UNIVERSIDAD
POLITECNICA
DE VALENCIA

DSIC

Trabajo Fin de Máster
Máster en Computación Paralela y Distribuida

Extensión de la librería SLEPc para el cálculo de todos los valores propios contenidos en un intervalo en problemas simétricos

Carmen Campos González
Director: José E. Román Moltó

17 de febrero de 2012

Índice general

Introducción

Dada una matriz $A \in \mathbb{C}^{n\times n}$ (o $A \in \mathbb{R}^{n\times n}$), el *problema (estándar) de valores propios* (o *autovalores*) busca los valores $\lambda \in \mathbb{C}$ y $x \in \mathbb{C}^n \backslash \{0\}$ que verifican la ecuación

$$Ax = \lambda x. \tag{1}$$

Este problema se corresponde con un caso particular del conocido como *problema generalizado de valores propios*, que busca las soluciones no triviales de la ecuación

$$Ax = \lambda Bx, \tag{2}$$

donde B es una matriz compleja (o real) de orden n.

Hay muchos ejemplos de aplicaciones en la ingeniería y en la ciencia en general, en los que el modelo matemático asociado plantea la resolución de un problema de valores propios, tanto en la forma estándar (1), como generalizada (2). En física, por ejemplo, los autovalores suelen estar conectados con frecuencias de vibración de objetos (por ejemplo puentes, ondas o partículas). En disciplinas como el análisis dinámico de estructuras de edificación, toma especial relevancia determinar dichas frecuencias de vibración. Los valores propios están presentes en conceptos matriciales, como los valores singulares, la norma, el número de condición y la potencia de una matriz. El problema de valores propios se plantea también en la resolución de sistemas de ecuaciones en derivadas parciales una vez han sido éstos discretizados.

El interés en este tipo de problemas ha dado lugar al desarrollo de diversos métodos de resolución, algunos de los cuales han sido implementados en distintas librerías computacionales para su utilización en la resolución de problemas en ingeniería. La elección de un método u otro vendrá determinado por las características del problema.

Un primer aspecto a considerar a la hora de escoger un método (y posiblemente una librería que lo implemente), es el tamaño del problema a resolver y la posible dispersión de la matriz que lo define. La dimensión de la matriz A puede determinar la utilización de una librería con un enfoque secuencial o paralelo. Con respecto a la dispersión, hay una división bastante clara entre los métodos que se suelen utilizar, dependiendo de si la matriz asociada es densa o dispersa (esta última normalmente asociada a problemas de gran tamaño). Los algoritmos utilizados en este último caso son algoritmos que buscan no modificar la matriz inicial con el fin de no perder la dispersión, y están basados, principalmente, en operaciones como la multiplicación matriz-vector que son fácilmente

paralelizables. Además, estos métodos permiten el cálculo de unos pocos autovalores situados en una determinada región del plano complejo. Por el contrario, los algoritmos más utilizados para matrices densas se basan en transformaciones de semejanza unitarias y descomposiciones matriciales ortogonales, y en la mayoría de casos, estos métodos calculan el espectro completo de la matriz.

Otro aspecto a tener en cuenta a la hora de abordar el problema es la posible simetría de la matriz, o si ésta es (semi)definida positiva o no. Para estos casos hay métodos específicos que sacan partido de estas características especiales.

A la hora de escoger el método de resolución, es importante determinar qué parte del espectro se desea calcular y si se requieren o no los vectores propios asociados, ya que hay métodos que calculan eficientemente todos los valores propios del problema, mientras que otros tienen la posibilidad de calcular sólo una parte del espectro. En determinados problemas físicos que se modelan mediante un problema de valores propios, sólo unos cuantos valores y vectores propios tienen correspondencia con magnitudes reales, y no hay interés en el cálculo del espectro completo. En problemas de gran dimensión tiene especial importancia ajustar, tanto como sea posible, la cantidad de valores a calcular, para reducir por un lado el tiempo de computación necesario, y por otro la cantidad de memoria requerida para almacenar los vectores calculados. Aunque en estos problemas la matriz asociada es dispersa, los vectores propios, en general, serán densos y su almacenamiento puede suponer un problema.

En la práctica la mayoría de los problemas reales de gran dimensión que se presentan, tan sólo requieren el cálculo de una mínima parte del espectro de la matriz que define el problema, y dicha parte debe ser convenientemente especificada para seleccionar la técnica más adecuada. Los valores de interés podrían ser, por ejemplo, los mayores o menores (en módulo), los más cercanos a un valor dado μ, o en el caso de tratarse de un problema hermitiano definido, todos los comprendidos en un determinado intervalo.

En definitiva, en el ámbito del cálculo de valores propios, surgen distintos tipos de problemas para los que se diseñan distintas soluciones que buscan aumentar la eficiencia aprovechando las características del problema concreto. En particular, este proyecto se centra en el problema generalizado simétrico definido de valores propios, es decir, en resolver la ecuación (2) para matrices A y B que son simétricas, B es semidefinida positiva y existe una combinación lineal de ambas que es definida positiva (el par (A,B) es simétrico definido). En este tipo de problemas, como se comprobará en la sección §1.1, todos los valores propios son reales por lo que podría cobrar sentido obtener todos los autovalores comprendidos en un determinado intervalo.

En este trabajo se plantean dos objetivos principales. En primer lugar, se pretende diseñar un método para la resolución, en entornos paralelos de memoria distribuida, del problema generalizado simétrico definido, de forma que permita calcular todos los autovalores solución del problema que estén contenidos en un determinado intervalo, así como un sistema de autovectores del subespacio invariante asociado a estos autovalores. El segundo objetivo de este trabajo es realizar una implementación paralela del método diseñado, para ser incluida en la librería *SLEPc* (*Scalable Library for Eigenvalue Problem Computations*). Esta librería, de código abierto, está especializada en la resolución, en sistema de memoria

distribuida, de problemas de valores propios de gran tamaño.

La implementación del método diseñado debe cumplir con dos características fundamentales:

1. Eficiencia en entornos paralelos de memoria distribuida.

2. Robustez y flexibilidad para ser utilizado en situaciones generales sin restricciones asociadas a aplicaciones concretas.

Durante la elaboración del proyecto se han estudiado varios trabajos relacionados, entre los que cabe destacar [8], en el que se hace un extenso estudio de un problema similar al planteado en este trabajo (para entornos secuenciales). La solución propuesta por [8] utiliza una técnica conocida como *spectrum slicing* (disección del espectro), en la que los autovalores se calculan por grupos, utilizando una secuencia de puntos (*shifts*), en el interior del intervalo, en torno a los cuales se obtienen los autovalores. Este enfoque se ha considerado adecuado para nuestro método ya que no impone limitaciones en el número de autovalores que se pueden calcular (aparte del espacio de almacenamiento), al contrario de lo que ocurre con otras técnicas en las que el número de autovalores que se puede obtener se ve limitado por el aumento del coste a medida que lo hace el número de valores calculados. El trabajo [8] se escoge como punto de partida, y es revisado en profundidad proponiendo tanto actualizaciones que incorporen nuevas técnicas existentes en los métodos de valores propios, como modificaciones con las que obtener mayor rendimiento en el entorno paralelo para el que se diseña. Otros trabajos que se han considerado en este estudio son [7], estudio anterior a [8], donde ya se utiliza la técnica de disección del espectro y algunas de las ideas básicas que aparecen en [8]. Un tercer trabajo revisado es [23], que presenta un diseño paralelo y busca solución para una aplicación concreta basándose en el trabajo realizado por [8]. Finalmente se ha considerado el trabajo [16], donde se propone un método basado en una adaptación del descrito en [18].

Estructura de la memoria

Esta memoria se estructura de la forma siguiente: En el primer capítulo se dan algunos conceptos teóricos relativos a valores propios y se presenta una visión general de los principales métodos de resolución, haciendo especial hincapié en las características de los métodos más adecuados para el caso que nos interesa. En el capítulo 2 se tratan todos los aspectos relativos al diseño del método descrito en la memoria. En primer lugar se hace una descripción y valoración de las técnicas utilizadas en los trabajos indicados anteriormente y que, de alguna manera, han sido tenidos en cuenta en nuestro estudio. En segundo lugar se especifica el entorno paralelo en el que estará encuadrado nuestro método, así como las implicaciones que ésto conlleva para el diseño del mismo. En tercer lugar, se describen los elementos principales del enfoque que presentamos, se detallan cuáles de las ideas estudiadas en otros trabajos han sido incluidas, y qué alternativas se han propuesto para dar solución a los problemas planteados. En definitiva, se justifican las decisiones de diseño tomadas y se proponen diversas opciones con el fin de comprobar cuáles de ellas

son más adecuadas para nuestro problema concreto. Finalmente, se dan detalles del algoritmo que describe el método diseñado. En el capítulo 3 se tratan temas relativos a la implementación del método. En primer lugar, se hace una revisión del software de valores propios utilizado en la actualidad. En segundo lugar, y para situar el marco sobre el que se realiza la implementación, se describen las librerías SLEPc, y PETSc (*Portable, Extensible Toolkit for Scientific Computation*), especializadas en problemas algebraicos para matrices dispersas de gran tamaño, diseñadas para ser utilizadas en entornos paralelos de memoria distribuida. Finalmente y como parte fundamental de este capítulo, se tratan detalles concretos relativos a la implementación del método realizada. En el capítulo 4, se presentan dos grupos de resultados numéricos. En primer lugar, las medidas de los tiempos obtenidos con las distintas opciones implementadas, con el fin de decidir cuáles de ellas son las más favorables y deben por tanto ser incluidas en el código final. En segundo lugar, se exponen los resultados de la evaluación de las prestaciones y del comportamiento paralelo de la implementación realizada. También se incluyen en este capítulo, a modo ilustrativo, los tiempos de ejecución de un problema concreto de cierta complejidad. Finalmente, en el capítulo 5, se presentan las conclusiones extraídas a lo largo de la elaboración del proyecto y se plantea una línea futura de ampliación.

Capítulo 1

Consideraciones teóricas previas

En este capítulo se muestran resultados teóricos relativos al concepto de valor propio y a los métodos de cálculo de autovalores, que se consideran importantes para la comprensión de las características especiales del problema que se quiere resolver, y de los métodos utilizados para ello.

En la exposición de las siguientes secciones, se supone que se trabaja con matrices reales o complejas. Cuando no se quiere hacer una referencia especial al espacio vectorial en el que se trabaja, se utiliza el símbolo $\mathbb{K}$ para representar tanto el cuerpo de los números complejos, como el de los reales.

1.1. Valores propios

En esta sección se da una definición formal de los conceptos de valor y vector propio, y se exponen algunas propiedades relevantes de los mismos.

1.1.1. Definición. Primeras propiedades

Definición 1. Dada una matriz $A \in \mathbb{K}^{n\times n}$, un escalar $\lambda \in \mathbb{C}$ se dice *valor propio* (o *autovalor*) de A, si $\exists x \in \mathbb{C}^n \backslash \{0\}$ tal que,

$$Ax = \lambda x. \tag{1.1}$$

Todo vector no nulo, x, verificando (1.1) se denomina *vector propio* o *autovector* (por la derecha) de A asociado a λ. El conjunto de autovalores de A se llama *espectro* de A y se denota $\Lambda(A)$.

Observación. $\lambda \in \mathbb{C}$ es un valor propio de $A \in \mathbb{C}^{n\times n}$ si y sólo si $(A - \lambda I)x = 0$ tiene solución no trivial, lo que se cumple sii la matriz $A - \lambda I$ es singular, o equivalentemente si $det(A - \lambda I) = 0$. Como consecuencia, los valores propios de A se corresponden con las raíces de su polinomio característico p_A

$$p_A(\lambda) := det(\lambda I - A) = (\lambda - \lambda_1)^{m_1}(\lambda - \lambda_2)^{m_2} \cdots (\lambda - \lambda_r)^{m_r}. \tag{1.2}$$

Las multiplicidades, m_i, como raíces del polinomio característico, reciben el nombre de *multiplicidades algebraicas* de los autovalores. Un valor propio con multiplicidad algebraica uno se dice *simple*, y *múltiple* en otro caso.

De la expresión (1.2) se siguen las siguientes consecuencias:

- Una matriz de dimensión n tiene $n = m_1 + \ldots + m_r$ valores propios (contando las multiplicidades).
- Matrices reales pueden tener autovalores complejos, aunque éstos siempre aparecen en pares conjugados (raíces complejas de un polinomio real). Por ejemplo, la matriz $\left(\begin{smallmatrix} 1 & -1 \\ 2 & -1 \end{smallmatrix}\right)$ tiene como autovalores $\lambda_1 = i$ y $\lambda_2 = -i$
- Cualquier polinomio mónico de grado n,

  $$p(\lambda) = c_0 + c_1\lambda + \ldots + c_{n-1}\lambda^{n-1} + \lambda^n$$

  puede ser visto como el polinomio característico de una matriz (*matriz companion*). El teorema de *Abel-Ruffini*, que afirma que si $n > 4$, en general, no es posible encontrar las soluciones de la ecuación $p(\lambda) = 0$ utilizando un número finito de operaciones aritméticas elementales, nos advierte de que los algoritmos para cálculo de valores propios no serán directos sino iterativos.

El polinomio característico no se utiliza como herramienta para el cálculo de valores propios. Para ello se necesitaría, por un lado, calcular los coeficientes del mismo, y por otro, utilizar algún método iterativo (por ejemplo métodos de Newton o Laguerre) para buscar sus raíces. Dado que el cálculo de dichos coeficientes es un problema mal condicionado, el polinomio característico sólo tiene utilidad teórica.

Definición 2. Sea $\langle .,. \rangle \colon \mathbb{K}^n \times \mathbb{K}^n \longrightarrow \mathbb{K}$ un producto escalar definido en $\mathbb{K}^n$ ($\mathbb{K} = \mathbb{C}$ o $\mathbb{R}$). Una matriz $A \in \mathbb{K}^{n\times n}$ se dice *autoadjunta* si $\forall x, y \in \mathbb{C}^n$ se cumple

$$\langle Ax, y\rangle = \langle x, Ay\rangle. \tag{1.3}$$

Observación. Las matrices reales *simétricas* ($A \in \mathbb{R}^{n\times n}$ con $A^T = A$) se corresponden con las matrices autoadjuntas para el producto escalar $\langle x, y\rangle = y^T x$, en $\mathbb{R}^n$, mientras que las matrices complejas *hermitianas* ($A \in \mathbb{C}^{n\times n}$ con $A^* = A$) se corresponden con las matrices autoadjuntas para el producto escalar $\langle x, y\rangle = y^* x$, en $\mathbb{C}^n$.

Teorema 1. *Si $A \in \mathbb{K}^{n\times n}$ es autoadjunta entonces todos sus autovalores son reales.*

Demostración. Sea λ un valor propio de A, entonces $\exists x \in \mathbb{C}^n\backslash\{0\}$, tal que $Ax = \lambda x$.

Para $x \in \mathbb{K}^n$, sea $\|x\| = \langle x, x\rangle$, la norma inducida por el producto escalar definido en $\mathbb{K}^n$. Podemos suponer que x, vector propio asociado a λ, tiene norma 1 (en otro caso trabajaríamos con $\frac{x}{\|x\|}$).

Se tiene

$$\lambda = \lambda\|x\| = \lambda\langle x, x\rangle = \langle \lambda x, x\rangle = \langle Ax, x\rangle = \langle x, Ax\rangle =$$
$$\overline{\langle Ax, x\rangle} = \overline{\langle \lambda x, x\rangle} = \overline{\lambda\langle x, x\rangle} = \overline{\lambda\|x\|} = \overline{\lambda}.$$

Observación. Se ha utilizado la linealidad por la izquierda y la hermiticidad del producto escalar. La última igualdad se sigue de que $\|x\| \neq 0$ siempre que $x \neq 0$.

□

Este teorema demuestra que si la matriz que define un problema de valores propios estándar es hermitiana (simétrica en el caso real), entonces los valores propios que resuelven el problema serán reales. Para el problema generalizado, veremos que, exigiendo la condición de que el par sea definido, se tiene un resultado similar.

1.1.2. Transformaciones de semejanza. Descomposiciones matriciales

Definición 3. Dos matrices $A, C \in \mathbb{K}^{n\times n}$ son *semejantes*, si existe una matriz regular $X \in \mathbb{K}^{n\times n}$, de forma que $A = XCX^{-1}$.

Teorema 2. *Sea $A, X \in \mathbb{K}^{n\times n}$, con X no singular, y $B = XAX^{-1}$ una matriz semejante a A. Entonces, A y B tienen los mismos valores propios, con la misma multiplicidad. Además, si $(\lambda, y) \in \mathbb{C} \times \mathbb{C}^n$ es un par propio de A entonces, el par (λ, Xy) lo es de B.*

Demostración. Para el primero de los resultados, veamos que las dos matrices tienen el mismo polinomio característico

$$\begin{aligned} det(\lambda I - B) &= det(X^{-1})det(\lambda I - B)det(X) = det(X^{-1}(\lambda I - B)X) \\ &= det(\lambda I - X^{-1}BX) = det(\lambda I - A) \end{aligned}$$

Por otro lado, si $(\lambda, y) \in \mathbb{C} \times \mathbb{C}^n$ es un par propio de A se tiene que $Ay = \lambda y$, de donde se sigue, $XAX^{-1}Xy = \lambda Xy$, y por tanto $BXy = \lambda Xy$. Así, $x = Xy$, es un vector propio de B. □

Definición 4. Una matriz de dice *diagonalizable* si es semejante a una matriz diagonal.

Observación. Por el Teorema 2, si una matriz es diagonalizable, sus valores propios coinciden con los elementos diagonales de la matriz diagonal a la que es semejante.

Proposición 3. *Una matriz $A \in \mathbb{C}^{n\times n}$, es diagonalizable sii existe una base de $\mathbb{C}^n$ formada con vectores propios de A.*

Demostración. Supongamos que A es diagonalizable, entonces $\exists X, D \in \mathbb{C}^{n\times n}$, con X regular y $D = (d_{ij})$ diagonal, tal que $X^{-1}AX = D$, de donde se sigue $AX = XD$. Sea x_j la j-ésima columna de X, para $j = 1, \ldots, n$ se tiene, $Ax_j = x_j d_{jj}$, es decir, x_j es un vector propio. Por otro lado, dado que X es regular, sus columnas forman una base de $\mathbb{C}^{n\times n}$.

El recíproco es inmediato tomando como X la matriz formada por los vectores propios.

□

No todas la matrices son diagonalizables, por lo que, en general, no siempre existe una forma diagonal semejante de donde obtener los autovalores. Por otro lado, errores en la matriz inicial pueden ser considerablemente amplificados si se realizan transformaciones de semejanza utilizando alguna matriz que aun siendo regular, podría estar cerca de ser

singular (número de condición elevado). Para evitar éste último problema, los algoritmos para cálculo de valores propios trabajan con matrices *unitarias* (*ortogonales* en el caso real). En la siguiente proposición se resumen algunas de las propiedades de las matrices unitarias.

Proposición 4. *Dada $U \in \mathbb{C}^{n\times n}$, equivalen:*

(a) $UU^* = U^*U = I$ *(Unitaria)*

(b) Las columnas de U forman una base ortonormal de $\mathbb{C}^n$

(c) U es una isometría (i.e. $\forall x \in \mathbb{C}^n$, $\|Ux\|_2 = \|x\|_2$)

Definición 5. Dos matrices son *unitariamente semejantes* si son semejantes y la matriz de la Definición 3 es unitaria.

Nota. Sea $\tilde{A} = A + E$ la perturbación de una matriz A, y U una matriz unitaria, entonces, al considerar la transformación de semejanza $\tilde{B} = U^*\tilde{A}U = U^*AU + U^*EU = B + F$, se tiene $\|F\| = \|E\|$. Es decir, las transformaciones de semejanza unitarias no amplifican los errores iniciales, motivo por el cual son utilizadas en los algoritmos de cálculo de valores propios.

Teorema 5 (Schur)**.** *Dada $A \in \mathbb{C}^{n\times n}$ existe una matriz unitaria U, de forma que $T := UAU^*$ es triangular superior.*

Nota. La matriz T del enunciado recibe el nombre de *forma de Schur* de la matriz A.

Demostración. En la demostración, se procede por inducción sobre el orden de A. El proceso se inicia contando con la existencia de un par propio arbitrario de A con el que reducir el problema a otro de una unidad menor. El método no es constructivo y no puede ser utilizado para obtener la forma de Schur de una matriz, hecho que confirma la observación anterior de que los algoritmos de cálculo de valores propios son iterativos. Por este motivo, y aunque se trata de una demostración sencilla, no consideramos necesario dar más detalle sobre ella. La demostración de este importante teorema puede encontrarse, por ejemplo, en [20]. □

Dado que siempre existe la forma de Schur de una matriz, que ésta implica una transformación de semejanza unitaria, y que los valores propios de una matriz triangular coinciden con los elementos de la diagonal (inmediato a partir del polinomio característico), la forma de Schur de una matriz podría ser una buena opción a la hora de calcular los valores propios. Como veremos en la sección §1.2, la forma de Schur de una matriz es la base de uno de los algoritmos más importantes para el cálculo de autovalores.

La forma de Schur de una matriz real, puede ser compleja (existen matrices reales con valores propios complejos). Sin embargo, es posible trabajando exclusivamente en aritmética real, y utilizando una transformación de semejanza ortogonal, obtener una matriz cuasitriangular, con bloques en la diagonal de tamaño máximo igual a dos (*teorema de Schur real*).

Proposición 6. *Si $A \in \mathbb{C}^{n\times n}$ es hermitiana (ortogonal en el caso real) entonces es unitariamente (ortogonalmente) diagonalizable.*

Demostración. Sea $T = U^*AU$, una forma de Schur de A, entonces

$$T^* = (U^*AU)^* = U^*A^*U = T,$$

de donde se sigue que T es diagonal y los vectores de Schur (columnas de U) son vectores propios de A. □

Observación. Hemos visto que toda matriz hermitiana de orden n tiene n valores propios reales y un sistema completo de vectores propios ortogonales (que coinciden con los vectores de Schur).

1.1.3. Transformación espectral

Cuando no hay interés en calcular el espectro completo de una matriz, tal como se verá en la sección §1.2, existen métodos que permiten el cálculo de unos pocos autovalores, típicamente los de mayor módulo. Si los valores deseados son otros, es posible utilizar técnicas de transformación del espectro con las que modificar el problema original de forma que los valores proporcionados por el método se correspondan con los buscados.

Proposición 7. *Dada $A \in \mathbb{C}^{n\times n}$ y $\sigma \in \mathbb{C}$. Si (λ, x) es un par propio de A, entonces*

(a) $(\lambda + \sigma, x)$ es un par propio de la matriz $A + \sigma I$.

Nota. Esta propiedad es un caso particular de una más general que dice que dado un polinomio p, $(p(\lambda), x)$ es par propio de la matriz $p(A)$.

(b) Si A es invertible, $(\frac{1}{\lambda}, x)$ es un par propio de A^{-1}.

(c) Si $\sigma \notin \Lambda(A)$, $(\frac{1}{\lambda-\sigma}, x)$ es un par propio de $(A - \sigma I)^{-1}$

Demostración. (a) Trivial comprobando $(A + \sigma I)x = (\lambda + \sigma)x$.

(b) Análogo al anterior.

(c) Consecuencia inmediata de (a) y (b).

□

Como consecuencia de la Proposición 7, si se dispone de un método de cálculo de valores propios con el que obtener los autovalores de mayor módulo, $\{\theta_i\}_{i=1}^p$, de la matriz $(A - \sigma I)^{-1}$, entonces, el conjunto de valores $\{\lambda_i = \sigma + \frac{1}{\theta_i}\}_{i=1}^p$ se corresponderá con los valores propios de A más próximos a σ. Esta transformación espectral, conocida como *shift and invert* (traslación e inversión), será utilizada en nuestro método para calcular grupos de autovalores en torno a desplazamientos seleccionados.

A continuación introducimos el concepto de *inercia* y la conocida *ley de la inercia de Sylvester*, que nos proporcionará una herramienta con la que validar valores obtenidos.

Definición 6. Dada una matriz hermitiana $A \in \mathbb{C}^{n\times n}$, se llama *inercia* de A, a la terna (n, p, c) donde n, p y $c \in \mathbb{N}$ son, respectivamente, el número de valores propios negativos, positivos e iguales a cero.

Nota. En lo sucesivo, nos referiremos a la inercia, como al número de valores propios menores que cero.

La inercia se utiliza para conocer el número de valores propios, de una matriz hermitiana, en el interior de un intervalo dado, $[\alpha, \beta]$. La cantidad de autovalores de A (hermitiana) en $[\alpha, \beta]$ puede obtenerse como la diferencia de la inercias de las matrices $(A - \beta I)$ y $(A - \alpha I)$.

Lema 8 (ley de la inercia de Sylvester)**.** *Establece que la inercia de una matriz es invariante bajo transformaciones de congruencia. Es decir, dada $A \in \mathbb{C}^{n\times n}$, hermitiana y $X \in \mathbb{C}^{n\times n}$, regular, entonces $B = X^*AX$ que es hermitiana y tiene la misma inercia que A.*

Nota. Toda matriz hermitiana S, se puede descomponer como $S = LDL^*$, con L triangular superior unidad y D diagonal. En caso de ser S indefinida el método simple que produce tal descomposición es inestable por lo que se suele utilizar el propuesto por *Bunch y Kaufman* en el que la matriz D que se obtiene es diagonal a bloques, con tamaño máximo de bloque igual a dos. Tal descomposición determina una congruencia entre las matrices S y D que hace posible conocer la inercia de S a partir de la de D.

1.1.4. Problema generalizado

Definición 7. Dadas $A, B \in \mathbb{K}^{n\times n}$, $\lambda \in \mathbb{C}$ es un valor *propio generalizado* del par (A, B) si $\exists x \in \mathbb{C}^n \backslash \{0\}$ (vector propio), de forma que

$$Ax = \lambda Bx. \tag{1.4}$$

Nota. Dado que en los procedimientos utilizados en el diseño de nuestro método, se trata el problema generalizado reduciéndolo a uno estándar, no nos detendremos en detallar métodos específicos para problemas generalizados. Comentamos, a continuación, algunas diferencias entre el problema estándar y el generalizado, que se consideran necesarias para entender algunos detalles de los métodos de resolución que se utilizarán.

1. Si $\exists z \in \mathcal{N}(A) \cap \mathcal{N}(B)\backslash\{0\}$, donde $\mathcal{N}(A) = \{x \in \mathbb{C}^n : Ax = 0\}$ representa el espacio nulo de la aplicación lineal definida por A, se tiene que $\forall\lambda \in \mathbb{C}$, $Az = 0 = \lambda Bz$ por lo que el par (λ, z) es un par propio de (A, B). Un par (A, B) que verifica $\mathcal{N}(A)\cap\mathcal{N}(B) = \{0\}$ se dice *regular*. En lo sucesivo, los pares de matrices considerados se asumen regulares.

2. Dos pares de matrices, (A_1, B_1) y (A_2, B_2) se dicen *equivalentes* si existen matrices regulares, W y V, de forma que

$$A_2 = WA_1V \quad \text{y} \quad B_2 = WB_1V.$$

Se tiene para pares de matrices equivalentes resultados análogos a los que se tiene en el problema estándar para las matrices semejantes, es decir, los pares equivalentes, (A_1, B_1) y (A_2, B_2), tienen los mismos valores propios, y los vectores propios (derechos) de (A_2, B_2) son los de (A_1, B_1) multiplicados por V^{-1}. Un par se dice *diagonalizable* si es equivalente a un par diagonal (las dos matrices que lo definen son diagonales).

3. Un par, (A, B) es *definido*, si las matrices que lo definen son hermitianas (simétricas) y existe una combinación lineal de ambas que es definida positiva. Este concepto generaliza el de matriz hermitiana en el problema estándar, en el sentido de que todo par definido es diagonalizable y sus valores propios son todos reales. En este caso, se tiene además que es posible obtener un sistema completo de autovectores, X, de modo que $X^*BX = I$. Estos resultados se demuestran en el Teorema 10 y la Proposición 11.

4. Si (λ, x) es un par propio de (A, B), entonces $(\frac{1}{\lambda}, x)$ lo es del par (B, A). Los vectores de $\mathcal{N}(A)$ se corresponden con vectores propios asociados al valor propio $\lambda = 0$, mientras que los vectores de $\mathcal{N}(B)$ se corresponden con vectores propios asociados a $\lambda = \infty$. Así, un problema generalizado puede tener valores propios infinitos.

5. Existe para pares de matrices una adaptación del teorema de Schur visto para el problema estándar. Dado que no se hace uso de él, nos limitamos a enunciarlo, como ejemplo del paralelismo que en muchas ocasiones podemos encontrar entre los problemas estándar y generalizado de valores propios.

   Teorema 9 (Forma de Schur Generalizada)**.** *Dado un par regular (A, B), existen matrices unitarias, U y V, tales que U^*AV y U^*BV son triangulares superiores.*

En determinados casos, el problema generalizado se puede resolver reduciéndolo a otro estándar equivalente. Por ejemplo, si B es regular, la ecuación $B^{-1}Ax = \lambda x$ es equivalente a (1.4). Esta reducción a un problema estándar puede no ser conveniente si B está mal condicionada. Si A es regular, pueden intercambiarse los papeles de A y B, y resolver el problema $A^{-1}Bx = \frac{1}{\lambda}x$.

Para un problema generalizado simétrico con B definida positiva, sea $B = CC^*$ la descomposición de *Cholesky* de B, entonces, de (1.4) se tiene

$$(C^{-1}AC^{-*})(C^*x) = \lambda(C^*x). \tag{1.5}$$

De este modo, si (λ, z) es un par propio de $Sz = \lambda z$ con $S = C^{-1}AC^{-*}$, entonces $(\lambda, C^{-*}z)$ es un par propio del problema original.

Centrándonos en el problema que pretendemos resolver (generalizado simétrico definido), y dado que estamos interesados en calcular valores en el interior del espectro, será necesario hacer uso de alguna transformación espectral de forma que los valores deseados se correspondan con los valores extremos del problema modificado. En el trabajo realizado

en [7] se utiliza la transformación de traslación e inversión (*shift-invert*) sobre el problema (1.5). Trabajos posteriores, como [17], proponen utilizar el problema transformado

$$(A - \sigma B)^{-1} B x = \theta x. \tag{1.6}$$

Para ver cómo se transforma el espectro en el nuevo problema, restamos σBx en la ecuación (1.4) y obtenemos $(A - \sigma B)x = (\lambda - \sigma)Bx$, de donde

$$(A - \sigma B)^{-1} B x = \frac{1}{\lambda - \sigma} x.$$

Es decir, si (λ, x) es un par propio de (1.4) entonces $(\frac{1}{\lambda - \sigma}, x)$ lo es de (1.6). Los autovalores de (1.4) más próximos a σ, se transforman en los de mayor módulo de (1.6). Esta transformación permite reducir el problema generalizado original a otro estándar que tiene los mismos vectores propios, y los valores propios se relacionan con los del problema original mediante una sencilla expresión aritmética. La reducción (1.6) será la que utilizaremos para calcular grupos de valores propios en torno a *shifts* seleccionados con los que completar el intervalo de interés. En este enfoque *multishift*, como ya se ha introducido con anterioridad, es muy útil poder conocer el número de autovalores entre dos desplazamientos dados (σ_1 y σ_2). En la sección §1.1.3 se vio que para el problema estándar de valores propios, la inercia permitía obtener esta información. Veamos que en el problema generalizado definido simétrico tenemos también dicha información a partir de la inercia de la matriz $(A - \sigma B)$, que coincide con el número de autovalores del problema original, menores que σ.

En efecto, si B es definida positiva el problema (1.5) tiene los mismos valores propios que el problema original. Por otro lado, la matriz que lo define, $S = C^{-1} A C^{-*}$ es hermitiana, por lo que el número de valores menores que σ se puede obtener a partir de la inercia de la matriz $(S - \sigma I)$, que es congruente con la matriz $C(S - \sigma I)C^* = A - \sigma B$.

Si B es semidefinida, el problema original tiene autovalores infinitos, pero en cualquier caso, podemos obtener el número de autovalores en el interior de un intervalo $[\sigma_1, \sigma_2]$, como la diferencia entre las inercias de $(A - \sigma_2 B)$ y $(A - \sigma_1 B)$.

Se hace notar que aunque las matrices A y B sean simétricas, el problema transformado (1.6) no tiene por qué serlo. Sin embargo, como se comprueba en la demostración del siguiente teorema, es posible escoger el producto escalar utilizado de forma que la matriz $(A - \sigma B)^{-1} B$ sea autoadjunta. Si, por ejemplo, B es definida positiva , se comprueba que la matriz que define el problema (1.6) es autoadjunta considerando el producto escalar $\langle x, y \rangle = y^* B x$.

Teorema 10. *Si $A, B \in \mathbb{C}^{n \times n}$ definen un problema generalizado simétrico definido, entonces*

(a) Los valores propios que resuelven el problema son reales.

(b) El par (A, B) tiene un sistema completo de valores propios.

Demostración. Para demostrar *(a)*, veamos, de forma equivalente, que son reales las soluciones del problema estándar $(A-\sigma B)^{-1}Bx = \theta x$ con $\sigma \notin \Lambda(A,B)$.

Dado que el par (A,B) es definido, sabemos que existe una combinación lineal de A y B que es definida positiva. Sea $H = \alpha A + \beta B$ dicha combinación lineal. Por ser H simétrica (combinación lineal de matrices simétricas) y definida positiva, la función dada por $\langle x,y\rangle = y^*Hy$ para $x,y \in \mathbb{C}^n$, define un producto escalar. En efecto, se comprueba de forma inmediata que la función definida es lineal por la izquierda ($\langle x+z,y\rangle = \langle x,y\rangle + \langle z,y\rangle$), hermitiana ($\langle x,y\rangle = \overline{\langle y,x\rangle}\ \forall x,y \in \mathbb{C}^n$) y definida positiva ($\langle x,x\rangle > 0\ \forall x \in \mathbb{C}^n \backslash 0$).

La matriz que define el problema de valores propios, $S = (A-\sigma B)^{-1}B$, es autoadjunta respecto al producto escalar definido. Escribiendo

$$H = \alpha A + \beta B = \alpha(A-\sigma B) + (\alpha\sigma + \beta)B$$

se tiene

$$\begin{aligned}(\alpha A + \beta B)(A-\sigma B)^{-1}B &= \alpha B + (\alpha\sigma+\beta)B(A-\sigma B)^{-1}B \\ &= B(A-\sigma B)^{-1}(\alpha(A-\sigma B) + (\alpha\sigma+\beta)B) \\ &= B(A-\sigma B)^{-1}(\alpha A + \beta B).\end{aligned}$$

Así, $\forall x,y \in \mathbb{C}^n$ se cumple

$$\begin{aligned}\langle x, Sy\rangle &= (Sy)^*Hx = y^*B^*((A-\sigma B)^{-1})^*Hx \\ &= y^*B(A-\sigma B)^{-1}(\alpha A+\beta B)x \\ &= y^*(\alpha A+\beta B)(A-\sigma B)^{-1}Bx = y^*HSx \\ &= \langle Sx, y\rangle.\end{aligned}$$

En virtud del Teorema 1 podemos concluir que los valores propios de S son reales.

Por otro lado, para demostrar *(b)*, considerando H, que es hermitiana definida positiva, es posible calcular los pares propios de (A,H) utilizando la reducción a problema estándar (1.5). Por tanto, el par (A,H) tiene un sistema completo de vectores propios. El resultado es consecuencia inmediata de que un vector x es autovector de (A,H) sii lo es del par (A,B).

□

Proposición 11. *Si (λ_1,x_1), (λ_2,x_2) son pares propios del problema generalizado simétrico $Ax = \lambda Bx$, y $\lambda_1 \neq \lambda_2$ entonces $x_1^*Bx_2 = 0$.*

Demostración. Inmediato observando que

$$x_1^*Bx_2 = \frac{1}{\lambda_2}x_1^*Ax_2 = \frac{1}{\lambda_2}(Ax_1)^*x_2 = \frac{\lambda_1}{\lambda_2}(Bx_1)^*x_2 = \frac{\lambda_1}{\lambda_2}x_1^*Bx_2.$$

Por ser $\lambda_1 \neq \lambda_2$ se sigue que $x_1^*Bx_2 = 0$.

□

Observación. Para el problema generalizado simétrico definido, existe un sistema completo de autovectores X, de forma que $X^*BX = I$. Si B es definida positiva, podemos encontrar una base de vectores propios B-ortogonales (considerando el producto escalar $\langle x, y\rangle = y^*Bx$).

Cuando se considera el problema generalizado simétrico definido de valores propios, el Teorema 10 nos dice que todos los valores propios que lo resuelven son reales, por lo que tiene sentido plantear la búsqueda de solución restringiéndose a un intervalo, $[\alpha, \beta]$. Por otro lado, hemos visto que, para este problema, las matrices $(A - \beta B)$ y $(A - \alpha B)$ son simétricas y es posible conocer el número, m, de autovalores contenidos en el dicho intervalo restando las inercias de dichas matrices. Finalmente, la Proposición 11 nos dice que es posible encontrar un conjunto de autovectores, $\{x_i\}_{i=1}^m$, asociados a los autovalores contenidos en el intervalo, $(A - \beta B)$ y $(A - \alpha B)$, de modo que $X^*BX = I_m$ (X matriz cuyas columnas son los vectores x_i).

1.2. Visión general de métodos de cálculo de valores propios

En esta sección se hace una breve descripción de los principales métodos existentes para el cálculo de valores propios. No se pretende hacer una revisión exhaustiva de todos ellos, sino que se incluyen tan sólo aquellos que, de alguna forma, son básicos para la comprensión tanto del problema que se pretende resolver, como de los propios métodos utilizados. En primer lugar, se describe un método básico, en el que ya se puede observar la importancia que tienen para el cálculo de valores propios las secuencias del tipo $\{x_0, Ax_0, A^2x_0 \ldots\}$, siendo x_0 un determinado vector de inicio. Estas secuencias cobrarán relevancia en la sección §1.2.3 al hablar de los espacios de Krylov. En dicha sección se describe el método seleccionado para el cálculo de valores propios en torno a los desplazamientos que genera nuestro algoritmo, el método *thick-restart Lanczos*. Este método precisa la resolución de problemas de valores propios de reducido tamaño, para lo que se utiliza el método *Iterativo QR*, que se describe en la sección §1.2.2.

1.2.1. Método de la potencia. Iteración inversa

Dada una matriz A, un método muy simple que permite aproximar un único vector propio de la matriz, es el *método de la potencia* (ver algoritmo 1). En éste, multiplicando repetidas veces un vector inicial, por la matriz A, se forma una secuencia de vectores, $\{x_0, Ax_0, A^2x_0, \ldots\}$, que bajo determinadas condiciones, converge a un autovector correspondiente al valor propio dominante (de mayor módulo).

En caso de haber convergencia, los vectores que se generan van aproximando la dirección del vector dominante. La normalización que se introduce en el paso 3 del algoritmo 1 es necesaria para evitar un posible overflow (si $\|A\| > 1$) o underflow (si $\|A\| < 1$) en el cómputo. Para aproximar el autovalor dominante que se proporciona en el paso 7 del algoritmo, se utiliza el *cociente de Rayleigh* del vector x,

$$\frac{x^*Ax}{x^*x}. \tag{1.7}$$

Algoritmo 1 Iteración de la potencia

Entrada: Matriz A, vector inicial x_0

Salida: Aproximación, $(\tilde{\lambda}, \tilde{x})$, del par propio dominante

1: **repetir**
2: $y_k = Ax_{k-1}$
3: $x_k = \frac{y_k}{\|y_k\|}$
4: $\lambda_k = x_k^* A x_k$
5: $k = k + 1$
6: **hasta** $\|Ax_k - \lambda_k x_k\|$ pequeño **o** $k > maxit$
7: **retornar** $\tilde{x} = x_k$ y $\tilde{\lambda} = \lambda_k$

El cociente de Rayleigh proporciona la mejor aproximación del autovalor correspondiente a x, es decir, es el valor μ que minimiza $\|Ax - \mu x\|$, y puede ser obtenido como solución del problema de mínimos cuadrados de dimensión $n \times 1$, $x\lambda \cong Ax$, cuyas ecuaciones normales son, en el caso real, $x^T x\lambda = x^T Ax$.

Para que el método converja es necesario que la matriz tenga un valor propio dominante, $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. La idea de por qué el método converge a un autovector asociado a λ_1, se ve muy facilmente para el caso en que A es diagonalizable. En este caso, existe una base de $\mathbb{C}^n$, $\{v_i\}_{i=1}^n$, formada con vectores propios de A. Considerando la expresión del vector inicial como combinación lineal de los vectores de dicha base, se tiene

$$\begin{aligned} x_0 &= \sum_{i=1}^{n} \alpha_i v_i, \\ x_k &= Ax_{k-1} = A^2 x_{k-2} = \cdots = A^k x_0, \\ x_k &= \sum_{i=1}^{n} \lambda_i^k \alpha_i v_i = \lambda_1^k \left(\alpha_1 v_1 + \sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^k \alpha_i v_i \right). \end{aligned} \tag{1.8}$$

Dado que $|\frac{\lambda_i}{\lambda_1}| < 1$; $\forall\ i > 1$, sucesivas potencias hacen que cualquier componente no asociada a u_1 tienda a cero. La velocidad de convergencia del método viene determinada por el mayor cociente $\frac{\lambda_2}{\lambda_1}$.

Hay varios factores que pueden afectar a la convergencia del método descrito:

- Cuando el vector inicial no tiene componente en el autovector dominante, al menos teóricamente, el método descrito no puede converger a dicho vector. Aún en este caso, en algunas ocasiones, los errores de redondeo pueden hacer que aparezcan valores no nulos en dicha componente y que la convergencia sea posible.
- En caso de haber más de un autovalor con el mismo módulo (por ejemplo $\lambda = 1$ y $\lambda = -1$), el método puede converger a una combinación lineal de los autovectores correspondientes.
- Si A es real con valores propios complejos, y el vector inicial se elige también real, la iteración de la potencia nunca convergerá a un vector complejo.

El método de la potencia tiene interés por ser, de algún modo, la base del resto de métodos que se describen. La lenta convergencia de este método hace que no se utilice para el cálculo de autovalores en la forma descrita. Hay sin embargo, otros métodos derivados directamente de éste, y que se describen a continuación, que sí tienen un uso más real.

La velocidad de convergencia en el método de la potencia puede ser mejorado utilizando una transformación espectral (ver sección §1.1.3) de modo que $|\frac{\hat{\lambda}_2}{\hat{\lambda}_1}| \ll |\frac{\lambda_2}{\lambda_1}|$, donde $\hat{\lambda}$ representa los autovalores del problema transformado. Por otro lado, al utilizar el método de la potencia junto con la transformación espectral de desplazamiento e inversión, $(A-\sigma I)^{-1}$, es posible aproximar el autovalor más cercano a un punto dado, σ, ya que dicho valor propio se corresponde con el de mayor módulo del problema transformado. La velocidad de convergencia será mayor cuanto más cercano esté el desplazamiento σ del autovalor que se quiere aproximar. A la hora de aplicar el método de la potencia a la matriz $(A-\sigma I)^{-1}$, ésta no se calcula explícitamente sino que se sustituye la multiplicación del paso 2 del algoritmo 1 por la resolución del sistema $(A-\sigma I)x_k = x_{k-1}$. Dicho paso requiere la factorización (una única vez) de la matriz $(A-\sigma I)$. Cuando el método de la potencia se aplica a la matriz A^{-1} (o $(A-\sigma I)^{-1}$), para obtener el valor propio de menor módulo (o más cercano a σ), el método se conoce como *Iteración inversa*.

Dado que el método de la potencia utiliza principalmente la operación de multiplicación matiz-vector, si se dispone de una implementación eficiente de dicha operación, este método será facilmente paralelizable. El método de la iteración inversa, sin embargo, necesita una factorización inicial y la resolución de un sistema triangular en cada iteración. Esto hace que este último método obtenga menor eficiencia en paralelo que el primero.

El método conocido como *Iteración del cociente de Rayleigh*, es una modificación del método de la iteración inversa, que consigue acelerar la convergencia, introduciendo en cada iteración un desplazamiento adecuado. El desplazamiento que utiliza es el cociente de Rayleigh de la última aproximación obtenida del autovector que busca. El coste por iteración de este método es alto, ya que utiliza en cada iteración un desplazamiento diferente y por tanto una factorización necesaria para resolver el sistema de ecuaciones asociado.

Deflación

Los métodos descritos en el apartado anterior permiten aproximar un único vector propio. Una vez obtenido un vector propio, con la técnica de la deflación es posible eliminar del cálculo las componentes en dicho vector, de forma que el método converja a un nuevo vector.

Suponiendo que ha sido calculado un par propio (λ_1, x_1) de A, considerando una matriz de Householder de forma que $Hx_1 = \alpha e_1$ (con $\alpha = 1$ si $\|x_1\| = 1$), y siendo e_1 el primer vector de la base canónica, se tiene que

$$HAH^{-1} = \begin{bmatrix} \lambda_1 & b^T \\ 0 & B \end{bmatrix},$$

donde la matriz B tiene los mismos valores propios que A salvo λ_1. Esta transformación permite trabajar con B para calcular λ_2.

Otro enfoque alternativo, más próximo al que se emplea en los métodos de cálculo de valores propios que utilizaremos en nuestro trabajo, parte de un vector w_1, de forma que $x_1^* w_1 = \lambda_1$. En caso de ser A hermitiana, tomaríamos $w = \lambda_1 x_1$ (considerando $\|x_1\| = 1$), y en general, w_1 puede escogerse como un autovector izquierdo asociado a λ_1 (convenientemente normalizado). De esta forma, la matriz $(A - w_1 x_1^*)$ tendrá como autovalores $0, \lambda_2, \ldots, \lambda_n$, con lo que λ_2 pasará a ser dominante.

Suponiendo que A es hermitiana, cuando en el método de la potencia aplicamos la matriz $(A - \lambda_1 x_1 x_1^*)$ a un vector v, se tiene

$$(A - \lambda_1 x_1 x_1^*)v = Av - x_1(Ax_1)^* v = Av - x_1 x_1^*(Av) = (I - x_1 x_1^*)Av.$$

Es decir, para que el método converja al vector propio asociado a λ_2, se elimina de Av la componente en x_1 y se trabaja en el complemento ortogonal de x_1. Esta idea aparecerá también en el método de Lanczos con reinicio (ver sección §1.3.2), que es el que utilizaremos en nuestro método, para el cálculo de valores en torno a desplazamientos seleccionados.

Iteración simultanea

El método de la *iteración simultánea* generaliza el método de la potencia, y permite calcular varios pares propios aplicando el método de la potencia simultaneamente, a un conjunto, X_0 de vectores de inicio. Los vectores que se obtienen en cada iteración, $X_{k+1} = AX_k$, generan un subespacio que converge al subespacio invariante asociado a los autovalores de mayor módulo. Al igual que en el método de la potencia, las columnas de X_k deben ser normalizadas para evitar overflow (o underflow). Por otro lado, el condicionamiento de la matriz X_k empeora a cada iteración, ya que las columnas van tomando progresivamente la dirección del autovector dominante. Estos problemas pueden solucionarse si a cada iteración se sustituye el sistema de vectores obtenido por otro que genere el mismo subespacio y que sea ortonormal. Así, se obtiene el algoritmo de la *iteración ortogonal* en el que, a cada iteración se sustituye el conjunto de vectores X_k por Q_k, obtenida en la descomposición QR de $X_k = Q_k R_k$.

1.2.2. Métodos para matrices densas. Iterativo QR

El problema que se plantea en este trabajo se centra en matrices dispersas de gran tamaño. Para resolverlo, se utilizan métodos de proyección (ver sección §1.2.3) para el cálculo de valores propios. Estos métodos buscan aproximaciones a la solución resolviendo un problema (proyectado) de dimensión reducida. Estos problemas pequeños se resuelven utilizando métodos específicos para matrices densas. La *Iteración QR* es uno de los métodos más generales e importantes para el cálculo de valores propios de matrices densas de tamaño medio, y es el que utilizamos cada vez que se necesita resolver un problema de dimensión reducida.

La iteración QR permite el cálculo de todos los valores propios de una matriz con coste $O(n^3)$, obteniendo la descomposición de Schur (ver teorema 5) de dicha matriz. A partir

de la matriz del problema, genera una sucesión de matrices unitariamente semejantes que convergen a una forma triangular.

Algoritmo 2 Iterativo QR con múltiple desplazamiento explícito

Entrada: matriz A

Salida: aproximación a la forma de Schur de A

1: $A_1 = H$ reducción de A a forma condensada
2: **para** $k = 1, 2, \ldots, maxit$ **hacer**
3: seleccionar desplazamientos $\{\sigma_i\}$
4: formar $P(A_k) = (A_k - \sigma_1 I)(A_k - \sigma_2 I)\cdots$
5: cálculo descomposición QR: $Q_k R_k = P(A)$
6: $A_{k+1} = Q_k^T A_k Q_k$
7: **si** elementos subdiagonales de A_{k+1} pequeños **entonces**
8: terminar retornando A_{k+1}
9: **fin si**
10: **fin para**

Para acelerar la convergencia del método, y al igual que se hacía en el método del cociente de Rayleigh, en cada iteración se trabaja con un problema desplazado utilizando un conjunto apropiado de desplazamientos. Hay varias estrategias a la hora de determinar dicho conjunto de desplazamientos.

Con el fin de reducir el coste por iteración del método (ver algoritmo 2), la matriz inicial se reduce a forma condensada (matriz Hessenberg o tridiagonal si hay simetría). De esta forma se reduce el coste de cada factorización QR (paso 5 del algoritmo 2). Además, se realiza un desplazamiento implícito con el que se consigue dicha factorización evitando formar explícitamente la matriz $P(A)$ (paso 4 del algoritmo 2).

Una de las principales limitaciones de la iteración QR es que ya en la primera fase, en la que se hace una reducción a forma condensada, se produce llenado de las matrices dispersas. Además, esta reducción no es eficiente en paralelo. Por otro lado, este método calcula el espectro completo, lo que es muy inusual en problemas de gran dimensión que trabajan con matrices dispersas.

Para la resolución del problema de valores propios simétrico existen, además, métodos específicos como por ejemplo, los métodos de *Bisección*, *Divide y vencerás* o *MRRR* (*múltiple representación relativamente robusta*). Algunos de ellos ofrecen la posibilidad de calcular sólo parte del espectro. Entre los métodos citados, destacamos el método de bisección, por la similitud existente entre el enfoque presentado en dicho método y el esquema de disección del espectro descrito en este trabajo. El método de bisección utiliza la inercia para obtener, para cada autovalor que se desea obtener, una secuencia de intervalos encajados de modo que el autovalor está contenido en todos ellos y el diámetro se va reduciendo hasta la tolerancia deseada. El método proporciona sólo autovalores, y si se desean también los vectores propios asociados, se recurre al método de la iteración inversa utilizando los autovalores calculados como *shift*. En caso de valores múltiples o clusters de autovalores, precisa reortogonalización para asegurar la ortogonalidad de los vectores que

proporciona.

1.2.3. Métodos para matrices dispersas

En esta sección se describen métodos adecuados para la resolución de problemas de valores propios de gran tamaño, en los que la matriz asociada es dispersa. En estos problemas, en los que la dimensión de la matriz puede ser del orden de millones, se requieren métodos que mantengan la dispersión de las matrices, para evitar posibles problemas de capacidad de almacenamiento. Por otro lado, dichos métodos deben también ser capaces de proporcionar sólo una parte del enorme espectro asociado a estos problemas.

Métodos de proyección. Subespacios de Krylov

Los métodos de proyección resuelven el problema de valores propios buscando aproximaciones a vectores propios en subespacios de dimensión reducida. Estos subespacios deben ser convenientemente seleccionados, para asegurar que contienen buenas aproximaciones de los autovectores asociados a los valores propios de interés.

Dada una matriz $A \in \mathbb{C}^{n\times n}$, y un subespacio $\mathcal{K}$, de dimensión m, un método de proyección ortogonal, busca una aproximación, $(\tilde{\lambda}, \tilde{x}) \in \mathbb{C}\times\mathcal{K}$, a un par propio, imponiendo la conocida como *condición de Galerkin*,

$$A\tilde{x} - \tilde{\lambda}\tilde{x} \perp \mathcal{K}, \tag{1.9}$$

o equivalentemente,

$$v^*(A\tilde{x} - \tilde{\lambda}\tilde{x}) = 0,\ \forall v \in \mathcal{K}. \tag{1.10}$$

El conocido como *procedimiento de Rayleigh-Ritz*, detallado en el algoritmo 3, proporciona un mecanismo para obtener las aproximaciones buscadas.

Algoritmo 3 Procedimiento de Rayleight-Ritz

1: obtener, $\{v_i\}_{i=1}^m$, base ortonormal de $\mathcal{K}$
2: calcular $B_m = V^*AV$ /*las columnas de V son los vectores $\{v_i\}$*/
3: obtener autovalores de B_m, y seleccionar los valores $\{\tilde{\lambda}_i\}_{i=1}^k$ de interés
4: calcular los autovectores, $\{y_i\}$, de B_m asociados a $\{\tilde{\lambda}_i\}_{i=1}^k$
5: retornar valores y vectores de Ritz, $\{(\tilde{\lambda}_i, \tilde{x}_i = Vy_i)\}_{i=1}^k$, como aproximaciones de los pares propios de A.

Los métodos de *Arnoldi* y *Lanczos*, detallados en las siguientes secciones, son métodos de proyección ortogonal sobre *espacios de Krylov*. Estos subespacios están generados por secuencias de vectores de la forma, $x, Ax, A^2x, \ldots$, siendo x un determinado vector. Escribiremos

$$\mathcal{K}_{k+1}(A, x) = span(x, Ax, A^2x, \ldots, A^kx), \tag{1.11}$$

para denotar el espacio de Krylov de orden $k+1$, generado a partir de x.

Tal como se vio en la descripción del método de la potencia (apartado §1.2.1), a partir de un vector inicial, x, este método genera una secuencia de vectores $x, Ax, A^2x, \ldots$, de

forma que en el paso k, el vector $A^k x$ aproxima la dirección del vector propio dominante. Esto nos da una idea de que los espacios de Krylov son ricos en direcciones de autovectores dominantes. El método de la potencia genera una secuencia que define un espacio de Krylov, pero a cada iteración se queda con la última dirección obtenida, descartando las previas, mientras que los métodos de proyección sobre espacios de Krylov, mantienen toda la información contenida en el subespacio generado, obteniendo mejores resultados en el cálculo de los autovalores.

Método de Arnoldi

La iteración de Arnoldi (ver algoritmo 4) construye una base ortonormal del espacio de Krylov.

Considerando

$$K_k = [x_0 \quad Ax_0 \quad \cdots \quad A^{k-1}x_0],$$

entonces $K_n^{-1} A K_n = C_n$ con C_n Hessenberg superior (*matriz companion*). Calculando la factorización QR de K_k, para tener una base mejor condicionada de $span(K_n)$, $Q_n R_n = K_n$, se tiene

$$Q_n^* A Q_n = R_n C_n R_n^{-1} =: H,$$

con H Hessenberg superior. Igualando la columna k-ésima de $AQ_n = Q_n H$, y despejando, se obtiene la recurrencia

$$h_{k+1,k} q_{k+1} = Aq_k - h_{1k} q_1 - \cdots - h_{kk} q_k. \tag{1.12}$$

utilizada en la iteración de Arnoldi.

Tras calcular cada uno de los vectores de la secuencia de Krylov (paso 3 del algoritmo 4), el proceso de Arnoldi utiliza el *procedimiento de Gram-Schmidt* para mantener la ortonormalidad en el conjunto de vectores que generan el subespacio de Krylov calculado (pasos 4-7 del algoritmo 4). Este proceso de ortogonalización es sensible a errores de redondeo, por lo que, normalmente, se opta por repetirlo dos veces para evitar la pérdida de ortogonalidad en los vectores que se obtienen a medida que el proceso avanza.

Si uno de los vectores generados en la iteración de Arnoldi es combinación lineal de los previos, tras el proceso de ortogonalización la norma del vector resultante (paso 8 del algoritmo 4) será cero. En este caso el proceso debe parar, ya que la secuencia de Krylov que se produce ya no podrá contener nuevos vectores linealmente independientes. Cuando esto ocurre los vectores calculados generan un subespacio invariante de A (subespacio generado por un conjunto de vectores propios de A).

En caso de tomar $k = n$, la iteración de Arnoldi produciría la reducción de A a forma Hessenberg superior, utilizando sólo operaciones de multiplicación matriz-vector y producto escalar de vectores.

El método de Arnoldi para el cálculo de valores propios combina, por un lado, el proceso de Arnoldi, con el que se construye una base ortonormal del espacio de Krylov, $\mathcal{K}_m(A, x)$, y por otro, el procedimiento de Rayleigh-Ritz para extraer, del subespacio creado, las aproximaciones de los pares propios buscados.

Algoritmo 4 Iteración de Arnoldi

Entrada: matriz $A \in \mathbb{K}^{n\times n}$, vector de inicio $v \in \mathbb{K}^n$, dimensión m
Salida: matrices Q tal que $Q^*Q = I_m$, y H Hessenberg superior, con $Q^*AQ = H$

1: $q_1 = \frac{v}{\|v\|_2}$
2: **para** $k = 1, 2, \ldots, m$ **hacer**
3: $u_k = Aq_k$
4: **para** $j = 1, \ldots, k$ **hacer**
5: $h_{jk} = q_j^* u_k$
6: $u_k = u_k - h_{jk} q_j$
7: **fin para**
8: $h_{k+1,k} = \|u_k\|_2$
9: **si** $h_{k+1,k} = 0$ **entonces**
10: parar
11: **fin si**
12: $q_{k+1} = \frac{u_k}{h_{k+1,k}}$
13: **fin para**

Considerando la relación $AQ_k = Q_k H_k + h_{k+1,k} q_{k+1} e_k^T$ (e_k, k-ésimo vector canónico), obtenida en el proceso de Arnoldi (inmediata a partir de (1.12)), y un par de Ritz, $(\tilde{\lambda}, \tilde{x} = Q_k y)$, calculado aplicando el procedimiento de Rayleigh-Ritz sobre las matrices Q_k y H_k se tiene,

$$\|A\tilde{x} - \tilde{\lambda}\tilde{x}\| = \|AQ_k y - \tilde{\lambda} Q_k y\| = \|AQ_k y - Q_k H_k y\| = \|(AQ_k - Q_k H_k)y\|. \qquad (1.13)$$

De la ecuación (1.13) se sigue que la expresión $h_{k+1,k}|e_k^T y|$ proporciona una estimación de la norma del residuo asociado a cada par de Ritz, con la que poder decidir si éste es o no aceptado como convergido. Cuando en la iteración de Arnoldi se genera un subespacio invariante de A ($h_{k+1,k} = 0$ en línea 8 del algoritmo 4), el procedimiento de Rayleigh-Ritz produce pares propios exactos. Si se quieren obtener nuevos autovalores se debe seleccionar un nuevo vector de inicio.

El método de Arnoldi descrito obtiene aproximaciones a pares propios resolviendo problemas de dimensión reducida, y está basado principalmente en operaciones de multiplicación matriz-vector. Esto dota al método de algunas características que lo hacen adecuado para ser utilizado en problemas dispersos de gran dimensión que se resuelven en paralelo sobre memoria distribuida. En primer lugar, no realiza operaciones que produzcan el llenado de la matriz dispersa del problema, en segundo lugar, permite el cálculo de una pequeña porción del espectro y finalmente, si se dispone de una implementación paralela eficiente de la operación de multiplicación matriz-vector, se podrá obtener un buen rendimiento en el cálculo de los valores de interés.

Cuando la matriz que define el problema de valores propios es hermitiana, la matriz Hessenberg del problema proyectado obtenida en la iteración de Arnoldi, se mantiene también hermitiana, por lo que pasa a ser una matriz tridiagonal. El método de Arnoldi adquiere así una forma más simple que da lugar al *método de Lanczos* que es estudiado

en la sección siguiente.

1.3. Método de Lanczos para el cálculo de valores propios

En esta sección nos centramos en describir el método de Lanczos. Este método será el que utilizaremos para el cálculo de autovalores en torno a los desplazamientos generados por la técnica de disección del espectro en la que se basa el método diseñado en este trabajo.

El método de Lanczos, al igual que el método de Arnoldi, es un método de proyección ortogonal sobre espacios de Krylov. Cuando la matriz del problema, A, es hermitiana (simétrica en el caso real), el método de Arnoldi produce, para el problema proyectado, una matriz que es tridiagonal simétrica (Hessenberg y hermitiana), es decir, produce la relación

$$AQ_k = Q_k T_k + \beta_k q_{k+1} e_k^T, \tag{1.14}$$

que expresamos de forma equivalente como

$$AQ_k = [Q_k \quad q_{k+1}] \begin{bmatrix} T_k \\ \beta_k e_k^* \end{bmatrix}. \tag{1.15}$$

La matriz T_k asociada es tridiagonal simétrica y real;

$$T_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{pmatrix}. \tag{1.16}$$

El hecho de ser real, se debe a que los elementos β_i se obtienen como normas de vectores, y a que los elementos diagonales, α_i, de una matriz hermitiana son siempre reales.

En el caso simétrico, la relación (1.14) produce una recurrencia de tres términos por lo que el bucle de ortogonalización del algoritmo de Arnoldi (linea 4 del algoritmo 4), se reduce a restar al nuevo vector generado, una adecuada combinación lineal de los dos últimos vectores de la base computados;

$$\beta_k q_{k+1} = A q_k - \alpha_k q_k - \beta_{k-1} q_{k-1},$$

donde

$$\alpha_k = q_k^* A q_k \text{ y } \beta_k = \|A q_k - \alpha_k q_k - \beta_{k-1} q_{k-1}\|.$$

Como consecuencia, si sólo se desean los valores propios, en principio no debería ser necesario almacenar todos los vectores de la base, sino sólo lo dos últimos para poder continuar con la iteración y obtener T_k del tamaño deseado, a partir de la cual obtener los valores de Ritz. El problema de este planteamiento es que los errores de redondeo que se producen al trabajar en precisión finita, hacen que gradualmente se pierda la ortogonalidad de los vectores de Lanczos calculados.

1.3.1. Reortogonalización

Hay dos tipos de pérdida de ortogonalidad en el procedimiento de Lanczos. Por un lado, se produce una falta de ortogonalidad similar a la descrita en el método de Arnoldi, que se soluciona repitiendo dos veces el proceso de ortogonalización de cada nuevo vector $q = Aq_k$, contra los precedentes, q_k y q_{k-1}. Por otro lado, la utilización de una recurrencia corta como la de Lanczos (de tres términos), conlleva la pérdida de ortogonalidad gradual de q respecto a Q_k. Esta última falta de ortogonalidad puede provocar incluso que los vectores de Lanczos formen un sistema linealmente dependiente y que el algoritmo genere duplicados espurios. Para dar solución a este problema existen varias estrategias de reortogonalización que dan lugar a diversas variantes del método de Lanczos.

Sin reortogonalización. El vector q calculado en cada iteración es almacenado sin reortogonalizar contra los vectores de Lanczos previos, Q_k. El método precisa mecanismos que le permitan distinguir los autovalores genuinos y descartar los falsos replicados.

Reortogonalización periódica. En este caso, si falla determinado test de ortogonalidad, en el que se hace una estimación de $\|Q_k^* Q_k - I\|$, q se ortogonaliza contra todos los vectores de Lanczos calculados.

Reortogonalización parcial. Cuando falla el test de ortogonalidad, q se ortogonaliza contra un subconjunto de vectores de Lanczos.

Reortogonalización selectiva. Utiliza para la reortogonalización vectores de Ritz.

Reortogonalización completa. El nuevo vector generado, q, se ortogonaliza contra todos los vectores de Lanczos calculados hasta el momento.

En la ortogonalización de cada vector contra los previos, los métodos de Arnoldi y Lanczos utilizan el proceso de Gram-Schmidt. Existen dos versiones de dicho proceso: la primera se denomina *Gram-Schmidt clásico* y se ha incluido en el algoritmo 5 (algoritmo de Lanczos). La segunda se conoce como *Gram-Schmidt modificado* y puede verse en el algoritmo 4 (iteración de Arnoldi). Aunque matemáticamente las dos versiones son equivalentes (realizan las mismas operaciones globales), numéricamente, el método de Gram-Schmidt modificado es más estable que el clásico. No obstante, hay trabajos [9] que muestran que cuando la versión clásica del método es utilizada dos veces, se consiguen mejores resultados tanto en estabilidad como en eficiencia. La mayor eficiencia de la versión clásica frente a la modificada se debe a que la primera puede llevarse a cabo utilizando operaciones matriciales en lugar de vectoriales.

Por lo general, las técnicas de reortogonalización descritas pueden hacen que la matriz del problema proyectado, T_k, pierda, en parte, su simetría y derive en una mátriz Hessenberg superior. Sin embargo, con la técnica de reortogonalización completa, los elementos por encima de las tres diagonales principales son suficientemente pequeños y pueden ser descartados.

La variante de Lanczos utilizada en nuestro método (ver algoritmo 5), incluye reortogonalización completa. Esta variante reduce considerablemente las diferencias entre los métodos de Arnoldi y Lanczos, ya que en ambos métodos todos los vectores generados en la iteración necesitan ser almacenados y son utilizados para la ortogonalización, que por otro lado, se repite en los dos métodos. Sin embargo, el hecho de que la matriz del problema proyectado se mantenga simétrica, en el caso de la iteración de Lanczos simplifica la utilización de las distintas técnicas de reinicio.

Algoritmo 5 Iteración de Lanczos con ortogonalización completa

Entrada: matriz $A \in \mathbb{F}^{n\times n}$, vector de inicio $v \in \mathbb{F}^n$, dimensión m

Salida: matrices $Q, T \in \mathbb{R}^{n\times n}$ con $Q^*AQ = T$, $Q^*Q = I_m$ y T tridiagonal simétrica definida por las diagonales $\{\alpha_i\}_{i=1}^{m}$ y $\{\beta_i\}_{i=1}^{m-1}$, β_m y q_{m+1} verificando la relación $AQ_m = Q_m T_m + \beta_m q_{m+1} e_m^T$

$q_1 = \frac{v}{\|v\|_2}$
para $k = 1, 2, \ldots, m$ **hacer**
  $u_k = Aq_k$
  $\alpha_k = q_k^* u_k$
  $u_k = u_k - Q_k Q_k^* u_k$ /*ortogonalización*/
  repetir ortogonalización
  $\beta_k = \|u_k\|$
  si $\beta_k = 0$ **entonces**
    parar
  fin si
  $q_{k+1} = \frac{u_k}{\beta_k}$
fin para

1.3.2. Técnicas de reinicio

Cuando se resuelven problemas de gran tamaño, se tiene interés en el cálculo de una reducida parte del espectro que se corresponde con los autovalores de una determinada parte escogida del plano complejo. Tanto el método de Lanczos como el de Arnoldi son adecuados para estos problemas, aunque no es posible conocer de antemano el número de iteraciones que serán necesarias hasta encontrar los valores buscados. Pretender continuar con la iteración de Lanczos hasta tener todos los valores deseados puede suponer varios problemas. En primer lugar, si la matriz, A, es grande, el almacenamiento de los vectores de la base del espacio de Krylov que se van generando, puede ser inviable. En segundo lugar, cada nuevo vector generado debe ser ortogonalizado contra un mayor número de vectores, por lo que se incrementa el tiempo necesario para ampliar la base del espacio de Krylov a medida que ésta crece. En tercer lugar, el tamaño del problema proyectado y por tanto el tiempo necesario para resolverlo aumenta con el número de iteraciones. Más aún, aunque pudiera resolverse un problema proyectado de superior tamaño, no habría garantía de que los valores que se obtengan sean los buscados. Estas consideraciones hacen que sea

conveniente, una vez que la base calculada ha alcanzado un determinado tamaño, reiniciar el proceso con un nuevo vector de inicio rico en las direcciones de interés que pueda producir de una forma más rápida, mejores aproximaciones. Para determinar el nuevo vector de inicio se utiliza la información proporcionada por las aproximaciones obtenidas tras una ejecución de Lanczos (o Arnoldi) de un determinado número de pasos. Cada vez que se reinicia el proceso, se evita la reaparición de vectores de Ritz previamente convergidos, utilizando deflación (ver §1.2.1).

Existen diversas técnicas de reinicio, aplicables al método de Lanczos y al de Arnoldi. Éstas se describen a continuación, centrándonos en su aplicación al método de Lanczos.

Tras una ejecución de k pasos del método de Lanczos, se dispone de un conjunto de valores de Ritz que separamos en dos subconjuntos, $\tilde{\Lambda}$ y $\Lambda^{\dagger}$ formados respectivamente, por valores de Ritz que serán o no aceptados.

Reinicio explícito

Este esquema, busca la formación de un vector de inicio que contenga direcciones favorables para la convergencia de los valores de interés. Para ello, utiliza un polinomio, p, de modo que $p(\tilde{\lambda}) \ll 1$ si $\tilde{\lambda} \in \Lambda^{\dagger}$, y $p(\tilde{\lambda}) \gg 1$ si $\tilde{\lambda} \in \tilde{\Lambda}$, y toma como vector de inicio $p(A)q_{k+1}$, siendo q_{k+1} el último vector de Lanczos calculado. Al aplicar la matriz $p(A)$ sobre q_{k+1}, se genera un vector de inicio con el que las direcciones de interés se verán potenciadas mientras que se reduce la posibilidad de que aparezcan las no deseadas. Aunque la formación de la matriz $p(A)$ no es necesaria para el cálculo del nuevo vector de inicio, el cálculo de $p(A)q_{k+1}$ por medio de sucesivas operaciones de multiplicación matriz vector, con una matriz generalmente de gran tamaño, tiene un sobrecoste importante, por ello, se recurre a técnicas de reinicio implícito que consiguen los efectos buscados sin incurrir en dicho sobrecoste.

Reinicio implícito

Con esta técnica desarrollada por [6], se consigue reiniciar el proceso desde $p(A)q_{k+1}$, como el caso del reinicio explícito, pero sin efectuar operaciones de multiplicación matriz-vector que involucren a la matriz de gran dimensión del problema original.

Tras efectuar m pasos de la iteración de Lanczos (o Arnoldi), se han generado la matriz T_m y los vectores, V_{m+1}, de la base del espacio de Krylov, $\mathcal{K}_m(v)$, verificando la relación,

$$AV_m = V_m T_m + f_m e_m^*. \tag{1.17}$$

Seleccionando un conjunto de desplazamientos, $\{\mu_i\}_{i=1}^{t}$, por ejemplo, como los descritos en el caso del reinicio explícito (denominados *desplazamientos exactos*), la relación (1.17) puede ser modificada de forma que se corresponda con la asociada al espacio de Krylov $\mathcal{K}(p(A)v)$. Esta transición se lleva a cabo efectuando t pasos del iterativo QR sobre la matriz del problema proyectado (ver algoritmo 6, pasos 1-4). Al truncar la igualdad que aparece en la línea 8 del algoritmo 6, obtenemos una nueva relación de Lanczos. Esto es debido al hecho de que el vector $e_m^* \bar{Q}_m$ tiene distinto de cero únicamente los $(t+1)$ últimos valores, por lo que $(f_m e_m^* \bar{Q}_m)(1 : m-t)$ tiene la forma esperada, $\hat{f}_{m-t} e_{m-t}^*$.

Mediante la técnica de reinicio descrita, se consigue comprimir la información de interés contenida en el espacio de Krylov generado tras m pasos de la iteración de Lanczos. La nueva relación de Lanczos obtenida tras el proceso de reinicio implícito puede ser nuevamente extendida hasta un tamaño m, utilizando el proceso de Lanczos. De esta forma se obtienen espacios de Krylov que gradualmente acumulan mejores aproximaciones a las direcciones de interés, manteniendo un tamaño constante.

Algoritmo 6 Reinicio implícito

Entrada: relación de Lanczos $AV_m = V_m T_m + f_m e_m^*$ asociada a un vector de inicio v, desplazamientos $\{\mu_i\}_{i=1}^t$

Salida: relación de Lanczos $A\hat{V}_{m-t} = \hat{V}_{m-t}\hat{T}_{m-t} + \hat{f}_{m-t}e_{m-t}^*$ asociada al vector de inicio $p(A)v$, siendo $p(A) = (A - \mu_1 I)\cdots(A - \mu_t I)$

1: **para** $i = 1, \ldots, t$ **hacer**
2: $\quad Q_i R_i = T_m - \mu_i I$ /*descomposición QR*/
3: $\quad T_m = Q_i^* T_m Q_i$
4: **fin para**
5: $\bar{Q}_m = Q_1 \cdots Q_t$
6: $\hat{V}_m = V_m \bar{Q}_m$
7: $\hat{T}_m = \bar{Q}_m^* \bar{T}_m \bar{Q}_m$
8: eliminar t últimas columnas de la igualdad $A\hat{V}_m = \hat{V}_m \hat{T}_m + f_m e_m^* \bar{Q}_m$

Thick-restart

La estrategia de reinicio de *Thick-restart* descrita en [22], trabaja, al igual que la de reinicio implícito, con espacios de Krylov, pero no con bases de Krylov (vectores de Lanczos). Parte de la idea de que los espacios de Krylov pueden ser también generados por otras bases distintas de las de Krylov. En concreto, esta técnica utiliza bases de vectores de Ritz para llevar a cabo el reinicio. Generaliza el concepto de relación de Arnoldi (o Lanczos), dando lugar a la relación de Krylov, que se utiliza para caracterizar espacios de Krylov. El siguiente teorema, que enunciamos sin demostración, proporciona una caracterización de los espacios de Krylov. Una demostración del mismo puede encontrarse en [19].

Teorema 12. *Un subespacio $\mathcal{V} = span\{v_1, \ldots, v_m, v_{m+1}\}$ es un subespacio de Krylov ($\mathcal{V} = \mathcal{K}_m(A, q)$ para algún q) si y sólo si existen $B_m \in \mathbb{C}^{m\times m}$, $b \in \mathbb{C}^m$ de forma que*

$$AV_m = V_m B_m + v_{m+1} b^*, \ \textit{con } V = [v_1, \ \cdots, \ v_m], \tag{1.18}$$

Definición 8. Una descomposición del tipo

$$AV_m = V_m B_m + v_{m+1} b_m^* \tag{1.19}$$

se llama *descomposición de Krylov*.

A partir del Teorema 12 se puede deducir que un subespacio que esté generado por algunos de los vectores de Ritz obtenidos a partir de una relación de Lanczos (1.14) , es un espacio de Krylov. Para comprobarlo, consideremos un par de Ritz, $(\tilde{\lambda}, \tilde{v})$ obtenido a partir de la relación (1.14) de orden m. El vector $\tilde{v}$ se puede expresar en la forma $\tilde{v} = Q_k y$, siendo y un vector propio de T_m asociado al valor de Ritz $\tilde{\lambda}$. El residuo asociado al par de Ritz verifica,

$$r := A\tilde{v} - \tilde{v}\tilde{\lambda} = \beta_m q_{m+1} e_m^* y = q_{m+1}\gamma, \quad \text{siendo} \quad \gamma = \beta_m e_m^* y.$$

Es decir, $A\tilde{v} - \tilde{v}\tilde{\lambda} \in \mathcal{K} := span\{q_{m+1}\}$. Considerando ahora un conjunto $\{(\tilde{\lambda}_i, \tilde{v}_i)\}_{i=1}^t$ de pares de Ritz, obtenidos a partir de la relación (1.14) de orden m, definiendo el vector b de forma que en la componente i-ésima toma el valor $\gamma_i = \beta_m e_m^* y_i$, siendo $y_i \in \mathbb{C}^m$ el vector propio tal que $\tilde{v}_i = Q_m y_i$, se tiene

$$AV_t - V_t\Lambda_t = q_{m+1} b_t^* \tag{1.20}$$

para $V_t = [v_1 \quad \cdots \quad v_t]$, $Y_t = [y_1 \quad \cdots \quad y_t]$ y $\Lambda_t = diag(\tilde{\lambda}_i)$, y por tanto

$$AV_t = V_t\Lambda_t + q_{m+1} b_t^*. \tag{1.21}$$

Utilizando el Teorema 12 se concluye que los vectores de Ritz, $\{\tilde{v}_i\}_{i=1}^t$,junto con el último vector de la iteración de Lanczos, q_{m+1}, generan un subespacio de Krylov.

La estrategia de reinicio descrita en esta sección consigue reiniciar el proceso de Lanczos a partir de los vectores de Ritz, sin formar los vectores de Lanczos que generan el subespacio de Krylov correspondiente. Es decir, continúa con la iteración de Lanczos a partir de la relación 1.20, y no a partir de una relación de Lanczos.

Dado que la variante de Lanczos que utiliza nuestro método incorpora la estrategia de reinicio que aquí se describe, y con el fin de facilitar la comprensión de determinados aspectos tratados en la sección 3.3, relativa a la implementación, a continuación se especifica el modo en que dicho reinicio es llevado a cabo. Como ya se ha indicado con anterioridad, se pretende continuar la iteración de Lanczos partiendo de la relación (1.20). Para ver la forma que adquiere la matriz que define dicha iteración, revisamos en primer lugar el modo en que se genera la matriz tridiagonal simétrica cuando se parte de una relación de Lanczos (1.17). Para extender la base asociada a dicha relación se crea el vector Aq_{m+1} que se ortogonaliza contra los vectores de la base, dando lugar a los coeficientes de la columna $(m+1)$ de la matriz del problema proyectado,

$$\begin{aligned} q_{m+2}\beta_{m+1} &= Aq_{m+1} - Q_m Q_m^* Aq_{m+1} - q_{m+1}q_{m+1}^* Aq_{m+1} \\ &= Aq_{m+1} - Q_m(AQ_m)^* q_{m+1} - q_{m+1}\alpha_{m+1} \\ &= Aq_{m+1} - Q_m(Q_m T_m + q_{m+1}\beta_m e_m^*)^* q_{m+1} - q_{m+1}\alpha_{m+1} \\ &= Aq_{m+1} - Q_m T_m Q_m^* q_{m+1} - Q_m e_m \beta_m q_{m+1}^* q_{m+1} - q_{m+1}\alpha_{m+1} \\ &= Aq_{m+1} - q_m\beta_m - q_{m+1}\alpha_{m+1}. \end{aligned} \tag{1.22}$$

Despejando Aq_{m+1} se obtiene la recurrencia de tres términos esperada.

Procedemos ahora de forma análoga, para extender la base asociada a la relación (1.20),

$$\begin{aligned} v_{t+2}\beta_{t+1} &= Aq_{m+1} - V_tV_t^*Aq_{m+1} - q_{m+1}q_{m+1}^*Aq_{m+1} \\ &= Aq_{m+1} - V_t(AV_t)^*q_{m+1} - q_{m+1}\alpha_{t+1} \\ &= Aq_{m+1} - V_t(V_t\Lambda_t + q_{m+1}b_t^*)^*q_{m+1} - q_{m+1}\alpha_{t+1} \\ &= Aq_{m+1} - V_t\Lambda_tV_t^*q_{m+1} - V_tb_tq_{m+1}^*q_{m+1} - q_{m+1}\alpha_{t+1} \\ &= Aq_{m+1} - \sum_{i=1}^{t} v_t\gamma_i - q_{m+1}\alpha_{m+1}. \end{aligned} \tag{1.23}$$

Se ha utilizado el hecho de que $V_t^*q_{m+1} = Y_tQ_m^*q_{m+1} = 0$. En el desarrollo de la ecuación (1.23) se ha mantenido la notación q_{m+1} para marcar la diferencia entre el último vector generado en la iteración de Lanczos anterior y los vectores de Ritz que intervienen en el reinicio. Este vector se correspondería con el vector v_{t+1} en la nueva base del subespacio de Krylov, $\mathcal{R}(V_m)$ (subespacio generado por los vectores columna de V_m). A partir de ahora, el proceso continúa tal como se describe en la ecuación (1.22), generando la recurrencia de tres términos.

La matriz que se genera tras el reinicio (matriz B_m en el algoritmo 7) tiene forma tridiagonal flecha simétrica,

$$B_m = \begin{pmatrix} \tilde{\lambda}_1 & & & \gamma_1 & & \\ & \ddots & & \vdots & & \\ & & \tilde{\lambda}_t & \gamma_t & & \\ \gamma_1 & \cdots & \gamma_t & \alpha_{t+1} & \ddots & \\ & & & \ddots & \ddots & \beta_{m-1} \\ & & & & \beta_{m-1} & \alpha_m \end{pmatrix}. \tag{1.24}$$

En el algoritmo 7 se detalla el proceso a seguir para reiniciar la iteración de Lanczos a partir de un conjunto de vectores de Ritz. Si $t = 0$, este algoritmo coincide con el algoritmo 5.

Al aplicar el procedimiento de Rayleigh-Ritz tras una iteración de Lanczos (de orden m), sin necesidad de formar los vectores de Ritz, se dispone, por un lado, de los valores de Ritz (valores propios del problema proyectado), y por otro, de la estimación del residuo correspondiente a cada par de Ritz: $\|r_i\| = \beta_m|e_m^Ty_i|$, siendo y_i el correspondiente autovector del problema proyectado. Esos datos son suficientes para ordenar y evaluar qué pares han convergido, y cuáles de los no convergidos son los más adecuados para el siguiente reinicio. Los vectores de Ritz (que pueden no haber sido formados) quedan así divididos en tres grupos: $[Y_c \quad Y_r \quad Y_d]$, que se corresponden con las categorías de *convergidos*, *aceptados para reinicio* y *descartados*, respectivamente. En este momento, la iteración de Lanczos puede reiniciarse descartando parte de los vectores obtenidos en la iteración anterior (Y_d) y manteniendo los más adecuados produciendo la matriz en forma de flecha (1.24), B. Los pares de Ritz convergidos se corresponden con valores $\gamma_i \approx 0$, por lo que la matriz se descompone como

$$B = \begin{pmatrix} C & 0 \\ 0 & \tilde{B} \end{pmatrix} \tag{1.25}$$

Algoritmo 7 Thick-Restart

Entrada: relación de Krylov $AV_t = V_t B_t + q_{m+1} b_t^*$ con $b = [\gamma_1, \cdots, \gamma_t]$, $\|q_{m+1}\| = 1$, dimensión m.

Salida: relación de Krylov $AV_m = V_m B_m + v_{m+1}\beta_m e_m^*$ extensión de la inicial.

1: $v_{t+1} = q_{m+1}$
2: $u_{t+1} = Av_{t+1}$
3: $\alpha_{t+1} = v_{t+1}^* u_{t+1}$
4: $u_{t+1} = u_{t+1} - \alpha_{t+1} v_{t+1} - \sum_{i=1}^{t} \gamma_i v_i$
5: $\beta_{t+1} = \|u_{t+1}\|$
6: **para** $k = t+1, 2, \ldots, m$ **hacer**
7: $v_{k+1} = \frac{u_k}{\beta_k}$
8: $u_k = Av_k$
9: $\alpha_k = v_k^* u_k$
10: $u_k = u_k - V_k V_k^* u_k$ /*ortogonalización*/
11: repetir ortogonalización
12: $\beta_k = \|u_k\|$
13: **si** $\beta_k = 0$ **entonces**
14: parar
15: **fin si**
16: **fin para**

La matriz C, que se corresponde con los pares convergidos, ya está en forma diagonal, por lo que es suficiente continuar el cálculo de los nuevos pares de Ritz con la matriz $\tilde{B}$. De este modo, cuando se dispone de pares convergidos, éstos no se incluyen en el conjunto de vectores para reinicio, sino que quedan *bloqueados*, de forma que no son ya modificados. Estos vectores sí continúan interviniendo en el proceso de Lanczos cuando al extender la base, el nuevo vector es ortogonalizado contra los anteriores. Al utilizar estos vectores en la ortogonalización, se está llevando a cabo la deflación de los mismos (ver sección §1.2.1).

Dadas las características del problema que se pretende resolver en este trabajo, el método de Lanczos con Thick-Restart es el que se ha considerado más adecuado para la resolución de los problemas locales que van asociados a la técnica de disección del espectro que utilizamos. En el algoritmo 8 se detalla, de forma concisa, el proceso que se sigue en dicho método. A continuación se dan detalles de cada uno de los pasos del algoritmo, indicando el número de la línea a la que hacen referencia:

- 1: Se obtiene una relación de Lanczos $AQ_m = Q_m T_m + \beta_m q_{m+1} e_m^T$, utilizando la iteración descrita en el algoritmo 5.
- 3: Los valores de Ritz $\{\tilde{\lambda}_i\}_{i=1}^m$ son los valores propios de la matriz proyectada B_m (o T_m). La estimación del residuo se obtiene a partir de la expresión $\|r_i\| = \beta_k |e_k^T y_i|$, donde y_i representan los vectores propios del problema proyectado.
- 4: La ordenación, en el caso del problema simétrico, es muy sencilla. Una vez se ha reducido la matriz B_m a su forma diagonal, se ordenan los valores propios de la

diagonal según el criterio de interés (normalmente por orden decreciente según el módulo). Los vectores propios $\{y_i\}$, deben ser ordenados según la misma ordenación que los valores de Ritz. Los vectores de Ritz quedarán ordenados convenientemente cuando se formen, multiplicando Q_m por los vectores del problema proyectado (ya ordenados). Para comprobar la calidad de las aproximaciones obtenidas se utiliza la estimación del residuo calculada en el paso previo. En la ordenación final suponemos que los pares convergidos quedan a la izquierda del resto.

- 5: Se descartan los vectores del conjunto Y_d.
- 6: Se forman, con la relación $\tilde{v}_i = Q_m y_i$, únicamente los vectores que serán posteriormente utilizados.
- 8: Según se describe en el algoritmo 7

Algoritmo 8 Procedimiento Thick-Restart Lanczos

Entrada: matriz A hermitiana, m tamaño problema proyectado, v vector inicio

1: calcular descomposición de Krylov inicial de orden m
2: **repetir**
3: obtener valores de Ritz y estimaciones de los residuos asociados a los pares de Ritz
4: comprobar convergencia y ordenar
5: truncar la descomposición de Krylov
6: formar vectores de Ritz no descartados
7: bloquear pares convergidos
8: extender a una nueva descomposición de Krylov de orden m
9: **hasta** número suficiente de convergidos **o** número de iteraciones máximas alcanzado

Cuando se utiliza el algoritmo de Lanczos descrito, para la resolución del problema generalizado simétrico, es necesario tener presentes algunas consideraciones que se tratan en la sección siguiente.

1.3.3. Problema generalizado simétrico definido. B-Lanczos

Cuando se quiere resolver el problema generalizado simétrico definido de valores propios, utilizando la transformación espectral de desplazamiento e inversión (1.6), la matriz del problema transformado $S = (A - \sigma B)^{-1}B$ deja, por lo general, de ser hermitiana. Sin embargo, tal como se vio en la demostración del Teorema 10, esta matriz sí es simétrica considerando el producto escalar asociado a B (en caso de ser B definida positiva).

En este caso, si se sustituye el producto escalar estándar $\langle x, y\rangle = y^*x$, por el producto escalar definido por B, $\langle x, y\rangle_B = y^*Bx$, el método de Lanczos puede ser aplicado a S, obteniendo la relación de Lanczos (1.14) y la matriz tridiagonal simétrica típica de esta relación. Hay que tener en cuenta que las condiciones de ortogonalidad que se cumplen entre los vectores de Lanczos, son respecto al nuevo producto escalar introducido. Esta variante del método de Lanczos se conoce como *B-Lanczos*.

Al aplicar B-Lanczos sobre S se obtiene la relación:

$$SQ_m = Q_m T_m + f_m e_m^*, \quad Q_m^* B Q_m = I_m, \quad Q_m^* B f_m = 0. \tag{1.26}$$

A partir de esta relación se obtienen los pares de Ritz de la forma habitual. Con B-Lanczos se puede utilizar sin cambios aparentes la técnica de reinicio thick-restart descrita (ver sección §1.3.2).

B singular.

En caso de ser B singular, dado que (A, B) es un par definido, sabemos que existe una combinación lineal de ambas matrices, H, que es definida positiva, por lo que el proceso de Lanczos podría llevarse a cabo utilizando el producto escalar definido por H. Este planteamiento falla si no es posible conocer tal combinación lineal.

Aún siendo B semidefinida, podemos utilizar el método B-Lanczos teniendo en cuenta las siguientes consideraciones: Al trabajar con la seminorma definida por B, las componentes en el espacio nulo de B pasan desapercibidas. Como consecuencia, al utilizar el B-producto, la iteración de Lanczos no se ve afectada, pero los vectores de Ritz obtenidos tienen acumuladas componentes en el espacio nulo de B que no han podido ser eliminadas durante el proceso. Por tanto, al utilizar B-Lanczos con B singular es necesario realizar algún tipo de postproceso que purifique los vectores contaminados.

Si consideramos un vector propio $\tilde{v}$ calculado con el método B-Lanczos, con B singular, dicho vector estará contaminado con componentes en el espacio nulo de B, es decir $\tilde{v} = v + c$, siendo v el vector propio genuino y $c \in \mathcal{N}(B)$. Dado que la matriz $(A - \sigma B)$ es invertible, las matrices B y S tienen el mismo espacio nulo ($\mathcal{N}(B) = \mathcal{N}(S)$). Como consecuencia, cuando multiplicamos el vector $\tilde{v}$ por S, las componentes en el espacio nulo de B se anulan, mientras que las componentes en la dirección de un vector propio de S mantienen dicha dirección sin cambios:

$$S\tilde{v} = Sv + Sc = \theta v + 0,$$

donde θ es el autovalor de S asociado a v. Una vez se han eliminado las componentes no deseadas, el vector puede ser normalizado.

Esta purificación explícita, requiere la multiplicación de cada vector calculado por la matriz S, lo que implica la resolución de los correspondientes sistemas de ecuaciones triangulares. Cuando se ha calculado una cantidad considerable de vectores (como ocurre, por ejemplo, cuando se están buscando todos los autovalores en un intervalo), y especialmente cuando se trabaja en paralelo con memoria distribuida, esta operación de purificación puede suponer un porcentaje considerable del tiempo total de ejecución.

En [17] se describe el modo de hacer esta purificación de forma implícita, evitando así el sobrecoste que conlleva la purificación explícita.

Considerando la relación (1.26), dado un vector propio de T_m, y, asociado al valor de ritz $\tilde{\theta}$, sea $\tilde{v} = Q_m y$ el vector de Ritz asociado. Entonces se tiene

$$S\tilde{v} = SQ_m y = Q_m T_m y + f_m e_m^* y = Q_m y \tilde{\theta} + f_m e_m^* y = \tilde{v}\tilde{\theta} + f_m e_m^* y \tag{1.27}$$

y normalizando

$$\frac{1}{\tilde{\theta}} S\tilde{v} = \tilde{v} + \frac{1}{\tilde{\theta}} f_m e_m^* y \tag{1.28}$$

La ecuación (1.28) proporciona un modo de efectuar el producto $S\tilde{v}$ de forma implícita, añadiendo a cada vector de Ritz obtenido una porción del último vector de Lanczos calculado. Esta operación se lleva a cabo antes de cada reinicio, con los vectores que se consideran convergidos y van a ser bloqueados.

1.4. Rational Krylov

Al utilizar el método thick-restart Lanczos junto con la transformación espectral de desplazamiento e inversión con un desplazamiento σ_1, es posible, utilizando un esquema similar al descrito en (1.3.2), reiniciar el proceso de Lanczos de modo que éste pueda ser visto como si hubiese sido iniciado utilizando otro desplazamiento, σ_2.

Tal como se muestra en [15], es posible transformar la base del subespacio de Krylov obtenida durante el cómputo desde un desplazamiento σ_1, en otra que genera el mismo subespacio y que puede ser vista como obtenida utilizando otro desplazamiento σ_2. Veremos cómo, partiendo de una relación de Krylov obtenida en el transcurso del algoritmo 7 aplicado a $S_1 = (A - \sigma_1 B)^{-1} B$,

$$S_1 \tilde{V}_m = \tilde{V}_{m+1} \begin{bmatrix} \tilde{\Theta}_m \\ \tilde{b}_m^* \end{bmatrix}, \tag{1.29}$$

donde $\tilde{V}_{m+1} = \begin{bmatrix} \tilde{V}_m & v_{m+1} \end{bmatrix}$, es posible derivar otra equivalente para $S_2 = (A - \sigma_2 B)^{-1} B$, asociada al mismo subespacio $\mathcal{R}(\tilde{V}_m)$ (espacio generado por los vectores columna de $\tilde{V}_m$).

En efecto, de la ecuación (1.29) se sigue

$$B\tilde{V}_m = (A - \sigma_1 B)\tilde{V}_{m+1} \begin{bmatrix} \tilde{\Theta}_m \\ \tilde{b}_m^* \end{bmatrix}. \tag{1.30}$$

Restando el término $\sigma_2 B\tilde{V}_{m+1} \begin{bmatrix} \tilde{\Theta}_m \\ \tilde{b}_m^* \end{bmatrix}$ en ambos lados de la igualdad (1.30) y reordenando obtenemos

$$B\tilde{V}_{m+1} L_m = (A - \sigma_2 B)\tilde{V}_{m+1} \begin{bmatrix} \tilde{\Theta}_m \\ \tilde{b}_m^* \end{bmatrix}, \tag{1.31}$$

con

$$L_m = \begin{bmatrix} I_m \\ 0 \end{bmatrix} + (\sigma_1 - \sigma_2) \begin{bmatrix} \tilde{\Theta}_m \\ \tilde{b}_m^* \end{bmatrix}. \tag{1.32}$$

Así, la relación de Krylov buscada,

$$S_2 W_m = \begin{bmatrix} W_m & w_{m+1} \end{bmatrix} \begin{bmatrix} G_m \\ t_m^* \end{bmatrix}, \tag{1.33}$$

puede obtenerse, utilizando la factorización QR de L_m, $L_m = Q_{m+1}\begin{bmatrix} R_m \\ 0 \end{bmatrix}$, para actualizar la base $W_{m+1} = \tilde{V}_{m+1}Q_{m+1}$ del subespacio de Krylov y la matriz proyectada,

$$\begin{bmatrix} G_m \\ t_m^* \end{bmatrix} = Q_{m+1}^* \begin{bmatrix} \tilde{\Theta}_m \\ \tilde{b}_m^* \end{bmatrix} R_m^{-1} = \frac{1}{\sigma_1 - \sigma_2}\left(I - Q_{m+1}^* \begin{bmatrix} R_m^{-1} \\ 0 \end{bmatrix}\right). \tag{1.34}$$

Nótese que la matriz resultante G_m no tiene ninguna estructura particular de no ceros.

Con este procedimiento es posible reiniciar el algoritmo 7 desde un nuevo *shift*, sin descartar la información del espacio de Krylov obtenida en una ejecución anterior con otro *shift*. Se hace notar que todos los cálculos necesarios para realizar el cambio de base se hacen sin necesidad de operar con las matrices A y B.

Capítulo 2

Diseño del método

En este capítulo, se tratan aspectos relativos al diseño del método presentado en este trabajo y se hace una descripción detallada del mismo. Comenzamos especificando el problema que se pretende resolver.

Entrada:

1. Matrices cuadradas A y B, verificando
   - Complejas hermitianas o reales simétricas,
   - $\mathcal{N}(A) \cap \mathcal{N}(B) = \{0\}$ (par regular),
   - B semidefinida positiva,
   - existe alguna combinación de A y B que es definida positiva.
2. Valores $\alpha, \beta \in \mathbb{R} \cup \{\pm\infty\}$, con α o β finito, que definen el intervalo de interés.

Salida:

1. Conjunto de autovalores del par (A, B) contenidos en $[\alpha, \beta]$,
$$\Lambda = \{\lambda \in [\alpha, \beta] : \exists x \in \mathbb{R}^n \text{ con } Ax = \lambda Bx\}.$$
2. Un conjunto de autovectores, $\{x_i\}_{i=1}^{s} \subset \mathbb{R}^n$ asociados a los autovalores Λ, de forma que $X^T BX = I_s$, siendo X una matriz que tiene por columnas los vectores x_i.

2.1. Análisis del problema: Trabajos previos

Como parte inicial de nuestro estudio se revisan trabajos en los que se tratan problemas similares al nuestro. Este estudio inicial permite extraer las ideas principales de cómo abordar el problema y pone de manifiesto algunas de las dificultades del mismo. Los

trabajos considerados son [7], [8], [23] y [16], a los que nos referiremos, partir de ahora, con las siglas ER, GLS, SIPS, y EA16, respectivamente.

Nos vamos a centrar en los tres primeros trabajos indicados, ya que abordan el problema utilizando, al igual que nosotros, la estrategia de disección del espectro. Todos ellos hacen uso de alguna de las variantes del algoritmo de Lanczos (ver §1.3), junto con la transformación espectral de traslación e inversión (ver §1.1.3), para obtener grupos de autovalores, en torno a *shifts* escogidos, con los que completar el intervalo requerido. También coinciden en la utilización de la información proporcionada por la inercia para validar subintervalos. Esta información se obtiene de la matriz diagonal D de la factorización

$$(A - \sigma B) = LDL^T. \tag{2.1}$$

En todos los trabajos estudiados surge la cuestión de cuántos *shifts* y cuántos valores propios en torno a éstos deben ser calculados. Dado que la convergencia se hace más lenta a medida que los autovalores obtenidos están más retirados del *shift*, interesa, por un lado, reducir el número de autovalores a calcular desde cada *shift*, mientras que por otro lado, interesa minimizar el número de factorizaciones, es decir, el número de *shifts* empleados. Cada método intenta alcanzar un compromiso entre ambos factores contrapuestos por medio de distintas heurísticas, que serán explicadas en la descripción de cada uno de ellos.

En la técnica de disección del espectro, el hecho de que los autovalores se obtienen por grupos, plantea dos dificultades a tener en cuenta. Por un lado, en determinadas ocasiones, no es posible decidir si algunos de los valores obtenidos son duplicidades, multiplicidades o valores muy próximos, de otro autovalor calculado previamente desde otro punto. Por otro lado, el conjunto de vectores calculados debe cumplir determinadas condiciones de ortogonalidad. Cuando se calculan valores bien separados, estas condiciones se consiguen sin necesidad de realizar ningún trabajo extra. Sin embargo, es necesario establecer procedimientos que aseguren la ortogonalidad cuando autovalores múltiples o en *cluster* (agrupaciones de valores propios muy próximos entre sí), son calculados desde distintos *shifts*. Cada uno de los trabajos revisados utiliza enfoques diferentes para resolver las dificultades descritas. La fiabilidad de las medidas adoptadas para abordar estos problemas determinarán en gran medida la robustez del método y la flexibilidad para ser utilizado en cualquier escenario.

Los trabajos considerados se describen a continuación, ordenados por fecha de publicación:

ER Considera que la matriz B, de la ecuación $Ax = \lambda Bx$ que define el problema, es definida positiva, y utiliza la reformulación a problema estándar (ver (1.5)).

Utiliza una versión del método de Lanczos con ortogonalización selectiva en la que no incluye ninguna técnica de reinicio para, en cada *shift* σ, resolver el problema $(D - \sigma I)^{-1}y = \theta y$ donde $D = C^{-1}AC^T$, con C de la descomposición de Cholesky de B. Los pares propios del problema original son (λ, x) con $\lambda = \sigma + \frac{1}{\theta}$, $x = C^{-T}y$. Las matrices D y $(D - \sigma I)^{-1}$ no se forman explícitamente, sino que se utiliza la expresión $(D - \sigma I)^{-1} = C^T(A - \sigma B)^{-1}C$ y la factorización (2.1), de modo que aplicar la matriz $(A - \sigma B)^{-1}$ sobre un vector, se traduce en la resolución de dos sistemas triangulares de ecuaciones.

Calcula el número óptimo de valores propios, `nev`, que deben ser computados desde cada *shift*, minimizando el tiempo necesario para el cálculo de cada autovalor. Ese tiempo se obtiene estimando tanto el tiempo necesario para ejecutar un número determinado de iteraciones del algoritmo de Lanczos, como la cantidad de valores que convergerán en dicha ejecución.

Una ejecución de Lanczos se detiene si han convergido `nev` autovalores o se ha alcanzado el número máximo de iteraciones permitidas (necesario para limitar los requerimientos de memoria del método).

La secuencia de *shifts* se inicia escogiendo el primero de ellos en el extremo izquierdo del intervalo, y se continúa seleccionando el resto hacia la derecha. Cuando se han obtenido todos los autovalores entre dos *shifts* consecutivos, σ_{i-1} y σ_i, se calcula un nuevo desplazamiento desde donde continuar, $\sigma_{i+1} = \sigma_i + 2(\lambda_p - \sigma_i)$, siendo λ_p el autovalor más a la derecha calculado desde σ_i. Si no se dispone de ningún autovalor a la derecha de σ_{i-1}, calcula $\sigma_{i+1} = \sigma_i + \frac{1}{\theta_j}$ siendo θ_j el mayor autovalor positivo de la matriz del problema proyectado en Lanczos (T). Si tampoco se dispone de ningún autovalor positivo de T, entonces se detiene la ejecución. Si tras hacer la factorización de $(A - \sigma_{i+1}B)$ se comprueba que quedan entre σ_i y σ_{i+1} demasiados autovalores para ser calculados desde σ_{i+1}, se escoge otro desplazamiento entre ellos.

En cada ejecución del método de Lanczos, se escoge un vector aleatorio de inicio. Se evita la convergencia a vectores obtenidos previamente ortogonalizando el vector inicial contra los calculados en $[\sigma_{i-1}, \sigma_i]$. Los nuevos vectores calculados pasan un test para determinar si son o no duplicados.

En el caso en que se compruebe (con la inercia) que faltan por calcular autovalores en $[\sigma_{i-1}, \sigma_i]$, se repite una nueva ejecución de Lanczos desde el último *shift* considerado, ortogonalizando el vector de inicio contra todos los calculados previamente. Podría ser necesario repetir este proceso varias veces en presencia de valores múltiples.

En el trabajo de ER se describe la estrategia básica de disección del espectro que seguiremos en nuestro método. Sin embargo, encontramos deficiente el enfoque que presenta para detectar duplicidades, o para asegurar las condiciones de ortogonalidad que deben verificar los autovectores. El trabajo que se describe a continuación, trata estos aspectos con mayor profundidad, y es el que tomaremos como base para nuestro estudio.

GLS En este trabajo se hace una descripción muy detallada del método de Lanczos que emplea. La variante utilizada, es una versión a bloques en la que se hace uso de varios esquemas de reortogonalización y que no incorpora ninguna estrategia de reinicio. El estudio incluye medidas de error para el algoritmo de Lanczos propuesto, así como un extenso análisis de las estrategias de reortogonalización que utiliza. Dado que nuestro método no utilizará esa variante del método Lanczos, no entramos en el estudio detallado de esta parte, y nos centraremos exclusivamente en los aspectos que podrían ser de interés para nuestro estudio.

Para el cálculo de cada grupo de autovalores utiliza la transformación espectral $(A - \sigma B)^{-1}Bx = \theta x$, descrita en (1.6). Dado que esta reformulación a problema estándar es la que nosotros utilizaremos, se hace un breve resumen de las características de la misma:

Los autovalores se modifican según la expresión $\theta = \frac{1}{\lambda - \sigma}$, mientras que los autovectores no se modifican. Tal como se vio en §1.1.4, la matriz $S = (A - \sigma B)^{-1}B$ es simétrica considerando el producto escalar definido por B. Este hecho, permite que para la resolución del problema transformado, se puedan utilizar métodos específicos para matrices simétricas (como Lanczos), con la única salvedad de sustituir el producto escalar ordinario por el B-producto escalar (ver §1.3.3).

La estrategia general de selección de *shifts* es similar a la propuesta en RE. Partiendo del extremo izquierdo del intervalo, se crea un subintervalo confiable inicial, en el que no falta ningún autovalor por encontrar, y se extiende hasta que el subintervalo confiable coincide con el intervalo completo.

Una ejecución de Lanczos termina cuando continuar calculando autovalores resulta ineficiente. En cada iteración se monitoriza la convergencia con el fin de estimar su velocidad y determinar cuándo el coste de calcular nuevos valores, que aumenta al hacerlo el coste por iteración y al disminuir la velocidad de convergencia, es tal que resulta rentable realizar una nueva factorización.

En cada nueva ejecución, utiliza como vectores de inicio aproximaciones obtenidas en la ejecución previa. La finalidad es iniciar la iteración de Lanczos con vectores que contienen direcciones favorables, aunque también advierte que no se consigue un efecto claro con ello.

En la estrategia de selección de *shifts* utiliza varios criterios: Cuando se quiere extender un intervalo confiable $[\sigma_1, \sigma_i]$, utiliza el conjunto de valores aproximados (no convergidos para una determinada tolerancia) obtenidos en la última ejecución. Si han convergido p autovalores a la derecha de σ_i, considera que convergerá la misma cantidad de valores a la izquierda del nuevo *shift* y sitúa éste entre dos valores aproximados consecutivos, dejando a su izquierda p aproximados. Si no dispone de suficientes valores aproximados utiliza un criterio similar al utilizado en el enfoque de ER, y da un salto duplicando la distancia entre el ultimo *shift* y el mayor valor obtenido. Si no han convergido valores a la derecha de σ_i, asume que hay un hueco en el espectro y utiliza la máxima distancia de salto utilizada hasta el momento, y la multiplica por diez.

Asume que el algoritmo de Lanczos que utiliza proporciona la multiplicidad completa de los autovalores si el tamaño de bloque escogido es superior a la máxima multiplicidad posible, aunque no debe ser mayor de 10 por limitaciones del algoritmo.

Utiliza la deflación como estrategia para tratar las dificultades asociadas a la técnica de disección del espectro. Ésta consiste en forzar la ortogonalidad de los vectores de Lanczos contra un grupo seleccionado de vectores. Con ella se busca en primer lugar mantener la ortogonalidad entre vectores calculados desde distintos shifts, en segundo lugar, facilitar la detección de duplicidades y en tercer lugar evitar que algunos valores sean recalculados. No nos extendemos en este punto ya que en la sección §2.3.2 se tratan estos aspectos en detalle.

Tras detener la ejecución en σ_i, pueden faltar valores por encontrar en $[\sigma_{i-1}, \sigma_i]$. Si se sospecha que dichos valores se corresponden con multiplicidades incompletas (tamaño de bloque deficiente), se repite la ejecución desde el mismo *shift*, haciendo deflación con todos los vectores obtenidos en la ejecución anterior. En otro caso, sitúa un nuevo *shift* entre los

dos últimos para buscar los valores desaparecidos.

Tras estudiar las características descritas, este trabajo se ha considerado un buen punto de partida. No obstante, a nuestro entender, plantea algunas deficiencias que se pretenden subsanar. En primer lugar, proponemos la utilización de una versión de Lanczos con reinicio (thick-restart) para evitar las limitaciones que conlleva trabajar con métodos que no incorporan técnicas de reinicio. En segundo lugar, buscaremos nuevos enfoques que permitan una aplicación más general del método, sin restricciones en cuanto a la multiplicidad de los autovalores y que no presuponga conocimiento alguno de dichas multiplicidades. Finalmente, estudiaremos posibles adaptaciones para que sea eficiente en entornos paralelos de memoria distribuida.

SIPs Se centra en un tipo de problema concreto, por lo que determinadas decisiones de diseño están motivadas por características específicas de dicho problema, donde se pretende aplicar. Presenta un enfoque no apropiado para dar solución genérica que pueda ser utilizado en cualquier otro escenario.

Utiliza una tolerancia para detectar *clusters* de autovalores y forzar que el cálculo de cada uno de ellos sea desde un único *shift*. No utiliza deflación por lo que la solución que da al problema de la duplicidad y ortogonalidad no es la más adecuada. Tiene un enfoque multishift paralelo interesante para ser utilizado como paralelismo de segundo nivel con el que se puede obtener muy buena escalabilidad en problemas en los que se busquen gran cantidad de valores propios.

EA16 En este trabajo se describe el diseño de un método implementado en la librería comercial *HSL 2000*, para resolver un problema más general que el que nosotros planteamos. Se basa en una variante del método *rational Krylov* descrito en [18]. Dado que plantea un enfoque diferente al que nosotros seguimos, no entramos en detalles del mismo. Sí se quiere hacer notar que utiliza actualizaciones de la relación de Krylov similares a las descritas en la sección §1.4.

2.2. Entorno paralelo

La *Computación Paralela*, entendida como la utilización de varios computadores para la realización de una determinada tarea común, tiene interés siempre que con ella se obtenga cierta ventaja frente a su ejecución en un único procesador. Las ventajas fundamentales serían: reducir el tiempo de cómputo necesario, o la posibilidad de abordar problemas de gran tamaño, los cuales no podría ser considerados sin la computación paralela. La computación paralela precisa comunicación entre los procesadores que intervienen, tanto para intercambiar datos como para coordinar sus acciones. La forma en que se lleva a cabo esta comunicación determina en gran medida la forma en que se diseñan los métodos que serán utilizados.

Dentro de las máquinas *MIMD* (Múltiple Instrucción sobre Múltiples Datos), en las que nos centramos, existen dos tipos principales de arquitecturas para procesamiento paralelo:

Sistemas de memoria compartida: Formados por un conjunto de procesadores que acceden a una memoria común utilizando el mismo espacio de direcciones. Tiene la ventaja de que los procesadores pueden comunicarse leyendo y escribiendo en esta memoria común. Este acceso concurrente a la memoria debe ser sincronizado (por ejemplo mediante cerrojos), lo que puede suponer una importante limitación en el número de procesadores que acceden a la memoria. Utilizando el soporte de un software apropiado, como por ejemplo, *OpenMP*, o *hilos POSIX*, estos sistemas son relativamente fáciles de programar.

Sistemas de memoria distribuida: En estos sistemas cada procesador tiene su propia memoria, y la comunicación entre los distintos procesadores se hace mediante paso de mensajes a través de una red de comunicación. En este tipo de sistemas, conocidos también como *Sistemas de paso de mensajes*, los datos se mantienen distribuidos entre los distintos procesadores por lo que son apropiados para aplicaciones de gran tamaño, que requieren mantener gran cantidad de datos en memoria. Los mensajes se utilizan tanto para sincronizar la computación como para intercambiar datos, y la latencia en el intercambio de los mismos supone una importante limitación en el tipo de aplicaciones que pueden ser paralelizadas en estos sistemas. Aunque existen APIs estandarizadas, como por ejemplo, *MPI* (*Message Passing Interface*), que facilitan las operaciones de comunicación y permiten hacer código portable en sistemas de memoria distribuida, en general estos sistemas son difíciles de programar, debido, en gran parte, a la importancia y dificultad de una adecuada distribución y gestión de los datos.

Aunque, en general, los entornos paralelos son más difíciles de programar, podemos encontrar librerías específicas de cada arquitectura, especializadas en determinadas áreas, como por ejemplo el álgebra lineal, que proporcionan implementaciones eficientes de operaciones básicas. Estas librerías añaden una capa de homogeneidad a la posible diversidad de los sistemas que se utilizan y facilitan el trabajo del programador que puede abstraerse, al menos en parte, del paralelismo subyacente.

En nuestro caso, pretendemos hacer un diseño para sistemas de memoria distribuida y la implementación se hará en el marco de la librería *SLEPc*, que es una extensión de la librería *PETSc*, de las que se hace una descripción en detalle en el apartado §3.2.2. La librería *PETSc* proporciona herramientas básicas de álgebra lineal para utilizar en la resolución, sobre memoria distribuida, de problemas que utilizan matrices dispersas. El hecho de utilizar la librería *PETSc* tiene varias implicaciones en el diseño el método. En primer lugar, no será necesario entrar en determinados aspectos como el reparto de las matrices entre los distintos procesadores, en segundo lugar, se dispone de útiles herramientas que simplifican las operaciones de comunicación más habituales en este tipo de problemas. Por otro lado, contamos con operaciones matriciales básicas como por ejemplo, la multiplicación matriz-vector, entre otras. Además, podemos también asumir que disponemos de métodos que proporcionan resultados más específicos de valores propios, incluidos en la librería *SLEPc*. Estos detalles se tratarán con más profundidad en el apartado de implementación (ver §3.3). En la descripción del método suponemos que contamos con determinados elementos proporcionados por el entorno del que se parte y nos centramos en los aspectos más específicos de mismo.

2.3. Descripción del método

Sobre la base de la información obtenida de los trabajos previos considerados, se diseñaron unas primeras pruebas de implementación utilizando *Matlab*, que sirvieron para poner de manifiesto las dificultades que conlleva utilizar un enfoque de disección del espectro (descritas en la sección §2.1), y para detectar posibles puntos débiles en los planteamientos de los trabajos estudiados.

Nuestro enfoque, del cual se han comentando ya algunos aspectos a lo largo de la memoria, y que se explica detalladamente en esta sección, surge inicialmente como forma de asegurar el correcto funcionamiento del método en problemas de espectro complicado, con autovalores de elevada multiplicidad o formando grandes *clusters*. En un primer paso se plantearon soluciones relativamente simples, que resultaron no ser adecuadas para abordar casos de cierta complejidad. Sin embargo, y dado que el método debía ser lo más genérico posible, en una segunda fase, se buscaron ejemplos de problemas desfavorables y se replanteó el enfoque para dar solución a los mismos. Una vez que el método dispuso de los elementos necesarios para resolver problemas genéricos sin necesidad de asumir restricciones sobre los mismos, se tuvieron en cuenta otros aspectos como la mejora de la eficiencia del método en el entorno paralelo para el que se diseña.

La descripción del método se divide en varios apartados. En primer lugar mostramos el planteamiento básico del que se parte. En segundo lugar se describe la estrategia de selección de *shifts*. A continuación introducimos el uso de la deflación como herramienta para conseguir robustez y se plantean diversas opciones para la utilización de la misma, que deberán ser evaluadas para comprobar cuál de ellas presenta mejores prestaciones. Por último, estudiamos la posibilidad de reutilizar información de la iteración de Lanczos de una ejecución a otra.

La estrategia básica de la que partimos consiste en utilizar el algoritmo de Lanczos con reinicio descrito en §1.3.2, para resolver el problema transformado $(A - \sigma B)^{-1}Bx = \theta x$ (ver (1.6) en sección §1.1.4), obtener los autovalores extremos de la matriz $S = (A - \sigma B)^{-1}B$, y, a partir de éstos, los autovalores del par (A, B) próximos a σ. Los valores buscados en el interior de un intervalo dado, $[\alpha, \beta]$, se consiguen escogiendo una secuencia adecuada de desplazamientos con los que hacer un barrido de dicho intervalo. En cada *shift* σ_i, se calcula una cantidad fija de autovalores (`nev`), permitiendo para ello una cantidad limitada de reinicios del método de Lanczos (`maxit`).

2.3.1. Selección de desplazamientos

El cálculo de autovalores se inicia siempre en uno de los extremos del intervalo. Generalmente, el primer desplazamiento, σ_1, coincide con el extremo izquierdo del intervalo, y el proceso de selección de nuevos desplazamientos continúa hacia la derecha. En caso de tratarse de un intervalo no acotado inferiormente, se inicia en el extremo derecho y el avance es hacia la izquierda. Para situar cada nuevo desplazamiento, σ_{i+1} (supuestos obtenidos todos los valores entre el extremo inicial y σ_i), se hace uso de información del espectro obtenida hasta el momento. Se considera la separación media de los autovalores

contenidos entre los dos últimos *shifts*,

$$\delta = \frac{|\sigma_i - \sigma_{i-1}|}{n_i},$$

siendo n_i la cantidad de autovalores entre σ_{i-1} y σ_i (diferencia de las inercias), y se calcula el nuevo desplazamiento como

$$\sigma_{i+1} = \tilde{\lambda} + \xi \frac{\texttt{nev}}{2}\delta, \tag{2.2}$$

donde $\tilde{\lambda}$ es el autovalor calculado más alejado de σ_1, `nev` es la cantidad de autovalores que se pretende calcular desde σ_i, y el valor ξ es 1 o -1 dependiendo del extremo inicial, e indica la dirección en la que se avanza. Para simplificar la descripción, a partir de ahora, asumimos que $\xi = 1$, es decir que el proceso se ha iniciado en el extremo izquierdo del intervalo.

Tras la realización de pruebas con el fin de valorar los criterios propuestos en los trabajos estudiados, la expresión para el cálculo del nuevo *shift* (2.2) se ha determinado en base a las siguientes consideraciones:

- Uno de los criterios utilizados en GLS se basa en la utilización de valores *aproximados* obtenidos al ejecutar desde el último *shift* σ_i. Al utilizar un método con reinicio, en la mayoría de los casos no hay suficientes valores aproximados. Además, en presencia de multiplicidades elevadas los valores aproximados obtenidos pueden no ser representativos de los valores propios que realmente hay, por lo que basar la localización de σ_{i+1} en este criterio, puede hacer que queden demasiados autovalores por calcular entre σ_i y σ_{i+1}.

- Utilizando tanto el criterio de los valores aproximados como el de duplicar la distancia al último autovalor calculado, en caso de que hayan convergido pocos valores a la derecha de σ_i, el nuevo *shift*, σ_{i+1}, quedará demasiado cerca del último, con lo que aumenta la probabilidad de recalcular valores ya obtenidos con anterioridad. Esto nos da la idea de que el nuevo desplazamiento debe escogerse suficientemente alejado de σ_i para facilitar la obtención de valores nuevos, y además de modo que el subintervalo $[\sigma_i, \sigma_{i+1}]$ pueda ser completado.

- Utilizar la separación media entre los autovalores calculados a la derecha de σ_i podría no ser representativa en caso de que hubiesen quedado sin determinar la correcta multiplicidad de los valores calculados. Esto no ocurre al considerar la separación media en $[\sigma_{i-1}, \sigma_i]$, intervalo muy próximo donde se tiene un conocimiento real de los valores que contiene.

Una distribución irregular del espectro puede hacer que el criterio (2.2) sitúe el nuevo *shift* demasiado lejos, de forma que el subintervalo $[\sigma_{i-1}, \sigma_i]$ no pueda ser completado desde éste. Incluso en el caso de que el *shift* quede a una distancia adecuada, puede ocurrir que la información de la inercia indique que falta algún autovalor por calcular en $[\sigma_{i-1}, \sigma_i]$, probablemente debido a alguna multiplicidad incompleta. En cualquiera de los dos casos, se utiliza un nuevo desplazamiento entre σ_i y σ_{i+1} para buscar los autovalores desaparecidos. Este proceso se repetirá siempre que sea necesario completar algún subintervalo.

2.3.2. Deflación

En este apartado se introduce la deflación como herramienta para solucionar los problemas inherentes a la técnica de disección del espectro. Éstos son, por un lado, poder distinguir cuando un valor es un duplicado o un valor multiple de otro previamente calculado, y por otro, asegurar la ortogonalidad de los vectores calculados desde varios *shifts*. La deflación consiste en hacer que el método de Lanczos trabaje en el complemento ortogonal (B-ortogonal) de un conjunto seleccionado de vectores. Esto se consigue ortogonalizando cada vector generado en la iteración de Lanczos contra dicho conjunto de vectores. Con esta técnica logramos, en primer lugar, que los autovectores calculados se mantengan ortogonales a los del conjunto seleccionado, y en segundo lugar, que los autovalores asociados a autovectores del conjunto no sean calculados.

La deflación conlleva un coste adicional (por la ortogonalización extra necesaria en cada iteración). Por otro lado, ésta puede tener un efecto positivo en el coste, dado que reduce la posibilidad de recalcular autovalores ya obtenidos.

En este trabajo se pretende evaluar el efecto que presentan distintos esquemas de deflación sobre el rendimiento del método. Para ello, se consideran tres estrategias diferentes a la hora de seleccionar el conjunto de vectores para deflación. Éstas se detallan a continuación.

Estrategia de deflación básica

Con ella se pretende evaluar el caso en el que la deflación considerada es la mínima posible. Es decir, la mínima que asegura la ortogonalidad entre los autovectores calculados desde distintos *shifts* (la iteración de Lanczos asegura la ortogonalidad del conjunto de autovectores obtenidos en una ejecución de la misma). Al tratarse de un problema simétrico, sólo es necesario considerar autovectores asociados a autovalores múltiples o muy próximos entre sí (en *cluster*). Así, en esta estrategia, el conjunto de vectores seleccionados para deflación se forma con los autovectores asociados al mayor autovalor calculado, y a partir de éste, todos los autovalores menores hasta que la distancia entre dos consecutivos sea mayor que $\tilde{\varepsilon}$ (tolerancia que determina cuándo dos autovalores pertenecen a un mismo *cluster*). Con este criterio se asegura la ortogonalidad en caso de haber un *cluster* de autovalores en el extremo derecho de los valores calculados. Este esquema requiere que la tolerancia $\tilde{\varepsilon}$ sea previamente determinada, lo que puede hacerse a partir de las características del espectro, si se tiene algún conocimiento sobre él, o a partir de la tolerancia, ϵ, utilizada para el criterio de parada en el algoritmo de Lanczos, por ejemplo tomando $\tilde{\varepsilon} = \sqrt{\epsilon}$. Para que no exista ambigüedad a la hora de decidir sobre la duplicidad de los nuevos valores obtenidos, al utilizar esta estrategia, cualquier autovalor obtenido a la izquierda de los considerados para deflación será descartado.

Estrategia de deflación GLS

Tras una ejecución de Lanczos, puede haber valores de Ritz no convergidos, situados a la izquierda de otros aceptados. Estos valores aproximados declaran la existencia de

autovalores no calculados entre el correspondiente *shift*, σ_i, y el mayor autovalor obtenido. Si en un caso como éste se utiliza la estrategia de deflación previa, estos valores no podrán ser obtenidos desde un nuevo *shift*, ya que, en caso de ser calculados en la ejecución, después serían descartados. El objetivo de esta segunda estrategia de deflación es posibilitar la computación de estos valores desde el nuevo *shift*, evitando así la utilización de un desplazamiento adicional para calcularlos. En este esquema, que es el utilizado en el trabajo GLS (ver §2.1), el conjunto de vectores seleccionados para deflación se forma con los autovectores asociados al mayor autovalor convergido, $\lambda^\dagger$, que queda a la izquierda de cualquier valor de Ritz no convergido, junto con todos los valores convergidos a su derecha. Al igual que en la estrategia básica, es necesario tener en cuenta posibles *clusters* de autovalores alrededor de $\lambda^\dagger$ (utilizando una tolerancia $\tilde{\varepsilon}$). También en este caso, cualquier autovalor que se calcule a la izquierda del menor autovalor considerado para deflación, será descartado.

Estrategia de deflación aumentada

En ocasiones, el valor $\lambda^\dagger$ (descrito en la estrategia GLS) puede quedar a la derecha de autovalores aún no calculados (si éstos no tienen correspondencia con valores de Ritz no convergidos). Esto ocurre con frecuencia, por ejemplo, en presencia de valores múltiples. En estos casos, si se utiliza cualquiera de las estrategias descritas con anterioridad, las multiplicidades incompletas no podrán ya ser computadas desde el nuevo *shift* (se descartarían). Así, con este tercer criterio se quiere evitar que autovalores pendientes de ser calculados puedan ser descartados. Para determinar el conjunto de vectores para deflación utilizado en esta estrategia se precisa la definición de vecinos izquierdo y derecho de un *shift*. Definimos el *vecino izquierdo*, σ_ℓ, de un *shift* dado σ_i, como el mayor *shift* utilizado, menor que σ_i (no necesariamente el último utilizado σ_{i-1}). De forma equivalente, se define el *vecino derecho* σ_r, como el menor *shift* utilizado mayor que σ_i. En general, un *shift* sólo tendrá vecino izquierdo, a no ser que se haya creado retrocediendo, para buscar valores desaparecidos. Así, con esta estrategia, se considera para deflación el conjunto de autovectores asociados a autovalores previamente calculados, que estén comprendidos entre σ_ℓ y σ_i (o entre σ_ℓ y σ_r si este último existe). De este modo, cualquier valor obtenido entre σ_ℓ y σ_i será aceptado y únicamente se rechazarán valores en el caso improbable de que éstos sean menores que σ_ℓ, valores que, por otro lado, serán duplicados de valores ya calculados, o fuera del intervalo de interés.

Aunque este esquema hace posible la obtención desde el nuevo *shift* de cualquier valor, situado en $[\sigma_\ell, \sigma_i]$, no calculado previamente, aún en este caso pueden quedar valores no obtenidos en dicho intervalo. Ésto puede deberse, por una parte, a que una inadecuada selección del desplazamiento ha dejado demasiados autovalores para ser calculados utilizando un único desplazamiento, o bien, simplemente, porque la iteración de Lanczos se ha detenido tras calcular `nev` autovalores, sin haber completado el subintervalo $[\sigma_\ell, \sigma_i]$. En este último caso, y con el fin de evitar una nueva factorización (nuevo *shift*) para el cálculo de unos pocos valores, nos planteamos la posible conveniencia de continuar la ejecución desde el mismo desplazamiento, al menos unos pocos reinicios más. Esta posible optimi-

zación cuya evaluación se presenta en el capítulo §4, tiene sentido sólo para la estrategia de deflación aumentada, y no para las otras. Para las pruebas de test, nos referiremos a esta opción como *completitud*.

El uso de esta estrategia de deflación, cuando se ha creado un *shift*, σ_i, retrocediendo para buscar valores desaparecidos, tiene el efecto de purgar de $[\sigma_\ell, \sigma_r]$ valores ya calculados (supuestamente de convergencia más favorable), facilitando la aparición del resto. De esta forma, aún en el caso de que falten por calcular autovalores de difícil convergencia, éstos serán finalmente calculados, acotándolos si es necesario mediante la utilización de varios *shifts*, y evitando su reaparición a medida que convergen. En definitiva, con esta técnica de deflación podemos asegurar que siempre que falten autovalores por calcular en un subintervalo, éste podrá ser completado. Por otra parte, se hace notar que dado que esta estrategia permite que el conjunto seleccionado para deflación contenga autovectores obtenidos desde más de un *shift*, con ella siempre se asegura la ortogonalidad entre los vectores propios que se obtienen. Es decir, se asegura la obtención de una base ortogonal del espacio invariante asociado a un valor propio (o *cluster*), aunque una elevada multiplicidad de éste (o del *cluster*), haga que para ello sea necesario utilizar más de dos *shifts*.

Las consideraciones anteriores nos muestran la necesidad de utilizar la estrategia de deflación aumentada siempre que se necesite cerrar un subintervalo incompleto, ya que es la única que asegura que todos los valores serán obtenidos y que los autovectores cumplirán con las condiciones de ortogonalidad requeridas. Sin embargo, cuando no se buscan valores desaparecidos, sino que en el desarrollo de la técnica de disección del espectro se ha completado un subintervalo y se pretende extender, estableciendo un nuevo *shift*, nos podemos plantear cuál de las estrategias descritas es más conveniente en términos de rendimiento. Para ello, se realiza la implementación de las tres opciones descritas, con el fin de evaluar las prestaciones que se obtienen utilizando cada una de ellas. Los resultados obtenidos se muestran en el capítulo 4.

2.3.3. Reinicio desplazado en Lanczos

Tanto en el trabajo de ER como GLS se plantea la utilización de un adecuado vector de inicio en la iteración de Lanczos, de modo que se mejore la convergencia del algoritmo. En un principio, ER considera como vector de inicio una combinación lineal de vectores de Ritz no convergidos de la iteración anterior, aunque finalmente opta por utilizar un vector aleatorio. GLS utiliza para iniciar la iteración de Lanczos a bloques, un conjunto de vectores de Ritz (no convergidos), de la ejecución anterior, aunque no indica que se consiga ninguna mejora con ello. Nosotros, sin embargo, planteamos una estrategia que combina el mecanismo descrito en 1.4 junto con técnicas de reinicio (descritas en sección §1.3.2), de modo que parte del espacio de Krylov creado desde un *shift* pueda ser reutilizado para el siguiente, evitando así que el espacio asociado al nuevo desplazamiento deba ser calculado de nuevo desde cero. Un esquema similar a éste se utiliza en el trabajo EA16 descrito en la sección §2.1.

Cuando se detiene una ejecución de Lanczos, se dispone de valores de Ritz no aceptados como convergidos, junto con los vectores de Ritz asociados a ellos. Estos vectores contienen

información espectral de interés que sería desaprovechada al continuar el cómputo desde otro *shift*. Por otro lado, tal como se muestra en la sección §1.4, al ejecutar desde un *shift*, σ_i, es posible reiniciar la iteración de Lanczos con un nuevo origen, σ_{i+1}. Es decir, es posible transformar una relación de Krylov asociada a $S_i = (A - \sigma_i B)^{-1} B$, producida al iterar desde σ_i,

$$S_i W_m^i = W_{m+1}^i \begin{bmatrix} \Theta_m^i \\ (b_m^i)^* \end{bmatrix},$$

en una relación de Krylov,

$$S_{i+1} W_m^{i+1} = W_{m+1}^{i+1} \begin{bmatrix} \Theta_m^{i+1} \\ (b_m^{i+1})^* \end{bmatrix},$$

asociada a $S_{i+1} = (A - \sigma_{i+1} B)^{-1} B$, con la que reiniciar la iteración de Lanczos desde el nuevo *shift*, σ_{i+1}. Por ello, nos planteamos investigar si puede ser interesante, para mejorar el rendimiento de nuestro método, iniciar el cómputo de autovalores desde un nuevo *shift*, σ_{i+1}, reutilizando parte de la relación de Krylov obtenida en el *shift* previo, σ_i. En concreto, la estrategia que se pretende evaluar consiste en actualizar la relación de Krylov obtenida en σ_i asociada a vectores de Ritz no convergidos, que se corresponden con valores que se encuentran en el subintervalo de interés. Esta actualización se realizaría siempre que el subespacio de Krylov disponible del *shift* previo sea útil para el nuevo, es decir, cuando el *shift* previo sea vecino del nuevo. No obstante, cuando se utilizan las estrategia de deflación mínima o GLS, en caso de que el nuevo *shift* haya sido creado para completar un subintervalo, no será posible utilizar esta optimización ya que, en otro caso, se podría presentar una situación en la que se utilizan para el reinicio vectores de Ritz que son duplicados de algunos de los vectores escogidos para deflación.

El algoritmo 9 muestra el modo en que se actualiza una base de vectores de Ritz obtenidos desde un *shift*, σ_i, de forma que pueda ser reutilizada desde otro, σ_{i+1}.

Algoritmo 9 Reinicio desplazado en Lanczos

Entrada: relación $S_i W_m^i = W_{m+1}^i \bar{H}_m^i$ con $\bar{H}_m^i = \begin{bmatrix} \Theta_m^i \\ (b_m^i)^* \end{bmatrix}$, σ_i, σ_{i+1}

Salida: relación $S_{i+1} W_m^{i+1} = W_{m+1}^{i+1} \bar{H}_m^{i+1}$ con $\bar{H}_m^{i+1} \begin{bmatrix} \Theta_m^{i+1} \\ (b_m^{i+1})^* \end{bmatrix}$

1: $L_m \leftarrow \begin{bmatrix} I_m \\ 0 \end{bmatrix} + (\sigma_i - \sigma_{i+1}) \bar{H}_m^i$

2: $Q_{m+1} \begin{bmatrix} R_m \\ 0 \end{bmatrix} = L_m$ /*factorización QR*/

3: $\bar{H}_m^{i+1} \leftarrow \frac{1}{\sigma_i - \sigma_{i+1}} \left(I - Q_{m+1}^* \begin{bmatrix} R_m^{-1} \\ 0 \end{bmatrix} \right)$

4: $W_{m+1}^{i+1} \leftarrow W_{m+1}^i Q_{m+1}$

Con el fin de evaluar la estrategia descrita, ésta se incluye como opción en la implementación de nuestro método y se realizan pruebas cuyos resultados se muestran en el capítulo 4.

2.3.4. Algoritmo

Algoritmo 10 Disección del espectro para cálculo de autovalores en un intervalo

Entrada: Matrices A y B, intervalo $[\alpha, \beta]$, parámetros para Lanczos {`nev`, `tol`, `max_it`}
Salida: $\{(\lambda_i, x_i)\}_{i=1}^{p}$ pares propios de (A, B) en $[\alpha, \beta]$

1: **si** $\alpha \neq -\infty$ **entonces**
2: $int0 = \alpha$; $dir = 1$ /*int0, extremo donde se inicia el proceso*/
3: **si no**
4: $int0 = \beta$; $dir = -1$
5: **fin si**
6: $P = \varnothing$ /*inicializar estructura de *shifts* pendientes*/
7: $P \leftarrow \sigma(int0)$ /*crea un *shift* en $int0$, sin vecinos, se almacena como pendiente*/
8: $m = \nu(\beta) - \nu(\alpha)$ /*número de autovalores en $[\alpha, \beta]$, ν inercia*/
9: inicializar $V[m]$ y $\Lambda[m]$ /*arrays para almacenar los pares propios*/
10: **mientras** $P \neq \varnothing$ **hacer**
11: $\sigma \leftarrow$ extraer *shift* de P /**shift* para procesar*/
12: factorizar $(A - \sigma B)$, obtener inercia $\nu(\sigma)$
13: $D \leftarrow$\{vectores para deflación seleccionados\}
14: $\{(\theta_i, x_i)\} \leftarrow$ Thick-Restart-Lanczos en $(A - \sigma B)^{-1}B$ restringido a $D^{\perp}$, con B-producto
15: $\{(\lambda_i, x_i)\} \leftarrow \{(\sigma + \frac{1}{\theta_i}, x_i)\}$, descartar no válidos y guardar en Λ y V
/*σ_ℓ y σ_r, vecinos de σ por ambos lados (pueden ser α o β)*/
16: **si** $[\sigma_\ell, \sigma]$ incompleto **entonces**
17: determinar nuevo *shift* entre σ_ℓ y σ y almacenar como pendiente
18: **fin si**
19: **si** $[\sigma, \sigma_r]$ incompleto **entonces**
20: determinar nuevo *shift* entre σ y σ_r y almacenar como pendiente
21: **fin si**
22: **fin mientras**
23: **retornar** Λ y V

En esta sección se detalla el proceso seguido al aplicar la técnica de disección del espectro en el método diseñado. En el algoritmo 10 se da una visión global del proceso, del cual se especifican algunos pasos en las anotaciones siguientes:

- 8: El cálculo de la inercia en α y β requiere sendas factorizaciones. Una de ellas será además utilizada para el cálculo de autovalores en el extremo inicial.
- 10: Bucle principal que se repite mientras quedan *shifts* por computar.
- 13: La selección del conjunto de autovectores para deflación se hace en función de la estrategia de deflación utilizada. A partir de las inercias y de la cantidad de vectores seleccionados para deflación que están asociados a autovalores entre σ_ℓ y σ, es posible conocer el número de autovalores a buscar en dicho subintervalo.

- 14: El proceso en el que se resuelve el subproblema de valores propios asociado a un *shift* se detalla en algoritmo 11.
- 15: La estrategia de deflación utilizada determina el criterio a seguir para decidir qué valores se descartan.

Algoritmo 11 Thick-Restart-Lanczos adaptado a la técnica de disección del espectro

---

Entrada: Matrices A y B, D conjunto de vectores para deflación, parámetros {`nev`, `tol`, `max_it`}, *shifts* actual y vecinos (σ, σ_ℓ y σ_r), m_ℓ y m_r (cantidad de autovalores a calcular en $[\sigma_\ell, \sigma]$ y $[\sigma, \sigma_r]$).

Salida: $\{(\theta_j, x_j)\}_{j=1}^{m}$, conjunto de m pares propios de $(A - \sigma B)^{-1}B$, con $\{x_j\}_{j=1}^{m}$ B-ortogonal a D.

1: obtener vector de inicio, o actualizar espacio de Krylov del *shift* previo
2: **repetir**
3: generar relación de Krylov de orden m /*Lanczos*/
4: $\{(\theta_j, x_j)\}_{j=1}^{m} \leftarrow$ solución problema proyectado
5: comprobar convergencia, ordenar y truncar descomposición
6: formar vectores de Ritz
7: purificación de vectores convergidos
8: calcular c_ℓ y c_r /*número de valores $\lambda = \sigma + \frac{1}{\theta}$ convergidos en $[\sigma_\ell, \sigma]$ y $[\sigma, \sigma_r]$*/
9: **hasta** ($c_\ell = m_\ell$ **y** $c_r = m_r$) **o** (convergidos$\geq$`nev`) **o** (iteraciones=`max_it`)

En el proceso descrito en el algoritmo 10 se genera una secuencia de subproblemas, que son transformaciones espectrales del problema original. Para la resolución de dichos problemas se utiliza el método thick-restart Lanczos (ver sección §1.3.2), por ser el que mejor se adapta a las características de dichos problemas. En la aplicación de este método incluimos algunas adaptaciones útiles para nuestro propósito. Éstas son, por ejemplo, las actualizaciones descritas en §2.3.3, que permiten reiniciar desde un *shift* con vectores de Ritz obtenidos desde otro *shift* vecino. En el algoritmo 11, se detalla la versión adaptada del método thick-restart Lanczos que empleamos, y del que se especifican algunos detalles que nos parecen de interés en las siguientes notas:

- 1: Si el *shift* previo es uno de los vecinos del *shift* actual, se utilizan un conjunto de los vectores de Ritz obtenidos en la ejecución de Lanczos desde el *previo*, para iniciar el cálculo desde el actual, tal como se describe en el algoritmo 9. Si se utilizan las estrategias de deflación mínima o la de GLS, este proceso no se utiliza en caso de que el nuevo *shift* haya sido creado para completar un intervalo.
- 3: En este paso se utiliza la versión del método de Lanczos descrito en el algoritmo 7.
- 5,6: Estos pasos se han detallado en la descripción del algoritmo 8.
- 7: La purificación de los vectores convergidos se lleva a cabo mediante la aplicación, de forma implícita, del operador $(A - \sigma B)^{-1}B$, tal como se describe en la sección§1.3.3.

- 8: Se calcula el número de valores obtenidos es cada subintervalo para comprobar si ya se ha obtenido la cantidad buscada. La cantidad de valores que se buscan en cada subintervalo (parámetro de entrada de este método), ha sido determinada previamente con la información proporcionada por la inercia y con el número de vectores que se deflactan en cada subintervalo.

Capítulo 3

Implementación en SLEPc

3.1. Librerías numéricas

3.1.1. Librerías para cálculo de valores propios

Existen varias librerías numéricas disponibles para el cálculo de valores propios que presentan un enfoque paralelo. En esta sección se hace una recopilación de aquellas que son adecuadas para la resolución de problemas dispersos de gran tamaño, centrándonos en las que incorporan métodos de resolución para el problema generalizado simétrico.

Algunas de las librerías que se describen utilizan un esquema de comunicación inverso, de modo que los métodos de estas librerías no operan directamente con las matrices del problema, sino que el usuario proporciona una función con la que realizar la multiplicación matriz-vector. Con este esquema se consigue la independencia entre el código asociado al método de resolución, y el formato de los datos que se utilizan. Por otro lado, esta metodología hace que el programador no pueda abstraerse del proceso de resolución.

ARPACK (ARnoldi PACKage) [13]. En ella se incluyen implementaciones en Fortran 77 de los métodos de Arnoldi y Lanczos con reinicio implícito y reortogonalización completa. Admite versiones en simple y doble precisión de aritmética real y compleja, y da solución tanto del problema estándar como del generalizado, utilizando un esquema de comunicación inversa. La última versión es la 2.1 (desde 1995). La versión paralela utiliza las librerías BLACS y MPI.

BLZPACK (Block LancZos PACKage) [14]. Proporciona una implementación en Fortran 77 de una versión a bloques del método de Lanczos sin reinicio, para la resolución del problema real simétrico de valores propios, tanto en la forma estándar como generalizada. Permite también el cálculo de todos los valores propios asociados a un intervalo. El enfoque de diseño que presenta, tanto para el método de Lanczos que incorpora como para la estrategia de disección del espectro que utiliza es del trabajo [8], descrito en la sección 2.1. Utiliza comunicación inversa. La versión paralela está implementada sobre MPI. La última versión es la 4.0 (desde 2000).

TRLAN [21] Proporciona una implementaciones secuencial y paralela del método thick-restart Lanczos [22]. Escrito en Fortran 90. Resuelve el problema estándar de valores propios. La versión actual es la 1.0 (desde 2006).

IRBLEIGS Código escrito para la resolución en MATLAB de una variante a bloques del método de Lanczos con reinicio implícito. Aunque el enfoque que presenta es para matrices dispersas, el tamaño de los problemas que resuelve es limitado (no paralelo).

El método implementado por la función *eigs* de MATLAB utiliza el método de Arnoldi con reinicio implícito (similar a ARPACK).

3.1.2. Librerías utilizadas

PETSc Acrónimo de *Portable, Extensible Toolkit for Scientific Computation* [4, 5]. Engloba un conjunto de estructuras de datos y rutinas para la resolución numérica en paralelo, de ecuaciones en derivadas parciales. Utiliza MPI estándar para las operaciones de comunicación por paso de mensajes.

- Requiere las librerías LAPACK, BLAS, y alguna implementación de MPI (no necesario si se configura sin MPI). Además, PETSc puede ser configurado para utilizar numerosas librerías externas, para las que ofrece interfaces de forma que éstas puedan ser utilizadas desde las funciones propias de PETSc. Uno de estos paquetes externos es MUMPS, que será requerido para la ejecución en paralelo del método implementado en el presente trabajo.
- La versión actual es la 3.2 (desde septiembre de 2011).

Se dan más detalles de esta librería en la sección 3.2.1

SLEPc Acrónimo de *Scalable Library for Eigenvalue Problem Computations* [11, 10]. Extiende la funcionalidad de PETSc proporcionando los elementos necesarios para la resolución en paralelo de problemas de valores propios de gran tamaño.

- Se basa en la librería PETSc, de la que es una extensión. Además de las interfaces a librerías externas proporcionadas por PETSc, SLEPc, a su vez, ofrece interfaces a librerías externas específicas de valores propios, como por ejemplo ARPACK o BLZPACK entre otras.
- La versión actual es la 3.2 (desde noviembre de 2011).

Se dan más detalles de esta librería en la sección 3.2.2

MUMPS Acrónimo de *a MUltifrontal Massively Parallel sparse direct Solver* [1]. Librería numérica especializada en la resolución paralela de sistemas de ecuaciones lineales por métodos directos.

- Realiza la resolución del sistema de ecuaciones disperso en tres fases: En primer lugar, se lleva a cabo la fase de análisis, en la que la matriz es procesada para mejorar sus características estructurales. En segundo lugar, la matriz A se factoriza como $A = LU$, o $A = LDL^T$ en caso de ser A simétrica. Finalmente se calcula la solución mediante sustituciones hacia atrás y hacia adelante.
- Para la fase de análisis en la que se lleva a cabo la reordenación de la matriz, MUMPS ofrece la posibilidad de utilizar librerías externas especializadas, para las que ofrece interfaz. En nuestro caso, al configurar PETSc con MUMPS, se ha escogido la librería PARMETIS para la ordenación.
- La versión secuencial requiere la librería BLAS, mientras que la versión paralela utiliza, además, las librerías SCALAPACK, BLACS, y MPI.
- Admite aritmética real y compleja.
- Versión actual: 4.10.0 (desde mayo de 2011).

PARMETIS [12]. Versión paralela sobre MPI de la librería METIS. Ambas librerías están especializadas en métodos para particionado de grafos, particionado de mallas de elementos finitos, y ordenación de matrices dispersas para reducir el llenado en las factorizaciones. En las pruebas de escalabilidad descritas en la sección 4.3 se utilizó la librería MUMPS configurada para la utilización de la librería PARMETIS para la ordenación que se lleva a cabo en la fase de análisis.

LAPACK (Linear Algebra PACKage) [2]. Librería que se ha convertido en un estándar de facto para la resolución en secuencial de los problemas más habituales del álgebra lineal. En la implementación realizada en este trabajo, se utilizan métodos proporcionados por esta librería para la resolución de cualquier problema algebraico asociado al problema proyectado (de dimensión reducida).

3.2. Las librerías SLEPc y PETSc

El método descrito en este trabajo se ha implementado en el marco de la librería SLEPc y está incluido en la versión 3.2 de la misma.

En esta sección se describen algunos aspectos sobre las librerías SLEPc y PETSc que son las que dan soporte al método implementado. Nos detendremos en aquellos aspectos de ambas librerías que estén directamente relacionados con la implementación realizada.

3.2.1. PETSc

Conjunto de herramientas software diseñadas para ser ejecutadas en paralelo, para la resolución numérica de ecuaciones en derivadas parciales que requieren la resolución de sistemas de ecuaciones dispersos de gran tamaño.

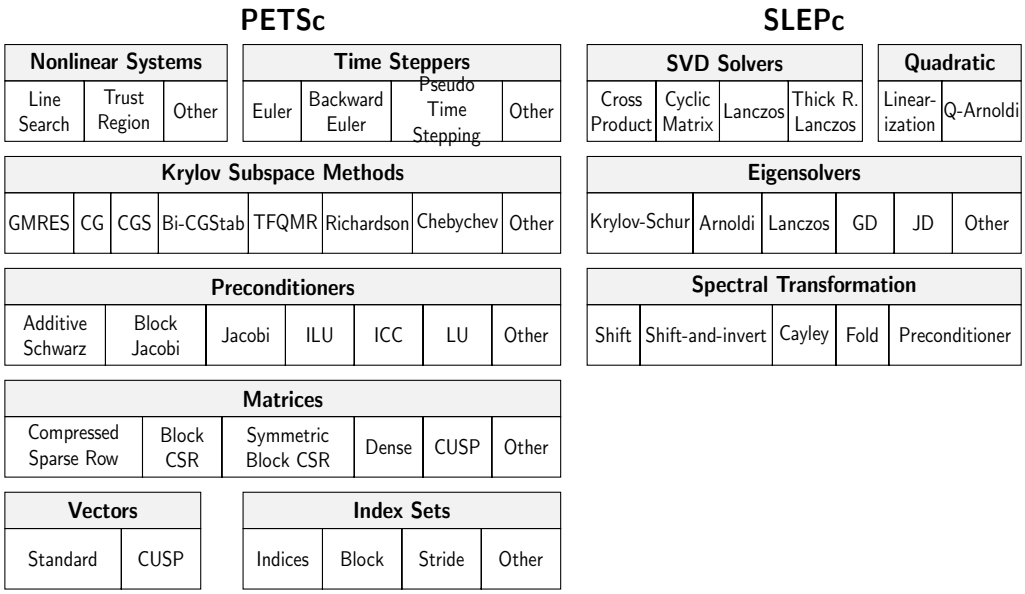

Figura 3.1: Estructura de clases de PETSc y SLEPc.

PETSc incluye entre sus características más destacadas la de ser portable a un amplio rango de plataformas paralelas, escalable a un elevado número de procesadores y flexible a la hora de especificar parámetros relevantes en tiempo de ejecución.

PETSc puede ser compilado para simple y doble precisión tanto para aritmética real como compleja.

La filosofía de programación que ofrece PETSc es la de producir códigos para la resolución de problemas complejos, juntando la funcionalidad de varios paquetes de software especializados. Presenta así un diseño orientado a objetos de modo que toda la funcionalidad que ofrece queda encapsulada en objetos abstractos con los que el programador puede trabajar directamente, a través de las interfaces que cada uno de ellos ofrece, sin necesidad de conocer las estructuras de datos subyacentes. Presenta una estructura jerárquica de modo que partiendo de objetos más básicos como los vectores o matrices, se crean otros más complejos, como objetos *solver* (abstracción de resolución de problemas) que se apoyan en los primeros. En la parte izquierda de la figura 3.1 se muestra la jerarquía de clases en que se estructura la librería PETSc.

Herencia y polimorfismo

PETSc da soporte a gran variedad de formatos de almacenamiento para matrices dispersas. Cada tipo de formato soportado da lugar a una subclase con métodos específicos para cada operación. Sin embargo, esta variedad de métodos no se ve reflejada en la interfaz de usuario. La clase Matriz (y cualquier otra asociada a los objetos principales de PETSc) es una clase polimórfica, de modo que la llamada a una determinada operación de la interfaz dará lugar a la ejecución de la rutina correspondiente al tipo de la matriz que

se utiliza.

Así, un objeto en PETSc, como por ejemplo una matriz, contiene además otra información (*metadatos*), como por ejemplo el tipo (asociado en este caso al formato de almacenamiento), que se utiliza para determinar tanto el modo en que se almacenan los datos, como las rutinas específicas que serán utilizadas en las llamadas a los métodos de la interfaz. Para otros objetos como por ejemplo un KSP, que ofrece una interfaz para la resolución de sistemas de ecuaciones lineales por métodos iterativos basados en subespacios de Krylov, el tipo define el método de resolución que se emplea (GMRES, CG, CGS, etc), dando lugar al correspondiente conjunto de subclases. En este caso, durante la ejecución del método correspondiente, un metadato mantiene información, por ejemplo sobre la convergencia del mismo.

PETSc utiliza este polimorfismo para que sea posible posponer la especificación del tipo de objeto a utilizar hasta el momento de la ejecución. Así, es posible indicar distintas opciones como el tipo de formato de matriz, o el método de resolución que se desea utilizar mediante opciones por línea de comando, sin modificar ni recompilar el código original.

De la funcionalidad proporcionada por PETSc, hemos utilizado para la implementación de nuestro método, principalmente, las librerías para tratamiento paralelo de matrices y vectores, de las que se dan algunos detalles en el siguiente apartado.

Matrices y Vectores en PETSc

Los Vectores y Matrices de PETSc no son sólo estructuras paralelas de datos. El manejo de tales objetos se hace siempre a través de las operaciones definidas en las interfaces asociadas a cada uno de ellos y el programador no trabaja directamente con las estructuras de datos internas.

Los métodos proporcionados por las interfaces para las clases Vec y Mat hacen que sea posible trabajar con vectores y matrices abstrayéndose en gran parte de paralelismo subyacente. PETSc facilita la construcción de forma paralela de vectores y matrices, solapando cómputo con comunicación.

Reparto de datos Los datos de una matriz paralela en PETSc, se distribuyen entre varios procesos mediante bloques consecutivos de filas.

$$\underbrace{\begin{bmatrix} p1 \\ \hline p2 \\ \hline p3 \\ \hline p4 \end{bmatrix}}_{y} = \underbrace{\left[\begin{array}{c} p1 \\ \hline p2 \\ \hline p3 \\ \hline p4 \end{array}\right]}_{A} \underbrace{\begin{bmatrix} p1 \\ \hline p2 \\ \hline p3 \\ \hline p4 \end{bmatrix}}_{x} \tag{3.1}$$

En (3.1) se muestra el reparto de datos entre 4 procesos, de una matriz y dos vectores involucrados en un producto matriz-vector.

Operaciones básicas Con el reparto de datos indicado, la operación de suma tanto de matrices como de vectores es muy simple y no requiere comunicación entre los distintos

procesos (cada uno de ellos realiza la suma con los datos locales). El producto de vectores se realiza mediante cálculos con las partes locales de cada uno de los vectores, más una reducción final con la que todos los procesos obtengan la suma del conjunto de todos los cálculos locales.

Para la operación de multiplicación matriz-vector $y = Ax$, cada proceso realiza el cálculo correspondiente a la parte de y que almacena, utilizando las filas de la matriz A almacenadas localmente, por lo que requiere comunicación punto a punto con cada uno de los procesos que contengan elementos del vector x que necesite.

Dado que los algoritmos iterativos para la resolución de sistemas de ecuaciones lineales que PETSc utiliza efectúan muchas operaciones de multiplicación matriz vector en las que la matriz se mantiene constante (o al menos su estructura), y sólo cambia el vector que interviene en la multiplicación, en la creación de una matriz, PETSc crea también, y de forma transparente para el usuario, las estructuras de datos de comunicación necesarias para que el envío de datos necesario para la multiplicación sea eficiente.

Al utilizar las operaciones para matrices y vectores ofrecidas por la interfaz, la comunicación entre los distintos procesos se hace de forma transparente para el usuario, que puede abstraerse del paralelismo que subyace.

Las operaciones que acabamos de describir son, de las ofrecidas por PETSc, las que utilizaremos principalmente.

Resolución de sistemas lineales por métodos directos Otra operación que nos proporciona esta librería es la de resolución de sistemas lineales. Esta operación la utilizamos al aplicar la matriz $(A - \sigma B)^{-1}$ sobre el vector Bx en cada iteración del método de Lanczos que utilizamos. Dado que los métodos de resolución de sistemas lineales que ofrece PETSc son todos iterativos, en este caso optamos por utilizar una de las interfaces a paquetes software externos que proporciona. En concreto, utilizamos la librería MUMPS para la resolución de los sistemas requeridos utilizando métodos directos. Esta opción nos permite además conocer la inercia de la matriz $(A - \sigma B)$, a partir de la descomposición LDL^T de la misma. Para indicar tales opciones, al utilizar el objeto KSP, que ofrece la funcionalidad de la resolución de los sistemas de ecuaciones lineales, se debe especificar que no utilice ningún método iterativo (*ksp_type=preonly*), y únicamente aplique el precondicionador que se le indique. Así, estableciendo que se utilice un precondicionador tipo Cholesky (*pc_type=cholesky*), se consigue que se aplique sobre el vector la descomposición de la matriz realizada (en secuencial no se requiere ningún paquete externo). Para una ejecución en paralelo indicaríamos además que utilice la librería MUMPS (*pc_factor_mat_solver_package mumps*) para que se utilice la factorización realizada por MUMPS.

3.2.2. SLEPc

SLEPc es una librería computacional para la resolución en paralelo, sobre memoria distribuida, de problemas de valores propios dispersos de gran tamaño. Proporciona solución a varios problemas relacionados, como son: El problema lineal de valores propios, tanto en la forma estándar como generalizada, el problema cuadrático de valores propios, y la

descomposición en valores singulares.

La librería SLEPc está basada en PETSc, que proporciona una infraestructura de clases adecuada para el tratamiento de los problemas algebraicos a los que SLEPc da solución. Tomando como base dicha infraestructura, SLEPc se centra en dar solución a nuevos problemas no incluidos en PETSc, extendiendo así la funcionalidad de ésta, mediante la incorporación de nuevos objetos. En la figura 3.1 se muestra la estructura de clases de PETSc y la extensión de clases proporcionada por SLEPc.

A continuación enumeramos los módulos incluidos en SLEPc, describiendo algunas de las opciones de los dos objetos utilizados en nuestro método.

EPS Objeto para la resolución de un problema de valores propios. Su interfaz proporciona todos los métodos necesarios para la especificación del problema, para la resolución mediante algunos de los métodos disponibles, y finalmente para la recuperación de la solución. La especificación de las características del problema a resolver, así como del método a utilizar para su resolución, puede hacerse tanto con llamadas a funciones específicas dentro del código, o mediante opciones desde línea de comando en el momento de la ejecución. Algunas de estas opciones (en su forma para línea de comandos) son, por ejemplo,

-eps_gen_hermitian indica que el problema asociado es generalizado hermitiano.

-eps_type método de resolución que se desea utilizar. Estos pueden ser, por ejemplo, Krylov-Schur (que pasa a ser thick-restart Lanczos si el problema es hermitiano), o Arnoldi, entre otros. Si no se especifica, ningún tipo, por defecto se utiliza Krylov-Schur.

-eps_nev cantidad de valores que requieren.

-eps_interval especifica un intervalo para el cálculo de todos los autovalores contenidos en él.

-eps_maxit número máximo de iteraciones permitidas para la resolución.

-eps_tol tolerancia a utilizar en el criterio de parada del método escogido

Internamente, este objeto contiene las estructuras de datos necesarias tanto para la ejecución del método de resolución escogido, como para el almacenamiento de la solución final. También mantiene información relativa al estado de la ejecución en un momento dado.

ST Módulo para el tratamiento de transformaciones espectrales. En todos los métodos de resolución implementados en SLEPc es necesaria la aplicación de un operador sobre vectores. Este operador puede ser la matriz original u otro definido por una transformación espectral. En cualquier caso, todo EPS (asociado a un problema) tiene asociado un objeto ST, que es el encargado de definir y manejar tal operador. A veces, aplicar el operador de la transformación espectral precisa de la resolución de sistemas lineales, por lo que todo ST tiene asociado un objeto KSP (proporcionado por PETSc) para la resolución de tales sistemas lineales. Las opciones necesarias para

este último objeto pueden a su vez ser especificadas a través del ST. Algunas de las opciones para este objeto son,

-st_type tipo de la transformación espectral que se realiza (ninguna por defecto). En nuestro caso, utilizamos *shift-invert*.

-st_shift para indicar el desplazamiento asociado a la trasformación espectral.

-st_ksp_type especificación del método de resolución para los sistemas lineales. Puede indicarse que no utilice ningún método iterativo (con la opción *preonly*) para que utilice un método directo.

-st_pc_type especificación del precondicionador a utilizar (por ejemplo *cholesky*).

SVD Módulo para el cálculo de la descomposición parcial en valores singulares.

QEP Objeto asociado a la resolución del problema cuadrático de valores propios.

La última versión de la librería SLEPc incluye una implementación del método descrito en este trabajo. El nuevo método añade funcionalidad al objeto EPS, ampliando las posibilidades de especificación del problema asociado. Así, a partir de la versión 3.2 de SLEPc, cuando se pretende resolver un problema simétrico (EPS_GHEP o EPS_HEP) es posible especificar un intervalo computacional con el que indicar que se requieren todos los autovalores (y autovectores) solución del problema que estén incluidos en dicho intervalo. A continuación se muestra, a modo de ejemplo, la línea de ejecución de un programa (ex13) que resuelve el problema generalizado de valores propios $Ax = \lambda x$, en el que la matriz A es el operador Laplaciano en 2-D, mientras que B es una matriz diagonal. La especificación del problema a resolver (aparte de las matrices que se han generado en el programa), se efectúa, en este caso, mediante opciones desde línea de comandos,

```
$ ./ex13 -eps_interval 0.4,0.8 -st_ksp_type preonly -st_pc_type cholesky
         -eps_gen_hermitian -st_type sinvert
```

Todas las opciones que se muestran en el ejemplo son necesarias cuando se quiere hacer uso de la nueva utilidad descrita (denominada *spectrum slicing*). La opción `st_pc_type cholesky` con la que se indica que utilice una descomposición LDL^T para la resolución de los sistemas triangulares, es necesaria además para que el método pueda obtener la inercia de la matriz factorizada. Dado que PETSc no dispone de ningún método paralelo para la obtención de dicha factorización, cuando este ejemplo se quiere ejecutar en paralelo, es necesario recurrir a un paquete externo para la realización de dicha factorización, y que además proporcione la inercia asociada. Utilizando MUMPS el ejemplo anterior quedaría de la siguiente forma:

```
$ ./ex13 -eps_interval 0.4,0.8 -st_ksp_type preonly -st_pc_type cholesky
         -eps_gen_hermitian -st_type sinvert
         -st_pc_factor_mat_solver_package mumps -mat_mumps_icntl_13 1
```

La última opción es requerida por MUMPS para el cálculo de la inercia.

Al utilizar la técnica de *spectrum slicing*, además de especificar las opciones indicadas, que son necesarias cuando se quieren obtener todos los autovalores comprendidos en un intervalo, es posible también especificar otras opciones incluidas en la interfaz del objeto EPS. Algunas de estas opciones pueden tener un importante efecto sobre el rendimiento del método. Este es el caso de la opción `-eps_nev`, que cuando se ha indicado un intervalo computacional, permite especificar el número de valores que se buscarán en torno a cada *shift* generado al utilizar la técnica de disección del espectro. Este parámetro, que por defecto tiene un valor igual a 40, debe ser cuidadosamente escogido (típicamente en el rango 40-120), ya que influirá en el rendimiento del método. De forma similar, la opción `-eps_maxit` permite indicar las iteraciones máximas permitidas para la ejecución en cada uno de los *shifts*. Normalmente, conviene que este parámetro tenga un valor pequeño (cuando la convergencia es lenta es preferible cambiar de *shift* antes que hacer un número excesivo de iteraciones).

3.3. Detalles de implementación

El proceso que se sigue en el método *spectrum slicing* implementado incluye la resolución de un conjunto de subproblemas de valores propios asociados a un secuencia de desplazamientos seleccionados. El conjunto de autovalores contenido en el intervalo especificado, solución del problema, se obtiene a partir de los autovalores proporcionados por los subproblemas resueltos.

En SLEPc todo problema de valores propios tiene asociado un objeto EPS que mantiene la información relativa a la especificación del problema y al estado de ejecución del mismo. Además, dispone de todas las estructuras de datos necesarias para la resolución del problema y para el almacenamiento de la solución final.

La implementación realizada utiliza la infraestructura proporcionada por el objeto EPS asociado al problema, para la resolución de los subproblemas generados en el método. Además, se mantiene una estructura de datos paralela para almacenar los autovalores y autovectores que se van calculando desde distintos *shifts*. En lo sucesivo, `eps` denotará el objeto EPS asociado al problema, V será un array de vectores (distribuidos) en el que se almacenan los autovectores a medida que son calculados desde distintos *shifts*, y Λ será un array para el almacenamiento de los autovalores (este array se encuentra replicado en todos los procesos que intervienen en el cálculo).

Cuando todos los autovalores en el interior del intervalo han sido encontrados, los arrays globales Λ y V contienen la solución del problema. A partir de entonces ya no se generan nuevos *shifts* ni subproblemas de valores propios asociados a ellos que haya que resolver. En ese momento, los arrays Λ y V se asocian al objeto EPS, `eps`, para retornar la solución.

El método implementado, tal como se describe en la sección 2.3, mantiene un array de *shifts* pendientes, donde éstos se almacenan a medida que se generan, a la espera de ser procesados. Un *shift*, es una estructura que mantiene información de utilidad que esté asociada con uno de los subproblemas que se resuelven. A continuación se describen algunos de los datos incluidos en dicha estructura.

shift

- Valor, σ, en torno al cual se calculan valores propios al resolver el subproblema asociado al *shift* (desplazamiento utilizado en la transformación espectral asociada).
- Inercia de la matriz $(A - \sigma B)$, que coincide con el número de valores propios menores que σ.
- Punteros a los *shifts* vecinos, σ_ℓ y σ_r.
- Número de valores que se buscan en los subintervalos izquierdo y derecho (m_ℓ y m_r respectivamente).
- Flags que informan de si se han encontrado los valores buscados en cada subintervalo vecino.
- Número de autovalores nuevos calculados en la resolución del subproblema asociado.
- Índice en el array global a partir del cual se almacenan los valores obtenidos (y aceptados).

Una vez descrito el proceso general y las estructuras de datos propias del método implementado, vamos a especificar otros aspectos relativos a la implementación. Para ello, vamos a usar como referencia el Algoritmo 10, que describe el método de disección del espectro diseñado.

- Al determinar el extremo en el que se inicia el proceso de creación de *shifts*, `int0`, también se determina el parámetro `dir`, que indica el sentido creciente o decreciente en el que el proceso avanza, desde un extremo del intervalo al otro (de `int0` a `int1`). Así, por ejemplo, si $\texttt{dir} = -1$, significa que se ha iniciado en el extremo derecho del intervalo y se avanza hacia el izquierdo. La definición de este parámetro, en la forma indicada, permite realizar un código independiente del extremo inicial, que trabaja de un modo uniforme para cualquiera de los sentidos. Todas las comparaciones numéricas destinadas a determinar un orden entre dos valores tienen este parámetro en cuenta. Así, por ejemplo, la expresión (3.2) caracterizaría un valor λ comprendido entre `int0` y `int1`, independientemente de si `int0` representa el extremo izquierdo o el derecho del intervalo.

$$\texttt{dir} * (\texttt{int0} - \lambda) < 0 \quad \& \quad \texttt{dir} * (\texttt{int1} - \lambda) > 0 \tag{3.2}$$

  Aunque por claridad, en la descripción del método, se habla de lado izquierdo y derecho, en la implementación del método no existen tales conceptos en el sentido de ser el izquierdo menor que el derecho.

- Los autovalores y autovectores se almacenan en los arrays globales a medida que éstos son calculados. Para mantener una ordenación de los mismos (creciente o decreciente en función de `dir`) se utiliza un vector global de índices.

- Al extraer un *shift* (líneas 11-12 del Algoritmo 10) para ser procesado, se inicializan todos los valores necesarios del objeto `eps`, para la resolución del subproblema

asociado. Esto incluye la factorización de la matriz $(A - \sigma B)$, a partir de la cual se obtiene la inercia (mediante la función de PETSc correspondiente).

- Una vez inicializada la estructura `eps`, se resuelve utilizando el método thick-restart Lanczos implementado en SLEPc que tiene las siguientes características:
  - En cada iteración de Lanczos el nuevo vector es explícitamente B-ortogonalizado contra todos los vectores previos (ortogonalización completa), y contra los vectores determinados para deflación.
  - En la ortogonalización se utiliza un método iterativo de Gram-Schmidt clásico, en el que se efectúa reortogonalización (hasta tres veces) sólo si es necesario.
  - La transformación espectral de *shift and invert* utiliza la resolución de sistemas triangulares por medio de métodos directos a través de PETSc.
  - La resolución del problema proyectado se realiza de forma secuencial en cada uno de los procesos, mediante llamadas a las rutinas de LAPACK.
- La actualización asociada al reinicio desplazado se lleva a cabo según el procedimiento descrito en el Algoritmo 9. Las operaciones en el espacio proyectado (líneas 1-3) se realizan de forma secuencial en cada proceso mediante rutinas de LAPACK. La actualización de la base en el espacio de Krylov (línea 4 del algoritmo 9) requiere operaciones en paralelo.

La paralelización está basada, por un lado, en la realización de operaciones colectivas con matrices y vectores distribuidos en la ejecución de Lanczos, y por otro, en las factorizaciones y resolución de sistemas de ecuaciones lineales en paralelo realizadas por MUMPS.

Capítulo 4

Resultados numéricos

En este capítulo se presentan los resultados obtenidos en las distintas pruebas realizadas para la evaluación del método implementado. En primer lugar se hace una descripción del conjunto de matrices problema que han sido utilizadas para las pruebas, así como del entorno en el que han sido llevadas a cabo. En segundo lugar, se presentan resultados de problemas de tamaño medio, ejecutados en secuencial, con los que se pretende realizar un primer análisis de las distintas opciones implementadas. En tercer lugar se muestran los resultados obtenidos en la ejecución en paralelo de pruebas que utilizan matrices de gran tamaño que tienen importantes requerimientos tanto de memoria, como de tiempo de ejecución. Con dichas pruebas en paralelo se busca, por un lado, determinar cuáles de las opciones implementadas deben quedar en el código final, y por otro llevar a cabo la evaluación de rendimiento del método propuesto. Finalmente se presentan pruebas con las que se quiere comprobar el correcto funcionamiento de nuestro método en un caso de especial complejidad.

4.1. Descripción de problemas test. Entorno de ejecución

La evaluación del método implementado se lleva a cabo mediante la resolución de un conjunto de problemas de test seleccionados. Con los problemas seleccionados se ha buscado cubrir la máxima variabilidad en cuanto a la multiplicidad de los autovalores a calcular. Así, se dispone de distintos tipos de problemas, desde unos en los que todos los valores calculados son simples, hasta otros en los que el conjunto de autovalores calculados presentan clusters de 200 elementos. También se han considerado para la selección de problemas otras características como el tamaño y la posible singularidad de las matrices asociadas. La Tabla 4.1 muestra un resumen de las características de interés de los problemas seleccionados.

Los cinco primeros problemas listados en la Tabla 4.1, se refieren al análisis de estructuras mecánicas, mientras que el resto de problemas test se refieren a la ciencia de los materiales. A continuación se detalla el conjunto de problemas test utilizados, indicando su procedencia.

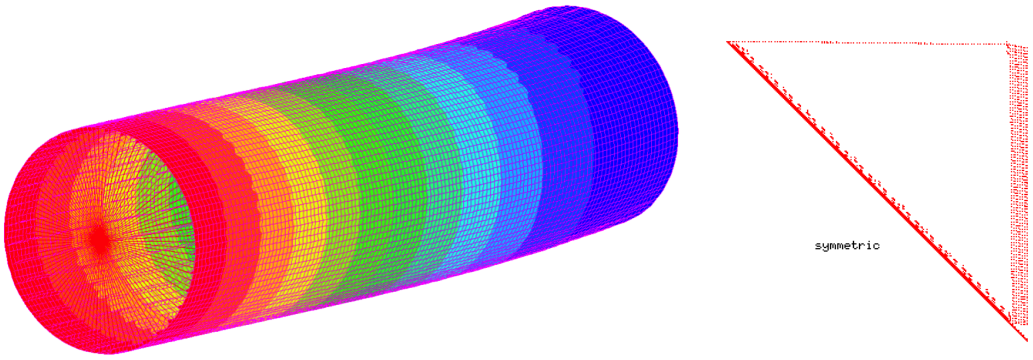

Figura 4.1: Modelado del fuselaje de un avión y patrón de la matriz asociada

fuse En este grupo se incluyen los problemas *fuse5k*, *fuse1m*, *fuse2m*, que han sido proporcionados por la empresa LMS-SAMTECH. Se corresponden con un modelo simplificado, pero realista, de fuselaje, en el que las frecuencias de vibración de interés se encuentran en el intervalo de [0-60] Hz. En la figura 4.1 se muestra una imagen del fuselaje modelado y de la estructura de la matriz asociada. Las matrices *fuse* se corresponden con un modelo paramétrico, que puede ser escalado a cualquier dimensión arbitraria. Los casos de mayor tamaño de este grupo (*fuse5k* y *fuse1m*) son los que se han utilizado para los estudios de escalabilidad de nuestro método. El problema *fuse1k* es un ejemplo de tamaño reducido, en el que los autovalores tienen típicamente multiplicidad dos. Este ejemplo se utiliza para la evaluación en secuencial de la implementación realizada.

v1v2-31 Proporcionada por la empresa LMS-SAMTECH. Problema de tamaño similar a *fuse1k* pero con autovalores simples.

bcsst12 Procedente de la colección de matrices de Harwell-Boeing. Es el problema de menor dimensión utilizado, y todos los autovalores asociados son simples.

graphene_m Proporcionado por los autores de [23]. Este ejemplo, de dimensión similar a *bcsst12*, presenta una distribución irregular del espectro, con elevada multiplicidad.

bencenes-2592 Generado utilizando el código SIESTA[1] para cálculos DFT autoconsistentes. Este problema, de cierta complejidad, ha sido seleccionado para evaluar la robustez de la implementación realizada.

Todas las ejecuciones se han realizado en el supercomputador CaesarAugusta de la Universidad de Zaragoza, perteneciente a la Red Española de Supercomputación (RES). Consiste en un cluster IBM BladeCenter que contiene 256 JS20 nodos blade, cada uno de ellos con dos procesadores PowerPC 970+ de 64 bits, que funcionan a 2.2 GHz de

[1] `http://www.icmab.es/siesta`

Tabla 4.1: Lista de problemas utilizados. Se muestra la dimensión de las matrices, el número elementos no nulos y si B es singular o no, también el intervalo computacional requerido con el número de autovalores que contiene. La última columna muestra la multiplicidad máxima en el intervalo considerado (*: *cluster* de radio 10^{-6}).

nombre	dimensión	no nulos	B sing.	intervalo	# autovals	multipl.
fuse5k	5,406	138,477	yes	$[99, 1{,}5 \cdot 10^7]$	1,145	2
fuse1m	1,036,698	∼29 mill.	yes	$[0, 1{,}4 \cdot 10^5]$	1,989	2
fuse2m	2,141,646	∼59 mill.	yes	$[0, 1{,}4 \cdot 10^5]$	2,039	2
v1v2-31	6,732	177,966	no	$[0, 10^8]$	1,444	1
bcsst_12	1,473	17,868	no	$[100, 10^8]$	578	1
graphene_m	1,600	67,200	no	$[-1, 0]$	921	12 (*)
benzene-2592	23,328	900,720	no	$]-\infty, -0{,}27]$	4,536	200 (*)

frecuencia, con una memoria por nodo de 4 GB, e interconectados por medio de una red Myrinet de baja latencia. Debido a las limitaciones de nuestra cuenta para el uso del cluster, tan sólo se ha hecho uso de 128 procesadores como máximo.

En todas las pruebas realizadas, se ha utilizado una tolerancia $eps\_tol = 10^{-10}$, para determinar cuándo un autovalor es aceptado como convergido, y se ha fijado en 10, el número máximo de reinicios permitidos en cada ejecución de Lanczos ($maxit = 10$). En todos los experimentos en los que se ha utilizado deflación mínima o GLS, se ha considerado una tolerancia de 10^{-5} para detectar clusters.

4.2. Pruebas en secuencial. Evaluación de opciones

En esta sección se muestran los resultados obtenidos en la ejecución sobre un procesador de los problemas test de tamaño reducido descritos en la tabla 4.1. Con estas pruebas se quiere evaluar el comportamiento del método en secuencial, y analizar el efecto que tienen las distintas opciones implementadas cuando se consideran problemas muy diversos, en cuanto a la multiplicidad de los autovalores que se obtienen. Las opciones que se analizan son: La estrategia de deflación utilizada, la posibilidad de forzar la completitud de un subintervalo, y la opción de reinicio desplazado. Los resultados obtenidos se muestran en la Tabla 4.2.

El hecho más destacable es que en todos los problemas considerados se puede observar una reducción significativa en el número total de iteraciones de Lanczos (directamente relacionado con el número de sistemas triangulares resueltos) en los casos en que se utiliza la estrategia de deflación aumentada, con respecto tanto de la estrategia de deflación mínima como de la GLS. Esto se debe al hecho de que en el primer caso se ha descartado un número menor de valores propios duplicados.

En lo que respecta al tiempo de ejecución, se observa que en el problema *graphene_m*, en el que la multiplicidad es elevada, la estrategia de deflación aumentada obtiene mejores

Tabla 4.2: Resultados con matrices pequeñas: def=estrategia de deflación (1=básica, 2=GLS, 3=aumentada), comp= opción de completitud de intervalo, rd=reinicio desplazado, tiempo=tiempo total de ejecución en segundos, nshift=número de *shifts* (en paréntesis el número de retrocesos para completar), res=número de reinicios, its=número total de iteraciones de Lanczos, desc=número de duplicados descartados. Ejecuciones tomando `nev`=40.

nombre	rd	def	comp	tiempo	nshift	res	its	desc
	0	1	0	166	86 (40)	141	8276	874
	0	2	0	139	66 (20)	118	6602	430
fuse5k	0	3	0	137	60 (16)	115	6128	280
	0	3	1	125	51 (8)	116	5548	157
	1	3	1	115	52 (8)	111	4867	141
	0	1	0	120	23 (2)	36	4212	488
	0	2	0	117	22 (1)	35	4049	429
v1v2-31	0	3	0	120	18 (0)	30	3340	90
	0	3	1	121	18 (0)	30	3340	90
	1	3	1	114	18 (0)	31	2825	78
	0	1	0	36.7	30 (17)	51	3517	392
	0	2	0	38.2	30 (17)	50	3464	278
graphene_m	0	3	0	32.6	27 (13)	46	3157	228
	0	3	1	30.6	22 (9)	48	2967	173
	1	3	1	24.0	19 (6)	42	2364	158
	0	1	0	15.6	49 (11)	90	4901	468
	0	2	0	17.0	49 (11)	90	4885	385
bcsst_12	0	3	0	15.7	41 (5)	81	4222	119
	0	3	1	15.0	36 (1)	85	4001	73
	1	3	1	13.6	36 (2)	80	3307	54

resultados que las otras. En el resto de problemas considerados, la diferencia de tiempo que se observa al comparar las tres estrategias de deflación no es significativa. Sin embargo, podemos prever que en la ejecución en paralelo, la reducción en el número de iteraciones que conlleva el uso de la estrategia aumentada, producirá una mayor ganancia de tiempo.

Se observa asimismo que la opción de forzar la *completitud* de un intervalo cuando faltan tan solo unos pocos valores propios permite en algunos casos reducir aún más el número de valores propios descartados, y con ello el tiempo de ejecución. Por otro lado, el uso de dicha opción no supone, en ningún caso, una penalización.

Finalmente, los resultados de la última línea de cada grupo de la tabla muestran que la incorporación del reinicio desplazado en Lanczos supone una importante mejora. En algunos casos se observa una reducción del tiempo de ejecución de hasta el 20 %.

4.3. Pruebas en paralelo. Evaluación de prestaciones

En esta sección se presentan dos tipos de resultados. Por un lado se muestran tiempos asociados a la evaluación en paralelo de las distintas opciones implementadas, con el fin de confirmar o refutar las hipótesis planteadas tras la valoración de las pruebas en secuencial. Por otro lado, y tras haber determinado las opciones que quedarán en la implementación final, se presentan los tiempos obtenidos en el estudio de escalabilidad de dicha implementación final.

Para la evaluación en paralelo de las opciones consideradas, se han llevado a cabo experimentos similares a los efectuados en la sección 4.2, utilizando en este caso el problema *fuse1m* (ver descripción en Tabla 4.1). Los requerimientos de memoria del problema considerado hacen que para la ejecución del mismo sea necesario un mínimo de 16 procesadores. Los resultados obtenidos, que se muestran en la Tabla 4.3, muestran gran similitud a los de la Tabla 4.2, en lo que se refiere a la reducción del número de iteraciones por efecto del uso de la estrategia de deflación aumentada y del reinicio desplazado en Lanczos. En esta ocasión, tal como se esperaba, la reducción en el número de iteraciones queda reflejada en una reducción del tiempo total de ejecución. En la Tabla 4.3 se ha incluido también, el desglose de tiempos para las principales operaciones que intervienen, lo que nos permite observar que el tiempo empleado en las factorizaciones numéricas y en la resolución de sistemas triangulares son proporcionales, respectivamente, al número de shifts utilizados y al número de iteraciones efectuadas. En lo que respecta al tiempo empleado para la ortogonalización, y aunque en principio cabría esperar un incremento del mismo para el caso de la deflación aumentada, se observa, sin embargo, que las diferencias obtenidas son poco marcadas. Esto se debe al hecho de que la mayor parte de dicho tiempo está asociado a la ortogonalización completa y al bloqueo dentro de una ejecución de Lanczos, y no a la deflación contra vectores de otros shifts.

La evaluación de las opciones consideradas nos permite decidir que la implementación final (incluida en SLEPc) utilizará la estrategia de deflación aumentada junto con la opción de forzar la completitud de los subintervalos, e incorporará además la técnica del reinicio desplazado.

Para el estudio de escalabilidad de la implementación final se utilizan los problemas de

Tabla 4.3: Resultados para el problema *fuse1m*, con 16 procesos, utiliza `nev`=80: [rt, def, comp, tiempo, nshift, res, its]=con igual significado que en Tabla 4.2, tNF=tiempo de factorización numérica, tTS=tiempo de resolución de sistemas triangulares, tOrt=tiempo de ortogonalización.

rt	def	comp	tiempo	nshift	res	its	tNF	tTS	tOrt
0	1	0	12286	34 (14)	51	6149	4012	5656	2179
0	2	0	10778	29 (9)	45	5283	3475	4863	2012
0	3	0	10339	27 (7)	43	4989	3181	4594	2144
0	3	1	9928	25 (6)	47	4892	2937	4508	2071
1	3	1	8691	24 (4)	39	3876	2841	3572	1830

Tabla 4.4: Resultados paralelos variando número de procesos (np) con *fuse1m* (`nev`=80) y *fuse2m* (`nev`=120). Estas pruebas se han ejecutado con estrategia de deflación aumentada, opción de completar intervalo y actualización de reinicio desplazado.

name	np	time	nshift	rest	its	tNF	tTS	tOrt
fuse1m	16	8691	24 (4)	39	3876	2841	3572	1830
	32	5181	24 (4)	42	4046	1147	2757	845
	64	3951	25 (6)	48	4340	725	2430	376
	128	3333	25 (5)	41	4084	524	2228	166
fuse2m	32	11744	17 (4)	27	4080	2378	5323	2872
	64	8060	18 (4)	25	4111	1275	4423	1211
	128	6931	18 (4)	26	4228	981	4407	535

test de gran tamaño *fuse1m* y *fuse2m*. En la tabla 4.4 se presentan los resultados obtenidos para ejecuciones efectuadas con distintos números de procesadores, comenzando con 16 para *fuse1m* y 32 para *fuse2m* (minimo requerido para cada uno de los problemas). Se puede apreciar una disminución continuada en los tiempos de ejecución, hasta 128 procesadores (número máximo de procesos considerado). El análisis de los tiempos desglosados por operaciones revela que, tal como era de esperar, la parte de la computación que más contribuye a entorpecer la escalabilidad es la aplicación de $(A-\sigma B)^{-1}$, es decir, la resolución de sistemas triangulares (hechos por MUMPS). Este hecho es el que ha motivado que en el diseño del método propuesto se haya intentado minimizar esta operación reduciendo el número de iteraciones de Lanczos tanto como sea posible.

El tiempo residual (tiempo total menos los tres tiempos de operaciones mostrados en la tabla), se corresponde con la factorización simbólica, la cual es reutilizada desde un shift al siguiente. Dicha factorización se ha llevado a cabo de forma secuencial (ya que la opción de hacerlo en paralelo daba como resultado mayores tiempos para estas matrices en particular), por lo que el tiempo es constante, independientemente del número de procesadores utilizados: alrededor de 400 segundos para *fuse1m* y 1000 segundos para *fuse2m*.

4.4. Ejemplo de aplicación

Para completar nuestro análisis, se muestran asimismo los tiempos de ejecución en paralelo para un problema de prueba que tiene clusters de valores propios de gran tamaño, el mayor de los cuales contiene 200 valores propios (considerando un radio de 10^{-6}). En tales casos, la necesidad de garantizar la ortogonalidad de los vectores propios en los clusters hace que para cualquiera de las estrategias de deflación consideradas precise efectuar ortogonalizaciones contra numerosos vectores. Sin embargo, en el caso de las estrategias de deflación mínima y GLS nos encontramos con una dificultad añadida para seleccionar un valor adecuado para la tolerancia $\tilde{\varepsilon}$ (necesaria para detectar clusters), ya que una mala elección puede suponer una deflación innecesaria (por ejemplo, $\tilde{\varepsilon} = 10^{-5}$ daría lugar a clusters de 400 valores propios). La tabla 4.5 muestra las ejecuciones con la deflación aumentada, utilizando un valor de `nev` igual a 100.

Este problema concreto constituye un ejemplo claro en el que la ejecución en paralelo es muy favorable, debido a que la factorización es extremadamente barata (la distribución de no-ceros se concentra en una banda muy estrecha en torno a la diagonal principal), y por lo tanto la mayor parte del tiempo consumido se debe a la ortogonalización, la cual presenta buena escalabilidad.

Tabla 4.5: Resultados paralelos variando número de procesos (np) con *benzene-2592*, `nev`=100, estrategia de deflación aumentada, opción para completar intervalo y actualización de reinicio desplazado.

np	time	nshift	rest	its	tNF	tTS	tOrt
2	3328	51 (31)	301	33978	15.6	641	2433
4	1720	52 (28)	302	34292	9.1	493	1093
8	1038	54 (30)	305	34679	6.0	408	551
16	698	52 (28)	304	34412	3.8	360	291

Capítulo 5

Conclusiones

Se ha desarrollado un método de disección del espectro en paralelo, que si bien está inspirado en el trabajo publicado por Grimes *et al.*, incorpora, por un lado, técnicas algorítmicas recientes, como el thick-restart Lanczos y el reinicio desplazado, mientras que por otro incorpora modificaciones destinadas a mejorar la implementación en paralelo.

Se ha diseñado una estrategia de deflación destinada a proporcionar la robustez necesaria para asegurar el correcto funcionamiento del método en cualquier contexto. Además esta estrategia permite reducir el número de factorizaciones y de sistemas triangulares a resolver, aún a costa de ortogonalizar más, consiguiendo con ello mejorar sustancialmente la escalabilidad de la implementación realizada.

Esta estrategia proporciona por tanto la robustez y flexibilidad necesarias para que pueda ser usado en situaciones generales, sin restricciones asociadas a problemas concretos que limiten su validez. Por ello, este método puede ser utilizado en problemas que posean una elevada multiplicidad de valores propios o incluso con grandes clusters de los mismos.

Los experimentos llevados a cabo demuestran que el código desarrollado es capaz de resolver problemas con millones de incógnitas, mostrando un buen comportamiento en paralelo hasta con 128 procesadores. Asimismo, se ha comprobado que la estrategia de deflación propuesta combinada con la completitud del intervalo y la actualización del reinicio desplazado, proporciona una reducción significativa del tiempo en todos los casos.

Como continuación de este estudio, nos proponemos explorar en el futuro estrategias con dos niveles de paralelismo, en el que el intervalo se divida en varios subintervalos, cada uno de ellos asignado a un subgrupo distinto de procesos, de forma similar a la estrategia a la seguida por [23].

Agradecimientos

Quiero dar las gracias a todas las personas e instituciones que con su ayuda han hecho posible la realización de este trabajo.

En primer lugar, y muy especialmente, mi agradecimiento a José E. Román, por su valiosa orientación y por la paciencia con que ha dirigido este trabajo.

También quiero mostrar mi gratitud a mi compañero de equipo Eloy Romero, por su ayuda desinteresada, así como al resto del grupo Grycap por la amable acogida que me han ofrecido, especialmente a Fernado Alvarruiz y David Guerrero por el apoyo que siempre me han brindado. Gracias también a mi familia por su comprensión.

Quiero así mismo agradecer la ayuda prestada por las siguientes instituciones:

- Ministerio de Ciencia e Innovación, por haber financiado el proyecto (TIN2009-07519)-*Métodos Avanzados y Técnicas Computacionales Novedosas para la Resolución Numérica de Problemas de Valores Propios de Gran Dimensión*, en el que se incluye este trabajo.
- Red Española de Supercomputación, por los recursos prestados.
- La empresa LMS-SAMTECH por las matrices de prueba proporcionadas.

Publicaciones

El presente trabajo ha dado lugar a la participación en el congreso internacional *International Conference on Scientific Computing (SC2011)*, que tuvo lugar en S. Margherita di Pula, Cerdeña (Italia), en octubre de 2011. En dicho congreso se realizó la ponencia de título *Spectrum Slicing Strategies based on Restarted Lanczos Methods*, quedando ésta reflejada en el *Book of Abstracts* de dicho congreso.

También ha dado lugar al artículo siguiente:

C. Campos, J. E. Román -*Spectrum Slicing Strategies based on Restarted Lanczos Method*. Este artículo ha sido enviado a la revista *Numerical Algorithms* para su consideración (actualmente en revisión).

Bibliografía

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23(1):15–41, 2001.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[3] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[4] Satish Balay, Jed Brown, , Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.2, Argonne National Laboratory, 2011.

[5] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[6] D. Calvetti, L. Reichel, and D. C. Sorensen. An implicitly restarted lanczos method for large symmetric eigenvalue problems. *Electron. Trans. Numer. Anal.*, 2:1–21, 1994.

[7] T. Ericsson and A. Ruhe. The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems. *Math. Comp.*, 35(152):1251–1268, 1980.

[8] Roger G. Grimes, John G. Lewis, and Horst D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM J. Matrix Anal. Appl.*, 15(1):228–272, 1994.

[9] V. Hernandez, J. E. Roman, and A. Tomas. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Comput.*, 33(7–8):521–540, 2007.

[10] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: Scalable Library for Eigenvalue Problem Computations. *Lecture Notes in Computer Science*, 2565:377–391, 2003.

[11] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, 2005.

[12] G. Karypis, K. Schloegel, and V. Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. Technical report, Version 1.0, Dept. of Computer Science, University of Minnesota, 1997.

[13] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1998.

[14] O. A. Marques. BLZPACK: Description and user's guide. Technical Report TR/PA/95/30, CERFACS, Toulouse, France, 1995.

[15] K. Meerbergen. Changing poles in the rational Lanczos method for the Hermitian eigenvalue problem. *Numer. Linear Algebra Appl.*, 8(1):33–52, 2001.

[16] K. Meerbergen and J. Scott. The design of a block rational Lanczos code with partial reorthogonalization and implicit restarting. Technical Report RAL-TR-2000-011, Rutherford Appleton Laboratory, 2000.

[17] Bahram Nour-Omid, Beresford N. Parlett, Thomas Ericsson, and Paul S. Jensen. How to implement the spectral transformation. *Math. Comp.*, 48(178):663–673, 1987.

[18] A. Ruhe. Rational Krylov subspace method. In Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, pages 246–249. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[19] G. W. Stewart. A Krylov–Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23(3):601–614, 2001.

[20] G. W. Stewart. *Matrix Algorithms. Volume II: Eigensystems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[21] K. Wu and H. Simon. Trlan user guide. Technical Report LBNL-42953, Lawrence Berkeley National Laboratory, 1999.

[22] Kesheng Wu and Horst Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 22(2):602–616, 2000.

[23] Hong Zhang, Barry Smith, Michael Sternberg, and Peter Zapol. SIPs: Shift-and-invert parallel spectral transformations. *ACM Trans. Math. Softw.*, 33(2):1–19, 2007.