Capítulo 1 Planteamiento del problema

1.1 Preliminares

La solución del sistema lineal

$$A\boldsymbol{x} = \boldsymbol{d}, \tag{1.1}$$

donde A es una matriz tridiagonal y $\boldsymbol{d}$ es el vector de términos independiente, dados por

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix} \quad \text{y} \quad \boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, \tag{1.2}$$

puede obtenerse utilizando el clásico método de eliminación de Gauss. Este método, que es puramente secuencial, consiste en anular los elementos que se encuentran por debajo de la diagonal principal utilizando las operaciones elementales adecuadas sobre la matriz y

el vector de términos independientes. Después obtenemos la última componente de la solución y finalmente, siguiendo un proceso de sustitución regresiva, calculamos el resto de las componentes de la solución.

El siguiente algoritmo recoge los pasos necesarios para resolver el sistema (1.1) mediante el método de eliminación de Gauss.

Algoritmo 1.1 Método de eliminación de Gauss para sistemas tridiagonales.

1 Para $i = 2, 3, \ldots, n$, calcular

1.1 $h = \dfrac{c_i}{a_i}$,

1.2 $a_i = a_i - hb_i$,

1.3 $d_i = d_i - hd_i$.

2 $x_n = \dfrac{d_n}{a_n}$.

3 Para $i = n-1, n-2, \ldots, 1$, calcular $x_i = \dfrac{d_i - x_{i+1}b_i}{a_i}$

Es fácil deducir del algoritmo 1.1 que el número de operaciones que se realizan es de $3(n-1)$ restas, $3(n-1)$ productos y $2n-1$ divisiones. Antes de hablar del coste computacional de este algoritmo debemos establecer la unidad que tomamos como base para medir el tiempo de ejecución del mismo. De aquí en adelante y a lo largo de esta memoria tomaremos como unidad de tiempo un *flop*, que representa el tiempo que supone la realización de una operación en coma flotante. Así, resolver un sistema tridiagonal utilizando el método de eliminación de Gauss por medio del algoritmo 1.1 supone un coste computacional de $8n - 7$ flops.

Definición 1.1 Sea A la matriz tridiagonal de la expresión (1.2). Llamamos **diagonal dominanza** de A a la expresión

$$\delta = \min_{1 \leq i \leq n} \frac{a_i}{c_i + b_i},$$

donde, por comodidad, hemos supuesto que $c_1 = b_n = 0$. Diremos que la matriz A es **diagonal dominante** si $\delta \geq 1$ y **estrictamente diagonal dominante** si $\delta > 1$.

La eliminación de Gauss es un método que resulta estable para sistemas cuya matriz de coeficientes es diagonal dominante o estrictamente diagonal dominante. De aquí en adelante y a lo largo de esta memoria consideraremos sistemas tridiagonales cuya matriz de coeficientes es diagonal dominante con el fin de asegurarnos su estabilidad desde el punto de vista numérico.

En adelante, denotaremos por I_n a la matriz identidad de tamaño $n \times n$ y por $\boldsymbol{e}_k$, para $k = 1, 2, \ldots, n$, la columna k-ésima de I_n.

En los distintos métodos del tipo *divide y vencerás* que se analizan a lo largo de capítulos posteriores es necesario resolver varios sistemas tridiagonales con la misma matriz de coeficientes y distintos vectores de términos independientes. En estos casos es recomendable utilizar un método de resolución basado en la descomposición (también llamada factorización) LU de la matriz de coeficientes. Dicha descomposición para matrices tridiagonales consiste en escribir la matriz A como producto de dos matrices: L, bidiagonal inferior, y U, bidiagonal superior, de manera que

$$A = LU = \begin{bmatrix} 1 & & & & & \\ l_2 & 1 & & & & \\ & l_3 & 1 & & & \\ & & l_4 & 1 & & \\ & & & \ddots & \ddots & \\ & & & & l_n & 1 \end{bmatrix} \begin{bmatrix} u_1 & b_1 & & & & \\ & u_2 & b_2 & & & \\ & & u_3 & b_3 & & \\ & & & u_4 & \ddots & \\ & & & & \ddots & b_{n-1} \\ & & & & & u_n \end{bmatrix}.$$

El siguiente algoritmo calcula la descomposición LU para la matriz tridiagonal A de la expresión (1.2).

Algoritmo 1.2 Descomposición LU para la matriz tridiagonal A de la expresión (1.2).

1 $u_1 = a_1$.

2 Para $i = 2, 3, \ldots, n$, calcular

2.1 $l_i = \dfrac{c_i}{u_{i-1}}$,

2.2 $u_i = a_i - b_{i-1} l_i$.

De acuerdo con el algoritmo 1.2, el cálculo de la descomposición LU de una matriz tridiagonal de tamaño $n \times n$ requiere un total de $n-1$ restas, $n-1$ productos y $n-1$ divisiones, lo que representa un coste computacionel de $3n-3$ flops.

A partir de la descomposición LU de una matriz tridiagonal A podemos resolver el sistema (1.1) de forma sencilla. Para ello, sustituimos A por LU en (1.1), con lo que nos queda el sistema $LU\boldsymbol{x} = \boldsymbol{d}$. Ahora, llamando $U\boldsymbol{x} = \boldsymbol{y}$, resolvemos inicialmente el sistema $L\boldsymbol{y} = \boldsymbol{d}$ y utilizamos su solución $\boldsymbol{y}$ para obtener la solución general $\boldsymbol{x}$ resolviendo $U\boldsymbol{x} = \boldsymbol{y}$. El siguiente algoritmo resume las operaciones necesarias para resolver el sistema (1.1) una vez calculada la descomposición LU de la matriz A.

Algoritmo 1.3 Resolución del sistema (1.1) utilizando la descomposición LU de la matriz A de la expresión (1.2).

1 Calcular la factorización LU de A mediante el algoritmo 1.2.

2 $y_1 = d_1$.

3 Para $i = 2, 3, \ldots, n$, calcular $y_i = d_i - y_{i-1} l_i$.

4 $x_n = \dfrac{y_n}{u_n}$.

5 Para $i = n-1, n-2, \ldots, 1$, calcular $x_i = \dfrac{y_i - x_{i+1} b_i}{u_i}$.

El número de operaciones necesarias para resolver el sistema (1.1), de acuerdo con el algoritmo 1.3, es de $5n-4$, ya que se realizan $2n-2$ restas, $2n-2$ productos y n divisiones, por lo que el coste computacional será de $5n-4$ flops.

En el desarrollo de algunos métodos del tipo *divide y vencerás* para sistemas tridiagonales debemos resolver sistemas particulares donde el vector de términos independientes es la primera o última columna de la matriz I_n. Es decir, tendremos que resolver sistemas de la forma $A\boldsymbol{x} = \boldsymbol{e}_1$ y $A\boldsymbol{x} = \boldsymbol{e}_n$. Supongamos que la descomposición LU de la matriz A es conocida. Veamos las modificaciones que deben introducirse respecto de los algoritmos anteriores para resolver estos sistemas particulares.

Algoritmo 1.4 Resolución del sistema $A\boldsymbol{x} = \boldsymbol{e}_1$ utilizando la descomposición LU de A.

1 Calcular la factorización LU de A mediante el algoritmo 1.2.

2 $y_1 = 1$.

3 Para $i = 2, 3, \ldots, n$, calcular $y_i = -y_{i-1} l_i$.

4 $x_n = \dfrac{y_n}{u_n}$.

5 Para $i = n-1, n-2, \ldots, 1$, calcular $x_i = \dfrac{y_i - x_{i+1} b_i}{u_i}$.

La diferencia respecto al algoritmo 1.3 está en el proceso de sustitución progresiva en el que se calculan las componentes de la solución parcial $\boldsymbol{y}$ realizando únicamente $n-1$ operaciones. En consecuencia, el coste computacional de este algoritmo es de $4n-3$ flops.

Algoritmo 1.5 Resolución del sistema $A\boldsymbol{x} = \boldsymbol{e}_n$ utilizando la descomposición LU.

1 Calcular la factorización LU de A mediante el algoritmo 1.2.

2 $x_n = 1$.

3 Para $i = n-1, n-2, \ldots, 1$, calcular $x_i = -\dfrac{x_{i+1}b_i}{u_i}$

Nótese que al resolver un sistema de la forma $A\boldsymbol{x} = \boldsymbol{e}_n$, no es necesario efectuar ninguna operación para la resolución del sistema auxiliar $L\boldsymbol{y} = \boldsymbol{e}_n$, por lo que las únicas operaciones que se realizan son para el cálculo de la solución final mediante un proceso de sustitución regresiva. Así, el coste computacional del algoritmo 1.5 es de $2n-1$ flops.

Finalizamos esta sección con la introducción de las fórmulas de Sherman-Morrison y Sherman-Morrison-Woodbury que nos permiten resolver un sistema de ecuaciones lineales cualquiera. Supongamos que podemos escribir la matriz A de la forma $A = G + \boldsymbol{u}\boldsymbol{v}^T$ donde $\boldsymbol{u}, \boldsymbol{v}$ son vectores de $\mathbb{R}^n$. Supongamos que es fácil calcular G^{-1}. La fórmula de Sherman-Morrison (véase por ejemplo Golub y Van Loan [52, página 51]), nos proporciona un método para calcular la inversa de A como

$$A^{-1} = G^{-1} - \frac{1}{1 + \boldsymbol{v}^T G^{-1}\boldsymbol{u}}\, G^{-1}\boldsymbol{u}\boldsymbol{v}^T G^{-1}. \tag{1.3}$$

Podemos resolver el sistema (1.1) utilizando la expresión (1.3) como

$$\boldsymbol{x} = A^{-1}\boldsymbol{d} = \left(G^{-1} - \frac{1}{1 + \boldsymbol{v}^T G^{-1}\boldsymbol{u}} G^{-1}\boldsymbol{u}\boldsymbol{v}^T G^{-1}\right)\boldsymbol{d} = \left(I - \frac{1}{1 + \boldsymbol{v}^T G^{-1}\boldsymbol{u}} G^{-1}\boldsymbol{u}\boldsymbol{v}^T\right) G^{-1}\boldsymbol{d}. \tag{1.4}$$

Si la matriz A puede escribirse de la forma $A = G + UV^T$ donde U y V son matrices de $\mathbb{R}^{n\times k}$ y conocemos de nuevo la inversa de G, entonces la fórmula de Sherman-Morrison-Woodbury nos prooporciona una expresión para el cálculo de la inversa de A, de la forma

$$A^{-1} = \left(A + UV^T\right)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}. \tag{1.5}$$

1.2 Algoritmos del tipo divide y vencerás

El conocido proverbio *divide y vencerás*, que en tantos ámbitos de la vida cotidiana se escucha, constituye en la actualidad una estrategia fundamental en la resolución de problemas. Debido al gran número de campos en los que se aplica podemos afirmar que es una estrategia básica en el diseño de ciertos algoritmos en computación. La idea básica en que se fundamenta esta estrategia es la de dividir un problema de gran tamaño que no podemos resolver de forma directa en diversos subproblemas más pequeñas que sí pueden resolverse. Posteriormente, se combinan las soluciones de estos subproblemas más pequeños para llegar a la solución del problema original. Podemos ver los textos de Aho, Hopcroft y Ullman [2, 3] para una descripción más detallada de los algoritmos de este tipo.

Así pues, el arquetipo *divide y vencerás* es un modelo de resolución de ciertos problemas que se basa en una estrategia muy simple. Esta estrategia la podemos resumir diciendo que cuando nos enfrentamos a un problema de gran tamaño, efectuamos los siguientes pasos:

(i) Tomamos el problema original y lo dividimos en problemas más pequeños del mismo tipo. Seguimos dividiendo estos problemas en problemas más pequeños hasta que llegamos a problemas de un tamaño tal que pueden resolverse de forma sencilla. Estos problemas los llamamos **problemas base**.

(ii) Resolvemos los problemas base.

(iii) Tomamos la solución de los problemas base y las combinamos para obtener la solución de problemas más grandes hasta que llegamos a la solución del problema original.

Como vemos, los algoritmos del tipo *divide y vencerás* son de tipo recursivo y presentan dos partes claramente diferenciadas. En la primera es donde se rompe el problema original en subproblemas más pequeños y se conoce con el nombre de ***split***; en la segunda, se unen los resultados parciales para obtener la solución general y se conoce con el nombre de ***merge***.

Al desarrollar un algoritmo del tipo *divide y vencerás*, debemos definir claramente los siguientes elementos:

Tamaño del problema. Debemos definir una función que nos proporcione el tamaño del problema original.

Problema base. Debemos identificar el tamaño del problema base puesto que es el problema más grande para el que tenemos una solución.

Solución del problema base. Debemos definir un procedimiento que tenga como entrada el problema base y como salida su solución, imponiendo la condición de que el tamaño de la entrada coincida con el tamaño del problema base.

Split. Es necesario establecer un procedimiento que divida el problema general en subproblemas. Así, si el problema original es $\mathcal{P}$, se debe crear un procedimiento cuya entrada sea $\mathcal{P}$ y cuya salida sean $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$, un conjunto de problemas cuyo tamaño debe ser menor que el tamaño de $\mathcal{P}$.

Merge. Es necesario establecer un procedimiento que permita unir las soluciones de los subproblemas para obtener la solución del problema original. Así, es necesario definir una función cuya entrada sea $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$, las soluciones de los problemas $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$, respectivamente y cuya salida sea $\mathcal{S}$, la solución del problema $\mathcal{P}$.

Esta idea básica, que en principio es puramente secuencial, puede fácilmente paralelizarse si los subproblemas, obtenidos por división del problema original, son independientes. De este modo, el algoritmo es fácilmente paralelizable únicamente para el caso en el que los subproblemas se resuelven de forma independiente unos de otros. Los pasos para desarrollar algoritmos paralelos del tipo *divide y vencerás* son esencialmente los mismos que para algoritmos secuenciales. Las esacasas diferencias se resumen en los siguientes puntos

- Para la resolución de los problemas base es muy probable que resulte mucho más eficiente una implementación secuencial que una implementación paralela. Esto es debido a que el tamaño de los problemas base suele ser muy pequeño y por tanto el tiempo del envío y recepción de datos entre los procesadores puede sobrepasar el tiempo de cálculo secuencial.

- Para poder resolver el problema en paralelo, los subproblemas que nos proporciona el procedimiento *split*, deben ser independientes entre sí.

También cabe reseñar que en los últimos años se han desarrollado nuevos lenguajes de programación diseñados específicamente para la paralelización automática de algoritmos del tipo *divide y vencerás*. Por ejemplo, podemos citar el proyecto de creación del lenguaje APERITIF (Automatic Parallelization of Divide and Conquer Algorithms), cuyo autor es Erlebach [43] de la Universidad Técnica de Munich, Alemania. El compilador de dicho lenguaje, llamado APRIL, traduce los programas de este lenguaje a programas en C, que pueden ejecutarse en computadores paralelos. Además, contiene ciertas llamadas a funciones específicas de paso de mensajes. El objetivo que persigue este proyecto es la creación de un lenguaje fácil de utilizar que nos permita una explotación eficiente de la potencia dispponible hoy en día a través de los computadores paralelos. El compilador y documentación relativa al mismo se puede obtener en [43]. Este programa ha sido desarrollado y probado en redes de workstations del tipo HP 9000/720 utilizando PVM.

De entre la enorme variedad de campos y problemas de la computación donde se aplica la estrategia *divide y vencerás* en paralelo comentamos brevemente algunos de ellos a continuación.

Problemas de ordenación (o clasificación). El problema de la ordenación o clasificación consiste en la reorganización de un conjunto de objetos en una secuencia específica. Este problema aparece como parte fundamental en muchos algoritmos. Los dos ejemplos más conocidos de ordenación son el método de ordenación por fusión ó mezcla, conocido como ***MergeSort*** y el método de ordenación rápido, conocido como ***QuickSort***.

El problema de la ordenación es fundamental en programación y consiste básicamente en la reorganización de una colección de elementos en orden ascendente o descendente. Más detalladamente, supongamos que nos dan una secuencia de n elementos

$a_1, a_2, \ldots, a_n$ y que cada elemento a_i tiene una clave asociada denotada por $[a_i]$. Supongamos también que existe una relación de orden total sobre las claves, es decir, que para cualesquiera tres valores de la clave $[a_m], [a_n], [a_p]$

(i) se cumple que $[a_m] < [a_n]$, $[a_m] = [a_n]$ o $[a_m] > [a_n]$ y

(ii) si $[a_m] < [a_n]$ y $[a_n] < [a_p]$ entonces $[a_m] < [a_p]$.

Bajo estas condiciones, el problema de ordenación consiste en reorganizar la colección de n elementos de tal forma que sus claves formen una secuencia no decreciente. Es decir, debe encontrarse una permutación σ del conjunto $\{1, 2, \ldots, n\}$ tal que $[a_{\sigma(1)}] \leq [a_{\sigma(2)}] \leq \cdots \leq [a_{\sigma(n)}]$. Mencionamos brevemente a continuación dos tipos de algoritmos de ordenación que utilizan algoritmos del tipo *divide y vencerás*.

El algoritmo de ordenación *MergeSort* fue uno de los primeros algoritmos de ordenación que se desarrollaron, siendo introducido por John von Newmann en 1945 (véase [4]). Es un ejemplo clásico de algoritmo del tipo *divide y vencerás*. Para ordenar un vector de n elementos divide éste en dos vectores de tamaño $n/2$ (fase divide del algoritmo). Estos dos vectores son ordenados de forma recursiva por el algoritmo de ordenación por mezcla (fase vencerás del algoritmo). Finalmente, los dos vectores ordenados son combinados para producir el vector final ordenado.

El algoritmo de ordenación *QuickSort*, propuesto por Hoare [62], sigue un esquema similar al de ordenación por mezcla: particiona una lista en dos listas y las reordena mezclando los resultados. Las diferencias radican en el proceso de división: ahora se elige un objeto x de la lista y se divide la lista en dos partes, la primera formada por los elementos anteriores a x y la segunda formada por los elementos posteriores. El proceso de ordenación posterior y mezcla no presenta diferencias con el caso anterior.

De entre la numerosa bibliografía que existe relacionada con estos algoritmos de ordenación, podemos citar entre otros Akl [4], Chandra, Jain, Basu y Kumar [20], Evans y Yousif [45] y Wheat y Evans [116].

Problemas de multiplicación de objetos. El problema de la multiplicación de objetos estructurados nos proporciona otra aplicación de estrategias del tipo *divide y vencerás* como podemos ver en Gonnet y Baeza-Yates [53, capítulo 6]. Supongamos que queremos multiplicar los enteros positivos n y m y supongamos que el entero positivo n puede escribirse, respecto de una base β como

$(n_l, \ldots, n_0)$, verificándose que

$$n = n_l\beta^l + n_{l-1}\beta^{l-1} + \cdots + n_1\beta + n_0, \qquad \text{con} \qquad 0 \leq n_0, \ldots, n_l < \beta.$$

Esta expresión nos sugiere que todo número entero puede verse como una tabla de dígitos. Al multiplicar n por m, las tablas que los representan pueden romperse en dos partes de longitud prácticamente igual. Así, podemos escribir que $n = a\beta^k + b$ y $m = c \cdot \beta^k + d$. Según Mignotte [93, página 34], el producto puede obtenerse mediante la expresión

$$nm = y\beta^{2k} + (x - y - z)\beta^k + z, \quad \text{con} \qquad x = (a + b)(c + d), \quad y = ac, \quad z = bd. \tag{1.6}$$

Los productos x, y, z se obtiene de forma recursiva utilizando la misma estrategia. Notemos que esta estrategia del tipo *divide y vencerás* nos proporciona un ahorro computacional, ya que sólo son necesarias tres multiplicaciones para calcular (1.6).

En cuanto al producto de dos matrices A y B, es conocido el algoritmo *divide y vencerás* de Strassen, (véase [74] para una descripción más detallada del algoritmo), que calcula el producto $C = AB$ utilizando una técnica del tipo *divide y vencerás* basada en la división de A y B en cuatro bloques, lo que permite que mediante siete productos sea posible calcular C, frente a los ocho productos necesarios mediante un algoritmo tradicional.

Geometría computacional. Diversos problemas de geometría computacional también utilizan estrategias de tipo *divide y vencerás*. Entre éstos cabe citar el problema del cálculo de la envoltura convexa de un conjunto de puntos (véase por ejemplo Edelsbrunner [41, página 145] y Preparata y Shamos [99, página 112]). En este caso, se construye una lista ordenada de puntos de acuerdo con el valor de su abcisa. Después, dicha lista se divide en dos nuevas listas de la misma longitud y se calcula recursivamente la envoltura convexa de ambas listas. Finalmente, las dos envolturas se mezclan para calcular la envoltura del conjunto inicial de puntos.

Preparata y Shamos [99] también utilizan la técnica *divide y vencerás* para una aplicación relacionada con el problema de la construcción de los diagramas de Voronoi.

Problemas de búsqueda binaria. Se trata de un problema de búsqueda en un vector ordenado S de claves. Para buscar una clave q en S, comparamos q con el elemento situado en la mitad del vector. Así, si el tamaño de S es n, comparamos q con el elemento

$S[n/2]$. Si q aparece antes de $S[n/2]$, entonces ya sabemos que q se encuentra en la mitad superior de nuestro conjunto; si no, debe estar en la mitad inferior del conjunto. Procedemos recursivamente de la misma forma hasta que finalmente obtenemos la clave, siendo el número de comparaciones que llevamos a cabo de orden $\log n$.

Problemas de procesamiento de imágenes. Aunque técnicas del tipo *divide y vencerás* no son utilizadas ampliamente en el procesamiento de imágenes, sí hay ciertos problemas donde este tipo de algoritmos han probado su eficiencia. Stout [105] examina diversas características de los algoritmos *divide y vencerás*, así como algunas de sus implicaciones para el diseño de máquinas y lenguajes que puedan soportar la programación y ejecución eficiente de algoritmos de este tipo. Otros problemas donde se aplican algoritmos de este tipo están relacionados con las técnicas de visualización de flujos para la representación de campos vectoriales. Una de éstas, conocida con el nombre de *Spot Noise*, utiliza texturas para la visualización de campos de flujo (véase van Wiijk [112]). Leeuw y Liere [35] proponen un algoritmo *Spot Noise* del tipo divide y vencerás y lo aplican a dos problemas concretos: un modelo de polución atmosférica y a una simulación numérica directa del flujo de una turbulencia.

1.3 Métodos de resolución de sistemas lineales tridiagonales

Existe una amplia literatura que aborda el problema general de la resolución de sistemas de ecuaciones lineales tridiagonales. Numerosos algoritmos se han propuesto a lo largo de las últimas décadas para resolver tales sistemas en paralelo, aprovechando el extraordinario auge y desarrollo de la computación paralela. Algunos de estos algoritmos han sido especialmente diseñados para ser ejecutados en máquinas con arquitecturas paralelas específicas, mientras que un buen número de ellos son de carácter general.

El primer método que resolvía sistemas tridiagonales fue el algoritmo de la reducción cíclica propuesto por Hockney [63] en 1965, precursor de una amplia familia de algoritmos basados en este método. Aunque el método original no era paralelo, posteriormente fue paralelizado para sistemas tridiagonales por bloques y para sistemas tridiagonales. La idea básica consiste en eliminar la mitad de las incógnitas, reagrupar las ecuaciones y eliminar, de nuevo, la mitad de las incógnitas. El proceso continúa de forma recursiva hasta que

podemos despejar una de las incógnitas que a su vez nos permite despejar el resto de incógnitas mediante un proceso de sustitución. Este algoritmo ha demostrado su potencia y eficiencia para la resolución de problemas matriciales en general y, en particular, para problemas donde aparecen matrices estructuradas. Algunos problemas concretos donde se ha aplicado este método con gran éxito es en la resolución de la ecuación de Poisson mediante una aproximación en diferencias finitas y para la resolución de ciertas recurrencias. El algoritmo ha sido ampliamente paralelizado y utilizado en una gran variedad de arquitecturas (véase Golub y Ortega [51]). Para una exposición y estudio exahustivo de este método y de algunas de sus aplicaciones, véase Gander y Golub [48] y Bondeli y Gander [14].

Muchos de los métodos propuestos para sistemas tridiagonales se puden agrupar en tres grandes familias: reducción cíclica, de la que ya se ha hablado anteriormente, *recursive doubling* y *divide y vencerás*. El método *recursive doubling* fue propuesto por Stone [103] y puede considerarse como el primer método diseñado explícitamente en paralelo para la resolución de sistemas tridiagonales. Es un método que resuelve recurrencias de primer orden y resulta estable únicamente para matrices estrictamente diagonal dominantes. Para un estudio más detallado de su estabilidad y del tipo de matrices que lo hacen estable, véase Dubois y Rodríguez [40] donde además se realizan comparaciones numéricas con el método de eliminación de Gauss. Tanto el método *recursive doubling* y de reducción cíclica se analizan detenidamente en la sección 1.3.3, ya que ambos se utilizan a lo largo de esta memoria.

Cabe mencionar el método propuesto por Larriba, Jorba y Navarro [79] para sistemas por bandas estrictamente diagonal dominantes y que se conoce con el nombre de *método de las particiones superpuestas*. Un estudio de la precisión de este nuevo método puede verse en Larriba, Jorba y Navarro [78].

1.3.1 Métodos *divide y vencerás* para sistemas tridiagonales

El primer método del tipo *divide y vencerás* para resolver sistemas tridiagonales data de 1978 y fue propuesto por Sameh y Kuck [100], quienes propusieron dos algoritmos paralelos para la solución de sistemas de este tipo. Estos algoritmos utilizaban rotaciones de Givens para realizar la eliminación de los elementos y requerían el uso de un número de procesadores con la única restricción de ser menor que el tamaño del sistema.

En 1984, Lawrie y Sameh [81] introdujeron una estrategia del tipo *divide y vencerás* para la resolución de sistemas tridiagonales por bloques, en el que utilizaban la eliminación de Gauss para la eliminación en los bloques. Además analizaron el comportamiento del algoritmo y su complejidad para máquinas paralelas con diferentes topologías en sus redes de conexión. Un año después, en 1985, Meier [91] obtuvo un algoritmo *divide y vencerás* que resolvía sistemas de ecuaciones por bandas. Esto supuso la primera generalización de esta estrategia para matrices no tridiagonales. Conroy [32] también obtuvo una generalización similar a la de Meier.

En 1987, Bar-On [6] introdujo una modificación importante en un algoritmo *divide y vencerás* para sistemas tridiagonales, consistente en la paralelización de la resolución del sistema tridiagonal reducido que se obtiene y cuya solución nos permite obtener las soluciones del sistema original. Además, demuestra que el algoritmo es comparable desde el punto de vista de su coste computacional con el método de reducción cíclica. Esta modificación es interesante desde el punto de vista de introducir un grado más de paralelización en los algoritmos basados en estrategias *divide y vencerás*. A lo largo de esta memoria veremos que algunas de las modificaciones que se realizan sobre algunos algoritmos básicos tienen como objetivo la paralelización del método en la resolución del sistema tridiagonal auxiliar que se construye.

Diversas comparaciones entre algoritmos del tipo *divide y vencerás* con otros métodos clásicos de resolución de sistemas tridiagonales, como el de reducción cíclica, fueron realizadas por Johnsson [68] y Johnsson y Ho [69], quienes llevaron a cabo las comparaciones en distintos tipos de computadores paralelos con diferentes topologías en sus redes de comunicación. Dongarra y Johnsson [39] propusieron un nuevo algoritmo *divide y vencerás* para sistemas por bandas, analizando su comportamiento y complejidad en diferentes arquitecturas.

En años posteriores diversos autores propusieron y analizaron algoritmos de este tipo para máquinas concretas, analizando el comportamiento de estos algoritmos cuando se utilizaban diversos modelos de comunicación. Entre estos trabajos cabe citar los desarrollados por Krechel, Plum y Stüben [75, 76], Cox y Knisely [33], Hoffmann y Potma [66] y Kumar [77].

En 1988 Van der Vorst [111] analizó dos versiones del algoritmo *divide y vencerás* para sistemas bidiagonales en máquinas paralelas y dos versiones en procesadores vectoriales. Un análisis del error en un algoritmo paralelo para resolver sistemas tridiagonales fue presentado por Van der Vorst [110] un año antes.

En 1991, Bondeli [13] propone un algoritmo *divide y vencerás* para sistemas tridiagonales válido tanto para máquinas paralelas como vectoriales. En dicho trabajo se realiza un estudio comparativo del coste computacional de este método frente a otros métodos clásicos para sistemas tridiagonales y analiza el caso de sistemas estrictamente diagonal dominantes. Este método, no válido para sistemas tridiagonales por bloques, es generalizado posteriormente para este tipo de sistemas por Mehrmann [90].

Sun, Zhang y Ni [106] proponen una variante del algoritmo *divide y vencerás* en el que el sistema tridiagonal auxiliar se resuelve utilizando el método *recursive doubling*. Además realizan comparaciones numéricas con otros métodos tradicionales.

En 1993, Larriba, Jorba y Navarro [79] establecen un nuevo criterio de parada para algoritmos *divide y vencerás* y muestran resultados en computadores vectoriales. Más recientemente, López [85] propone en su tesis doctoral una arquitectura unificada para algoritmos *divide y vencerás* en sistemas tridiagonales.

1.3.2 Método del *recursive doubling*

Si escribimos con detalle el sistema (1.1) tendremos

$$\left.\begin{array}{ccccccccccccc}
a_1x_1 & + & b_1x_2 & & & & & & & & & = & d_1 \\
c_2x_1 & + & a_2x_2 & + & b_2x_3 & & & & & & & = & d_2 \\
 & & c_3x_2 & + & a_3x_3 & + & b_3x_4 & & & & & = & d_3 \\
 & & & \ddots & & \ddots & & & & & & \vdots & \\
 & & & & c_{n-1}x_{n-2} & + & a_{n-1}x_{n-1} & + & b_{n-1}x_n & = & d_{n-1} \\
 & & & & & & c_nx_{n-1} & + & a_nx_n & = & d_n
\end{array}\right\}, \tag{1.7}$$

que puede representarse mediante una relación de recurrencia de tres términos de la forma

$$c_i x_{i-1} + a_i x_i + b_i x_{i+1} = d_i, \qquad i = 1, 2, \ldots, n, \tag{1.8}$$

con $c_1 = b_n = 0$. Por comodidad en la notación, supondremos también que $x_0 = x_{n+1} = 0$.

Despejando x_{i+1} de la ecuación (1.8) obtenemos, para $i = 1, 2, \ldots, n-1$, que

$$x_{i+1} = \alpha_i x_i + \beta_i x_{i-1} + \gamma_i,$$

siendo

$$\alpha_i = -\frac{a_i}{b_i}, \qquad \beta_i = -\frac{c_i}{b_i}, \qquad \gamma_i = \frac{d_i}{b_i}. \tag{1.9}$$

Esta fórmula de recurrencia puede escribirse matricialmente como

$$\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_i & \beta_i & \gamma_i \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ x_{i-1} \\ 1 \end{bmatrix}.$$

Si definimos, para $i = 1, 2, \ldots, n-1$,

$$\boldsymbol{x}_{i-1} = \begin{bmatrix} x_i \\ x_{i-1} \\ 1 \end{bmatrix} \qquad \text{y} \qquad B_i = \begin{bmatrix} \alpha_i & \beta_i & \gamma_i \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1.10}$$

entonces tenemos que

$$\boldsymbol{x}_i = B_i \boldsymbol{x}_{i-1} \tag{1.11}$$

y como

$$\boldsymbol{x}_0 = \begin{bmatrix} x_1 \\ x_0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ 0 \\ 1 \end{bmatrix}, \tag{1.12}$$

el único elemento que necesitamos calcular para comenzar la recursión es x_1.

Ahora bien, de la ecuación (1.11) tenemos que

$$\boldsymbol{x}_1 = B_1\boldsymbol{x}_0, \quad \boldsymbol{x}_2 = B_2B_1\boldsymbol{x}_0, \quad \dots, \quad \boldsymbol{x}_n = B_nB_n\cdots B_2B_1\boldsymbol{x}_0.$$

Si, para $i = 1, 2, \dots, n$, escribimos $C_i = B_iB_{i-1}\cdots B_2B_1$, entonces

$$\boldsymbol{x}_i = C_i\boldsymbol{x}_0 \tag{1.13}$$

y en particular

$$\boldsymbol{x}_n = C_n\boldsymbol{x}_0. \tag{1.14}$$

Por la forma que tienen las matrices B_i es evidente que

$$C_n = \begin{bmatrix} l_{00} & l_{01} & l_{02} \\ l_{10} & l_{11} & l_{12} \\ 0 & 0 & 1 \end{bmatrix},$$

por tanto, como $x_{n+1} = x_0 = 0$, de la ecuación (1.14) tendremos que $0 = l_{00}x_1 + l_{02}$ por lo que

$$x_1 = -\frac{l_{02}}{l_{00}}. \tag{1.15}$$

En consecuencia, para empezar la recursión necesitamos calcular l_{00} y l_{02}. El siguiente ejemplo detalla las operaciones que se realizan para resolver un sistema tridiagonal utilizando este método.

Ejemplo 1.1 *Supongamos que queremos resolver el sistema (1.1) donde* $n = 12$,

$$A = \begin{bmatrix} 2 & 1 & & & & & & & & & & \\ 1 & 4 & 1 & & & & & & & & & \\ & -1 & 4 & -1 & & & & & & & & \\ & & -2 & 4 & 2 & & & & & & & \\ & & & -2 & 4 & 2 & & & & & & \\ & & & & 2 & -4 & 1 & & & & & \\ & & & & & 1 & 2 & 1 & & & & \\ & & & & & & -2 & 4 & 1 & & & \\ & & & & & & & -1 & 4 & -1 & & \\ & & & & & & & & 1 & -4 & 1 & \\ & & & & & & & & & -2 & 6 & 1 \\ & & & & & & & & & & 1 & 2 \end{bmatrix} \quad y \quad \boldsymbol{d} = \begin{bmatrix} 3 \\ 6 \\ 2 \\ 4 \\ 4 \\ -1 \\ 4 \\ 3 \\ 2 \\ -2 \\ 5 \\ 3 \end{bmatrix}.$$

Entonces, a partir de la expresión (1.9) obtenemos que

$$B_1 = \begin{bmatrix} -2 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; B_2 = \begin{bmatrix} -4 & -1 & 6 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; B_3 = \begin{bmatrix} 4 & -1 & -2 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; B_4 = \begin{bmatrix} -2 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; B_5 = \begin{bmatrix} -2 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; B_6 = \begin{bmatrix} 4 & -2 & -1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$B_7 = \begin{bmatrix} -2 & -1 & 4 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ B_8 = \begin{bmatrix} -4 & 2 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ B_9 = \begin{bmatrix} 4 & -1 & -2 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ B_{10} = \begin{bmatrix} 4 & -1 & -2 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ B_{11} = \begin{bmatrix} -6 & 2 & 5 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ B_{12} = \begin{bmatrix} -2 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

luego

$$C_1 = \begin{bmatrix} -2 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad C_2 = \begin{bmatrix} 7 & 4 & -6 \\ -2 & -1 & 3 \\ 0 & 0 & 1 \end{bmatrix}, \quad \ldots, \qquad C_{12} = \begin{bmatrix} 1106996 & 626732 & -1106996 \\ -609200 & -344902 & 609201 \\ 0 & 0 & 1 \end{bmatrix}$$

y por la expresión (1.15) se tiene que $x_1 = -\dfrac{l_{02}}{l_{00}} = 1$, *luego* $\boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$. *Ahora, de la ecuación (1.13) calculamos* $\boldsymbol{x}_i$, *para* $i = 1, 2, \ldots, 12$,

obteniendo el vector solución, dado por

$$\boldsymbol{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T.$$

1.3.3 Método de reducción cíclica

Ya vimos en la sección 1.3.2 que el sistema (1.1) puede ser descrito mediante la relación de recurrencia de tres términos dada por la expresión (1.8). La idea en que se basa el método de la reducción cíclica es tomar conjuntos de tres ecuaciones consecutivas superponiendo la última del conjunto anterior y aplicar operaciones elementales sobre la ecuación central con el fin de anular ciertos coeficientes. Tomando las ecuaciones centrales que han sido modificadas, podemos construir sistemas tridiagonales auxiliares hasta que al final del proceso queda una sola ecuación. Veamos este proceso con detalle.

Suponemos que el número de ecuaciones del sistema es $2^m - 1$, siendo m un entero positivo. Podemos escribir tres ecuaciones adyacentes de la expresión (1.8) como

$$\left.\begin{array}{rcrcrcrcl} c_{i-1}x_{i-2} & + & a_{i-1}x_{i-1} & + & b_{i-i}x_i & & & = & d_{i-1} \\ & & c_i x_{i-1} & + & a_i x_i & + & b_i x_{i+1} & = & d_i \\ & & & & c_{i+1}x_i & + & a_{i+1}x_{i+1} + b_{i+1}x_{i+2} & = & d_{i+1} \end{array}\right\} \tag{1.16}$$

para $i = 1, 3, 5, \ldots, 2^m - 3$; ahora modificamos la ecuación central realizando las siguientes operaciones elementales por filas

(i) restamos a la ecuación central la primera ecuación multiplicada por $\alpha_i = \dfrac{c_i}{a_{i-1}}$,

(ii) restamos a la nueva ecuación central la tercera ecuación multiplicada por $\gamma_i = \dfrac{b_i}{a_{i+1}}$,

con lo que la nueva ecuación central del sistema (1.16) se transforma en

$$c_i^{(1)} x_{i-2} + a_i^{(1)} x_i + b_i^{(1)} x_{i+2} = d_i^{(1)}, \tag{1.17}$$

donde

$$c_i^{(1)} = -\alpha_i c_i, \quad b_i^{(1)} = -\gamma_i b_{i+1}, \quad a_i^{(1)} = a_i - \alpha_i b_{i-1} - \gamma_i c_{i+1}, \quad d_i^{(1)} = d_i + \alpha_i d_{i-1} + \gamma_i d_{i+1}. \tag{1.18}$$

Los superíndices que aparecen en las expresiones (1.17) y (1.18) indican que nos encontramos en la primera fase de la reducción cíclica. Las ecuaciones (1.17) establecen nuevas relaciones entre las variables impares, ya que eliminamos los coeficientes de las variables

pares en la ecuación central. Además, constituyen un sistema tridiagonal, dado por

$$\left.\begin{array}{ccccccccccccc}
a_1^{(1)}x_1 & + & b_1^{(1)}x_3 & & & & & & & & & = & d_1^{(1)} \\
c_3^{(1)}x_1 & + & a_3^{(1)}x_3 & + & b_3^{(1)}x_5 & & & & & & & = & d_3^{(1)} \\
 & & c_5^{(1)}x_3 & + & a_5^{(1)}x_5 & + & b_5^{(1)}x_7 & & & & & = & d_5^{(1)} \\
 & & & \ddots & & \ddots & & \ddots & & & & \vdots & \\
 & & & & c_{n-3}^{(1)}x_{n-5} & + & a_{n-3}^{(1)}x_{n-3} & + & b_{n-3}^{(1)}x_{n-3} & = & d_{n-3}^{(1)} \\
 & & & & & & c_{n-1}^{(1)}x_{n-3} & + & a_{n-1}^{(1)}x_{n-1} & = & d_{n-1}^{(1)}
\end{array}\right\}.$$

En este nuevo sistema tridiagonal, el número de ecuaciones se ha reducido a la mitad. Claramente, este proceso descrito hasta aquí puede repetirse sucesivamente hasta que, después de $s = \log(2^m) - 1$ niveles de reducción, sólamente tenemos una ecuación central, para $i = 2^{m-1}$. Esta ecuación es $a_{2^{m-1}}^{(s)} x_{2^{m-1}} = d_{2^{m-1}}^{(s)}$, de donde podemos despejar la variable central como

$$x_{2^{m-1}} = \frac{d_{2^{m-1}}^{(s)}}{a_{2^{m-1}}^{(s)}}. \tag{1.19}$$

Las variables que quedan por determinar pueden ser calculadas siguiendo un proceso conocido como *fan-in*. Como conocemos la variable $x_{2^{u-1}}$, podemos determinar las variables intermedias x_i, para $i = \dfrac{2^m}{4}, \dfrac{3 \cdot 2^m}{4}$, mediante la expresión

$$x_i = \frac{d_i^{(s-1)} - c_i^{(s-1)} x_{i-\frac{2^m}{4}} - b_i^{(s-1)} x_{i+\frac{2^m}{4}}}{a_i^{(s-1)}}. \tag{1.20}$$

Este proceso puede aplicarse de forma repetida hasta que se obtienen todas las componentes de la solución. Veamos un ejemplo que resuma las características de este método.

Ejemplo 1.2 *Supongamos que queremos resolver el sistema (1.1) donde* $n = 15$,

$$A = \begin{bmatrix}
2 & 3 & & & & & & & & & & & & & \\
4 & -1 & 2 & & & & & & & & & & & & \\
 & 1 & -1 & 3 & & & & & & & & & & & \\
 & & 4 & 5 & 1 & & & & & & & & & & \\
 & & & 2 & -1 & 3 & & & & & & & & & \\
 & & & & 1 & 4 & -1 & & & & & & & & \\
 & & & & & 2 & 1 & 2 & & & & & & & \\
 & & & & & & -3 & 5 & 2 & & & & & & \\
 & & & & & & & 4 & -1 & 6 & & & & & \\
 & & & & & & & & -1 & 5 & 2 & & & & \\
 & & & & & & & & & 7 & 2 & 3 & & & \\
 & & & & & & & & & & 1 & -5 & 1 & & \\
 & & & & & & & & & & & 4 & 1 & 1 & \\
 & & & & & & & & & & & & 3 & 1 & 4 \\
 & & & & & & & & & & & & & 3 & 1
\end{bmatrix} \quad y \quad \boldsymbol{d} = \begin{bmatrix} 5 \\ 5 \\ 3 \\ 10 \\ 4 \\ 4 \\ 5 \\ 4 \\ 9 \\ 6 \\ 12 \\ 3 \\ 6 \\ 8 \\ 4 \end{bmatrix}.$$

En la primera fase del proceso de reducción cíclica tomamos las siguientes ternas de ecuaciones

$$\left.\begin{array}{rcrcrcrcl}
2x_1 & + & 3x_2 & & & & & = & 5 \\
4x_1 & - & x_2 & + & 2x_3 & & & = & 5 \\
 & & x_2 & - & x_3 & + & 3x_4 & = & 3
\end{array}\right\},
\qquad
\left.\begin{array}{rcrcrcrcl}
x_2 & - & x_3 & + & 3x_4 & & & = & 3 \\
 & & 4x_3 & + & 5x_4 & + & x_5 & = & 10 \\
 & & & & 2x_4 & - & x_5 & + & 3x_6 = 4
\end{array}\right\},$$

$$\left.\begin{array}{rcrcrcrcrcl} 2x_4 & - & x_5 & + & 3x_6 & & & = & 4 \\ & & x_5 & + & 4x_6 & - & x_7 & = & 4 \\ & & & & 2x_6 & + & x_7 & + & 2x_8 & = & 5 \end{array}\right\},\qquad \left.\begin{array}{rcrcrcrcrcl} 2x_6 & + & x_7 & + & 2x_8 & & & = & 5 \\ & - & 3x_7 & + & 5x_8 & + & 2x_9 & = & 4 \\ & & & & 4x_8 & - & x_9 & + & 6x_{10} & = & 9 \end{array}\right\},$$

$$\left.\begin{array}{rcrcrcrcrcl} 4x_8 & - & x_9 & + & 6x_{10} & & & = & 9 \\ & - & x_9 & + & 5x_{10} & + & 2x_{11} & = & 6 \\ & & & & 7x_{10} & + & 2x_{11} & + & 3x_{12} & = & 12 \end{array}\right\},\qquad \left.\begin{array}{rcrcrcrcrcl} 7x_{10} & + & 2x_{11} & + & 3x_{12} & & & = & 12 \\ & & x_{11} & - & 5x_{12} & + & x_{13} & = & -3 \\ & & & & 4x_{12} & + & x_{13} & + & x_{14} & = & 6 \end{array}\right\},$$

$$\left.\begin{array}{rcrcrcrcrcl} 4x_{12} & + & x_{13} & + & x_{14} & & & = & 6 \\ & & 3x_{13} & + & x_{14} & + & 4x_{15} & = & 8 \\ & & & & 3x_{14} & + & x_{15} & = & 6 \end{array}\right\}.$$

Efectuando sobre las ecuaciones centrales de estas ternas las operaciones elementales descritas por las expresiones (1.17) y (1.18) obtenemos el sistema

$$\left.\begin{array}{rcrcrcrcrcrcrcl} -5x_2 & + & 6x_4 & & & & & & & & & & = & 1 \\ 4x_2 & + & 19x_4 & + & 3x_6 & & & & & & & & = & 26 \\ & & 2x_4 & + & 9x_6 & + & 2x_8 & & & & & & = & 13 \\ & & & & 6x_6 & + & 19x_8 & + & 12x_{10} & & & & = & 37 \\ & & & & & & -4x_8 & - & 8x_{10} & - & 3x_{12} & & = & -15 \\ & & & & & & & & -\tfrac{7}{2}x_{10} & - & \tfrac{21}{2}x_{12} & - & x_{14} & = & -15 \\ & & & & & & & & & & -12x_{12} & - & 14x_{14} & = & -26 \end{array}\right\}.$$

Agrupamos de nuevo en ternas de ecuaciones consecutivas, con lo que obtenemos

$$\left.\begin{array}{rcrcrcl} -5x_2 & + & 6x_4 & & & = & 1 \\ 4x_2 & + & 19x_4 & + & 3x_6 & = & 26 \\ & & 2x_4 & + & 9x_6 + 2x_8 & = & 13 \end{array}\right\}, \quad \left.\begin{array}{rcrcrcrcrcl} & & 2x_4 & + & 9x_6 & + & 2x_8 & & & = & 13 \\ & & & & 6x_6 & + & 19x_8 & + & 12x_{10} & = & 37 \\ & & & & & - & 4x_8 & - & 8x_{10} - 3x_{12} & = & -15 \end{array}\right\}, \tag{1.21}$$

$$\left.\begin{array}{rcrcrcl} -4x_8 & - & 8x_{10} & - & 3x_{12} & = & -15 \\ & - & \frac{7}{2}x_{10} & - & \frac{21}{2}x_{12} - x_{14} & = & -15 \\ & & & - & 12x_{12} - 14x_{14} & = & -26 \end{array}\right\}. \tag{1.22}$$

Volvemos a efectuar las operaciones elementales habituales sobre las ecuaciones centrales, para llegar al sistema tridiagonal de tres ecuaciones

$$\left.\begin{array}{rcrcrcl} \frac{347}{15}x_4 & - & \frac{2}{3}x_8 & & & = & \frac{337}{15} \\ \frac{-4}{3}x_4 & + & \frac{35}{3}x_8 & - & \frac{9}{2}x_{12} & = & \frac{35}{6} \\ & & \frac{7}{4}x_8 & - & \frac{933}{112}x_{12} & = & -\frac{737}{112} \end{array}\right\}. \tag{1.23}$$

Después de este paso únicamente podemos formar una terna con estas tres ecuaciones, que es exactamente el conjunto de ecuaciones (1.23). Repitiendo de nuevo las operaciones elementales que se han realizado en etapas anteriores, reducimos la ecuación central a una única ecuación lineal cuya incógnita es x_8*, de la forma* $\frac{1152867}{107917}x_8 = \frac{1152867}{107917}$*, lo que significa que* $x_8 = 1$*. Una vez calculada la componente central de la solución, efectuamos un proceso de sustitución regresiva conducente a la obtención del resto de componentes de la solución. Así, a partir de la componente* x_8*, podemos obtener las componentes* x_4 *y* x_{12} *por simple sustitución en la terna de ecuaciones (1.23), obteniéndose que* $x_4 = x_{12} = 1$*. Ahora, a partir de las variables* x_4*,* x_8 *y* x_{12} *y sustituyendo de forma adecuada en (1.21) y (1.22) se obtiene que* $x_2 = x_6 = x_{10} = x_{14} = 1$*.*

Siguiendo un proceso de sustitución regresiva similar, se obtiene el resto de las componentes del vector solución, resultando que $x_1 = x_2 = \cdots = x_{14} = x_{15} = 1$, *por lo que la solución es el vector cuyas componentes son todas iguales a uno.*

En algunos casos tendremos que resolver un sistema tridiagonal de la forma $B\boldsymbol{z} = \boldsymbol{h}$ mediante el método de reducción cíclica donde

$$B = \begin{bmatrix} a_0 & 1 & & & & & & \\ 1 & a_1 & b_1 & & & & & \\ & c_2 & a_2 & 1 & & & & \\ & & 1 & a_3 & b_3 & & & \\ & & & c_4 & a_4 & 1 & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & c_{2p-4} & a_{2p-4} & 1 \\ & & & & & & 1 & a_{2p-3} \end{bmatrix}, \quad \boldsymbol{z} = \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ \vdots \\ z_{2p-4} \\ z_{2p-3} \end{bmatrix} \quad \text{y} \quad \boldsymbol{h} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ \vdots \\ h_{2p-4} \\ h_{2p-3} \end{bmatrix}. \tag{1.24}$$

Siguiendo un proceso similar al descrito en esta sección para la resolución del sistema (1.1), inicialmente tomamos tres ecuaciones adyacentes

$$\left.\begin{array}{rcrcrcrcl} c_{i-1}z_{i-2} & + & a_{i-1}z_{i-1} & + & z_i & & & = & h_{i-1} \\ & & z_{i-1} & + & a_iz_i & + & b_iz_{i+1} & = & h_i \\ & & & & c_{i+1}z_i & + & a_{i+1}z_{i+1} & + & z_{i+2} = h_{i+1} \end{array}\right\}. \tag{1.25}$$

El primer paso de la reducción cíclica consiste en multiplicar la primera ecuación por $\alpha_i = \dfrac{1}{a_{i-1}}$ y restarla de la segunda ecuación, con lo que la ecuación central se convierte en

$$-\frac{c_{i-1}}{a_{i-1}}z_{i-2} + \left(a_i - \frac{1}{a_{i-1}}\right)z_i + b_iz_{i+1} = h_i - \frac{h_{i-1}}{a_{i-1}}. \tag{1.26}$$

El segundo paso consiste en multiplicar la tercera ecuación por $\gamma_i = \dfrac{b_i}{a_{i+1}}$ y restarla de la segunda ecuación, con lo que la ecuación central queda modificada como

$$-\frac{c_{i-1}}{a_{i-1}} z_{i-2} + \left(a_i - \frac{1}{a_{i-1}} - \frac{b_i c_{i+1}}{a_{i+1}} \right) z_i - \left(\frac{b_i}{a_{i+1}} \right) z_{i+2} = h_i - \frac{h_{i-1}}{a_{i-1}} - \frac{b_i h_{i+1}}{a_{i+1}},$$

que puede escribirse como

$$c_i^{(1)} z_{i-2} + a_i^{(1)} z_i + b_i^{(1)} z_{i+2} = h_i^{(1)}, \tag{1.27}$$

donde

$$\left.\begin{aligned} c_i^{(1)} &= -\frac{c_{i-1}}{a_{i-1}} \\ a_i^{(1)} &= a_i - \frac{1}{a_{i-1}} - \frac{b_i c_{i+1}}{a_{i+1}} \\ b_i^{(1)} &= -\frac{b_i}{a_{i+1}} \\ h_i^{(1)} &= h_i - \frac{h_{i-1}}{a_{i-1}} - \frac{b_i h_{i+1}}{a_{i+1}} \end{aligned}\right\}. \tag{1.28}$$

Se utiliza la notación $a_i^{(1)}$, $b_i^{(1)}$ y $c_i^{(1)}$ para denotar los coeficientes que acompañan a las variables de la ecuación i-ésima en el primer nivel de reducción de variables. Análogamente, los coeficientes del segundo nivel de reducción para la ecuación i-ésima se denotarán por $a_i^{(2)}$, $b_i^{(2)}$ y $c_i^{(2)}$ y así sucesivamente. El resto del proceso es similar al descrito anteriormente.

1.4 Métodos *divide y vencerás* para calcular la inversa de una matriz

El problema del cálculo de la inversa de una matriz A es mucho más costoso, desde el punto de vista computacional, que el de resolver el sistema (1.1), por lo que la mayoría de problemas que incluyen el cálculo de inversas pueden ser reformulados en términos de la resolución de sistema lineales. Por ello, podemos afirmar en términos generales que el cálculo explícito de la inversa de una matriz debe evitarse siempre que sea posible. Sin embargo, como veremos más detalladamente a continuación, en algunas ocasiones su cálculo es necesario.

En ingeniería estructural, el cálculo de la inversa de una matriz es un problema que aparece con cierta frecuencia. La razón de esto es que en el análisis de una estructura en equilibrio aparecen sistemas con un número muy alto de ecuaciones e incógnitas. A menudo, dichas ecuaciones son lineales aun incluso después de considerar deformaciones sobre el material. Por ejemplo, consideramos el problema de una viga elástica horizontal sujeta en cada extremo y sometida a fuerzas en n puntos de la misma $1, 2, \ldots, n$. Si $\boldsymbol{x} \in \mathbb{R}^n$ es la relación de fuerzas en estos puntos y $\boldsymbol{d} \in \mathbb{R}^n$ representa la desviación de la viga en estos n puntos, entonces por la ley de Hook, tenemos un sistema lineal donde la matriz de coeficientes recibe el nombre de matriz de flexibilidad. Su inversa recibe el nombre de matriz **stiffness**. A menudo, es necesario calcular la inversa de esta matriz debido al significado físico de sus columnas. Así, la primera columna de la matriz de *stiffness* representa las fuerzas que deben aplicarse en los n puntos con el fin de producir una desviación particular en el punto 1 y una desviación nula en el resto de puntos. El mismo razonamiento es válido para el resto de las columnas.

Otro problema donde es necesario el cálculo explícito de una matriz inversa es el llamado **spring-mass problem** en física, en el que algunas masas se encuentran suspendidas verticalmente por una serie de muelles y debemos considerar las constantes de los muelles y los desplazamientos de cada muelle desde su posición de equilibrio. Cuando aplicamos la segunda ley de Newton a este problema obtenemos un sistema cuya matriz de coeficientes es tridiagonal y diagonal dominante. La inversa de esta matriz es la matriz de *stiffness*, que está directamente relacionada con el desplazamiento de las masas cuando algunas fuerzas externas se imponen sobre ellas. Así, el elemento (i, j) de esta matriz inversa nos proporciona el desplazamiento de la masa i cuando sobre la masa j se impone una fuerza externa de una unidad de fuerza.

También podemos encontrar ciertas aplicaciones fuera del campo de la ingeniería o la física donde se deben calcular matrices inversas. Entre éstas encontramos aplicaciones en el campo de la teoría de sistemas dinámicos, relacionadas con la determinación de los puntos fijos de un sistema dinámico mediante el método de Newton (véase, por ejemplo, Stoer y Burlirsch [102]). Para una descripción más detallada de ciertas aplicaciones de las inversas, véase Keener y Bogar [73] y Meurant [92], quien describe ciertas aplicaciones relacionadas con precondicionadores y proporciona un gran número de referencias relacionadas con matrices inversas.

Diversos autores han estudiado el problema del cálculo de la inversa de una matriz A tridiagonal, irreducible y diagonal dominante, proponiendo algoritmos eficientes para su cálculo. Podemos citar a Lewis [84], Meurant [92], Usnami [108] y Huang y McColl [67]. Spaletta y Evans [101] y Mehrmann [90] proponen el método *recursive decoupling* para el cálculo de la inversa de una matriz tridiagonal.

Climent, Tortosa y Zamora [31] proponen un algoritmo BSP *divide y vencerás* para calcular la inversa de una matriz tridiagonal, irreducible y diagonal dominante, que constituye una aplicación de la fórmula de Sherman-Morrison para matrices con estas características. En dicho artículo se estudian dos algoritmos BSP siguiendo diferentes modelos de comunicación entre los procesadores, realizándose un estudio numérico teórico y experimental de los tiempos que se obtienen para los dos algoritmos en una máquina IBM SP2. Finalmente, se realiza un estudio comparativo computacional entre estos algoritmos y un algoritmo tradicional del tipo *divide y vencerás* basado en el método de Bondeli para la resolución de sistemas tridiagonales, adaptado al cálculo de la inversa de una matriz tridiagonal. Los resultados teóricos y experimentales demuestran que los algoritmos del tipo *divide y vencerás* que calculan directamente la matriz inversa son más rápidos que el algoritmo clásico de resolución de sistemas tridiagonales modificado para calcular la inversa.

1.5 Breve sumario de aplicaciones de los sitemas tridiagonales

De entre las numerosas aplicaciones y problemas en el ámbito de la computación donde aparecen sistemas tridiagonales citamos muy brevemente algunos de los más notables, en los que la resolución de dichos sistemas constituye una parte esencial del proceso conducente a la obtención de la solución de dichos problemas.

- **Ecuaciones diferenciales parciales.** Diversos métodos iterativos para la solución de ecuaciones diferenciales parciales necesitan resolver un conjunto de sistemas tridiagonales múltiples como resultado de un proceso de discretización por mallas. Entre estos métodos destaca el *Alternating Direction Implicit* (ADI), estudiado por Johnsson, Saad y Schultz [70] para computadores paralelos y por Ortega y Voigt [97] entre otros.

- **Ajuste de curvas por esplines cúbicos.** El ajuste de curvas se utiliza en muchas aplicaciones. Por ejemplo, en Diseño Asistido por Ordenador (CAD) se utiliza el ajuste de curvas para definir la forma de las superficies que forman los objetos diseñados. También se utiliza en la modelización de datos obtenidos de diferentes procesos. El proceso de ajuste de una curva se desarrolla a partir de un conjunto de puntos dados. Entre las diferentes técnicas que se utilizan para obtener un modelo de curva que se ajuste a ese conjunto de puntos destacamos los esplines cúbicos naturales y la interpolación por B-esplines. En el proceso de cálculo que requiere la aplicación de estas técnicas, es necesario resolver sistemas tridiagonales cuya matriz de coeficientes es Toeplitz y diagonal dominante, siendo su diagonal dominanza 2. Véase Chung y Shen [22] para una descripción más detallada de esta aplicación.

- **Discretización de ecuaciones diferenciales.** El comportamiento eléctrico de las neuronas puede modelizarse por medio de ecuaciones diferenciales parciales que evolucionan con el tiempo, como puede verse en Hines [61]. La discretización por diferencias finitas de estas ecuaciones dan lugar a sistemas tridiagonales que deben resolverse para comprender la evolución de las neuronas con el tiempo. La solución de estos sistemas puede desarrollarse tanto por métodos directos como iterativos. Los sistemas tridiagonales que deben resolverse mediante esta técnica son diagonal dominantes ya que las ecuaciones diferenciales parciales son de tipo parabólico, variando su tamaño de forma considerable con el dominio de la discretización. Si el número de dominios es muy grande, el paralelismo se obtiene de una forma natural asignando un conjunto de sistemas tridiagonales a cada procesador, intentando que la carga computacional se encuentre equilibrada.