Vol. **16**, **2**, 227–249 (2000)

Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

Cálculo de los valores propios de matrices tridiagonales simétricas mediante la iteración de Laguerre

José M. Badía Contelles

Departamento de Informática
Universidad Jaume I de Castellón
12071 Castellón, España
Tel.: 34-964-72 82 95, Fax: 34-964-72 84 35
e-mail: badia@inf.uji.es

Antonio M. Vidal Maciá

Departamento de Informática
Universidad Politécnica de Valencia
s [Version 1.0]: Basics, fonts, en 46071 Valencia, España
Tel.: 34-963-87 70 00, Fax: 34-963-87 73 58
e-mail: avidal@dsic.uji.es

Resumen

En este trabajo se proponen dos nuevos algoritmos que utilizan la iteración de Laguerre para calcular los valores propios de una matriz tridiagonal simétrica. El primer algoritmo aplica esta iteración en el marco del método de bisección, logrando unas prestaciones mejores que las de cualquier otro método de cálculo de raíces usado hasta ahora. El segundo algoritmo utiliza el paradigma divide y vencerás y combina la utilización de modificaciones de rango uno con la iteración de Laguerre. Estos dos nuevos algoritmos se comparan con otros cinco que utilizan diferentes métodos para resolver el mismo problema de valores propios. Desde el punto de vista experimental se analiza cómo influyen las características de las matrices en las prestaciones de los algoritmos. Los *clusters* y los valores propios ocultos son determinantes en el comportamiento de los algoritmos de bisección, mientras que las prestaciones de los algoritmos de tipo divide y vencerás dependen fundamentalmente del número de deflaciones detectadas. También probamos que nuestro algoritmo de bisección es más rápido que los basados en el método de divide y vencerás en casi todos los casos y proporciona más precisión en el cálculo de los valores propios que la iteración QR.[†]

Palabras clave: Iteración de Laguerre, matrices tridiagonales simétricas, valores propios, método de bisección, método divide y vencerás.

EXPLOITING THE LAGUERRE ITERATION FOR SOLVING THE SYMMETRIC TRIDIAGONAL EIGENPROBLEM

Summary

In this paper we propose two new algorithms that apply the Laguerre iteration for computing the eigenvalues of symmetric tridiagonal matrices. The first algorithm applies this iteration in the frame of the bisection method and the results obtained surpass the best root-finding methods used so far. The second algorithm uses a divide-and-conquer algorithm that combines rank-one modifications with the Laguerre iteration. We compare these two new algorithms with other five algorithms that use different methods for solving the same eigenproblem. In the experimental analysis of the algorithms we focus on the influence of characteristics of the matrices in their performance. Clusters and hidden eigenvalue determine the behavior of the bisection

[†] El presente trabajo se integra en el proyecto CICYT " Algoritmos paralelos para el cálculo de los valores propios de matrices dispersas y estructuradas" (TIC96-1062-C03).

 ISSN: 0213–1315 Recibido: Marzo 1999

algorithms, while the performance of the divide-and-conquer algorithms depend on the number of deflations detect. We also show that our bisection algorithm is faster than divide-and-conquer algorithms almost in all cases and provides more accurate eigenvalues than the QR iteration.

Keywords: Laguerre iteration, symmetric tridiagonal matrices, eigenvalues, bisection method, divide-and-conquer method.

INTRODUCCIÓN

El cálculo de los valores propios de matrices tridiagonales simétricas es especialmente importante, no sólo debido a la importancia intrínseca de esta clase de matrices, sino también porque aparecen en la solución del problema de valores propios de matrices simétricas generales.

Existen numerosos métodos para calcular los valores propios, y a veces los vectores propios, de matrices tridiagonales simétricas. Probablemente el más utilizado, debido a sus excelentes prestaciones para calcular todos los valores propios, es la iteración QR. Sin embargo, recientemente, con la consolidación de los ordenadores con arquitecturas paralelas otros métodos están adquiriendo una relevancia similar. Podemos señalar por ejemplo el método de homotopía, los métodos de tipo divide y vencerás y el método de bisección. Cada una de estas aproximaciones tiene ventajas e inconvenientes en lo que respecta al tiempo de ejecución, la precisión de los resultados, la posibilidad de evaluar sólo una parte del espectro de una matriz, etc.

En este artículo estudiamos los resultados obtenidos cuando se aplican las iteraciones de Laguere y quasi-Laguerre a los métodos de tipo divide y vencerás y al método de bisección para resolver el problema tridiagonal simétrico de valores propios. Presentamos dos nuevos algoritmos que usan ambas clases de iteraciones y comparamos estos algoritmos con otros previamente desarrollados por otros autores y con algunas de las rutinas del LAPACK.

Una de las principales características de ambos métodos, divide y vencerás y bisección, es su dependencia del problema. Más específicamente, dependen de las características de las matrices cuyos valores propios se quiere calcular. En el caso del método de bisección, el número de *clusters* de valores propios y el número de valores propios ocultos determina el tipo de ejecución, mientras que en el método divide y vencerás este parámetro depende del número de deflaciones detectadas. En este trabajo hemos elegido un conjunto amplio de matrices de prueba que contiene 12 tipos de matrices con diferentes características espectrales. Durante el análisis experimental de los algoritmos siempre hemos intentando relacionar las prestaciones con las características de cada tipo de matriz.

En el caso del método de bisección la velocidad del algoritmo depende directamente de la técnica de acelaración usada en la aproximación final de los valores propios. En este trabajo se proponen dos nuevos métodos de búsqueda de raíces para llevar a cabo esta tarea y se comparan con algunos de los métodos más comúnmente utilizados hasta ahora. Los resultados obtenidos muestran que la aplicación de la iteración de Laguerre ofrece las mejores prestaciones.

En el caso del método divide y vencerás proponemos un nuevo algoritmo que combina el uso de modificaciones de rango uno con la iteración de Laguerre. Este algoritmo se compara con otros algoritmos de tipo divide y vencerás basados en diferentes modificaciones en la fase de la división y distintos métodos para la aproximación final de los valores propios.

El resto del artículo está organizado de la siguiente forma: en la próxima sección presentamos una amplia revisión del trabajo previo que refleja el estado del arte y damos una visión general de la base teórica de los algoritmos. A continuación se describe tanto el algoritmo de bisección como un algoritmo genérico de tipo divide y vencerás. Posteriormente mostramos como aplicar la iteración de Laguerre para aproximar los valores propios. Se sigue con

una revisión de las características de las matrices de prueba y se analizan los resultados experimentales. Por último, se ofrecen las principales conclusiones de este trabajo.

ESTADO DEL ARTE

En los últimos años diferentes autores han desarrollado numerosos algoritmos para resolver el problema tridiagonal simétrico de valores propios. Sin embargo, todos estos algoritmos pueden ser clasificados en cuatro métodos básicos: la iteración QR, el método de homotopía, los métodos de bisección y multisección (BM) y el método divide y vencerás (DV). En esta sección revisaremos el origen y las principales contribuciones para cada uno de estos métodos. Nos centraremos especialmente en los dos últimos métodos (BM y DV), describiendo muy brevemente las bases teóricas de los algoritmos que se han implementado.

Se han presentado varios trabajos que comparan algunos de los métodos previamente citados. Podríamos citar por ejemplo el artículo[19], donde los autores presentan algunas ideas generales que deben seguirse para diseñar buenas rutinas de cálculo de valores propios en el caso simétrico. Estas ideas son exigidas por ejemplo a las rutinas incluidas en el LAPACK[3]. También puede encontrarse un estudio comparativo práctico de cuatro métodos diferentes en[34].

La iteración QR[26] ha sido considerada como el método más eficiente para calcular todos los valores propios de una matriz. En el caso tridiagonal simétrico el coste de la factorización QR es $O(n)$ por iteración. Sin embargo, no hay implementaciones paralelas escalables eficientes de este método, por ello con el auge de los ordenadores con arquitecturas paralelas se han venido considerando otros tipos de métodos. Más aún, métodos tales como DV y BM ofrecen prestaciones competitivas incluso en el caso secuencial y además proporcionan habitualmente resultados más precisos.

Un método que se está estudiando actualmente es el método de homotopía, también llamado método de continuación. Este método ha recibido atención recientemente debido a su alto grado de paralelismo intrínseco. Por tanto, los algoritmos basados en él son excelentes candidatos para ser utilizados en los modernos multiprocesadores. El método ha sido aplicado tanto al caso simétrico como no simétrico como puede verse por ejemplo en las referencias[30,32,36,33,35,38].

Antecedentes del método de bisección

El método de bisección ha dado lugar a un gran número de implementaciones debido a su flexibilidad para calcular una parte del espectro de una matriz y debido también a su paralelismo natural. Los métodos de bisección y multisección en el caso tridiagonal están basados en la posibilidad de dividir el espectro de una matriz. Más concretamente, es posible definir una función denominada $neg_n(c)$ que calculada para un valor cualquiera c proporcione el número de valores propios a la izquierda y a la derecha de este punto. Esta idea fue explotada inicialmente por Wallace Givens[23,24]. Este autor utilizó las propiedades de las secuencias de Sturm[26] como método para dividir el espectro.

Si definimos una matriz tridiagonal simétrica $\mathbf{T}$ como

$$\mathbf{T} = \begin{bmatrix} a_1 & b_1 & & & 0 \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ 0 & & & b_{n-1} & a_n \end{bmatrix} \tag{1}$$

y definimos la secuencia de Sturm de la matriz en el punto c como

$$\{p_0(c), p_1(c), \ldots, p_n(c)\} \tag{2}$$

con

$$\begin{cases} p_0(c) = 1 \\ p_i(c) = \det(T_i - cI) \qquad i = 1, 2, \ldots, n \end{cases}$$

siendo T_i la submatriz guía principal de dimensión ixi de la matriz $\mathbf{T}$, entonces el número de cambios de signo de esta secuencia es igual al número de valores propios de $\mathbf{T}$ menores que c.

El cálculo de esta secuencia puede hacerse utilizando la siguiente recurrencia bien conocida

$$\begin{cases} p_0(c) = 1, \quad p_1(c) = a_1 - c \\ p_i(c) = (a_i - c)p_{i-1}(c) - b_{i-1}^2 p_{i-2}(c) \qquad i = 1, 3, \ldots, n \end{cases} \tag{3}$$

El método original propuesto por Givens fue mejorado en trabajos posteriores por diferentes autores[28,29,47] utilizando diversas técnicas de aceleración.

En las referencias[47,48] se establece que el método de bisección es una muy buena técnica para aislar valores propios, pero que puede ser notablemente acelerado si se combina con algún método de cálculo de raíces que converja más rápido que la bisección pura. Wilkinson cita como ejemplo el método de Newton (convergencia cuadrática) y el método de Laguerre (convergencia cúbica).

Posteriormente se presentaron algunas versiones mejoradas del método de bisección. Especialmente importante es el trabajo de la referencia[11], donde se utiliza por primera vez una versión modificada de la secuencia de Sturm que evita los problemas de desbordamiento de la secuencia original.

Concretamente, si definimos

$$q_i(c) = \frac{p_i(c)}{p_{i-1}(c)} \qquad i = 1, 2, \ldots, n \tag{4}$$

puede demostrarse fácilmente que el número de elementos negativos de esta secuencia es igual al número de valores propios de $\mathbf{T}$ menores que c. Así pues, podemos llamar a este valor $neg_n(c)$.

La secuencia (4) se puede calcular utilizando la siguiente recurrencia

$$\begin{cases} q_0(c) = 1, \quad q_1(c) = a_1 - c \\ q_i(c) = (a_i - c) - \dfrac{b_{i-1}^2}{q_{i-1}(c)} \qquad i = 2, 3, \ldots, n \end{cases} \tag{5}$$

La rutina implementada en la referencia[11] ha servido como base y referencia de casi todos los algoritmos posteriores que utilizan el método de bisección. Efectivamente, una versión de esta subrutina se incluyó en la librería EISPACK[4] con el nombre *bisect*.

El uso de la secuencia modificada de Sturm (4) no está en cambio exento de problemas. Entre las dificultades que presenta la aplicación de esta secuencia podemos señalar por ejemplo que la recurrencia (5) falla si alguno de sus elementos es cero.

Algunos autores[12,18] han resuelto este problema utilizando diferentes soluciones, aunque la idea básica de todas ellas es reemplazar el elemento nulo por un valor muy pequeño *eps*, que se puede elegir como la precisión de la máquina. Debido a las propiedades especiales

del problema simétrico de valores propios, esta modificación no afecta sustancialmente a la precisión de los resultados obtenidos.

En la referencia[13] el método BM se divide en dos fases. Durante la fase de aislamiento se calculan, aplicando el algoritmo de bisección pura, aquellos intervalos que sólo contienen un valor propio. Posteriormente, durante la fase de extracción, se puede utilizar un algoritmo de cálculo de raíces más rápido para calcular los valores propios como los ceros de la función $q(x)$ definida como el último elemento de la recurrencia (5). Por ejemplo, en la referencia[42] se usa una combinación de los algoritmos de bisección y secante y de la iteración de Newton. En la referencia[39] se sugiere cualquier variación del método de la secante. En la referencia[37] se aplica el método de Newton y la iteración Zeroin. En la referencia[10] se aplica interpolación polinómica cúbica así como la iteración de Newton de primer y segundo orden. En la referencia[43] se utiliza una modificación de rutina clásica de Newton y finalmente en las referencias[5,9] se aplica la iteración de Laguerre y se compara con algunas de las rutinas previas.

Más aún, algunos autores dividen la fase de extracción en dos subfases. Por ejemplo, en la referencia[12] los autores inician el proceso de extracción aplicando varios pasos de bisección hasta que obtienen un intervalo donde el polinomio característico es monótono. Entonces utilizan el método Pegasus[21] para aproximar los valores propios con una convergencia rápida. En la referencia[45] el autor utiliza también el método de Pegasus, pera para calcular los valores propios de matrices Toeplitz hermitianas.

Por último, debemos subrayar que la aplicación de los métodos de aceleración descritos anteriormente no siempre produce una convergencia más rápida hacia los valores propios. A menudo los valores propios de matrices tridiagonales tienden a agruparse en *clusters* (conjuntos de valores propios numéricamente indistinguibles). En estos casos la aplicación de técnicas de aceleración puede ser menos eficiente que el uso del algoritmo de bisección pura.

Otra característica a tener en cuenta cuando se implementa un algoritmo que utiliza el método de bisección son los valores propios ocultos[39]. Decimos que un valor propio de una matriz tridiagonal simétrica $\mathbf{T}$ de tamaño n es un valor propio oculto cuando está muy próximo a un valor propio de su submatriz guía principal de tamaño $n-1$. En este caso el denominador y el numerador de la expresión (4) tienden simultáneamente a cero.

Por otra parte debemos subrayar el hecho de que el método de bisección puede ser aplicado al caso general simétrico, aunque con un coste muy alto. Sin embargo, hay algúnos tipos de matrices estructuradas, denominadas eficientemente estructuradas en la referencia[46], para que el coste de este método pueda ser notablemente reducido. Recientemente se han llevado a cabo ciertas implementaciones secuenciales del método en el caso de matrices de Toeplitz[45]. Además, también hay algunas implementaciones paralelas del método de bisección en el caso de matrices de Toeplitz banda[7] y en el caso de matrices Toeplitz más Hankel[6].

Antecedentes del método divide y vencerás

Varios autores han diseñado e implementado algoritmos que aplican métodos de tipo divide y vencerás para el cálculo de los valores propios de matrices tridiagonales simétricas utilizando diferentes aproximaciones. Las soluciones obtenidas hasta ahora pueden ser clasificadas en función de tres aspectos: la clase de modificación aplicada durante la fase de división, el método usado para evaluar el polinomio característico durante la fase de actualización y la técnica de búsqueda de raíces utilizada para aproximar los valores propios.

La idea básica de todos los algoritmos de tipo divide y vencerás es muy similar. Supongamos que se quiere calcular los valores propios de una matriz tridiagonal simétrica por (1).

Empezamos aplicando algún tipo de modificación M de forma que

$$\tilde{T} = \mathbf{T} + M = \begin{pmatrix} \tilde{T}_0 & 0 \\ 0 & \tilde{T}_1 \end{pmatrix} \tag{6}$$

Normalmente se elige M como una modificación de rango uno o dos. En el caso de las modificaciones de rango uno, una posible elección de M es $\alpha w w^T$ donde $\alpha = b_k$, el k-ésimo elemento de la subdiagonal de $\mathbf{T}$ y $w = (0, \ldots, 1, 1, \ldots, 0)^T$ un vector de dimensión n.

Así hemos dividido la matriz original $\mathbf{T}$ en dos submatrices tridiagonales simétricas $\tilde{T}_0$ y $\tilde{T}_1$. Ahora tenemos que calcular los valores propios de estas submatrices y utilizarlos para aproximar los valores propios de la matriz original $\mathbf{T}$.

Denotemos por $\tilde{d}_i$ ciertos números que separan los valores de $\tilde{T}$ y por λ_i los valores propios de $\mathbf{T}$ con $i = 1, 2, \ldots, n$. Hasta ahora varios autores han dividido $\mathbf{T}$ usando modificaciones M de rango uno o rango dos. En ambos casos, la posición de los separadores $\tilde{d}_i$ define intervalos que delimitan la posición de los valores propios λ_i[26,34]. Nuestro objetivo es aproximar los valores propios utilizando algún tipo de función que se anule en cada uno de ellos en los intervalos definidos por los separadores.

El primer autor que utilizó el método fue Cuppen[17], teniendo en cuenta las ideas de las referencias[16,25,48]. Cuppen utilizó modificaciones de rango uno para dividir la matriz original y aproximó los valores propios a partir de los separadores utilizando la función secular. Más concretamente, Cuppen utilizó la técnica de interpolación racional para realizar esta aproximación.

Otra clase de método divide y vencerás fue propuesto en las referencias[14,15]. En este caso los autores utilizan secuencias de Sturm para aproximar las raíces del polinomio característico. En las referencias[18,40,41] los autores usan diferentes combinaciones del método de Newton y del método de bisección para llevar a cabo la aproximación. Más recientemente, en la referencia[5], se utiliza la iteración de Laguerre para realizar esta tarea.

En otro artículo reciente[34] los autores utilizan modificaciones de rango dos para abordar la fase de división. En dicho artículo se aplica también la iteración de Laguerre para aproximar los valores propios. Por último, en la referencia[22] se utiliza un algoritmo de tipo quasi-Laguerre para realizar la aproximación final.

Cuando se diseñan algoritmos divide y vencerás y cuando se analizan sus prestaciones, un factor muy importante a tener en cuenta es la deflación. Decimos que ocurre una deflación cuando podemos obtener directamente un valor propio de $\mathbf{T}$ a partir de los separadores sin ningún proceso de aproximación.

Podemos definir dos clases de deflación: supongamos que se sabe que valor propio está situado entre dos separadores contiguos $\tilde{d}_i$ y $\tilde{d}_{i+1}$. La primera clase de deflación ocurre cuando λ_i es igual a uno de estos separadores o la distancia entre el separador y el valor propio es menor que un número muy pequeño Δ. La segunda clase de deflación ocurre cuando la distancia entre dos separadores es menor que 2Δ, y por tanto podemos aproximar los valores propios utilizando simplemente el punto medio de los separadores.

ALGORITMO DE BISECCIÓN

Una de las características fundamentales del método de bisección es su flexibilidad. Mientras otros métodos como la iteración QR o el método de Cuppen nos obligan a calcular todos los valores propios, el método BM permite calcular cualquier valor propio determinado o un grupo consecutivo de ellos.

La idea básica del método de bisección es el uso adecuado de la función $neg_n(c)$ y la secuencia de Sturm modificada (5). Para implementar este algoritmo empezamos con un intervalo inicial (a, b) que contienen todos los valores propios que se quiere calcular. Este intervalo se puede obtener por ejemplo por medio de los círculos de Gershgorin[26].

Cuando bisectamos el intervalo inicial, definimos dos subintervalos (a, b) y (b, c), y del cálculo de $n_c = neg_n(c)$ podemos conocer los valores propios contenidos en cada uno de ellos. Si repetimos este proceso, se puede obtener finalmente un intervalo (a, b) que aísla un valor propio o que contienen un *cluster* de valores propios con una separación menor que una cota dada.

Una vez aislado el valor propio, podemos usar la función $q(x)$ definida por el último elemento de la secuencia de Sturm modificada. Podemos entonces aplicar un método de cálculo de raíces más rápido que la bisección pura, para extraer el valor propio aislado, como la única raíz de esta función en el intervalo.

Después de definir como se aproxima cada valor propio individual, podemos describir como explotar estas ideas para extraer cualquier grupo de valores propios consecutivos. Para abordar esta tarea utilizaremos una cola de intervalos que contienen valores propios. Durante el proceso de aislamiento de cada valor propio, los intervalos producidos que contienen cualquiera de los valores propios que deben ser calculados se almacenan en una cola. Concretamente, la cola contiene los límites (a, b) de los intervalos y el valor de la función $neg_n(c)$ en estos puntos (n_a, n_b). Cada vez que extraemos un valor propio o un *cluster*, empezamos de nuevo el proceso con un intervalo nuevo tomado de la cola para aproximar el siguiente valor propio. El algoritmo `calcula-grupo` resume este proceso de cálculo de un grupo cualquiera de valores propios. En este algoritmo, un intervalo de la forma (a^r, a^s) contiene todos los valores propios comprendidos entre λ_{r+1} y λ_r y tol es un valor que se puede definir de la misma forma que se hace en el criterio de parada (10).

```
programa calcula-grupo (p, q, tol, λ, (p : q))
/* Dados dos enteros p y q tales que 1 <= p < q <= n, este programa calcula
todos los valores propios de T entre λ_p y λ_q */
  Calcula el intervalo inicial (a_r, a_s) usando círculos de Gershgorin
  Almacena el intervalo inicial en la cola
  mientras cola no vacía
    si ((a_r, a_s) está incluido en (a_{p-1}, a_q)) y ((s = r + 1))entonces
    /* contiene sólo un valor propio de T */
    extraer (a_r, a_s, tol, λ_s)
    si cola no vacía entonces
      obtener un nuevo intervalo (a_r, a_s) de la cola
    finsi
  sino
    si |a_r − a_s| < tol entonces /* cluster de valores propios */
      λ_i = (a_r + a_s)/2,    i = r + 1, ..., s
    sino
      c = (a_r + a_s)/2;    k = neg_n(c)
      si (k = r) o (k <= p − 1) entonces
       r = k;    a_r = c
      sino si (k = s) o (k >= q) entonces
       s = k;    a_s = c
      sino
       almacena el intervalo (c, a_s) en la cola
       s = k;    a_s = c
      finsi
    finsi
  finsi
finmientras
```

El algoritmo utilizado para implementar la función `extraer` $(a_r, a_s, tol, \lambda_s)$ condiciona la velocidad del método, excepto cuando tenemos muchos *clusters* de valores propios. En la referencia[5] se comprueba que la iteración de Laguerre es la que mejores resultados proporciona al implementar esta función.

En este artículo implementamos dos nuevas técnicas de extracción y las comparamos con otras ampliamente utilizadas hasta ahora. Primero implementamos la iteración de Laguerre para calcular los ceros del polinomio característico en cada intervalo aislado, segundo implementamos un método que inicia la fase de extracción aplicando varios pasos de la iteración de Laguerre con el fin de obtener un intervalo donde la función $q(x)$ sea continua y por último aplicamos el método Pegasus para aproximar los valores propios como los ceros de esta función.

ALGORITMOS DIVIDE Y VENCERÁS

Todos los algoritmos de tipo divide y vencerás para calcular valores propios constan básicamente de tres fases. Durante la fase de división la matriz original se divide en dos submatrices utilizando alguna clase de modificación. Durante la segunda fase se calculan los valores propios de las submatrices. Por último, durante la fase de actualización (*updating*) del método se aproximan los valores propios de la matriz original a partir de los valores propios de sus submatrices.

El verdadero potencial del método divide y vencerás surge de la posibilidad de aplicarlo recursivamente. Una vez hemos dividido la matriz original, podemos calcular los valores propios de las submatrices utilizando cualquier método clásico. Por ejemplo, podemos utilizar la iteración QR, que ha sido considerada hasta ahora como el método secuencial más eficiente. Sin embargo, podemos también aproximar el cálculo de los valores propios de las submatrices aplicando de nuevo un esquema divide y vencerás. Este proceso se puede repetir en los sucesivos niveles hasta obtener matrices de tamaño 1×1 o 2×2 cuyos valores propios se pueden calcular fácilmente.

Supongamos que se aplican k etapas de división sobre una matriz de tamaño n, obteniéndose 2^k submatrices de orden $2^p = n/2^k$. El siguiente código secuencial define los principales pasos para calcular los valores propios de la matriz original utilizando un método divide y vencerás genérico. Las diferentes aproximaciones citadas como antecedentes del método se pueden obtener realizando los pasos $(^*)$ y $(^{**})$ de forma particular.

Método divide y vencerás

```
Dividir la matriz original en 2^k submatrices de tamaño 2^p (*)
Calcular los valores propios de las 2^k de tamaño 2^p
para i = 1 hasta k (niveles de división)
    para j = 1 hasta 2^(k-i) (pares de submatrices consecutivas)
      Mezclar y ordenar los valores propios de las submatrices 2j-1 y 2j
      de tamaño 2^(p+i-1) para que actúen como separadores de una matriz de
      tamaño 2^(p+i)
      para r = 1 hasta 2^(p+i)
        Aplicar algún método de cálculo de raíces para calcular el valor
        propio λ_r de la submatriz j de tamaño 2^(p+i) (**)
      finpara
    finpara
finpara
```

LAS ITERACIONES DE LAGUERRE Y QUASI-LAGUERRE

La aplicación de la iteración de Laguerre como método para acelerar la extracción de los valores propios ya fue sugerida por Wilkinson[48]. Esta técnica se ha utilizado en la referencia[34] como un paso básico en un método de tipo divide y vencerás. Se puede demostrar que el método de Laguerre tiene convergencia cúbica en un entorno de cualquier valor propio real simple.

Cada iteración del método de Laguerre se basa en la siguiente expresión

$$L_{\pm}(x) = x + \frac{n}{\left(-\frac{p'(x)}{p(x)}\right) \pm \sqrt{(n-1)\left[(n-1)\left(-\frac{p'(x)}{p(x)}\right)^2 - n\left(-\frac{p''(x)}{p(x)}\right)\right]}} \tag{7}$$

donde el polinomio característico $p(x) = \det(T - \lambda I)$ se puede evaluar como el n-ésimo término de la recurrencia (3). Como se puede ver en (7), para evaluar una iteración de Laguerre necesitamos no sólo $p(x)$, sino también sus derivadas primera $p'(x)$ y segunda $p''(x)$. Por las mismas razones que se calculaba la secuencia de Sturm modificada en lugar de la original, utilizamos las siguientes recurrencias escaladas relacionadas con cada derivada

$$r_i = \frac{p_i'}{p_i} \quad \text{y} \quad s_i = \frac{p_i''}{p_i} \qquad i = 0, 1, \ldots, n$$

Estas recurrencias se pueden obtener diferenciando (3) y escalando cada término

$$\begin{cases} r_0 = 0, \quad r_1 = -\dfrac{1}{q_1} \\ r_i = \dfrac{1}{q_i}\left[(a_i - \lambda) r_{i-1} - 1 - \left(\dfrac{b_{i-1}^2}{q_{i-1}}\right) r_{i-2}\right] \qquad i = 2, 3, \ldots, n \end{cases} \tag{8}$$

$$\begin{cases} s_0 = 0, \quad s_1 = 0 \\ s_i = \dfrac{1}{q_i}\left[(a_i - \lambda) s_{i-1} - 2r_{i-1} - \left(\dfrac{b_{i-1}^2}{q_{i-1}}\right) s_{i-2}\right] \qquad i = 2, 3, \ldots, n \end{cases} \tag{9}$$

Para calcular (7) utilizamos el último elemento de las recurrencias anteriores

$$\frac{p'(x)}{p(x)} = r_n \quad \text{y} \quad \frac{p''(x)}{p(x)} = s_n$$

Otro importante aspecto del método iterativo es el criterio de parada utilizado. A partir del análisis de la precisión que se hace en la referencia[34] para la iteración de Laguerre, hemos elegido la siguiente condición

$$|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}| \leq \max\{\delta, |\tilde{\lambda}_i^{(k)}|\varepsilon\} \tag{10}$$

donde $\tilde{\lambda}_i^{(k)}$ es la k-ésima aproximación al valor propio exacto λ_i, ε la precisión de la máquina y δ la cota de error absoluto dada por

$$\delta = 2,5\varepsilon \max_{1 \leq j \leq n-1} \{|b_j| + |b_{j+1}|\}$$

Por otra parte, la iteración quasi-Laguerre ha sido desarrollada en la referencia[22] como una alternativa a la iteración de Laguerre. En dicho artículo los autores utilizan la iteración quasi-Laguerre durante la fase de actualización del algoritmo divide y vencerás. Esta nueva

iteración ofrece mejores prestaciones que la iteración de Laguerre en muchos casos e incluso mejora las prestaciones de la iteración QR para alguna clase de matrices.

El principal objetivo de la iteración quasi-Laguerre es evitar la evaluación de $p''(x)$, que es relativamente cara en cuanto a tiempo de cálculo en el caso de la iteración de Laguerre. La nueva iteración mantiene la monotonía de la anterior. Sin embargo, mientras el algoritmo que utiliza la iteración de Laguerre converge cúbicamente en un entorno de los valores propios, el que se basa en la iteración quasi-Laguerre converge globalmente con una razón de convergencia final $\sqrt{2}+1$ para encontrar los ceros de cualquier polinomio con todos sus ceros reales[22].

La siguiente expresión nos proporciona la modificación aplicada en cada iteración quasi-Laguerre

$$x_{\pm}^{(k-1)} = x_{\pm}^{(k)} + \frac{2[n-(x_{\pm}^{(k-1)}-x_{\pm}^{(k)})q(x_{\pm}^{(k-1)})]}{-(x_{\pm}^{(k-1)}-x_{\pm}^{(k)})R - 2q(x_{\pm}^{(k-1)}) \pm \sqrt{R[(x_{\pm}^{(k-1)}-x_{\pm}^{(k)})^2 R + 4(n-1)]}} \qquad (11)$$

donde

$$R = n\left(\frac{q(x_{\pm}^{(k)}) - q(x_{\pm}^{(k-1)})}{x_{\pm}^{(k-1)} - x_{\pm}^{(k)}}\right) - q(x_{\pm}^{(k)})q(x_{\pm}^{(k-1)})$$

$x_{\pm}^{(k-1)}$ y $x_{\pm}^{(k)}$ son dos aproximaciones sucesivas al valor propio y $q(x) = q_n(x)$ en (5).

La iteración quasi-Laguerre, como la iteración de Laguerre, se puede modificar para acelerar su convergencia en un entorno de un *cluster* de valores propios, tal y como se puede ver en las referencias[34,22].

ANÁLISIS EXPERIMENTAL

Matrices de prueba

Una de las características fundamentales de los algoritmos que utilizan el método de bisección o métodos de tipo divide y vencerás es su dependencia del problema. Esto es, el comportamiento de los algoritmos depende en gran medida de las características de las matrices cuyos valores propios queremos calcular. Los resultados experimentales que hemos obtenido demuestran que los algoritmos probados son directamente dependientes de la distribución de los valores propios a lo largo del espectro. Es más, en el caso de los métodos de tipo bisección el comportamiento de los algoritmos depende del número de valores propios incluidos en *clusters* y de la presencia de valores propios ocultos. Por otro lado, el comportamiento de los algoritmos de tipo divide y vencerás depende del número de deflaciones detectadas durante su ejecución.

Por lo tanto, si queremos realizar un buen análisis experimental de los algoritmos, debemos escoger un conjunto apropiado de matrices de prueba. Las matrices incluidas en este conjunto deben tener diferentes distribuciones de los valores propios. En concreto es interesante el uso de matrices conteniendo *clusters* de valores propios y de aquellas cuyos valores propios den lugar a un número variable de deflaciones. Diversos autores han utilizado diferentes conjuntos de matrices[17,18,38]. Para obtener los resultados mostrados en este artículo hemos utilizado un conjunto que cumple todas las características citadas, en concreto el descrito en la referencia[34].

Algunas de las matrices utilizadas (tipos 1-5) tienen valores propios que pueden calcularse de modo exacto mediante el uso de diferentes fórmulas, otras (tipos 6 y 7) tienen valores propios que no son conocidos a priori. Por último, las matrices de los tipos 8 y 12 se generan mediante el uso de la rutina *dlatms* incluida en el paquete LAPACK[20].

Tipo	Elementos	Valores propios	Referencias
1	$a_i = a$ $b_i = b$	$\left\{ a + 2b\cos\dfrac{k\pi}{n+1} \right\}_{k=1}^{n}$	ref. 27, p. 137 ref. 18, p. 39
2	$a_1 = a - b;\ a_n = a + b$ $a_i = a;\ i = 2, \ldots, n-1$ $b_i = b;\ i = 1, \ldots, n-1$	$\left\{ a + 2b\cos\dfrac{(2k-1)\pi}{2n} \right\}_{k=1}^{n}$	ref. 27, p. 138 ref. 18, p. 38 ref. 17
3	$a_i = \begin{cases} a & \text{con } i \text{ impar} \\ b & \text{con } i \text{ par} \end{cases}$ $b_i = 1$	$\left\{ \dfrac{a + b \pm \sqrt{(a-b)^2 + 16\cos^2\frac{k\pi}{n+1}}}{2} \right\}_{k=1}^{\frac{n}{2}}$ y a si n es impar	ref. 27, p. 139
4	$a_i = 0$ $b_i = \sqrt{i(n-i)}$	$\{-n + 2k - 1\}_{k=1}^{n}$ Distribuidos uniformemente	ref. 27, p. 140 ref. 18, p. 38
5	$a_i = -[(2i-1)(n-1) - 2(i-1)^2]$ $b_i = i(n-1)$	$\{-k(k-1)\}_{k=1}^{n}$	ref. 27, p. 141 ref. 17
6	$a_i = \begin{cases} \dfrac{m}{2} - i + 1 & i = 1, \ldots, \dfrac{m}{2} \\ i - \dfrac{m}{2} & i = \dfrac{m}{2} + 1, \ldots, m \end{cases}$ $m = \begin{cases} n & \text{con } n \text{ par} \\ n+1 & \text{con } n \text{ impar} \end{cases}$ $b_i = 1$	La mayor parte de los valores propios emparejados en *clusters*	matrices W_n^+ de Wilkinson ref. 48, p.308
7	Generadas aleatoriamente con una distribución uniforme [0,1]	Aleatorios	
8	Generadas usando la rutina del LAPACK *dlamts*	Siguen una distribución aritmética	ref. 20
9		Distribución geométrica$\{q^k\}, k = 1, \ldots, n$ para algún $q \in (0,1)$	
10		Un valor propio 1, el resto en $(-\varepsilon, \varepsilon)$	
11		Distribuidos de modo equidistante en el intervalo [0,1] exceptuando un valor propio muy pequeño	
12		Distribuidos de modo equidistante en el intervalo $[10^{-12} - \varepsilon, 10^{-12} + \varepsilon]$ excepto un valor propio igual a 1	

Tabla I. Matrices de prueba

Algoritmos probados

En las siguientes secciones comparamos las presentaciones de siete algoritmos para el cálculo de todos los valores propios de matrices tridiagonales simétricas. Cuatro de estos algoritmos utilizan estrategias de tipo divide y vencerás, dos de ellos utilizan el método de bisección y el último utiliza el iterativo QR. Dos algoritmos, *dstrid* y *dstdv*, han sido implementados por los autores del presente estudio. Otros dos son implementaciones proporcionadas por otros autores, mientras los restantes tres algoritmos son rutinas del LAPACK. Todos los métodos han sido implementados y probados utilizando reales en doble precisión en el lenguaje Fortran sobre un procesador MIPS R10000 integrado en una arquitectura Silicon Graphics.

1. El algoritmo *dsterf* es la versión libre de raíces (*root-free*) del método iterativo QR incluido en el LAPACK. Esta versión del iterativo QR es la recomendada por los autores de dicho paquete cuando se quieren calcular tan sólo los valores propios.
2. El algoritmo *dstebz* es la rutina que utiliza el método de bisección incluida en el LAPACK. Esta rutina utiliza la bisección pura tanto para aislar como para extraer los valores propios.
3. El algoritmo *dstrid* es nuestra implementación del método de bisección en dos fases. Este algoritmo utiliza bisección pura para aislar los valores propios y un método de aproximación de raíces más rápido para su extracción. En un apartado posterior comparamos varios métodos de aproximación de raíces con el fin de determinar el más conveniente para abordar la fase de extracción.
4. El algoritmo *dstedc* es la rutina que utiliza el método divide y vencerás incluida en el paquete LAPACK[44]. Esta rutina implementa el método de Cuppen combinando modificaciones de rango uno con el uso de la interpolación racional para la aproximación final de los valores propios. Para poder utilizar esta rutina la hemos modificado ligeramente con el fin de que continúe la fase de división hasta llegar a matrices con tamaño 2×2.
5. El algoritmo *dspmg* es el algoritmo divide y vencerás implementado en la referencia[34]. Este algoritmo utiliza modificaciones de rango dos durante la fase de división y aplica la iteración de Laguerre durante la fase de reconstrucción.
6. El algoritmo *dsmsl* es el algoritmo divide y vencerás implementado en la referencia[22]. Este algoritmo modifica el *dspmg* aplicando la iteración quasi-Laguerre durante la fase de reconstrucción.
7. El algoritmo *dstdv* es nuestra implementación del método divide y vencerás. Este algoritmo utiliza modificaciones de rango uno durante la fase de división y aplica la iteración de Laguerre para calcular los ceros del polinomio característico durante la fase de construcción.

Características de las matrices de prueba

En este apartado mostramos las características de las matrices de prueba utilizadas que determinan el comportamiento de los algoritmos de bisección y de tipo divide y vencerás.

La Tabla II muestra el número de *clusters* (*cl*), los valores propios incluidos en estos *clusters* (*nvp*) y el número de valores propios ocultos (*hi*) para los 12 tipos de matrices de prueba.

Los resultados de la Tabla II nos permiten agrupar los 12 tipos de matrices de prueba en diferentes clases en función del número de *clusters* y de valores propios ocultos. Las matrices de los tipos 1, 2 y 3 no tienen ningún *cluster* ni ningún valor propio oculto. Las matrices de tipo 6 (matrices de Wilkinson) tienen la mayor parte de sus valores propios emparejados en

clusters. Todos los valores propios de las matrices de los tipos 6, 10 y 12 son valores propios ocultos y en el caso de los dos últimos tipos todos excepto uno se incluyen en un número muy reducido de *clusters*.

	Tamaño de la matriz																	
	32			64			128			256			512			1024		
Tipo	*cl*	*nvp*	*hi*	*cl*	*nvp*	*hi*	*cl*	*nvp*	*hi*	*cl*	*nvp*	*hi*	*cl*	*nvp*	*hi*	*cl*	*nvp*	*hi*
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	6	0	0	40	0	0	132	0	0	336	0	0	776
5	0	0	3	0	0	19	0	0	64	0	0	165	0	0	385	0	0	846
6	4	8	18	20	40	64	52	104	128	116	232	256	244	488	512	500	1000	1024
7	0	0	5	0	0	29	0	0	96	0	0	216	0	0	470	0	0	993
8	0	0	10	0	0	36	0	0	109	0	0	210	0	0	475	0	0	950
9	0	0	14	0	0	47	0	0	83	0	0	226	0	0	479	0	0	985
10	4	31	32	3	63	64	3	127	128	3	255	256	14	511	512	3	1023	1024
11	0	0	15	0	0	45	0	0	106	0	0	228	0	0	492	0	0	996
12	5	31	32	13	63	64	5	127	128	6	255	256	5	511	512	6	1023	1024

Tabla II. *Clusters* y valores propios ocultos en las matrices de prueba

Los resultados mostrados en la Tabla III también nos permiten agrupar los 12 tipos de matrices de prueba en distintas clases, pero en este caso en función del número de deflaciones ocurridas durante la fase de reconstrucción de los algoritmos de tipo divide y vencerás. Los algoritmos *dstdv* y *dspmg* detectan una cantidad de deflaciones bastante distinta con varios de los tipos de matrices probadas, lo que determina en gran medida los resultados experimentales a que dan lugar.

	Algoritmo			
	dstdv		*dspmg*	
Tipo	def1	def2	def1	def2
1	0	3588	0	0
2	0	3586	8	0
3	2	3084	0	0
4	130	512	128	0
5	203	512	202	0
6	4299	512	4105	773
7	3382	0	2635	217
8	911	0	893	2
9	1104	0	936	45
10	161	9029	263	8948
11	3766	0	3592	62
12	378	8769	628	8586

Tabla III. Número de deflaciones de tipo 1 y 2; $n = 1024$

Algoritmos de aproximación de raíces

En este apartado comparamos seis algoritmos de aproximación de raíces utilizándolos para el cálculo de los valores propios. Estos algoritmos pueden utilizarse durante la fase de extracción de los algoritmos de tipo bisección o durante la fase de reconstrucción de los algoritmos de tipo divide y vencerás. Comparamos los distintos algoritmos en el caso del método de bisección. Tres de los algoritmos (*peg, bis* y *nwt*) han sido utilizados previamente por otros autores, mientras los tres restantes (*lpe, lag* y *qlag*) se implementan por primera vez en este artículo. El análisis realizado se centra en estudiar la influencia de las características de los distintos tipos de matrices de prueba sobre el comportamiento de los distintos algoritmos de aproximación.

1. El algoritmo *bis* implementa el método de bisección pura para abordar tanto la fase de aislamiento como la de extracción de los valores propios.
2. El algoritmo *nwt* implementa el método de Newton tal y como se describe en la referencia[43].
3. El algoritmo *peg* divide la fase de extracción en dos subfases. Este algoritmo aplica varias etapas de bisección a partir de cada intervalo aislado conteniendo un valor propio hasta que obtiene un intervalo donde la función $q(x)$, definida como el último elemento de la recurrencia (5), es continua. A partir de este intervalo estrictamente aislado, el algoritmo aplica el método Pegasus para aproximar el valor propio.
4. El algoritmo *lpe* es similar al algoritmo *peg*, pero hace uso de la iteración de Laguerre para abordar la subfase de aislamiento estricto.
5. El algoritmo *lag* utiliza la iteración de Laguerre tal y como se ha descrito en el apartado anterior para llevar a cabo la extracción de cada valor propio.
6. El algoritmo *qlag* aplica la iteración quasi-Laguerre descrita en la referencia[22] para abordar la fase de extracción.

	Métodos de aproximación de raíces					
Tipo	*peg*	*lpe*	*nwt*	*bis*	*lag*	*qlag*
1	1,18	0,97	1,18	4,76	0,7	0,98
2	1,17	0,95	1,16	4,63	0,69	0,98
3	1,17	0,99	1,19	4,53	0,71	0,97
4	4,34	0,59	1,02	4,86	0,54	0,99
5	4,45	0,73	1,39	4,77	0,69	0,98
6	2,64	2,57	2,76	2,67	2,56	2,81
7	4,77	0,78	1,61	4,71	0,78	1,15
8	4,48	0,73	1,43	4,62	0,71	0,96
9	3,59	0,62	1,9	3,49	0,61	1,19
10	0,02	0,01	0,02	0,01	0,01	0,01
11	4,74	0,78	1,57	4,67	0,79	1,13
12	0,01	0,01	0,01	0,01	0,01	0,01

Tabla IV. Duración en segundos de los diferentes métodos de aproximaciones de raíces; $n = 1024$

La Tabla IV muestra la duración en segundos del algoritmo *dstrid* utilizando cada uno de los métodos de aproximación de raíces antes descritos para el cálculo de los distintos valores propios.

Los resultados en la Tabla IV demuestran que las prestaciones obtenidas con el algoritmo *dstrid* dependen del tipo de matriz. De hecho, el comportamiento de los distintos métodos

usados durante la fase de extracción depende en gran medida del número de *clusters* y de valores propios ocultos. Las distintas clases de matrices que hemos estudiado ofrecen comportamientos bien diferenciados dependiendo de la técnica de aproximación utilizada durante la fase de extracción. Por ejemplo, en el caso de las matrices de tipos 10 y 12, todos los valores propios se aproximan utilizando unas pocas etapas de bisección, dado que todos ellos excepto uno, se incluyen en unos pocos *clusters*.

Los resultados mostrados en la Tabla IV definen la iteración de Laguerre (*lag*) como el algoritmo de aproximación más rápido para los distintos tipos de matrices.

En las referencias[22,34] se demuestra que en el marco de los métodos de tipo divide y vencerás descritos, la aplicación de la iteración quasi-Laguerre mejora los resultados ofrecidos con la iteración de Laguerre. En la Tabla V podemos ver como el algoritmo *dsmsl* ofrece mejores resultados que el algoritmo *dspmg* en la mayoría de los casos. Sin embargo, en el marco del método de bisección, la iteración de Laguerre es siempre más rápida que la iteración quasi-Laguerre. Este hecho es debido a que los puntos iniciales utilizados para la aproximación son diferentes en ambos métodos. Mientras en los algoritmos de tipo divide y vencerás estos puntos son los valores propios de las matrices divididas, en los algoritmos de tipo bisección las iteraciones comienzan desde valores obtenidos mediante bisección pura.

Los malos resultados obtenidos con el algoritmo *bis* prueban que la aplicación de métodos alternativos durante la fase de extracción acelera en gran medida la convergencia hacia los valores propios del método de bisección pura.

En el caso de las matrices de Wilkinson (tipo 6) la aplicación de cualquiera de los métodos alternativos a la bisección no mejora los resultados del algoritmo *bis*. En este caso prácticamente todos los valores propios se emparejan en *clusters*, lo que provoca que los valores propios se acaben aproximando por bisección pura.

El uso de la iteración de Laguerre durante la subfase de aislamiento estricto (*lpe*) siempre da lugar a mejores resultados que la aplicación de la bisección (*peg*). La mejora obtenida es sustancial con matrices de tipos 4, 5, 7, 8, 9 y 11, donde el número de valores propios ocultos es muy grande. Ello es debido a que el método *peg* acaba realizando el aislamiento por bisección pura en presencia de este tipo de valores propios.

	Algoritmos						
Tipo	*dsterf*	*dstebz*	*dstrid*	*dstedc*	*dspmg*	*dsmsl*	*dstdv*
1	0,4	6,57	0,7	1,11	1,61	0,88	0,98
2	0,41	6.46	0,69	1,66	1,47	1,02	1,15
3	0,35	6,31	0,71	2,01	1,56	1,15	1,29
4	0,41	6,95	0,54	1,35	1,54	0,94	1
5	0,4	6,92	0,68	1,37	1,37	1,06	1,14
6	0,28	3,5	2,56	0,13	0,33	0,27	0,36
7	0,48	6,48	0,77	0,35	0,66	0,56	0,63
8	0,41	6,11	0,71	1,87	1,1	1,16	1,47
9	0,25	4,5	0,62	0,49	0,52	0,57	0,69
10	0,48	0,02	0,01	0,24	0,06	0,07	0,23
11	0,5	6,27	0,79	0,27	0,6	0,52	0,57
12	0,41	0,02	0,01	0,19	0,02	0,02	0,23

Tabla V. Duración en segundos de los algoritmos calculando todos los valores propios; $n = 1024$

Prestaciones experimentales

En la Tabla V se presentan los tiempos de ejecución en segundos para los siete algoritmos probados en el caso en que se calculan todos los valores propios de matrices de tamaño 1024. En dicha tabla podemos comprobar que los mejores resultados se obtienen en general con el algoritmo *dsterf*, mientras que los peores resultados son los del algoritmo de bisección pura (*dstebz*).

Tal y como hemos comentado anteriormente, los resultados de todos los algoritmos que utilizan métodos de tipo divide y vencerás dependen del tipo de matriz. Su comportamiento experimental se relaciona directamente con el número de deflaciones destacadas, el coste de su detección y el provecho obtenido en la reducción del tiempo de ejecución en este caso. Por ejemplo, los resultados de los algoritmos *dstdv* y *dspmg* pueden justificarse en gran medida a partir de las deflaciones mostradas en la Tabla III.

Por otro lado, cuál es el mejor algoritmo de tipo divide y vencerás también depende del tipo de matriz tratada. Mientras el algoritmo *dsmsl* es el mejor para matrices de tipos 1 a 5 y 12, *dstedc* es el mejor para las matrices de tipos 6, 7, 9 y 11, mientras el algoritmo *dspmg* es el que ofrece mejores resultados con las matrices de tipo 8 y 10.

El uso de la iteración de Laguerre es una muy buena alternativa para abordar la fase de reconstrucción de los algoritmos de tipo divide y vencerás. Cuando se combina con el uso de modificaciones de rango uno, *dstdv* ofrece mejores resultados que cuando se utilizan modificaciones de rango dos *dspmg*. Sin embargo, este comportamiento no es el mismo para los 12 tipos de matrices y varía sustancialmente con el número de deflaciones.

Cabe también citar que el uso de la iteración quasi-Laguerre mejora los resultados obtenidos con la iteración de Laguerre, tal y como se observa si se comparan los resultados de los algoritmos *dspmg* y *dsmsl*.

Finalmente, en lo que respecta a la rutina de tipo divide y vencerás incluida en el LAPACK, *dstedc*, ésta ofrece los mejores resultados para algunos tipos de matrices, pero los otros algoritmos de este tipo la sobrepasan en la mayoría de los casos. Más aún, el cálculo de todos los valores propios utilizando la rutina *dstedc* requiere un espacio de almacenamiento de $O(n^2)$, mientras los otros algoritmos de tipo divide y vencerás tienen un coste $O(n)$ en este caso[44].

Por otra parte, si comparamos los dos algoritmos que utilizan el método de bisección, podemos ver claramente como el uso de un método alternativo a la bisección para la aproximación de raíces durante la fase de extracción permite que el algoritmo *dstrid* supere claramente los resultados del algoritmo *dstebz*. De hecho, el algoritmo *dstrid* es entre 5 y 10 veces más rápido que la rutina del LAPACK con matrices cuyos tamaños varían entre 64 y 1024.

Asimismo, cabe destacar el comportamiento especial de los distintos algoritmos en el caso de las matrices de Wilkinson (tipo 6). En este caso el uso de una técnica de bisección (*dstrid* y *dstebz*) ofrece los peores resultados, dado que no es posible aislar casi ningún valor propio y éstos acaban siendo aproximados por bisección pura. En contraposición, los algoritmos de tipo divide y vencerás pueden explotar el gran número de deflaciones producidas en este caso para obtener muy buenos resultados.

En general, el algoritmo de bisección en dos fases, *dstrid*, ofrece mejores resultados que los de tipo divide y vencerás, excepto cuando estos últimos detectan un gran número de deflaciones.

Para finalizar, cabe decir que la iteración QR es el único método cuyas prestaciones no dependen del tipo de matriz, y en muchos casos es el que obtiene los mejores resultados. No obstante, los otros algoritmos son mejores si pueden explotar las características especiales de las matrices. En todo caso, los resultados del algoritmo *dstrid* casi siempre se hallan próximos a los del *dsterf*.

Precisión de los resultados

Con el fin de comparar la precisión de los resultados obtenidos con los siete algoritmos estudiados, comenzamos por utilizar las matrices de tipos 1 a 5. Con estos tipos de matrices es posible obtener los valores propios λ_i a partir de fórmulas bien conocidas.

Sea $\tilde{\lambda}$ el vector que contiene las aproximaciones calculadas con cada uno de los algoritmos a los valores propios exactos λ. La Tabla VI muestra el error relativo dado por

$$e_r = \frac{\|\lambda - \tilde{\lambda}\|_2}{\|\lambda\|_2} \tag{12}$$

Para facilitar el análisis de los resultados, éstos se han dividido por la precisión de la máquina $\varepsilon = 2,22\text{E-}16$.

La Tabla VI muestra la precisión de los resultados obtenidos por los distintos algoritmos con matrices de tamaño 1024. Sin embargo, hemos realizado el mismo análisis con distintos tamaños de matrices. Todos los resultados obtenidos demuestran que los algoritmos basados en el método de bisección (*dstrid, dstebz*) y aquellos que utilizan una estrategia de tipo divide y vencerás y el criterio de parada dado por (10), proporcionan una precisión similar independientemente del tamaño de las matrices. Por otro lado, los algoritmos *dsterf* y *dstedc* dan lugar a resultados cada vez menos precisos con el aumento del tamaño de la matriz. Este comportamiento había sido ya detectado en las referencias[22,34].

Para calcular la precisión de los resultados con los doce tipos de matrices de prueba hemos utilizado otros dos parámetros tal y como se hace en la referencia[22]. Concretamente, si llamamos $\tilde{Q}$ la matriz que contiene los vectores propios calculados de $\mathbf{T}$ y si llamamos $\tilde{D}$ la matriz diagonal conteniendo los valores propios calculados, podemos obtener el residuo y la separación de la ortogonalidad a partir de

$$\text{residuo} = \max_i \frac{\|(T\tilde{Q} - \tilde{Q}\tilde{D})e_i\|_2}{|\tilde{\lambda}|_{\max}} \tag{13}$$

$$\text{separación de la ortogonalidad} = \max_i \|(\tilde{Q}^T\tilde{Q} - I)e_i\|_2 \tag{14}$$

Hemos utilizado una rutina basada en la iteración inversa del LAPACK (*dstein*) para calcular los vectores propios de las diferentes matrices, excepto en el caso del algoritmo *dstedc*, que proporciona tanto los valores como los vectores propios. Por lo tanto, los resultados de precisión incluyen no sólo los errores producidos durante el cálculo de los valores propios, sino también los acumulados en el proceso de cálculo de los vectores propios.

Los resultados obtenidos con los parámetros (13) y (14) nos llevan a conclusiones similares a las obtenidas en la referencia[22] y definen la misma relación entre los algoritmos que la ofrecida por el parámetro (12). Esta afirmación es cierta excepto para las matrices de tipos 10 y 12, donde la integración de prácticamente todos sus valores propios en *clusters* nos lleva a un crecimiento sustancial en el error relacionado con la ortogonalidad.

Algunos estudios teóricos de la iteración QR[1,2] muestran que, al menos en el caso no simétrico, los valores propios sufren problemas de precisión debidos al criterio de parada. Nuestro análisis experimental confirma este comportamiento en el caso simétrico.

La Tabla VI demuestra que los cuatro algoritmos (*dstrid, dspmg, dsmsl, dstdv*) que utilizan el criterio de parada dado por (10) logran resultados muy similares y ofrecen los valores propios más precisos. Por otra parte, los resultados menos precisos son los obtenidos por los algoritmos *dstedc* y *dsterf*.

La Tabla VII muestra los valores del parámetro (2) cuando modificamos el valor de ε en el criterio de parada (10). Aplicamos esta modificación a los algoritmos *dstrid* y *dstdv* manteniendo constantes los parámetros en el algoritmo *dsterf*. Podemos ver como

	Tipo de matriz				
Algoritmo	1	2	3	4	5
dsterf	1,500	1,331	2,059	10,200	2,085
dstebz	0,661	0,527	0,771	0,655	0,667
dstrid	0,476	0,291	0,497	0,003	0,050
dstedc	5,668	5,822	1,964	1,356	0,779
dspmg	0,476	0,290	0,496	0,004	0,052
dsmsl	0,475	0,289	0,499	0,326	0,273
dstdv	0,479	0,290	0,499	0,066	0,135

Tabla VI. Precisión de los resultados; $n = 1024$

		Epsilon en (10)								
T.	Alg.	2,2E-16	1,0E-15	1,0E-14	1,0E-13	1,0E-12	1,0E-11	1,0E-10	1,0E-9	1,0E-9
1	*dsterf*	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500	1,500
	dstrid	0,476	0,475	0,475	0,476	0,475	0,476	0,475	0,475	8,665
	dstdv	0,479	0,479	0,478	0,479	0,480	0,480	0,479	0,476	0,969
2	*dsterf*	1,331	1,331	1,331	1,331	1,331	1,331	1,331	1,331	1,331
	dstrid	0,291	0,290	0,290	0,290	0,290	0,290	0,290	0,289	3,865
	dstdv	0,290	0,290	0,289	0,289	0,289	0,289	0,289	0,290	1,086
3	*dsterf*	2,059	2,059	2,059	2,059	2,059	2,059	2,059	2,059	2,059
	dstrid	0,497	0,497	0,495	0,496	0,538	0,495	0,494	0,492	11,098
	dstdv	0,499	0,499	0,499	0,498	0,498	0,498	0,498	0,498	3940472,778
4	*dsterf*	10,201	10,201	10,201	10,201	10,201	10,201	10,201	10,201	10,201
	dstrid	0,003	0,003	0,003	0,003	0,003	0,010	0,004	0,004	0,004
	dstdv	0,066	0,279	2,124	23,013	353,408	4695,901	20797,565	531858,465	4528980,693
5	*dsterf*	2,085	2,085	2,085	2,085	2,085	2,085	2,085	2,085	2,085
	dstrid	0,050	0,050	0,051	0,051	0,052	0,052	0,054	0,055	6,100
	dstdv	0,135	0,135	4,991	10,092	277,928	3859,087	27037,266	441216,961	6872010,105

Tabla VII. Evolución de la precisión con el valor de ε en (10); $n = 1024$

esta modificación afecta ligeramente a la precisión de los valores propios obtenidos con el algoritmo *dstrid*, mientras el algoritmo *dstdv* tan sólo se ve afectado por un gran incremento de ε para las matrices de tipos 4 y 5. Este comportamiento de los algoritmos es debido a la convergencia cúbica de la iteración de Laguerre. También hemos probado la precisión del algoritmo *dstrid* utilizando métodos alternativos a este algoritmo de aproximación. El efecto de modificar el valor de ε en la precisión de los valores propios depende del algoritmo de aproximación utilizado. Los algoritmos con menor velocidad de convergencia, como el de bisección pura y, en algunos casos, el método Pegasus, dan lugar a una pérdida de precisión más rápida cuando incrementamos el valor de ε. Por otro lado, los algoritmos con una convergencia más rápida, como la iteración de Newton o la de Laguerre tan sólo se ven afectados por grandes incrementos de ε.

Al modificar el valor de ε en el criterio de parada, además de afectar a la precisión de los resultados obtenidos, reducimos el número de iteraciones necesarias para alcanzar la convergencia, y por tanto reducimos el tiempo de ejecución de los algoritmos. La Figura 1 muestra la evolución del tiempo de ejecución de los algoritmos *dstrid* y *dstdv* con respecto al algoritmo *dsterf*.

Finalmente, la Tabla VIII muestra la duración de los algoritmos *dsterf, dstrid* y *dstdv* con un valor de ε igual a 1,0E-08 en (10). Este es el valor para el cual la precisión de los resultados calculados por el algoritmo *dsterf* mejora los resultados dados por los otros dos algoritmos (Tabla VII). En todos los casos el tiempo de ejecución de los algoritmos *dstrid*

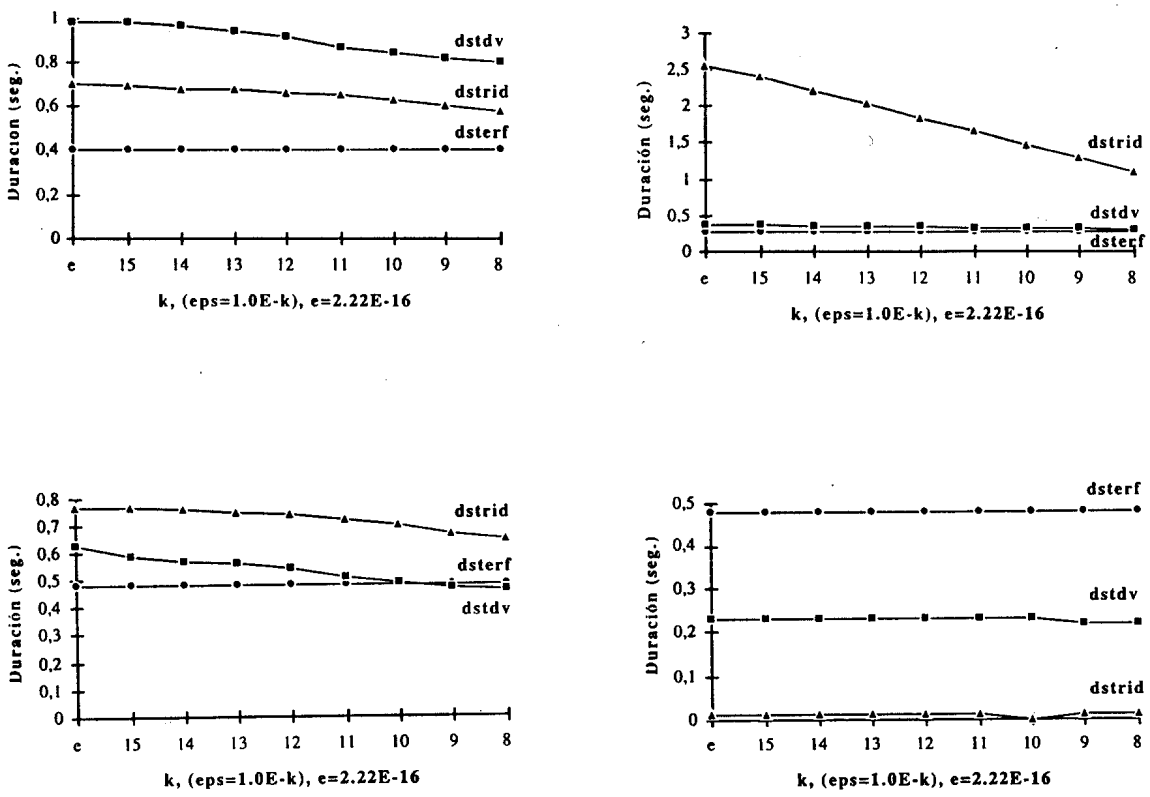

Figura 1. Evolución de la duración de los algoritmos modificando el criterio de parada

y *dstdv* se acerca mucho más al del algoritmo *dsterf* que en los casos contemplados en la Tabla V, cuando el valor de ε es 2,22E-16.

	Tipo de matriz											
Alg	1	2	3	4	5	6	7	8	9	10	11	12
dsterf	0,4	0,41	0,35	0,41	0,4	0,28	0,48	0,41	0,25	0,48	0,5	0,41
dstrid	0,58	0,56	0,58	0,4	0,56	1,09	0,65	0,59	0,32	0,01	0,66	0
dstdv	0,8	0,93	1,04	0,87	0,94	0,3	0,46	1,22	0,43	0,22	0,47	0,23

Tabla VIII. Duración de los algoritmos con $\varepsilon = 1,0E$-08 en (10)

CONCLUSIONES

En el presente artículo proponemos dos nuevos algoritmos basados en dos de los métodos más conocidos para el cálculo de los valores propios de matrices tridiagonales simétricas: los métodos de bisección y divide y vencerás.

En el caso del método de bisección mostramos como una correcta elección de la técnica de aproximación de raíces utilizada para el cálculo de los valores propios determina la velocidad del algoritmo en todos los casos. En el marco del método de bisección utilizamos por primera vez la iteración de Laguerre para llevar a cabo esta tarea, lo que da lugar a mejores resultados que los obtenidos con otros métodos ampliamente utilizados como la iteración de Newton o el método Pegasus. Más aún, una combinación adecuada de la iteración de Laguerre con el método Pegasus también da lugar a muy buenos resultados a la hora de abordar la extracción de los valores propios.

En el análisis experimental realizado nos hemos centrado en la influencia de las características de las matrices en las prestaciones de los algoritmos. En el caso del método de bisección podemos ver como el número de *clusters* y de valores propios ocultos determina el comportamiento de las distintas técnicas de extracción y con ello también el del algoritmo en su conjunto.

En lo que respecta a los algoritmos basados en la estrategia divide y vencerás hemos implementado un algoritmo que utiliza modificaciones de rango uno durante la fase de división y la iteración de Laguerre durante la fase de reconstrucción (*updating*). Comparamos este algoritmo con otros que utilizan diferentes técnicas en ambas fases. El comportamiento relativo de los algoritmos depende del número de deflaciones que detectan con cada tipo de matriz.

La comparación de todos los algoritmos implementados demuestra que para muchos tipos de matrices el algoritmo de bisección en dos fases ofrece mejores resultados que los algoritmos de tipo divide y vencerás. Sin embargo, en el caso de las matrices de Wilkinson, este último tipo de algoritmos mejora los resultados del método de bisección debido a la particular distribución de los valores propios en este tipo de matrices.

Por otro lado, mientras el uso de la iteración quasi-Laguerre durante la fase de reconstrucción de los algoritmos de tipo divide y vencerás mejora los resultados ofrecidos por la iteración de Laguerre cuando realizamos esta comparación para abordar la fase de extracción del método de bisección, siempre obtenemos peores resultados usando la primera de ellas.

Los resultados presentados muestran que los algoritmos de bisección y divide y vencerás pueden mejorar las prestaciones del iterativo QR incluido en el LAPACK cuando las características del espectro pueden ser explotadas de modo ventajoso. Más aún, los valores propios obtenidos con el algoritmo *dsterf* son bastante menos precisos que los proporcionados por los algoritmos *dstrid* y *dstdv*.

En el caso de muchos tipos de matrices, cuando utilizamos un método de aproximación de raíces con una convergencia cúbica como la iteración de Laguerre, un gran incremento de las tolerancias incluidas en el criterio de parada no afecta sustancialmente a la precisión de los valores propios calculados. Cuando realizamos esta modificación con los algoritmos *dstrid* y *dstdv* para obtener una precisión similar al algoritmo *dsterf*, los tiempos de ejecución de los primeros se acercan a los de este e incluso lo mejoran en algún caso.

Por último, no debemos olvidar otra importante ventaja de los métodos de bisección y divide y vencerás. Mientras la iteración QR no da lugar a implementaciones escalables en paralelo, tanta la bisección como la estrategia divide y vencerás poseen un gran paralelismo potencial[5]. Más aún, el método de bisección permite calcular cualquier parte del espectro, mientras la iteración QR nos obliga a calcular siempre todos los valores propios.

AGRADECIMIENTOS

Queremos expresar nuestra gratitud al profesor Zhonggang Zeng por toda la información que nos ha proporcionado sobre los algoritmos *dspmg* y *dsmsl*. También queremos agradecer a los revisores del artículo sus valiosos consejos.

REFERENCIAS

1 M. Ahues, A. Largillier y F. Tisseur, "Stability and stopping criterion for the QR algorithm", *XIII Householder Symposium*, (1996).

2 M. Ahues y F. Tisseur, "A new deflation criterion for the QR algorithm", *LAPACK Working Note*, **122**, (1997).

3 E. Anderson, Z. Bay y C. Bischof *et al.*, "*LAPACK user's guide*", (1992).

4 L. Argonne Nat, "*EISPACK. Eigensystem package*", (1972).

5 J.M. Badía, "Algoritmos paralelos para el cálculo de los valores propios de matrices estructuradas", Tesis doctoral, Universidad Politécnica de Valencia, (1996).

6 J.M. Badía y A.M. Vidal, "Parallel computation of the eigenstructure of Toeplitz-plus-Hankel matrices on multicomputers", *Lecture Notes in Computer Science*, J. Dongarra y J. Wasniewski (Eds.), Vol. **879**, pp. 33–40, Springer-Verlag, (1994).

7 J.M. Badía y A.M. Vidal, "Efficient solution of the eigenproblem of banded symmetric Toeplitz matrices on multicomputers", *III Euromicro Workshop on Parallel and Distributed Processing*, pp. 416–423, IEEE Computer Society Press, San Remo, (1995).

8 J.M. Badía y A.M. Vidal, "Exploiting the parallel divide-and-conquer method to solve the symmetric tridiagonal eigenproblem", *VI Euromicro Workshop on Parallel and Distributed Processing*, pp. 13–19, IEEE Madrid, (1998).

9 J.M. Badía y A.M. Vidal, "The symmetric tridiagonal eigenproblem on massively parallel computers", *High Performance Algorithms for Structured Matrix Problems*, P. Arbenz, M. Paprzycki y A. Sameh (Eds.), NOVA Science Publishers, pp. 91–112, (1998).

10 G. Baker, "Accelerated bisection techniques for tri- and quintadiagonal matrices", *Int. J. Num. Meth. Engng.*, Vol. **35**, pp. 203–218, (1992).

11 W. Barth, R.S. Martin y J.H. Wilkinson, "Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method", *Numerische Mathematik*, Vol. **9**, pp. 386–393, (1967).

12 A. Baserman y P. Weidner, "A parallel algorithm for determining all eigenvalues of large real symmetric tridiagonal matrices", *Parallel Computing*, Vol. **18**, pp. 1129–1141, (1992).

13 H.J. Bernstein, "An accelerated bisection method for the calculation of eigenvalues of a symmetric tridiagonal matrix", *Numerische Mathematik*, Vol. **43**, pp. 153–160, (1984).

14 D. Bini y L. Gemignani, "An iteration scheme for the divide-and-conquer eigenvalue solvers", Manuscrito, Dipartimento di Matematica, Università di Pisa, (1991).

15 D. Bini y V. Pan, "Practical improvement of the divide-and-conquer eigenvalue algorithms", *Computing*, Vol. **48**, pp. 109–123, (1992).

16 J.R. Bunch, C.P. Nielsen y D.C. Sorensen, "Rank-one modification of the symmetric eigenproblem", *Numerische Mathematik*, Vol. **31**, pp. 31–48, (1978).

17 J.J.M. Cuppen, "A divide-and-conquer method for the symmetric tridiagonal eigenproblem", *SIAM J. Sci. Stat. Comp.*, Vol. **8**, N° 2, pp. 139–154, (1981).

18 U. De Ros, "Soluzione parallela su una rete di transputer del problema degli autovalori per una matrice tridiagonale simmetrica", Tesis doctoral, Politecnico di Milano, Facoltà di Ingegneria, Dipartimento di Matematica, (1993).

19 J. Demmel, J. Du Croz, S. Hammarling *et al.*, "Guidelines fot the design of symmetric eigenroutines, SVD and iterative refinement for linear systems", MCS-TM-111, *LAPACK Working Note*, N° 4, Argonne National Laboratory, (1988).

20 J. Demmel y A. McKenney, " A test matrix generation suite", *LAPACK Working Note*, Vol. **9**, Courant Institute, New York, (1992).

21 M. Dowell y P. Jaratt, "The Pegasus method for computing the root of an equation", *BIT*, Vol. **12**, pp. 503–508, (1972).

22 Q. Du, J. Ming, T.Y. Li *et al.*, "Quasi Laguerre iteration in solving symmetric tridiagonal eigenvalue problems", *SIAM J. Sci. Stat. Comp.*, Vol. **17**, N° 6, pp. 1347–1368, (1996).

23 J.W. Givens, "A method of computing eigenvalues and eigenvectors suggested by classical results on symmetric matrices", *U.S. Nat. Bur. Standards Applied Mathematics Series*, Vol. **29**, pp. 117–122, (1953).

24 J.W. Givens, "Numerical computations of the characteristic values of a real symmetric matrix", *ORNI-1574*, Oak Ridge national Laboratory, (1954).

25 G.H. Golub, "Some modified matrix eigenvalue problems", *SIAM Review*, Vol. **15**, pp. 318–334, (1973).

26 G.H. Golub y C.F. Van Loan, "*Matrix computation*", John Hopkins University Press, (1989).

27 R.T. Gregory y D.L. Karney, "*A collection of matrices fot testing computational algorithms*", Robert E. Krieger Publishing Company, Huntington, N.Y., (1978).

28 W. Kahan, "Accurate eigenvalues of a symmetric tridiagonal matrix," *Technical Report CS41*, Computer Science Dpt., Stanford University, (1966).

29 W. Kahan y J. Varah, "Two working algorithms for the eigenvalues of a symmetric tridiagonal matrix", *Technical Report CS43*, Computer Science Dpt., Stanford University, (1966).

30 K. Li y T.Y. Li, "An algorithm for symmetric tridiagonal eigenproblems divide-and-conquer with homotopy continuation", *SIAM J. Sci. Stat. Comp.*, Vol. **14**, pp. 735–751, (1993).

31 R.C. Li y H. Ren, "An efficient tridiagonal eigenvalue solver on the CM-5 with Laguerre's iteration", Preprint, C.S. Department, Berkeley, University of California, (1993).

32 T.Y. Li y N.H. Rhee, "Homotopy algorithm for symmetric eigenvalue problems", *Numerische Mathematik*, Vol. **55**, pp. 265–280, (1989).

33 T.Y. Li y Z. Zeng, "Homotopy-determinant algorithm solving nonsymmetric eigenvalue problems", *Math. Comp.*, Vol. **59**, pp. 483–502, (1992).

34 T.Y. Li y Z. Zeng, "The Laguerre iteration in solving the symmetric tridiagonal eigenproblem, revisited", *SIAM J. Sci. Stat. Comp.*, Vol. **13**, N° 5, pp. 1145–1173, (1993).

35 T.Y. Li, Z. Zeng, y L. Cong, "Solving eigenvalue problems of real nonsymmetric matrices with real homotopies", *SIAM J. Numer. Anal.*, Vol. **29**, pp. 229–248, (1992).

36 T.Y. Li, H. Zhang y X.H. Sun, "Parallel homotopy algorithm for symmetric tridiagonal eigenvalue problems", *SIAM J. Sci. Stat. Comp.*, Vol. **12**, pp. 464–485, (1991).

37 S.S. Lo, B. Philippe y A. Sameh, "A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem", *SIAM J. Sci. Stat. Comp.*, Vol. **8**, N° 2, pp. 155–165, (1987).

38 M. Oettli, "*The homotopy method applied to the symmetric eigenproblem*", Swiss Federale Institute of Technology, Zurich, (1995).

39 B.N. Parlett, "*The symmetric eigenvalue problem*", Prentice-Hall, Inc., (1980).

40 R. Pavani y U. De Ros, "A distributed divide-and-conquer approach to the parallel symmetric eigenvalue problem", *The International Conference on High-Performance Computing and Networking*, Milano, (1995).

41 R. Pavani y U. De Ros, "A parallel alorithm for the symmetric eigenvalue problem", *International Congress on Industrial and Applied Mathematics*, Hamburg, (1995).

42 V. Pereyra y G. Scherer, "Eigenvalues of symmetric tridiagonal matrices: a fast, accurate and reliable algorithm", *J. Inst. Math. Appl.*, Vol. **12**, pp. 209–222, (1973).

43 R.M.S. Ralha, "Parallel solution of the symmetric tridiagonal eigenvalue problem on a transputer network", *SEMNI 93*, pp. 1026–1033, La Coruña, España, (1993).

44 J. Rutter, "A serial implementation of Cuppen's divide-and-conquer algorithm for the symmetric eigenvalue problem", *UCB/CSD 94/799*, Computer Science Division, University of California, (1994).

45 W.F. Trench, "Numerical solution of the eigenvalue problem for hermitian Toeplitz matrices", *SIAM J. Matrix Anal. Appl.*, Vol. **10**, N° 2, pp. 135–146, (1989).

46 W.F. Trench, "Numerical solution of the eigenvalue problem for efficiently structured hermitian matrices", *Lin. Alg. Appl.*, Vol. **154–156**, pp. 415–432, (1991).

47 J.H. Wilkinson, "Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection", *Numerische Mathematik*, Vol. **4** , pp. 362–367, (1962).

48 J.H. Wilkinson, "*The algebraic eigenvalue problem*", Oxford University Press, Oxford, (1965).