en las cuales se usan múltiples operaciones de comunicación de matrices y/o vectores. La librería pretende proporcionar la misma facilidad de uso y portabilidad que el BLAS ofrece para la computación en Álgebra Lineal, pero ahora para las comunicaciones necesarias en arquitecturas con memoria distribuida.

Existen distintas versiones de BLACS, donde cada una de ellas depende de la librería de comunicación sobre la cual está implementada.

A.4.1. Nomenclatura

Las rutinas de BLACS tienen una nomenclatura propia, la cual permite identificar fácilmente el tipo de datos y el tipo de operación que realiza. Las rutinas pueden tener tres tipos de nomenclatura:

1. Rutinas con la grafía

   XYYZZ2D

   donde:

   - X − es el identificador del tipo de datos con los cuales trabaja la rutina, con la nomenclatura:
     - I − para números enteros.
     - S − para números reales en simple precisión.
     - D − para números reales en doble precisión.
     - C − para números complejos en simple precisión.
     - Z − para números complejos en doble precisión.
   - YY − es el identificador del tipo de matrices con las cuales trabaja la rutina, con la nomenclatura:
     - GE − matriz general.
     - TR − matriz trapezoidal.
   - ZZ − es el identificador del tipo de operación que realiza la rutina, con la nomenclatura:
     - SD − para el envío de un proceso a otro.
     - RV − para la recepción de un envío de otro proceso.
     - BS − para la difusión de envío a varios procesos.
     - BR − para la recepción de una difusión.

2. Rutinas con la grafía

   XGZZZ2D

   donde:

- X − sigue siendo el identificador del tipo de datos con los cuales trabaja la rutina.
- ZZZ − es el identificador del tipo de operación de reducción que realiza la rutina, con la nomenclatura:
  - AMX − para la operación de reducción del mayor valor absoluto.
  - AMN − para la operación de reducción del menor valor absoluto.
  - SUM − para la operación de reducción de la suma.

3. Rutinas con la grafía

   BLACS_Z

   donde:

   - BLACS_ − es un identificador semejante al usado por la librería MPI donde todas sus rutinas empiezan por `MPI_`, o del mismo modo que todas las rutinas de la librería PVM empiezan por `pvm_`.
   - Z − sigue siendo el identificador del tipo de operación que realiza la rutina. Este identificador puede ser de inicialización (`PINFO`, `SETUP`, `GET`, `SET`, `GRIDINIT`, `GRIDMAP`), de destrucción (`FREEBUFF`, `GRIDEXIT`, `ABORT`, `EXIT`), de obtención de información (`GRIDINFO`, `PNUM`) o de ámbito general (`PCOORD`, `BARRIER`).

A.4.2. Otras características

Las rutinas de BLACS suelen trabajar sobre una malla bidimensional de procesadores. Esta malla no necesita ser una malla física real, sino que puede ser cualquier topología física. Sin embargo, todas las rutinas de BLACS necesitan la información respecto a los índices que indican las coordenadas de cualquier procesador dentro de la malla.

También, las rutinas de BLACS que involucran a más de dos procesadores, necesitan definir el **ámbito de aplicación** de la operación a realizar. Este parámetro puede ser de tres tipos: del tipo `R` para involucrar todos los procesadores de la fila; del tipo `C` para involucrar todos los procesadores de la columna; o del tipo `A` para involucrar todos los procesadores de la malla.

Además del **ámbito de aplicación**, las rutinas de BLACS necesitan también del parámetro que define el **contexto de comunicación**. Este parámetro permite identificar el conjunto de procesadores sobre los cuales se realizan las tareas de comunicación, dando a BLACS la posibilidad de trabajar sobre mallas lógicas de procesadores.

El **contexto de comunicación** es definido por la variable `ICONTXT` y siempre que se crea un contexto, él consume recursos de BLACS y, por lo tanto, es conveniente liberarlos cuando no se van a utilizar más.

Apéndice B

Resultados con SCALAPACK

En este apéndice se presentan los tiempos de ejecución obtenidos con la rutina `PDGEBRD` de la librería SCALAPACK, empleada en el cluster Kefren, con el objetivo de testar los bloques matriciales de tamaño 32×32, 64×64 y 128×128, en la distribución cíclica de los datos y para todas las configuraciones posibles con 2, 4, 6, 8 y 10 procesadores, en el caso de las matrices cuadradas.

B.1. Tiempos de ejecución de PDGEBRD

Las tablas siguientes exponen los tiempos de ejecución obtenidos en el cluster Kefren.

Bloques de tamaño 32×32								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×2	3.82	12.26	28.34	54.30	90.58	147.19	214.08	304.57
2×1	3.92	12.36	28.46	54.50	93.83	149.69	219.31	307.24
Bloques de tamaño 64×64								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×2	4.08	13.00	29.22	56.67	92.32	149.12	212.57	305.85
2×1	4.08	13.98	29.97	58.84	95.83	151.81	215.92	310.25
Bloques de tamaño 128×128								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×2	4.82	14.43	31.38	58.24	98.13	151.09	222.27	312.29
2×1	4.93	14.59	32.01	59.36	101.33	155.41	232.16	319.68

Tabla B.1: Tiempos de ejecución de `PDGEBRD` con 2 Procesadores.

Bloques de tamaño 32×32								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×4	2.53	6.97	15.29	29.22	51.30	80.28	115.31	163.90
2×2	2.50	6.77	15.05	28.74	48.56	77.24	113.08	157.32
4×1	3.32	8.20	16.26	31.73	52.50	82.50	121.01	165.08
Bloques de tamaño 64×64								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×4	2.72	7.85	16.96	31.55	51.43	79.67	117.07	165.06
2×2	2.54	7.67	16.03	31.14	49.05	78.27	115.72	159.48
4×1	3.37	8.63	18.32	33.62	54.48	83.20	123.49	168.80
Bloques de tamaño 128×128								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×4	3.53	9.97	19.51	35.29	57.57	87.52	131.05	176.77
2×2	3.11	8.79	21.45	35.07	55.17	84.06	130.01	169.91
4×1	4.19	10.16	21.63	37.70	60.15	91.40	142.22	178.04

Tabla B.2: Tiempos de ejecución de `PDGEBRD` con 4 Procesadores.

Bloques de tamaño 32×32								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×6	2.16	5.39	11.32	21.27	34.65	56.34	80.12	113.21
2×3	1.91	4.73	10.22	19.43	31.75	49.41	71.95	102.28
3×2	1.92	4.88	10.52	19.96	32.19	50.56	72.74	103.32
6×1	2.50	6.80	12.57	21.83	36.10	57.37	82.05	114.81
Bloques de tamaño 64×64								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×6	2.37	6.29	12.82	23.53	37.85	58.88	84.30	117.93
2×3	2.10	5.53	11.20	21.10	33.94	51.53	73.66	104.67
3×2	2.03	5.80	11.70	21.31	38.35	52.42	74.27	104.94
6×1	3.03	7.73	14.28	24.39	39.94	61.00	86.31	118.87
Bloques de tamaño 128×128								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×6	3.10	8.14	15.41	26.84	43.29	65.24	92.90	128.35
2×3	2.57	6.83	13.96	23.67	38.10	56.95	80.85	110.28
3×2	2.71	6.92	14.73	24.40	40.25	57.38	81.50	112.40
6×1	3.90	9.31	17.15	28.99	45.72	68.13	95.52	129.74

Tabla B.3: Tiempos de ejecución de `PDGEBRD` con 6 Procesadores.

Bloques de tamaño 32×32								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×8	2.18	4.98	9.95	17.21	27.83	43.21	62.86	89.69
2×4	1.72	4.00	8.53	16.75	26.95	39.66	58.55	82.81
4×2	1.98	4.12	8.74	16.99	26.47	41.72	61.45	81.90
8×1	2.80	6.20	11.90	20.05	31.16	45.60	66.70	93.36
Bloques de tamaño 64×64								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×8	2.20	5.76	12.01	19.94	31.40	47.81	68.65	95.24
2×4	1.77	4.66	10.13	17.75	27.98	42.09	60.36	84.23
4×2	2.05	4.82	12.23	18.34	28.78	42.95	65.99	84.33
8×1	3.02	7.11	14.40	23.28	35.84	50.05	72.44	99.97
Bloques de tamaño 128×128								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×8	2.80	7.27	13.73	23.66	39.03	55.82	79.03	105.77
2×4	2.27	5.72	12.96	19.94	31.52	47.26	72.47	90.58
4×2	2.57	5.97	12.10	20.44	33.34	48.59	79.73	90.94
8×1	3.70	9.02	16.54	26.90	40.28	58.76	82.92	111.48

Tabla B.4: Tiempos de ejecución de `PDGEBRD` con 8 Procesadores.

Bloques de tamaño 32×32								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×10	2.07	4.35	8.81	15.46	24.94	37.64	54.26	75.14
2×5	1.72	3.49	7.20	13.32	21.63	33.55	49.54	69.30
5×2	1.80	3.91	7.54	14.20	22.96	36.18	51.32	71.38
10×1	2.71	5.89	11.44	19.90	29.24	42.44	59.04	80.22
Bloques de tamaño 64×64								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×10	2.23	5.44	10.55	18.31	28.35	42.15	60.26	83.66
2×5	1.61	4.16	10.02	14.90	23.38	36.68	51.13	72.18
5×2	1.80	4.52	8.75	16.53	25.68	38.48	54.39	74.80
10×1	3.10	7.24	13.43	22.79	34.13	48.07	65.54	87.57
Bloques de tamaño 128×128								
Malla	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
1×10	2.92	6.73	13.14	21.66	33.70	49.59	69.06	94.75
2×5	2.17	5.18	12.26	17.41	27.65	40.92	63.65	78.59
5×2	2.53	5.92	10.51	19.19	29.35	43.56	61.41	82.03
10×1	3.96	8.98	16.54	26.99	39.77	55.39	77.14	103.09

Tabla B.5: Tiempos de ejecución de `PDGEBRD` con 10 Procesadores.

Esta hoja ha sido dejada en blanco de forma intencionada.

Apéndice C

Resultados con SLICOT

En este apéndice se presentan los resultados obtenidos con la rutina AB09AD de la librería SLICOT, empleada en la resolución de los casos de prueba expuestos en el capítulo 6.

C.1. Rutina AB09AD

A continuación se expone la cabecera y los comentarios de la rutina AB09AD de la librería SLICOT.

```
SUBROUTINE AB09AD( DICO, JOB, EQUIL, ORDSEL, N, M, P, NR, A, LDA,
$                   B, LDB, C, LDC, HSV, TOL, IWORK, DWORK, LDWORK,
$                   IWARN, INFO )
C
C     RELEASE 5.0, SLICOT COPYRIGHT 1999.
C
C     PURPOSE
C
C     To compute a reduced order model (Ar,Br,Cr) for a stable original
C     state-space representation (A,B,C) by using either the square-root
C     or the balancing-free square-root Balance & Truncate (B & T)
C     model reduction method.
C
C     ARGUMENTS
C
```

```
C     Mode Parameters
C
C     DICO    CHARACTER*1
C             Specifies the type of the original system as follows:
C             = 'C':  continuous-time system;
C             = 'D':  discrete-time system.
C
C     JOB     CHARACTER*1
C             Specifies the model reduction approach to be used
C             as follows:
C             = 'B':  use the square-root Balance & Truncate method;
C             = 'N':  use the balancing-free square-root
C                     Balance & Truncate method.
C
C     EQUIL   CHARACTER*1
C             Specifies whether the user wishes to preliminarily
C             equilibrate the triplet (A,B,C) as follows:
C             = 'S':  perform equilibration (scaling);
C             = 'N':  do not perform equilibration.
C
C     ORDSEL  CHARACTER*1
C             Specifies the order selection method as follows:
C             = 'F':  the resulting order NR is fixed;
C             = 'A':  the resulting order NR is automatically determined
C                     on basis of the given tolerance TOL.
C
C     Input/Output Parameters
C
C     N       (input) INTEGER
C             The order of the original state-space representation, i.e.
C             the order of the matrix A.  N >= 0.
C
C     M       (input) INTEGER
C             The number of system inputs.  M >= 0.
C
C     P       (input) INTEGER
C             The number of system outputs.  P >= 0.
C
C     NR      (input/output) INTEGER
C             On entry with ORDSEL = 'F', NR is the desired order of the
C             resulting reduced order system.  0 <= NR <= N.
C             On exit, if INFO = 0, NR is the order of the resulting
C             reduced order model. NR is set as follows:
C             if ORDSEL = 'F', NR is equal to MIN(NR,NMIN), where NR
C             is the desired order on entry and NMIN is the order of a
C             minimal realization of the given system; NMIN is
C             determined as the number of Hankel singular values greater
C             than N*EPS*HNORM(A,B,C), where EPS is the machine
C             precision (see LAPACK Library Routine DLAMCH) and
C             HNORM(A,B,C) is the Hankel norm of the system (computed
C             in HSV(1));
C             if ORDSEL = 'A', NR is equal to the number of Hankel
C             singular values greater than MAX(TOL,N*EPS*HNORM(A,B,C)).
C
C     A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
C             On entry, the leading N-by-N part of this array must
C             contain the state dynamics matrix A.
C             On exit, if INFO = 0, the leading NR-by-NR part of this
C             array contains the state dynamics matrix Ar of the reduced
```

```
C             order system.
C
C     LDA     INTEGER
C             The leading dimension of array A.  LDA >= MAX(1,N).
C
C     B       (input/output) DOUBLE PRECISION array, dimension (LDB,M)
C             On entry, the leading N-by-M part of this array must
C             contain the original input/state matrix B.
C             On exit, if INFO = 0, the leading NR-by-M part of this
C             array contains the input/state matrix Br of the reduced
C             order system.
C
C     LDB     INTEGER
C             The leading dimension of array B.  LDB >= MAX(1,N).
C
C     C       (input/output) DOUBLE PRECISION array, dimension (LDC,N)
C             On entry, the leading P-by-N part of this array must
C             contain the original state/output matrix C.
C             On exit, if INFO = 0, the leading P-by-NR part of this
C             array contains the state/output matrix Cr of the reduced
C             order system.
C
C     LDC     INTEGER
C             The leading dimension of array C.  LDC >= MAX(1,P).
C
C     HSV     (output) DOUBLE PRECISION array, dimension (N)
C             If INFO = 0, it contains the Hankel singular values of
C             the original system ordered decreasingly. HSV(1) is the
C             Hankel norm of the system.
C
C     Tolerances
C
C     TOL     DOUBLE PRECISION
C             If ORDSEL = 'A', TOL contains the tolerance for
C             determining the order of reduced system.
C             For model reduction, the recommended value is
C             TOL = c*HNORM(A,B,C), where c is a constant in the
C             interval [0.00001,0.001], and HNORM(A,B,C) is the
C             Hankel-norm of the given system (computed in HSV(1)).
C             For computing a minimal realization, the recommended
C             value is TOL = N*EPS*HNORM(A,B,C), where EPS is the
C             machine precision (see LAPACK Library Routine DLAMCH).
C             This value is used by default if TOL <= 0 on entry.
C             If ORDSEL = 'F', the value of TOL is ignored.
C
C     Workspace
C
C     IWORK   INTEGER array, dimension (LIWORK)
C             LIWORK = 0, if JOB = 'B';
C             LIWORK = N, if JOB = 'N'.
C
C     DWORK   DOUBLE PRECISION array, dimension (LDWORK)
C             On exit, if INFO = 0, DWORK(1) returns the optimal value
C             of LDWORK.
C
C     LDWORK  INTEGER
C             The length of the array DWORK.
C             LDWORK >= MAX(1,N*(2*N+MAX(N,M,P)+5)+N*(N+1)/2).
C             For optimum performance LDWORK should be larger.
```

```
C
C     Warning Indicator
C
C     IWARN   INTEGER
C             = 0:  no warning;
C             = 1:  with ORDSEL = 'F', the selected order NR is greater
C                   than the order of a minimal realization of the
C                   given system. In this case, the resulting NR is
C                   set automatically to a value corresponding to the
C                   order of a minimal realization of the system.
C
C     Error Indicator
C
C     INFO    INTEGER
C             = 0:  successful exit;
C             < 0:  if INFO = -i, the i-th argument had an illegal
C                   value;
C             = 1:  the reduction of A to the real Schur form failed;
C             = 2:  the state matrix A is not stable (if DICO = 'C')
C                   or not convergent (if DICO = 'D');
C             = 3:  the computation of Hankel singular values failed.
C
C     METHOD
C
C     Let be the stable linear system
C
C          d[x(t)] = Ax(t) + Bu(t)
C          y(t)    = Cx(t)                                  (1)
C
C     where d[x(t)] is dx(t)/dt for a continuous-time system and x(t+1)
C     for a discrete-time system. The subroutine AB09AD determines for
C     the given system (1), the matrices of a reduced order system
C
C          d[z(t)] = Ar*z(t) + Br*u(t)
C          yr(t)   = Cr*z(t)                                (2)
C
C     such that
C
C           HSV(NR) <= INFNORM(G-Gr) <= 2*[HSV(NR+1) + ... + HSV(N)],
C
C     where G and Gr are transfer-function matrices of the systems
C     (A,B,C) and (Ar,Br,Cr), respectively, and INFNORM(G) is the
C     infinity-norm of G.
C
C     If JOB = 'B', the square-root Balance & Truncate method of [1]
C     is used and, for DICO = 'C', the resulting model is balanced.
C     By setting TOL <= 0, the routine can be used to compute balanced
C     minimal state-space realizations of stable systems.
C
C     If JOB = 'N', the balancing-free square-root version of the
C     Balance & Truncate method [2] is used.
C     By setting TOL <= 0, the routine can be used to compute minimal
C     state-space realizations of stable systems.
C
C     REFERENCES
C
C     [1] Tombs M.S. and Postlethwaite I.
C         Truncated balanced realization of stable, non-minimal
C         state-space systems.
C         Int. J. Control, Vol. 46, pp. 1319-1330, 1987.
C
```

```
C     [2] Varga A.
C         Efficient minimal realization procedure based on balancing.
C         Proc. of IMACS/IFAC Symp. MCTS, Lille, France, May 1991,
C         A. El Moudui, P. Borne, S. G. Tzafestas (Eds.),
C         Vol. 2, pp. 42-46.
C
C     NUMERICAL ASPECTS
C
C     The implemented methods rely on accuracy enhancing square-root or
C     balancing-free square-root techniques.
C                                                          3
C     The algorithms require less than 30N  floating point operations.
C
```

C.2. Resultados de los casos de prueba

Seguidamente son exhibidos los resultados obtenidos con la rutina AB09AD en la resolución de la reducción de modelos de los sistemas lineales de control establecidos como casos de prueba en el capítulo 6.

C.2.1. Resultados del caso de prueba 1

```
############################
######    Cluster ODIN    ######
############################
############################
### CASO DE PRUEBA =  1
### N =           10
### M =           5
### P =           3
############################
### Matriz A
-8.3269   -8.3269   -8.3269   -8.3269   -8.3269   -8.3269   -8.3269
-8.3269  -8.3269   -8.3269
-16.2856  -16.2856  -16.2856  -16.2856  -16.2856  -16.2856  -16.2856
-16.2856  -16.2856   -8.3269
-17.1119  -17.1119  -17.1119  -17.1119  -17.1119  -17.1119  -17.1119
-17.1119   -7.9586    0.0000
-12.5068  -12.5068  -12.5068  -12.5068  -12.5068  -12.5068  -12.5068
-9.1533   0.0000    0.0000
-11.7364  -11.7364  -11.7364  -11.7364  -11.7364  -11.7364   -3.3536
0.0000   0.0000    0.0000
-18.0134  -18.0134  -18.0134  -18.0134  -18.0134   -8.3829    0.0000
0.0000   0.0000    0.0000
-16.2997  -16.2997  -16.2997  -16.2997   -9.6306    0.0000    0.0000
0.0000   0.0000    0.0000
-10.1943  -10.1943  -10.1943   -6.6691    0.0000    0.0000    0.0000
0.0000   0.0000    0.0000
-3.7800   -3.7800   -3.5252    0.0000    0.0000    0.0000    0.0000
0.0000   0.0000    0.0000
-0.2548   -0.2548    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   0.0000    0.0000
### Matriz B
1.0000    0.0000    0.0000    0.0000    0.0000
0.0000    1.0000    0.0000    0.0000    0.0000
0.0000    0.0000    1.0000    0.0000    0.0000
0.0000    0.0000    0.0000    1.0000    0.0000
```

```
0.0000    0.0000    0.0000    0.0000    1.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
### Matriz C
1.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   0.0000    0.0000
0.0000    1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   0.0000    0.0000
0.0000    0.0000    1.0000    0.0000    0.0000    0.0000    0.0000
0.0000   0.0000    0.0000
##############################
###   RUTINA   AB09AB    ###
##############################
### IWARN =     0
### INFO =         2
### NR =           10
### TOL =   1.000000000000000E-002
##############################
### Matriz Ar
-94.2546    1.4347    2.0356  -13.3555   -9.8124    6.1887   15.1519
-8.7436  -8.4802    1.4085
0.0000   27.5239   -7.0608    3.8477    3.8905    4.1502    3.1737
-1.5315   2.9394   -1.8550
0.0000    0.0000  -11.6907    1.7292    1.0155    2.7127   -0.3652
0.7176   3.0607   -0.5259
0.0000    0.0000    0.0000    6.4670    3.5866    1.7349    0.0575
-0.4836   5.4940   -0.1683
0.0000    0.0000    0.0000    0.0000   -4.6811   -4.8158   -0.5454
-0.8152  -3.9092    1.4220
0.0000    0.0000    0.0000    0.0000    0.0000    3.1321   -0.5986
-0.5633   2.8231   -1.4461
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.9739
0.7188   3.8167   -2.5283
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   0.3912    0.7410
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000  -1.2662    0.0835
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   0.0000   -0.5548
### Matriz Br
-0.2422   -0.4736   -0.4929   -0.3518   -0.2893
0.1847    0.3597    0.3377    0.1779   -0.1215
0.1251    0.2384    0.1031   -0.1355   -0.5462
0.1922    0.3479   -0.0916   -0.4961   -0.3397
-0.2049   -0.3452    0.2422    0.4700   -0.2731
0.1631    0.2056   -0.3227   -0.0636    0.5032
0.6843   -0.1250   -0.4711    0.4418   -0.1961
-0.5603    0.5362   -0.4598    0.3705   -0.1362
-0.0580   -0.0580    0.1376   -0.0540   -0.1956
0.0217    0.0217   -0.0947    0.1407   -0.2544
### Matriz Cr
-0.2422    0.1847    0.1251    0.1922   -0.2049    0.1631    0.6843
-0.5603  -0.0580    0.0217
-0.4736    0.3597    0.2384    0.3479   -0.3452    0.2056   -0.1250
0.5362  -0.0580    0.0217
-0.4929    0.3377    0.1031   -0.0916    0.2422   -0.3227   -0.4711
-0.4598   0.1376   -0.0947
```

C.2.2. Resultados del caso de prueba 2

```
##############################
######     Cluster ODIN     ######
##############################
##############################
```

```
### CASO DE PRUEB =  2
### N =           16
### M =           1
### P =            1
############################
### Matriz A
-1.0000  100.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
-100.0000   -1.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000   -1.0000  200.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000 -200.0000   -1.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000   -1.0000  400.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000 -400.0000   -1.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   -2.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -3.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000   -4.0000    0.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000   -5.0000    0.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000   -6.0000
0.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-7.0000    0.0000   0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   -8.0000
0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000  -9.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   0.0000  -10.0000
### Matriz B
10.0000
10.0000
10.0000
10.0000
10.0000
10.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
```

```
1.0000
1.0000
### Matriz C
10.0000   10.0000   10.0000   10.0000   10.0000   10.0000
1.0000    1.0000   1.0000    1.0000    1.0000    1.0000
1.0000    1.0000   1.0000    1.0000
##############################
###   RUTINA   AB09AB    ###
##############################
### IWARN =      0
### INFO =          0
### NR =            9
### TOL =   1.000000000000000E-002
##############################
### Matriz Ar
-1.0000   -3.6667  128.0119  195.0807 -114.9609    0.3771
1.3770   0.5743   -0.1670
3.6667   -1.0000 -338.5404  -75.1152    8.3331   -0.0158
-0.0591  -0.0260    0.0085
-128.0119  338.5404   -1.0000    1.5919   -1.0016    5.6018
0.0061  -0.0070    0.0056
-195.0807   75.1153   -1.5917   -1.0004   -0.9742   10.1513
0.0821   0.0816   -0.0419
114.9609   -8.3331    1.0016    0.9741   -1.0000 -147.8659
-0.0267  -0.0050   -0.0008
-0.3771    0.0158   -5.6018  -10.1513  147.8659   -1.0000
0.0871   0.0355   -0.0094
-1.3800    0.0579   -0.0181   -0.0222    0.0343   -0.0877
-3.2731  -2.3825    0.7787
-0.5784    0.0243   -0.0075   -0.0091    0.0142   -0.0368
-2.3829  -4.3574    2.3630
0.1701   -0.0071    0.0022    0.0027   -0.0042    0.0108
0.7790   2.3631   -4.6031
### Matriz Br
-24.4247
1.0250
-0.1248
-0.0654
0.1982
-1.5442
-2.8981
-1.1882
0.3484
### Matriz Cr
-24.4247    1.0250   -0.1248   -0.0654    0.1982   -1.5442
-2.8981  -1.1882    0.3484
```

C.2.3. Resultados del caso de prueba 3

```
##############################
######     Cluster ODIN     ######
##############################
##############################
### CASO DE PRUEBA =  3
### N =           10
### M =           1
### P =            1
##############################
### Matriz A
-2.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000   -4.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -8.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000  -16.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
```

```
0.0000    0.0000    0.0000    0.0000  -32.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000  -64.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-128.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000 -256.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000 -512.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000-1024.0000
### Matriz B
1.4142
2.0000
2.8284
4.0000
5.6569
8.0000
11.3137
16.0000
22.6274
32.0000
### Matriz C
1.4142    2.0000    2.8284    4.0000    5.6569    8.0000
11.3137   16.0000   22.6274   32.0000
##############################
###   RUTINA   AB09AB    ###
##############################
### IWARN =      0
### INFO =         0
### NR =           6
### TOL =   1.000000000000000E-002
##############################
### Matriz Ar
-157.7251  191.9248  137.8482  -80.4645  -43.3412  -22.3720
191.9248 -284.3217 -247.6603  160.2389  89.7343   46.9043
137.8482 -247.6603 -288.9659  236.6130 148.4939   80.8478
-80.4645  160.2389  236.6130 -268.4304 -216.4457 -132.4198
-43.3412   89.7343  148.4939 -216.4457 -246.3694 -195.6607
-22.3720   46.9043   80.8478 -132.4198 -195.6607 -223.5095
### Matriz Br
-31.4104
26.8668
15.5659
-8.3444
-4.3691
-2.2356
### Matriz Cr
-31.4104   26.8668   15.5659   -8.3444   -4.3691   -2.2356
```

C.2.4. Resultados del caso de prueba 4

```
##############################
######     Cluster ODIN     ######
##############################
##############################
### CASO DE PRUEBA =  4
### N =           30
### M =           5
### P =            3
##############################
### Matriz A
19.4208   18.4457   18.3760   18.4105   18.4457   18.3753   18.4098
18.4457   18.3739   18.4083   18.4457   18.3710   18.4052   18.4457
18.3648   18.3982   18.4457   18.3508   18.3805   18.4457   18.3153
18.3224   18.4457   18.1992   18.0052   18.4457   17.5648   12.8204
```

```
18.4457    7.1951
17.4008   18.4457   18.3760   17.4005   17.4357   17.3653   17.3998
17.4357   17.3639   17.3983   17.4357   17.3610   17.3952   17.4357
17.3548   17.3882   17.4357   17.3408   17.3705   17.4357   17.3053
17.3124   17.4357   17.1892   16.9952   17.4357   16.5548   11.8104
17.4357    6.1851
19.4208   18.4457   20.3960   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
18.4007   18.4356   18.3659   19.4205   18.4356   18.3652   18.3997
18.4356   18.3638   18.3982   18.4356   18.3609   18.3951   18.4356
18.3547   18.3881   18.4356   18.3407   18.3704   18.4356   18.3052
18.3123   18.4356   18.1891   17.9951   18.4356   17.5547   12.8103
18.4356    7.1850
17.3806   17.4155   17.3458   17.3803   18.4356   18.3652   17.3796
17.4155   17.3437   17.3781   17.4155   17.3408   17.3750   17.4155
17.3346   17.3680   17.4155   17.3206   17.3503   17.4155   17.2851
17.2922   17.4155   17.1690   16.9750   17.4155   16.5346   11.7902
17.4155    6.1649
19.4208   19.4557   19.3860   19.4205   18.4356   20.4054   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
18.3802   18.4151   18.3454   18.3799   18.4151   18.3447   19.4198
18.4151   18.3433   18.3777   18.4151   18.3404   18.3746   18.4151
18.3342   18.3676   18.4151   18.3202   18.3499   18.4151   18.2847
18.2918   18.4151   18.1686   17.9746   18.4151   17.5342   12.7898
18.4151    7.1645
17.3396   17.3744   17.3048   17.3393   17.3744   17.3041   17.3386
18.4151   18.3433   17.3371   17.3744   17.2998   17.3340   17.3744
17.2936   17.3270   17.3744   17.2796   17.3093   17.3744   17.2441
17.2512   17.3744   17.1280   16.9340   17.3744   16.4936   11.7492
17.3744    6.1239
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
18.4151   20.4245   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
18.3380   18.3728   18.3031   18.3376   18.3728   18.3024   18.3369
18.3728   18.3010   19.4183   18.3728   18.2981   18.3324   18.3728
18.2919   18.3254   18.3728   18.2780   18.3076   18.3728   18.2424
18.2496   18.3728   18.1263   17.9324   18.3728   17.4919   12.7475
18.3728    7.1222
17.2551   17.2899   17.2203   17.2548   17.2899   17.2196   17.2541
17.2899   17.2182   17.2526   18.3728   18.2981   17.2495   17.2899
17.2091   17.2425   17.2899   17.1951   17.2248   17.2899   17.1596
17.1667   17.2899   17.0435   16.8495   17.2899   16.4091   11.6646
17.2899    6.0394
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   18.3728   20.4638   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
18.2483   18.2831   18.2134   18.2479   18.2831   18.2127   18.2472
18.2831   18.2113   18.2457   18.2831   18.2084   19.4152   18.2831
18.2022   18.2357   18.2831   18.1883   18.2179   18.2831   18.1527
18.1598   18.2831   18.0366   17.8427   18.2831   17.4022   12.6578
18.2831    7.0325
17.0757   17.1105   17.0408   17.0753   17.1105   17.0401   17.0746
17.1105   17.0387   17.0732   17.1105   17.0358   17.0701   18.2831
18.2022   17.0631   17.1105   17.0157   17.0453   17.1105   16.9801
16.9873   17.1105   16.8640   16.6701   17.1105   16.2296   11.4852
17.1105    5.8599
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   18.2831
20.5474   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
```

```
19.4557    8.2051
18.0459   18.0807   18.0111   18.0455   18.0807   18.0104   18.0448
18.0807   18.0090   18.0434   18.0807   18.0060   18.0403   18.0807
17.9998   19.4082   18.0807   17.9859   18.0155   18.0807   17.9503
17.9575   18.0807   17.8342   17.6403   18.0807   17.1999   12.4554
18.0807    6.8301
16.6709   16.7058   16.6361   16.6706   16.7058   16.6354   16.6699
16.7058   16.6340   16.6684   16.7058   16.6311   16.6653   16.7058
16.6249   16.6584   18.0807   17.9859   16.6406   16.7058   16.5754
16.5825   16.7058   16.4593   16.2653   16.7058   15.8249   11.0805
16.7058    5.4552
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   18.0807   20.7358   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
17.5304   17.5652   17.4955   17.5300   17.5652   17.4948   17.5293
17.5652   17.4934   17.5279   17.5652   17.4905   17.5248   17.5652
17.4843   17.5178   17.5652   17.4704   19.3905   17.5652   17.4348
17.4420   17.5652   17.3187   17.1248   17.5652   16.6843   11.9399
17.5652    6.3146
15.6399   15.6747   15.6051   15.6396   15.6747   15.6044   15.6388
15.6747   15.6030   15.6374   15.6747   15.6001   15.6343   15.6747
15.5939   15.6273   15.6747   15.5799   15.6095   17.5652   17.4348
15.5515   15.6747   15.4283   15.2343   15.6747   14.7939   10.0494
15.6747    4.4241
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   17.5652   21.2157
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
15.8470   15.8818   15.8122   15.8466   15.8818   15.8115   15.8459
15.8818   15.8100   15.8445   15.8818   15.8071   15.8414   15.8818
15.8009   15.8344   15.8818   15.7870   15.8166   15.8818   15.7514
19.3324   15.8818   15.6353   15.4414   15.8818   15.0010   10.2565
15.8818    4.6312
12.2731   12.3080   12.2383   12.2728   12.3080   12.2376   12.2721
12.3080   12.2362   12.2706   12.3080   12.2333   12.2675   12.3080
12.2271   12.2606   12.3080   12.2131   12.2428   12.3080   12.1776
12.1847   15.8818   15.6353   11.8675   12.3080   11.4271    6.6827
12.3080    1.0574
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   15.8818   22.7830   19.0152   19.4557   18.5748   13.8304
19.4557    8.2051
6.6485    6.6833    6.6136    6.6481    6.6833    6.6129    6.6474
6.6833   6.6115    6.6459    6.6833    6.6086    6.6428    6.6833
6.6024    6.6359    6.6833    6.5885    6.6181    6.6833    6.5529
6.5600   6.6833    6.4368   19.0152    6.6833    5.8024    1.0580
6.6833   -4.5673
-6.1239   -6.0891   -6.1588   -6.1243   -6.0891   -6.1594   -6.1250
-6.0891  -6.1609   -6.1264   -6.0891   -6.1638   -6.1295   -6.0891
-6.1700   -6.1365   -6.0891   -6.1839   -6.1543   -6.0891   -6.2195
-6.2123  -6.0891   -6.3356   -6.5295    6.6833    5.8024  -11.7144
-6.0891  -17.3397
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152    6.6833   31.3472   13.8304
19.4557    8.2051
-143.7128 -143.6779 -143.7476 -143.7131 -143.6779 -143.7483 -143.7138
-143.6779 -143.7497 -143.7153 -143.6779 -143.7526 -143.7184 -143.6779
-143.7588 -143.7253 -143.6779 -143.7728 -143.7431 -143.6779 -143.8083
-143.8012 -143.6779 -143.9244 -144.1184 -143.6779 -144.5588   13.8304
-143.6779 -154.9285
-306.8463 -306.8115 -306.8812 -306.8467 -306.8115 -306.8819 -306.8474
-306.8115 -306.8833 -306.8489 -306.8115 -306.8862 -306.8519 -306.8115
-306.8924 -306.8589 -306.8115 -306.9063 -306.8767 -306.8115 -306.9419
-306.9347 -306.8115 -307.0580 -307.2519 -306.8115 -307.6924 -312.4368
```

```
-143.6779 -154.9285
19.4208   19.4557   19.3860   19.4205   19.4557   19.3853   19.4198
19.4557   19.3839   19.4183   19.4557   19.3810   19.4152   19.4557
19.3748   19.4082   19.4557   19.3608   19.3905   19.4557   19.3253
19.3324   19.4557   19.2092   19.0152   19.4557   18.5748   13.8304
-143.6779  171.3386
### Matriz B
30.0000   29.0000   28.0000   27.0000   26.0000
29.0000   30.0000   29.0000   28.0000   27.0000
28.0000   29.0000   30.0000   29.0000   28.0000
27.0000   28.0000   29.0000   30.0000   29.0000
26.0000   27.0000   28.0000   29.0000   30.0000
25.0000   26.0000   27.0000   28.0000   29.0000
24.0000   25.0000   26.0000   27.0000   28.0000
23.0000   24.0000   25.0000   26.0000   27.0000
22.0000   23.0000   24.0000   25.0000   26.0000
21.0000   22.0000   23.0000   24.0000   25.0000
20.0000   21.0000   22.0000   23.0000   24.0000
19.0000   20.0000   21.0000   22.0000   23.0000
18.0000   19.0000   20.0000   21.0000   22.0000
17.0000   18.0000   19.0000   20.0000   21.0000
16.0000   17.0000   18.0000   19.0000   20.0000
15.0000   16.0000   17.0000   18.0000   19.0000
14.0000   15.0000   16.0000   17.0000   18.0000
13.0000   14.0000   15.0000   16.0000   17.0000
12.0000   13.0000   14.0000   15.0000   16.0000
11.0000   12.0000   13.0000   14.0000   15.0000
10.0000   11.0000   12.0000   13.0000   14.0000
9.0000   10.0000   11.0000   12.0000   13.0000
8.0000    9.0000   10.0000   11.0000   12.0000
7.0000    8.0000    9.0000   10.0000   11.0000
6.0000    7.0000    8.0000    9.0000   10.0000
5.0000    6.0000    7.0000    8.0000    9.0000
4.0000    5.0000    6.0000    7.0000    8.0000
3.0000    4.0000    5.0000    6.0000    7.0000
2.0000    3.0000    4.0000    5.0000    6.0000
1.0000    2.0000    3.0000    4.0000    5.0000
### Matriz C
30.0000   29.0000   28.0000   27.0000   26.0000   25.0000   24.0000
23.0000   22.0000   21.0000   20.0000   19.0000   18.0000   17.0000
16.0000   15.0000   14.0000   13.0000   12.0000   11.0000   10.0000
9.0000    8.0000    7.0000    6.0000    5.0000    4.0000    3.0000
2.0000    1.0000
29.0000   30.0000   29.0000   28.0000   27.0000   26.0000   25.0000
24.0000   23.0000   22.0000   21.0000   20.0000   19.0000   18.0000
17.0000   16.0000   15.0000   14.0000   13.0000   12.0000   11.0000
10.0000    9.0000    8.0000    7.0000    6.0000    5.0000    4.0000
3.0000    2.0000
28.0000   29.0000   30.0000   29.0000   28.0000   27.0000   26.0000
25.0000   24.0000   23.0000   22.0000   21.0000   20.0000   19.0000
18.0000   17.0000   16.0000   15.0000   14.0000   13.0000   12.0000
11.0000   10.0000    9.0000    8.0000    7.0000    6.0000    5.0000
4.0000    3.0000
############################
###   RUTINA   AB09AB    ###
############################
### IWARN =    0
### INFO =       2
### NR =          30
### TOL =   1.000000000000000E-002
############################
### Matriz Ar
163.1336 -168.8521    8.3798    0.0000  -12.5408    9.4048    0.0000
14.1769  -10.7156    0.0000  -16.3046    0.0000  -18.1179   13.9583
21.7476    0.0000    0.0000  -25.1027   29.6868    0.0000   36.3318
0.0000    0.0000  -46.8486   41.8469  -51.1762  -65.9125   92.8046
158.8839 1199.5474
157.6087  163.1336    0.0000    0.8803   10.7691   -8.0762    0.2463
-12.8362    9.7022    0.1303   14.8847    0.0948   16.5776  -12.7458
-19.9113   -0.0808   -0.0746   22.9897  -27.1912   -0.0717  -33.2799
```

```
0.0696   -0.0703   42.9125  -38.3121   46.8644   60.3641  -84.9951
-145.5152-1098.6182
0.0000    0.0000  163.1336   -0.6561   -8.0263    6.0192   -0.1836
9.5669   -7.2311   -0.0971  -11.0936   -0.0706  -12.3554    9.4995
14.8400    0.0602    0.0556  -17.1344   20.2658    0.0535   24.8038
-0.0519    0.0524  -31.9830   28.5543  -34.9283  -44.9898   63.3474
108.4535  818.8076
0.0000    0.0000    0.0000   12.7724   13.2727   -0.7363    0.0000
-1.1100    0.8390    0.0000    1.2766    0.0000    1.4185   -1.0929
-1.7027    0.0000    0.0000    1.9654   -2.3243    0.0000   -2.8446
0.0000    0.0000    3.6680   -3.2764    4.0068    5.1606   -7.2660
-12.4397  -93.9173
0.0000    0.0000    0.0000  -12.2909   12.7724    0.0000    0.2747
-0.7385    0.5582    0.1453    0.9854    0.1057    1.1369   -0.8470
-1.3786   -0.0901   -0.0832    1.5988   -1.8943   -0.0800   -2.3209
0.0776   -0.0784    2.9918   -2.6513    3.2547    4.1975   -5.9129
-10.1245  -76.4416
0.0000    0.0000    0.0000    0.0000    0.0000   12.7724   -0.2060
0.5538   -0.4186   -0.1090   -0.7390   -0.0793   -0.8526    0.6352
1.0339    0.0676    0.0624   -1.1990    1.4206    0.0600    1.7405
-0.0582    0.0588   -2.2437    1.9883   -2.4408   -3.1479    4.4343
7.5928   57.3265
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    3.5738
-3.7325    0.2348    0.0000    0.3572    0.0000    0.3969   -0.3058
-0.4764    0.0000    0.0000    0.5499   -0.6504    0.0000   -0.7959
0.0000    0.0000    1.0263   -0.9168    1.1211    1.4440   -2.0331
-3.4807  -26.2791
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    3.4219
3.5738    0.0000   -0.1643   -0.1538   -0.1195   -0.2182    0.1355
0.2778    0.1019    0.0941   -0.3291    0.3932    0.0904    0.4841
-0.0878    0.0887   -0.6233    0.5329   -0.6655   -0.8635    1.2190
2.0888   15.7734
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    3.5738    0.1242    0.1162    0.0903    0.1650   -0.1024
-0.2100   -0.0770   -0.0711    0.2487   -0.2972   -0.0684   -0.3659
0.0663   -0.0670    0.4711   -0.4028    0.5030    0.6527   -0.9214
-1.5788  -11.9223
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    1.8905    1.9873    0.0000    0.2100   -0.1618
-0.2520    0.0000    0.0000    0.2909   -0.3440    0.0000   -0.4210
0.0000    0.0000    0.5429   -0.4849    0.5931    0.7638   -1.0755
-1.8412  -13.9009
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -1.7983    1.8905    0.1374    0.0545   -0.0044
-0.0836   -0.1172   -0.1082    0.1061   -0.1301   -0.1040   -0.1627
0.1009   -0.1020    0.2086   -0.1589    0.2103    0.2782   -0.3953
-0.6788   -5.1285
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    1.3749    1.4534   -0.1176
-0.1833    0.0000    0.0000    0.2116   -0.2502    0.0000   -0.3062
0.0000    0.0000    0.3949   -0.3527    0.4313    0.5555   -0.7822
-1.3391  -10.1102
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000   -1.3007    1.3749    0.0417
-0.0202   -0.1302   -0.1203    0.0341   -0.0454   -0.1156   -0.0594
0.1122   -0.1133    0.0753   -0.0368    0.0627    0.0889   -0.1292
-0.2235   -1.6918
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    1.8905
0.0657    0.1003    0.0927   -0.0841    0.1034    0.0890    0.1295
-0.0864    0.0873   -0.1659    0.1247   -0.1662   -0.2203    0.3133
0.5381    4.0661
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
1.1726   -1.0970    0.1444   -0.0129    0.0214    0.1387    0.0308
-0.1346    0.1360   -0.0382   -0.0025   -0.0182   -0.0332    0.0516
0.0911    0.6929
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
1.2533    1.1726    0.0000   -0.1804    0.2134    0.0000    0.2611
```

```
0.0000    0.0000   -0.3367    0.3008   -0.3678   -0.4738    0.6671
1.1420    8.6222
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    1.0829   -1.1694    0.1971    0.0000    0.2412
0.0000    0.0000   -0.3110    0.2778   -0.3397   -0.4375    0.6160
1.0546    7.9624
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    1.0027    1.0829   -0.0071   -0.1601   -0.0140
0.1554   -0.1570    0.0162    0.0278   -0.0094   -0.0008   -0.0045
-0.0108   -0.0873
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    1.0406   -0.9502    0.0062
-0.1838    0.1856   -0.0059   -0.0447    0.0256    0.0197   -0.0210
-0.0323   -0.2369
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    1.1396
### Matriz Br
-0.7071   -0.7071   -0.7071   -0.7071   -0.7071
-21.2754  -21.2249  -21.0735  -20.8211  -20.4678
14.3292   14.2916   14.1787   13.9906   13.7273
0.7071    0.7071    0.7071    0.7071    0.7071
-19.2399  -19.1836  -19.0147  -18.7332  -18.3391
12.8979   12.8557   12.7290   12.5179   12.2223
0.7071    0.7071    0.7071    0.7071    0.7071
17.2269   17.1632   16.9723   16.6541   16.2086
-11.4857  -11.4376  -11.2933  -11.0527  -10.7160
0.7071    0.7071    0.7071    0.7071    0.7071
-15.2464  -15.1732  -14.9536  -14.5876  -14.0752
0.7071    0.7071    0.7071    0.7071    0.7071
-12.4572  -12.3759  -12.1319  -11.7252  -11.1558
11.1433   11.0807   10.8927   10.5794   10.1407
10.5594   10.4618   10.1689    9.6807    8.9973
-0.7071   -0.7071   -0.7071   -0.7071   -0.7071
-0.7071   -0.7071   -0.7071   -0.7071   -0.7071
-7.6069   -7.4942   -7.1561   -6.5926   -5.8038
4.3498    4.2165    3.8167    3.1503    2.2174
-0.7071   -0.7071   -0.7071   -0.7071   -0.7071
-4.1592   -4.3223   -3.2312   -0.8858    1.1333
0.7071    0.7071   -0.7071   -0.7071   -0.7071
-0.7071   -0.7071   -0.7071   -0.7071   -0.7071
0.4678    0.6781    0.9029    1.1422    1.8022
11.0775   10.8897   10.6889   10.4751    9.8856
-10.1960   -9.9663   -9.7207   -9.4592   -8.7383
-9.6730   -9.3771   -9.0608   -8.7240   -7.7955
9.9560    9.5394    9.0940    8.6199    7.3126
12.8391   12.1258   11.3633   10.5517   11.1174
80.7775   86.1626   91.1764   95.8188  100.0898
### Matriz Cr
-0.7071  -21.2754   14.3292    0.7071  -19.2399   12.8979    0.7071
17.2269  -11.4857    0.7071  -15.2464    0.7071  -12.4572   11.1433
10.5594   -0.7071   -0.7071   -7.6069    4.3498   -0.7071   -4.1592
0.7071   -0.7071    0.4678   11.0775  -10.1960   -9.6730    9.9560
12.8391   80.7775
-0.7071  -21.2249   14.2916    0.7071  -19.1836   12.8557    0.7071
17.1632  -11.4376    0.7071  -15.1732    0.7071  -12.3759   11.0807
10.4618   -0.7071   -0.7071   -7.4942    4.2165   -0.7071   -4.3223
0.7071   -0.7071    0.6781   10.8897   -9.9663   -9.3771    9.5394
12.1258   86.1626
-0.7071  -21.0735   14.1787    0.7071  -19.0147   12.7290    0.7071
16.9723  -11.2933    0.7071  -14.9536    0.7071  -12.1319   10.8927
10.1689   -0.7071   -0.7071   -7.1561    3.8167   -0.7071   -3.2312
-0.7071   -0.7071    0.9029   10.6889   -9.7207   -9.0608    9.0940
11.3633   91.1764
```

C.2.5. Resultados del caso de prueba 5

```
############################
######     Cluster ODIN     ######
############################
############################
### CASO DE PRUEBA =  5
### N =           10
### M =           5
### P =            5
############################
### Matriz A
-3.3756   -2.1434    1.0181   -1.0633    0.8205    0.1140   -0.7002
0.8607    0.8255    0.6617
-2.1434   -5.6308    1.2576    0.6668   -1.8667    0.1845    0.1229
-1.4271    0.4052    0.4425
1.0181    1.2576   -2.3104   -0.5130   -1.0171    1.1039   -0.0251
-0.3337   -1.7931    0.5759
-1.0633    0.6668   -0.5130   -6.0330   -0.2938   -1.4711    0.4304
0.0641   -1.9094   -0.0281
0.8205   -1.8667   -1.0171   -0.2938   -6.9276   -0.0061    1.8648
0.5276   -0.4092    1.0124
0.1140    0.1845    1.1039   -1.4711   -0.0061   -5.1120    1.2591
0.6767    0.4542   -1.8782
-0.7002    0.1229   -0.0251    0.4304    1.8648    1.2591   -5.1441
0.1098   -0.6807    1.8555
0.8607   -1.4271   -0.3337    0.0641    0.5276    0.6767    0.1098
-6.7870   -0.6372   -0.4407
0.8255    0.4052   -1.7931   -1.9094   -0.4092    0.4542   -0.6807
-0.6372   -5.7431    0.5050
0.6617    0.4425    0.5759   -0.0281    1.0124   -1.8782    1.8555
-0.4407    0.5050   -2.7934
### Matriz B
0.6053    7.2428   -1.2374   -4.4336    0.0000
0.2572   -4.6627   -2.8820    6.9555    0.0000
-0.8548    3.7386    0.4861    3.7244    0.0000
0.1571   -2.3645   -2.2734    5.4334    0.0000
-0.1855   -5.8457    2.6628   -3.8987    0.0000
0.9312    0.4584    0.0416   -0.1625    0.0000
-1.2994    2.9680   -1.5471    1.6614    0.0000
0.3048    0.5344    0.8090    7.4875    0.0000
-0.6103   -4.0295   -2.1400   -8.3354    0.0000
0.3905    0.2703   -2.1626   -6.5497    0.0000
### Matriz C
0.3717    0.9174   -0.0905   -0.2971   -0.1401   -0.7291    0.8497
-0.4846    1.0462    0.8469
-0.6913    1.1260   10.9493   -2.4710   -2.2293   -1.1222   -1.2691
3.3598   -0.9399    2.4090
3.3122    4.6878    2.5704    5.5695    0.2178    4.9699   -2.8571
-4.9396   -0.3931    0.5366
4.9120   -4.9443   -0.3629   -4.8990   -0.1559    1.9210   -1.5599
-3.6381   -1.7188    4.6773
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000
############################
###   RUTINA   AB09AB    ###
############################
### IWARN =     0
### INFO =         0
### NR =           9
### TOL =   1.000000000000000E-002
############################
### Matriz Ar
-0.6558   -0.1132    0.7042   -0.3354   -0.0122   -0.0601
0.1330   -0.0149    0.0062
-0.1131   -1.4222    0.3406    0.2117    1.1553   -1.0317
0.4360   -0.1891    0.1156
0.7083    0.3381   -3.8738    1.4378    0.0741   -0.7547
-1.0919    0.3357   -0.0515
```

```
-0.3362    0.2119    1.4233   -4.4230   -2.4680   -0.9123
1.2779    0.2302   -0.3727
-0.0095    1.1532    0.0410   -2.4525   -5.8716    1.1486
-2.2092    0.2149   -0.5835
-0.0370   -1.0478   -0.8817   -0.8078    1.2515   -6.8502
-0.4262   -0.1484    0.7393
0.1515    0.4231   -1.1963    1.3619   -2.1290   -0.4429
-6.7112   -0.8974    0.2513
-0.0654   -0.1537    0.6200    0.0005   -0.0060   -0.1121
-0.9050   -5.7666    2.1430
0.0059    0.1163   -0.0186   -0.3799   -0.5649    0.9126
0.3895    1.7645   -4.9649
### Matriz Br
0.2225    2.7499    0.4455  -13.8174    0.0000
0.0110   -9.8929   -0.7135   -2.9835    0.0000
0.2384    0.5420    2.5033    8.8761    0.0000
0.0916    1.5232    4.1488   -3.4676    0.0000
-0.8087    4.8697   -0.2428    0.5194    0.0000
-0.7339   -4.2877    3.0242   -0.7944    0.0000
-0.5650    1.5874   -1.0435    1.9047    0.0000
-1.5116   -0.6315   -0.1239   -0.9169    0.0000
0.4963    0.5007   -0.0125    0.1559    0.0000
### Matriz Cr
0.5485    0.4429   -1.0987   -1.2144    0.4403    0.0718
0.8264    0.3566   -0.1266
-3.5226   -8.3401   -2.2826   -2.4806    1.1505    5.8091
1.7199   -1.3916   -1.2174
-3.1856   -2.4470   -4.3920   -3.1335   -5.3748   -3.7684
-5.2721   -0.9337   -0.6160
7.0731   -3.8066   -3.6094    3.3218    2.2937   -0.4221
-2.5641   -3.2216    2.2185
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000
```

C.2.6. Resultados del caso de prueba 6

```
############################
######     Cluster ODIN     ######
############################
############################
### CASO DE PRUEBA =  6
### N =           10
### M =           5
### P =           3
############################
### Matriz A
-1.0000    1.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000   -1.0000    1.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -1.0000    1.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000   -1.0000    1.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000   -1.0000    1.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000   -1.0000
1.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-1.0000    1.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   -1.0000    1.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -1.0000    1.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000   -1.0000
### Matriz B
```

```
1.0000    0.0000    0.0000    0.0000    0.0000
0.0000    1.0000    0.0000    0.0000    0.0000
0.0000    0.0000    1.0000    0.0000    0.0000
0.0000    0.0000    0.0000    1.0000    0.0000
0.0000    0.0000    0.0000    0.0000    1.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
### Matriz C
1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    1.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    1.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
############################
###   RUTINA   AB09AB    ###
############################
### IWARN =     0
### INFO =         0
### NR =           5
### TOL =   1.000000000000000E-002
############################
### Matriz Ar
-0.1579   -0.4817    0.0564    0.1540   -0.0278
0.2984   -0.5303    0.7296    0.0296    0.0609
0.0108   -0.4128   -1.0195   -0.6851    0.2782
-0.0303   -0.0140    0.3163   -1.5746   -0.6348
0.0221   -0.0912   -0.2257    0.2404   -1.7177
### Matriz Br
-0.4475   -0.5622   -0.5504   -0.3867   -0.1763
-0.6070   -0.3056    0.2979    0.5440    0.3919
0.5596   -0.3708   -0.4384    0.2734    0.5314
0.3438   -0.6671    0.5204    0.0170   -0.4070
0.0025   -0.0888    0.3812   -0.6924    0.6061
### Matriz Cr
-0.4475   -0.6070    0.5596    0.3438    0.0025
-0.5622   -0.3056   -0.3708   -0.6671   -0.0888
-0.5504    0.2979   -0.4384    0.5204    0.3812
```

C.2.7. Resultados del caso de prueba 7

```
############################
######     Cluster ODIN     ######
############################
############################
### CASO DE PRUEBA =  7
### N =           10
### M =           5
### P =            3
############################
### Matriz A
-1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000   -2.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -3.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000   -4.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000   -5.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000   -6.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

```
-7.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000   -8.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000   -9.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000  -10.0000
### Matriz B
1.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
0.0000    0.0000    0.0000    0.0000    0.1000
### Matriz C
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000
1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
1.0000    1.0000    1.0000    0.1000
############################
###   RUTINA   AB09AB    ###
############################
### IWARN =     0
### INFO =         0
### NR =           2
### TOL =   1.0000000000000000E-002
############################
### Matriz Ar
-0.9775    0.8768
-0.1146   -5.0787
### Matriz Br
-0.9985    0.0000    0.0000    0.0000   -0.0775
0.0292    0.0000    0.0000    0.0000   -0.2993
### Matriz Cr
0.0000    0.0000
0.0000    0.0000
-1.0788   -2.6722
```

Apéndice **D**

Rutinas de LAPACK y de SCALAPACK

En este apéndice se exponen las cabeceras y los comentarios de las principales rutinas de LAPACK y de SCALAPACK empleadas en el marco de la tesis.

D.1. Rutinas de LAPACK

D.1.1. Rutina DGEBRD

```
SUBROUTINE DGEBRD( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK,
$                  INFO )
*
*  Purpose
*  =======
*
*  DGEBRD reduces a general real M-by-N matrix A to upper or lower
*  bidiagonal form B by an orthogonal transformation: Q**T * A * P = B.
*
*  If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.
*
*  Arguments
*  =========
```

```
*
*  M       (input) INTEGER
*          The number of rows in the matrix A.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns in the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the M-by-N general matrix to be reduced.
*          On exit,
*          if m >= n, the diagonal and the first superdiagonal are
*            overwritten with the upper bidiagonal matrix B; the
*            elements below the diagonal, with the array TAUQ, represent
*            the orthogonal matrix Q as a product of elementary
*            reflectors, and the elements above the first superdiagonal,
*            with the array TAUP, represent the orthogonal matrix P as
*            a product of elementary reflectors;
*          if m < n, the diagonal and the first subdiagonal are
*            overwritten with the lower bidiagonal matrix B; the
*            elements below the first subdiagonal, with the array TAUQ,
*            represent the orthogonal matrix Q as a product of
*            elementary reflectors, and the elements above the diagonal,
*            with the array TAUP, represent the orthogonal matrix P as
*            a product of elementary reflectors.
*          See Further Details.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  D       (output) DOUBLE PRECISION array, dimension (min(M,N))
*          The diagonal elements of the bidiagonal matrix B:
*          D(i) = A(i,i).
*
*  E       (output) DOUBLE PRECISION array, dimension (min(M,N)-1)
*          The off-diagonal elements of the bidiagonal matrix B:
*          if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1;
*          if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.
*
*  TAUQ    (output) DOUBLE PRECISION array dimension (min(M,N))
*          The scalar factors of the elementary reflectors which
*          represent the orthogonal matrix Q. See Further Details.
*
*  TAUP    (output) DOUBLE PRECISION array, dimension (min(M,N))
*          The scalar factors of the elementary reflectors which
*          represent the orthogonal matrix P. See Further Details.
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (input) INTEGER
*          The length of the array WORK.  LWORK >= max(1,M,N).
*          For optimum performance LWORK >= (M+N)*NB, where NB
*          is the optimal blocksize.
*
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
```

```
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value.
*
*  Further Details
*  ===============
*
*  The matrices Q and P are represented as products of elementary
*  reflectors:
*
*  If m >= n,
*
*     Q = H(1) H(2) . . . H(n)  and  P = G(1) G(2) . . . G(n-1)
*
*  Each H(i) and G(i) has the form:
*
*     H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'
*
*  where tauq and taup are real scalars, and v and u are real vectors;
*  v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit in A(i+1:m,i);
*  u(1:i) = 0, u(i+1) = 1, and u(i+2:n) is stored on exit in A(i,i+2:n);
*  tauq is stored in TAUQ(i) and taup in TAUP(i).
*
*  If m < n,
*
*     Q = H(1) H(2) . . . H(m-1)  and  P = G(1) G(2) . . . G(m)
*
*  Each H(i) and G(i) has the form:
*
*     H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'
*
*  where tauq and taup are real scalars, and v and u are real vectors;
*  v(1:i) = 0, v(i+1) = 1, and v(i+2:m) is stored on exit in A(i+2:m,i);
*  u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit in A(i,i+1:n);
*  tauq is stored in TAUQ(i) and taup in TAUP(i).
*
*  The contents of A on exit are illustrated by the following examples:
*
*  m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):
*
*    (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
*    (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
*    (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
*    (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
*    (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
*    (  v1  v2  v3  v4  v5 )
*
*  where d and e denote diagonal and off-diagonal elements of B, vi
*  denotes an element of the vector defining H(i), and ui an element of
*  the vector defining G(i).
*
*  ====================================================================
```

D.1.2. Rutina DGESVD

```
SUBROUTINE DGESVD( JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT,
$                   WORK, LWORK, INFO )
*
*  Purpose
*  =======
*
*  DGESVD computes the singular value decomposition (SVD) of a real
*  M-by-N matrix A, optionally computing the left and/or right singular
```

```
*  vectors. The SVD is written
*
*       A = U * SIGMA * transpose(V)
*
*  where SIGMA is an M-by-N matrix which is zero except for its
*  min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and
*  V is an N-by-N orthogonal matrix.  The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and
*  are returned in descending order.  The first min(m,n) columns of
*  U and V are the left and right singular vectors of A.
*
*  Note that the routine returns V**T, not V.
*
*  Arguments
*  =========
*
*  JOBU    (input) CHARACTER*1
*          Specifies options for computing all or part of the matrix U:
*          = 'A':  all M columns of U are returned in array U:
*          = 'S':  the first min(m,n) columns of U (the left singular
*                  vectors) are returned in the array U;
*          = 'O':  the first min(m,n) columns of U (the left singular
*                  vectors) are overwritten on the array A;
*          = 'N':  no columns of U (no left singular vectors) are
*                  computed.
*
*  JOBVT   (input) CHARACTER*1
*          Specifies options for computing all or part of the matrix
*          V**T:
*          = 'A':  all N rows of V**T are returned in the array VT;
*          = 'S':  the first min(m,n) rows of V**T (the right singular
*                  vectors) are returned in the array VT;
*          = 'O':  the first min(m,n) rows of V**T (the right singular
*                  vectors) are overwritten on the array A;
*          = 'N':  no rows of V**T (no right singular vectors) are
*                  computed.
*
*          JOBVT and JOBU cannot both be 'O'.
*
*  M       (input) INTEGER
*          The number of rows of the input matrix A.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns of the input matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the M-by-N matrix A.
*          On exit,
*          if JOBU = 'O',  A is overwritten with the first min(m,n)
*                          columns of U (the left singular vectors,
*                          stored columnwise);
*          if JOBVT = 'O', A is overwritten with the first min(m,n)
*                          rows of V**T (the right singular vectors,
*                          stored rowwise);
*          if JOBU .ne. 'O' and JOBVT .ne. 'O', the contents of A
*                          are destroyed.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
```

```
*
*  S       (output) DOUBLE PRECISION array, dimension (min(M,N))
*          The singular values of A, sorted so that S(i) >= S(i+1).
*
*  U       (output) DOUBLE PRECISION array, dimension (LDU,UCOL)
*          (LDU,M) if JOBU = 'A' or (LDU,min(M,N)) if JOBU = 'S'.
*          If JOBU = 'A', U contains the M-by-M orthogonal matrix U;
*          if JOBU = 'S', U contains the first min(m,n) columns of U
*          (the left singular vectors, stored columnwise);
*          if JOBU = 'N' or 'O', U is not referenced.
*
*  LDU     (input) INTEGER
*          The leading dimension of the array U.  LDU >= 1; if
*          JOBU = 'S' or 'A', LDU >= M.
*
*  VT      (output) DOUBLE PRECISION array, dimension (LDVT,N)
*          If JOBVT = 'A', VT contains the N-by-N orthogonal matrix
*          V**T;
*          if JOBVT = 'S', VT contains the first min(m,n) rows of
*          V**T (the right singular vectors, stored rowwise);
*          if JOBVT = 'N' or 'O', VT is not referenced.
*
*  LDVT    (input) INTEGER
*          The leading dimension of the array VT.  LDVT >= 1; if
*          JOBVT = 'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M,N).
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK;
*          if INFO > 0, WORK(2:MIN(M,N)) contains the unconverged
*          superdiagonal elements of an upper bidiagonal matrix B
*          whose diagonal is in S (not necessarily sorted). B
*          satisfies A = U * B * VT, so it has the same singular values
*          as A, and singular vectors related by U and VT.
*
*  LWORK   (input) INTEGER
*          The dimension of the array WORK.
*          LWORK >= MAX(1,3*MIN(M,N)+MAX(M,N),5*MIN(M,N)).
*          For good performance, LWORK should generally be larger.
*
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit.
*          < 0:  if INFO = -i, the i-th argument had an illegal value.
*          > 0:  if DBDSQR did not converge, INFO specifies how many
*                superdiagonals of an intermediate bidiagonal form B
*                did not converge to zero. See the description of WORK
*                above for details.
*
*  =====================================================================
```

D.1.3. Rutina DGEBD2

```
SUBROUTINE DGEBD2( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO )
*
*  Purpose
*  =======
*
*  DGEBD2 reduces a real general m by n matrix A to upper or lower
*  bidiagonal form B by an orthogonal transformation: Q**T * A * P = B.
*
*  If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.
*
*  Arguments
*  =========
*
*  M       (input) INTEGER
*          The number of rows in the matrix A.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns in the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the m by n general matrix to be reduced.
*          On exit,
*          if m >= n, the diagonal and the first superdiagonal are
*            overwritten with the upper bidiagonal matrix B; the
*            elements below the diagonal, with the array TAUQ, represent
*            the orthogonal matrix Q as a product of elementary
*            reflectors, and the elements above the first superdiagonal,
*            with the array TAUP, represent the orthogonal matrix P as
*            a product of elementary reflectors;
*          if m < n, the diagonal and the first subdiagonal are
*            overwritten with the lower bidiagonal matrix B; the
*            elements below the first subdiagonal, with the array TAUQ,
*            represent the orthogonal matrix Q as a product of
*            elementary reflectors, and the elements above the diagonal,
*            with the array TAUP, represent the orthogonal matrix P as
*            a product of elementary reflectors.
*          See Further Details.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  D       (output) DOUBLE PRECISION array, dimension (min(M,N))
*          The diagonal elements of the bidiagonal matrix B:
*          D(i) = A(i,i).
*
*  E       (output) DOUBLE PRECISION array, dimension (min(M,N)-1)
*          The off-diagonal elements of the bidiagonal matrix B:
*          if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1;
*          if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.
*
*  TAUQ    (output) DOUBLE PRECISION array dimension (min(M,N))
*          The scalar factors of the elementary reflectors which
*          represent the orthogonal matrix Q. See Further Details.
*
*  TAUP    (output) DOUBLE PRECISION array, dimension (min(M,N))
*          The scalar factors of the elementary reflectors which
*          represent the orthogonal matrix P. See Further Details.
*
*  WORK    (workspace) DOUBLE PRECISION array, dimension (max(M,N))
```

```
*
*  INFO    (output) INTEGER
*          = 0: successful exit.
*          < 0: if INFO = -i, the i-th argument had an illegal value.
*
*  Further Details
*  ===============
*
*  The matrices Q and P are represented as products of elementary
*  reflectors:
*
*  If m >= n,
*
*     Q = H(1) H(2) . . . H(n)  and  P = G(1) G(2) . . . G(n-1)
*
*  Each H(i) and G(i) has the form:
*
*     H(i) = I - tauq * v * v**T  and G(i) = I - taup * u * u**T
*
*  where tauq and taup are real scalars, and v and u are real vectors;
*  v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit in A(i+1:m,i);
*  u(1:i) = 0, u(i+1) = 1, and u(i+2:n) is stored on exit in A(i,i+2:n);
*  tauq is stored in TAUQ(i) and taup in TAUP(i).
*
*  If m < n,
*
*     Q = H(1) H(2) . . . H(m-1)  and  P = G(1) G(2) . . . G(m)
*
*  Each H(i) and G(i) has the form:
*
*     H(i) = I - tauq * v * v**T  and G(i) = I - taup * u * u**T
*
*  where tauq and taup are real scalars, and v and u are real vectors;
*  v(1:i) = 0, v(i+1) = 1, and v(i+2:m) is stored on exit in A(i+2:m,i);
*  u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit in A(i,i+1:n);
*  tauq is stored in TAUQ(i) and taup in TAUP(i).
*
*  The contents of A on exit are illustrated by the following examples:
*
*  m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):
*
*    (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
*    (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
*    (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
*    (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
*    (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
*    (  v1  v2  v3  v4  v5 )
*
*  where d and e denote diagonal and off-diagonal elements of B, vi
*  denotes an element of the vector defining H(i), and ui an element of
*  the vector defining G(i).
*
*  =====================================================================
```

D.1.4. Rutina DSTEBZ

```
SUBROUTINE DSTEBZ( RANGE, ORDER, N, VL, VU, IL, IU, ABSTOL, D, E,
$                  M, NSPLIT, W, IBLOCK, ISPLIT, WORK, IWORK,
$                  INFO )
*
*  Purpose
*  =======
```

```
*
*  DSTEBZ computes the eigenvalues of a symmetric tridiagonal
*  matrix T.  The user may ask for all eigenvalues, all eigenvalues
*  in the half-open interval (VL, VU], or the IL-th through IU-th
*  eigenvalues.
*
*  To avoid overflow, the matrix must be scaled so that its
*  largest element is no greater than overflow**(1/2) *
*  underflow**(1/4) in absolute value, and for greatest
*  accuracy, it should not be much smaller than that.
*
*  See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal
*  Matrix", Report CS41, Computer Science Dept., Stanford
*  University, July 21, 1966.
*
*  Arguments
*  =========
*
*  RANGE   (input) CHARACTER*1
*          = 'A': ("All")   all eigenvalues will be found.
*          = 'V': ("Value") all eigenvalues in the half-open interval
*                           (VL, VU] will be found.
*          = 'I': ("Index") the IL-th through IU-th eigenvalues (of the
*                           entire matrix) will be found.
*
*  ORDER   (input) CHARACTER*1
*          = 'B': ("By Block") the eigenvalues will be grouped by
*                              split-off block (see IBLOCK, ISPLIT) and
*                              ordered from smallest to largest within
*                              the block.
*          = 'E': ("Entire matrix")
*                              the eigenvalues for the entire matrix
*                              will be ordered from smallest to
*                              largest.
*
*  N       (input) INTEGER
*          The order of the tridiagonal matrix T.  N >= 0.
*
*  VL      (input) DOUBLE PRECISION
*  VU      (input) DOUBLE PRECISION
*          If RANGE='V', the lower and upper bounds of the interval to
*          be searched for eigenvalues.  Eigenvalues less than or equal
*          to VL, or greater than VU, will not be returned.  VL < VU.
*          Not referenced if RANGE = 'A' or 'I'.
*
*  IL      (input) INTEGER
*  IU      (input) INTEGER
*          If RANGE='I', the indices (in ascending order) of the
*          smallest and largest eigenvalues to be returned.
*          1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0.
*          Not referenced if RANGE = 'A' or 'V'.
*
*  ABSTOL  (input) DOUBLE PRECISION
*          The absolute tolerance for the eigenvalues.  An eigenvalue
*          (or cluster) is considered to be located if it has been
*          determined to lie in an interval whose width is ABSTOL or
*          less.  If ABSTOL is less than or equal to zero, then ULP*|T|
*          will be used, where |T| means the 1-norm of T.
*
*          Eigenvalues will be computed most accurately when ABSTOL is
```

```
*          set to twice the underflow threshold 2*DLAMCH('S'), not zero.
*
*  D       (input) DOUBLE PRECISION array, dimension (N)
*          The n diagonal elements of the tridiagonal matrix T.
*
*  E       (input) DOUBLE PRECISION array, dimension (N-1)
*          The (n-1) off-diagonal elements of the tridiagonal matrix T.
*
*  M       (output) INTEGER
*          The actual number of eigenvalues found. 0 <= M <= N.
*          (See also the description of INFO=2,3.)
*
*  NSPLIT  (output) INTEGER
*          The number of diagonal blocks in the matrix T.
*          1 <= NSPLIT <= N.
*
*  W       (output) DOUBLE PRECISION array, dimension (N)
*          On exit, the first M elements of W will contain the
*          eigenvalues.  (DSTEBZ may use the remaining N-M elements as
*          workspace.)
*
*  IBLOCK  (output) INTEGER array, dimension (N)
*          At each row/column j where E(j) is zero or small, the
*          matrix T is considered to split into a block diagonal
*          matrix.  On exit, if INFO = 0, IBLOCK(i) specifies to which
*          block (from 1 to the number of blocks) the eigenvalue W(i)
*          belongs.  (DSTEBZ may use the remaining N-M elements as
*          workspace.)
*
*  ISPLIT  (output) INTEGER array, dimension (N)
*          The splitting points, at which T breaks up into submatrices.
*          The first submatrix consists of rows/columns 1 to ISPLIT(1),
*          the second of rows/columns ISPLIT(1)+1 through ISPLIT(2),
*          etc., and the NSPLIT-th consists of rows/columns
*          ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.
*          (Only the first NSPLIT elements will actually be used, but
*          since the user cannot know a priori what value NSPLIT will
*          have, N words must be reserved for ISPLIT.)
*
*  WORK    (workspace) DOUBLE PRECISION array, dimension (4*N)
*
*  IWORK   (workspace) INTEGER array, dimension (3*N)
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  some or all of the eigenvalues failed to converge or
*                were not computed:
*                =1 or 3: Bisection failed to converge for some
*                         eigenvalues; these eigenvalues are flagged by a
*                         negative block number.  The effect is that the
*                         eigenvalues may not be as accurate as the
*                         absolute and relative tolerances.  This is
*                         generally caused by unexpectedly inaccurate
*                         arithmetic.
*                =2 or 3: RANGE='I' only: Not all of the eigenvalues
*                         IL:IU were found.
*                         Effect: M < IU+1-IL
*                         Cause:  non-monotonic arithmetic, causing the
```

```
*                             Sturm sequence to be non-monotonic.
*                       Cure:   recalculate, using RANGE='A', and pick
*                             out eigenvalues IL:IU.  In some cases,
*                             increasing the PARAMETER "FUDGE" may
*                             make things work.
*               = 4:    RANGE='I', and the Gershgorin interval
*                       initially used was too small.  No eigenvalues
*                       were computed.
*                       Probable cause: your machine has sloppy
*                                       floating-point arithmetic.
*                       Cure: Increase the PARAMETER "FUDGE",
*                             recompile, and try again.
*
*  Internal Parameters
*  ===================
*
*  RELFAC  DOUBLE PRECISION, default = 2.0e0
*          The relative tolerance.  An interval (a,b] lies within
*          "relative tolerance" if  b-a < RELFAC*ulp*max(|a|,|b|),
*          where "ulp" is the machine precision (distance from 1 to
*          the next larger floating point number.)
*
*  FUDGE   DOUBLE PRECISION, default = 2
*          A "fudge factor" to widen the Gershgorin intervals.  Ideally,
*          a value of 1 should work, but on machines with sloppy
*          arithmetic, this needs to be larger.  The default for
*          publicly released versions should be large enough to handle
*          the worst machine around.  Note that this has no effect
*          on accuracy of the solution.
*
*  =====================================================================
```

D.1.5. Rutina DSTEDC

```
SUBROUTINE DSTEDC( COMPZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK,
$                  LIWORK, INFO )
*
*  Purpose
*  =======
*
*  DSTEDC computes all eigenvalues and, optionally, eigenvectors of a
*  symmetric tridiagonal matrix using the divide and conquer method.
*  The eigenvectors of a full or band real symmetric matrix can also be
*  found if DSYTRD or DSPTRD or DSBTRD has been used to reduce this
*  matrix to tridiagonal form.
*
*  This code makes very mild assumptions about floating point
*  arithmetic. It will work on machines with a guard digit in
*  add/subtract, or on those binary machines without guard digits
*  which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2.
*  It could conceivably fail on hexadecimal or decimal machines
*  without guard digits, but we know of none.  See DLAED3 for details.
*
*  Arguments
*  =========
*
*  COMPZ   (input) CHARACTER*1
*          = 'N':  Compute eigenvalues only.
```

```
*          = 'I':  Compute eigenvectors of tridiagonal matrix also.
*          = 'V':  Compute eigenvectors of original dense symmetric
*                  matrix also.  On entry, Z contains the orthogonal
*                  matrix used to reduce the original matrix to
*                  tridiagonal form.
*
*  N       (input) INTEGER
*          The dimension of the symmetric tridiagonal matrix.  N >= 0.
*
*  D       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, the diagonal elements of the tridiagonal matrix.
*          On exit, if INFO = 0, the eigenvalues in ascending order.
*
*  E       (input/output) DOUBLE PRECISION array, dimension (N-1)
*          On entry, the subdiagonal elements of the tridiagonal matrix.
*          On exit, E has been destroyed.
*
*  Z       (input/output) DOUBLE PRECISION array, dimension (LDZ,N)
*          On entry, if COMPZ = 'V', then Z contains the orthogonal
*          matrix used in the reduction to tridiagonal form.
*          On exit, if INFO = 0, then if COMPZ = 'V', Z contains the
*          orthonormal eigenvectors of the original symmetric matrix,
*          and if COMPZ = 'I', Z contains the orthonormal eigenvectors
*          of the symmetric tridiagonal matrix.
*          If  COMPZ = 'N', then Z is not referenced.
*
*  LDZ     (input) INTEGER
*          The leading dimension of the array Z.  LDZ >= 1.
*          If eigenvectors are desired, then LDZ >= max(1,N).
*
*  WORK    (workspace/output) DOUBLE PRECISION array,
*                                      dimension (LWORK)
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (input) INTEGER
*          The dimension of the array WORK.
*          If COMPZ = 'N' or N <= 1 then LWORK must be at least 1.
*          If COMPZ = 'V' and N > 1 then LWORK must be at least
*                         ( 1 + 3*N + 2*N*lg N + 3*N**2 ),
*                         where lg( N ) = smallest integer k such
*                         that 2**k >= N.
*          If COMPZ = 'I' and N > 1 then LWORK must be at least
*                         ( 1 + 4*N + N**2 ).
*          Note that for COMPZ = 'I' or 'V', then if N is less than or
*          equal to the minimum divide size, usually 25, then LWORK need
*          only be max(1,2*(N-1)).
*
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  IWORK   (workspace/output) INTEGER array, dimension (MAX(1,LIWORK))
*          On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.
*
*  LIWORK  (input) INTEGER
*          The dimension of the array IWORK.
*          If COMPZ = 'N' or N <= 1 then LIWORK must be at least 1.
*          If COMPZ = 'V' and N > 1 then LIWORK must be at least
*                         ( 6 + 6*N + 5*N*lg N ).
*          If COMPZ = 'I' and N > 1 then LIWORK must be at least
```

```
*                         ( 3 + 5*N ).
*          Note that for COMPZ = 'I' or 'V', then if N is less than or
*          equal to the minimum divide size, usually 25, then LIWORK
*          need only be 1.
*
*          If LIWORK = -1, then a workspace query is assumed; the
*          routine only calculates the optimal size of the IWORK array,
*          returns this value as the first entry of the IWORK array, and
*          no error message related to LIWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit.
*          < 0:  if INFO = -i, the i-th argument had an illegal value.
*          > 0:  The algorithm failed to compute an eigenvalue while
*                working on the submatrix lying in rows and columns
*                INFO/(N+1) through mod(INFO,N+1).
*
*  =====================================================================
```

D.1.6. Rutina DSTEIN

```
SUBROUTINE DSTEIN( N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK,
$                  IWORK, IFAIL, INFO )
*
*  Purpose
*  =======
*
*  DSTEIN computes the eigenvectors of a real symmetric tridiagonal
*  matrix T corresponding to specified eigenvalues, using inverse
*  iteration.
*
*  The maximum number of iterations allowed for each eigenvector is
*  specified by an internal parameter MAXITS (currently set to 5).
*
*  Arguments
*  =========
*
*  N       (input) INTEGER
*          The order of the matrix.  N >= 0.
*
*  D       (input) DOUBLE PRECISION array, dimension (N)
*          The n diagonal elements of the tridiagonal matrix T.
*
*  E       (input) DOUBLE PRECISION array, dimension (N-1)
*          The (n-1) subdiagonal elements of the tridiagonal matrix
*          T, in elements 1 to N-1.
*
*  M       (input) INTEGER
*          The number of eigenvectors to be found.  0 <= M <= N.
*
*  W       (input) DOUBLE PRECISION array, dimension (N)
*          The first M elements of W contain the eigenvalues for
*          which eigenvectors are to be computed.  The eigenvalues
*          should be grouped by split-off block and ordered from
*          smallest to largest within the block.  ( The output array
*          W from DSTEBZ with ORDER = 'B' is expected here. )
*
*  IBLOCK  (input) INTEGER array, dimension (N)
*          The submatrix indices associated with the corresponding
```

```
*          eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to
*          the first submatrix from the top, =2 if W(i) belongs to
*          the second submatrix, etc.  ( The output array IBLOCK
*          from DSTEBZ is expected here. )
*
*  ISPLIT  (input) INTEGER array, dimension (N)
*          The splitting points, at which T breaks up into submatrices.
*          The first submatrix consists of rows/columns 1 to
*          ISPLIT( 1 ), the second of rows/columns ISPLIT( 1 )+1
*          through ISPLIT( 2 ), etc.
*          ( The output array ISPLIT from DSTEBZ is expected here. )
*
*  Z       (output) DOUBLE PRECISION array, dimension (LDZ, M)
*          The computed eigenvectors.  The eigenvector associated
*          with the eigenvalue W(i) is stored in the i-th column of
*          Z.  Any vector which fails to converge is set to its current
*          iterate after MAXITS iterations.
*
*  LDZ     (input) INTEGER
*          The leading dimension of the array Z.  LDZ >= max(1,N).
*
*  WORK    (workspace) DOUBLE PRECISION array, dimension (5*N)
*
*  IWORK   (workspace) INTEGER array, dimension (N)
*
*  IFAIL   (output) INTEGER array, dimension (M)
*          On normal exit, all elements of IFAIL are zero.
*          If one or more eigenvectors fail to converge after
*          MAXITS iterations, then their indices are stored in
*          array IFAIL.
*
*  INFO    (output) INTEGER
*          = 0: successful exit.
*          < 0: if INFO = -i, the i-th argument had an illegal value
*          > 0: if INFO = i, then i eigenvectors failed to converge
*               in MAXITS iterations.  Their indices are stored in
*               array IFAIL.
*
*  Internal Parameters
*  ===================
*
*  MAXITS  INTEGER, default = 5
*          The maximum number of iterations performed.
*
*  EXTRA   INTEGER, default = 2
*          The number of iterations performed after norm growth
*          criterion is satisfied, should be at least 1.
*
*  =====================================================================
```

D.1.7. Rutina DSTEMR

```
SUBROUTINE DSTEMR( JOBZ, RANGE, N, D, E, VL, VU, IL, IU,
$                  M, W, Z, LDZ, NZC, ISUPPZ, TRYRAC, WORK, LWORK,
$                  IWORK, LIWORK, INFO )
*
*  Purpose
*  =======
*
*  DSTEMR computes selected eigenvalues and, optionally, eigenvectors
```

```
*  of a real symmetric tridiagonal matrix T. Any such unreduced matrix has
*  a well defined set of pairwise different real eigenvalues, the corresponding
*  real eigenvectors are pairwise orthogonal.
*
*  The spectrum may be computed either completely or partially by specifying
*  either an interval (VL,VU] or a range of indices IL:IU for the desired
*  eigenvalues.
*
*  Depending on the number of desired eigenvalues, these are computed either
*  by bisection or the dqds algorithm. Numerically orthogonal eigenvectors are
*  computed by the use of various suitable L D L^T factorizations near clusters
*  of close eigenvalues (referred to as RRRs, Relatively Robust
*  Representations). An informal sketch of the algorithm follows.
*
*  For each unreduced block (submatrix) of T,
*     (a) Compute T - sigma I  = L D L^T, so that L and D
*         define all the wanted eigenvalues to high relative accuracy.
*         This means that small relative changes in the entries of D and L
*         cause only small relative changes in the eigenvalues and
*         eigenvectors. The standard (unfactored) representation of the
*         tridiagonal matrix T does not have this property in general.
*     (b) Compute the eigenvalues to suitable accuracy.
*         If the eigenvectors are desired, the algorithm attains full
*         accuracy of the computed eigenvalues only right before
*         the corresponding vectors have to be computed, see steps c) and d).
*     (c) For each cluster of close eigenvalues, select a new
*         shift close to the cluster, find a new factorization, and refine
*         the shifted eigenvalues to suitable accuracy.
*     (d) For each eigenvalue with a large enough relative separation compute
*         the corresponding eigenvector by forming a rank revealing twisted
*         factorization. Go back to (c) for any clusters that remain.
*
*  For more details, see:
*  - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations
*    to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
*    Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
*  - Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and
*    Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
*    2004.  Also LAPACK Working Note 154.
*  - Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric
*    tridiagonal eigenvalue/eigenvector problem",
*    Computer Science Division Technical Report No. UCB/CSD-97-971,
*    UC Berkeley, May 1997.
*
*  Further Details
*  1.DSTEMR works only on machines which follow IEEE-754
*  floating-point standard in their handling of infinities and NaNs.
*  This permits the use of efficient inner loops avoiding a check for
*  zero divisors.
*
*  Arguments
*  =========
*
*  JOBZ    (input) CHARACTER*1
*          = 'N':  Compute eigenvalues only;
*          = 'V':  Compute eigenvalues and eigenvectors.
*
*  RANGE   (input) CHARACTER*1
```

```
*          = 'A': all eigenvalues will be found.
*          = 'V': all eigenvalues in the half-open interval (VL,VU]
*                 will be found.
*          = 'I': the IL-th through IU-th eigenvalues will be found.
*
*  N       (input) INTEGER
*          The order of the matrix.  N >= 0.
*
*  D       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, the N diagonal elements of the tridiagonal matrix
*          T. On exit, D is overwritten.
*
*  E       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, the (N-1) subdiagonal elements of the tridiagonal
*          matrix T in elements 1 to N-1 of E. E(N) need not be set on
*          input, but is used internally as workspace.
*          On exit, E is overwritten.
*
*  VL      (input) DOUBLE PRECISION
*  VU      (input) DOUBLE PRECISION
*          If RANGE='V', the lower and upper bounds of the interval to
*          be searched for eigenvalues. VL < VU.
*          Not referenced if RANGE = 'A' or 'I'.
*
*  IL      (input) INTEGER
*  IU      (input) INTEGER
*          If RANGE='I', the indices (in ascending order) of the
*          smallest and largest eigenvalues to be returned.
*          1 <= IL <= IU <= N, if N > 0.
*          Not referenced if RANGE = 'A' or 'V'.
*
*  M       (output) INTEGER
*          The total number of eigenvalues found.  0 <= M <= N.
*          If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.
*
*  W       (output) DOUBLE PRECISION array, dimension (N)
*          The first M elements contain the selected eigenvalues in
*          ascending order.
*
*  Z       (output) DOUBLE PRECISION array, dimension (LDZ, max(1,M) )
*          If JOBZ = 'V', and if INFO = 0, then the first M columns of Z
*          contain the orthonormal eigenvectors of the matrix T
*          corresponding to the selected eigenvalues, with the i-th
*          column of Z holding the eigenvector associated with W(i).
*          If JOBZ = 'N', then Z is not referenced.
*          Note: the user must ensure that at least max(1,M) columns are
*          supplied in the array Z; if RANGE = 'V', the exact value of M
*          is not known in advance and can be computed with a workspace
*          query by setting NZC = -1, see below.
*
*  LDZ     (input) INTEGER
*          The leading dimension of the array Z.  LDZ >= 1, and if
*          JOBZ = 'V', then LDZ >= max(1,N).
*
*  NZC     (input) INTEGER
*          The number of eigenvectors to be held in the array Z.
*          If RANGE = 'A', then NZC >= max(1,N).
*          If RANGE = 'V', then NZC >= the number of eigenvalues in (VL,VU].
*          If RANGE = 'I', then NZC >= IU-IL+1.
*          If NZC = -1, then a workspace query is assumed; the
```

```
*          routine calculates the number of columns of the array Z that
*          are needed to hold the eigenvectors.
*          This value is returned as the first entry of the Z array, and
*          no error message related to NZC is issued by XERBLA.
*
*  ISUPPZ  (output) INTEGER ARRAY, dimension ( 2*max(1,M) )
*          The support of the eigenvectors in Z, i.e., the indices
*          indicating the nonzero elements in Z. The i-th computed eigenvector
*          is nonzero only in elements ISUPPZ( 2*i-1 ) through
*          ISUPPZ( 2*i ). This is relevant in the case when the matrix
*          is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.
*
*  TRYRAC  (input/output) LOGICAL
*          If TRYRAC.EQ..TRUE., indicates that the code should check whether
*          the tridiagonal matrix defines its eigenvalues to high relative
*          accuracy.  If so, the code uses relative-accuracy preserving
*          algorithms that might be (a bit) slower depending on the matrix.
*          If the matrix does not define its eigenvalues to high relative
*          accuracy, the code can uses possibly faster algorithms.
*          If TRYRAC.EQ..FALSE., the code is not required to guarantee
*          relatively accurate eigenvalues and can use the fastest possible
*          techniques.
*          On exit, a .TRUE. TRYRAC will be set to .FALSE. if the matrix
*          does not define its eigenvalues to high relative accuracy.
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
*          On exit, if INFO = 0, WORK(1) returns the optimal
*          (and minimal) LWORK.
*
*  LWORK   (input) INTEGER
*          The dimension of the array WORK. LWORK >= max(1,18*N)
*          if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'.
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  IWORK   (workspace/output) INTEGER array, dimension (LIWORK)
*          On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.
*
*  LIWORK  (input) INTEGER
*          The dimension of the array IWORK.  LIWORK >= max(1,10*N)
*          if the eigenvectors are desired, and LIWORK >= max(1,8*N)
*          if only the eigenvalues are to be computed.
*          If LIWORK = -1, then a workspace query is assumed; the
*          routine only calculates the optimal size of the IWORK array,
*          returns this value as the first entry of the IWORK array, and
*          no error message related to LIWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
*          On exit, INFO
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  if INFO = 1X, internal error in DLARRE,
*                if INFO = 2X, internal error in DLARRV.
*                Here, the digit X = ABS( IINFO ) < 10, where IINFO is
*                the nonzero error code returned by DLARRE or
*                DLARRV, respectively.
*
```

```
*  =====================================================================
```

D.1.8. Rutina DSTEQR

```
SUBROUTINE DSTEQR( COMPZ, N, D, E, Z, LDZ, WORK, INFO )
*
*  Purpose
*  =======
*
*  DSTEQR computes all eigenvalues and, optionally, eigenvectors of a
*  symmetric tridiagonal matrix using the implicit QL or QR method.
*  The eigenvectors of a full or band symmetric matrix can also be found
*  if DSYTRD or DSPTRD or DSBTRD has been used to reduce this matrix to
*  tridiagonal form.
*
*  Arguments
*  =========
*
*  COMPZ   (input) CHARACTER*1
*          = 'N':  Compute eigenvalues only.
*          = 'V':  Compute eigenvalues and eigenvectors of the original
*                  symmetric matrix.  On entry, Z must contain the
*                  orthogonal matrix used to reduce the original matrix
*                  to tridiagonal form.
*          = 'I':  Compute eigenvalues and eigenvectors of the
*                  tridiagonal matrix.  Z is initialized to the identity
*                  matrix.
*
*  N       (input) INTEGER
*          The order of the matrix.  N >= 0.
*
*  D       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, the diagonal elements of the tridiagonal matrix.
*          On exit, if INFO = 0, the eigenvalues in ascending order.
*
*  E       (input/output) DOUBLE PRECISION array, dimension (N-1)
*          On entry, the (n-1) subdiagonal elements of the tridiagonal
*          matrix.
*          On exit, E has been destroyed.
*
*  Z       (input/output) DOUBLE PRECISION array, dimension (LDZ, N)
*          On entry, if  COMPZ = 'V', then Z contains the orthogonal
*          matrix used in the reduction to tridiagonal form.
*          On exit, if INFO = 0, then if  COMPZ = 'V', Z contains the
*          orthonormal eigenvectors of the original symmetric matrix,
*          and if COMPZ = 'I', Z contains the orthonormal eigenvectors
*          of the symmetric tridiagonal matrix.
*          If COMPZ = 'N', then Z is not referenced.
*
*  LDZ     (input) INTEGER
*          The leading dimension of the array Z.  LDZ >= 1, and if
*          eigenvectors are desired, then  LDZ >= max(1,N).
*
*  WORK    (workspace) DOUBLE PRECISION array, dimension (max(1,2*N-2))
*          If COMPZ = 'N', then WORK is not referenced.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
```

```
*          > 0:  the algorithm has failed to find all the eigenvalues in
*                a total of 30*N iterations; if INFO = i, then i
*                elements of E have not converged to zero; on exit, D
*                and E contain the elements of a symmetric tridiagonal
*                matrix which is orthogonally similar to the original
*                matrix.
*
*  =====================================================================
```

D.1.9. Rutina DSTERF

```
SUBROUTINE DSTERF( N, D, E, INFO )
*
*  Purpose
*  =======
*
*  DSTERF computes all eigenvalues of a symmetric tridiagonal matrix
*  using the Pal-Walker-Kahan variant of the QL or QR algorithm.
*
*  Arguments
*  =========
*
*  N       (input) INTEGER
*          The order of the matrix.  N >= 0.
*
*  D       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, the n diagonal elements of the tridiagonal matrix.
*          On exit, if INFO = 0, the eigenvalues in ascending order.
*
*  E       (input/output) DOUBLE PRECISION array, dimension (N-1)
*          On entry, the (n-1) subdiagonal elements of the tridiagonal
*          matrix.
*          On exit, E has been destroyed.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  the algorithm failed to find all of the eigenvalues in
*                a total of 30*N iterations; if INFO = i, then i
*                elements of E have not converged to zero.
*
*  =====================================================================
```

D.1.10. Rutina DLATMS

```
SUBROUTINE DLATMS( M, N, DIST, ISEED, SYM, D, MODE, COND, DMAX,
$                  KL, KU, PACK, A, LDA, WORK, INFO )
*
*  Purpose
*  =======
*
*     DLATMS generates random matrices with specified singular values
*     (or symmetric/hermitian with specified eigenvalues)
*     for testing LAPACK programs.
*
*     DLATMS operates by applying the following sequence of
*     operations:
*
```

```
*       Set the diagonal to D, where D may be input or
*          computed according to MODE, COND, DMAX, and SYM
*          as described below.
*
*       Generate a matrix with the appropriate band structure, by one
*          of two methods:
*
*       Method A:
*           Generate a dense M x N matrix by multiplying D on the left
*                and the right by random unitary matrices, then:
*
*           Reduce the bandwidth according to KL and KU, using
*           Householder transformations.
*
*       Method B:
*           Convert the bandwidth-0 (i.e., diagonal) matrix to a
*                bandwidth-1 matrix using Givens rotations, "chasing"
*                out-of-band elements back, much as in QR; then
*                convert the bandwidth-1 to a bandwidth-2 matrix, etc.
*                Note that for reasonably small bandwidths (relative to
*                M and N) this requires less storage, as a dense matrix
*                is not generated.  Also, for symmetric matrices, only
*                one triangle is generated.
*
*       Method A is chosen if the bandwidth is a large fraction of the
*           order of the matrix, and LDA is at least M (so a dense
*           matrix can be stored.)  Method B is chosen if the bandwidth
*           is small (< 1/2 N for symmetric, < .3 N+M for
*           non-symmetric), or LDA is less than M and not less than the
*           bandwidth.
*
*       Pack the matrix if desired. Options specified by PACK are:
*          no packing
*          zero out upper half (if symmetric)
*          zero out lower half (if symmetric)
*          store the upper half columnwise (if symmetric or upper
*                triangular)
*          store the lower half columnwise (if symmetric or lower
*                triangular)
*          store the lower triangle in banded format (if symmetric
*                or lower triangular)
*          store the upper triangle in banded format (if symmetric
*                or upper triangular)
*          store the entire matrix in banded format
*       If Method B is chosen, and band format is specified, then the
*          matrix will be generated in the band format, so no repacking
*          will be necessary.
*
*  Arguments
*  =========
*
*  M      - INTEGER
*           The number of rows of A. Not modified.
*
*  N      - INTEGER
*           The number of columns of A. Not modified.
*
*  DIST   - CHARACTER*1
*           On entry, DIST specifies the type of distribution to be used
*           to generate the random eigen-/singular values.
```

```
*           'U' => UNIFORM( 0, 1 )  ( 'U' for uniform )
*           'S' => UNIFORM( -1, 1 ) ( 'S' for symmetric )
*           'N' => NORMAL( 0, 1 )   ( 'N' for normal )
*           Not modified.
*
*  ISEED  - INTEGER array, dimension ( 4 )
*           On entry ISEED specifies the seed of the random number
*           generator. They should lie between 0 and 4095 inclusive,
*           and ISEED(4) should be odd. The random number generator
*           uses a linear congruential sequence limited to small
*           integers, and so should produce machine independent
*           random numbers. The values of ISEED are changed on
*           exit, and can be used in the next call to DLATMS
*           to continue the same random number sequence.
*           Changed on exit.
*
*  SYM    - CHARACTER*1
*           If SYM='S' or 'H', the generated matrix is symmetric, with
*             eigenvalues specified by D, COND, MODE, and DMAX; they
*             may be positive, negative, or zero.
*           If SYM='P', the generated matrix is symmetric, with
*             eigenvalues (= singular values) specified by D, COND,
*             MODE, and DMAX; they will not be negative.
*           If SYM='N', the generated matrix is nonsymmetric, with
*             singular values specified by D, COND, MODE, and DMAX;
*             they will not be negative.
*           Not modified.
*
*  D      - DOUBLE PRECISION array, dimension ( MIN( M , N ) )
*           This array is used to specify the singular values or
*           eigenvalues of A (see SYM, above.)  If MODE=0, then D is
*           assumed to contain the singular/eigenvalues, otherwise
*           they will be computed according to MODE, COND, and DMAX,
*           and placed in D.
*           Modified if MODE is nonzero.
*
*  MODE   - INTEGER
*           On entry this describes how the singular/eigenvalues are to
*           be specified:
*           MODE = 0 means use D as input
*           MODE = 1 sets D(1)=1 and D(2:N)=1.0/COND
*           MODE = 2 sets D(1:N-1)=1 and D(N)=1.0/COND
*           MODE = 3 sets D(I)=COND**(-(I-1)/(N-1))
*           MODE = 4 sets D(i)=1 - (i-1)/(N-1)*(1 - 1/COND)
*           MODE = 5 sets D to random numbers in the range
*                    ( 1/COND , 1 ) such that their logarithms
*                    are uniformly distributed.
*           MODE = 6 set D to random numbers from same distribution
*                    as the rest of the matrix.
*           MODE < 0 has the same meaning as ABS(MODE), except that
*              the order of the elements of D is reversed.
*           Thus if MODE is positive, D has entries ranging from
*              1 to 1/COND, if negative, from 1/COND to 1,
*           If SYM='S' or 'H', and MODE is neither 0, 6, nor -6, then
*              the elements of D will also be multiplied by a random
*              sign (i.e., +1 or -1.)
*           Not modified.
*
```

```
*  COND   - DOUBLE PRECISION
*           On entry, this is used as described under MODE above.
*           If used, it must be >= 1. Not modified.
*
*  DMAX   - DOUBLE PRECISION
*           If MODE is neither -6, 0 nor 6, the contents of D, as
*           computed according to MODE and COND, will be scaled by
*           DMAX / max(abs(D(i))); thus, the maximum absolute eigen- or
*           singular value (which is to say the norm) will be abs(DMAX).
*           Note that DMAX need not be positive: if DMAX is negative
*           (or zero), D will be scaled by a negative number (or zero).
*           Not modified.
*
*  KL     - INTEGER
*           This specifies the lower bandwidth of the  matrix. For
*           example, KL=0 implies upper triangular, KL=1 implies upper
*           Hessenberg, and KL being at least M-1 means that the matrix
*           has full lower bandwidth.  KL must equal KU if the matrix
*           is symmetric.
*           Not modified.
*
*  KU     - INTEGER
*           This specifies the upper bandwidth of the  matrix. For
*           example, KU=0 implies lower triangular, KU=1 implies lower
*           Hessenberg, and KU being at least N-1 means that the matrix
*           has full upper bandwidth.  KL must equal KU if the matrix
*           is symmetric.
*           Not modified.
*
*  PACK   - CHARACTER*1
*           This specifies packing of matrix as follows:
*           'N' => no packing
*           'U' => zero out all subdiagonal entries (if symmetric)
*           'L' => zero out all superdiagonal entries (if symmetric)
*           'C' => store the upper triangle columnwise
*                  (only if the matrix is symmetric or upper triangular)
*           'R' => store the lower triangle columnwise
*                  (only if the matrix is symmetric or lower triangular)
*           'B' => store the lower triangle in band storage scheme
*                  (only if matrix symmetric or lower triangular)
*           'Q' => store the upper triangle in band storage scheme
*                  (only if matrix symmetric or upper triangular)
*           'Z' => store the entire matrix in band storage scheme
*                      (pivoting can be provided for by using this
*                      option to store A in the trailing rows of
*                      the allocated storage)
*
*           Using these options, the various LAPACK packed and banded
*           storage schemes can be obtained:
*           GB                - use 'Z'
*           PB, SB or TB      - use 'B' or 'Q'
*           PP, SP or TP      - use 'C' or 'R'
*
*           If two calls to DLATMS differ only in the PACK parameter,
*           they will generate mathematically equivalent matrices.
*           Not modified.
*
*  A      - DOUBLE PRECISION array, dimension ( LDA, N )
*           On exit A is the desired test matrix.  A is first generated
```

```
*           in full (unpacked) form, and then packed, if so specified
*           by PACK.  Thus, the first M elements of the first N
*           columns will always be modified.  If PACK specifies a
*           packed or banded storage scheme, all LDA elements of the
*           first N columns will be modified; the elements of the
*           array which do not correspond to elements of the generated
*           matrix are set to zero.
*           Modified.
*
*  LDA    - INTEGER
*           LDA specifies the first dimension of A as declared in the
*           calling program.  If PACK='N', 'U', 'L', 'C', or 'R', then
*           LDA must be at least M.  If PACK='B' or 'Q', then LDA must
*           be at least MIN( KL, M-1) (which is equal to MIN(KU,N-1)).
*           If PACK='Z', LDA must be large enough to hold the packed
*           array: MIN( KU, N-1) + MIN( KL, M-1) + 1.
*           Not modified.
*
*  WORK   - DOUBLE PRECISION array, dimension ( 3*MAX( N , M ) )
*           Workspace.
*           Modified.
*
*  INFO   - INTEGER
*           Error code.  On exit, INFO will be set to one of the
*           following values:
*             0 => normal return
*            -1 => M negative or unequal to N and SYM='S', 'H', or 'P'
*            -2 => N negative
*            -3 => DIST illegal string
*            -5 => SYM illegal string
*            -7 => MODE not in range -6 to 6
*            -8 => COND less than 1.0, and MODE neither -6, 0 nor 6
*           -10 => KL negative
*           -11 => KU negative, or SYM='S' or 'H' and KU not equal to KL
*           -12 => PACK illegal string, or PACK='U' or 'L', and SYM='N';
*                  or PACK='C' or 'Q' and SYM='N' and KL is not zero;
*                  or PACK='R' or 'B' and SYM='N' and KU is not zero;
*                  or PACK='U', 'L', 'C', 'R', 'B', or 'Q', and M is not
*                  N.
*           -14 => LDA is less than M, or PACK='Z' and LDA is less than
*                  MIN(KU,N-1) + MIN(KL,M-1) + 1.
*            1  => Error return from DLATM1
*            2  => Cannot scale to DMAX (max. sing. value is 0)
*            3  => Error return from DLAGGE or SLAGSY
*
*  =====================================================================
```

D.2. Rutinas de SCALAPACK

D.2.1. Rutina PDGEBRD

```
SUBROUTINE PDGEBRD( M, N, A, IA, JA, DESCA, D, E, TAUQ, TAUP,
$                   WORK, LWORK, INFO )
*
*  Purpose
*  =======
```

```
*
*  PDGEBRD reduces a real general M-by-N distributed matrix
*  sub( A ) = A(IA:IA+M-1,JA:JA+N-1) to upper or lower bidiagonal
*  form B by an orthogonal transformation: Q' * sub( A ) * P = B.
*
*  If M >= N, B is upper bidiagonal; if M < N, B is lower bidiagonal.
*
*  Notes
*  =====
*
*  Each global data object is described by an associated description
*  vector.  This vector stores the information required to establish
*  the mapping between an object element and its corresponding process
*  and memory location.
*
*  Let A be a generic term for any 2D block cyclicly distributed array.
*  Such a global array has an associated description vector DESCA.
*  In the following comments, the character _ should be read as
*  "of the global array".
*
*  NOTATION        STORED IN      EXPLANATION
*  --------------- -------------- --------------------------------------
*  DTYPE_A(global) DESCA( DTYPE_ )The descriptor type.  In this case,
*                                 DTYPE_A = 1.
*  CTXT_A (global) DESCA( CTXT_ ) The BLACS context handle, indicating
*                                 the BLACS process grid A is distribu-
*                                 ted over. The context itself is glo-
*                                 bal, but the handle (the integer
*                                 value) may vary.
*  M_A    (global) DESCA( M_ )    The number of rows in the global
*                                 array A.
*  N_A    (global) DESCA( N_ )    The number of columns in the global
*                                 array A.
*  MB_A   (global) DESCA( MB_ )   The blocking factor used to distribute
*                                 the rows of the array.
*  NB_A   (global) DESCA( NB_ )   The blocking factor used to distribute
*                                 the columns of the array.
*  RSRC_A (global) DESCA( RSRC_ ) The process row over which the first
*                                 row of the array A is distributed.
*  CSRC_A (global) DESCA( CSRC_ ) The process column over which the
*                                 first column of the array A is
*                                 distributed.
*  LLD_A  (local)  DESCA( LLD_ )  The leading dimension of the local
*                                 array.  LLD_A >= MAX(1,LOCr(M_A)).
*
*  Let K be the number of rows or columns of a distributed matrix,
*  and assume that its process grid has dimension p x q.
*  LOCr( K ) denotes the number of elements of K that a process
*  would receive if K were distributed over the p processes of its
*  process column.
*  Similarly, LOCc( K ) denotes the number of elements of K that a
*  process would receive if K were distributed over the q processes of
*  its process row.
*  The values of LOCr() and LOCc() may be determined via a call to the
*  ScaLAPACK tool function, NUMROC:
*          LOCr( M ) = NUMROC( M, MB_A, MYROW, RSRC_A, NPROW ),
*          LOCc( N ) = NUMROC( N, NB_A, MYCOL, CSRC_A, NPCOL ).
*  An upper bound for these quantities may be computed by:
```

```
*          LOCr( M ) <= ceil( ceil(M/MB_A)/NPROW )*MB_A
*          LOCc( N ) <= ceil( ceil(N/NB_A)/NPCOL )*NB_A
*
*  Arguments
*  =========
*
*  M       (global input) INTEGER
*          The number of rows to be operated on, i.e. the number of rows
*          of the distributed submatrix sub( A ). M >= 0.
*
*  N       (global input) INTEGER
*          The number of columns to be operated on, i.e. the number of
*          columns of the distributed submatrix sub( A ). N >= 0.
*
*  A       (local input/local output) DOUBLE PRECISION pointer into the
*          local memory to an array of dimension (LLD_A,LOCc(JA+N-1)).
*          On entry, this array contains the local pieces of the
*          general distributed matrix sub( A ). On exit, if M >= N,
*          the diagonal and the first superdiagonal of sub( A ) are
*          overwritten with the upper bidiagonal matrix B; the elements
*          below the diagonal, with the array TAUQ, represent the
*          orthogonal matrix Q as a product of elementary reflectors,
*          and the elements above the first superdiagonal, with the
*          array TAUP, represent the orthogonal matrix P as a product
*          of elementary reflectors. If M < N, the diagonal and the
*          first subdiagonal are overwritten with the lower bidiagonal
*          matrix B; the elements below the first subdiagonal, with the
*          array TAUQ, represent the orthogonal matrix Q as a product of
*          elementary reflectors, and the elements above the diagonal,
*          with the array TAUP, represent the orthogonal matrix P as a
*          product of elementary reflectors. See Further Details.
*
*  IA      (global input) INTEGER
*          The row index in the global array A indicating the first
*          row of sub( A ).
*
*  JA      (global input) INTEGER
*          The column index in the global array A indicating the
*          first column of sub( A ).
*
*  DESCA   (global and local input) INTEGER array of dimension DLEN_.
*          The array descriptor for the distributed matrix A.
*
*  D       (local output) DOUBLE PRECISION array, dimension
*          LOCc(JA+MIN(M,N)-1) if M >= N; LOCr(IA+MIN(M,N)-1) otherwise.
*          The distributed diagonal elements of the bidiagonal matrix
*          B: D(i) = A(i,i). D is tied to the distributed matrix A.
*
*  E       (local output) DOUBLE PRECISION array, dimension
*          LOCr(IA+MIN(M,N)-1) if M >= N; LOCc(JA+MIN(M,N)-2) otherwise.
*          The distributed off-diagonal elements of the bidiagonal
*          distributed matrix B:
*          if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1;
*          if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.
*          E is tied to the distributed matrix A.
*
*  TAUQ    (local output) DOUBLE PRECISION array dimension
*          LOCc(JA+MIN(M,N)-1). The scalar factors of the elementary
*          reflectors which represent the orthogonal matrix Q. TAUQ
```

```
*          is tied to the distributed matrix A. See Further Details.
*
*  TAUP    (local output) DOUBLE PRECISION array, dimension
*          LOCr(IA+MIN(M,N)-1). The scalar factors of the elementary
*          reflectors which represent the orthogonal matrix P. TAUP
*          is tied to the distributed matrix A. See Further Details.
*
*  WORK    (local workspace/local output) DOUBLE PRECISION array,
*                                         dimension (LWORK)
*          On exit, WORK( 1 ) returns the minimal and optimal LWORK.
*
*  LWORK   (local or global input) INTEGER
*          The dimension of the array WORK.
*          LWORK is local input and must be at least
*          LWORK >= NB*( MpA0 + NqA0 + 1 ) + NqA0
*
*          where NB = MB_A = NB_A,
*          IROFFA = MOD( IA-1, NB ), ICOFFA = MOD( JA-1, NB ),
*          IAROW = INDXG2P( IA, NB, MYROW, RSRC_A, NPROW ),
*          IACOL = INDXG2P( JA, NB, MYCOL, CSRC_A, NPCOL ),
*          MpA0 = NUMROC( M+IROFFA, NB, MYROW, IAROW, NPROW ),
*          NqA0 = NUMROC( N+ICOFFA, NB, MYCOL, IACOL, NPCOL ).
*
*          INDXG2P and NUMROC are ScaLAPACK tool functions;
*          MYROW, MYCOL, NPROW and NPCOL can be determined by calling
*          the subroutine BLACS_GRIDINFO.
*
*          If LWORK = -1, then LWORK is global input and a workspace
*          query is assumed; the routine only calculates the minimum
*          and optimal size for all work arrays. Each of these
*          values is returned in the first entry of the corresponding
*          work array, and no error message is issued by PXERBLA.
*
*  INFO    (global output) INTEGER
*          = 0:  successful exit
*          < 0:  If the i-th argument is an array and the j-entry had
*                an illegal value, then INFO = -(i*100+j), if the i-th
*                argument is a scalar and had an illegal value, then
*                INFO = -i.
*
*  Further Details
*  ===============
*
*  The matrices Q and P are represented as products of elementary
*  reflectors:
*
*  If m >= n,
*
*     Q = H(1) H(2) . . . H(n)  and  P = G(1) G(2) . . . G(n-1)
*
*  Each H(i) and G(i) has the form:
*
*     H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'
*
*  where tauq and taup are real scalars, and v and u are real vectors;
*  v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit in
*  A(ia+i:ia+m-1,ja+i-1);
*  u(1:i) = 0, u(i+1) = 1, and u(i+2:n) is stored on exit in
*  A(ia+i-1,ja+i+1:ja+n-1);
*  tauq is stored in TAUQ(ja+i-1) and taup in TAUP(ia+i-1).
*
```

```
*  If m < n,
*
*     Q = H(1) H(2) . . . H(m-1)  and  P = G(1) G(2) . . . G(m)
*
*  Each H(i) and G(i) has the form:
*
*     H(i) = I - tauq * v * v'  and G(i) = I - taup * u * u'
*
*  where tauq and taup are real scalars, and v and u are real vectors;
*  v(1:i) = 0, v(i+1) = 1, and v(i+2:m) is stored on exit in
*  A(ia+i+1:ia+m-1,ja+i-1);
*  u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit in
*  A(ia+i-1,ja+i:ja+n-1);
*  tauq is stored in TAUQ(ja+i-1) and taup in TAUP(ia+i-1).
*
*  The contents of sub( A ) on exit are illustrated by the following
*  examples:
*
*  m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):
*
*    (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
*    (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
*    (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
*    (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
*    (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
*    (  v1  v2  v3  v4  v5 )
*
*  where d and e denote diagonal and off-diagonal elements of B, vi
*  denotes an element of the vector defining H(i), and ui an element of
*  the vector defining G(i).
*
*  Alignment requirements
*  ======================
*
*  The distributed submatrix sub( A ) must verify some alignment proper-
*  ties, namely the following expressions should be true:
*  ( MB_A.EQ.NB_A .AND. IROFFA.EQ.ICOFFA )
*
*  ============================================================================
```

D.2.2. Rutina PDGESVD

```
SUBROUTINE PDGESVD(JOBU,JOBVT,M,N,A,IA,JA,DESCA,S,U,IU,JU,DESCU,
$                  VT,IVT,JVT,DESCVT,WORK,LWORK,INFO)
*
*  Purpose
*  =======
*
*  PDGESVD computes the singular value decomposition (SVD) of an
*  M-by-N matrix A, optionally computing the left and/or right
*  singular vectors. The SVD is written as
*
*       A = U * SIGMA * transpose(V)
*
*  where SIGMA is an M-by-N matrix which is zero except for its
*  min(M,N) diagonal elements, U is an M-by-M orthogonal matrix, and
*  V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA
*  are the singular values of A and the columns of U and V are the
*  corresponding right and left singular vectors, respectively. The
*  singular values are returned in array S in decreasing order and
```

```
*  only the first min(M,N) columns of U and rows of VT = V**T are
*  computed.
*
*  Notes
*  =====
*  Each global data object is described by an associated description
*  vector. This vector stores the information required to establish
*  the mapping between an object element and its corresponding process
*  and memory location.
*
*  Let A be a generic term for any 2D block cyclicly distributed array.
*  Such a global array has an associated description vector DESCA.
*  In the following comments, the character _ should be read as
*  "of the global array".
*
*  NOTATION        STORED IN      EXPLANATION
*  --------------- -------------- --------------------------------------
*  DTYPE_A(global) DESCA( DTYPE_ )The descriptor type.  In this case,
*                                 DTYPE_A = 1.
*  CTXT_A (global) DESCA( CTXT_ ) The BLACS context handle, indicating
*                                 the BLACS process grid A is distribu-
*                                 ted over. The context itself is glo-
*                                 bal, but the handle (the integer
*                                 value) may vary.
*  M_A    (global) DESCA( M_ )    The number of rows in the global
*                                 array A.
*  N_A    (global) DESCA( N_ )    The number of columns in the global
*                                 array A.
*  MB_A   (global) DESCA( MB_ )   The blocking factor used to distribute
*                                 the rows of the array.
*  NB_A   (global) DESCA( NB_ )   The blocking factor used to distribute
*                                 the columns of the array.
*  RSRC_A (global) DESCA( RSRC_ ) The process row over which the first
*                                 row of the array A is distributed.
*  CSRC_A (global) DESCA( CSRC_ ) The process column over which the
*                                 first column of the array A is
*                                 distributed.
*  LLD_A  (local)  DESCA( LLD_ )  The leading dimension of the local
*                                 array.  LLD_A >= MAX(1,LOCr(M_A)).
*
*  Let K be the number of rows or columns of a distributed matrix, and
*  assume that its process grid has dimension r x c. LOCr( K ) denotes
*  the number of elements of K that a process would receive if K were
*  distributed over the r processes of its process column. Similarly,
*  LOCc( K ) denotes the number of elements of K that a process would
*  receive if K were distributed over the c processes of its process
*  row. The values of LOCr() and LOCc() may be determined via a call
*  to the ScaLAPACK tool function, NUMROC:
*          LOCr( M ) = NUMROC( M, MB_A, MYROW, RSRC_A, NPROW ),
*          LOCc( N ) = NUMROC( N, NB_A, MYCOL, CSRC_A, NPCOL ).
*  An upper bound for these quantities may be computed by:
*          LOCr( M ) <= ceil( ceil(M/MB_A)/NPROW )*MB_A
*          LOCc( N ) <= ceil( ceil(N/NB_A)/NPCOL )*NB_A
*
*  Arguments
*  =========
*
*          MP = number of local rows in A and U
```

```
*          NQ = number of local columns in A and VT
*          SIZE = min( M, N )
*          SIZEQ = number of local columns in U
*          SIZEP = number of local rows in VT
*
*  JOBU    (global input) CHARACTER*1
*          Specifies options for computing U:
*          = 'V':  the first SIZE columns of U (the left singular
*                  vectors) are returned in the array U;
*          = 'N':  no columns of U (no left singular vectors) are
*                  computed.
*
*  JOBVT   (global input) CHARACTER*1
*          Specifies options for computing V**T:
*          = 'V':  the first SIZE rows of V**T (the right singular
*                  vectors) are returned in the array VT;
*          = 'N':  no rows of V**T (no right singular vectors) are
*                  computed.
*
*  M       (global input) INTEGER
*          The number of rows of the input matrix A.  M >= 0.
*
*  N       (global input) INTEGER
*          The number of columns of the input matrix A.  N >= 0.
*
*  A       (local input/workspace) block cyclic DOUBLE PRECISION
*          array,
*          global dimension (M, N), local dimension (MP, NQ)
*          On exit, the contents of A are destroyed.
*
*  IA      (global input) INTEGER
*          The row index in the global array A indicating the first
*          row of sub( A ).
*
*  JA      (global input) INTEGER
*          The column index in the global array A indicating the
*          first column of sub( A ).
*
*  DESCA   (global input) INTEGER array of dimension DLEN_
*          The array descriptor for the distributed matrix A.
*
*  S       (global output) DOUBLE PRECISION   array, dimension SIZE
*          The singular values of A, sorted so that S(i) >= S(i+1).
*
*  U       (local output) DOUBLE PRECISION   array, local dimension
*          (MP, SIZEQ), global dimension (M, SIZE)
*          if JOBU = 'V', U contains the first min(m,n) columns of U
*          if JOBU = 'N', U is not referenced.
*
*  IU      (global input) INTEGER
*          The row index in the global array U indicating the first
*          row of sub( U ).
*
*  JU      (global input) INTEGER
*          The column index in the global array U indicating the
*          first column of sub( U ).
*
*  DESCU   (global input) INTEGER array of dimension DLEN_
*          The array descriptor for the distributed matrix U.
*
*  VT      (local output) DOUBLE PRECISION   array, local dimension
```

```
*          (SIZEP, NQ), global dimension (SIZE, N).
*          If JOBVT = 'V', VT contains the first SIZE rows of
*          V**T. If JOBVT = 'N', VT is not referenced.
*
*  IVT     (global input) INTEGER
*          The row index in the global array VT indicating the first
*          row of sub( VT ).
*
*  JVT     (global input) INTEGER
*          The column index in the global array VT indicating the
*          first column of sub( VT ).
*
*  DESCVT   (global input) INTEGER array of dimension DLEN_
*          The array descriptor for the distributed matrix VT.
*
*  WORK    (local workspace/output) DOUBLE PRECISION   array, dimension
*          (LWORK)
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (local input) INTEGER
*          The dimension of the array WORK.
*
*          LWORK >= 1 + 6*SIZEB + MAX(WATOBD, WBDTOSVD),
*
*          where SIZEB = MAX(M,N), and WATOBD and WBDTOSVD refer,
*          respectively, to the workspace required to bidiagonalize
*          the matrix A and to go from the bidiagonal matrix to the
*          singular value decomposition U*S*VT.
*
*          For WATOBD, the following holds:
*
*          WATOBD = MAX(MAX(WPDLANGE,WPDGEBRD),
*                       MAX(WPDLARED2D,WP(pre)LARED1D)),
*
*          where WPDLANGE, WPDLARED1D, WPDLARED2D, WPDGEBRD are the
*          workspaces required respectively for the subprograms
*          PDLANGE, PDLARED1D, PDLARED2D, PDGEBRD. Using the
*          standard notation
*
*          MP = NUMROC( M, MB, MYROW, DESCA( CTXT_ ), NPROW),
*          NQ = NUMROC( N, NB, MYCOL, DESCA( LLD_ ), NPCOL),
*
*          the workspaces required for the above subprograms are
*
*          WPDLANGE = MP,
*          WPDLARED1D = NQ0,
*          WPDLARED2D = MP0,
*          WPDGEBRD = NB*(MP + NQ + 1) + NQ,
*
*          where NQ0 and MP0 refer, respectively, to the values obtained
*          at MYCOL = 0 and MYROW = 0. In general, the upper limit for
*          the workspace is given by a workspace required on
*          processor (0,0):
*
*          WATOBD <= NB*(MP0 + NQ0 + 1) + NQ0.
*
*          In case of a homogeneous process grid this upper limit can
*          be used as an estimate of the minimum workspace for every
*          processor.
*
*          For WBDTOSVD, the following holds:
*
*          WBDTOSVD = SIZE*(WANTU*NRU + WANTVT*NCVT) +
*                     MAX(WDBDSQR,
```

```
*                         MAX(WANTU*WPDORMBRQLN, WANTVT*WPDORMBRPRT)),
*
*          where
*
*                          1, if left(right) singular vectors are wanted
*          WANTU(WANTVT) =
*                          0, otherwise
*
*          and WDBDSQR, WPDORMBRQLN and WPDORMBRPRT refer respectively
*          to the workspace required for the subprograms DBDSQR,
*          PDORMBR(QLN), and PDORMBR(PRT), where QLN and PRT are the
*          values of the arguments VECT, SIDE, and TRANS in the call
*          to PDORMBR. NRU is equal to the local number of rows of
*          the matrix U when distributed 1-dimensional "column" of
*          processes. Analogously, NCVT is equal to the local number
*          of columns of the matrix VT when distributed across
*          1-dimensional "row" of processes. Calling the LAPACK
*          procedure DBDSQR requires
*
*          WDBDSQR = MAX(1, 4*SIZE )
*
*          on every processor. Finally,
*
*          WPDORMBRQLN = MAX( (NB*(NB-1))/2, (SIZEQ+MP)*NB)+NB*NB,
*          WPDORMBRPRT = MAX( (MB*(MB-1))/2, (SIZEP+NQ)*MB )+MB*MB,
*
*          If LWORK = -1, then LWORK is global input and a workspace
*          query is assumed; the routine only calculates the minimum
*          size for the work array. The required workspace is returned
*          as the first element of WORK and no error message is issued
*          by PXERBLA.
*
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*
*          > 0:  if DBDSQR did not converge
*                If INFO = MIN(M,N) + 1, then PDGESVD has detected
*                heterogeneity by finding that eigenvalues were not
*                identical across the process grid. In this case, the
*                accuracy of the results from PDGESVD cannot be
*                guaranteed.
*
*  =====================================================================
*
*  The results of PDGEBRD, and therefore PDGESVD, may vary slightly
*  from run to run with the same input data. If repeatability is an
*  issue, call BLACS_SET with the appropriate option after defining
*  the process grid.
*
*  Alignment requirements
*  ======================
*
*  The routine PDGESVD inherits the same alignement requirement as
*  the routine PDGEBRD, namely:
*
*  The distributed submatrix sub( A ) must verify some alignment proper-
*  ties, namely the following expressions should be true:
*  ( MB_A.EQ.NB_A .AND. IROFFA.EQ.ICOFFA )
*          where NB = MB_A = NB_A,
*          IROFFA = MOD( IA-1, NB ), ICOFFA = MOD( JA-1, NB ),
*
```

```
*  =====================================================================
```

D.2.3. Rutina PDSTEBZ

```
SUBROUTINE PDSTEBZ( ICTXT, RANGE, ORDER, N, VL, VU, IL, IU,
$                   ABSTOL, D, E, M, NSPLIT, W, IBLOCK, ISPLIT,
$                   WORK, LWORK, IWORK, LIWORK, INFO )
*
*  Purpose
*  =======
*
*  PDSTEBZ computes the eigenvalues of a symmetric tridiagonal matrix in
*  parallel. The user may ask for all eigenvalues, all eigenvalues in
*  the interval [VL, VU], or the eigenvalues indexed IL through IU. A
*  static partitioning of work is done at the beginning of PDSTEBZ which
*  results in all processes finding an (almost) equal number of
*  eigenvalues.
*
*  NOTE : It is assumed that the user is on an IEEE machine. If the user
*         is not on an IEEE mchine, set the compile time flag NO_IEEE
*         to 1 (in SLmake.inc). The features of IEEE arithmetic that
*         are needed for the "fast" Sturm Count are : (a) infinity
*         arithmetic (b) the sign bit of a single precision floating
*         point number is assumed be in the 32nd bit position
*         (c) the sign of negative zero.
*
*  See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal
*  Matrix", Report CS41, Computer Science Dept., Stanford
*  University, July 21, 1966.
*
*  Arguments
*  =========
*
*  ICTXT   (global input) INTEGER
*          The BLACS context handle.
*
*  RANGE   (global input) CHARACTER
*          Specifies which eigenvalues are to be found.
*          = 'A': ("All")   all eigenvalues will be found.
*          = 'V': ("Value") all eigenvalues in the interval
*                           [VL, VU] will be found.
*          = 'I': ("Index") the IL-th through IU-th eigenvalues (of the
*                           entire matrix) will be found.
*
*  ORDER   (global input) CHARACTER
*          Specifies the order in which the eigenvalues and their block
*          numbers are stored in W and IBLOCK.
*          = 'B': ("By Block") the eigenvalues will be grouped by
*                              split-off block (see IBLOCK, ISPLIT) and
*                              ordered from smallest to largest within
*                              the block.
*          = 'E': ("Entire matrix")
*                              the eigenvalues for the entire matrix
*                              will be ordered from smallest to largest.
*
*  N       (global input) INTEGER
*          The order of the tridiagonal matrix T.  N >= 0.
*
```

```
*  VL      (global input) DOUBLE PRECISION
*          If RANGE='V', the lower bound of the interval to be searched
*          for eigenvalues.  Eigenvalues less than VL will not be
*          returned.  Not referenced if RANGE='A' or 'I'.
*
*  VU      (global input) DOUBLE PRECISION
*          If RANGE='V', the upper bound of the interval to be searched
*          for eigenvalues.  Eigenvalues greater than VU will not be
*          returned.  VU must be greater than VL.  Not referenced if
*          RANGE='A' or 'I'.
*
*  IL      (global input) INTEGER
*          If RANGE='I', the index (from smallest to largest) of the
*          smallest eigenvalue to be returned.  IL must be at least 1.
*          Not referenced if RANGE='A' or 'V'.
*
*  IU      (global input) INTEGER
*          If RANGE='I', the index (from smallest to largest) of the
*          largest eigenvalue to be returned.  IU must be at least IL
*          and no greater than N.  Not referenced if RANGE='A' or 'V'.
*
*  ABSTOL  (global input) DOUBLE PRECISION
*          The absolute tolerance for the eigenvalues.  An eigenvalue
*          (or cluster) is considered to be located if it has been
*          determined to lie in an interval whose width is ABSTOL or
*          less.  If ABSTOL is less than or equal to zero, then ULP*|T|
*          will be used, where |T| means the 1-norm of T.
*          Eigenvalues will be computed most accurately when ABSTOL is
*          set to the underflow threshold DLAMCH('U'), not zero.
*          Note : If eigenvectors are desired later by inverse iteration
*          ( PDSTEIN ), ABSTOL should be set to 2*PDLAMCH('S').
*
*  D       (global input) DOUBLE PRECISION array, dimension (N)
*          The n diagonal elements of the tridiagonal matrix T.  To
*          avoid overflow, the matrix must be scaled so that its largest
*          entry is no greater than overflow**(1/2) * underflow**(1/4)
*          in absolute value, and for greatest accuracy, it should not
*          be much smaller than that.
*
*  E       (global input) DOUBLE PRECISION array, dimension (N-1)
*          The (n-1) off-diagonal elements of the tridiagonal matrix T.
*          To avoid overflow, the matrix must be scaled so that its
*          largest entry is no greater than overflow**(1/2) *
*          underflow**(1/4) in absolute value, and for greatest
*          accuracy, it should not be much smaller than that.
*
*  M       (global output) INTEGER
*          The actual number of eigenvalues found. 0 <= M <= N.
*          (See also the description of INFO=2)
*
*  NSPLIT  (global output) INTEGER
*          The number of diagonal blocks in the matrix T.
*          1 <= NSPLIT <= N.
*
*  W       (global output) DOUBLE PRECISION array, dimension (N)
*          On exit, the first M elements of W contain the eigenvalues
*          on all processes.
*
*  IBLOCK  (global output) INTEGER array, dimension (N)
```

```
*          At each row/column j where E(j) is zero or small, the
*          matrix T is considered to split into a block diagonal
*          matrix.  On exit IBLOCK(i) specifies which block (from 1
*          to the number of blocks) the eigenvalue W(i) belongs to.
*          NOTE:  in the (theoretically impossible) event that bisection
*          does not converge for some or all eigenvalues, INFO is set
*          to 1 and the ones for which it did not are identified by a
*          negative block number.
*
*  ISPLIT  (global output) INTEGER array, dimension (N)
*          The splitting points, at which T breaks up into submatrices.
*          The first submatrix consists of rows/columns 1 to ISPLIT(1),
*          the second of rows/columns ISPLIT(1)+1 through ISPLIT(2),
*          etc., and the NSPLIT-th consists of rows/columns
*          ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.
*          (Only the first NSPLIT elements will actually be used, but
*          since the user cannot know a priori what value NSPLIT will
*          have, N words must be reserved for ISPLIT.)
*
*  WORK    (local workspace) DOUBLE PRECISION array,
*          dimension ( MAX( 5*N, 7 ) )
*
*  LWORK   (local input) INTEGER
*          size of array WORK must be >= MAX( 5*N, 7 )
*          If LWORK = -1, then LWORK is global input and a workspace
*          query is assumed; the routine only calculates the minimum
*          and optimal size for all work arrays. Each of these
*          values is returned in the first entry of the corresponding
*          work array, and no error message is issued by PXERBLA.
*
*  IWORK   (local workspace) INTEGER array, dimension ( MAX( 4*N, 14 ) )
*
*  LIWORK  (local input) INTEGER
*          size of array IWORK must be >= MAX( 4*N, 14, NPROCS )
*          If LIWORK = -1, then LIWORK is global input and a workspace
*          query is assumed; the routine only calculates the minimum
*          and optimal size for all work arrays. Each of these
*          values is returned in the first entry of the corresponding
*          work array, and no error message is issued by PXERBLA.
*
*  INFO    (global output) INTEGER
*          = 0 :  successful exit
*          < 0 :  if INFO = -i, the i-th argument had an illegal value
*          > 0 :  some or all of the eigenvalues failed to converge or
*                 were not computed:
*              = 1 : Bisection failed to converge for some eigenvalues;
*                    these eigenvalues are flagged by a negative block
*                    number.  The effect is that the eigenvalues may not
*                    be as accurate as the absolute and relative
*                    tolerances. This is generally caused by arithmetic
*                    which is less accurate than PDLAMCH says.
*              = 2 : There is a mismatch between the number of
*                    eigenvalues output and the number desired.
*              = 3 : RANGE='i', and the Gershgorin interval initially
*                    used was incorrect. No eigenvalues were computed.
*                    Probable cause: your machine has sloppy floating
*                    point arithmetic.
```

```
*                   Cure: Increase the PARAMETER "FUDGE", recompile,
*                   and try again.
*
*  Internal Parameters
*  ===================
*
*  RELFAC  DOUBLE PRECISION, default = 2.0
*          The relative tolerance.  An interval [a,b] lies within
*          "relative tolerance" if  b-a < RELFAC*ulp*max(|a|,|b|),
*          where "ulp" is the machine precision (distance from 1 to
*          the next larger floating point number.)
*
*  FUDGE   DOUBLE PRECISION, default = 2.0
*          A "fudge factor" to widen the Gershgorin intervals.  Ideally,
*          a value of 1 should work, but on machines with sloppy
*          arithmetic, this needs to be larger.  The default for
*          publicly released versions should be large enough to handle
*          the worst machine around.  Note that this has no effect
*          on the accuracy of the solution.
*
*  =====================================================================
```