UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

Implementación paralela de métodos iterativos para la resolución de problemas polinómicos de valores propios

Tesis Doctoral

Programa de Doctorado en Informática

Autora:
Carmen Campos González

Director:
José E. Román Moltó

Mayo 2017

Agradecimientos

Quiero expresar mi más profundo agradecimiento a José Román, por su excelente trabajo como director y por su dedicación y valiosa aportación a esta tesis. Su gran entrega como investigador ha sido para mí un ejemplo y un estímulo constantes.

En segundo lugar, me gustaría mencionar al profesor Daniel Kressner ya que algunos de sus trabajos han sido fuente de inspiración en mayor o menor medida de ideas que considero relevantes para esta tesis.

También quiero manifestar mi agradecimiento a Andrés Tomás y Eloy Romero por sus contribuciones a SLEPc, algunas de las cuales han servido de referencia para los desarrollos software realizados en esta tesis.

Mi reconocimiento al Ministerio de Educación Cultura y Deporte por la beca otorgada para la realización de esta tesis, y a la Red Española de Supercomputación por los recursos computacionales proporcionados para este trabajo.

En el ámbito personal, me gustaría agradecer, de nuevo a José Román, el permitirme entrar a formar parte del proyecto SLEPc; a mi compañero de equipo Alejandro Lamas, sus constantes muestras de ánimo y su sentido del humor; a los compañeros de Departamento, Fernando Alvarruiz y David Guerrero, la buena disposición que siempre han mostrado para escucharme y ayudarme; finalmente a mi familia, su apoyo continuado e incondicional.

Resumen

El problema polinómico de valores propios aparece en muchas áreas de la computación científica y técnica. Puede verse como una generalización del problema lineal de valores propios en el que la ecuación $P(\lambda)x = 0$, que define el problema, involucra un polinomio $P(\lambda)$, de grado d, en el parámetro λ del autovalor, y $d+1$ coeficientes matriciales. A su vez, el problema polinómico de valores propios es un caso particular del problema no lineal de valores propios, $T(\lambda)x = 0$, en el que T es una función matricial no lineal. Estos problemas aparecen en diversas áreas de aplicación como acústica, mecánica de fluidos, análisis de estructuras, o fotónica.

Esta tesis se centra en el estudio de métodos para la resolución numérica del problema polinómico de valores propios, así como la adaptación de dichos métodos al caso más general no lineal. Principalmente, se consideran métodos de proyección, que son apropiados para el caso de matrices dispersas de gran dimensión cuando se requiere solo un pequeño porcentaje de los valores y vectores propios. Los algoritmos se estudian desde el punto de vista de la computación de altas prestaciones, teniendo en consideración aspectos como la eficiencia (computacional y de memoria) y la computación paralela.

SLEPc, Scalable Library for Eigenvalue Problem Computations, es una biblioteca software para la resolución de problemas de valores propios de gran dimensión en paralelo. Es de propósito general y puede usarse para problemas estándares y generalizados, simétricos y no simétricos, con aritmética real o compleja. Como fruto de esta tesis, se han desarrollado diversos solvers para problemas polinómicos y no lineales, los cuales se han incluido en las últimas versiones de este software.

Por un lado, se abordan métodos basados en la linealización del problema polinómico, que resuelven un problema lineal equivalente de dimensión varias veces la del inicial. Entre ellos se destaca el método TOAR, que representa la base del subespacio de búsqueda de una forma eficiente en términos de memoria, y es adecuado para manejar el aumento de dimensión del problema lineal. La tesis también propone variantes específicas para el caso particular de matrices simétricas. En todos estos métodos se consideran diversos aspectos para dotar a las implementaciones de robustez y flexibilidad, tales como transformaciones espectrales, escalado, y técnicas de extracción.

Además de los métodos de linealización, se proponen métodos de tipo Newton, como el método de Jacobi-Davidson con deflación y el método de Newton para pares invariantes. Por sus características, este último no suele utilizarse como un método

en sí mismo sino como técnica de refinamiento de las soluciones obtenidas con otro método.

Los métodos anteriores pueden aplicarse a la resolución del problema no lineal, utilizando técnicas como la interpolación polinómica o racional, siendo necesario en algunos casos adaptar los algoritmos. La tesis cubre también estos casos.

Para todos los algoritmos considerados se han realizado implementaciones paralelas en SLEPc, y se ha estudiado su comportamiento numérico y sus prestaciones paralelas en problemas procedentes de aplicaciones reales.

Resum

El problema polinòmic de valors propis apareix en moltes àrees de la computació científica i tècnica. Pot veure's com una generalització del problema lineal de valors propis en el qual l'equació $P(\lambda)x = 0$, que defineix el problema, involucra un polinomi $P(\lambda)$, de grau d, en el paràmetre λ de l'autovalor, i d+1 coeficients matricials. Al seu torn, el problema polinòmic de valors propis és un cas particular del problema no lineal de valors propis, $T(\lambda)x = 0$, en el qual T és una funció matricial no lineal. Aquests problemes apareixen en diverses àrees d'aplicació com a acústica, mecànica de fluids, anàlisis d'estructures, o fotònica.

Aquesta tesi se centra en l'estudi de mètodes per a la resolució numèrica del problema polinòmic de valors propis, així com l'adaptació d'aquests mètodes al cas més general no lineal. Principalment, es consideren mètodes de projecció, que són apropiats per al cas de matrius disperses de gran dimensió quan es requereix solament un reduït percentatge dels valors i vectors propis. Els algorismes s'estudien des del punt de vista de la computació d'altes prestacions, tenint en consideració aspectes com l'eficiència (computacional i de memòria) i la computació paral·lela.

SLEPc, Scalable Library for Eigenvalue Problem Computations, és una biblioteca software per a la resolució de problemes de valors propis de gran dimensió en paral·lel. És de propòsit general i pot usar-se per a problemes estàndards i generalitzats, simètrics i no simètrics, amb aritmètica real o complexa. Com a fruit d'aquesta tesi, s'han desenvolupat diversos solvers per a problemes polinòmics i no lineals, els quals s'han inclòs en les últimes versions d'aquest software.

D'una banda, s'aborden mètodes basats en la linealització del problema polinòmic, que resolen un problema lineal equivalent de dimensió diverses vegades la de l'inicial. Entre ells es destaca el mètode TOAR, que representa la base del subespai de cerca d'una forma eficient en termes de memòria, i és adequat per a manejar l'augment de dimensió del problema lineal. La tesi també proposa variants específiques per al cas particular de matrius simètriques. En tots aquests mètodes es consideren diversos aspectes per a dotar a les implementacions de robustesa i flexibilitat, tals com a transformacions espectrals, escalat, i tècniques d'extracció.

A més dels mètodes de linealització, es proposen mètodes de tipus Newton, com el mètode de Jacobi-Davidson amb deflació i el mètode de Newton per a parells invariants. Per les seues característiques, aquest últim no sol utilitzar-se com un mètode en si mateix sinó com a tècnica de refinament de les solucions obtingudes amb un altre mètode.

Els mètodes anteriors poden aplicar-se a la resolució del problema no lineal, utilitzant tècniques com la interpolació polinòmica o racional, sent necessari en alguns casos adaptar els algorismes. La tesi cobreix també aquests casos.

Per a tots els algorismes considerats s'han realitzat implementacions paral·leles en SLEPc, i s'ha estudiat el seu comportament numèric i les seues prestacions paral·leles en problemes procedents d'aplicacions reals.

Abstract

The polynomial eigenvalue problem appears in many areas of scientific and technical computing. It can be seen as a generalization of the linear eigenvalue problem in which the equation $P(\lambda)x = 0$, that defines the problem, involves a polynomial $P(\lambda)$, of degree d, in the parameter λ (the eigenvalue), and $d+1$ matrix coefficients. In its turn, the polynomial eigenvalue problem is a particular case of the nonlinear eigenvalue problem, $T(\lambda)x = 0$, in which T is a nonlinear matrix function. These problems appear in diverse areas of application such as acoustics, fluid mechanics, structure analysis, or photonics.

This thesis focuses on the study of methods for the numerical resolution of the polynomial eigenvalue problem, as well as the adaptation of such methods to the most general nonlinear case. Mainly, we consider methods of projection, that are appropriate for the case of sparse matrices of large dimension, where only a small percentage of eigevalues and eigenvectors are required. The algorithms are studied from the point of view of high-performance computing, considering issues like (computational and memory) efficiency and parallel computation.

SLEPc, Scalable Library for Eigenvalue Problem Computations, is a software library for the parallel solution of large-scale eigenvalue problems. It is of general purpose and can be used for standard and generalized problems, both symmetric and nonsymmetric, with real or complex arithmetic. As a result of this thesis, we have developed several solvers for polynomial an nonlinear eigenproblems, which have included in the last versions of this software.

On one hand, we consider methods based on the linearization of the polynomial problem, that solves an equivalent linear eigenproblem of dimension several times the initial size. Among them, the TOAR method stands out, that repre- sents the search subspace basis in an efficient way in terms of memory, and is appropriate to handle the increase of dimension of the linear problem. The thesis also proposes specific variants for the particular case of symmetric matrices. In all these methods we consider several aspects to provide the implementations with robustness and flexibility, such as spectral transformations, scaling, and techniques of extraction.

In addition to the methods of linearization, we propose methods of Newton type, such as the method of Jacobi-Davidson with deflation and the method of Newton for invariant pairs. Due to its characteristics, the latter is not usually employed as a proper method, but as a technique for refinement of the solutions obtained with another method.

The previous methods can also be applied to the resolution of the nonlinear problem, using techniques like polynomial or rational interpolation, being necessary in some cases to adapt the algorithms. This thesis covers also these cases.

For all the considered algorithms we have made parallel implementations in SLEPc, and have studied its numerical behaviour and its parallel performance in problems coming from real applications.

Índice general

Capítulo 1

Introducción y objetivos

En el marco de la computación científica, en la que se unen varias disciplinas para dar solución computacional a modelos matemáticos de problemas de la ciencia y la ingeniería, encontramos que el cálculo de valores y vectores propios está presente en gran variedad de aplicaciones científicas, como son, por ejemplo, el análisis de estabilidad en sistemas dinámicos, el estudio de la resonancia en circuitos electrónicos, la obtención de los modos naturales de vibración en estructuras, el estudio de la configuración electrónica molecular, o el procesamiento de señales, entre muchas otras (ver por ejemplo [77]). El problema de valores propios se formula, en su forma estándar, mediante la ecuación

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

donde $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C}$ son incógnitas a calcular, y $A \in \mathbb{C}^{n\times n}$ es el dato que describe el problema. Para la resolución del problema (1.1) existe software especializado con el que los investigadores de distintos campos pueden realizar dicho cálculo. En muchas aplicaciones de ingeniería, la matriz A es típicamente *dispersa* (con un elevado porcentaje de elementos cero) y de gran dimensión (habitualmente obtenida mediante discretización de un sistema continuo). Problemas de este tipo, en los que además sólo se requiere el cálculo de una pequeña porción del espectro, se resuelven mediante métodos específicos llamados *iterativos*, en oposición a los denominados *directos* que son adecuados para el cálculo de todo el espectro. Una diferencia importante entre ambos tipos de métodos, que limita la utilización de estos últimos cuando el tamaño del problema es elevado, es que los métodos directos modifican la matriz inicial haciendo que se pierda la dispersión, aunque en contrapartida calculan todo el espectro con bastante exactitud. Aún en el caso de que sólo se requiera una pequeña cantidad de pares propios, los problemas de gran dimensión presentan grandes exigencias computacionales, tanto en memoria como en tiempo de cálculo, por lo que el software especializado en este tipo de problemas suele desarrollarse sobre un paradigma de computación paralela.

En este contexto se sitúa la biblioteca SLEPc [39, 72], *Scalable Library for Eigenvalue Problem Computations*, dedicada a la resolución, en entornos de computación paralela, de problemas de valores propios dispersos de gran dimensión. Esta bibliote-

ca, de distribución libre, tiene un amplio reconocimiento en la comunidad científica. El objeto principal de SLEPc ha sido, desde su inicio en 2002, dar soporte al problema lineal de valores propios, para el que se han incorporado desde entonces diversos métodos, aplicables a una amplia variedad de problemas. Sin embargo, SLEPc también se ha extendido dando cobertura al problema polinómico y no lineal de valores propios, así como a otros problemas relacionados, como es el cálculo de valores singulares, la resolución de ecuaciones matriciales, o el producto de una función matricial por un vector.

El trabajo realizado en esta tesis contribuye al desarrollo de SLEPc en el ámbito del problema polinómico de valores propios. En ella se estudia dicho problema y se aborda la implementación software de varios métodos para su resolución numérica. También contribuye al campo de los problemas no lineales, mediante la implementación de métodos que de alguna forma se relacionan con el problema polinómico.

1.1. Problema polinómico de valores propios

En esta tesis se estudia la resolución numérica del problema polinómico de valores propios (PEP, del inglés *polynomial eigenvalue problem*), el cual se formula en términos de la ecuación

$$P(\lambda)x = 0, \tag{1.2}$$

en la que $x \in \mathbb{C}^n$, con $x \neq 0$, y $P(\lambda)$ representa un polinomio matricial en el parámetro $\lambda \in \mathbb{C}$,

$$P(\lambda) = A_0 + \lambda A_1 + \lambda^2 A_2 + \cdots + \lambda^d A_d, \tag{1.3}$$

donde $A_i \in \mathbb{C}^{n\times n}$, para $i = 0, \ldots, d$. Los nombres *polinomio matricial*, *matriz polinómica* o λ-*matriz* son denominaciones sinónimas para (1.3). Mientras que la primera expresión hace referencia a que se trata de un polinomio en un determinado parámetro, cuyos coeficientes son matrices, la segunda enfatiza el hecho de que también se trata de una matriz cuyas entradas son polinomios, y la última destaca la dependencia en el parámetro λ. Un estudio detallado sobre polinomios matriciales y sus propiedades puede encontrarse en [31].

En lo que sigue, siempre se considerará que el polinomio matricial, $P(\lambda)$, que define un PEP, es *regular*, lo que significa que el determinante de la matriz polinómica asociada no es el polinomio nulo, es decir, $\det(P(\lambda)) \neq 0$ para algún $\lambda \in \mathbb{C}$.

Un par $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C}$ verificando (1.2) se dice un *par propio* del polinomio matricial (1.3). De forma equivalente se dice que x es un *vector propio* (o autovector) derecho asociado al *valor propio* λ (o autovalor). En este mismo contexto, se denomina vector propio izquierdo de (1.3), asociado al autovalor λ, a un vector no nulo $y \in \mathbb{C}^n$ verificando la ecuación

$$y^* P(\lambda) = 0, \tag{1.4}$$

donde se utiliza asterisco $*$ para denotar la matriz traspuesta conjugada de una matriz dada (o vector).

Definiendo el grado de un polinomio matricial (1.3) como el máximo valor $d \in \mathbb{N}$, tal que A_d no es cero, se tiene que el problema generalizado de valores propios

$$Ax = \lambda Bx, \tag{1.5}$$

con $A, B \in \mathbb{C}^{n\times n}$, es un caso particular de (1.2) con grado uno (problema lineal). Si además B es la matriz identidad se tiene el problema estándar (1.1). También, cuando el grado del polinomio matricial es 2, el problema (1.2) se dice *cuadrático*.

Los valores propios de (1.3) se corresponden con las raíces del polinomio $\det(P(\lambda))$. Para cada uno de ellos se define la *multiplicidad algebraica* como su multiplicidad como raíz de $\det(P(\lambda))$, y se dice que un autovalor es *simple* si su multiplicidad es 1. En un problema de dimensión n y grado d, si A_d en (1.3) es no singular, se tiene que existen dn autovalores de la matriz polinómica (contando las multiplicidades). En el caso de que A_d sea singular, se dice que $P(\lambda)$ tiene *valores propios en el infinito*, ver [55]. La multiplicidad del valor propio infinito se define en función de la multiplicidad del valor propio cero en el polinomio matricial *reverso* de P, definido por

$$\mathrm{rev}(P(\lambda)) := \lambda^d P(1/\lambda). \tag{1.6}$$

Una consecuencia del hecho de que una matriz polinómica tenga dn autovalores, y que marca una diferencia notable con el problema lineal, es que, aún en el caso de que los autovalores de un determinado PEP sean todos distintos, si consideramos dn autovectores asociados a dichos autovalores, éstos no formarán un conjunto linealmente independiente.

Aplicaciones que plantean PEPs

Resolución de sistemas de ecuaciones diferenciales de coeficientes constantes. La necesidad de resolución de un PEP aparece, de forma general, en aplicaciones donde se formula un sistema de ecuaciones diferenciales ordinarias, de orden $d > 1$ y coeficientes constantes, del tipo

$$A_0 + A_1\frac{\mathrm{d}u(t)}{\mathrm{d}t} + A_2\frac{\mathrm{d}^2u(t)}{\mathrm{d}t^2} + \cdots + A_d\frac{\mathrm{d}^du(t)}{\mathrm{d}t^d} = f(t),$$

para una función $u : \mathbb{R} \to \mathbb{C}^n$. La búsqueda de soluciones del tipo $u(t) := x\mathrm{e}^{\lambda t}$, con $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C}$, para la ecuación homogénea ($f = 0$), plantea la resolución de un PEP en la forma (1.3). Por ejemplo, el análisis de vibraciones de sistemas como edificios, vehículos o máquinas, da lugar a una ecuación diferencial ordinaria de segundo orden con coeficientes constantes [54], que origina un PEP cuadrático, cuyos valores y vectores propios dan lugar, respectivamente, a las llamadas frecuencias y modos naturales de vibración. A modo de ejemplo, consideramos el caso particular, con $n = 1$, de un cuerpo de masa M que se mueve, sobre ruedas, sujeto por la izquierda a un muelle elástico y a un amortiguador que se muestra en la Figura 1.1.

Denotando por $f(t)$ una fuerza externa que tira del cuerpo hacia la derecha, se tiene que las fuerzas que influyen en el movimiento, denotado por $u(t)$, del cuerpo verifican la ecuación

$$M\ddot{u} + C\dot{u} + Ku = f, \tag{1.7}$$

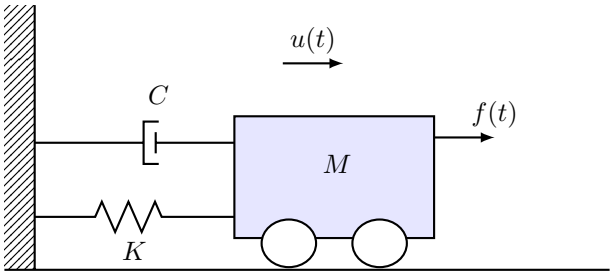

Figura 1.1: Cuerpo de masa M sujeto a un muelle y a un amortiguador que se mueve por acción de una fuerza f.

donde K es la constante de rigidez del muelle (resistencia estática al movimiento), y $C\dot{u}$ representa la amortiguación (o pérdida dinámica de energía del sistema). Una solución de la ecuación homogénea para (1.7) es de la forma $u(t) = \alpha_1 e^{\lambda_1 t} + \alpha_2 e^{\lambda_2 t}$, donde λ_1 y λ_2 son soluciones de la ecuación característica $Mr^2 + Cr + K = 0$, y α_1, α_2 son constantes a determinar por las condiciones iniciales. Al considerar sistemas con más de un cuerpo, o discretizaciones, por ejemplo, mediante el método de elementos finitos, de operadores en espacios de dimensión infinita, se obtiene una ecuación similar a (1.7) en la que M, K y C son matrices de dimensión n, el número de cuerpos o de elementos en la discretización. Suponiendo que los autovalores de $\lambda^2 M + \lambda C + K$, denotados por $\{\lambda_i\}_{i=1}^{2n}$, son distintos, una solución general, en este caso, tiene la forma

$$u(t) = \sum_{i=1}^{2n} \alpha_i x_i e^{\lambda_i t},$$

donde x_i representa el vector propio asociado a λ_i [89].

Resolución del problema no lineal de valores propios mediante interpolación polinómica. El problema no lineal de valores propios (NEP, del inglés *nonlinear eigenvalue problem*) plantea la resolución de una ecuación del tipo

$$T(\lambda)x = 0, \tag{1.8}$$

para $x \in \mathbb{C}^n \setminus \{0\}$, y $T : \Omega \to \mathbb{C}^{n\times n}$ con $\Omega \subset \mathbb{C}$. Un par $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C}$ que satisface (1.8) se denomina par propio de T. De forma equivalente también se dice que x y λ son, respectivamente, un vector y un valor propio de T. Como ejemplos de NEPs se tienen los problemas polinómicos (1.2), en los que $T = P$. En ocasiones, la resolución de un problema no lineal (1.8) se aborda aproximando el operador inicial T mediante un polinomio P que lo interpola en determinados puntos, y resolviendo el PEP asociado a dicho polinomio [28]. En general, la expresión más conveniente para el polinomio interpolador no es siempre del tipo (1.3), sino que a veces (especialmente si P tiene grado elevado) es preferible expresar P en función de una base de polinomios distinta de la de monomios. Así, por ejemplo, si se utilizan los nodos

de Chebyshev para la interpolación, es más conveniente utilizar una expresión de P en función de los polinomios de Chebyshev. En términos más generales, si el operador T admite una expresión en forma de desarrollo en serie de una determinada familia de polinomios, puede ser de utilidad disponer de métodos para la resolución de PEPs que puedan trabajar directamente sobre la aproximación del operador proporcionada por una suma parcial del desarrollo. En esta tesis, se considerará que el polinomio que define un PEP puede estar expresado en la forma genérica

$$P(\lambda) = \Phi_0(\lambda)A_0 + \Phi_1(\lambda)A_1 + \cdots + \Phi_d(\lambda)A_d, \tag{1.9}$$

en función de una base $\{\Phi_j\}_{j=0}^{d}$ de polinomios con coeficientes reales en la que Φ_j es de grado j, para $j = 0, \ldots, d$. En particular se tendrán en cuenta familias de polinomios ortogonales, como son, por ejemplo, las bases de polinomios de Chebyshev, Legendre, Laguerre o Hermite.

Las aplicaciones que plantean un NEP que se resuelve mediante interpolación polinómica constituyen importantes ejemplos de aplicación del problema polinómico de valores propios. Estos casos, son además ejemplos que ilustran la necesidad de métodos y solucionadores (*solvers*) polinómicos que puedan abordar problemas expresados en la forma más genérica (1.9).

1.2. Métodos generales para la resolución de PEPs

A la hora de afrontar la resolución numérica de problemas polinómicos de valores propios, encontramos en la bibliografía diversos métodos que son, en general, adaptaciones al caso polinómico de métodos clásicos desarrollados para el problema estándar o el generalizado. Cada uno de estos métodos presenta ventajas e inconvenientes, especialmente cuando éstos se estudian bajo el punto de vista computacional y en el contexto de problemas de gran dimensión. A continuación se revisan brevemente algunos métodos considerados, haciendo hincapié en la problemática que éstos presentan cuando son considerados para realizar una implementación software.

1.2.1. Linealización

La forma más habitual de resolver un PEP es la de plantear un problema de autovalores generalizado, $(A - \lambda B)z = 0$, en cierta forma equivalente, que tenga los mismos valores propios que el problema polinómico inicial. Un polinomio matricial del tipo

$$L(\lambda) = A - \lambda B, \tag{1.10}$$

denominado un *haz de matrices*, se dice que es una *linealización* de $P(\lambda)$ si existen matrices polinómicas $U(\lambda)$ y $V(\lambda)$ de determinante constante no nulo (*unimodulares*), de forma que

$$U(\lambda)L(\lambda)V(\lambda) = \begin{bmatrix} P(\lambda) & 0 \\ 0 & I_{(d-1)n} \end{bmatrix}, \tag{1.11}$$

donde $I_{(d-1)n}$ es la matriz identidad de dimensión $(d-1)n$. La ventaja de trabajar con transformaciones unimodulares es que éstas preservan la estructura espectral

finita, esto es, el conjunto de autovalores finitos (incluidas las multiplicidades). Un caso particular de linealizaciones son las denominadas *linealizaciones fuertes*, en la cuales $L(\lambda)$ y $\mathrm{rev}(L(\lambda))$ son linealizaciones de $P(\lambda)$ y $\mathrm{rev}(P(\lambda))$, respectivamente. Estas últimas tienen la característica de preservar también la estructura espectral cuando existen valores propios en el infinito [31].

Una vez se ha obtenido una linealización para un PEP concreto, es posible obtener los valores propios del problema polinómico calculando los autovalores de la linealización mediante cualquiera de los método existentes para problemas lineales. En [62] se prueba que, asociado a un polinomio matricial, existe un extenso espacio de linealizaciones que comparten el mismo espectro y que presentan diferentes características, por ejemplo, en cuanto al condicionamiento de sus autovalores[1] [44]. Dicha variedad de linealizaciones también ofrece gran flexibilidad a la hora de escoger una linealización que preserve propiedades estructurales, por ejemplo, en PEPs simétricos o hamiltonianos [70, 61, 67].

En el contexto de problemas de gran dimensión, el principal inconveniente de abordar un PEP mediante linealización es el aumento en la dimensión del problema a resolver, que pasa a ser d veces la dimensión del problema inicial, siendo d el grado del polinomio matricial asociado. Este hecho hace que, desde un punto de vista computacional, tenga especial relevancia el desarrollo e implementación de métodos específicos que, teniendo en cuenta la especial estructura de los problemas lineales que se generan, obtienen mejoras en cuanto a la utilización de memoria y coste computacional [66, 53, 85, 59].

1.2.2. Métodos de tipo Newton

El problema de valores propios asociado a (1.3) puede verse como un sistema de n ecuaciones y $(n{+}1)$ incógnitas, en las variables $(x, \lambda) \in \mathbb{C}^n \times \mathbb{C} \cong \mathbb{C}^{n+1}$, siendo n la dimensión de $P(\lambda)$. Añadiendo una ecuación adicional, $w^*x = 1$ (con $w \in \mathbb{C}^n$ fijo), que imponga una condición de normalización para el vector propio, y definiendo la función

$$F\left(\begin{bmatrix} x \\ \lambda \end{bmatrix}\right) = \begin{bmatrix} P(\lambda)x \\ w^*x - 1 \end{bmatrix}, \tag{1.12}$$

la resolución del PEP (1.2) se puede abordar mediante el cálculo de las raíces de la función F utilizando el conocido método de Newton. Bajo determinadas asunciones sobre el jacobiano de la función F, que en un punto $z = \left[\begin{smallmatrix} x \\ \lambda \end{smallmatrix}\right]$ tiene la forma

$$J(z) = \begin{bmatrix} P(\lambda) & P'(\lambda)x \\ w^* & 0 \end{bmatrix}, \tag{1.13}$$

dado un cero $\tilde{z}$ de F para el que $J(\tilde{z})$ es invertible, partiendo de un valor inicial z_0 suficientemente cercano a $\tilde{z}$, el método de Newton genera, iterativamente, una sucesión de valores $\{z_k\}_{k=0}^{\infty}$, dados por la expresión

$$z_{k+1} = z_k - [J(z_k)]^{-1} F(z_k), \tag{1.14}$$

[1] El condicionamiento de un problema hace referencia a la sensibilidad de la solución frente a perturbaciones en los datos de entrada del mismo [40]

que converge cuadráticamente al cero $\tilde{z}$, que determina un par propio de (1.2).

El método de la *iteración inversa*, generalizado para problemas polinómicos, se deriva de las ecuaciones (1.13) y (1.14), de las que se deduce que la dirección del nuevo vector aproximación es

$$v_{k+1} := P(\lambda_k)^{-1} P'(\lambda_k) x_k. \tag{1.15}$$

Escogiendo w inicialmente de forma que $w^* x_0 = 1$, se obtienen las aproximaciones que dicho método genera,

$$\lambda_{k+1} = \lambda_k - \frac{w^* x_k}{w^* v_{k+1}}, \tag{1.16}$$

$$x_{k+1} = v_{k+1}/(w^* v_{k+1}). \tag{1.17}$$

Una variación de este método se obtiene sustituyendo el vector de normalización w, a cada paso de la iteración inversa, por $w_k := P(\lambda_k)^* y_k$, siendo y_k una aproximación al vector propio izquierdo asociado a $\tilde{\lambda}$, de donde se obtiene

$$\lambda_{k+1} = \lambda_k - \frac{y_k^* P(\lambda_k) x_k}{y_k^* P'(\lambda_k) x_k}, \tag{1.18}$$

es decir, en cada iteración se toma como aproximación al autovalor el *cociente de Rayleigh generalizado* [54].

Un inconveniente que presenta el método de Newton es que necesita una aproximación inicial a un par propio para asegurar la convergencia. Este hecho hace que dicho enfoque se utilice principalmente para mejorar la exactitud de valores propios previamente calculados por algún otro método o, por ejemplo, en aplicaciones que necesitan resolver una secuencia de PEPs en los que la solución de cada uno de los problemas varía ligeramente con respecto del anterior.

En el caso de que se quieran obtener (o refinar) varios pares propios, en los que los autovalores estén muy próximos unos de otros, este enfoque puede tener dificultades en cuanto a la correcta dirección de convergencia, pudiendo converger repetidas veces al mismo par. Existen otras versiones más elaboradas que evitan dicho problema, planteando la ecuación a resolver en función de pares de la forma $(X, \Lambda) \in \mathbb{C}^{n\times k} \times \mathbb{C}^{k\times k}$, que posibilitan la obtención de k pares propios a la vez [51].

En términos computacionales, el refinamiento de pares propios utilizando el método de Newton supone muchas veces un coste incluso superior al de la obtención de la aproximación inicial. Esto es debido a que resuelve un sistema de ecuaciones por cada par propio y por cada iteración del método de Newton, además de que se utiliza una matriz diferente (que posiblemente hay que factorizar) en cada una de dichas resoluciones. El reto a la hora de incorporar este tipo de técnicas en una implementación con la que tratar problemas de gran dimensión, es buscar la forma de reducir el tiempo requerido por tales resoluciones, por ejemplo introduciendo algún nivel de paralelismo adicional, o buscando métodos específicos que tengan en cuenta la forma del sistema a resolver.

Jacobi–Davidson

Relacionado con el método de Newton, el método Jacobi-Davidson [82] es un método de proyección en el que se genera un espacio de búsqueda utilizando una ecuación de corrección tipo Newton. Este método, desarrollado inicialmente para la resolución de problemas lineales, ha sido extendido para problemas polinómicos y no lineales [81, 16, 91].

En el caso más general, los métodos de proyección [77, Sec. 4.3] buscan aproximaciones a autovectores en un determinado subespacio, llamado de búsqueda, imponiendo la condición de Petrov–Galerkin, la cual exige que el residuo asociado a dicho vector sea ortogonal a un determinado subespacio, denominado de test. Es decir, representando las bases de los espacios de búsqueda y test mediante dos matrices V y $W \subset \mathbb{C}^{n\times k}$, respectivamente, y expresando un vector de V en la forma $u = Vy$, la condición de Petrov–Galerkin para (1.2) se expresa mediante

$$W^*P(\lambda)Vy = 0, \tag{1.19}$$

para $\lambda \in \mathbb{C}$. La técnica de proyección utiliza pares propios del problema de dimensión reducida (1.19) (problema proyectado), para generar aproximaciones a pares propios, en las que el autovector está contenido en el subespacio de búsqueda V.

El método de Jacobi–Davidson extiende el subespacio de búsqueda V, en caso de que al aplicar la técnica de proyección (1.19) se obtenga una aproximación (u, μ) que no cumpla con la tolerancia deseada. Para ello resuelve la ecuación

$$\left(I - \frac{P'(\mu)uw^*}{w^*P'(\mu)u}\right) P(\mu)(I - uu^*)t = -P(\mu)u, \tag{1.20}$$

con la que obtiene un vector $t \perp u$ con el que amplía el subespacio de búsqueda e inicia un nuevo proceso de extracción (1.19). La ecuación (1.20) surge [80] al utilizar la iteración de Newton (1.14), a partir de (u, μ), considerando que el nuevo vector x se expresa en la forma $x = u + t$, con $t \perp u$. A partir de la primera de las dos ecuaciones que se obtienen al sustituir en (1.14) las expresiones (1.12) y (1.13), denotando $\Delta\lambda = \lambda - \mu$, se tiene

$$P(\mu)t - P'(\mu)u\Delta\lambda = -P(\mu)u. \tag{1.21}$$

Considerando un vector w ortogonal al residuo, $w^*P(\mu)u = 0$, y multiplicando la ecuación anterior por w^*, se obtiene una expresión para $\Delta\lambda$, que al ser sustituida en (1.21), y dado que $t = (I - uu^*)t$, produce la ecuación de corrección del método de Jacobi–Davidson (1.20). Una de las opciones que con más frecuencia se escogen para el subespacio de test [16, 91, 64, 48, 46] es la de hacerlo coincidir con el de búsqueda (proyección ortogonal). Otras opciones para W dan distintas propiedades de convergencia [27, 83, 81].

Una de las ventajas de este método es que no requiere demasiada exactitud en la resolución de la ecuación de corrección, la cual puede ser resuelta utilizando métodos iterativos precondicionados. Esta característica hace que sea atractivo cuando se abordan problemas de gran dimensión en los que no es factible la utilización

de métodos directos para la resolución de sistemas lineales. En dicho contexto, el método Jacobi–Davidson es también atractivo porque, a diferencia de los métodos basados en linealización, trabaja siempre manteniendo la misma dimensión del problema original. Por otro lado, el hecho de ser un método de proyección que mantiene aproximaciones a autovectores en el subespacio de búsqueda, hace que éste tenga mejores posibilidades de convergencia que el método de Newton. Como principal inconveniente, se puede destacar que la convergencia de los pares propios es uno a uno, haciendo que este método no sea competitivo (en tiempo de cálculo) para casos en los que es viable la utilización de otros métodos de proyección en los que es posible la convergencia a varios pares propios simultáneamente (por ejemplo cuando se utilizan métodos de Krylov sobre la linealización).

Es posible encontrar implementaciones software en entornos de computación paralela del método de Jacobi–Davidson para problemas de valores propios lineales [73]. En el caso de problemas polinómicos, encontramos artículos en los que este método se ha utilizado para la resolución de PEPs en aplicaciones concretas [64, 48, 46].

Un aspecto a tener en cuenta al realizar una implementación de este método para PEP, es la deflación de los autovectores ya convergidos. Dicho aspecto, no trivial, ha sido recientemente estudiado en [27]. Mientras que en el caso lineal la reconvergencia hacia autovectores ya calculados se consigue trabajando en el complemento ortogonal de éstos, en el caso no lineal, esta medida, aunque es la que habitualmente se utiliza, impide la obtención de varios pares propios que comparten el mismo autovector.

1.2.3. Integral de contorno

Los métodos de *integral de contorno* tienen en común la utilización de una integral compleja de línea, sobre una curva (de Jordan) Γ que determina una región $\Sigma \subset \mathbb{C}$ que contiene un conjunto de autovalores $\{\lambda_i\}_{i=1}^k$ que se desea calcular.

Los métodos descritos en [17, 6] para la resolución del problema no lineal (1.8), para $T : \Omega \subset \mathbb{C} \to \mathbb{C}^{n\times n}$ holomorfa, se basan en que la función $z \rightsquigarrow T(z)^{-1}$ es analítica en Ω, salvo en el conjunto de valores propios, y en que en un entorno de cada uno de estos puntos, la no linealidad de dicha función se puede expresar en función de los vectores propios del autovalor (ver por ejemplo [17]). Suponiendo que los autovalores en Σ son simples, se tiene la siguiente expresión para $T(z)^{-1}$,

$$T(z)^{-1} = \sum_{\lambda_i \in \Sigma} \frac{v_i w_i^*}{z - \lambda_i} + R(z), \quad z \in \Sigma, \tag{1.22}$$

donde R es una función analítica, y v_i, w_i son los autovectores derecho e izquierdo de λ_i, respectivamente, normalizados de forma que $w_i^* T'(\lambda_i) v_i = 1$, para $i = 1, \ldots, k$. En este caso, también se cumple [17] que

$$\frac{1}{2\pi \mathrm{i}} \oint_\Gamma z^m T(z)^{-1} \mathrm{d}z = \sum_{i=1}^{k} \lambda_i^m v_i w_i^*, \tag{1.23}$$

para todo $m \in \mathbb{N}$. Calculando la integral de línea numéricamente, a partir de un conjunto de puntos $\{z_j\}_{j=1}^N$ de cuadratura en Γ y pesos $\{w_j\}_{j=1}^N$, la ecuación anterior

queda

$$\sum_{i=1}^{k} \lambda_i^m v_i w_i^* = \sum_{j=1}^{N} w_j z_j^m T(z_j)^{-1}. \tag{1.24}$$

A partir de (1.24), es posible extraer información sobre los autovectores. Por ejemplo, en [94] aplican ambos lados de la ecuación (1.24) sobre vectores aleatorios (variando el valor de m) para generar un subespacio a partir del cual extraen los pares propios mediante proyección ortogonal.

Una cuestión que se plantea para estos métodos es cómo determinar el número de autovalores, k, en la región Σ. En [63] se propone una estimación basada en la aproximación numérica de la integral de contorno

$$k = \frac{1}{2\pi \mathrm{i}} \oint_{\Gamma} \mathrm{tr}(T(z)^{-1} T'(z)) \mathrm{d}z,$$

donde dada una matriz M, el operador tr(M) representa la traza de la matriz.

Este tipo de métodos no son objeto de estudio en esta tesis. SLEPc incluye implementaciones de métodos que siguen este esquema, basados en [78, 79, 6, 63], para la resolución de problemas lineales y para no lineales. El problema polinómico puede también ser resuelto en SLEPc mediante integral de contorno, como caso particular del problema no lineal.

1.3. Software científico para la resolución de PEPs

Existen en la actualidad diversas bibliotecas software de libre distribución, dedicadas a la resolución, en paralelo, de problemas dispersos de valores propios lineales, tanto en su forma estándar (1.1), como generalizada (1.5). Entre estas bibliotecas podemos encontrar variedad de métodos para la resolución del problema lineal. Sin embargo, en el caso de problemas polinómicos no encontramos, a excepción de SLEPc, ninguna biblioteca software que aborde dichos problemas desde el marco de la computación paralela.

Para dimensiones pequeñas o moderadas, encontramos alguna implementación software de libre distribución para la resolución de problemas cuadráticos o no lineales, en lenguaje MATLAB. Este es el caso de QUADEIG, descrito en [37], que permite el cálculo de los valores propios y de los autovectores derechos e izquierdos, así como el número de condición de los autovalores en problemas cuadráticos densos. Sigue un enfoque en el que se construye una linealización que se selecciona teniendo en cuenta el condicionamiento del problema resultante, y se resuelve mediante un método directo (*iterativo QZ*) que limita su aplicación a problemas de tamaño medio. También, QARPACK, descrito en [12], aborda la resolución de problemas dispersos no lineales hermíticos que verifican la propiedad de que sus valores propios están caracterizados mediante un principio variacional. Utiliza dicha característica para inducir una ordenación entre los autovalores de forma que le permite obtener los autovalores cuya parte real está contenida en un determinado intervalo. El enfoque que utiliza es principalmente de tipo Jacobi–Davidson.

La falta de sofware específico hace que los investigadores de distintos campos aborden a menudo la resolución numérica de problemas de valores propios polinómicos, mediante la formación explícita de las matrices asociadas a una linealización del PEP, para poder así utilizar alguna de las bibliotecas software para cálculo de valores propios disponible. Sin embargo, el incremento de la dimensión del problema, puede hacer que esa opción no siempre sea viable, especialmente en problemas de gran dimensión o grado elevado del polinomio que define el PEP. En cualquier caso, aún siendo viable, dicha opción siempre supone un sobrecoste considerable, tanto de cómputo como de memoria. Otras veces, las características del problema hacen que sea más conveniente abordar la resolución del PEP directamente, sin recurrir a una linealización; por ejemplo, mediante un método tipo Jacobi–Davidson. Estas consideraciones ponen de manifiesto la necesidad y utilidad de un desarrollo software específico con el que los científicos puedan resolver estos problemas. Con el fin de dar cobertura a la resolución del problema polinómico de valores propios, la biblioteca SLEPc dispone, desde la versión 3.1 (año 2010) de un módulo para la resolución del problema cuadrático de valores propios, y desde la versión 3.5 en 2014, incluye un nuevo módulo que engloba de forma más general la funcionalidad específica para la resolución del problema polinómico de valores propios. En dicho módulo, y como fruto de esta tesis, se han incluido diversos métodos y opciones de resolución, con las que de forma que sea posible abordar problemas de aplicación con distintas características.

1.4. La biblioteca SLEPc

La biblioteca SLEPc se basa en la biblioteca computacional PETSc (Portable, Extensible Toolkit for Scientific Computación [10]), la cual proporciona diversas herramientas software destinadas principalmente a la resolución numérica, en paralelo, de ecuaciones en derivadas parciales que forman parte de aplicaciones científicas de gran escala. Se estructura como un conjunto de componentes con los que el usuario interactúa a través de la interfaz de éstos, pudiéndose abstraer, en gran medida, del paralelismo subyacente. Incluye objetos para el manejo (en paralelo) de determinadas estructuras de datos como vectores o matrices, además de objetos solucionadores que encapsulan rutinas o métodos específicos para el tratamiento de varios tipos de problemas como la resolución de sistemas de ecuaciones (lineales y no-lineales), o la integración de ecuaciones diferenciales. SLEPc extiende la funcionalidad de la biblioteca PETSc, proporcionando varios componentes para el tratamiento y resolución del problema de valores propios. PETSc y SLEPc han sido diseñadas sobre un paradigma de memoria distribuida, para ser eficientes en sistemas de computación de alto rendimiento, escalables a un número elevado de procesadores y portables a la mayoría de arquitecturas paralelas. Ambas bibliotecas proporcionan gran flexibilidad al especificar parámetros relevantes para la resolución, que pueden ser proporcionados en tiempo de ejecución sin tener que recompilar el código de aplicación. A continuación se describen brevemente algunos aspectos de la biblioteca PETSc que pueden ayudar a comprender algunas de las decisiones de diseño llevadas a cabo, así

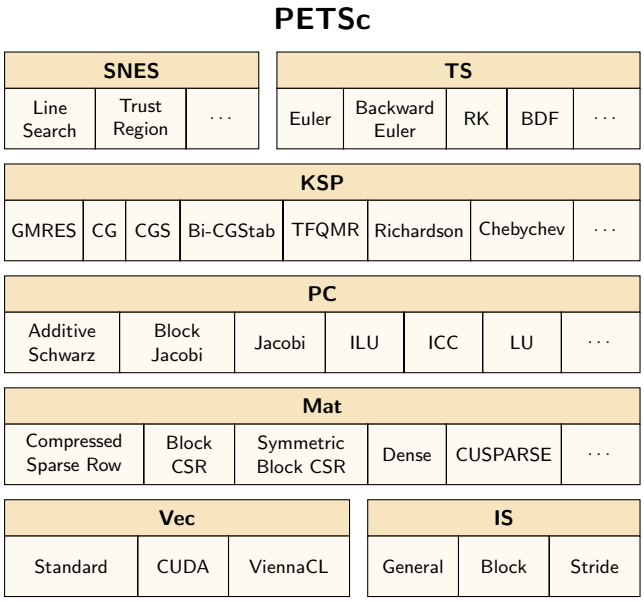

Figura 1.2: Estructura de clases de PETSc.

como algunos de los detalles de implementación que se proporcionan en esta tesis. También se incluye una descripción de los principales elementos u objetos en los que se estructura SLEPc.

1.4.1. Características de la biblioteca PETSc

La biblioteca PETSc está escrita en lenguaje C (utilizable directamente desde C++), aunque también dispone de interfaces para Fortran 77/90 y Python [24]. PETSc ha sido diseñada siguiendo un modelo de programación de paso de mensajes, para ser ejecutada principalmente en máquinas paralelas de memoria distribuida. Utiliza el estándar MPI (*Message Passing Interface*) [68] para la comunicación entre los procesos, lo que la hace portable a casi cualquier arquitectura paralela. Dicha comunicación es gestionada de forma interna por la biblioteca, aunque de un modo flexible, permitiendo la implicación de los usuarios en el proceso, y posibilitando que el programador no avanzado pueda evitar el uso explícito de las funciones de comunicación y abstraerse de ésta.

PETSc define y utiliza un tipo neutro de datos (denominados *escalares*), que permite compilar la biblioteca determinando tanto la aritmética a utilizar, que puede ser real o compleja (no ambas simultáneamente), como la precisión, que puede ser simple, doble o cuádruple.

A pesar de utilizar un lenguaje sin soporte explícito para ello, PETSc ha sido diseñada siguiendo un modelo de programación orientado a objetos, de modo que toda la funcionalidad que ofrece queda encapsulada en clases abstractas con las que

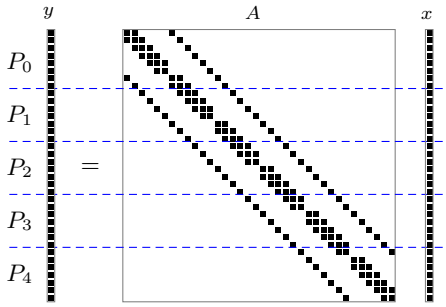

Figura 1.3: Distribución paralela para los datos de una matriz y dos vectores PETSc involucrados en un producto matriz-vector.

el programador puede trabajar sin necesidad de conocer las estructuras de datos subyacentes. Presenta una estructura jerárquica en la que, partiendo de objetos más básicos como los vectores o matrices, se crean otros más complejos, los objetos *solucionadores* o *solvers*, que se apoyan en los primeros y representan abstracciones de conjuntos de métodos para la resolución de determinados problemas. La figura 1.2 muestra la jerarquía de clases en que se estructura la biblioteca PETSc, así como algunas de las subclases de cada una de las clases principales, las cuales se corresponden con diferentes implementaciones de los métodos asociados a la interfaz de una clase particular. En el caso de matrices, cada subclase se asocia, por ejemplo, a un formato diferente de almacenamiento de los datos, mientras que en el caso de objetos solucionadores, cada subclase se corresponde con un método distinto de resolución. El polimorfismo de las clases que forman PETSc permite que parámetros relevantes como puede ser, por ejemplo, el método de resolución a utilizar por un determinado solver, sea especificado en tiempo de ejecución (mediante opciones por línea de comando), ya que la llamada a una función de interfaz de una clase determinada se traduce, durante la ejecución, en la llamada a la rutina específica de la subclase asociada a la instancia creada. Utilizando este enfoque, PETSc engloba también dentro de la misma estructura de clases, implementaciones provenientes de bibliotecas externas.

En las implementaciones realizadas en esta tesis se han utilizado, principalmente, las clases más básicas de manejo de estructuras de datos, que son, las matrices (`Mat`), los vectores (`Vec`) y los conjuntos de índices (`IS`), así como los objetos asociados a la resolución de sistemas de ecuaciones lineales, dados por los métodos iterativos de Krylov (`KSP`) y los precondicionadores (`PC`). No se han utilizado, sin embargo, los asociados a los métodos de resolución de sistemas de ecuaciones no lineales (`SNES`) ni los integradores (`TS`). A continuación se dan algunos detalles sobre las clases y funcionalidad de PETSc utilizadas.

Matrices y vectores. El reparto de datos en una matriz paralela de PETSc se hace mediante bloques consecutivos de filas. La Figura 1.3 muestra el reparto de datos, entre 5 procesos, de una matriz y dos vectores involucrados en un

producto matriz-vector. El usuario no suele acceder directamente a la estructura paralela de datos incluida en estos objetos, sino que las operaciones con ellos se llevan a cabo a través de la funciones definidas en su interfaz, las cuales han sido optimizadas para ser eficientes en su ejecución paralela, mediante el solapamiento de cómputo y comunicación. Con el reparto de datos indicado en la Figura 1.3, la operación de suma de vectores es muy simple y no requiere comunicación entre los distintos procesos (cada uno de ellos realiza la suma con los datos locales), en el caso de matrices, dicha operación tampoco conlleva comunicación, y es igual de simple sólo si las matrices involucradas tienen la misma estructura de elementos no cero. El producto escalar de vectores requiere, aparte de los cálculos locales, una operación de reducción entre el conjunto de procesos para la suma de los resultados parciales. Un esquema similar siguen las operaciones para el cálculo de las p-normas vectoriales, la norma infinito matricial o la norma de Frobenius. 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, y recibiendo, mediante comunicación punto a punto, los elementos del vector x que necesita. PETSc optimiza esta operación (muy utilizada en los métodos iterativos incluidos en PETSc), al crear una matriz, mediante la formación, de forma transparente para el usuario, de las estructuras de datos de comunicación necesarias para que el envío de datos requerido para operar con dicha matriz sea eficiente. Por otro lado, PETSc consigue obtener eficiencia de cómputo dentro cada proceso MPI utilizando las bibliotecas BLAS [18] y LAPACK [4] para las operaciones de álgebra lineal que se efectúan a nivel local.

Resolución de sistemas de ecuaciones lineales. PETSc ofrece varios métodos iterativos para la resolución de sistemas de ecuaciones lineales, accesibles a través de la interfaz de la clase `KSP`. Dichos métodos, entre los que se incluyen, por ejemplo, el método del *gradiente conjugado* o el del residuo mínimo generalizado (CG y GMRES respectivamente, en la Figura 1.2), no modifican la matriz del sistema y basan la mayor parte del cálculo en la operación de multiplicación matriz-vector, y en operar con un precondicionador, o aproximación a la matriz inversa del sistema. PETSc engloba la funcionalidad de dichos precondicionadores en la clase `PC`, para la que proporciona gran variedad de implementaciones, por ejemplo, Jacobi, que utiliza la diagonal de la matriz del sistema para construir la aproximación. Los métodos directos que utilizan factorizaciones completas como LU o Cholesky, también se incluyen como precondicionadores, para ser utilizadas junto a un objeto `KSP` de un tipo especial que no hace referencia a ningún método concreto, sino que simplemente aplica el precondicionador. Para dar soporte a ese tipo de métodos (no iterativos), PETSc incluye, además, subclases que sirven de envoltorio para la utilización de implementaciones proporcionadas por bibliotecas externas, por ejemplo la biblioteca MUMPS [1].

Herramientas para el análisis de rendimiento y monitorización. Para la eva-

luación del software realizado en esta tesis, se han utilizado varias herramientas para el estudio del comportamiento de los programas, proporcionadas por PETSc. Esta biblioteca ofrece por un lado la posibilidad de realizar una recogida sistemática de información sobre la ejecución, que se muestra de forma resumida al final de la misma y que incluye registros sobre tiempos de ejecución (especificando número y duración de llamadas a determinadas rutinas), sobre la media de operaciones en coma flotante (FLOPs) realizados, y sobre la cantidad de memoria utilizada (especificando objetos creados). Además, dispone de varias rutinas, o monitores, con los que realizar un seguimiento del proceso de ejecución, para visualizar, por ejemplo, el error de la solución que se genera a cada paso del proceso iterativo.

1.4.2. Estructura de la biblioteca SLEPc

SLEPc mantiene el enfoque de diseño de PETSc, y con base en la infraestructura que esta biblioteca proporciona, extiende su funcionalidad añadiendo nuevas clases para la resolución de problemas relacionados con el cálculo de valores propios. La Figura 1.4 muestra la estructura de clases de SLEPc. Ésta incluye varios módulos de objetos solucionadores, encargados de dar soporte a la resolución del problema de valores propios, lineal (`EPS`), polinómico (`PEP`) y no lineal (`NEP`), además de otros problemas relacionados, como la descomposición en valores singulares (`SVD`) y el producto de una función de una matriz por un vector (`MFN`). Estas dos últimas clases no son objeto de estudio en esta tesis, mientras que el desarrollo de la clase `PEP` es el tema central de la misma. Para cada una de estas clases existen un conjunto de subclases en las que se implementan distintos métodos de resolución para el problema asociado. Así, por ejemplo, en la clase `EPS` se dispone de varios métodos de Krylov, además de métodos como Jacobi-Davidson, gradiente conjugado o integral de contorno, entre otros. Los métodos disponibles para la clase `PEP`, así como alguno de los existentes para `NEP` se describirán en profundidad a lo largo de la tesis. En el Apéndice A se dan detalles de la interfaz de usuario para la clase `PEP`, y de algunos ejemplos de las opciones disponibles.

Aparte de las clases principales mencionadas, existe un conjunto de clases auxiliares que se utilizan desde éstas, de las que se dan algunos detalles a continuación.

`ST` engloba la funcionalidad necesaria para el manejo de transformaciones espectrales, o cambios de variable en el parámetro λ asociado al autovalor. Algunas de las trasformaciones disponibles en la clase `ST` son la de *desplazamiento* (*shift*), que efectúa el cambio $\lambda = \theta + \sigma$ para un desplazamiento dado σ, o la de *desplazamiento e inversión* (*shift-and-invert*) dada por $\lambda = 1/\theta + \sigma$. Al aplicar dichas transformaciones se genera un nuevo problema con los mismos autovectores que el original y con autovalores modificados según la transformación empleada. Éstas pueden ser utilizadas junto con diversos métodos de resolución, como los basados en subespacios de Krylov, para los que su utilización tiene el efecto de influir en la convergencia de los mismos.

Algunas de las clases para cálculo de valores propios (`EPS` y `PEP`) incluyen in-

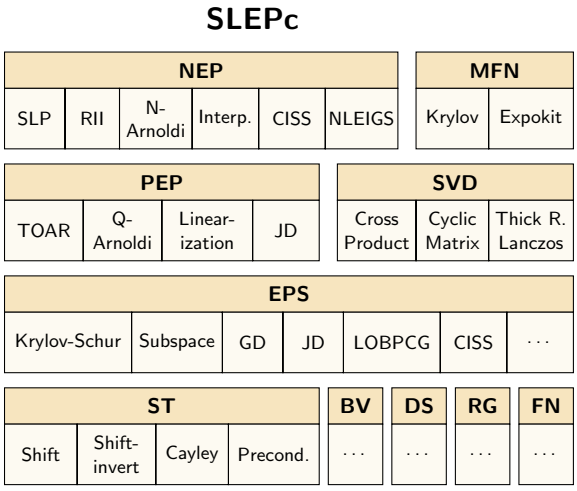

Figura 1.4: Componentes de SLEPc.

ternamente un objeto `ST` encargado de llevar a cabo las operaciones necesarias para la transformación del problema inicial según el cambio de variable seleccionado, incluyendo en ocasiones la formación explícita de las matrices del problema transformado, que quedan almacenadas en el objeto `ST`. Los solvers que utilizan un `ST` operan con las matrices del problema a través de las rutinas que ofrece esta clase, pudiendo así trabajar abstrayéndose de la transformación espectral utilizada. Así, por ejemplo, en la resolución de un problema estándar con la matriz A, la llamada a una rutina de `ST` que aplica el operador del problema sobre un vector x, se traduce internamente, en el producto $(A - \sigma I)x$ cuando se utiliza la transformación de shift, y en la operación $(A - \sigma I)^{-1}x$ al utilizar la de shift-and-invert. Esta última operación se lleva a cabo mediante la resolución de un sistema de ecuaciones lineales vía un objeto `KSP` de PETSc incluido en la estructura de datos de la clase `ST`.

El paralelismo en esta clase se realiza al nivel de los objetos paralelos que ésta incluye y de las operaciones que con éstos realiza.

`BV` clase utilizada principalmente para representar una base de vectores. En general, engloba estructuras de datos y rutinas para operar de forma eficiente con un conjunto de vectores. Las distintas implementaciones existentes en esta clase hacen referencia a la forma en que dichos vectores son almacenados. Éstas pueden ser, entre otras formas, un array de vectores PETSc, o una matriz PETSc en la que cada columna representa un vector.

La mayoría de clases solvers de SLEPc incluyen internamente un objeto `BV` para almacenar y trabajar con la base del subespacio que éstos generan. Esta clase proporciona funcionalidades relativas a la ortogonalización de vectores y está optimizada para que dichas operaciones sean eficientes en su ejecución,

minimizando la comunicación global [38] y favoreciendo en lo posible las operaciones a bloques frente a las vectoriales en los cálculos locales.

La clase `BV` utiliza por defecto Gram–Schmidt clásico iterado para la ortogonalización de los vectores, aunque dispone de varias opciones para dicha operación. Su diseño permite la utilización de un producto escalar no estándar, definido mediante una matriz hermítica definida positiva. También dispone de rutinas que trabajan con un *producto escalar indefinido* (forma cuadrática definida mediante una matriz hermítica no necesariamente definida positiva), para dar soporte a métodos que trabajan con estos *pseudo-productos*.

`DS` es la clase responsable de la resolución de problemas (normalmente de valores propios) asociados a los problemas proyectados de dimensión reducida, que la mayoría de solvers existentes en SLEPc generan. Éstos cálculos se llevan a cabo de forma redundante en todos los procesos, ya que los datos de los mismos están también en todos ellos. Los cálculos asociados se hacen de forma eficiente utilizando, cuando es posible, rutinas de las bibliotecas BLAS y LAPACK. Siempre que los problemas proyectados que se generen no sean demasiado grandes, la secuencialización que se introduce en los cálculos de este objeto no penaliza el rendimiento paralelo global.

Existe un tipo de `DS` por cada tipo de problema proyectado que es generado por alguna de las clases principales de SLEPc, por ejemplo, para la resolución del problema estándar simétrico, el polinómico, etc.

`RG` clase que define una región en el plano complejo. La mayoría de objetos solvers tienen internamente un objeto `RG`, que utiliza para especificar, por ejemplo, una zona exclusiva de búsqueda de autovalores. Los tipos implementados son intervalos (que incluyen los de 2 dimensiones), polígonos y elipses.

Este objeto no tiene estructuras de datos paralelas ni realiza operaciones de comunicación entre los procesos, sino que se replica en todos los procesos, sin que ello suponga penalización alguna en el rendimiento paralelo.

`FN` clase para la definición de funciones. Se utilizan, por ejemplo, para especificar un problema no lineal asociado a un objeto `NEP`, o para indicar la función matricial que opera en un objeto `MFN`. Existen en esta clase varias funciones predefinidas como, por ejemplo, las racionales (de las que los polinomios son un caso particular), la función exponencial, o la raíz cuadrada, además de varios tipos de combinaciones de éstas. En esta clase se incluyen rutinas para la evaluación en un punto tanto de las funciones asociadas, como de sus derivadas. Para la mayoría de la funciones predefinidas también permite su evaluación sobre una matriz. Esta clase tampoco utiliza estructuras de datos paralelas ni comunicación entre procesos.

La paralelización en las clases solvers de SLEPc se obtiene principalmente por el paralelismo presente en los objetos incluidos en éstas, es decir, en las operaciones con matrices y vectores, en la resolución de sistemas lineales con las matrices de los

precondicionadores y en las operaciones de ortogonalización dentro de las bases de vectores. Sin embargo, para mejorar la escalabilidad en algunos solvers, éstos utilizan también un segundo nivel de paralelismo, en el que se trabaja con varios grupos de comunicadores MPI.

1.5. Objetivos y estructura de la tesis

El objetivo primero de esta tesis, tal como se ha mencionado antes, es proporcionar cobertura a la resolución del problema polinómico de valores propios en SLEPc. Con anterioridad al inicio de esta tesis, existía ya la posibilidad de resolver problemas cuadráticos en SLEPc. El solver cuadrático inicial, posteriormente integrado en el solver polinómico, permitía la resolución de dichos problemas vía linealización, formando explícitamente las matrices del problema lineal asociado. También existía una implementación del método Q-Arnoldi [66], variante que resuelve el problema lineal asociado reduciendo los requerimientos de memoria necesarios. En esta tesis nos hemos propuesto, partiendo del estado inicial descrito, desarrollar nuevos solvers que permitan la resolución de problemas polinómicos, con grado mayor o igual que 2, expresados de forma genérica en función de bases polinómicas (1.9) que puedan ser distintas de la base de monomios. Se persigue, por un lado, la incorporación de un conjunto variado de métodos que proporcionen flexibilidad para abordar la resolución de problemas con diferentes características. Por otro lado, se pretende realizar la implementación de dichos métodos de modo que éstos sean robustos y presenten la máxima eficiencia computacional en entornos paralelos.

En esta tesis se describen cada uno de los métodos estudiados e implementados, haciendo hincapié en las dificultades que éstos han planteado, tanto desde un punto de vista numérico como computacional, y las soluciones que para cada uno de ellos se han propuesto.

En primer lugar, en el capítulo 2 se describen los solvers polinómicos basados en linealización. Tal como se ha apuntado en la breve descripción de dichos solvers (apartado 1.2.1), éstos plantean la problemática del incremento de la dimensión del problema que generan, por lo que se hace necesaria la incorporación de algoritmos específicos que buscan principalmente la reducción de la memoria necesaria. En el capítulo 3 se plantea la resolución de problemas cuadráticos simétricos de forma que se preserve la propiedad de simetría a lo largo de la resolución. Tal como se verá en dicho capítulo, la dificultad que aquí se plantea es de tipo numérico. En el capítulo 4 se describen solvers basados en el método de Newton. En primer lugar se aborda la incorporación del refinamiento iterativo de tipo Newton como una forma de mejorar la exactitud de los resultados cuando las características del problema lo requieren. El reto en este caso es la reducción del coste computacional. En segundo lugar se describe el solver basado en Jacobi–Davidson. En esta ocasión el objetivo ha sido buscar un planteamiento que permita la correcta deflación de los vectores convergidos. En el capítulo 5 se describen varias implementaciones realizadas para la solución del problema no lineal de valores propios. Dichas implementaciones están en cierta forma relacionadas con el trabajo llevado a cabo para problemas polinómicos. En la

sección 5.1 se describe el solver para no lineales basado en interpolación polinómica, mientras que en la sección 5.2 se presenta un solver basado en interpolación racional más linealización. Por último, en la sección 5.3, se describe la forma en que se realiza el refinamiento iterativo de tipo Newton de pares propios de problemas no lineales. Para finalizar, en el capítulo 6 se exponen las conclusiones y aportaciones de la tesis, y se incluyen algunas notas sobre líneas de trabajo futuro.

Capítulo 2

Resolución vía linealización de PEPs

Los problemas lineales de valores propios han sido estudiados, desde hace tiempo, por numerosos autores, y para su resolución existe una considerable variedad de métodos y bibliotecas software específicos. Por ello, puede resultar atractiva la idea de resolver un problema polinómico por medio del planteamiento y resolución de un problema lineal de alguna forma equivalente. La expresión de resolución vía linealización no hace referencia a la práctica, bastante habitual en computación científica, de aproximar el problema inicial mediante otro lineal, para, por medio de la resolución del problema lineal, obtener aproximaciones a las soluciones del problema original. En este caso, se trata de construir un problema lineal que tenga los mismos autovalores que el original. El precio a pagar no será de falta de exactitud, ya que no es una aproximación, sino de coste computacional, debido a que la dimensión del problema lineal que se genera se multiplica por el grado del polinomio matricial del cual se quieren obtener los autovalores. Aún así, la linealización sigue siendo una de las formas más utilizadas para la resolución de PEPs.

En este capítulo se describen los solvers polinómicos de SLEPc basados en linealización. Éstos, siguiendo el esquema descrito en la sección 1.2.1, utilizan y resuelven una linealización fuerte

$$L(\lambda) = A - \lambda B, \tag{2.1}$$

asociada al polinomio matricial $P(\lambda)$ que define el problema. En la sección 2.1.3 se proporcionan detalles de la linealización utilizada, la cual está basada en la descrita por Amiraslani y co-autores en [2, 3]. Si dicha linealización se forma explícitamente, será entonces posible utilizar para su resolución cualquiera de los solvers lineales disponibles en SLEPc, aunque con un alto coste en memoria si el grado de P es elevado. Para poder dar cobertura a aplicaciones donde los requerimientos de memoria puedan ser una limitación, se han incorporado métodos capaces de ahorrar memoria haciendo uso de la particular estructura a bloques de la linealización utilizada. Los desarrollos realizados se basan en los métodos Q-Arnoldi [66] y TOAR [85, 59], que

se describen en la sección 2.1.2, y en el trabajo [53], que presenta una extensión del método de TOAR para problemas polinómicos.

Los distintos solvers basados en linealización considerados se describen en la sección 2.2. Todos ellos son variantes del método de Krylov–Schur para problemas lineales, el cual es una versión con reinicio implícito del método de Arnoldi. La diferencia principal entre dichos solvers radica en la forma en que éstos representan los vectores de la base de Krylov que generan.

En la sección 2.3 se abordan cuestiones relacionadas con el reinicio del método de Arnoldi en las distintas variantes consideradas. En dicha sección se plantea y resuelve el problema de cómo conseguir, en la variante TOAR, la deflación de los vectores de Ritz convergidos, de forma que los vectores de la base de TOAR con los que éstos se representan no vuelvan a modificarse.

Otros aspectos relevantes que se abordan en este capítulo son, en la sección 2.4, la posibilidad de mejorar el condicionamiento de los autovalores de la linealización por medio del escalado de las matrices del problema polinómico [13], y en la sección 2.5, el modo de determinar los autovectores del problema polinómico a partir de los obtenidos para la linealización, para lo que se ha utilizado una adaptación del enfoque planteado por Betcke y Kressner en [15].

Finalmente, en la sección 2.6 se muestran los resultados numéricos de las pruebas realizadas, con varios casos de test, para evaluar la robustez y la eficiencia en paralelo de las implementaciones realizadas.

2.1. Resultados previos

2.1.1. Notación

Por simplicidad en la notación, en este capítulo, a la hora de describir los métodos involucrados en la resolución de una linealización (2.1) asociada a un PEP de dimensión n y grado d, consideraremos que un vector v de dimensión dn, o en general, una matriz $V \in \mathbb{C}^{dn\times k}$ están divididos en d bloques de n filas, en la forma,

$$v = \begin{bmatrix} v^0 \\ \vdots \\ v^{d-1} \end{bmatrix}, \qquad V = \begin{bmatrix} V^0 \\ \vdots \\ V^{d-1} \end{bmatrix}, \tag{2.2}$$

donde $v^i \in \mathbb{C}^n$ y $V^i \in \mathbb{C}^{n\times k}$ para $i = 0, \ldots, d-1$. Utilizaremos superíndices para denotar cada uno de los bloques de la forma dividida (2.2).

También utilizaremos las expresiones I_m y e_i para denotar, respectivamente, la matriz identidad de orden m, y el i-ésimo vector canónico (columna i de la matriz identidad), para el que se asume que su longitud es la que asegure la concordancia de las dimensiones de las expresiones en las que aparezca. Dado $u \in \mathbb{C}^n$, si no se especifica otra cosa, la expresión $\|u\|$ denotará la 2-norma de u. El producto de Kronecker de dos matrices A y B se denotará mediante la expresión $A \otimes B$. El operador vectorización se define como el que al actuar sobre una matriz $R \in \mathbb{C}^{p\times q}$

Algoritmo 2.1 Arnoldi

Entrada: Matriz $S \in \mathbb{C}^{n\times n}$ que define (2.3), vector inicial $v_1 \in \mathbb{C}^n$
Salida: Matrices $H_k \in \mathbb{C}^{k\times k}$, $V_{k+1} \in \mathbb{C}^{n\times(k+1)}$ y $h_{k+1,k} \in \mathbb{R}$ que satisfacen (2.5)

1: $v_1 = v_1/\|v_1\|$
2: $V_1 \leftarrow v_1$
3: **para** $j = 1, 2, \ldots, k$ **hacer**
4: $\quad w = Sv_j$
5: $\quad h_j = V_j^* w$
6: $\quad \tilde{w} = w - V_j h_j$
7: $\quad h_{j+1,j} = \|\tilde{w}\|$
8: $\quad v_{j+1} = \tilde{w}/h_{j+1,j}$
9: $\quad V_{j+1} \leftarrow \begin{bmatrix} V_j & v_{j+1} \end{bmatrix}$
10: $\quad H_{j+1} \leftarrow \begin{bmatrix} H_j & h_j \\ 0 & h_{j+1,j} \end{bmatrix}$
11: **fin para**

produce el vector $\operatorname{vec} R \in \mathbb{C}^{pq}$ cuyas entradas son las columnas de R colocadas en orden, empezando por la primera, una a continuación de la otra.

2.1.2. Métodos de Krylov para la resolución vía linealización de problemas cuadráticos

En esta sección se describen los métodos Q-Arnoldi [66] y TOAR [85, 59, 60], los cuales son variantes del método de Arnoldi, eficientes en términos de memoria, específicos para la resolución vía linealización de problemas cuadráticos.

El método de Arnoldi [8] utilizado para la resolución del problema de valores propios lineal estándar

$$Sz = \lambda z, \tag{2.3}$$

es un método de proyección ortogonal sobre un subespacio de Krylov, definido para $u \in \mathbb{C}^n$ y $k \in \mathbb{N}$ por

$$\mathcal{K}_k(S, u) = \operatorname{span}\{u, Su, \ldots, S^k u\}. \tag{2.4}$$

Partiendo de un vector inicial v_1 de norma 1, dicho método obtiene iterativamente, para $k = 2, 3 \ldots$, una base ortonormal $\{v_1, \ldots, v_{k+1}\}$ de $\mathcal{K}_k(S, v_1)$, y una matriz Hessenberg superior H_k, de forma que se verifica la relación de Arnoldi,

$$SV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^*, \tag{2.5}$$

en la que $V_k = \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix}$. En el Algoritmo 2.1 se muestran los pasos seguidos en el método de Arnoldi para generar la relación (2.5). Los principales pasos que se realizan son: la extensión del subespacio de Krylov (paso 4), la ortogonalización y normalización del nuevo vector (pasos 5, 6 y 8), y la actualización de la matriz H_k (paso 10) con los coeficientes obtenidos en los pasos de ortonormalización.

Utilizando el procedimiento descrito en (1.19), para el caso en el que los espacios de búsqueda y test coinciden (proyección ortogonal), el método de Arnoldi

obtiene aproximaciones a vectores propios, en el subespacio de Krylov $\mathcal{K}_k(S, v_1)$. Los pares propios (y, μ) de H_k generan vectores, $\hat{z} = V_k y$, y valores, μ, de Ritz, que proporcionan aproximaciones a los pares propios del problema original (2.3). La bondad de dichas aproximaciones puede ser evaluada utilizando la norma del residuo $r(\mu, \hat{z}) := S\hat{z} - \mu\hat{z}$, el cual puede ser obtenido a partir de la relación (2.5),

$$\rho(\hat{z}, \mu) := \|S\hat{z} - \mu\hat{z}\| = h_{k+1,k}|e_k^* y|. \tag{2.6}$$

Dicha expresión se suele utilizar para determinar, sin ningún coste adicional, cuáles de las aproximaciones obtenidas se consideran convergidas.

En términos computacionales, las operaciones de mayor coste y que requieren comunicación entre los procesos, son la de expansión del subespacio y la de ortonormalización. La primera de ellas, que en el Algoritmo 2.1 aparece como una multiplicación matriz-vector, se corresponde en muchas ocasiones con la resolución de un sistema de ecuaciones lineales. Esto ocurre, por ejemplo, cuando se resuelve un problema lineal generalizado $Ax = \lambda Bx$, ya que el operador que se utiliza es $B^{-1}A$, si no se lleva a cabo ninguna transformación espectral, o el operador $(A-\sigma B)^{-1}B$ en caso de utilizar la de shift-and-invert. En la implementación del método de Krylov–Schur (variante con reinicio de Arnoldi) incluida en SLEPc, la resolución de los sistemas de ecuaciones lineales se hace vía la clase `KSP` de PETSc, mientras que las operaciones relativas a la ortogonalización y la resolución del problema proyectado se realizan, respectivamente, a través de las clases `BV` y `DS` de SLEPc, descritas en la sección 1.4.2.

El método Q-Arnoldi (*Quadratic Arnoldi* [66]) resuelve un PEP de grado 2, $Q(\lambda)x = 0$, con

$$Q(\lambda) = A_0 + \lambda A_1 + \lambda^2 A_2, \tag{2.7}$$

aplicando una variante del método de Arnoldi sobre la linealización $L(\lambda) = A - \lambda B$, dada por

$$A = \begin{bmatrix} 0 & I_n \\ -A_0 & -A_1 \end{bmatrix}, \quad B = \begin{bmatrix} I_n & \\ & A_2 \end{bmatrix}. \tag{2.8}$$

Esta expresión define una linealización fuerte, conocida como *primera forma compañera* de Q. Los vectores propios de Q pueden obtenerse de forma inmediata a partir de los de L, ya que estos últimos tienen la forma

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}, \tag{2.9}$$

siendo (x, λ) un par propio de Q.

Para la resolución, mediante el método de Arnoldi, del problema generalizado definido por (A, B), éste se reduce a forma estándar $Sz = \lambda z$, tomando, por ejemplo, $S = B^{-1}A$, en el caso de que A_2 sea regular. El método Q-Arnoldi tiene en cuenta la estructura por bloques del problema resultante,

$$S = \begin{bmatrix} 0 & I_n \\ -A_2^{-1}A_0 & -A_2^{-1}A_1 \end{bmatrix}, \tag{2.10}$$

y reduce la cantidad de memoria necesaria trabajando de forma que almacena únicamente las n primeras entradas de cada uno de los vectores de Arnoldi que genera. Para ello, considera que éstos están divididos según (2.2), y utiliza la relación entre los dos bloques de los vectores de Arnoldi,

$$V_k^1 = V_k^0 H_k + h_{k+1,k} v_{k+1}^0 e_k^*, \tag{2.11}$$

que surge cuando se igualan las n primeras filas en la relación (2.5), al considerar la matriz S definida en (2.10).

A continuación se muestra la forma en que el método de Q-Arnoldi, partiendo de una relación de Arnoldi de orden j, en la que únicamente están almacenados de forma explícita el vector de continuación v_{j+1} (completo) y el primer bloque de V_j, genera una relación de Arnoldi de orden $j+1$:

1. La expansión del subespacio de Krylov $w = Sv_{j+1}$, en el paso 4 del Algoritmo 2.1, se lleva a cabo operando directamente con las matrices del problema cuadrático, en la forma

$$\begin{cases} w^0 = v_{j+1}^1, \\ w^1 = -A_2^{-1}(A_0 v_{j+1}^0 + A_1 v_{j+1}^1), \end{cases} \tag{2.12}$$

   por lo que la matriz S no se crea explícitamente.

2. Los cálculos para la ortogonalización de w contra las columnas de V_{j+1} (pasos 5 y 6 del Algoritmo 2.1) se realizan utilizando la relación (2.11),

$$\begin{aligned} \tilde{w} :=& w - \begin{bmatrix} V_{j+1}^0 \\ [V_j^1 \quad v_{j+1}^1] \end{bmatrix} V_{j+1}^* w = \\ =& \begin{bmatrix} w^0 \\ w^1 \end{bmatrix} - \begin{bmatrix} V_{j+1}^0 \\ [V_j^0 H_j + h_{j+1,j} v_{j+1}^0 e_j^* \quad v_{j+1}^1] \end{bmatrix} h_{j+1}, \end{aligned} \tag{2.13}$$

   donde los coeficientes de ortogonalización h_{j+1} (elementos de la columna $j+1$ de H_{j+1}), vienen dados por

$$\begin{aligned} h_{j+1} :=& \begin{bmatrix} V_{j+1}^0 \\ [V_j^1 \quad v_{j+1}^1] \end{bmatrix}^* \begin{bmatrix} w^0 \\ w^1 \end{bmatrix} = \\ =& (V_{j+1}^0)^* w^0 + \begin{bmatrix} H_j^* (V_j^0)^* w^1 + h_{j+1,j} (v_{j+1}^0)^* w^1 e_j \\ (v_{j+1}^1)^* w^1 \end{bmatrix}. \end{aligned} \tag{2.14}$$

3. La normalización de $\tilde{w}$ da como resultado un nuevo vector de Arnoldi v_{j+2} que debe ser almacenado completo para ser utilizado en la extensión del subespacio de Krylov en la siguiente iteración. Por otro lado, se descarta el vector v_{j+1}^1 que ya no volverá a ser utilizado.

El método TOAR descrito en [85, 59] mantiene la idea, al igual que Q-Arnoldi, de recuperar cada bloque $\{V_k^i\}_{i=0}^1$ siempre que sea necesario, a partir de un conjunto de vectores en $\mathbb{C}^n$. En el caso de Q-Arnoldi, el conjunto de vectores viene dado por $\{V_k^0 e_j\}_{j=1}^k \cup \{v_{k+1}^0, v_{k+1}^1\}$, mientras que TOAR escoge tales vectores de manera que formen una base ortonormal del subespacio

$$\operatorname{span}(\cup_{i=0}^1 \{V_k^i e_j\}_{j=1}^k \cup \{v_{k+1}^i\}_{i=0}^1) = \operatorname{span}(\{V_k^0 e_j\}_{j=1}^k \cup \{v_{k+1}^i\}_{i=0}^1). \tag{2.15}$$

La igualdad en (2.15) se deduce de la relación (2.11). De esta forma, el método TOAR no da un protagonismo especial a ninguno de los bloques $\{V_k^i\}_{i=0}^1$, sino que genera una matriz de columnas ortogonales, $U_{k+2} \in \mathbb{C}^{n\times(k+2)}$, matrices $\{G_k^i\}_{i=0}^1 \subset \mathbb{C}^{(k+2)\times k}$ y vectores $\{g_{k+1}^i\}_{i=0}^1 \subset \mathbb{C}^{k+2}$ a partir de los cuales puede recuperar los vectores de Arnoldi (columnas de V_k),

$$V_k^i = U_{k+2} G_k^i, \quad i = 0, 1, \tag{2.16a}$$

$$v_{k+1}^i = U_{k+2} g_{k+1}^i, \quad i = 0, 1. \tag{2.16b}$$

A continuación se describe la forma en que este método lleva a cabo las operaciones de una iteración de la recurrencia de Arnoldi. Se parte de una relación de Arnoldi (2.5) de orden j, en la que tanto los bloques V_j^0 y V_j^1 como el vector de continuación v_{j+1} están expresados según el formato de TOAR (2.16). Entonces, los pasos de TOAR que producen una relación de orden $j+1$ son:

1. Para la extensión del subespacio de Krylov, el método de TOAR añade una nueva columna a la matriz U_{j+2} y genera los coeficientes $g^i \in \mathbb{C}^{j+3}$ de forma que los dos fragmentos de $w = Sv_{k+1}$ puedan ser expresados como

   $$w^i = U_{k+3} g^i, \quad i = 0, 1, \tag{2.17}$$

   donde $U_{j+3} := \begin{bmatrix} U_{j+2} & u_{j+3} \end{bmatrix}$ es la nueva matriz extendida. Para ello, en primer lugar se obtiene el vector $w^1 = -A_2^{-1}(A_0 v_{j+1}^0 + A_1 v_{j+1}^1)$ a partir de (2.12) (reconstruyendo v_{j+1}^0 y v_{j+1}^1), y a continuación éste se ortogonaliza contra las columnas de U_{j+2} y se normaliza, dando lugar a u_{j+3}. Tomando $g^1 \in \mathbb{C}^{j+3}$ como los coeficientes de ortogonalización junto con el factor de normalización, se satisface (2.17) para $i = 1$. Por otro lado, a partir de (2.12) tenemos que $w^0 = v_{j+1}^1 = U_{j+2} g_{j+1}^1$, por lo que, definiendo $g^0 := \begin{bmatrix} g_{j+1}^1 \\ 0 \end{bmatrix}$ se tiene que la relación (2.17) se cumple también para $i = 0$.

2. Para generar el nuevo vector de Arnoldi, ortogonalizando w contra las columnas de V_{j+1}, en primer lugar se calculan los coeficientes de ortogonalización

   $$\begin{aligned} h_{j+1} &:= \begin{bmatrix} V_{j+1}^0 \\ V_{j+1}^1 \end{bmatrix}^* \begin{bmatrix} w^0 \\ w^1 \end{bmatrix} = \\ &= \begin{bmatrix} G_{j+1}^0 \\ 0 \end{bmatrix}^* U_{j+3}^* U_{j+3} g^0 + \begin{bmatrix} G_{j+1}^1 \\ 0 \end{bmatrix}^* U_{j+3}^* U_{j+3} g^1 = \\ &= \begin{bmatrix} \tilde{G}_{j+1}^0 \\ \tilde{G}_{j+1}^1 \end{bmatrix}^* \begin{bmatrix} g^0 \\ g^1 \end{bmatrix}, \quad \text{con } \tilde{G}_{j+1}^i := \begin{bmatrix} G_{j+1}^i \\ 0 \end{bmatrix}, \end{aligned} \tag{2.18}$$

y así, el vector resultante de la ortogonalización es

$$\begin{aligned}\tilde{w} &:= w - V_{j+1}V_{j+1}^* w = \\ &= \begin{bmatrix} U_{j+3}g^0 \\ U_{j+3}g^1 \end{bmatrix} - \begin{bmatrix} U_{j+3}\tilde{G}_{j+1}^0 \\ U_{j+3}\tilde{G}_{j+1}^1 \end{bmatrix} h_{j+1} \\ &= \begin{bmatrix} U_{j+3} & \\ & U_{j+3} \end{bmatrix} \left(\begin{bmatrix} g^0 \\ g^1 \end{bmatrix} - \begin{bmatrix} \tilde{G}_{j+1}^0 \\ \tilde{G}_{j+1}^1 \end{bmatrix} h_{j+1} \right).\end{aligned} \tag{2.19}$$

Dado que los bloques V_j^i están expresados en la forma (2.16), la ortogonalización en (2.18) y (2.19) puede ser formulada en términos de las columnas de las matrices G_j^i en lugar de las de V_j^i. Esto conduce a la observación importante de que en cada iteración de TOAR se realiza la ortogonalización de un vector de longitud n (al calcular u_{j+3}) y otra de longitud $2(j+3)$ (al ortogonalizar g contra las columnas de $\tilde{G}_{j+1}$), en lugar de la ortogonalización de un vector de longitud dn que se llevaría a cabo en el método de Arnoldi básico aplicado sobre la matriz S de dimensión dn.

3. En el paso de normalización, el método TOAR también reduce el número de operaciones que realiza, respecto a las requeridas por el método de Arnoldi. Para $\tilde{g} := g - \tilde{G}_{j+1}h_{j+1}$, se tiene que $\|\tilde{w}\| = \|\tilde{g}\|$, por lo que la normalización de $\tilde{w}$ puede realizarse normalizando $\tilde{g}$, lo que da lugar a $g_{j+2} := \tilde{g}/h_{j+2,j+1}$, con $h_{j+2,j+1} := \|\tilde{g}\|$, y al nuevo vector de Arnoldi expresado en la forma $v_{j+2}^i = U_{j+3}g_{j+2}^i$, para $i = 0, 1$.

Tanto Q-Arnoldi como TOAR fueron desarrollados inicialmente para la resolución de problemas cuadráticos. Sin embargo, estos métodos pueden ser generalizados para la resolución de problemas polinómicos de grado arbitrario [53]. SLEPc dispone de implementaciones de ambos métodos, aunque solo el de TOAR ha sido extendido para polinomios de grado mayor que 2. Por un lado, este método tiene una expresión más simple de los bloques V_k^i en el caso de que el polinomio matricial esté expresado en una base diferente de la de monomios. Por otro lado, el método Q-Arnoldi presenta riesgo de inestabilidad [66], por ejemplo, en el caso de tener en (2.11) un elevado valor de $\|H_k\|$. Este riesgo no se da en TOAR ya que éste representa la matriz de los vectores de Arnoldi, V_k, como producto de dos matrices con columnas ortonormales, y recupera cada bloque de V_k a partir de una matriz cuyas columnas forman una base ortonormal. En la sección 2.2.3 se explica en detalle la implementación del método de TOAR realizada.

2.1.3. Linealización de matrices polinómicas expresadas en función de bases de polinomios genéricas

En este apartado se dan detalles sobre la linealización que utilizamos para resolver un PEP expresado en la forma genérica (1.9), en términos de una base de polinomios que puede ser diferente de la usual de monomios. En concreto se da soporte a bases

de polinomios descritas en [3], las cuales satisfacen una recurrencia de tres términos del tipo

$$\lambda\,\Phi_j(\lambda) = \alpha_j\,\Phi_{j+1}(\lambda) + \beta_j\,\Phi_j(\lambda) + \gamma_j\,\Phi_{j-1}(\lambda), \quad \text{para } j = 1, 2, \ldots \tag{2.20}$$

donde $\Phi_{-1} \equiv 0$, $\Phi_0 \equiv 1$, y para $j = 0, 1, \ldots$, α_j, β_j y γ_j son reales, $\alpha_j > 0$ y $\alpha_j = \frac{c_j}{c_{j+1}}$, donde c_j es el coeficiente principal de $\Phi_j(\lambda)$. Entre éstas se encuentra cualquier base de polinomios ortogonales con grado creciente. Algunos conjuntos de bases polinómicas que satisfacen (2.20) son:

Monomios: caso particular tomando $\alpha_j = 1$ y $\beta_j = \gamma_j = 0$, $\forall j \geq 0$.

Chebyshev de primera clase: $T_0(\lambda) = 1$, $T_1(\lambda) = \lambda$, $T_{j+1}(\lambda) = 2\lambda T_j(\lambda) - T_{j-1}(\lambda)$, con coeficientes $\alpha_0 = 1$, $\alpha_j = \frac{1}{2}$, $\beta_j = 0$, $\gamma_j = \frac{1}{2}$.

Chebyshev de segunda clase: $U_0(\lambda) = 1$, $U_1(\lambda) = 2\lambda$, $U_{j+1}(\lambda) = 2\lambda U_j(\lambda) - U_{j-1}(\lambda)$, con coeficientes $\alpha_j = \frac{1}{2}$, $\beta_j = 0$, $\gamma_j = \frac{1}{2}$.

Legendre: $P_0(\lambda) = 1$, $P_1(\lambda) = \lambda$, $(j+1)P_{j+1}(\lambda) = (2j+1)\lambda P_j(\lambda) + jP_{j-1}(\lambda)$, con coeficientes $\alpha_j = j+1$, $\beta_j = -2j$, $\gamma_j = j$.

Laguerre: $L_0(\lambda) = 1$, $L_1(\lambda) = 1 - \lambda$, $(j+1)L_{j+1}(\lambda) = (2j+1-\lambda)L_j(\lambda) - jL_{j-1}(\lambda)$, con $\alpha_0 = -1$, $\alpha_j = -(j+1)$, $\beta_0 = 0$, $\beta_j = (2j+1)$, $\gamma_j = -j$.

Hermite: $H_0(\lambda) = 1$, $H_1(\lambda) = 2\lambda$, $H_j{+}1(\lambda) = 2\lambda H_j(\lambda) - 2jH_{j-1}(\lambda)$, con coeficientes $\alpha_j = \frac{1}{2}$, $\beta_j = 0$, $\gamma_j = j$.

Para la resolución mediante linealización de un PEP definido en la forma genérica (1.9) utilizamos una linealización fuerte similar a la descrita por Amiraslani y co-autores en [3]. Ésta viene dada por

$$L(\lambda) = A - \lambda B, \tag{2.21}$$

donde

$$A = \begin{bmatrix} \beta_0 I & \alpha_0 I & & & & & \\ \gamma_1 I & \beta_1 I & \alpha_1 I & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \gamma_{d-2} I & \beta_{d-2} I & \alpha_{d-2} I \\ \tilde{A}_0 & \tilde{A}_1 & \tilde{A}_2 & \cdots & \tilde{A}_{d-3} & \tilde{A}_{d-2} & \tilde{A}_{d-1} \end{bmatrix}, \; B = \begin{bmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & & c_d A_d \end{bmatrix},$$

con $\tilde{A}_j = -c_{d-1}A_j$ $(j = 0, \ldots, d-3)$, $\tilde{A}_{d-2} = -c_{d-1}A_{d-2} + c_d\gamma_{d-1}A_d$ y $\tilde{A}_{d-1} = -c_{d-1}A_{d-1} + c_d\beta_{d-1}A_d$. En el caso de utilizar la base de monomios, la linealización (2.21) coincide con la primera forma compañera de P [86]. En ese mismo caso, la linealización presentada en [3] se reduce a la segunda forma compañera de P. La utilización de la linealización (2.21) está motivada por nuestro interés en calcular autovectores derechos de (1.9), los cuales pueden ser fácilmente obtenidos a partir

de los calculados para (2.21). La linealización propuesta en [3] sería, por su parte, adecuada para el cálculo de los autovectores izquierdos.

De forma similar a la demostración realizada en [3], se puede probar que el haz de matrices (2.21) es una linealización fuerte de (1.9), por lo que ambos problemas tienen el mismo espectro. Además, los autovectores derechos z de (2.21) tienen la forma

$$z = \begin{bmatrix} x \\ \Phi_1(\lambda)x \\ \vdots \\ \Phi_{d-1}(\lambda)x \end{bmatrix}, \tag{2.22}$$

donde x es un autovector derecho de (1.9), por lo que esta expresión permite obtener fácilmente los autovectores de $P(\lambda)$ a partir de los de $L(\lambda)$.

Dado que en secciones posteriores se utilizará la descomposición LU a bloques de (2.21), para terminar esta sección se muestra cómo sería esta factorización, la cual se obtiene procediendo de forma similar a [2], y viene dada por

$$(A - \lambda B)\Pi = L_\lambda U_\lambda, \tag{2.23}$$

donde $\Pi = \left[\begin{smallmatrix} 0 & I_n \\ I_{(d-1)n} & 0 \end{smallmatrix}\right]$ es una matriz de permutación, y

$$L_\lambda = \begin{bmatrix} \alpha_0 I & & & & & & \\ (\beta_1 - \lambda)I & \alpha_1 I & & & & & \\ \gamma_2 I & (\beta_2 - \lambda)I & \ddots & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & (\beta_{d-2} - \lambda)I & \alpha_{d-2} I & & \\ -c_{d-1}A_1 & -c_{d-1}A_2 & \dots & \hat{A}_{d-2} & \hat{A}_{d-1} & -c_{d-1}P(\lambda) \end{bmatrix}, \tag{2.24}$$

$$U_\lambda = \begin{bmatrix} I & & & -\frac{\Phi_1(\lambda)}{\Phi_0(\lambda)}I \\ & I & & -\frac{\Phi_2(\lambda)}{\Phi_0(\lambda)}I \\ & & \ddots & \vdots \\ & & I & -\frac{\Phi_{d-1}(\lambda)}{\Phi_0(\lambda)}I \\ & & & I \end{bmatrix}, \tag{2.25}$$

con $\hat{A}_{d-2} = -c_{d-1}A_{d-2} + c_d\gamma_{d-1}A_d$ y $\hat{A}_{d-1} = -c_{d-1}A_{d-1} + c_d(\beta_{d-1} - \lambda)A_d$.

2.2. Solvers basados en linealización

En esta sección se describen los distintos solvers de la clase `PEP` de SLEPc, que abordan la resolución del problema polinómico de valores propios vía linealización. La forma más inmediata de resolver un PEP por medio de una linealización, es la de formar explícitamente las matrices que la componen. Como ya se ha comentado

con anterioridad, esta opción tiene la ventaja de que permite la utilización directa de cualquiera de los solvers lineales de SLEPc. Por otro lado, ésta podría no ser una opción válida en muchos casos, ya que el aumento de tamaño de las nuevas matrices puede ser una limitación severa, tanto por la memoria que requieren, como por el incremento en la complejidad computacional en operaciones como la factorización LU que a veces se requieren. Estas consideraciones han motivado que la opción de la formación explícita de las matrices de la linealización, disponible desde la primera versión del antiguo solver para problemas cuadráticos `QEP`, no se haya extendido para polinomios de grado mayor que 2. En la construcción explícita en paralelo de las matrices de la linealización (2.8), se evita el movimiento de datos y se reduce el posible desequilibrio en la cantidad de datos y trabajo, que produciría un reparto directo, por bloques de filas consecutivas, de las matrices de la linealización. Por ejemplo, para la resolución con dos procesos, de un problema cuadrático $P(\lambda) = K + \lambda C + \lambda^2 M$, se intercalan las submatrices de P localmente almacenadas en cada proceso, de forma que la primera matriz de la linealización $A = \left[\begin{smallmatrix} 0 & I \\ -K & -C \end{smallmatrix}\right]$, se almacena de forma paralela con la distribución

$$\tilde{A} = \left[\begin{array}{cc|cc} 0 & I & 0 & 0 \\ -K_{11} & -C_{11} & -K_{12} & -C_{12} \\ \hline 0 & 0 & 0 & I \\ -K_{21} & -C_{21} & -K_{22} & -C_{22} \end{array}\right], \tag{2.26}$$

donde $[K_{11}, K_{12}]$ se corresponde con la parte de $K = \left[\begin{smallmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{smallmatrix}\right]$ localmente almacenada en el primer proceso (la misma notación se emplea para C). El problema que se resuelve con dicha distribución viene dado por el par $(\tilde{A}, \tilde{B})$, con $\tilde{A} := \pi A \pi^*$ y $\tilde{B} := \pi B \pi^*$, donde

$$\pi = \left[\begin{smallmatrix} & & I & \\ & I & & \\ & & & I \end{smallmatrix}\right].$$

Ambos problemas tienen los mismos autovalores y sus vectores propios se relacionan mediante la permutación π, que es tenida en cuenta al recuperar los autovectores del problema cuadrático original a partir de los del lineal resuelto.

Para la resolución del problema lineal cuando las matrices de la linealización no se forman explícitamente (siempre con grado mayor que 2), se utilizan distintas variantes del método de Arnoldi (descrito en la sección 2.1.2). La principal diferencia entre las variantes empleadas radica en la forma en que los vectores de Arnoldi son almacenados, y en el modo en que éstos se recuperan en caso de que su almacenamiento no haya sido explícito. En función de estos parámetros, se tienen las siguientes variantes:

Arnoldi Básico almacena explícitamente los vectores de Arnoldi completos, de dimensión d veces la del problema polinómico original, n. El cálculo con las matrices de la linealización (2.21) se hace implícitamente y de forma eficiente, operando por bloques, utilizando las matrices del problema polinómico tal como se describe en la sección 2.2.2.

Q-Arnoldi almacena únicamente el primer bloque (longitud n) de cada uno de los vectores de Arnoldi que se generan, y trabaja reconstruyendo el resto de bloques cuando necesita operar on ellos. La implementación disponible en SLEPc se corresponde con el método desarrollado por Meerbergen en [66], y descrito en 2.2. Esta variante sólo está disponible para problemas cuadráticos expresados en base de monomios. No ha sido extendida para polinomios de mayor grado porque la variante de TOAR se ha considerado más conveniente en dichos casos.

TOAR opera de forma implícita con los vectores de Arnoldi expresados de forma similar a (2.16) a partir de una base ortonormal en $\mathbb{C}^n$ que crece a medida que se extiende el subespacio de Krylov. La implementación realizada se basa en el algoritmo original para grado 2 [85, 59], y en su generalización para grado arbitrario de problemas polinómicos expresados en términos de bases de Chebyshev desarrollado en [53].

Antes de pasar a describir los elementos relevantes de las variantes implementadas, se exponen a continuación los dos enfoques posibles, implementados en SLEPc, al utilizar una transformación espectral juntamente con un solver polinómico que aborda la resolución de un problema lineal.

2.2.1. Transformación espectral en problemas polinómicos resueltos vía linealización

Los métodos que se describen en este capítulo se basan en aplicar el método de Arnoldi sobre la linealización fuerte $L(\lambda)$ dada en (2.21). La convergencia típica de dicho método hace que a medida que el número de iteraciones aumenta, los valores de Ritz se aproximen a los autovalores externos de $L(\lambda)$. Por ello, cuando se quieren obtener autovalores situados en el interior del espectro, por ejemplo los cercanos a un determinado valor σ dado, se realiza una transformación espectral de desplazamiento e inversión

$$\lambda = \frac{1}{\theta} + \sigma, \tag{2.27}$$

que genera un nuevo problema lineal $\tilde{L}(\theta)$ con los mismos autovectores que el original y con autovalores θ que se relacionan con los de $L(\lambda)$ según (2.27). Así, al resolver el nuevo problema, utilizando el método de Arnoldi, se obtienen los autovalores λ de $L(\lambda)$ y $P(\lambda)$ cercanos a σ.

Si las matrices de la linealización L se forman explícitamente y se utiliza un solver lineal de la clase `EPS` para su resolución, el objeto `ST` (ver sección 1.4.2) incluido en dicho solver será el encargado de generar las matrices del nuevo problema lineal transformado, de forma que el solver trabajará sin conocimiento de la transformación realizada. Sin embargo, para los casos en que la linealización no se construye explícitamente, será el solver polinómico el que debe ser capaz de operar implícitamente con las matrices del problema lineal aun en el caso de que se haya realizado sobre éste una transformación espectral como se detalla en la sección 2.2.2.

Otra forma de proceder al aplicar una transformación espectral en estos problemas, es realizar el cambio de variable sobre el problema polinómico, obteniendo un nuevo polinomio $\hat{P}(\theta)$ del que posteriormente se puede construir su linealización $\hat{L}(\theta)$ (de forma explícita o no). Este último enfoque, también incluido en la implementación realizada en SLEPc, tiene la ventaja de que puede ser manejado internamente por el objeto `ST` de la clase `PEP`, y es válida para cualquiera de los solvers que en este capítulo se describen. Como contrapartida, este enfoque tiene un sobrecoste, especialmente de memoria, porque se crean las matrices del problema polinómico transformado.

Para la utilización del último esquema descrito, se ha ampliado la funcionalidad de la clase `ST` para que incluya el tratamiento de cambios de variable en problemas polinómicos. En el caso de la trasformación espectral de desplazamiento, en la que se realiza el cambio de variable $\lambda = \theta + \sigma$ sobre $P(\lambda)$, el polinomio transformado que se obtiene es

$$\begin{aligned}\tilde{P}(\theta) := P(\theta+\sigma) &= A_0 + (\theta+\sigma)A_1 + \cdots + (\theta+\sigma)^d A_d = \\ &= A_0 + (\theta+\sigma)A_1 + \cdots + \left(\sum_{j=0}^{d}\binom{d}{j}\theta^j\sigma^{d-j}\right)A_d, \end{aligned} \tag{2.28}$$

a partir del cual, después de reordenar y agrupar los términos asociados a las distintas potencias de θ, obtenemos $\tilde{P}(\theta) = T_0 + \theta T_1 + \cdots + \theta^d T_d$ definido por las matrices

$$T_k = \sum_{j=k}^{d}\binom{j}{k}\sigma^{j-k}A_j = \sum_{j=0}^{d-k}\binom{j+k}{k}\sigma^j A_{j+k}, \tag{2.29}$$

que son las calculadas internamente por el objeto `ST`, cuando es requerido. En caso de realizar una transformación de desplazamiento e inversión, simplemente se trabaja con el polinomio reverso $\mathrm{rev}(\tilde{P}(\theta))$, definido en (1.6).

2.2.2. Arnoldi Básico

El método de Arnoldi Básico se comporta como el solver lineal Krylov–Schur disponible en SLEPc, excepto por el hecho de que no forma explícitamente las matrices de la linealización y opera con ellas utilizando únicamente las matrices A_i, de dimensión n, que definen el problema polinómico original. La forma en que el cálculo implícito con las matrices del problema lineal se lleva a cabo está íntimamente ligada a la linealización que se utiliza y a la posible transformación espectral que se quiera aplicar. Por ejemplo, la expresión (2.12) muestra la forma en que se realiza el producto de $B^{-1}A$ por un vector, para el problema (2.8) en el caso de que no se aplique ninguna transformación espectral. Para un caso más general de grado arbitrario d, considerando que los vectores se dividen en la forma (2.2), las ecuaciones equivalentes para dicho cálculo serían:

$$\begin{cases} w^i = v^{i+1}, \quad i = 0, \ldots, d-2, \\ w^{d-1} = -A_d^{-1}(A_0 v^0 + \cdots + A_{d-1}v^{d-1}). \end{cases} \tag{2.30}$$

A continuación se muestra la forma en que se opera implícitamente con las matrices de la linealización (2.21) asociada a un problema polinómico del tipo (1.9), en los casos en que se realice sobre la linealización una transformación espectral de desplazamiento o de desplazamiento con inversión. En el primer caso, tomando A y B dadas en (2.21), el producto matriz-vector $w = B^{-1}(A - \sigma B)v$, para $v \in \mathbb{C}^{dn}$, utilizando únicamente las matrices $\{A_j\}_{j=0}^{d-1}$, viene dado por las expresiones

$$\begin{cases} w^0 = (\beta_0 - \sigma)v^0 + \alpha_0 v^1, \\ w^j = \gamma_j v^{j-1} + (\beta_j - \sigma)v^j + \alpha_j v^{j+1}, \quad j = 1, \ldots, d-2, \\ w^{d-1} = -\alpha_{d-1} A_d^{-1}\Big(\sum_{j=0}^{d-1} A_j v^j\Big) + \gamma_{d-1} v^{d-2} + (\beta_{d-1} - \sigma)v^{d-1}. \end{cases} \tag{2.31}$$

En el caso de utilizar una transformación de desplazamiento e inversión, obtenemos expresiones similares a (2.31) para el calculo implícito de $w = (A - \sigma B)^{-1}Bv$, utilizando la descomposición LU de $(A - \sigma B)$ dada en (2.23), mediante la expresión

$$w = (A - \sigma B)^{-1}Bv = \Pi U_\sigma^{-1} L_\sigma^{-1} Bv. \tag{2.32}$$

En primer lugar se obtiene $t = L_\sigma^{-1}Bv$ mediante la recurrencia

$$\begin{cases} t^0 = \alpha_0^{-1} v^0, \\ t^1 = \alpha_1^{-1}(v^1 + (\sigma - \beta_1)t^0), \\ t^j = \alpha_j^{-1}(v^j + (\sigma - \beta_j)t^{j-1} - \gamma_j t^{j-2}), \quad j = 2, \ldots, d-2, \\ t^{d-1} = -P(\sigma)^{-1}(A_1 t^0 + \cdots + A_{d-2} t^{d-3} + A_{d-1} t^{d-2} + A_d \hat{t}^{d-1}), \end{cases} \tag{2.33}$$

donde $\hat{t}^{d-1} = \alpha_{d-1}^{-1}(-\gamma_{d-1} t^{d-3} + (\sigma - \beta_{d-1})t^{d-2} + v^{d-1})$. Dicha recurrencia se obtiene realizando un proceso de eliminación hacia adelante a bloques en $L_\sigma t = Bv$ dada por

$$\begin{bmatrix} \alpha_0 I & & & & \\ (\beta_1 - \sigma)I & \alpha_1 I & & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & (\beta_{d-2} - \sigma)I & \alpha_{d-2} I \\ -c_{d-1}A_1 & -c_{d-1}A_2 & \ldots & \tilde{A}_{d-2} & \tilde{A}_{d-1} \; -c_{d-1}P(\sigma) \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{d-2} \\ t_{d-1} \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-2} \\ c_d A_d v_{d-1} \end{bmatrix}. \tag{2.34}$$

Una vez calculado t, considerando ahora un proceso de sustitución regresiva a bloques en $U_\sigma \Pi^* w = t$ dada por

$$\begin{bmatrix} I & & & -\frac{\phi_1(\sigma)}{\phi_0(\sigma)} I \\ & I & & -\frac{\phi_2(\sigma)}{\phi_0(\sigma)} I \\ & & \ddots & \vdots \\ & & I & -\frac{\phi_{d-1}(\sigma)}{\phi_0(\sigma)} I \\ & & & I \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \ldots \\ w_{d-1} \\ w_0 \end{bmatrix} = \begin{bmatrix} t_0 \\ t_1 \\ \ldots \\ t_{d-2} \\ t_{d-1} \end{bmatrix}, \tag{2.35}$$

y denotando $q := t_{d-1}$, obtenemos finalmente las relaciones

$$\begin{cases} w^0 = q, \\ w^j = \dfrac{\Phi_j(\sigma)}{\Phi_0(\sigma)} q + t^{j-1}, \quad j = 1, \ldots, d-1. \end{cases} \tag{2.36}$$

La generalización del método de TOAR para PEPs del tipo (1.9), que a continuación se presenta, también utiliza las recurrencias (2.31), (2.33) y (2.36) para operar implícitamente con las matrices de la linealización. Sin embargo, este método no almacena los vectores de Arnoldi completos (tamaño dn) sino que representa la matriz V_k, que tiene por columnas los vectores de Arnoldi asociados a un subespacio de Krylov de dimensión k, en la forma $(I_d \otimes U)G$ con $U \in \mathbb{C}^{n\times q}$, $G \in \mathbb{C}^{q\times k}$ y q típicamente igual a $k-1+d$. Esta representación hace que las recurrencias anteriores puedan expresarse en términos de las columnas de la matriz G y de que no sea necesario reconstruir los vectores de Arnoldi para operar con ellas.

2.2.3. TOAR polinómico

En la sección 2.1.2 vimos que la estructura a bloques particular de la linealización compañera (2.8) de un polinomio matricial de grado 2, inducía una relación entre las dos mitades de la matriz V_k de los vectores de Arnoldi (2.11). En el caso de un polinomio de grado d, considerando el mismo tipo de linealización e igualando los primeros $d-1$ bloques de filas en la ecuación de Arnoldi (2.5), obtenemos relaciones similares para los distintos bloques de V_k (dividido en la forma (2.2)), dadas por

$$V_k^{i+1} = V_k^i H_k + h_{k+1,k} v_{k+1}^i e_k^*, \quad i = 0, \ldots, d-2. \tag{2.37}$$

A partir de dichas relaciones se deduce una igualdad de subespacios análoga a la mostrada para grado dos en (2.15),

$$\operatorname{span}(\cup_{i=0}^{d-1} \{V_k^i e_j\}_{j=1}^k \cup \{v_{k+1}^i\}_{i=0}^{d-1}) = \operatorname{span}(\{V_k^0 e_j\}_{j=1}^k \cup \{v_{k+1}^i\}_{i=0}^{d-1}), \tag{2.38}$$

la cual nos dice que cada una de las d partes de los $k+1$ vectores de Arnoldi, representados en las columnas de V_{k+1}, pueden ser obtenidas a partir de combinaciones lineales de $k+d$ vectores, por lo que éstos pueden ser representados utilizando un esquema del tipo TOAR como el visto en (2.38) para grado dos. Es decir, la matriz V_k se puede obtener a partir de expresiones del tipo

$$V_k^i = U_{k+d} G_k^i, \quad i = 0, \ldots, d-1, \tag{2.39a}$$

$$v_{k+1}^i = U_{k+d} g_{k+1}^i, \quad i = 0, \ldots, d-1, \tag{2.39b}$$

con $U_{k+d} \in \mathbb{C}^{n\times(k+d)}$ de columnas ortonormales, matrices $\{G_k^i\}_{i=0}^{d-1} \subset \mathbb{C}^{(k+d)\times k}$ y vectores $\{g_{k+1}^i\}_{i=0}^{d-1} \subset \mathbb{C}^{k+d}$.

En el caso de PEPs expresados en función de bases de polinomios, al calcular una relación de Arnoldi con la matriz $S := B^{-1}A$ para resolver una linealización del

tipo (2.21), de forma similar a (2.37) podemos obtener las ecuaciones,

$$\begin{cases} V_k^1 = \alpha_0^{-1}(-\beta_0 V_k^0 + V_k^0 H_k + h_{k+1,k} v_{k+1}^0 e_k^*), \\ V_k^i = \alpha_{i-1}^{-1}(-\beta_{i-1} V_k^{i-1} - \gamma_{i-1} V_k^{i-2} + \\ \qquad + V_k^{i-1} H_k + h_{k+1,k} v_{k+1}^{i-1} e_k^*), \qquad i = 2, \ldots, d-1, \end{cases} \tag{2.40}$$

que relacionan los diferentes bloques, $\{V_k^i\}_{i=0}^{d-1}$, de la matriz que representa los vectores de Arnoldi. A partir de dichas relaciones se deduce que, también en este caso, se verifica la igualdad de subespacios (2.38), por lo que es posible utilizar un esquema de tipo TOAR para representar los vectores de Arnoldi en la forma (2.39). A la misma conclusión llegaríamos en el caso de realizar una transformación espectral de desplazamiento e inversión, igualando cada bloque de filas en la ecuación

$$BV_k = (A - \sigma B) V_k H_k + h_{k+1,k}(A - \sigma B) v_{k+1} e_k^*, \tag{2.41}$$

que se obtiene al generar una relación de Arnoldi (2.11) utilizando la matriz $S = (A - \sigma B)^{-1} B$.

Al expresar una relación de Arnoldi (2.11) utilizando una representación de los vectores de Arnoldi siguiendo un esquema de tipo TOAR

$$v_k = \begin{bmatrix} U_{k+d} & & \\ & \ddots & \\ & & U_{k+d} \end{bmatrix} \begin{bmatrix} g_k^0 \\ \vdots \\ g_k^{d-1} \end{bmatrix}, \tag{2.42}$$

dicha relación toma la forma

$$S\hat{U}_{k+d} G_k = \hat{U}_{k+d} G_k H_k + h_{k+1,k} \hat{U}_{k+d} g_{k+1} e_k^*, \tag{2.43}$$

donde $\hat{U}_{k+d} := I_d \otimes U_{k+d}$. Los pasos necesarios para la generación de las matrices implicadas en dicha relación se muestran en el Algoritmo 2.2. Aparte de las operaciones dependientes de la linealización utilizada, que se han encapsulado en la función *expand* de la línea 5 de dicho Algoritmo, el resto de operaciones pueden verse como una extensión directa, para grado mayor que 2, de las descritas en la sección 2.1.2 para TOAR cuadrático.

Los principales pasos en el método TOAR polinómico son los siguientes. En primer lugar (líneas 2 y 3), se genera una base ortonormal de dimensión a lo sumo d, y coeficientes g_1, con los que se representa el vector de inicio de Arnoldi en formato TOAR. A cada iteración del algoritmo de Arnoldi, se extiende el subespacio de Krylov (línea 5), lo que conlleva por un lado la extensión, a lo sumo con un vector, de la base de TOAR utilizada para representar los bloques de los vectores de Arnoldi, y por otro la obtención de los coeficientes g para la representación en función de dicha base, de las d partes del nuevo vector en el subespacio de Krylov. La ortonormalización (líneas 6 y 7) de éste último, para generar implícitamente el nuevo vector que extiende la base de Arnoldi, se hace accediendo únicamente a los coeficientes g que representan dicho vector en función de la base de TOAR.

Se hace notar que, aunque en el paso 2 del Algoritmo 2.2 se ha supuesto, por simplicidad, que inicialmente se genera una base de d vectores (columnas de U_d), y que ésta se extiende (línea 5) con un vector en cada iteración, en la implementación realizada en SLEPc, se tiene en cuenta que la matriz Q obtenida en la factorización QR inicial podría tener menos de d columnas, y también que el vector generado en una determinada iteración j, en el paso 5, podría pertenecer al rango de U_{j-1+d}, con lo que la base de TOAR no se extendería en esa ocasión, aunque sí se generarían los coeficientes g_j correspondientes que representan el vector de Arnoldi para dicha iteración. Esta situación se presenta, por ejemplo, en el caso inusual de que se tenga una base de TOAR de dimensión n igual a la del problema polinómico. A partir de ese momento, al extender la base de Arnoldi sólo se generarían los coeficientes para representar los nuevos vectores en la base de TOAR.

Algoritmo 2.2 TOAR para problemas de valores propios polinómicos

Entrada: Matrices $\{A_i\}_{i=0}^{d} \subset \mathbb{C}^{n\times n}$ del polinomio matricial, vectores $\{w^i\}_{i=0}^{d-1} \subset \mathbb{C}^n$ (d partes del vector de inicio), numero de iteraciones $k \in \mathbb{N}$

Salida: $H_k \in \mathbb{C}^{k\times k}$, $h_{k+1,k} \in \mathbb{R}$, $U_{k+d} \in \mathbb{C}^{n\times(k+d)}$, $\{G_k^i\}_{i=0}^{d} \subset \mathbb{C}^{(k+d)\times k}$, $\{g_{k+1}^i\}_{i=0}^{d-1} \subset \mathbb{C}^{k+d}$ verificando (2.43), con G_k y g_{k+1} divididas según (2.2)

1: $G_0 \leftarrow [\ \ ]$, $H_0 \leftarrow [\ \ ]$

2: /* Ortogonalización vectores iniciales $\{w^i\}_{i=0}^{d-1}$ */
$[Q, R] = \mathrm{qr}(\begin{bmatrix} w^0 & \cdots & w^{d-1} \end{bmatrix})$ /* factorización QR reducida */
$U_d \leftarrow Q$

3: /* Normalización primer vector de Arnoldi */
$g_1 \leftarrow \mathrm{vec}\, R / \| \mathrm{vec}\, R \|$

4: **para** $j = 1, \ldots, k$ **hacer**

5: /* Extensión subespacio de Krylov */
$[u_{j+d}, g] = \mathrm{expand}(\{A_i\}_{i=0}^{d}, U_{j-1+d}, g_j)$
$U_{j+d} \leftarrow \begin{bmatrix} U_{j-1+d} & u_{j+d} \end{bmatrix}$

6: /* Ortogonalización mediante Gram-Schmidt */
$G_j^i \leftarrow \begin{bmatrix} G_{j-1}^i & g_j^i \\ 0 & 0 \end{bmatrix}$, $i = 0, \ldots, d-1$
$h_j = G_j^* g, \quad \hat{g} \leftarrow g - G_j h_j$

7: /* Normalización nuevo vector de Arnoldi */
$h_{j+1,j} = \|\hat{g}\|, \quad g_{j+1} \leftarrow \hat{g}/h_{j+1,j}$

8: /* Actualización matriz problema proyectado */
$H_j \leftarrow \left[\begin{array}{c|c} H_{j-1} \\ 0 & h_j \end{array}\right]$

9: **fin para**

Como ya se ha comentado antes, las operaciones que se realizan en el cálculo implícito de $w = Sv_j$, para la extensión del subespacio de Krylov en la iteración j, son dependientes de la linealización utilizada y de si se está o no realizando una transformación espectral. En el Algoritmo 2.2 estas operaciones se han considerado incluidas en la función

$$[u_{j+d}, g] = \mathrm{expand}(\{A_i\}_{i=0}^{d}, U_{j-1+d}, g_j), \tag{2.44}$$

en la que se obtienen $u_{j+d} \in \mathbb{C}^n$ y $g \in \mathbb{C}^{d(j+d)}$ tales que $w = (I_d \otimes [\, U_{j-1+d} \; u_{j+d} \,])g$. Una vez w se ha ortonormalizado (implícitamente en pasos 6 y 7) el vector de Arnoldi para esta iteración se representa como $v_j = (I_d \otimes U_{j-1+d})g_j$.

A continuación se detalla la forma en que se llevan a cabo los cálculos implicados en (2.44) cuando se utiliza una linealización del tipo (2.21), para los casos en que se realiza una transformación espectral de desplazamiento, o una de desplazamiento con inversión. Estos cálculos utilizarán las expresiones (2.31) o (2.36) obtenidas para el producto matriz-vector implícito en el método de Arnoldi Básico.

En el caso de utilizar una transformación de desplazamiento, las ecuaciones (2.31) muestran que w^i pertenece al rango de U_{j-1+d}, para $i = 0, \ldots, d-2$, y proporcionan expresiones con las que obtener los coeficientes g^i para representar cada una de las partes w^i en función de la base de TOAR. Éstas vienen dadas por,

$$\begin{cases} g^0 := (\beta_0 - \sigma)g_j^0 + \alpha_0 g_j^1, \\ g^i := \gamma_i g_j^{i-1} + (\beta_i - \sigma)g_j^i + \alpha_i g_j^{i+1}, \quad i = 1, \ldots, d-2. \end{cases} \tag{2.45}$$

Por otro lado, el cálculo del vector w^{d-1} se realiza explícitamente igual que en (2.31), para el método de Arnoldi Básico, sustituyendo, en este caso, cada v_j^i por $U_{j-1+d}g_j^i$, $i = 0, \ldots, d$–1. Típicamente, el vector calculado formará, junto con la base de TOAR, un sistema independiente de vectores, y en este caso, se utilizará para extender el conjunto ortonormal U_{j-1+d}. Para ello, se define

$$u_{j+d} := \frac{\hat{u}}{\|\hat{u}\|_2}, \quad \text{y} \quad g^{d-1} := \begin{bmatrix} \hat{g} \\ \|\hat{u}\|_2 \end{bmatrix}, \tag{2.46}$$

siendo $\hat{g} := U_{j-1+d}^* w^{d-1}$ y $\hat{u} := w^{d-1} - U_{j-1+d}\hat{g}$.

En el caso de realizar una transformación de desplazamiento e inversión, de forma similar se utilizan las recurrencias (2.33) y (2.36). En primer lugar se definen los coeficientes

$$\begin{cases} \tilde{g}^0 := \alpha_0^{-1} g_j^0, \\ \tilde{g}^1 := \alpha_1^{-1}(g_j^1 + (\sigma - \beta_1)\tilde{g}^0), \\ \tilde{g}^i := \alpha_i^{-1}(g_j^i + (\sigma - \beta_i)\tilde{g}^{i-1} - \gamma_i \tilde{g}^{i-2}), \quad i = 2, \ldots, d-2, \end{cases} \tag{2.47}$$

que se utilizan para calcular el vector t^{d-1} de igual forma que en (2.33), sustituyendo ahora cada t^i por $U_{j-1+d}\tilde{g}^i$ para $i = 0, \ldots, d-2$. En segundo lugar, este vector se utiliza para extender U_{j-1+d} (si no pertenece a su rango), definiendo

$$g^0 := \begin{bmatrix} \hat{g} \\ \|\hat{u}\|_2 \end{bmatrix} \text{ y } u_{j+d} := \frac{\hat{u}}{\|\hat{u}\|_2}, \tag{2.48}$$

donde $\hat{g} := U_{j-1+d}^* t^{d-1}$ y $\hat{u} := t^{d-1} - U_{j-1+d}\hat{g}$. Finalmente se obtienen los coeficientes g^i que expresan los vectores w^i en función de la base de TOAR, definiendo

$$g^i := \frac{\Phi_i(\sigma)}{\Phi_0(\sigma)} g^0 + \tilde{g}^{i-1}, \quad i = 1, \ldots, d-1. \tag{2.49}$$

2.3. Reinicio implícito

En muchas ocasiones, la convergencia en el método de Arnoldi es lenta y se requieren muchas iteraciones hasta obtener buenas aproximaciones de los pares propios deseados. Este hecho puede suponer un serio inconveniente cuando se resuelven problemas de gran dimensión, ya que por un lado, la cantidad de memoria necesaria para almacenar los vectores de Arnoldi que se generan puede ser demasiado elevada y, además, ésta no se conoce de antemano. Por otro lado, el coste de la ortogonalización se incrementa en cada iteración, ya que aumenta la dimensión de la base del subespacio de Krylov calculada. Es por ello que para tener control sobre los requerimientos de memoria y sobre el coste de la ortogonalización, este tipo de métodos son normalmente reiniciados.

Para los solvers basados en linealización que se describen en este capítulo, hemos optado por seguir un esquema de reinicio implícito como el utilizado en el solver lineal por defecto en SLEPc, que implementa el método de Krylov–Schur [84]. El reinicio en dicho método se basa en la utilización de relaciones de Krylov

$$SV_k = V_k C + v_{k+1} b^*, \tag{2.50}$$

que son una generalización de las relaciones de Arnoldi (2.5), en el sentido de que C y b no tienen una estructura particular, C no tiene por qué estar en forma Hessenberg, y el vector b puede ser distinto de e_k. Las bases que aparecen en las relaciones de Krylov son distintas de las que aparecen en Arnoldi, pero, al igual que estas últimas, también generan subespacios de Krylov.

El reinicio en Krylov–Schur utiliza el hecho de que, siempre que C en (2.50) tenga la forma $C = \left[\begin{smallmatrix} C_1 & \star \\ 0 & C_2 \end{smallmatrix}\right]$, con $C_1 \in \mathbb{C}^{p\times p}$ y $C_2 \in \mathbb{C}^{(k-p)\times(k-p)}$, la relación (2.50) puede ser truncada obteniendo otra de dimensión p, que puede ser de nuevo extendida hasta orden k. El objetivo es mantener una base de Krylov de dimensión acotada en la que se mantienen las direcciones más deseadas y de la que se eliminan las de menor interés. Los pasos seguidos por esta técnica de reinicio para truncar y reiniciar una relación (2.50) son:

1. Cálculo de la descomposición de Schur de C, ordenada de forma que los p valores de Ritz de mayor interés se mantienen en las primeras posiciones de la diagonal de T,
$$CQ_k = Q_k T. \tag{2.51}$$
En caso de utilizar aritmética real, la matriz T que se calcular es la forma real de Schur asociada a C, y el valor de p se debe escoger de forma que al truncar no se corte un bloque 2×2 de la diagonal.

2. Escribiendo $Q_k = [Q_p, Q']$, y descartando las columnas asociadas a Q', la relación de Krylov de orden k se trunca a otra de orden p,
$$S\tilde{V}_p = \tilde{V}_p T_1 + v_{k+1}\tilde{b}^*, \tag{2.52}$$
donde $T_1 = Q_p^* C Q_p$, $\tilde{V}_p = V_k Q_p$ y $\tilde{b} = Q_p^* b$.

3. Reinicio de la iteración de Arnoldi según el Algoritmo 2.1, expandiendo el subespacio de Krylov mediante la expresión $w = Sv_{k+1}$.

Cuando el solver polinómico sigue la variante que hemos denominado Arnoldi Básico, el esquema de reinicio implícito descrito se aplica directamente sin necesidad de modificación alguna. En el caso de Q-Arnoldi, también es posible utilizar el mismo enfoque para el reinicio, ya que la relación de Krylov (2.52), obtenida al truncar otra relación previa, también determina para las dos mitades de la matriz $\tilde{V}_p$, una relación análoga a (2.11) que permite obtener V_p^1 a partir de V_p^0 y v_{k+1}.

En el caso de TOAR, al aplicar los pasos 1 y 2 del reinicio a una relación de Arnoldi de orden k en formato TOAR (2.43), se obtiene una nueva relación en la que la matriz actualizada de los vectores de Krylov tiene la forma

$$\tilde{V}_p = (I_d \otimes U_{k+d}) G_k Q_p. \tag{2.53}$$

Esto significa que tras la actualización, únicamente la matriz G_k se modifica y trunca, mientras que la base U_{k+d} de TOAR permanece sin cambios, es decir, el número de columnas en U_{k+d} y de filas en $\tilde{G}_p := G_k Q_p$ siguen siendo $k+d$. Esto hace que para realizar el reinicio en esta variante sea necesario un paso adicional que reduzca los vectores de la base de TOAR a una dimensión acorde al número de columnas en $\tilde{G}_p$. Para conseguir dicha reducción, se ha seguido el enfoque descrito en [53], que utiliza el hecho de que a partir de la ecuación (2.52), con $\tilde{V}_p$ en la forma (2.53), se producen relaciones del tipo (2.16), que permiten deducir que la matriz

$$\Omega_{p+1} := \begin{bmatrix} \tilde{G}_p^0 & g_{k+1}^0 & \dots & \tilde{G}_p^{d-1} & g_{k+1}^{d-1} \end{bmatrix} \tag{2.54}$$

tiene el mismo rango que $\begin{bmatrix} \tilde{V}_p^0 & v_{k+1}^0 & \dots & \tilde{V}_p^{d-1} & v_{k+1}^{d-1} \end{bmatrix}$, que vale $p+d$. De esta forma es posible asegurar que la descomposición SVD compacta de la matriz (2.54),

$$\Omega_{p+1} = \check{U} \check{\Sigma} \check{V}^*, \tag{2.55}$$

tiene una matriz de valores singulares $\check{\Sigma}$ de dimensión menor o igual que $p+d$. Aunque en nuestra implementación se tiene en cuenta que esta dimensión puede ser menor que $p+d$, para mayor simplicidad en la exposición supondremos que ésta es $p+d$, y que $\check{U}$ y $\check{V}$ tienen dimensiones $(k+d) \times (p+d)$ y $d(p+1) \times (p+d)$, respectivamente. Dividiendo ahora la matriz $\check{V}^*$ en la forma

$$\check{V}^* = \begin{bmatrix} \hat{V}^0 & \dots & \hat{V}^{d-1} \end{bmatrix},$$

con $\hat{V}^i \in \mathbb{C}^{(p+d)\times(p+1)}$, en virtud de (2.55), se tiene

$$\begin{bmatrix} \tilde{G}_p^i & g_{k+1}^i \end{bmatrix} = \check{U} \check{\Sigma} \hat{V}^i, \quad i = 0, \dots, d-1, \tag{2.56}$$

por lo que actualizando las matrices

$$\begin{aligned} \hat{U}_{p+d} &\leftarrow U_{k+d} \check{U}, \\ \hat{G}_{p+1}^i &\leftarrow \check{\Sigma} \check{V}^i, \quad i = 0, \dots, d-1, \end{aligned} \tag{2.57}$$

obtenemos que los vectores de Krylov para (2.52) pueden ser representados ahora en la forma (2.16),

$$\begin{bmatrix}\tilde{V}_p^i & v_{k+1}^i\end{bmatrix} = U_{k+d}\begin{bmatrix}\tilde{G}_p^i & g_{k+1}^i\end{bmatrix} = \hat{U}_{p+d}\begin{bmatrix}\hat{G}_p^i & g_{k+1}^i\end{bmatrix}, \quad i = 0, \ldots, d-1, \tag{2.58}$$

siendo $\hat{G}_{p+1}^i$ y $\hat{U}_{p+d}$ matrices con $p+d$ filas y columnas, respectivamente.

2.3.1. Deflación de autovectores convergidos

La técnica de reinicio de Krylov-Schur [84] contempla la posibilidad de hacer deflación de los vectores de Krylov que generan subespacios invariantes ya convergidos. Con ello se consigue, por un lado, favorecer la convergencia de autovectores fuera de estos subespacios, especialmente en el caso de calcular autovectores asociados a grupos de autovalores con poca separación entre ellos o de multiplicidad mayor que uno. Por otro lado, se consigue la reducción del coste computacional ya que los vectores de Krylov asociados a los subespacios convergidos no vuelven a actualizarse en sucesivos reinicios (se bloquean).

Una vez obtenida una relación de Arnoldi truncada (2.52), si las primeras q entradas del vector $\tilde{b}$ son cero, dado que la matriz T_1 es triangular superior (a bloques si es real), se tiene que las q primeras columnas $\tilde{V}_q$ de $\tilde{V}_p$ generan un subespacio invariante. En el caso de que las columnas de $\tilde{V}_q$ generen un subespacio invariante aproximado, las correspondientes entradas de $\tilde{b}$ no serán cero, aunque sí muy pequeñas y podrían ser fijadas a cero. Veamos que con ello se consigue la deflación de los vectores $\tilde{V}_q$ y que éstos no vuelven ya a ser modificados en el siguiente reinicio. Al extender una relación truncada (2.52) a otra de orden k, la matriz del problema proyectado H_k resultante tiene la forma

$$H_k = \left[\begin{array}{cc|c} T_{11} & T_{12} & M_1 \\ 0 & T_{22} & M_2 \\ \hline 0 & b_2^* & \\ 0 & 0 & H \end{array}\right], \tag{2.59}$$

donde b_2 se corresponde con las últimas $(p-q)$ posiciones de $\tilde{b}$, la matriz T_1 se ha escrito en la forma

$$T_1 = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22}\end{bmatrix},$$

en la que la matriz T_{11} tiene dimensión $q\times q$, y H es una matriz Hessenberg superior. Así, para reducir H_k a forma triangular (a bloques), es posible utilizar una matriz unitaria con la forma

$$\tilde{Q} = \begin{bmatrix} I_q & 0 \\ 0 & Q\end{bmatrix}, \tag{2.60}$$

siendo Q también una matriz unitaria. De esta forma, se reduce en q unidades, la dimensión del problema proyectado a resolver, y también el número de vectores de Krylov que se actualizan, al multiplicar V_k por $\tilde{Q}$.

Al incluir en el método de TOAR la opción de bloquear los vectores de Krylov $\tilde{V}_q$ que generan un subespacio invariante aproximado, utilizando el esquema de Krylov–Schur, que fija a cero las q primeras posiciones del vector b en (2.52), se obtiene el mismo efecto en cuanto a la reducción de la dimensión del problema proyectado a resolver en el siguiente reinicio. Sin embargo, al actualizar los vectores de Krylov $V_k = (I_d \otimes U_{k+d})G_k$ representados implícitamente en formato TOAR, utilizando una matriz del tipo (2.60), los primeros q vectores de Krylov no se modifican, pero sí lo hacen las matrices U_{k+d} y G_k, en las operaciones (2.57), necesarias para adaptar la dimensión de la base de TOAR a la nueva dimensión de la base del subespacio de Krylov. Es decir, no se modifican los vectores de Krylov bloqueados pero sí su representación en formato TOAR, por lo que no se obtiene la reducción del coste computacional que se obtendría en otras variantes al realizar la operación de deflación. A continuación se explican las modificaciones incluidas en la variante con deflación de TOAR para obtener el bloqueo de vectores a nivel de la base de TOAR. En la proposición 2.1 se prueba que es posible expresar los vectores de Krylov que generan subespacios invariantes en formato TOAR de forma que los vectores de TOAR implicados no se modifiquen en sucesivos reinicios.

Proposición 2.1. *Dada una relación de Krylov de orden k, generada mediante la variante de TOAR, utilizando la linealización* (2.21)*,*

$$SV_k = V_k T_k + v_{k+1} b^*, \tag{2.61}$$

con $V_k = (I_d \otimes U_{k+d})G_k$ y T_k triangular superior (a bloques), con estructura

$$T_k = \begin{array}{c} \\ {\scriptstyle q} \\ {\scriptstyle p-q} \\ {\scriptstyle k-p} \end{array} \begin{array}{c} \begin{array}{ccc} {\scriptstyle q} & {\scriptstyle p-q} & {\scriptstyle k-p} \end{array} \\ \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ & T_{22} & T_{23} \\ & & T_{33} \end{bmatrix} \end{array}, \tag{2.62}$$

de forma que las q primeras entradas de b son cero. Entonces, se tiene que existen matrices $\tilde{U}_{p+d} \in \mathbb{C}^{n\times(p+d)}$ y $\tilde{G}_{p+1} \in \mathbb{C}^{d(p+d)\times(p+1)}$ con

$$\tilde{G}^i_{p+1} = \begin{bmatrix} \tilde{G}^i_{11} & \tilde{G}^i_{12} \\ 0 & \tilde{G}^i_{22} \end{bmatrix} \tag{2.63}$$

y $\tilde{G}^i_{11} \in \mathbb{C}^{q\times q}$, para $i = 0, \ldots, d-1$, tales que $\begin{bmatrix} V_p & v_{k+1} \end{bmatrix} = (I_d \otimes \tilde{U}_{p+d})\tilde{G}_{p+1}$.

Demostración. Dado que las q primeras entradas de b son cero, definiendo $T_q := T_{11}$, y $G_q \in \mathbb{C}^{d(k+d)\times q}$ como las q primeras columnas de G_k, se tiene que se cumple la relación

$$S(I_d \otimes U_{k+d})G_q = (I_d \otimes U_{k+d})G_q T_q. \tag{2.64}$$

Teniendo en cuenta la estructura por bloques de S, la ecuación (2.64) proporciona relaciones entre las filas de bloques de V_q, $V^i_q = U_{k+d}G^i_q$, de las que se deduce que tanto el rango de $\begin{bmatrix} V^0_q & \ldots & V^{d-1}_q \end{bmatrix}$, como el de $\Omega_1 := \begin{bmatrix} G^0_q & \ldots & G^{d-1}_q \end{bmatrix}$, es q. Así, procediendo de forma similar al reinicio en TOAR descrito en 2.3, utilizando

la descomposición SVD compacta de $\Omega_1 = \check{U}_1\check{\Sigma}_1\check{V}_1^*$, y dividiendo $\check{V}_1^*$ en la forma $\check{V}_1^* = \begin{bmatrix}\check{V}_1^0 & \dots & \check{V}_1^{d-1}\end{bmatrix}$, se tiene

$$V_q^i = U_{k+d}G_q^i = U_{k+d}\check{U}_1\check{\Sigma}_1\check{V}_1^i, \quad i = 0, \dots, d-1. \tag{2.65}$$

Por otro lado, al truncar (2.61) a otra relación de orden p, dando como resultado

$$S(I_d \otimes U_{k+d})G_p = (I_d \otimes U_{k+d})G_pT_p + U_{k+d}g_{k+1}b_p^*, \tag{2.66}$$

donde b_p representa los p primeros elementos de b, se tiene que la matriz

$$\begin{bmatrix}G_p^0 & g_{k+1}^0 & \dots & G_p^{d-1} & g_{k+1}^{d-1}\end{bmatrix}$$

tiene rango $p+d$. También, si dividimos $G_p = \begin{bmatrix}G_q & G_2\end{bmatrix}$, considerando

$$\Omega_2 := \begin{bmatrix}G_2^0 & g_{k+1}^0 & \dots & G_2^{d-1} & g_{k+1}^{d-1}\end{bmatrix}$$

y $\tilde{\Omega}_2 := (I_{k+d} - \check{U}_1\check{U}_1^*)\Omega_2$, se sigue que $\tilde{\Omega}_2$ tendrá rango (como máximo) $(p+d-q)$. Así, considerando la SVD compacta de $\tilde{\Omega}_2 = \check{U}_2\check{\Sigma}_2\check{V}_2^*$, tenemos

$$\Omega_2 = \check{U}_2\check{\Sigma}_2\check{V}_2^* + \check{U}_1\check{U}_1^*\Omega_2,$$

por lo que dividiendo $\check{V}_2^*$ como $\check{V}_2^* = \begin{bmatrix}\check{V}_2^0, \dots, \check{V}_2^{d-1}\end{bmatrix}$, con $\check{V}_2^i \in \mathbb{C}^{(p+d-q)\times(p+1)}$, podemos concluir que

$$\begin{aligned} V_p^i &= U_{k+d}\begin{bmatrix}G_q^i & G_2^i\end{bmatrix} = U_{k+d}\begin{bmatrix}\check{U}_1\check{\Sigma}_1\check{V}_1^i & \check{U}_2\check{\Sigma}_2\check{V}_2^i + \check{U}_1\check{U}_1^*\begin{bmatrix}G_2^i & g_{k+1}^i\end{bmatrix}\end{bmatrix} \\ &= U_{k+d}\begin{bmatrix}\check{U}_1 & \check{U}_2\end{bmatrix}\check{G}^i \end{aligned} \tag{2.67}$$

para

$$\check{G}^i := \begin{bmatrix}\check{\Sigma}_1\check{V}_1^i & \check{U}_1^*\begin{bmatrix}G_2^i & g_{k+1}^i\end{bmatrix} \\ 0 & \check{\Sigma}_2\check{V}_2^i\end{bmatrix}.$$

Finalmente, el resultado se sigue de (2.67), definiendo

$$\begin{aligned} &\tilde{U}_{p+d} := U_{k+d}\begin{bmatrix}\check{U}_1 & \check{U}_2\end{bmatrix} \text{ y} \\ &\tilde{G}_{p+1}^i := \check{G}^i, \quad i = 0, \dots, d-1. \end{aligned} \tag{2.68}$$

□

Notas a la Proposición 2.1

1. Por simplicidad, se ha supuesto que la base de TOAR, con la que se expresan los vectores $[V_p, v_{k+1}]$, es de dimensión $p+d$. Sin embargo, tal como se comentó en las anotaciones del Algoritmo 2.2, la implementación del método realizada en SLEPc tiene en cuenta que la dimensión de dicha base puede ser menor.

2. En aritmética de precisión finita, podría suceder que la base $\begin{bmatrix}\check{U}_1 & \check{U}_2\end{bmatrix}$, obtenida a partir de dos factorizaciones SVD distintas, no sea completamente ortogonal, por ello, en la implementación del reinicio realizada, se utiliza la factorización QR, $\begin{bmatrix}\check{U}_1 & \check{U}_2\end{bmatrix} = \check{Q}\check{R}$, para definir la base de TOAR en la forma $\tilde{U}_{p+d} := U_{k+d}\check{Q}$ en lugar de la expresada en (2.68), y se actualiza la matriz de coeficientes de TOAR de forma acorde con este cambio, $\tilde{G}^i_{p+1} := \check{R}\check{G}^i$, $i = 0, \ldots, d-1$.

Supongamos ahora que en las condiciones de la Proposición 2.1, tras expresar los vectores V_k según la nueva base de TOAR determinada por la proposición, la relación (2.61) se trunca a otra de orden p para ser nuevamente extendida a orden k. En esta situación, se tendría una relación similar a (2.61), con una matriz proyectada del tipo (2.59), y vectores de Krylov $\begin{bmatrix}V_k & v_{k+1}\end{bmatrix} = (I_d \otimes U_{k+d})\begin{bmatrix}G_k & g_{k+1}\end{bmatrix}$ en los que la matrices G^i_k tienen la forma (2.63). Si en este caso se realizan las operaciones de reinicio, que incluyen una actualización de la base V_k utilizando una matriz unitaria truncada de forma

$$\tilde{Q}_t = \begin{bmatrix} I_q & 0 \\ 0 & Q \end{bmatrix}\begin{bmatrix} I_p \\ 0 \end{bmatrix} = \begin{bmatrix} I_q & 0 \\ 0 & Q_t \end{bmatrix},$$

donde Q_t representa las primeras $p-q$ columnas de Q, entonces, para cada $i = 0, \ldots, d-1$, se verificaría la siguiente relación,

$$\begin{bmatrix}V^i_k\tilde{Q}_t & v_{k+1}\end{bmatrix} = U_{k+d}\begin{bmatrix}G^i_k\tilde{Q}_t & g^i_{k+1}\end{bmatrix} = U_{k+d}\begin{bmatrix} G^i_{11} & G^i_{12}Q_t & g^i_1 \\ 0 & G^i_{22}Q_t & g^i_2 \end{bmatrix}, \tag{2.69}$$

donde, para cada $i = 0, \ldots, d-1$, se ha utilizado la notación

$$g^i_{k+1} = \begin{bmatrix} g^i_1 \\ g^i_2 \end{bmatrix}, \text{ con } g^i_1 \in \mathbb{C}^{q\times 1}.$$

Así, definiendo $\Omega := \begin{bmatrix}G^0_{22}Q_t & g^0_2 & \cdots & G^{d-1}_{22}Q_t & g^{d-1}_2\end{bmatrix}$, que tiene rango, como máximo, igual a $(p-q+d)$, y considerando ahora la SVD de $\Omega = \check{U}\check{\Sigma}\check{V}^*$, se tiene

$$\begin{bmatrix}V^i_k\tilde{Q}_t & v_{k+1}\end{bmatrix} = \tilde{U}_{p+d}\begin{bmatrix} G^i_{11} & \begin{bmatrix}G^i_{12}Q_t & g^i_1\end{bmatrix} \\ 0 & \check{\Sigma}\check{V}^i \end{bmatrix}, \tag{2.70}$$

donde $\check{V}^* = \begin{bmatrix}\check{V}^0 & \cdots & \check{V}^{d-1}\end{bmatrix}$, y

$$\tilde{U}_{p+d} := U_{k+d}\begin{bmatrix} I_q & 0 \\ 0 & \check{U} \end{bmatrix} \in \mathbb{C}^{n\times(p+d)}.$$

De esta forma, cuando se bloquean los vectores de Krylov implícitos V_q, también lo hacen los primeros vectores U_q de TOAR, que se mantendrán sin actualizar en los subsiguientes reinicios. Con ello se consigue que en cada reinicio se reduzca, por un lado, el número de vectores a actualizar, y por otro, la dimensión de la SVD que se realiza.

2.4. Escalado de problemas polinómicos

Dado un polinomio matricial $P(\lambda)$, en [44] se prueba que distintas linealizaciones para P tendrán, en general, diferentes características en cuanto al condicionamiento de sus autovalores, o dicho de otro modo, en la variabilidad de los mismos ante perturbaciones de los datos de entrada (las matrices de la linealización). A la hora de resolver computacionalmente un problema de valores propios (o cualquier problema en general), es importante, tratar de mejorar, en la medida de lo posible, el condicionamiento del problema, ya que la solución final obtenida puede estar lejos de la solución exacta si dicho número de condición es elevado.

Bectke [13] propone una técnica de escalado que actúa sobre el parámetro λ del problema polinómico (1.2), con el que se mejora el condicionamiento de los valores propios asociados a la linealización compañera dada en (2.21), particularizada para base de monomios. En esta sección se describe brevemente en qué consiste dicha técnica de escalado y la forma en que ésta ha sido implementada, sin modificar las matrices iniciales. Aun no siendo específico para la resolución de problemas polinómicos vía linealización, también se incluye aquí la descripción de otro tipo de escalado, propuesto por Betcke en el mismo artículo, con el que se influye en el condicionamiento de los autovalores del problema polinómico mediante un escalado diagonal de las matrices que lo definen.

Escalado en el parámetro de la λ-matriz. En este enfoque se realiza un cambio de escala en el parámetro del polinomio matricial $P(\lambda)$ mediante el cambio de variable $\lambda = \rho\theta$, tomando $\rho := \sqrt[d]{\|A_0\|_2/\|A_d\|_2}$. El problema asociado al polinomio $\tilde{P}(\theta) := P(\rho\theta)$ producido por dicho cambio, tiene los mismos vectores propios que el problema original, mientras que los autovalores respectivos se relacionan según $\theta = \lambda/\rho$. Esta estrategia de escalado, descrita en [13], es específica para problemas polinómicos (descritos mediante la base de monomios) que se resuelven utilizando la linealización compañera. Con ella se busca minimizar el número de condición de los autovalores que se calculan en la linealización. Es una generalización, para problemas polinómicos, de la técnica propuesta en [30] por Fan, Lin, y Van Dooren para problemas de valores propios cuadráticos. Este último esquema también incluye un factor de escalado global

$$\delta_2 := \frac{2}{\|A_0\|_2 + \|A_1\|_2\sqrt{\|A_0\|_2/\|A_2\|_2}}, \tag{2.71}$$

con el que se escalan las tres matrices que definen el problema cuadrático, de forma que las matrices asociadas al problema transformado tienen norma próxima a uno.

En la implementación para este esquema de escalado, realizada en SLEPc, se utiliza la ∞-norma en lugar de la 2-norma,

$$\rho := \sqrt[d]{\frac{\|A_0\|_\infty}{\|A_d\|_\infty}}, \tag{2.72}$$

y se incluye, también para polinomios de grado mayor que 2, un factor de escalado global dado por

$$\delta := \frac{d}{\|A_0\|_\infty + \|\rho A_1\|_\infty + \cdots + \|\rho^{d-1} A_{d-1}\|_\infty}. \tag{2.73}$$

Este valor se corresponde con el recíproco de la media de las normas de las matrices transformadas según el escalado en el parámetro (excepto A_d) y coincide para $d = 2$ con la expresión (2.71) si se utiliza la 2-norma. Aunque en los solvers polinómicos de `PEP` la opción de cálculo del factor de escala está disponible sólo en el caso de que el problema se exprese en términos de la base de monomios, se ha incluido también la posibilidad de realizar un escalado sobre el parámetro del autovalor siempre que el mencionado factor sea proporcionado por el usuario.

La operación de escalado se lleva a cabo sin modificar las matrices del problema inicial. En el caso de usar la base de monomios, el polinomio resultante del cambio de variable es $P(\theta) = \sum_{j=0}^{d} \theta^j (\rho^j A_j)$, por lo que es suficiente escalar convenientemente cada vector resultado en las multiplicaciones matriz-vector con las matrices A_j. Para el caso general, se procede de igual forma, pero escalando además los coeficientes que intervienen en la definición de la base utilizada.

Proposición 2.2. *Sean $\{\Phi_j\}_{j\in\mathbb{N}}$ y $\{\tilde{\Phi}_j\}_{j\in\mathbb{N}}$ dos bases de polinomios que satisfacen sendas recurrencias de 3 términos* (2.20)*, definidas cada una de ellas, respectivamente por los coeficientes $\{\{\alpha_j\},\{\beta_j\},\{\gamma_j\}\}_{j\in\mathbb{N}}$ y $\{\{\tilde{\alpha}_j\},\{\tilde{\beta}_j\},\{\tilde{\gamma}_j\}\}_{j\in\mathbb{N}}$, tales que verifican las relaciones $\tilde{\alpha}_j = \alpha_j$, $\tilde{\beta}_j = \beta_j/\rho$, y $\tilde{\gamma}_j = \gamma_j/\rho^2$, $j \in \mathbb{N}$. Entonces, se cumple que $\Phi_j(\lambda) = \tilde{\Phi}_j(\theta)\rho^j$, $j \in \mathbb{N}$.*

Demostración. La prueba se hace por inducción sobre el índice j. En primer lugar, dado que por un lado se tiene $\Phi_0(\lambda) = 1 = \tilde{\Phi}_0(\theta)$, y también que

$$\Phi_1(\lambda) = \alpha_0^{-1}(\lambda - \beta_0) = \alpha_0^{-1}(\rho\theta - \beta_0) = \alpha_0^{-1}(\theta - \frac{\beta_0}{\rho})\rho = \tilde{\Phi}_1(\theta)\rho,$$

comprobamos que el enunciado se cumple para $j = 0$ y $j = 1$.

Suponiendo ahora que $\Phi_j(\lambda) = \tilde{\Phi}_j(\theta)\rho^j \quad \forall j \le k$, se tiene entonces

$$\begin{aligned}
\Phi_{k+1}(\lambda) &= \alpha_k^{-1}((\lambda - \beta_k)\,\Phi_k(\lambda) - \gamma_k\,\Phi_{k-1}(\lambda)) = \\
&= \alpha_k^{-1}\big((\theta - \frac{\beta_k}{\rho})\tilde{\Phi}_k(\theta)\rho^k - \frac{\gamma_k}{\rho}\tilde{\Phi}_{k-1}\rho^{k-1}\big)\rho = \\
&= \alpha_k^{-1}\big((\theta - \frac{\beta_k}{\rho})\tilde{\Phi}_k(\theta) - \frac{\gamma_k}{\rho^2}\tilde{\Phi}_{k-1}(\theta)\big)\rho^{k+1} = \tilde{\Phi}_{k+1}(\theta)\rho^{k+1},
\end{aligned}$$

por lo que la proposición es cierta para $k+1$, y por tanto para todo $j \in \mathbb{N}$. □

Utilizando la Proposición 2.2, se tiene la siguiente expresión para el polinomio transformado,

$$\tilde{P}(\theta) = P(\rho\theta) = \sum_{j=0}^{d} \Phi_j(\rho\theta)A_j = \sum_{j=0}^{d} \tilde{\Phi}_j(\theta)\rho^j A_j,$$

con $\tilde{\Phi}_j$ relacionada con Φ_j según dicha proposición. Por ello, en el caso general se procede escalando los coeficientes $\tilde{\alpha}_j = \alpha_j$, $\tilde{\beta}_j = \beta_j/\rho$, y $\tilde{\gamma}_j = \gamma_j/\rho^2$ y se escala implícitamente las matrices A_j en la misma forma mencionada para el caso de base de monomios, escalando el vector resultado de operar con dichas matrices. Además, para la base de monomios se tiene que $\tilde{\Phi}_j = \Phi_j$ y ambos procedimientos coinciden.

Escalado diagonal de la matriz polinómica. En este esquema, el polinomio matricial del problema original $P(\lambda)$ se transforma mediante la multiplicación del mismo, a ambos lados, por dos matrices diagonales no singulares D_1 y D_2,

$$\tilde{P}(\lambda) := D_1 P(\lambda) D_2. \tag{2.74}$$

El problema polinómico resultante tiene los mismos autovalores que el asociado a P, mientras que los autovectores x y $\tilde{x}$ para P y $\tilde{P}$, respectivamente, se relacionan según la expresión $\tilde{x} = D_2^{-1}x$. Esta técnica de escalado, descrita en [13], está enfocada a minimizar el número de condición asociado a cada autovalor particular λ, por lo que las matrices que se utilizan dependen del conjunto de autovalores que se pretende obtener. Betcke describe un algoritmo para el cálculo eficiente de las matrices D_1 y D_2, para el que se requiere una estimación de la magnitud de los autovalores a calcular. Dicho algoritmo ha sido implementado en SLEPc, aunque dado el coste computacional extra que supone el cálculo de las matrices D_1 y D_2, esta característica está disponible como opción, para ser utilizado en los casos en los que el mal condicionamiento de los autovalores afecte a su cálculo.

Al igual que en la opción de escalado sobre el parámetro, la obtención de las matrices D_1 y D_2 sólo está disponible cuando el polinomio matricial está definido mediante la base de monomios, aunque también existe la posibilidad de efectuar el escalado diagonal en el caso general, siempre y cuando las matrices (sólo su diagonal) sean proporcionadas por el usuario. También para este esquema de escalado, las matrices iniciales no se modifican, sino que se actúa sobre los vectores que intervienen en la multiplicación con éstas.

2.5. Extracción de pares propios

En esta sección se aborda el problema de cómo obtener vectores propios de un problema polinómico (1.9), a partir de los obtenidos en la resolución de la linealización (2.21) asociada. Se presentan varias estrategias que han sido incorporadas en los solvers polinómicos de SLEPc. Todas ellas son adaptaciones para polinomios descritos en la forma genérica (1.9), de las propuestas en [15] para problemas definidos según la base de monomios. Las técnicas de extracción descritas en [15] están expresadas en términos de pares invariantes, que son una generalización, en el contexto de PEPs, de la noción de subespacio invariante en problemas lineales.

2.5.1. Pares invariantes

La obtención de pares invariantes en problemas polinómicos, al igual que la de subespacios invariantes en problemas lineales, presenta ventajas numéricas frente al cálculo de autovectores aislados, especialmente en presencia de valores propios múltiples. Dicho concepto ha sido definido en [51] para problemas no lineales y en [15] para polinómicos del tipo (1.2).

Definición 2.3. Dado un polinomio matricial P definido como en (1.9), un par de matrices $(X, H) \in \mathbb{C}^{n\times k} \times \mathbb{C}^{k\times k}$, se dice un par invariante para P si éstas verifican

$$\mathbb{P}(X, H) := A_0 X\, \Phi_0(H) + A_1 X\, \Phi_1(H) + \cdots + A_d X\, \Phi_d(H) = 0, \tag{2.75}$$

donde $\Phi_i(H)$ representa la función matricial definida por el polinomio Φ_i [41].

Considerando de nuevo el problema lineal $Sz = \lambda z$, se tiene que todo subespacio invariante generado por las columnas de una matriz $V \in \mathbb{C}^{n\times k}$ de rango por columnas completo, determina de forma única una matriz $H \in \mathbb{C}^{k\times k}$ de forma que $SV = VH$, y todo autovalor de H lo es también de S. Dado un par invariante (X, H) asociado a un problema polinómico, en [51, 15] se prueba una propiedad similar sobre los autovalores de H, exigiendo que el par invariante considerado sea minimal, es decir, que la matriz

$$\begin{bmatrix} X \\ XH \\ \vdots \\ XH^{m-1} \end{bmatrix}, \tag{2.76}$$

tenga rango completo, para algún $m \geq 1$. En el contexto de matrices polinómicas expresadas en la forma (1.9), utilizamos un concepto similar para asegurar que los autovalores de H, para un par invariante (X, H) obtenido vía la linealización (2.21), son también valores propios del problema polinómico.

Proposición 2.4. *Sea (Z, H) un par invariante del problema linealizado* (2.21) *tal que Z tiene rango por columnas completo, entonces se cumple:*

1. *$Z = V_d(Z^0, H)$, donde $V_m(X, H)$ se define para $m \in \mathbb{N}$, $X \in \mathbb{C}^{n\times k}$ y $H \in \mathbb{C}^{k\times k}$ como*

$$V_m(X, H) := \begin{bmatrix} X \\ X\, \Phi_1(H) \\ \vdots \\ X\, \Phi_{m-1}(H) \end{bmatrix}. \tag{2.77}$$

2. *(Z^0, H) es un par invariante para el problema polinómico* (1.9) *de forma que cada autovalor de H lo es también del problema definido por* (1.9).

Demostración. Dado que (Z, H) es un par invariante para la linealización, se tiene que $AZ - BZH = 0$ para A y B dados en (2.21). Igualando los primeros d–1 bloques

de filas de esta ecuación obtenemos

$$\begin{cases} Z^1 = \alpha_0^{-1}(Z^0 H - \beta_0 Z^0), \\ Z^i = \alpha_{i-1}^{-1}(Z^{i-1} H - \beta_{i-1} Z^{i-1} - \gamma_{i-1} Z^{i-2}), \quad i = 2, \ldots, d-1, \end{cases} \tag{2.78}$$

lo que nos permite probar fácilmente por inducción que

$$Z^i = Z^0\, \Phi_i(H), \quad i = 0, 1, \ldots, d-1. \tag{2.79}$$

Para probar la segunda afirmación, igualando el último bloque de filas de la ecuación $AZ - BZH = 0$ se obtiene

$$\begin{aligned} 0 &= -c_{d-1} \sum_{i=0}^{d-1} A_i Z^i + c_d \gamma_{d-1} A_d Z^{d-2} + c_d \beta_{d-1} A_d Z^{d-1} - c_d A_d Z^{d-1} H \\ &= -c_{d-1} \sum_{i=0}^{d-1} A_i Z^0\, \Phi_i(H) - c_d A_d Z^0 (\Phi_{d-1}(H) H - \beta_{d-1}\, \Phi_{d-1}(H) - \gamma_{d-1}\, \Phi_{d-2}(H)) \\ &= -c_{d-1} \sum_{i=0}^{d-1} A_i Z^0\, \Phi_i(H) - c_d \alpha_{d-1} A_d Z^0\, \Phi_d(H) = -c_{d-1} \sum_{i=0}^{d} A_i Z^0\, \Phi_i(H), \end{aligned} \tag{2.80}$$

de donde se concluye que (Z^0, H) es un par invariante para (1.9). Por otro lado, dado un valor propio λ de H, éste lo es también de la linealización (Z tiene rango completo), y por tanto del problema polinómico. □

Proposición 2.5. *Sea (X, H) un par invariante asociado al polinómico determinado por* (1.9)*, entonces $Z := V_d(X, H)$ es un par invariante para la linealización* (2.21)*.*

Demostración. La prueba es una simple verificación de la igualdad $AZ = BZH$. Al igual que en la Proposición 2.4, los primeros $d-1$ bloques de filas de esta igualdad se pueden comprobar utilizando la recurrencia (2.20), y la última, usando el hecho de que (X, H) satisface la condición de par invariante (2.75). □

En general, cuando tratemos con pares invariantes para un polinomio (1.9) obtenidos vía la linealización (2.21), nos será útil la siguiente definición de minimalidad.

Definición 2.6. Diremos que un par invariante (X, H) para (1.9) es Φ-minimal si $V_d(X, H)$ dado en (2.77), tiene rango completo.

Esta definición coincide con la de (2.76) cuando se considera la base de monomios, y tal como se prueba en [27] ambas definiciones son equivalentes.

Como consecuencia de las Proposiciones 2.4 y 2.5 se tiene que, dado un par invariante (X, H) Φ-minimal del problema polinómico definido por (1.9), cada autovalor de H lo es también del problema polinómico. Este resultado es similar al obtenido en [15] para pares invariantes minimales asociados al problema polinómico definido por bases de monomios (1.2).

La proposición 2.4 establece que siempre es posible extraer un par invariante Φ-minimal para el problema polinómico dado por (1.9), a partir de uno obtenido para la linealización asociada, (Z, H), simplemente tomando el primer bloque Z^0. A continuación veremos que en la mayoría de los casos cada bloque Z^i puede ser parte de un par invariante y que es posible utilizar otras alternativas de extracción que utilicen todos los bloques.

Proposición 2.7. *Dado un par invariante (Z, H) de la linealización (2.21) con Z de rango completo, y un valor $i \in \{0, \ldots, d-1\}$, se tiene que (Z^i, H) es un par invariante Φ-minimal para el problema polinómico sí y sólo sí $\Phi_i(H)$ es no singular.*

Demostración. En primer lugar, probamos por inducción sobre el índice de bloque de filas, i, que $(Z^i, H) = (Z^0\,\Phi_i(H), H)$ es un par invariante para (1.9). Por la Proposición 2.4 sabemos que es cierto para $i = 0$, es decir

$$\mathbb{P}(Z^0, H) = 0. \tag{2.81}$$

Postmultiplicando ahora (2.81) por H, restando β_0 veces (2.81), y escalando por α_0^{-1}, obtenemos

$$\begin{aligned} 0 &= \alpha_0^{-1}\left(A_0 Z^0 H - A_0 Z^0 \beta_0 + \cdots + A_d Z^0\,\Phi_d(H) H - A_d Z^0\,\Phi_d(H)\beta_0\right) = \\ &= A_0 Z^0\,\Phi_1(H) + \cdots + A_d Z^0\,\Phi_1(H)\,\Phi_d(H) = \mathbb{P}(Z^0\,\Phi_1(H), H). \end{aligned}$$

Con ello se tiene probado el resultado para $i = 1$. Procediendo de forma similar probaríamos que $\mathbb{P}(Z^0\,\Phi_i(H), H) = 0$, asumiendo que tanto $(Z^0\,\Phi_{i-2}(H), H)$ como $(Z^0\,\Phi_{i-1}(H), H)$ son pares invariantes. Por otro lado, para V_d dado en (2.77) y para $i = 1, \ldots, d-1$, se tiene que $\operatorname{rank} V_d(Z^i, H) = \operatorname{rank} V_d(Z^0, H)\,\Phi_i(H)$. Por tanto (Z^i, H) es Φ-minimal sí y sólo sí $\Phi_i(H)$ es no singular. □

2.5.2. Esquemas de extracción de pares invariantes

Una vez se ha obtenido, utilizando uno de los solvers de la sección 2.2, un subespacio invariante (aproximado) Z de la linealización (2.21), si H representa el cociente de Rayleigh asociado a Z, en virtud de la proposición 2.7, se tiene que siempre que $\Phi_i(H)$ sea no singular (esto es, los autovalores de H no son raíces de Φ_i), el par (Z^i, H) puede ser considerado un par invariante (aproximado) Φ-minimal del problema polinómico. A continuación se muestran las diferentes estrategias de extracción incluidas para los solvers polinómicos basados en linealización de SLEPc.

Ninguna Para formar el par invariante se utiliza el primer bloque de Z. Esta es la estrategia utilizada por defecto cuando el problema polinómico está definido en la forma genérica (1.9).

Residuo En esta estrategia, se escoge el bloque Z^i que minimiza la norma del residuo,

$$\rho(Z^i, H) := \frac{\|\mathbb{P}(Z^i, H)\|_F}{\|Z^i\|_F}. \tag{2.82}$$

Norma Esta es la estrategia por defecto en el caso de problemas definidos utilizando la base de monomios. Para formar el par invariante en estos problemas, se escoge el bloque Z^{d-1} si $\|H\| > 1$ o Z^0 en otro caso. De esta forma se selecciona el bloque con mayor norma. En la extensión de este enfoque, para los problemas definidos en la forma genérica (1.9), se calculan las normas $\|\Phi_i(H)\|$, para $i = 1, \ldots, d-1$, y se selecciona el de mayor valor. En el caso de resolución mediante el método de TOAR, dado que $Z^i = UG^i$, con U de columnas ortogonales, es suficiente calcular las normas $\|G^i\|$, lo que simplifica el proceso, ya que en este caso no se realizan cálculos con vectores distribuidos en paralelo.

Estructurada Esta estrategia se basa en el hecho de que un par invariante para la linealización (2.21) tiene la forma (2.77), y busca una matriz $X \in \mathbb{C}^{n\times k}$ que minimice $\|V_d(X,H) - Z\|_F$. Una ligera modificación de la demostración en [15, teorema 11], nos proporciona la siguiente expresión de la solución,

$$X = \left(\sum_{i=0}^{d-1} Z^i\, \Phi_i(H)^*\right) \left(\sum_{i=0}^{d-1} \Phi_i(H)\, \Phi_i(H)^*\right)^{-1}. \tag{2.83}$$

Este enfoque está disponible sólo en el caso de utilizar el método de TOAR para la resolución del problema polinómico, para evitar el sobrecoste que supondría en otro caso. Para la obtención de X se utilizan los bloques G^i de la representación de TOAR $Z^i = UG^i$, con lo que la ecuación (2.83) toma la forma

$$X = U\left(\sum_{i=0}^{d-1} G^i\, \Phi_i(H)^*\right) \left(\sum_{i=0}^{d-1} \Phi_i(H)\, \Phi_i(H)^*\right)^{-1}. \tag{2.84}$$

Una vez extraído un par invariante (X, H) asociado a (1.9), los pares propios del problema polinómico vienen dados por (Xy, λ), siendo (y, λ) un par propio de H.

2.6. Evaluación computacional de los solvers basados en linealización

En esta sección se presentan resultados obtenidos en las pruebas realizadas para evaluar la exactitud y el rendimiento computacional de los solvers polinómicos descritos en este capítulo, en especial de la variante TOAR, por ser el principal objeto de estudio en este capítulo. Para la variante Q-Arnoldi, también se pueden ver resultados de su evaluación en el capítulo 3 que es específico para problemas cuadráticos.

2.6.1. Entorno de ejecución

El sistema de computación utilizado para los experimentos es el supercomputador *Tirant*, situado en la Universidad de Valencia, que forma parte de la Red Española de Supercomputación.

Red Española de Supercomputación
Servei d'Informatica
Universitat de València
TIRANT
supercomputador

El supercomputador Tirant es un clúster IBM compuesto por 512 nodos de cómputo JS21 en blade, cada uno de ellos con 2 procesadores PowerPC 970MP dual core de 64-bits, con una frecuencia de funcionamiento de 2.2 GHz y con 4 GB de RAM (2048 cores y 2TByte en total). La interconexión de los nodos se realiza mediante una red Myrinet de baja latencia, y dispone un sistema de ficheros distribuido GPFS, con 44 TB de almacenamiento. Tirant funciona utilizando un sistema operativo basado en Linux (SUSE Linux Enterprise Server 10).

El software utilizado han sido las versiones 3.6 de PETSc y SLEPc, junto con MUMPS 5.0 que se utiliza en la resolución de sistemas lineales mediante métodos directos. Las compilaciones se han realizado con el compilador de GNU, gcc-4.6.1, y se ha utilizado MPICH2 para la implementación de MPI.

Todas las ejecuciones se han realizado con un único proceso MPI por nodo, utilizando hasta un máximo de 256 procesos MPI.

2.6.2. Medida del error en los resultados

En la resolución numérica de problemas, la exactitud que cabe esperar en los resultados que se obtienen viene determinada tanto por el condicionamiento del problema como por la estabilidad del algoritmo utilizado. En álgebra lineal numérica, es frecuente evaluar la estabilidad de los algoritmos en función del error hacia atrás de los resultados que éstos producen. Este último concepto hace referencia a la proximidad existente entre el problema original y otro problema para el que la solución calculada sea la exacta. El error de la solución calculada con respecto de la solución exacta (o error hacia adelante), se relaciona con el error hacia atrás de la solución y con el número de condición del problema según la siguiente desigualdad [40],

$$\text{Error hacia adelante} \leq \text{Número de condición} \times \text{Error hacia atrás}. \tag{2.85}$$

Para evaluar la calidad y estabilidad de los algoritmos implementados, se estudia el error hacia atrás (relativo) de los pares propios calculados, que se define para un par propio aproximado derecho (x, λ) de un problema polinómico de valores propios del tipo (1.9) mediante

$$\eta_P(x,\lambda) = \min\{\epsilon : (P(\lambda) + \Delta P(\lambda))\, x = 0,\ \|\Delta A_j\|_2 \leq \epsilon \|A_j\|_2,\ j = 0, \ldots, d\}, \tag{2.86}$$

donde, dadas $\{\Delta A_i\}_{i=0}^d$ el polinomio ΔP se define por

$$\Delta P := \sum_{i=0}^{d} \Phi_i(\lambda)\Delta A_i. \tag{2.87}$$

Tisseur [86, Teorema 1] da una expresión explícita de dicho error para un problema polinómico definido en términos de la base de monomios (1.2). Basándonos en la prueba realizada por Tisseur, en la siguiente proposición obtenemos una expresión del error hacia atrás para el caso general con otras bases de polinomios.

Proposición 2.8. *El error hacia atrás de un par propio derecho aproximado* (x, λ) *de* P *definido en* (1.9)*, viene dado por la fórmula*

$$\eta_P(x, \lambda) = \frac{\|P(\lambda)x\|_2}{\left(\sum_{i=0}^{d} |\, \Phi_i(\lambda)| \|A_i\|_2\right) \|x\|_2}. \tag{2.88}$$

Demostración. Denotamos por $\alpha_P(x, \lambda)$ la parte derecha de la ecuación (2.88). Consideramos, en primer lugar, $\epsilon > 0$ y $\{\Delta A_j\}_{j=0}^d$ tales que $\|\Delta A_j\|_2 \leq \epsilon \|A_j\|_2$ y

$$(P(\lambda) + \Delta P(\lambda))\, x = 0. \tag{2.89}$$

Entonces, se tiene que

$$\|P(\lambda)x\|_2 \leq \|\Delta P(\lambda)\|_2 \|x\|_2 \leq \left(\sum_{i=0}^{d} |\, \Phi_i(\lambda)| \|A_i\|_2\right) \|x\|_2 \epsilon, \tag{2.90}$$

de donde se deduce de forma inmediata que $\alpha_P(x, \lambda) \leq \eta_P(x, \lambda)$. En segundo lugar, tomando las matrices perturbación

$$\Delta A_j := -\frac{1}{\sum_{i=0}^{d} |\, \Phi_i(\lambda)| \|A_i\|_2} \operatorname{sign} \Phi_j(\lambda) \|A_j\|_2 P(\lambda) x \frac{x^*}{\|x\|_2}, \tag{2.91}$$

donde

$$\operatorname{sign} \mu := \begin{cases} \bar{\mu}/|\mu| & \text{si } \mu \neq 0 \\ 0 & \text{si } \mu = 0, \end{cases} \tag{2.92}$$

se tiene, por un lado, que

$$\begin{aligned} &(P(\lambda) + \Delta P(\lambda))\, x = \\ &P(\lambda)x - \frac{1}{\sum_{i=0}^{d} |\, \Phi_i(\lambda)| \|A_i\|_2} \left(\sum_{j=0}^{d} \Phi_j(\lambda) \operatorname{sign} \Phi_j(\lambda) \|A_j\|_2\right) P(\lambda)x = 0, \end{aligned} \tag{2.93}$$

y por otro que $\|\Delta A_j\|_2 < \alpha_P(x, \lambda)\|A_j\|_2$, por lo que $\eta_P(x, \lambda) \leq \alpha_P(x, \lambda)$. □

En la expresión (2.88) se puede comprobar que existe una estrecha relación entre el error hacia atrás y el residuo asociado a un par propio aproximado. En la interfaz de la clase `PEP` de SLEPc se dispone de rutinas para el cálculo del residuo y del error hacia atrás de las soluciones calculadas. Para éste último se utiliza la expresión dada en (2.88), aunque utilizando la norma infinito (en lugar de la 2-norma), ya que ésta puede ser fácilmente calculada para matrices paralelas distribuidas por filas.

Tabla 2.1: Resultados de ejecución con el solver TOAR para problemas polinómicos de distinta dimensión (*dim*) y grado (*grd*). Se solicita el cálculo de *nev* autovalores situados en distintas partes de espectro (*which*), y se utilizan *ncv* columnas para la base de Krylov. En los resultados se incluye el error hacia atrás máximo η_P, el número de reinicios realizados (*its*) y el tiempo de ejecución (en segundos) con 16 procesos MPI. Para el criterio de parada se ha utilizado una tolerancia de 10^{-8}.

nombre	grd	dim	nev	which	ncv	η_P	its	tiempo
qd_cylinder	3	690,718	10	próx. a 0.1	40	2×10^{-10}	9	479
qd_pyramid	5	∼12 mill	5	próx. a 0.4	40	2×10^{-11}	1	1991
sleeper	2	1 mill	40	próx. a -0.9	80	5×10^{-12}	2	82
pdde_stability	2	250,000	4	$\|\lambda\|_2 = 1$	100	4×10^{-11}	154	582
planar_waveguide	4	50,001	2	menor real	60	6×10^{-7}	53	56
acoustic_wave_2d	2	999,000	10	próx. a 0	25	1×10^{-9}	4	65
butterfly	4	90,000	1	próx. a 0.1	100	3×10^{-9}	183	684
loaded_string	10	1 mill	7	próx. a 0	40	6×10^{-12}	1	44

2.6.3. Resultados numéricos

Los problemas de test utilizados para la evaluación de las implementaciones realizadas, y algunas características de éstos, se recogen en la Tabla 2.1. En dicha tabla, se muestran también resultados con los que se comprueba el buen comportamiento del solver TOAR en varias ejecuciones con distintos problemas. El primero de ellos surge en el cálculo de la estructura electrónica de puntos cuánticos vía discretización de la ecuación de Schrödinger [48], mientras que el resto pertenecen a la colección NLEVP [14]. Todos ellos se corresponden con problemas polinómicos excepto el último, denominado loaded_string, que es un problema racional que resolvemos mediante interpolación polinómica, y utiliza la expresión general de problemas polinómicos considerando en este caso la base de Chebyshev. Todos los cálculos se han realizado utilizando aritmética real, excepto los relativos al problema pdde_stability en el que las matrices coeficiente que lo definen son complejas. En general, se han utilizado los parámetros de ejecución por defecto, esto es, la variante de reinicio con bloqueo de los autovectores convergidos, sin ningún tipo de escalado, y con extracción en función de la norma, aunque para el problema planar_waveguide ha sido necesaria la utilización del escalado sobre el parámetro y la extracción estructurada, al igual que el problema damped_beam, que se ha ejecutado con escalado en el parámetro. También apuntamos que en el problema pdde_stability se ha utilizado un criterio de ordenación definido mediante la interfaz de usuario (ver Apendice A) para calcular los autovalores que satisfacen $|\lambda| = 1$. El criterio de convergencia en el método de Krylov–Schur se determina mediante la expresión del residuo (2.6), que se toma relativo al módulo del valor de Ritz, utilizando una tolerancia que se ha tomado igual a 10^{-8}.

Para la evaluación del comportamiento paralelo, consideramos en primer lugar

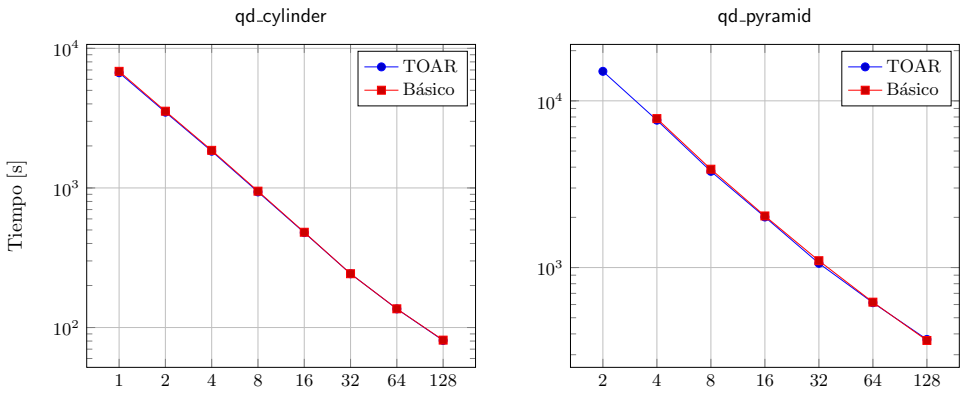

Figura 2.1: Tiempo de ejecución (en segundos) con los dos tests provenientes de la simulación de puntos cuánticos: qd_cylinder (izquierda) y qd_pyramid (derecha), para los métodos TOAR y Arnoldi Básico, utilizando hasta 128 procesos MPI y con parámetros de ejecución los mostrados en la Tabla 2.1.

los test provenientes de la simulación de puntos cuánticos, qd_cylinder y qd_pyramid. El primero de ellos se corresponde con un polinomio matricial de grado 3 y el segundo con uno de grado 5. Se ha utilizado una transformación de desplazamiento e inversión para calcular los valores más cercanos a $\sigma = 0{,}4$ en el caso de qd_cylinder, y $\sigma = 0{,}1$ para qd_pyramid. La resolución de los sistemas lineales, requeridos en la expansión del subespacio de Krylov (2.33), se lleva a cabo mediante el solver Bi-CGStab (proporcionado por PETSc), utilizando Jacobi a bloques (con descomposición ILU en los subdominios locales) como precondicionador. La Figura 2.1 muestra los tiempos de ejecución en paralelo para las variantes TOAR y Arnoldi Básico, para un número creciente de procesos MPI. Debido a los requerimientos de memoria, el problema de grado 5, qd_pyramid, necesita un mínimo de 2 procesos en TOAR y 4 en Arnoldi Básico (más adelante se evalúan las distintas necesidades de memoria de ambos métodos). En este caso no se observa ninguna diferencia apreciable entre las dos variantes, las cuales presentan un rendimiento bastante bueno, con escalabilidad casi lineal. En principio cabría esperar una reducción de tiempo para la variante TOAR frente a la de Arnoldi, debido a que en este último la dimensión de los vectores en la ortogonalización es d veces mayor a la de TOAR, siendo d el grado del polinomio. Sin embargo, el análisis de los tiempos de ejecución de las diferentes operaciones involucradas, revela que en el caso del problema qd_pyramid, el porcentaje de tiempo correspondiente a la resolución de los sistemas lineales (de igual dimensión en ambas variantes) está entre el 93 % y 96 % en el caso de TOAR y entre 90 % y 93 % en el de Arnoldi Básico (para el problema qd_cylinder son aún mayores), por lo que el tiempo de ortogonalización para ambas variantes es casi despreciable. Este hecho es el que determina que en la Figura 2.1 las gráficas relativas a TOAR y Arnoldi Básico prácticamente se solapen. La buena escalabilidad presente en estas ejecuciones está principalmente determinada por la de los solvers lineales, que en este caso tienen un

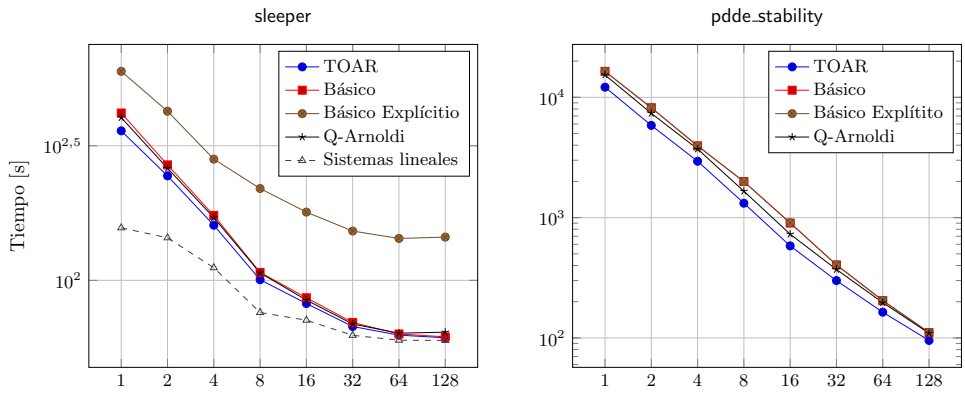

Figura 2.2: Tiempos de ejecución (en segundos) utilizando hasta 128 procesos MPI con los métodos TOAR, Q-Arnoldi y Arnoldi Básico, con las variantes implícita y explícita, en la resolución de dos problemas de NLEVP: sleeper (izquierda) y pdde_stability (derecha). Los parámetros de ejecución se muestran en la Tabla 2.1.

buen comportamiento.

En segundo lugar, en la Figura 2.2 se muestra el rendimiento de dos problemas con diferente comportamiento en cuanto a la escalabilidad que presentan. A la izquierda, se tiene una ejecución con el problema sleeper, en el que la escalabilidad final, al igual que ocurría en las ejecuciones asociadas a los problemas de la simulación de puntos cuánticos, viene determinada principalmente por la escalabilidad asociada a la resolución de los sistemas lineales, la cual no muestra, en este caso, un comportamiento tan bueno. En contraste, en la ejecución del problema pdde_stability, a la derecha, se observa una escalabilidad casi lineal, consecuencia de que en este caso el tiempo de ejecución dominante es el de la operación de ortogonalización. Los sistemas lineales en la ejecución de sleeper (izquierda), se han resuelto mediante un método directo, utilizando el solver paralelo proporcionado por la biblioteca MUMPS, para el que únicamente se han considerado los parámetros por defecto. El tiempo asociado a dichas resoluciones, para la ejecución con TOAR (incluido también en el gráfico), supone aproximadamente un 44 % del tiempo total si se utiliza un único proceso, pero crece hasta un 98 % al utilizar 128 procesos. El hecho de que, en general, los métodos directos no escalan bien a un número elevado de procesos, hace que el rendimiento, en este ejemplo, decaiga rápidamente al aumentar éstos hasta 128. Respecto a los casos de la Figura 2.1, la cantidad de trabajo asociada a la operación de ortogonalización es ahora más significativa (principalmente para pocos procesos), por lo que se puede apreciar que la ejecución con TOAR es algo más rápida que la relativa al método de Arnoldi Básico, mientras que Q-Arnoldi está entre medias de ambas. También se ha incluido la ejecución en la que las matrices del problema lineal se forman explícitamente, pudiendo así comprobar que esta opción es considerablemente más lenta debido a que las resoluciones de sistemas lineales se hacen, en este caso, con una matriz de dimensión el doble que en el resto. Por otro

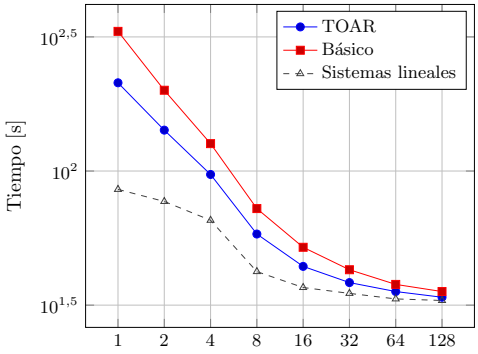

Figura 2.3: Tiempo de ejecución (en segundos) utilizando hasta 64 procesos MPI para los métodos TOAR y Arnoldi Básico con el problema loaded_string de la colección NLEVP, resueltos via interpolación utilizando un polinomio de grado 10, expresado en función de la base de Chebyshev. Los parámetros de ejecución se muestran en la Tabla 2.1. La línea discontinua muestra el tiemplo empleado en la resolución de los sistemas lineales (incluyendo la factorización), para el solver TOAR.

lado, en la ejecución correspondiente al problema pdde_stability (Figura 2.1 derecha) se da el caso de que el porcentaje de tiempo asociado a la resolución de sistemas lineales es despreciable frente al correspondiente a otras operaciones. En este ejemplo, la matriz con la que se resuelven los sistemas lineales es diagonal, por lo que dicha operación pasa a ser trivial comparado con el elevado coste asociado con la ortogonalización de una base de Krylov grande (ncv=100). Dado el buen rendimiento de la operación de ortogonalización, cuyo tiempo domina la ejecución, se puede apreciar una escalabilidad casi lineal en este caso. Observamos que también en estas ejecuciones, TOAR es más rápido que Q-Arnoldi y Arnoldi Básico. Además, dado que la matriz B es también diagonal, podemos ver, que, como es de esperar, coinciden los tiempos de Arnoldi Básico, con y sin formar explícitamente las matrices de la linealización.

Para finalizar la evaluación del rendimiento paralelo, se muestran resultados de un caso de test que utiliza una representación polinómica con una base distinta de la de monomios. El problema loaded_string de la colección NLEVP es un problema de valores propios racional cuya resolución ha sido abordada vía interpolación polinómica, utilizando la base de polinomios de Chebyshev para representar el polinomio interpolador, que se ha escogido de grado 10. El intervalo de interés para este problema es $[4, 400]$, donde podemos encontrar 6 autovalores. Se hace notar, que aunque el polinomio de interpolación ha sido generado de forma automática por SLEPc, utilizando el objeto `NEP`, tal como de explica en el capítulo 5, aquí se omiten dichos detalles, centrándonos exclusivamente en la evaluación de la resolución del problema polinómico resultante. Aunque las aproximaciones obtenidas con dicho problema cumplen con la tolerancia exigida, no sucederá lo mismo al evaluar el residuo respecto del problema original de esas mismas aproximaciones. La Figura 2.3

Tabla 2.2: Consumo de memoria (en megabytes) por proceso MPI, al ejecutar con 16 procesos, contando todos los objetos del programa, incluyendo la base de vectores de Krylov, las matrices y factorizaciones.

	qd_cylinder	qd_pyramid	sleeper	pdde_stability	loaded_string
TOAR	50	1275	100	39	168
Arnoldi Básico	97	2666	184	92	372

muestra los tiempos de ejecución para los solvers TOAR y Arnoldi básico. Dado que en este caso la ortogonalización de la base de vectores en la variante Arnoldi básico precisa aproximadamente 10 veces más operaciones que en TOAR, se hace evidente la ganancia de este último respecto a Arnoldi básico, tanto en rendimiento como en memoria (ver Tabla 2.2). En la Figura 2.3 también se incluye el tiemplo empleado en la resolución de los sistemas lineales con MUMPS (incluyendo la factorización), que está entre el 40 % y el 97 % del tiempo total de cómputo en TOAR, y entre el 26 % y el 93 % en Arnoldi básico.

En lo referente al consumo de memoria, en la Tabla 2.2 se comparan los requerimientos totales por proceso para TOAR y Arnoldi Básico en varias ejecuciones con distintos problemas. Tal como era de esperar, en todos los casos se observa un ahorro significativo de memoria en la variante de TOAR. Las cantidades que se muestran en la Tabla 2.2 incluyen tanto la memoria necesaria para almacenar la base de Krylov (explícita o implícitamente), además de otros elementos como las matrices del problema. Aún así, la diferencia entre ambas variantes se hace más evidente con polinomios de mayor grado, aunque esta diferencia depende del porcentaje de memoria que supone, sobre el total, la base de Krylov.

2.7. Conclusiones

Con el fin de dotar a SLEPc de solvers para la resolución de problemas polinómicos de valores propios, la primera línea de estudio seguida, abordada en este capítulo, ha sido la de plantear la resolución de un PEP recurriendo a una linealización del mismo. Con este enfoque se ha continuado con el que había ya en SLEPc, con anterioridad a esta tesis, para problemas cuadráticos. Se han considerado, para su incorporación en la biblioteca, diferentes variantes del método de Arnoldi, las cuales difieren en la forma de llevar a cabo los cálculos implicados, y en el uso más o menos eficiente a nivel de memoria, cuando dicho método se aplica sobre una linealización particular, con una determinada estructura. Esto ha dado lugar a varios solvers polinómicos en SLEPc, los que en este capítulo hemos denominado Q-Arnoldi, TOAR y Arnoldi Básico (con y sin formación explícita de la linealización), de los cuales sólo TOAR y Arnoldi Básico con linealización implícita se han extendido para grado mayor que dos.

El solver que implementa el método de TOAR polinómico se ha incorporado como

el solver por defecto en SLEPc para la resolución de PEPs, ya que ha demostrado ser el más eficiente en términos computacionales. Dicho solver ha sido equipado con distintas características deseables, como son la posibilidad de definir el PEP utilizando una base de polinomios distinta a la de monomios, o el soporte para transformaciones espectrales, aplicadas bien sobre el problema polinómico o bien sobre la linealización. Otras opciones incluidas en este solver son, en primer lugar, la posibilidad de escalar las matrices que definen el PEP, para mejorar, si es necesario, el condicionamiento de la linealización, o directamente el de algunos autovalores del problema polinómico, y en segundo lugar, la incorporación de varias estrategias de extracción de los autovectores del problema polinómico a partir de los de la linealización.

Aparte de abordar el estudio de las diferentes adaptaciones requeridas para la incorporación de los distintos elementos considerados en los solvers desarrollados, la principal contribución realizada en este capítulo ha sido la de conseguir, para la variante TOAR, una forma efectiva de bloqueo de los vectores que determinan un subespacio invariante.

Como resultado del trabajo expuesto en este capítulo, en SLEPc se dispone de una colección de solvers paralelos para la resolución del problema polinómico, con una interfaz de usuario fácil de usar (ver apéndice A), con parámetros por defecto adecuados, en general, para muchos tipos de problemas, aunque también con suficiente flexibilidad para poder afinar algunos de los parámetros de ejecución en problemas que así lo requieran. Con las implementaciones realizadas, se ha podido evaluar el comportamiento paralelo de los solvers incorporados, para los que se ha comprobado que presentan muy buena escalabilidad, siempre que se disponga de solvers y precondicionadores paralelos, para la resolución de sistemas lineales, que escalen bien.

Capítulo 3

Resolución de problemas cuadráticos simétricos

En este capítulo nos centramos en la resolución de un tipo particular de PEP, el problema cuadrático simétrico, el cual está definido por un polinomio de grado dos,

$$Q(\lambda)x = 0, \tag{3.1a}$$

$$Q(\lambda) = \lambda^2 M + \lambda C + K, \tag{3.1b}$$

en el que las matrices coeficiente que lo definen son reales simétricas o complejas hermíticas, $M^* = M$, $C^* = C$ y $K^* = K$. Este tipo de problemas tienen ciertas características especiales, como por ejemplo, que sus autovalores son reales, o aparecen en pares complejos conjugados (aún en el caso de que las matrices no sean reales) [89]. Un ejemplo de problema cuadrático simétrico, que cumple además que todos sus autovalores son reales, es el problema cuadrático hiperbólico, que se caracteriza por las propiedades de que M es definida positiva y

$$(x^*Cx)^2 > 4(x^*Mx)(x^*Kx), \quad \forall x \in \mathbb{C}^n \setminus \{0\}. \tag{3.2}$$

Los problemas cuadráticos hiperbólicos, y en general los simétricos, surgen por ejemplo, en aplicaciones de análisis de vibración de estructuras [70, 54].

Aunque la resolución de los problemas de valores propios cuadráticos (QEP, del inglés *quadratic eigenvalue problem*) simétricos puede ser abordada mediante cualesquiera de los solvers polinómicos descritos en el capítulo 2, nos planteamos el desarrollo de solvers específicos que puedan obtener ciertas ventajas numéricas en la resolución de dichos problemas, frente a los solvers genéricos que no tienen en cuenta las características particulares de los mismos. Con dicha especialización se puede, por ejemplo, mejorar la precisión o la convergencia de los autovalores que se calculan, así como asegurar que las soluciones que se obtienen cumplen con determinadas características que pueden ser relevantes para el problema físico subyacente. Este objetivo incide en uno de los objetivos determinados para esta tesis de ofre-

cer distintos tipos de solvers con los que sea posible adaptarse a las características particulares de las distintas aplicaciones.

Podemos encontrar numerosos trabajos que muestran la existencia de un interés general por métodos específicos que preserven la estructura del problema inicial a lo largo del proceso de resolución, tanto en problemas con estructura simétrica, como hamiltoniana o palindrómica [70, 67, 11, 61].

El enfoque escogido al abordar la resolución de los problemas cuadráticos simétricos está en la línea de los solvers descritos en el capítulo 2, y se basa en el esquema planteado en [70], donde se combina el uso de una linealización simétrica del QEP, junto con un método de Krylov, el método pseudo-Lanczos, que consigue mantener las características estructurales del problema durante su resolución, trasladando la condición de simetría al problema proyectado que genera.

Los solvers que se describen en este capítulo resuelven el problema cuadrático de valores propios $Q(\lambda)$ definido en (3.1), mediante la creación y resolución de la linealización $L(\lambda) = A - \lambda B$ definida en [70, 89], y que viene dada por las matrices

$$A = \begin{bmatrix} 0 & -K \\ -K & -C \end{bmatrix}, \quad B = \begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix}, \tag{3.3}$$

donde se asume que K es no singular (en otro caso se trabajaría con el polinomio reverso de Q, o se realizaría una transformación espectral). La linealización (3.3) consigue mantener la estructura simétrica del QEP, y cumple con propiedades espectrales presentes en el problema original, en cuanto a que sus autovalores son reales o aparecen en pares conjugados. Además, en el caso de que $Q(\lambda)$ sea hiperbólico, el haz de matrices $A - \lambda B$ es definido y, al igual que ocurre con el problema cuadrático, sus autovalores son todos reales [45]. Los autovectores de (3.1) pueden ser fácilmente obtenidos a partir de los de la linealización $L(\lambda)$, los cuales tienen la forma

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}, \tag{3.4}$$

siendo x un autovector del problema cuadrático.

Los problemas lineales generalizados en los que $A - \lambda B$ es un haz simétrico definido pueden ser resueltos por el método B-Lanczos, que consigue preservar la simetría a lo largo de la ejecución, utilizando un producto escalar no estándar (el definido por B suponiendo que es definida positiva), con el que construye una base B-ortogonal V, verificando $V^*BV = I$, del subespacio de Krylov, y un problema proyectado simétrico (con autovalores reales), a partir del cual obtiene aproximaciones a los pares propios. La linealización (3.3), aun siendo simétrica, no será en general, definida positiva, por lo que el método B-Lanczos no podrá ser utilizado. En su lugar, el método pseudo-Lanczos [70, 9, 56, 65] es una variante que, a costa de utilizar un pseudo-producto escalar (definido por B), para el que es posible encontrar vectores $w \neq 0$ de forma que $w^*Bw \leq 0$, consigue mantener la simetría del problema original, generando problemas proyectados que son haces simétricos indefinidos. Este método tiene el inconveniente de ser potencialmente inestable, debido al uso de un pseudo-producto escalar. Se puede encontrar información sobre la utilización de

pseudo-productos escalares en álgebra lineal en [32]. Otros autores también han considerado adaptar otros métodos (LOBPCG) al uso de una ortogonalización respecto de una matriz indefinida [52].

En este capítulo se describen los solvers específicos desarrollados para la resolución del problema (3.1), los cuales han sido basados en la utilización de la linealización (3.3) conjuntamente con el método pseudo-Lanczos. Nos proponemos incorporar en dicho método dos elementos importantes desde un punto de vista computacional. Éstos son, por un lado, incluir una técnica de reinicio eficaz con la que reducir el coste computacional y limitar los requerimientos de memoria necesarios, y por otro lado, derivar variantes que hagan un uso eficiente de la memoria al estilo de los métodos Q-Arnoldi y TOAR descritos en el capítulo 2. Dichas variantes han sido denominadas, respectivamente, Q-Lanczos y STOAR.

El capítulo se inicia dando algunas definiciones que se utilizarán a lo largo del mismo, y describiendo los métodos con los que se resolverán los problemas de valores propios de dimensión reducida asociados a haces de matrices simétricas indefinidas (problemas proyectados), que surgen al incorporar la técnica de reinicio, en el método pseudo-Lanczos, el cual se describe en la sección 3.2. En las secciones 3.3.1 y 3.3.2 se presentan los nuevos métodos Q-Lanczos y STOAR, respectivamente. En la sección 3.4.1 se tratan cuestiones relacionadas con la detección de inestabilidad, y en la 3.4.2 se muestran resultados de los experimentos numéricos realizados, además de los de evaluación del rendimiento paralelo. Para finalizar, se incluyen las conclusiones del trabajo presentado en el capítulo, y algunas notas sobre extensiones.

3.1. Matrices pseudo-simétricas

En esta sección se define el concepto de matriz pseudo-simétrica, así como otros conceptos relacionados con éste que serán utilizados a lo largo del capítulo. El estudio de las matrices pseudo-simétricas está motivado por el hecho de que el método pseudo-Lanczos, descrito en la sección 3.2, genera un problema proyectado que está definido por una matriz real pseudo-simétrica, de la cual se debe obtener una diagonalización a bloques con unas características determinadas. Estos requerimientos especiales en cuanto a la resolución del problema proyectado, vienen determinados por la técnica de reinicio que se describe en la sección 3.2.2. La biblioteca SLEPc encapsula en la clase auxiliar `DS` todo lo relevante al manejo y resolución de los problemas (proyectados) de dimensión reducida que se generan en los solvers de las clases principales (ver sección 1.4.2). Existe una subclase por cada tipo de problema proyectado a resolver, y por norma general, los métodos de resolución que se implementan en dichas subclases, utilizan las rutinas de cálculo de valores propios proporcionados por la biblioteca LAPACK. Sin embargo, y dado que no se ha encontrado en dicha biblioteca rutina alguna con la que calcular la diagonalización a bloques de una matriz pseudo-simétrica en la forma requerida, ha sido necesario el estudio de dichos problemas y la implementación de rutinas para su resolución dentro de la subclase `DSGHIEP` (relacionada con la resolución de problemas de valores propios generalizados simétricos y no definidos positivos). Los métodos estudiados

se describen en la sección 3.1.1.

Dada una matriz hermítica y definida positiva, $M \in \mathbb{C}^{n\times n}$, ésta define un producto escalar en $\mathbb{C}^n$, dado por

$$\langle x, y\rangle_M := y^* M x \quad \text{para} \quad x, y \in \mathbb{C}^n. \tag{3.5}$$

Cuando M no es definida positiva, dicha forma cuadrática no define ya un producto escalar. No obstante, en este caso diremos que la expresión (3.5) define un pseudo-producto escalar o un producto escalar indefinido. En este contexto, denominaremos *pseudo-norma* (definida por la matriz hermítica M), a la función

$$\|x\|_M := \operatorname{sgn}(\langle x, x\rangle_M)\sqrt{|\langle x, x\rangle_M|} \quad \text{para} \quad x \in \mathbb{C}^n, \tag{3.6}$$

donde $\operatorname{sgn}(\alpha)$ denota el signo de un número real α. La expresión (3.6) se corresponde con una norma sólo en el caso de que M sea definida positiva. En relación con estos conceptos, a continuación se detalla alguna nomenclatura específica de este capítulo:

- Se dice que dos vectores distintos x e y, son M-ortogonales si $\langle x, y\rangle_M = 0$, y que un vector $u \in \mathbb{C}^n$ está M-normalizado si $|\,\|u\|_M\,| = 1$.
- Diremos que las columnas de una matriz $V = [v_1, \ldots, v_k]$ son mutuamente M-ortogonales, si V^*MV es una matriz diagonal no singular. Si además dichas columnas están M-normalizadas, se dice que éstas son M-ortonormales. En este último caso, $\Omega := V^*MV$ es una signatura (matriz diagonal cuyos elementos diagonales son 1 ó -1). Cuando se quiera hacer referencia a dicha diagonal, expresaremos la ortonormalidad de las columnas de V diciendo que son (M, Ω)-ortogonales.

Definición 3.1. Dada una matriz hermítica M, decimos que una matriz S es M-simétrica si verifica $S^*M = MS$.

En el caso de que M sea definida positiva, el hecho de que una matriz S sea M-simétrica es equivalente a que S sea autoadjunta con respecto al producto escalar definido por M. Por otro lado, cuando la matriz M de la Definición 3.1 es una signatura, la condición de que una matriz S sea M-simétrica es equivalente a que S sea el producto de una matriz simétrica por una signatura, ya que si $S^*\Omega = \Omega S$, se tiene $S = \Omega S^* \Omega$ con $S^*\Omega$ simétrica, y si $S = \Omega T$ con T simétrica, entonces S verifica la definición 3.1. Esta última consideración caracteriza las matrices que son simétricas con respecto de una signatura y motiva la siguiente definición.

Definición 3.2. Una matriz S se dice que es pseudo-simétrica si puede expresarse en la forma $S = \Omega T$, donde Ω es una signatura y T es simétrica real o hermítica compleja.

Aunque para una signatura se verifica $\Omega^{-1} = \Omega$, a lo largo de este capítulo, y por conveniencia notacional, utilizaremos expresiones del tipo $S = \Omega^{-1}T$, que serán igualmente válidas aun en el caso de que Ω sea una diagonal, y no necesariamente una signatura.

3.1.1. Diagonalización de una matriz pseudo-simétrica

Dada una matriz pseudo-simétrica, $\Omega^{-1}T \in \mathbb{R}^{n\times n}$, producto de una signatura Ω^{-1} y una matriz simétrica T, nos planteamos la forma de obtener una matriz semejante que sea real, pseudo-simétrica y diagonal a bloques, con tamaño máximo de bloque igual a 2. Resolver un problema de valores propios con la matriz $\Omega^{-1}T$ es equivalente a resolver un problema generalizado dado por el par simétrico (T, Ω), que en general será indefinido. Por ello se plantea, de forma equivalente, la obtención de matrices Q, $\tilde{\Omega}$ y $D \in \mathbb{R}^{n\times n}$ tales que D es diagonal a bloques con tamaño máximo de bloque igual a 2 y $\tilde{\Omega}$ es una signatura, verificando

$$Q^*\Omega Q = \tilde{\Omega}, \tag{3.7a}$$
$$Q^*TQ = D. \tag{3.7b}$$

Las relaciones (3.7) implican que Q es $(\Omega, \tilde{\Omega})$-ortogonal y D es simétrica, y que

$$\Omega^{-1}TQ = Q\tilde{\Omega}^{-1}D, \tag{3.8}$$

es decir, $\tilde{\Omega}^{-1}D$ es real, diagonal a bloques, pseudo-simétrica y semejante a $\Omega^{-1}T$. Diremos que una descomposición del tipo (3.8) es una diagonalización real pseudo-simétrica de $\Omega^{-1}T$.

Para el cálculo de las matrices Q, $\tilde{\Omega}$ y D que aparecen en (3.7), se han considerado dos enfoques. El primero de ellos se basa en un método similar al iterativo QZ, que trabaja manteniendo la estructura simétrica del par inicial (T, Ω) en los problemas equivalentes que genera. El segundo enfoque considerado utiliza, en un primer paso, métodos genéricos para la resolución de problemas generalizados no simétricos y recupera la estructura simétrica requerida en un paso adicional.

Diagonalización preservando la estructura pseudo-simétrica. El primer enfoque considerado se basa en el método descrito en [92]. Éste, en primer lugar reduce la matriz pseudo-simétrica inicial $\Omega^{-1}T$ a forma tridiagonal (manteniendo pseudo-simetría), y después aplica la iteración HR sobre el problema resultante, para generar las matrices Q, $\tilde{\Omega}$ y D, que satisfacen (3.7)-(3.8). Este método consigue mantener la estructura a lo largo de su ejecución, utilizando transformaciones que son ortogonales con respecto a determinadas signaturas.

El proceso de tridiagonalización se lleva a cabo haciendo ceros en la matriz inicial, de forma similar al caso simétrico, aunque para mantener la pseudo-simetría en este caso los reflectores de Householder se combinan con rotaciones hiperbólicas

$$H_r = \begin{bmatrix} c & s \\ s & c \end{bmatrix}, \qquad c^2 - s^2 = \pm 1. \tag{3.9}$$

Estas transformaciones no son unitarias y pueden tener un número de condición arbitrariamente grande, por lo que su utilización puede hacer que los errores de redondeo generados durante el cálculo sean amplificados, produciendo pérdida de exactitud en la solución. Por otro lado, dado un vector x, no siempre es posible

construir una rotación hiperbólica, H_r, tal que $H_r x = \alpha e_1$, por lo que el proceso de tridiagonalización para matrices pseudo-simétricas podría fallar en algún momento. Las consideraciones anteriores dan una idea de que, a diferencia del caso simétrico, la reducción a forma tridiagonal para el caso pseudo-simétrico puede ser computacionalmente problemática y debe realizarse de forma cuidadosa. La implementación realizada en esta tesis e incluida en SLEPc se basa en el método propuesto por Tisseur [88], en el que se utiliza una combinación de transformaciones ortogonales e hiperbólicas, intentando reducir al máximo estas últimas, y controlando su número de condición. En dicha implementación se ha tenido en cuenta la estructura particular de las matrices donde habitualmente se aplica, que son las generadas por el proceso de Lanczos con reinicio (sección 3.2.2), que tienen la forma

$$T = \begin{bmatrix} D_1 & & & \Gamma_1 & & \\ & \ddots & & \vdots & & \\ & & D_m & \Gamma_m & & \\ \Gamma_1^T & \cdots & \Gamma_m^T & \alpha_{p+1} & \ddots & \\ & & & \ddots & \ddots & \beta_{k-1} \\ & & & & \beta_{k-1} & \alpha_k \end{bmatrix}, \tag{3.10}$$

donde D_i, $i = 1, \ldots, m$, son bloques 1×1 o 2×2, y Γ_i son vectores de dimensiones consistentes con D_i. Dado que la submatriz diagonal inferior de (3.10) está ya en forma tridiagonal, para minimizar la cantidad de transformaciones hiperbólicas requeridas, además de la implementación del método [88], se ha incluido una variante que trabaja únicamente en la parte con forma de flecha (submatriz superior izquierda) de (3.10).

Tras la reducción a forma tridiagonal pseudo-simétrica, en un segundo paso se utiliza el método HR descrito en [92], el cual es una iteración de tipo GR (similar al iterativo QR) que mantiene la pseudo-simetría a lo largo de la iteración. Este método utiliza rotaciones hiperbólicas (3.9) en la fase de *bulge-chasing* por lo que, al igual que el proceso de tridiagonalización descrito, presenta riesgos de inestabilidad numérica, y puede fallar siempre que no sea posible encontrar una rotación hiperbólica adecuada para hacer ceros en un determinado vector. Dado que no existen implementaciones del método HR disponibles en bibliotecas como LAPACK, entre los métodos implementados para la subclase `DSGHIEP`, se ha incluido una implementación del método HR basada en [92].

Diagonalización alternativa recuperando pseudo-simetría. Para la obtención de las matrices Q, $\tilde{\Omega}$ y D de (3.7)-(3.8), en este segundo enfoque se considera un procedimiento alternativo que parte de la descomposición en valores propios de $\Omega^{-1}T$, calculada, por ejemplo, utilizando el iterativo QR disponible en LAPACK. Tomando como punto de partida dicha descomposición $\Omega^{-1}TY = Y\Theta$, y representando de forma separada la parte real e imaginaria de los vectores propios, en un segundo paso se obtiene una descomposición real equivalente, $TY_r = \Omega Y_r \Theta_r$, en la cual Θ_r es diagonal a bloques. Para recuperar la estructura pseudo-simétrica, los

vectores reales Y_r son Ω-ortonormalizados, para lo que se calcula una descomposición HR de la matriz Y_r, que tiene la forma $Y_r = QR$, con R triangular superior y tal que $Q^*\Omega Q = \tilde{\Omega}$. Finalmente, se obtienen los bloques diagonales de D, a partir de la expresión $D = Q^*TQ = \tilde{\Omega}R\Theta_r R^{-1}$. Se hace notar que en la implementación realizada no se calcula R^{-1} explícitamente, sino que se obtienen los bloques diagonales de D y se ponen a cero el resto de elementos (que podrían no ser cero en precisión finita).

Una propiedad general de los pares hermíticos indefinidos (A, B), es que si B es no singular y $B^{-1}A$ es diagonalizable, entonces existe un conjunto completo de autovectores, W, tales que W^TBW es una signatura [9]. Como consecuencia de dicha propiedad, se tiene que en la matriz Y_r obtenida en el segundo paso del proceso descrito, las columnas asociadas con autovalores reales simples de $\Omega^{-1}T$ forman, de por sí, un conjunto de vectores Ω-ortogonales. Aunque este hecho hace que, en principio, sólo sea necesario realizar la Ω-ortogonalización entre pares de columnas de Y_r asociadas con un autovalor complejo (partes real e imaginaria), o entre columnas asociadas con un autovalor múltiple, se ha optado por calcular una descomposición HR de la matriz Y_r completa, con el fin de mejorar el nivel de Ω-ortogonalidad del resultado.

La descomposición HR puede realizarse mediante el proceso de Gram-Schmidt con el pseudo-producto escalar determinado por Ω (ver por ejemplo [74]) o, alternativamente, trabajando con una combinación de reflectores de Householder y rotaciones hiperbólicas, de forma similar al proceso de tridiagonalización descrito en [88]. Esta última opción es la que se ha utilizado en la implementación realizada, ya que ésta proporciona información sobre el número de condición de las transformaciones que se aplican, dato que puede utilizarse para reducir el riesgo de inestabilidad global en los siguientes casos:

- Al realizar la ortogonalización de los vectores correspondientes a la parte real e imaginaria de un autovector complejo, el orden en que ésta se lleva a cabo puede influir considerablemente en la exactitud del resultado. Por ello, y con el fin de minimizar el error que se introduce en dicho proceso, el orden en que se realiza la ortogonalización se determina evaluando el número de condición de las transformaciones requeridas en cada caso.

- Tal como se describe en la sección 3.2.2, al reiniciar el método de Lanczos, la relación de Krylov que éste genera se trunca y sólo una parte de la descomposición del problema proyectado calculada se mantiene. Es decir, si se han calculado matrices correspondientes a (3.7)

$$D = \begin{matrix} p \\ q \end{matrix} \begin{bmatrix} \overset{p}{D_p} & \overset{q}{} \\ & D_q \end{bmatrix}, \quad Q = \begin{bmatrix} Q_p & Q_q \end{bmatrix}, \text{ y } \quad \tilde{\Omega} = \begin{bmatrix} \tilde{\Omega}_p & \\ & \tilde{\Omega}_q \end{bmatrix}, \tag{3.11}$$

  de dimensión p+q, en las que $\tilde{\Omega}_p^{-1}D_p$ contiene los autovalores más deseados de $\Omega^{-1}T$, sólo las matrices D_p, Q_p y $\tilde{\Omega}_p$ se utilizarán para actualizar la relación de Krylov. Por tanto, cuando se calcula la descomposición HR, se realiza una

permutación de columnas con el fin de mover al grupo Q_q, que será eliminado, aquellas columnas cuya ortogonalización pudiera producir un fallo en la ejecución o conllevar la utilización de una transformación con un elevado número de condición. En éste último caso, las columnas incluidas en Q_q no necesitan ser ortogonalizadas, y las matrices $\tilde{\Omega}$ y D deben ser permutadas convenientemente.

3.2. Pseudo-Lanczos con reinicio en problemas generalizados simétricos indefinidos

El método de Lanczos, en la variante descrita en [29], es un método de Krylov específico para la resolución de problemas generalizados simétricos,

$$Ax = \lambda Bx, \tag{3.12}$$

en los que el par (A, B) es real simétrico (o complejo hermítico) y definido positivo. Esta variante utiliza un producto escalar no estándar, definido por B (o una combinación lineal de A y B adecuada), respecto del cual es simétrica la matriz S que se obtiene al reducir el problema inicial a forma estándar. Dicha matriz S viene dada por uno de los operadores

$$\begin{aligned} &S = B^{-1}A, \\ &S = (A - \sigma B)^{-1}B, \quad \sigma \in \mathbb{R}. \end{aligned} \tag{3.13}$$

Siguiendo un proceso similar al descrito en el Algoritmo 2.1, utilizando un producto escalar no estándar, el método de Lanczos genera una relación similar a la del método de Arnoldi (2.5),

$$SV_k = V_k T_k + \beta_k v_{k+1} e_k^*, \tag{3.14}$$

en la que la base del subespacio de Krylov V_k es B-ortonormal y la matriz del problema proyectado que genera,

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-2} & \alpha_{k-1} & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{bmatrix}, \tag{3.15}$$

mantiene la simetría del problema inicial. Dicha propiedad unida a la forma Hessenberg que produce el método de Arnoldi (y el de Lanczos), produce la estructura de tridiagonal simétrica típica del problema proyectado en el método de Lanczos.

El hecho de preservar la estructura simétrica del problema, también en el problema proyectado, ofrece ciertas ventajas a la hora de obtener aproximaciones que verifiquen propiedades esperables para los pares propios del problema a resolver. Por ejemplo, al utilizar el procedimiento de Rayleigh-Ritz con la matriz simétrica (3.15),

los valores de Ritz que se obtienen son siempre reales. Además, la multiplicación de la base de Lanczos V_k por los autovectores (ortogonales) de T_k, produce vectores de Ritz B-ortogonales. Es decir, los pares de Ritz que se obtienen a partir de la relación de Lanczos (3.14), cumplen con propiedades que están presentes en la solución del problema original.

El método pseudo-Lanczos [70] es una variante del de Lanczos para la resolución de problemas generalizados simétricos (3.12), para los que podría no existir una combinación de A y B definida positiva (problemas simétricos no definidos positivos). Con el fin de preservar la simetría en el problema proyectado que genera, esta variante utiliza el producto indefinido determinado por B. En la sección 3.2.1 se describe el método pseudo-Lanczos relacionándolo con la variante no simétrica del método de Lanczos [25] de la cual deriva. Tal como ya se ha comentado en otros capítulos de esta tesis, los métodos que generan subespacios de búsqueda requieren versiones con reinicio que permitan acotar los requerimientos totales de memoria. Con el fin de disponer, en la implementación realizada, de una versión con reinicio del método pseudo-Lanczos, han sido necesarias algunas modificaciones de las cuales se dan detalles en el apartado 3.2.2.

3.2.1. Método pseudo-Lanczos

La variante no simétrica del método Lanczos se corresponde con un método de proyección oblicua que permite el cálculo tanto de vectores propios derechos como izquierdos en problemas no simétricos. Partiendo de dos vectores iniciales, verificando $v_1^* w_1 = 1$, y utilizando un proceso de Gram-Schmidt de dos lados en las recurrencias

$$p_k = Sv_k - V_k W_k^* S v_k = Sv_k - \hat{\gamma}_{k-1} v_{k-1} - \hat{\alpha}_k v_k, \tag{3.16a}$$

$$q_k = S^* w_k - W_k V_k^* S^* w_k = S^* w_k - \bar{\hat{\beta}}_{k-1} w_{k-1} - \bar{\hat{\alpha}}_k w_k, \tag{3.16b}$$

donde $p_k = \hat{\beta}_k v_{k+1}$, $q_k = \bar{\hat{\gamma}}_k w_{k+1}$, $q_k^* p_k = \hat{\gamma}_k \hat{\beta}_k$ y $\hat{\alpha}_k = w_k^* S v_k$, el método de Lanczos no simétrico [25] genera bases V_k y W_k biortogonales ($V_k^* W_k = I$) para los subespacios de Krylov $\mathcal{K}_k(S, v_1)$ y $\mathcal{K}_k(S^*, w_1)$, respectivamente, así como la matriz de proyección (de S sobre V_k a lo largo de W_k),

$$\hat{T}_k := W_k^* S V_k = \begin{bmatrix} \hat{\alpha}_1 & \hat{\gamma}_1 & & & \\ \hat{\beta}_1 & \hat{\alpha}_2 & \hat{\gamma}_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \hat{\beta}_{k-2} & \hat{\alpha}_{k-1} & \hat{\gamma}_{k-1} \\ & & & \hat{\beta}_{k-1} & \hat{\alpha}_k \end{bmatrix}, \tag{3.17}$$

verificando las relaciones

$$SV_k = V_k \hat{T}_k + \hat{\beta}_k v_{k+1} e_k^*, \tag{3.18a}$$

$$S^* W_k = W_k \hat{T}_k^* + \bar{\hat{\gamma}}_k w_{k+1} e_k^*. \tag{3.18b}$$

Imponiendo la condición de Petrov–Galerkin en la técnica de proyección (1.19), las relaciones (3.18) proporcionan tripletes (θ, x, y) que aproximan conjuntos de valor propio y los autovectores derecho e izquierdo correspondientes.

El método pseudo-Lanczos [70] es una forma particular de Lanczos no simétrico que se obtiene cuando la matriz del problema S es simétrica con respecto al producto escalar (indefinido o no) determinado por una matriz B simétrica (o hermítica). Esto ocurre, por ejemplo, en la resolución de un problema generalizado simétrico (3.12), en el que la matriz S dada en (3.13) es simétrica respecto al producto escalar definido por B. En este caso, partiendo de vectores iniciales v_1 y $w_1 := Bv_1(v_1^*Bv_1)^{-1}$, las recurrencias (3.16) producen bases V_k y W_k, relacionadas según $W_k = BV_k\Omega_k^{-1}$, siendo $\Omega_k := V_k^*BV_k$. Dicha relación hace que esta variante no requiera el cálculo explícito de los vectores w_k, por lo que trabaja únicamente con la recurrencia (3.16a), y genera la relación (3.18a), a partir de la cual obtiene aproximaciones a los autovectores derechos.

Definiendo $T_k := \Omega_k\hat{T}_k = V_k^*BSV_k$ y $\beta_k = \omega_{k+1}\hat{\beta}_{k+1}$, la relación (3.16a) que genera el método pseudo-Lanczos toma la forma

$$SV_k = V_k\Omega_k^{-1}T_k + \beta_k\omega_{k+1}^{-1}v_{k+1}e_k^*. \tag{3.19}$$

La biortogonalidad de las bases V_k y W_k de (3.18), se traduce, para este caso, en la B-ortogonalidad de los vectores V_k de (3.19), que verifican $V_k^*BV_k = \Omega_k$ y $V_k^*Bv_{k+1} = 0$. Por otro lado, la matriz $\Omega_k^{-1}T_k$ del problema proyectado en (3.19), es equivalente al problema generalizado (T_k, Ω_k), que es real (aún en el caso de que A y B en (3.12) sean hermíticas complejas) y simétrico (en general no definido positivo). Como consecuencia, se tiene que el método pseudo-Lanczos mantiene la condición de simetría en el problema proyectado que genera, obteniendo con ello ciertas ventajas, al igual que ocurría en el método de Lanczos, a la hora de mantener propiedades deseables en las aproximaciones que genera.

El Algoritmo 3.1 muestra, en la forma implementada en SLEPc, los pasos que sigue el método pseudo-Lanczos para el cálculo de la base de vectores de Lanczos V_k y las matrices que forman el problema proyectado $\hat{T}_k = \Omega_k^{-1}T_k$,

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix} \quad \text{y} \quad \Omega_k = \begin{bmatrix} \omega_1 & & & \\ & \omega_2 & & \\ & & \ddots & \\ & & & \omega_k \end{bmatrix}. \tag{3.20}$$

En el paso 8 del Algoritmo 3.1, podemos ver que los vectores de Lanczos son B-normalizados, por lo que Ω_k es una signatura (y la matriz $\Omega_k^{-1}T_k$ es pseudo-simétrica). Sin embargo, mantenemos la notación Ω_k^{-1}, ya que es posible utilizar otras normalizaciones de forma que Ω_k sea diagonal, pero no una signatura. En este caso, la tridiagonal $\hat{T}_k$ podría no ser pseudo-simétrica, aunque el problema proyectado (Ω_k, T_k) seguiría siendo un par simétrico (indefinido).

En el Algoritmo 3.1 podemos ver que existen algunas diferencias con respecto a la recurrencia corta (3.16a). Dichos cambios están motivados por diversas cuestiones

Algoritmo 3.1 Pseudo-Lanczos

Entrada: Matrices hermíticas $A, B \in \mathbb{C}^{n\times n}$ que definen S como en (3.13), vector inicial $v_1 \in \mathbb{C}^n$

Salida: Matrices $T_k, \Omega_k \in \mathbb{C}^{k\times k}$, $V_{k+1} \in \mathbb{C}^{n\times(k+1)}$ y $\beta_k, \omega_{k+1} \in \mathbb{R}$ que satisfacen (3.19)

1: $V_1 \leftarrow v_1 = v_1/\|v_1\|_B$
2: $\Omega_1 \leftarrow \omega_1 = v_1^* B v_1$
3: **para** $j = 1, 2, \ldots, k$ **hacer**
4: $\quad u = Sv_j$
5: $\quad h = V_j^* B u$
6: $\quad u = u - V_j \Omega_j^{-1} h$
7: $\quad \alpha_j = e_j^* h$
8: $\quad v_{j+1} = u/\|u\|_B$
9: $\quad \omega_{j+1} = v_{j+1}^* B v_{j+1}$
10: $\quad \beta_j = \omega_{j+1}\|u\|_B$
11: $\quad V_{j+1} \leftarrow \left[\begin{smallmatrix} V_j & v_{j+1} \end{smallmatrix}\right], \quad \Omega_{j+1} \leftarrow \left[\begin{smallmatrix} \Omega_j & \\ & \omega_{j+1} \end{smallmatrix}\right]$
12: **fin para**

relacionadas con su ejecución en aritmética de precisión finita y, en general, dichos cambios están presentes también en la implementación del método de Lanczos que existe en SLEPc. En primer lugar, para mantener la B-ortogonalidad de los vectores de Lanczos tras generar un nuevo vector del subespacio de Krylov, se opta por realizar su ortogonalización contra todos los vectores de Lanczos previamente calculados (líneas 5 y 6 del Algoritmo 3.1), por lo que todos ellos son almacenados (línea 11). Para más información sobre pérdida de ortogonalidad en Lanczos ver [69]. En segundo lugar, por razones de eficiencia en la ejecución en paralelo, para la ortogonalización (líneas 5 y 6) se utiliza la variante clásica del procedimiento de Gram-Schmidt, aunque para obtener una exactitud similar a la que se obtendría con el método de Gram-Schmidt modificado, este paso debe repetirse dos veces (para más detalles sobre implementación paralela de Gram-Schmidt iterado ver [38]). Por último, al realizar ortogonalización completa, debido a la precisión finita, se obtienen coeficientes de ortogonalización no nulos, más allá de los dos últimos vectores de Lanczos calculados. Sin embargo, al formar la matriz T_k, se descartan todos los coeficientes de Gram-Schmidt obtenidos, excepto los elementos de la diagonal α_j (los β_j se determinan mediante la normalización). De forma similar, sólo se almacenan los elementos de la diagonal de Ω_k, aunque $V_k^* B V_k$ no sea exactamente diagonal (los valores ω_j se redondean a ± 1).

Una vez obtenida una relación de pseudo-Lanczos (3.19), se utiliza un procedimiento similar al de Rayleigh-Ritz (técnica de proyección) para obtener aproximaciones, $(\tilde{x}_i, \tilde{\theta}_i) = (V_k y_i, \tilde{\theta}_i)$, de los pares propios buscados. Los autovectores del problema proyectado $\Omega_k^{-1} T_k$ proporcionan aproximaciones en $\text{span}(V_k)$, verificando $S\tilde{x}_i - \tilde{\theta}_i \tilde{x}_i \perp \text{span}(W_k)$ (condición de Petrov-Galerkin). Multiplicando la descomposición (3.19) por y_i se obtiene el residuo para cada vector derecho de Petrov

calculado,

$$r_i = SV_k y_i - \tilde{\theta}_i V_k y_i = \beta_k \omega_{k+1}^{-1} v_{k+1} e_k^* y_i. \tag{3.21}$$

En la implementación realizada se considera que una aproximación está convergida si la norma del residuo (3.21) es menor que una determinada tolerancia $\texttt{tol}_{\text{conv}}$,

$$\|r_i\| = |\beta_k \omega_{k+1}^{-1} e_k^* y_i| \|v_{k+1}\| < \texttt{tol}_{\text{conv}}. \tag{3.22}$$

Los métodos de Lanczos (simétricos o no) pueden sufrir una terminación prematura afortunada si en el proceso de expansión del subespacio de Krylov se obtiene una base de un subespacio invariante (esto es $Sv_k \in \text{span}(V_k)$ o $S^* w_k \in \text{span}(W_k)$ para algún k). En este caso, las aproximaciones obtenidas mediante el proceso de Rayleigh-Ritz se corresponden con pares propios exactos. Por otro lado, en el método de Lanczos no simétrico, la utilización de bases biortogonales puede provocar una terminación también prematura si las recurrencias (3.16) producen vectores no nulos, p_k y q_k que verifiquen $q_k^* p_k = 0$. En este caso, la terminación prematura se dice severa ya que no es posible continuar con el proceso de expansión de los subespacios de Krylov. Existen en la bibliografía diversos trabajos que abordan el problema de la terminación prematura severa en el método de Lanczos, como son por ejemplo las variantes de *look-ahead* [71], bloque adaptativo [7] o reinicio implícito [26]. La forma escogida en nuestra implementación para tratar el caso de terminación prematura severa está en la línea del último trabajo citado.

En el contexto del método pseudo-Lanczos, se produce una terminación prematura cuando se genera un vector u con B-norma igual a cero. Dicha terminación prematura es severa si el vector u generado no es idénticamente nulo. Además del caso (poco probable) de que se produzca una terminación prematura severa exacta, es necesario también considerar los casos en los que se obtiene un vector $u \neq 0$ con $u^* B u \approx 0$, ya que estas situaciones pueden hacer que se introduzca inestabilidad en el proceso. Para detectar estos casos, en la implementación realizada se comprueba, a cada iteración del Algoritmo 3.1, la posible ortogonalidad entre los vectores u (calculado en la línea 6) y Bu. Se considera que se produce una terminación prematura (no necesariamente exacta) si

$$\cos \angle(u, Bu) = \frac{|u^* B u|}{\|u\| \|Bu\|} < \texttt{tol}_{\text{break}}, \tag{3.23}$$

donde $\texttt{tol}_{\text{break}}$ representa una determinada tolerancia. Para poder continuar con la ejecución en el caso de que se presente una terminación prematura severa, una posibilidad es descartar la base calculada hasta el momento y empezar con un nuevo vector de inicio. Nosotros hemos optado por realizar un reinicio implícito tal como se propone en [26], si bien el tipo de reinicio en nuestro caso difiere del utilizado en dicho trabajo. Se ha podido comprobar el correcto funcionamiento de esta estrategia únicamente con casos sintéticos, ya que en las ejecuciones prácticas realizadas nunca se ha detectado una terminación prematura severa. La forma en que se realiza el reinicio del método pseudo-Lanczos en nuestra implementación se explica en el siguiente apartado.

3.2.2. Reinicio en pseudo-Lanczos

El reinicio del método de Lanczos se hace necesario, al igual que en el método de Arnoldi (ver sección 2.3), para evitar los inconvenientes que puede presentar un crecimiento excesivo de la base del subespacio de Krylov, tanto por los requerimientos de memoria, como por el coste de la ortogonalización completa de cada nuevo vector. En el caso del método pseudo-Lanczos, nosotros optamos por adaptar la técnica de reinicio grueso, desarrollada en [93] para problemas simétricos, que es un caso particular de la técnica de reinicio implícito incluida en el algoritmo de Krylov-Schur. Tal como se menciona en la sección 2.3, este último método utiliza relaciones de Krylov, en la forma,

$$SU_k = U_k C_k + u_{k+1} b^*, \tag{3.24}$$

que generalizan las de Arnoldi (o Lanczos). Para el desarrollo de la técnica de reinicio en pseudo-Lanczos, resulta útil la definición de un tipo especial de relaciones de Krylov a las que llamamos simétricas. Diremos que una relación de Krylov (3.24) es simétrica si las matrices S y C_k son, a su vez, simétricas con respecto a los productos definidos, respectivamente, por una matriz hermítica B, y por $\Omega_k := U_k^* B U_k$. Para este tipo de relaciones existe otra relación de Krylov implícita, asociada a la base $W_k := BU_k\Omega_k^{-1}$. Para derivar dicha relación, es suficiente con multiplicar (3.24) a la izquierda por B y a la derecha por Ω_k^{-1} y así, debido a la simetría de S y C_k, se obtiene

$$S^* W_k = W_k C_k^* + w_{k+1} d^*, \tag{3.25}$$

donde $w_{k+1} = Bu_{k+1}$ y $d = \Omega_k^{-1} b$. Como consecuencia se tiene que las columnas de U_k y W_k generan subespacios de Krylov para S y S^*, respectivamente.

Para efectuar el reinicio en el método pseudo-Lanczos consideramos relaciones de Krylov simétricas en las cuales U_k tiene columnas B-ortogonales y el cociente de Rayleigh C_k es real y tiene la forma $C_k = \Omega_k^{-1} T_k$, para T_k simétrica y $\Omega_k := U_k^* B U_k$,

$$SU_k = U_k \Omega_k^{-1} T_k + u_{k+1} b^*. \tag{3.26}$$

Nos referiremos a este tipo de relaciones de Krylov como pseudo-ortogonales. Un ejemplo de estas últimas es la relación (3.19), generada por el Algoritmo 3.1.

Otro concepto que utilizaremos en el contexto del reinicio en pseudo-Lanczos, es el de relaciones equivalentes. Al multiplicar por la derecha una relación de Krylov pseudo-ortogonal (3.26) por una matriz Ω_k-ortogonal, Q_k, obtenemos otra relación de Krylov que sigue siendo pseudo-ortogonal,

$$S\tilde{U}_k = \tilde{U}_k \tilde{\Omega}_k^{-1} \tilde{T}_k + u_{k+1} \tilde{b}^*, \tag{3.27}$$

donde $\tilde{U}_k = U_k Q_k$, $\tilde{\Omega}_k = Q_k^* \Omega_k Q_k$, $\tilde{T}_k = Q_k^* T_k Q_k$, y $\tilde{b} = Q_k^* b$. En esta situación diremos que ambas relaciones (3.26) y (3.27) son equivalentes.

La estrategia para reiniciar una relación de pseudo-Lanczos (3.26) es la de calcular otra equivalente tal que $\tilde{\Omega}_k^{-1}\tilde{T}_k$ tenga una forma diagonal a bloques

$$\tilde{\Omega}_k^{-1}\tilde{T}_k = \begin{array}{c} {\scriptstyle p} \\ {\scriptstyle k-p} \end{array} \begin{array}{c} \begin{array}{cc} {\scriptstyle p} & {\scriptstyle k-p} \end{array} \\ \begin{bmatrix} D_{11} & \\ & D_{22} \end{bmatrix} \end{array}. \tag{3.28}$$

De esta forma, es posible truncar la descomposición generando otra que sigue siendo pseudo-ortogonal y tiene por matriz proyectada la correspondiente al bloque diagonal principal D_{11}. La descomposición truncada, de orden p, puede entonces extenderse hasta orden k utilizando el Algoritmo 3.1.

A continuación se resumen los pasos incluidos en el método pseudo-Lanczos para llevar a cabo el reinicio grueso. Partiendo de una relación del tipo (3.19), generada mediante el Algoritmo 3.1, se calculan, en primer lugar, las matrices Q_k, $\tilde{\Omega}_k$ y D_k, asociadas a una diagonalización pseudo-simétrica (3.8) del par indefinido (T_k, Ω_k), de forma que los p valores de Ritz más deseados (o los bloques asociados) estén en la submatriz correspondiente al bloque diagonal principal de D_k (de orden p). Entonces, postmultiplicando la descomposición (3.19) por $Q_k = [Q_p, Q_{p+1:k}]$, se obtiene la relación equivalente

$$SV_kQ_k = V_kQ_k(Q_k^*\Omega_kQ_k)^{-1}Q_k^*T_kQ_k + \beta_k\omega_{k+1}^{-1}v_{k+1}e_k^*Q_k, \tag{3.29}$$

que puede ser truncada (para cada p que no rompa un bloque 2×2) dando lugar a

$$S\tilde{V}_p = \tilde{V}_p\tilde{\Omega}_p^{-1}D_p + v_{k+1}b^*, \tag{3.30}$$

donde $b = \beta_k\omega_{k+1}^{-1}Q_p^*e_k$ y $\tilde{V}_p = V_kQ_p$. Esta nueva ecuación satisface la definición de relación de Krylov pseudo-ortogonal, y por tanto puede utilizarse como punto de partida para reanudar la iteración de pseudo-Lanczos. A partir de entonces, son extendidas tanto la base $\tilde{V}_p$, como la matriz del cociente de Rayleigh, la cual obtiene la forma característica para este tipo de reinicio, esto es, una parte con forma de flecha más una tridiagonal (3.10),

$$\tilde{T}_k = \begin{bmatrix} D_p & b\,e_1^* \\ e_1b^* & \hat{T}_k \end{bmatrix}, \quad \text{donde} \qquad \hat{T}_k = \begin{bmatrix} \alpha_{p+1} & \beta_{p+1} & & & \\ \beta_{p+1} & \alpha_{p+2} & \ddots & & \\ & \ddots & \ddots & \beta_{k-1} & \\ & & \beta_{k-1} & \alpha_k & \end{bmatrix}. \tag{3.31}$$

El método pseudo-Lanczos tiene dos fuentes de inestabilidad potenciales: la primera viene determinada por la ortogonalización con un producto escalar indefinido (pasos 5 y 6 del Algoritmo 3.1), y la segunda por la actualización (3.29) que se lleva a cabo en el proceso de reinicio descrito. Dicha actualización supone la multiplicación por una matriz que es ortogonal respecto al pseudo-producto definido por una signatura. Con el fin de evitar una actualización que implique la multiplicación por una matriz de elevado número de condición, se ha considerado la diagonalización alternativa descrita en 3.1.1. En ella se evita realizar transformaciones que supongan un número de condición alto, y deja las columnas no ortogonalizadas en la parte que será descartada al truncar.

La técnica de reinicio grueso descrita permite reducir la dimensión del subespacio de Krylov, al tiempo que purga direcciones no deseadas. Al reducir la dimensión de la base almacenada, reduce también el coste de la ortogonalización. Otra oportunidad para la reducción del coste computacional se consigue al hacer deflación con

los vectores de la base de pseudo-Lanczos que se corresponden con valores convergidos. Dichos vectores quedan bloqueados y no se vuelven a actualizar en reinicios posteriores. En este caso el cociente de Rayleigh tiene la forma

$$C_k = \Omega_k^{-1} \begin{bmatrix} D_\ell & 0 \\ 0 & G \end{bmatrix}, \tag{3.32}$$

donde D_ℓ es simétrica y diagonal a bloques (con tamaño máximo de bloque 2), y G es simétrica. Esto supone la reducción del tamaño de la diagonalización a calcular (ahora sólo necesaria para G), aunque no el de la base de pseudo-Lanczos, es decir, no se reduce el coste de la B-ortogonalización con respecto a estos vectores bloqueados.

3.3. Resolución de QEPs simétricos mediante linealización

En esta sección se aborda la resolución de problemas cuadráticos de valores propios para el caso en que las matrices coeficientes son simétricas reales, o hermíticas complejas. El enfoque que seguimos en este tipo de problemas es el de construir una linealización simétrica equivalente (3.3), la cual se resuelve mediante el método pseudo-Lanczos descrito en la sección anterior. Al igual que en el problema no simétrico, tratado en el capítulo 2, también ahora se consideran esquemas que tienen en cuenta la estructura de la linealización y minimizan la cantidad de memoria necesaria para el almacenamiento de los vectores de Krylov que se generan. En primer lugar, se presenta el que llamamos método de Q-Lanczos, el cual es una adaptación, para el caso simétrico, del método de Q-Arnoldi [66] descrito en 2.1.2.

3.3.1. Q-Lanczos

Al resolver el QEP simétrico (3.1) aplicando el método pseudo-Lanczos a la linealización (3.3), se generan relaciones del tipo (3.19), las cuales expresamos, de forma más explícita, como

$$\begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} V_k^0 \\ V_k^1 \end{bmatrix} = \begin{bmatrix} V_k^0 & v^0 \\ V_k^1 & v^1 \end{bmatrix} \Omega_{k+1}^{-1} \underline{T}_k, \tag{3.33}$$

donde $\underline{T}_k \in \mathbb{R}^{(k+1)\times k}$ representa una tridiagonal extendida, formada por la matriz T_k en la parte superior y $\beta_k e_k^*$ en la fila inferior. Al igual que en el método de Q-Arnoldi, la estructura de la matriz en (3.33) determina la dependencia entre las mitades superior e inferior de los vectores de Krylov,

$$V_k^1 = \begin{bmatrix} V_k^0 & v^0 \end{bmatrix} \Omega_{k+1}^{-1} \underline{T}_k. \tag{3.34}$$

Esto hace que sea posible, siguiendo un esquema similar al utilizado en Q-Arnoldi, representar la relación de pseudo-Lanczos (3.33) sin necesidad de almacenar la base V_k entera, sino únicamente la parte superior V_k^0. La adaptación al problema cuadrático simétrico del método de Q-Arnoldi, es lo que nosotros bautizamos como método

Algoritmo 3.2 Q-Lanczos

Entrada: Matrices hermíticas M, C, $K \in \mathbb{C}^{n\times n}$ que definen el problema, vectores iniciales w^0 y $w^1 \in \mathbb{C}^n$

Salida: Matrices $\underline{T}_k \in \mathbb{C}^{(k+1)\times k}$, $\Omega_{k+1} \in \mathbb{C}^{(k+1)\times(k+1)}$, y vectores $V_k^0 \in \mathbb{C}^{n\times k}$ y $v^0, v^1 \in \mathbb{C}^n$ que satisfacen (3.33) para $V_k^1 = V_{k+1}\Omega_{k+1}^{-1}\underline{T}_k$

1: /* Normalización vector inicial w */
$\delta = \operatorname{sgn}(-w^{0*}Kw^0 + w^{1*}Mw^1)\sqrt{|-w^{0*}Kw^0 + w^{1*}Mw^1|}$ /* $\delta = \|w\|_B$ */
$v^0 = w^0/\delta, \quad v^1 = w^1/\delta, \quad \omega_1 = \operatorname{sgn}(\delta)$ /* $\omega_1 = v^*Bv$ */

2: $V_1^0 \leftarrow \begin{bmatrix} v^0 \end{bmatrix}$

3: **para** $j = 1, 2, \ldots, k$ **hacer**

4: /* Extensión subespacio de Krylov $w = Sv$ */
$w^0 = v^1$
$w^1 = -M^{-1}(Kv^0 + Cv^1)$

5: /* Coeficientes de Gram-Schmidt $h_j = (V_j^*BV_j)^{-1}V_j^*Bw$ */
$t = \underline{T}_{j-1}^*\Omega_j^{-1}\begin{bmatrix} V_{j-1}^0 & v^0 \end{bmatrix}^* Mw^1$
$h_j = \Omega_j^{-1}\begin{bmatrix} -(V_{j-1}^0)^*Kw^0 + t \\ -v^{0*}Kw^0 + v^{1*}Mw^1 \end{bmatrix}$

6: /* Ortogonalización $\tilde{w} = w - V_j h_j$ */
$\tilde{w}^0 = w^0 - \begin{bmatrix} V_{j-1}^0 & v^0 \end{bmatrix} h_j$
$\tilde{w}^1 = w^1 - \left[\begin{bmatrix} V_{j-1}^0 & v^0 \end{bmatrix}\Omega_j^{-1}\underline{T}_{j-1} \quad v^1\right] h_j$

7: /* Normalización */
$h_{j+1,j} = \|\tilde{w}\|_B, \quad \omega_{j+1} = \operatorname{sgn}(\|\tilde{w}\|_B)$
$v^0 = \tilde{w}^0/h_{j+1,j}, \quad v^1 = \tilde{w}^1/h_{j+1,j}$

8: /* Almacenamos nuevo vector de Krylov (parte de arriba)*/
$V_{j+1}^0 \leftarrow \begin{bmatrix} V_j^0 & v^0 \end{bmatrix}$

9: **fin para**

Q-Lanczos. Éste último trabaja de forma similar a Q-Arnoldi, aunque debe tener en cuenta la B-ortogonalización de los vectores de Krylov.

El Algoritmo 3.2 muestra la forma en que Q-Lanczos lleva a cabo la iteración de pseudo-Lanczos para calcular los vectores V_k^0 y el cociente de Rayleigh $\Omega_k^{-1}T_k$. En el algoritmo, V_j^0 y V_j^1 denotan, para $j \in \mathbb{N}$, las mitades superior e inferior de la base de pseudo-Lanczos V_j. En general, de forma similar a como se hizo en el capítulo 2, dado un vector x, se utiliza x^0 y x^1 para denotar las dos mitades de x.

Al igual que en la iteración de pseudo-Lanczos (Algoritmo 3.1), el cociente de Rayleigh generado por Q-Lanczos tiene valores fuera de la tridiagonal que no son exactamente cero. Al contrario de lo que ocurre en pseudo-Lanczos, donde estos valores son descartados, Q-Lanczos necesita almacenar todos los coeficientes de Gram-Schmidt obtenidos en la ortogonalización (completa), para poder recuperar, cada vez que es necesario, los vectores V_k^1. Una vez ha sido generada una relación (3.33), los elementos fuera de la tridiagonal de $\Omega_k^{-1}T_k$ se descartan, para continuar, entonces, con la pseudo-diagonalización del problema proyectado y el proceso del reinicio tal como se ha descrito en la sección 3.2.

Reinicio en Q-Lanczos. El reinicio de Krylov-Schur puede ser fácilmente incorporado al método de Q-Lanczos. Tal como se ha visto en la sección 3.2.2, para truncar una relación (3.33), generada con Q-Lanczos, a otra de menor orden, se hace en primer lugar una actualización de ésta utilizando una transformación pseudo-ortogonal Q_k conveniente, y en segundo lugar se trunca la descomposición resultante descartando las columnas finales de la relación (y de la base de Krylov), según (3.29) y (3.30), obteniendo

$$\begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} \tilde{V}_p^0 \\ \tilde{V}_p^1 \end{bmatrix} = \begin{bmatrix} \tilde{V}_p^0 & v^0 \\ \tilde{V}_p^1 & v^1 \end{bmatrix} \tilde{\Omega}_{p+1}^{-1} \underline{D}_p, \tag{3.35}$$

donde $\tilde{V}_p = V_k Q_p$, siendo Q_p las primeras p columnas de Q_k, $\tilde{\Omega}_p = Q_p^* \Omega_k Q_p$ y

$$\underline{D}_p = \begin{bmatrix} Q_p^* T_k Q_p \\ e_{k+1}^* \underline{T}_k Q_p \end{bmatrix}.$$

En la relación truncada (3.35), también se satisface la dependencia

$$\tilde{V}_p^1 = \begin{bmatrix} \tilde{V}_p^0 & v^0 \end{bmatrix} \tilde{\Omega}_{p+1}^{-1} \underline{D}_p, \tag{3.36}$$

por lo que la relación truncada puede ser extendida, nuevamente, a otra de orden k utilizando el Algoritmo 3.2.

Se hace notar que el método de Q-Lanczos tiene varias fuentes de inestabilidad potencial. Por un lado, las inherentes al proceso de pseudo-Lanczos, y por otro, la que se puede producir, al igual que en Q-Arnoldi, cuando la norma de la matriz del problema proyectado es elevada. Aunque esta última por sí sola no es relevante, en Q-Lanczos tiene el efecto de potenciar la primera. Esto hace, tal como se podrá comprobar en la sección 3.4 de resultados, que las posibilidades de éxito de una ejecución con Q-Lanczos sean menores, en general, que una con pseudo-Lanczos sobre la linealización.

3.3.2. STOAR

Para abordar la resolución del QEP simétrico (3.1) de forma que se mantenga la propiedad de simetría a lo largo de la ejecución, se han considerado, hasta el momento, dos opciones. En primer lugar la utilización del método pseudo-Lanczos sobre la linealización (3.3), y en segundo lugar, el uso de Q-Lanczos con el fin de ahorrar memoria aprovechando la estructura especial de la linealización (3.3). Dichos métodos son adaptaciones, respectivamente, de Krylov-Schur y Q-Arnoldi, que son métodos para el caso general no simétrico. Estas dos opciones son las que inicialmente se incluyeron en la biblioteca SLEPc hasta la aparición de TOAR [85, 59, 22], descrito en la sección 2.1.2. Este método recoge las ideas de Q-Arnoldi, en cuanto a que genera implícitamente los vectores de Krylov reduciendo la memoria necesaria para su almacenamiento, al tiempo que aborda el problema con un nuevo esquema evitando los potenciales problemas de inestabilidad que pueden aparecer en Q-Arnoldi. Con el fin de obtener mejores condiciones de estabilidad que las obtenidas con Q-Lanczos, nos planteamos la adaptación de TOAR para la resolución de problemas

cuadráticos simétricos. La adaptación desarrollada, a la que denominamos STOAR, comparte con Q-Lanczos que ambos son variantes de pseudo-Lanczos que utilizan la linealización (3.3) y que aprovechan la estructura de la misma para generar vectores de pseudo-Lanczos, almacenando únicamente vectores de dimensión n (el tamaño del problema inicial). Al igual que el método TOAR, se basa en calcular una base ortonormal, U_{k+1}, de span $\left(\begin{bmatrix} V_k^0 & v^0 & v^1 \end{bmatrix}\right)$, a partir de la cual se recuperan tanto V_k^0 como V_k^1 cuando son necesarios.

Tal como se vio en la sección 2.1.2, TOAR representa la base de Krylov como producto de dos matrices con columnas ortogonales,

$$V_k = \begin{bmatrix} U_{k+1} & \\ & U_{k+1} \end{bmatrix} \begin{bmatrix} G_k^0 \\ G_k^1 \end{bmatrix}, \tag{3.37}$$

donde G_k^0 y G_k^1 son las coordenadas respecto de U_{k+1} de V_k^0 y V_k^1, respectivamente. Sin embargo, aunque STOAR representa los vectores de Krylov de forma similar a TOAR, debido a la B-ortogonalidad de dichos vectores en el método pseudo-Lanczos, las matrices de la descomposición (3.37) en el caso de STOAR no son ya ortogonales. Utilizando dicha descomposición, la condición de B-ortogonalidad de la base V_k se expresa mediante,

$$\Omega_k = V_j^* B V_j = G_j^* \begin{bmatrix} U_{j+1}^* & \\ & U_{j+1}^* \end{bmatrix} \begin{bmatrix} -K & \\ & M \end{bmatrix} \begin{bmatrix} U_{j+1} & \\ & U_{j+1} \end{bmatrix} G_j. \tag{3.38}$$

Así, definiendo la matriz hermítica

$$\hat{B}_j = \begin{bmatrix} -\hat{K}_j & 0 \\ 0 & \hat{M}_j \end{bmatrix}, \quad \text{con} \quad \hat{K}_j = U_j^* K U_j, \quad \hat{M}_j = U_j^* M U_j, \tag{3.39}$$

se obtiene que las columnas de la matriz G_j de (3.37), en el caso de STOAR no son ortogonales, sino $\hat{B}_{j+1}$-ortogonales. La relación de pseudo-Lanczos que este método genera toma la forma,

$$\begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} U_{k+1}G_k^0 \\ U_{k+1}G_k^1 \end{bmatrix} = \begin{bmatrix} U_{k+2}G_{k+1}^0 \\ U_{k+2}G_{k+1}^1 \end{bmatrix} \Omega_{k+1}^{-1} \underline{T}_k, \tag{3.40}$$

donde $\hat{T}_k = \Omega_k^{-1} T_k$ es tridiagonal pseudo-simétrica, las columnas de U_{k+2} son ortogonales y las de G_{k+1} son $\hat{B}_{k+2}$-ortogonales.

En el Algoritmo 3.3, se detalla la forma en que el método STOAR genera una relación de pseudo-Lanczos del tipo (3.40). Dicho algoritmo generaliza el mostrado en la sección 2.1.2 para el método TOAR utilizando un producto escalar no estándar para la ortogonalización con Gram-Schmidt de los vectores de Krylov (creados de forma implícita). Para generar las columnas de U_j, STOAR (al igual que TOAR) utiliza el proceso de Gram-Schmidt convencional (pasos 1 y 6 del Algoritmo 3.3). Por otro lado, la B-normalización de un vector de la forma,

$$\tilde{w} = \begin{bmatrix} U_j & \\ & U_j \end{bmatrix} g, \tag{3.41}$$

Algoritmo 3.3 STOAR

Entrada: Matrices hermíticas M, C, $K \in \mathbb{C}^{n\times n}$ que definen el QEP y vectores iniciales w^0,$w^1 \in \mathbb{C}^n$

Salida: $\hat{\underline{T}}_k \in \mathbb{C}^{(k+1)\times k}$, $U_{k+2} \in \mathbb{C}^{n\times(k+2)}$, $G^0_{k+1}, G^1_{k+1} \in \mathbb{C}^{(k+2)\times(k+1)}$ que satisfacen (3.40)

$u_1 = w^0/\|w^0\|, \quad g^0 = \begin{bmatrix} \|w^0\| \\ 0 \end{bmatrix}$

$\hat{M}_1 = u_1^* M u_1, \quad \hat{K}_1 = u_1^* K u_1$ /* Inicialización de $\hat{M}_1, \hat{K}_1$ */

$\tilde{u} = w^1 - u_1 u_1^* w^1, \quad u_2 = \tilde{u}/\|\tilde{u}\|, \quad g^1 = \begin{bmatrix} u_1^* w^1 \\ \|\tilde{u}\| \end{bmatrix}$

$U_2 \leftarrow \begin{bmatrix} u_1 & u_2 \end{bmatrix}$

/* Actualización de $\hat{M}_2$ y $\hat{K}_2$ */

$\hat{M}_2 \leftarrow \begin{bmatrix} \hat{M}_1 & u_2^* M U_1 \\ U_1^* M u_2 & u_2^* M u_2 \end{bmatrix}, \quad \hat{K}_2 \leftarrow \begin{bmatrix} \hat{K}_1 & u_2^* K U_1 \\ U_1^* K u_2 & u_2^* K u_2 \end{bmatrix}$

$\beta = \operatorname{sgn}(-g^{0*}\hat{K}_2 g^0 + g^{1*}\hat{M}_2 g^1)\sqrt{-g^{0*}\hat{K}_2 g^0 + g^{1*}\hat{M}_2 g^1}$, /* $\beta = \|g\|_{\hat{B}_2}$ */

$g_1^0 = g^0/\beta, \quad g_1^1 = g^1/\beta, \quad G_1 \leftarrow \begin{bmatrix} g_1^0 \\ g_1^1 \end{bmatrix}$

$\omega_1 = \operatorname{sgn}(\beta), \quad \beta = \omega_1^{-1}\beta, \quad \Omega_1 \leftarrow \omega_1$ /* $\omega_1 = g_1^* \hat{B}_2 g_1$ */

para $j = 1, 2, \ldots, k$ **hacer**

  /* Expansión subespacio de Krylov, $w = Sv_j$ */

  $w^1 = -M^{-1}(KU_{j+1}g_j^0 + CU_{j+1}g_j^1)$

  /* Expansión base de TOAR U_{j+1} */

  $g^0 = \begin{bmatrix} g_j^1 \\ 0 \end{bmatrix}$ /* $w^0 = v_j^1$ */

  $\hat{w} = U_{j+1}^* w^1, \quad \tilde{u} = w^1 - U_{j+1}\hat{w}, \quad \alpha = \|\tilde{u}\|$

  $u_{j+2} = \tilde{u}/\alpha, \quad g^1 = \begin{bmatrix} \hat{w} \\ \alpha \end{bmatrix}$

  /* Actualización de $\hat{M}_{j+2}$ y $\hat{K}_{j+2}$ */

  $\hat{M}_{j+2} \leftarrow \begin{bmatrix} \hat{M}_{j+1} & u_{j+2}^* M U_{j+1} \\ U_{j+1}^* M u_{j+2} & u_{j+2}^* M u_{j+2} \end{bmatrix}, \quad \hat{K}_{j+2} \leftarrow \begin{bmatrix} \hat{K}_{j+1} & u_{j+2}^* K U_{j+1} \\ U_{j+1}^* K u_{j+2} & u_{j+2}^* K u_{j+2} \end{bmatrix}$

  /* Coeficientes de Gram-Schmidt */

  $\underline{G}_j^0 = \begin{bmatrix} G_j^0 \\ 0 \end{bmatrix}, \quad \underline{G}_j^1 = \begin{bmatrix} G_j^1 \\ 0 \end{bmatrix},$

  $h = \Omega_j^{-1}\underline{G}_j^* \hat{B}_{j+2} \begin{bmatrix} g^0 \\ g^1 \end{bmatrix}$

  $\alpha_j = e_j^* \Omega_j h$

  /* Ortogonalización de Gram-Schmidt */

  $\tilde{g}^i = g^i - \underline{G}_j h, \quad i = 0, 1$

  /* Normalización con $\|\tilde{w}\|_B = \|\tilde{g}\|_{\hat{B}_{j+2}}$ */

  $\beta_j = \|\tilde{g}\|_{\hat{B}_{j+2}}, \quad g_{j+1} = \tilde{g}/\beta_j, \quad \omega_{j+1} = \operatorname{sgn}(\beta_j)$

  $\beta_j = \omega_{j+1}\beta_j, \quad \Omega_{j+1} = \begin{bmatrix} \Omega_j & \\ & \omega_{j+1} \end{bmatrix}$

  /* Almacenamiento nuevo vector de Krylov (implícito) */

  $U_{j+2} \leftarrow \begin{bmatrix} U_{j+1} & u_{j+2} \end{bmatrix}$

  $G_{j+1} \leftarrow \begin{bmatrix} \underline{G}_j^0 & g_{j+1}^0 \\ \underline{G}_j^1 & g_{j+1}^1 \end{bmatrix}$

fin para

en los pasos 2 y 8 del Algoritmo 3.3, se realiza mediante la $\hat{B}_j$-normalización de g, utilizando el hecho de que $\|\tilde{w}\|_B = \|g\|_{\hat{B}_j}$. Los coeficientes de Gram-Schmidt se calculan, en el paso 6 del Algoritmo 3.3, según

$$\begin{aligned} h = \Omega_j^{-1} V_j^* B w &= \Omega_j^{-1} \begin{bmatrix} G_j^0 \\ G_j^1 \end{bmatrix}^* \begin{bmatrix} U_{j+1}^* & \\ & U_{j+1}^* \end{bmatrix} \begin{bmatrix} -KU_{j+1} g_j^1 \\ M(U_{j+1}\hat{w} + u_{j+2}\alpha) \end{bmatrix} \\ &= \Omega_j^{-1} \begin{bmatrix} G_j^0 \\ 0 \\ G_j^1 \\ 0 \end{bmatrix}^* \begin{bmatrix} U_{j+2}^* & \\ & U_{j+2}^* \end{bmatrix} \begin{bmatrix} -KU_{j+2} \begin{bmatrix} g_j^1 \\ 0 \end{bmatrix} \\ MU_{j+2} \begin{bmatrix} \hat{w} \\ \alpha \end{bmatrix} \end{bmatrix}. \end{aligned} \tag{3.42}$$

Al igual que en el caso de TOAR, en el método de STOAR se puede producir una terminación prematura afortunada si se obtiene un vector nulo $\tilde{g}$ en el paso 8 del Algoritmo 3.3. Por otro lado, si en ese mismo paso se obtiene un vector no nulo $\tilde{g}$ que genere un valor de β_j igual a cero, es señal de que en el proceso de pseudo-Lanczos se ha producido una terminación prematura severa.

Reinicio en STOAR. El esquema de reinicio descrito en la sección 2.1.2 para TOAR puede ser incorporado en STOAR de forma casi inmediata. Para llevar a cabo el reinicio de una relación de pseudo-Lanczos del tipo (3.40), en primer lugar, se calcula una diagonalización pseudo-simétrica del par (T_k, Ω_k), y se realizan las actualizaciones (3.29) y (3.30) correspondientes al reinicio grueso de pseudo-Lanczos, con lo que se obtiene una relación,

$$\begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} U_{k+1}\hat{G}_p^0 \\ U_{k+1}\hat{G}_p^1 \end{bmatrix} = \begin{bmatrix} U_{k+2}\hat{G}_{p+1}^0 \\ U_{k+2}\hat{G}_{p+1}^1 \end{bmatrix} \tilde{\Omega}_{p+1} \underline{D}_p,$$

en la que las matrices $\hat{G}_p^0$ y $\hat{G}_p^1$ son de dimensión $(k+1) \times p$. Al igual que ocurría al truncar una descomposición de TOAR, con este proceso se ha conseguido reducir la dimensión de la base de Krylov, pero no la cantidad de vectores en U_k, utilizados para su representación. Por ello, para reducir su número a una cantidad acorde a la nueva dimensión del subespacio de Krylov, se requiere un segundo paso en el que se lleva a cabo el proceso descrito en la sección 2.1.2 para TOAR.

Transformación espectral. En la implementación de SLEPc, cuando se utiliza una transformación espectral junto con el solver STOAR, ésta se realiza siempre sobre el problema cuadrático (ver sección 2.2.1), y es el objeto `ST` incluido en la clase `PEP` el encargado de construir las tres matrices asociadas al QEP transformado $\tilde{Q}(\theta) = \theta^2\tilde{M} + \theta\tilde{C} + \tilde{K}$, con

$$\tilde{M} = \sigma^2 M + \sigma C + K, \quad \tilde{C} = 2\sigma M + C, \quad K = M. \tag{3.43}$$

Al contrario de lo que ocurre en el solver TOAR, para STOAR no está disponible la opción de realizar la transformación sobre el problema definido por la linealización, ya que este solver no ha sido preparado para que pueda trabajar de forma implícita con las matrices de la linealización cuando sobre ésta se realiza una transformación espectral.

3.4. Evaluación computacional de los solvers cuadráticos simétricos

En esta sección se presentan resultados de una serie de experimentos llevados a cabo para analizar el comportamiento de los solvers desarrollados. Para su evaluación se utilizan las implementaciones de dichos métodos realizadas en SLEPc. También se incluye la descripción del modo en que es tratada la inestabilidad potencial de los métodos implementados.

La calidad de los pares propios calculados se mide, al igual que en los experimentos realizados en el capítulo 2, utilizando el error hacia atrás (2.88) dado, para un QEP, por la expresión,

$$\eta_Q(x,\lambda) = \frac{\|Q(\lambda)x\|_2}{(|\lambda^2|\|M\|_2 + |\lambda|\|C\|_2 + \|K\|_2)\,\|x\|_2}, \tag{3.44}$$

en la que se sustituimos la 2-norma matricial por la norma infinito.

3.4.1. Detección de inestabilidad

Tal como se ha visto en las secciones previas, los métodos que utilizan un producto escalar indefinido presentan riesgo de inestabilidad. En los métodos derivados de pseudo-Lanczos estudiados en este capítulo, se han distinguido dos momentos en los que se puede introducir inestabilidad asociada a un producto indefinido. Éstos son, por un lado, la pseudo-ortogonalización en la iteración de pseudo-Lanczos, y por otro, la diagonalización pseudo-simétrica del problema proyectado. Aunque no hemos podido encontrar en la bibliografía una forma para conseguir que estos métodos sean tan estables como otros relacionados basados en transformaciones ortogonales, se ha estudiado la forma de detectar cuándo dicha inestabilidad se produce, para poder así decidir sobre el curso de una ejecución una vez que dicha inestabilidad se presenta. Hemos podido observar que una de las consecuencias más inmediatas de la inestabilidad es la pérdida de simetría, es decir, que la relación de Krylov que se genera (3.26) deja de ser simétrica. Por ello, para evitar que el método produzca soluciones incorrectas, hemos incluido un mecanismo sencillo en el que se comprueba la simetría de la matriz T_k en cada iteración de la recurrencia de pseudo-Lanczos, y detiene la ejecución siempre que $\|T_k - T_k^*\|_F / \|T_k\|_F$ es mayor que una determinada tolerancia. Cuando ese caso se presenta, el estado de terminación del solver polinómico es de no convergencia, aunque sí retorna los pares propios calculados hasta el momento de pérdida de simetría.

3.4.2. Resultados numéricos

El entorno de ejecución utilizado para los experimentos es el descrito en la sección 2.6.1. Los problemas de test utilizados pertenecen a la colección NLEVP [14]. Todos los problemas considerados están definidos por matrices reales simétricas y se han resuelto utilizando aritmética real, excepto el problema `sign1` en el que las

matrices que lo definen son complejas hermíticas y se resuelve utilizando aritmética compleja. En la mayoría de los experimentos realizados se utiliza una transformación de desplazamiento e inversión (2.27) para calcular autovalores cercanos a un determinado valor σ, excepto en el caso de sign1 en el que se calculan autovalores con parte real en $[-0{,}9, 0{,}9]$.

Se hace notar que, en el caso de Krylov-Schur y pseudo-Lanczos, la transformación espectral se realiza sobre el problema linealizado, es decir, al extender la base de Krylov en la iteración del método, se utiliza el operador $(A - \sigma B)^{-1}B$, donde A y B son las matrices de la linealización (3.3). Por el contrario, en los casos de Q-Lanczos y STOAR, la transformación espectral se realiza directamente sobre el problema cuadrático (3.43).

En la Tabla 3.1 se muestran resultados de ejecución con 1 procesador para 6 problemas, de distintas dimensiones, de la colección NLEVP. En la tabla se muestra el error hacia atrás máximo obtenido al calcular 10 pares propios, con una base de tamaño $k = 25$ y una tolerancia de 10^{-8}. También se muestra el tiempo total de ejecución (incluyendo la factorización matricial inicial). Para la resolución del problema proyectado se ha utilizado, en todas las ejecuciones, la diagonalización alternativa, combinada con la tridiagonalización pseudo-simétrica, descritas en la sección 3.1.1. Esto es, se realiza, en primer lugar la tridiagonalización, y en segundo lugar el iterativo QR (con LAPACK), seguido de pseudo-ortogonalización.

A partir de los resultados obtenidos, podemos ver que los métodos indefinidos propuestos presentan ventajas e inconvenientes. En algunos problemas, estos métodos muestran una convergencia más rápida comparados con sus homólogos no simétricos, como puede verse, por ejemplo en shaft. También, estos métodos pueden, a veces, proporcionar resultados más exactos (por ejemplo spring). En el caso del problema sleeper, que tiene autovalores con multiplicidad 2, se ha observado que los métodos que mantienen la simetría no perturban los valores reales fuera del eje real, mientras que los métodos no simétricos, a veces, generan un par conjugado complejo (con parte imaginaria pequeña), en lugar de un valor real de multiplicidad 2. Además, cuando se resuelven problemas complejos hermíticos con aritmética compleja (por ejemplo sign1), siempre se tiene la garantía de que los métodos indefinidos obtienen una solución compleja conjuntamente con el par conjugado correspondiente, $(\lambda, \bar{\lambda})$, mientras que en el caso no simétrico puede ocurrir que no se obtenga alguno de ellos (dependiendo de la convergencia).

La principal desventaja de los métodos simétricos desarrollados es que, ocasionalmente, puede fallar la convergencia de los mismos. No obstante, el mecanismo descrito al inicio de esta sección, que detecta la presencia de inestabilidad mediante el control de la pérdida de simetría, hace que estos métodos, potencialmente inestables, sean fiables, en el sentido de que la solución que éstos generan es siempre correcta. Así, cuando el solver detecta la aparición de inestabilidad, la iteración se detiene y se retornan los pares propios convergidos hasta el momento (menos de los solicitados). A partir de los resultados que se muestran en la Tabla 3.1 podemos comprobar que el fallo de convergencia es más frecuente en Q-Lanczos (por ejemplo en sign1, sleeper y spring), mientras que pseudo-Lanczos y STOAR se comportan razonablemente bien. Dicho comportamiento de Q-Lanczos era de alguna forma es-

Tabla 3.1: Resultados de varios test provenientes de la colección NLEVP. Se calculan 10 pares propios, con una base de tamaño $k = 25$, variando el método utilizado. La tabla muestra el número de autovalores convergidos (*nconv*), la cantidad de reinicios (*its*), error hacia atrás máximo $\eta_Q(x, \lambda)$ y el tiempo de ejecución (para un proceso).

nombre	metodo	nconv	its	η_Q	tiempo
	Krylov-Schur	11	2	$1{,}7 \times 10^{-11}$	16
gen_hyper2	Pseudo-Lanczos	11	2	$1{,}2 \times 10^{-9}$	18
$n = 1000$	Q-Arnoldi	11	2	$7{,}3 \times 10^{-11}$	2.3
$\sigma = -1$	Q-Lanczos	11	2	$6{,}7 \times 10^{-9}$	3.6
	TOAR	11	2	$8{,}7 \times 10^{-11}$	2.3
	STOAR	11	2	$5{,}9 \times 10^{-10}$	2.6
	Krylov-Schur	20	1	$8{,}1 \times 10^{-13}$	0.66
schrodinger	Pseudo-Lanczos	24	1	$5{,}9 \times 10^{-12}$	0.32
$n = 1998$	Q-Arnoldi	20	1	$2{,}2 \times 10^{-13}$	0.21
$\sigma = -0{,}3$	Q-Lanczos	16	1	$5{,}6 \times 10^{-7}$	0.24
$k = 50$	TOAR	20	1	$3{,}2 \times 10^{-12}$	0.21
	STOAR	16	1	$2{,}7 \times 10^{-10}$	0.34
	Krylov-Schur	12	81	$1{,}4 \times 10^{-9}$	1.5
shaft	Pseudo-Lanczos	12	2	$4{,}4 \times 10^{-10}$	0.20
$n = 400$	Q-Arnoldi	12	83	$2{,}3 \times 10^{-6}$	0.99
$\sigma = -10$	Q-Lanczos	12	2	$2{,}9 \times 10^{-9}$	0.17
	TOAR	10	186	$2{,}9 \times 10^{-9}$	1.7
	STOAR	10	2	$6{,}0 \times 10^{-11}$	0.19
	Krylov-Schur	10	3	$5{,}7 \times 10^{-10}$	0.67
sign1	Pseudo-Lanczos	10	6	$1{,}4 \times 10^{-10}$	1.0
$n = 1000$	Q-Arnoldi	10	4	$1{,}6 \times 10^{-11}$	0.028
no t. espectral	Q-Lanczos	0	6	-	-
	TOAR	10	4	$5{,}0 \times 10^{-10}$	0.58
	STOAR	10	6	$1{,}0 \times 10^{-11}$	0.85
	Krylov-Schur	11	3	$2{,}9 \times 10^{-12}$	185
sleeper	Pseudo-Lanczos	10	3	$2{,}8 \times 10^{-13}$	202
$n = 1000000$	Q-Arnoldi	10	3	$1{,}9 \times 10^{-10}$	130
$\sigma = -0{,}9$	Q-Lanczos	2	3	$4{,}0 \times 10^{-9}$	86
	TOAR	10	3	$5{,}7 \times 10^{-14}$	123
	STOAR	10	3	$4{,}3 \times 10^{-13}$	138
	Krylov-Schur	10	2	$6{,}9 \times 10^{-12}$	151
spring	Pseudo-Lanczos	10	2	$4{,}7 \times 10^{-13}$	162
$n = 1000000$	Q-Arnoldi	10	2	$2{,}2 \times 10^{-7}$	111
$\sigma = -10$	Q-Lanczos	6	1	$8{,}8 \times 10^{-8}$	91
	TOAR	10	2	$9{,}6 \times 10^{-12}$	107
	STOAR	10	2	$4{,}1 \times 10^{-13}$	116

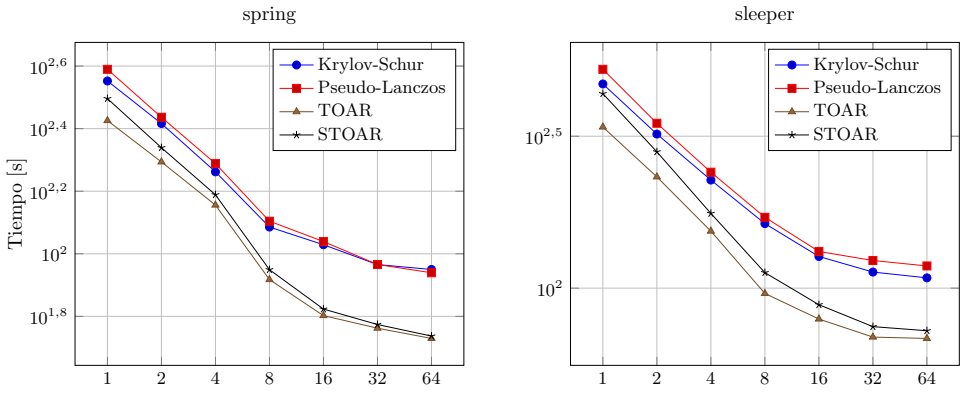

Figura 3.1: Comparación de tiempos de ejecución paralela con los problemas spring (izquierda) y sleeper (derecha), para distintos métodos, variando el número de procesos. Los parámetros de ejecución son los mismos que en la Tabla 3.1, aunque solicitando 30 autovalores.

perado, tal como se había hecho notar en la sección 3.3.1.

La Figura 3.1 muestra los tiempos de ejecución paralela con los problemas spring y sleeper, ambos para una dimensión de problema de 1 millón. Se evalúan los métodos de Krylov-Schur, pseudo-Lanczos, TOAR y STOAR, considerando una cantidad creciente de procesos. En este caso se solicitaron 30 autovalores, y se determinó un tamaño de base máximo igual a 60 vectores. Se puede ver que la escalabilidad está lejos de la ideal, lo que se puede atribuir al hecho de que los sistemas lineales, necesarios para la expansión del subespacio de Krylov, se resuelven, en paralelo, utilizando un método directo (proporcionado por MUMPS), cuya escalabilidad es bastante limitada. En la Figura 3.1 podemos también observar que los tiempos para TOAR y STOAR están por debajo de los métodos que operan con la linealización explícita (Krylov-Schur y pseudo-Lanczos), ya que en este último caso la matriz que se factoriza es de dimensión 2 veces mayor que en los otros casos. Además, se puede comprobar que los métodos indefinidos son, en general, algo más lentos que los homólogos no simétricos debido a la utilización de un producto escalar no estándar que realiza un producto matriz-vector por cada producto entre vectores.

3.5. Conclusiones

En este capítulo se han propuesto varios métodos capaces de aprovechar la simetría en la resolución de QEPs simétricos. En el primero de ellos se amplía el método pseudo-Lanczos, incorporando una técnica de reinicio grueso, que se hace necesaria al utilizar dicho método en aplicaciones reales con importantes requerimientos computacionales. Dicha ampliación ha requerido el estudio e implementación de métodos para la diagonalización (a bloques) de matrices pseudo-simétricas

(de tamaño reducido), para los que se han considerado varias técnicas con las que minimizar el riesgo potencial de inestabilidad que supone la utilización de las transformaciones hiperbólicas que en dichos métodos se utilizan. También se ha abordado la cuestión de cómo tratar en el método de pseudo-Lanczos la condición de terminación severa. Por otra parte, se ha incorporado un mecanismo que comprueba la simetría del cociente de Rayleigh durante la ejecución, para asegurar que el método de pseudo-Lanczos, potencialmente inestable, devuelve únicamente resultados correctos. Los otros métodos considerados, Q-Lanczos y STOAR, son variantes del primero, por lo que incluyen todos los elementos incorporados en éste, en los se hace un uso más eficiente de la memoria, evitando almacenar la base de Krylov de forma completa. Estos métodos son adaptaciones al caso simétrico, de los solvers genéricos Q-Arnoldi y TOAR. Los esquemas de ahorro de memoria que estos métodos utilizan son importantes especialmente en problemas de muy elevada dimensión, o en PEPs con grado elevado. Por ello el trabajo realizado puede considerarse como un primer paso para el desarrollo de solvers polinómicos simétricos de grado arbitrario.

En las pruebas realizadas hemos podido ver que, tal como se esperaba, los métodos que se han propuesto no son tan robustos como sus equivalentes, Arnoldi básico, Q-Arnoldi y TOAR, aunque en general presentan un buen comportamiento, tanto en los resultados que proporcionan (en especial pseudo-Lanczos y STOAR), como en el rendimiento paralelo. Los solvers cuadráticos simétricos han mostrado tener cierta ventaja en algunas situaciones, en las que han proporcionado una convergencia más rápida o resultados más exactos. Se ha podido comprobar que la técnica incorporada para la detección de pérdida de simetría ha resultado ser eficaz, evitando que se retornen resultados incorrectos.

Los métodos propuestos han dado lugar a varios solvers polinómicos que han sido incluidos en SLEPc. Éstos son los que en este capítulo hemos denominado pseudo-Lanczos y STOAR. Con ello, se ha abordado uno de los objetivos planteados en esta tesis de dotar a SLEPc de un conjunto variado de solvers que proporcionen flexibilidad en la resolución de problemas con diferentes características.

Capítulo 4

Solvers basados en el método de Newton

En este capítulo se describen dos métodos que, de alguna forma, utilizan la iteración de Newton para generar aproximaciones a pares invariantes. En el primero de ellos, se aprovecha la convergencia local de Newton para mejorar o refinar una aproximación a par propio obtenida con algún otro método. El segundo, se corresponde con el método Jacobi–Davidson polinómico, el cual utiliza una ecuación de corrección de Newton para obtener los vectores con los que expandir el subespacio de búsqueda (ver sección 1.2.2). En el caso de este segundo método, se incluye un esquema de deflación, que se plantea en términos de ampliación de pares invariantes.

La resolución de un PEP vía linealización no siempre puede ser considerado un proceso que mantenga la estabilidad hacia atrás, aún en el caso de que se utilice un método estable hacia atrás para la resolución del problema asociado a la linealización [42]. Este hecho ha motivado la incorporación en SLEPc, como complemento a los métodos descritos en los capítulos anteriores, de un método con el que sea posible mejorar la exactitud de las soluciones obtenidas cuando éstas lo requieran. La opción considerada para ello es la de utilizar la iteración de Newton, descrita en la sección 1.2.2, tomando como aproximación inicial las soluciones calculadas, que se quieren mejorar. Dicho esquema ha sido utilizado, en el contexto del problema lineal generalizado, por Tisseur [87].

La utilización del método de Newton para el refinamiento de varios pares propios de forma individual, puede producir la convergencia repetida al mismo par propio, especialmente en el caso de que dichos pares propios estén próximos unos a otros. En los trabajos de Betcke y Kressner [15] para problemas polinómicos, y Kressner [51] para problemas no lineales, se evita el problema de la reconvergencia a un mismo par propio utilizando una versión a bloques del método de Newton, la cual se plantea en términos de convergencia de pares invariantes (definidos en la sección 2.5.1). Otro trabajo en el que también se evita la reconvergencia en el contexto del método de Newton aplicado en la resolución de problemas no lineales, puede encontrarse en [47].

El enfoque seguido en este caso es el de generar problemas transformados que tienen los mismos autovalores que el problema original, excepto por los ya convergidos que pasan a valer infinito en el nuevo problema.

La implementación del refinamiento iterativo realizada en esta tesis sigue el enfoque planteado en los trabajos de Betcke y Kressner [15, 51]. En la sección 4.1.2 se dan detalles del método utilizado, el cual es una generalización para PEPs definidos en la forma (1.9), del descrito en [15] para problemas polinómicos expresados en la base de monomios. Para la implementación se ha utilizado como referencia el código Matlab para problemas no lineales proporcionado por Kressner en [51], adaptándolo al problema considerado. En la sección 4.2 se abordan aspectos relativos a la resolución en paralelo de los sistemas lineales que son generados por el método de refinamiento iterativo, los cuales suponen un elevado coste computacional. Se describen varias propuestas de resolución implementadas, entre las que se ha considerado la posibilidad de reorganizar el grupo de procesos participantes en varios subgrupos comunicadores, para obtener mejoras en la escalabilidad, especialmente cuando se utilizan métodos directos. En la sección 4.4 se evalúan y comparan las distintas opciones implementadas.

Si bien la motivación del primer método considerado en este capítulo ha sido la de mejorar la robustez de los solvers incluidos en SLEPc, en el caso del segundo de los métodos, el objetivo es proporcionar solvers para el tratamiento de aplicaciones con características especiales, para las que pudiesen ser inadecuados los solvers basados en métodos de Krylov descritos en los capítulos anteriores. En términos generales, a la hora de encontrar valores propios situados en una región reducida del plano complejo, son muy eficientes los métodos basados en el método Arnoldi, utilizados conjuntamente con una transformación espectral de desplazamiento e inversión. Sin embargo, en la fase de expansión del subespacio de Krylov, dichos métodos requieren la resolución de sistemas de ecuaciones lineales, que deben ser resueltos con suficiente exactitud para asegurar el correcto funcionamiento de los mismos. Si para la resolución de dichos sistemas se utiliza un método iterativo de Krylov como, por ejemplo GMRES (*generalized minimal residual*), se hace necesario disponer de un buen precondicionador que asegure la convergencia con la exactitud necesaria. Cuando ello no es posible, se recurre a la utilización de métodos directos para la resolución de los sistemas lineales, lo que en general limita la escalabilidad de los mismos. En ocasiones, la dimensión y características de las matrices del problema no permiten este último esquema. En este contexto se opta a menudo por la utilización de métodos como Jacobi-Davison, el cual es también un método de proyección, que extiende el subespacio de búsqueda mediante la resolución de una ecuación de tipo Newton, sin requerir demasiadas exigencias en cuanto a la exactitud de las soluciones. En el ámbito de la resolución del problema polinómico, el método Jacobi–Davidson presenta la ventaja adicional, frente a los solvers basados en linealización, de trabajar manteniéndose en la dimensión del problema original.

Las consideraciones anteriores, unidas al hecho de que en la biblioteca SLEPc, en el ámbito del problema lineal de valores propios, la clase `EPS` dispone de varios solvers de tipo Davidson [73], entre los que se encuentra una implementación del método Jacobi–Davidson, ha motivado la inclusión también dentro de la clase `PEP`,

de un solver basado en dicho método.

Aunque el método Jacobi–Davidson fue desarrollado en principio para la resolución del problema lineal de valores propios [82], encontramos en la bibliografía distintos trabajos donde se proponen extensiones de dicho método para la resolución del problema polinómico y no lineal [81, 16, 91, 27]. También encontramos trabajos donde dichas extensiones han sido utilizadas dentro de aplicaciones específicas [64, 48, 46].

Para evitar la reconvergencia en el contexto del método Jacobi–Davidson, el enfoque que se sigue en el caso de problemas lineales es el de hacer deflación de los pares convergidos, lo que se consigue manteniendo el espacio de búsqueda ortogonal a los autovectores calculados. Sin embargo, en el caso de problemas polinómicos, dicho esquema de deflación, aunque utilizado en ocasiones, no permite la obtención de autovalores cuyo autovector asociado sea combinación lineal de los ya calculados. Effenberger [27] propone una técnica de deflación en el contexto del problema no lineal de valores propios, planteada en términos de ampliación de pares invariantes. En dicho trabajo se incluye una descripción completa de cómo utilizar dicho esquema de deflación en el ámbito del método Jacobi–Davidson.

La implementación del método Jacobi–Davidson incluida en SLEPc sigue el enfoque propuesto por Effenberger. Los cambios introducidos van dirigidos a obtener una adaptación adecuada para la resolución del problema polinómico definido en la forma genérica (1.9). En la sección 4.3 se dan detalles del método utilizado, y se abordan algunas cuestiones relativas a la implementación paralela del mismo.

4.1. Refinamiento de pares invariantes mediante el método de Newton

En esta sección se dan detalles del método de refinamiento iterativo de pares invariantes cuya implementación ha sido incluida en SLEPc. El enfoque utilizado se basa en el propuesto en [15], para problemas polinómicos definidos según la base de monomios; y en [51], para problemas no lineales. En dichos trabajos, y como parte de la definición de la función de la cual se calculan los ceros mediante el método de Newton, se plantea una condición de ortogonalidad que utiliza la expresión (2.76). En nuestro caso, se ha creído más conveniente plantear dicha condición de ortogonalidad mediante la expresión (2.77), en la que se utiliza la misma base de polinomios con la que se define el PEP. A continuación se dan algunos resultados relativos a pares invariantes que se utilizarán en la sección 4.1.2 para asegurar la convergencia del método de Newton.

4.1.1. Pares invariantes simples

En la sección (2.5.1) se introdujo el concepto de par invariante Φ-minimal de un polinomio en la forma (1.9), como un modo de asegurar que un autovalor asociado al par invariante es a su vez un autovalor del polinomio matricial. En este capítulo,

se requiere el concepto de par invariante simple para asegurar la convergencia del método de Newton.

Definición 4.1. Un par invariante Φ-minimal, (X, H), del polinomio (1.9) se dice simple si la multiplicidad algebraica de cada autovalor de H coincide con la multiplicidad algebraica del mismo autovalor en (1.9).

Al adaptar el método de Newton propuesto en [15], al contexto del refinamiento de pares invariantes Φ-minimales de un PEP definido según (1.9), se ha variado, respecto a dicho trabajo, la condición de ortogonalidad que aparece en la definición de la función que se utiliza en el método de Newton (4.6). Esta variación hace que, para demostrar la convergencia del método de Newton que se detalla en la sección 4.1.2, no podamos utilizar, directamente, los resultados proporcionados en dicho trabajo. Para ese fin, utilizaremos la Proposición 4.2, cuyo enunciado y demostración son similares al resultado [15, Teorema 7], utilizado en dicho trabajo para demostrar la convergencia del método de Newton aplicado al refinamiento de pares invariantes de polinomios definidos en función de la base de monomios.

Previo a dicho resultado, a continuación se da una expresión, obtenida a partir de la recurrencia (2.20), para el cálculo recursivo de la derivada de Fréchet $\mathrm{D}\,\Phi_j(H)$ de la función matricial Φ_j en H, para $j \geq 0$:

$$\mathrm{D}\,\Phi_j(H)(\Delta H) = \begin{cases} 0, & j = 0 \\ \alpha_0^{-1}\Delta H, & j = 1 \\ \alpha_{j-1}^{-1}\big(\mathrm{D}\,\Phi_{j-1}(H)(\Delta H)(H - \beta_{j-1}I) + \Phi_{j-1}(H)\Delta H - \\ \qquad - \gamma_{j-1}\,\mathrm{D}\,\Phi_{j-2}(H)(\Delta H)\big), & j > 1. \end{cases} \tag{4.1}$$

Proposición 4.2. *Sea (X, H) un par invariante Φ-minimal para el polinomio P definido en* (1.9)*. Entonces el sistema lineal*

$$\begin{cases} \mathbb{P}(\Delta X, H) + \displaystyle\sum_{j=0}^{d} A_j X\,\mathrm{D}\,\Phi_j(H)(\Delta H) = \mathbb{P}(X, H) \\ (W^0)^*\Delta X + \displaystyle\sum_{j=1}^{d-1} (W^j)^*\left(\Delta X\,\Phi_j(H) + X\,\mathrm{D}\,\Phi_j(H)(\Delta H)\right) = 0, \end{cases} \tag{4.2}$$

con $\mathbb{P}$ definido en (2.75)*, tiene una única solución $(\Delta X, \Delta H)$ si y sólo si (X, H) es simple.*

Demostración. Probaremos, de forma equivalente, que el sistema lineal homogéneo asociado a (4.2) tiene como única solución $(\Delta X, \Delta H) = (0, 0)$ sii (X, H) es simple. Dado que $A - \lambda B$ definida en (2.21) es una linealización fuerte para el polinomio (1.9), se tiene que (X, H) es simple sii (V, H) es un par invariante simple para dicha linealización, siendo $V := V_d(X, H)$ definido en (2.77). Esto es equivalente [15] a que el sistema

$$\begin{cases} A\Delta V - B\Delta V H - BV\Delta H = 0, \\ W^*\Delta V = 0, \end{cases} \tag{4.3}$$

tiene como única solución $(\Delta V, \Delta H) = (0, 0)$. A continuación veremos (por reducción al absurdo), que esta última condición es equivalente a que el sistema lineal homogéneo asociado a (4.2) tiene como única solución $(\Delta X, \Delta H) = (0, 0)$.

Supongamos en primer lugar que $(\Delta X, \Delta H)$ es una solución no nula del sistema lineal homogéneo asociado a (4.2). Entonces, definiendo

$$\Delta V = \begin{bmatrix} \Delta X\,\Phi_0(H) + X\,\mathrm{D}\,\Phi_0(H)(\Delta H) \\ \Delta X\,\Phi_1(H) + X\,\mathrm{D}\,\Phi_1(H)(\Delta H) \\ \vdots \\ \Delta X\,\Phi_{d-1}(H) + X\,\mathrm{D}\,\Phi_{d-1}(H)(\Delta H) \end{bmatrix}, \tag{4.4}$$

y utilizando las recurrencias (2.20) y (4.1), se tiene

$$\begin{aligned}
&(e_1 \otimes I)^T(A\Delta V - B\Delta V H) = \beta_0\Delta X + \alpha_0\Delta X\,\Phi_1(H) + \alpha_0 X\,\mathrm{D}\,\Phi_1(H)(\Delta H) - \Delta X H \\
&\quad = \Delta X\,\Phi_0(H)H + X\,\Phi_0(H)\Delta H - \Delta X H = X\Delta H, \\
&(e_{j+1} \otimes I)^T(A\Delta V - B\Delta V H) = \gamma_j(\Delta X\,\Phi_{j-1}(H) + X\,\mathrm{D}\,\Phi_{j-1}(H)(\Delta H)) \\
&\quad + \beta_j(\Delta X\,\Phi_j(H) + X\,\mathrm{D}\,\Phi_j(H)(\Delta H)) + \alpha_j(\Delta X\,\Phi_{j+1}(H) + X\,\mathrm{D}\,\Phi_{j+1}(H)(\Delta H)) \\
&\quad - (\Delta X\,\Phi_j(H) + X\,\mathrm{D}\,\Phi_j(H)(\Delta H))H = \Delta X\,\Phi_j(H)H + X(\Phi_j(H)\Delta H \\
&\quad + \mathrm{D}\,\Phi_j(H)(\Delta H)H) - (\Delta X\,\Phi_j(H) + X\,\mathrm{D}\,\Phi_j(H)(\Delta H))H = X\,\Phi_j(H)\Delta H, \\
&(e_d \otimes I)^T(A\Delta V - B\Delta V H) = -c_{d-1}\mathbb{P}(\Delta X, H) - c_{d-1}\sum_{j=1}^{d} A_j X\,\mathrm{D}\,\Phi_j(H)(\Delta H) \\
&\quad + c_d A_d\Delta X\,\Phi_{d-1}(H)H + c_d A_d X\,(\Phi_{d-1}(H)\Delta H + \mathrm{D}\,\Phi_{d-1}(H)(\Delta H)H) \\
&\quad - c_d A_d(\Delta X\,\Phi_{d-1}(H) + X\,\mathrm{D}\,\Phi_{d-1}(H)(\Delta H))H = c_d A_d X\,\Phi_{d-1}(H)\Delta H,
\end{aligned}$$

de donde se sigue que $A\Delta V - B\Delta V H = BV\Delta H$, y que $(\Delta V, \Delta H)$ es una solución no nula de (4.3) (la condición $W^*\Delta V = 0$ es inmediata).

Supongamos ahora que $(\Delta V, \Delta H) \neq (0, 0)$ es una solución no nula de (4.3). Dividiendo ΔV en d bloques de filas, que denotamos por ΔV_i para $i = 0, \ldots, d-1$, para los primeros $(d-1)$ bloques se cumplen las relaciones,

$$\begin{aligned}
&\beta_0\Delta V_0 + \alpha_0\Delta V_1 - \Delta V_0 H = X\Delta H, \\
&\gamma_j\Delta V_{j-1} + \beta_j\Delta V_j + \alpha_j\Delta V_{j+1} - \Delta V_j H = X\,\Phi_j(H)\Delta H, \quad 0 < j < d-1,
\end{aligned}$$

a partir de las cuales, utilizando (2.20) y (4.1), por inducción se obtiene

$$\begin{aligned}
\Delta V_1 &= X\,\mathrm{D}\,\Phi_1(H)(\Delta H) + \Delta V_0, \Phi_1(H), \\
\Delta V_{j+1} &= \alpha_j^{-1}(X\,\Phi_j(H)\Delta H + (X\,\mathrm{D}\,\Phi_j(H)(\Delta H) + \Delta V_0\,\Phi_j(H))(H - \beta_j I) \\
&\quad - \gamma_j(X\,\mathrm{D}\,\Phi_{j-1}(H)(\Delta H) + \Delta V_0\,\Phi_{j-1}(H)) \\
&= X\,\mathrm{D}\,\Phi_{j+1}(H)(\Delta H) + \Delta V_0\,\Phi_{j+1}(H), \quad 0 < j < d-1.
\end{aligned} \tag{4.5}$$

De las expresiones obtenidas en (4.5) se deduce que $(\Delta V_0, \Delta H)$ es no nulo, ya que de otra forma, $(\Delta V, \Delta H)$ sería nulo. Utilizando dichas expresiones junto con la condición $W^*\Delta V = 0$, se tiene también que $(\Delta V_0, \Delta H)$ verifica la segunda ecuación del sistema (4.2).

Igualando ahora el último bloque de filas en (4.3), y sustituyendo las expresiones (4.5) obtenemos

$$
\begin{aligned}
0 &= \sum_{i=0}^{d-1} c_{d-1} A_i \Delta V_i - c_d A_d(\gamma_{d-1}\Delta V_{d-2} + \Delta V_{d-1}(\beta_{d-1} I - H) \\
&- V_{d-1}\,\Phi_{d-1}(H)\Delta H) = c_{d-1} \sum_{i=0}^{d} A_i (X\,\mathrm{D}\,\Phi_i(H)(\Delta H) + \Delta V_0\,\Phi_i(H)) \\
&- c_d A_d X(\gamma_{d-1}\,\mathrm{D}\,\Phi_{d-2}(H)(\Delta H) + \mathrm{D}\,\Phi_{d-1}(H)(\Delta H)(\beta_{d-1} I - H) - \Phi_{d-1}(H)\Delta H) \\
&- c_d A_d \Delta V_0(\gamma_{d-1}\,\Phi_{d-2}(H) + \Phi_{d-1}(H)(\beta_{d-1} I - H) - \Phi_{d-1}(H)\Delta H) \\
&= c_d \sum_{i=0}^{d} (X\,\mathrm{D}\,\Phi_i(H)(\Delta H) + \Delta V_0\,\Phi_i(H)),
\end{aligned}
$$

de donde se concluye que $(\Delta V_0, \Delta H)$ verifica el sistema homogéneo asociado a (4.2). □

4.1.2. Método de Newton para la resolución de PEPs

En esta sección se revisa el método de Newton para problemas no lineales de valores propios desarrollado en [15], y se detalla la forma en que éste ha sido adaptado para su implementación en SLEPc. En primer lugar, y de forma similar a los trabajos de Betcke y Kressner [15, 51], planteamos el problema del cálculo de un par invariante para un PEP expresado en la forma (1.9), en términos de la búsqueda de los ceros de una función. Para ello, consideramos el problema equivalente de obtener un par $(X, H) \in \mathbb{C}^{n\times k} \times \mathbb{C}^{k\times k}$ de forma que se satisface

$$\mathbb{P}(X, H) = 0, \qquad \text{y} \tag{4.6a}$$

$$\mathbb{V}(X, H) := W^* V_d(X, H) - I_k = 0, \tag{4.6b}$$

para $\mathbb{P}$ dada en (2.75), $V_d(X, H)$ definido en (2.77) y $W \in \mathbb{C}^{dn\times k}$ una matriz con rango por columnas completo. A la hora de aplicar el método de Newton para la resolución de dicho sistema de ecuaciones no lineales, es necesario comprobar la invertibilidad del operador lineal

$$\mathbb{L}(X, H) := (\mathrm{D}\,\mathbb{P}(X, H), \mathrm{D}\,\mathbb{V}(X, H)), \tag{4.7}$$

donde $\mathrm{D}\,\mathbb{P}(X, H)$ y $\mathrm{D}\,\mathbb{V}(X, H)$ son las derivadas de Fréchet en (X, H) de $\mathbb{P}$ y $\mathbb{V}$, respectivamente. Una expresión explícita de dicho operador aparece en el sistema planteado en la Proposición 4.2, con la que se prueba que, en el caso de que el par invariante considerado sea simple, dicho operador es invertible y el método de Newton aplicado a (4.6) converge (localmente) de forma cuadrática. Así, dada una aproximación $(\tilde{X}, \tilde{H})$ a un par invariante simple (X, H) de (1.9), ésta puede utilizarse como punto de partida del método de Newton aplicado a (4.6), para obtener una solución refinada que esté más cerca de (X, H).

Siempre que la aproximación inicial $(\tilde{X}, \tilde{H})$ sea suficientemente cercana a (X, H), el método de Newton genera una sucesión $\{(X_i, H_i)\}_{i\in\mathbb{N}}$, dada por

$$(X_{i+1}, H_{i+1}) = (X_i, H_i) - (\mathbb{L}(X_i, H_i))^{-1}(\mathbb{P}(X_i, H_i), \mathbb{V}(X_i, H_i)), \tag{4.8}$$

que converge cuadraticamente a (X, H).

Algoritmo 4.1 Método de Newton para el refinamiento de pares invariantes

Entrada: Par inicial $(X_0, H_0) \in \mathbb{C}^{n\times k} \times \mathbb{C}^{k\times k}$ tal que $V_d(X_0, H_0)^* V_d(X_0, H_0) = I_k$
Salida: Solución aproximada (X_{i+1}, H_{i+1}) para (4.6)

1: $W \leftarrow V_d(X_0, H_0)$
2: **para** $i = 0, 1, \ldots,$ maxit **hacer**
3: Cálculo del residuo $R \leftarrow \mathbb{P}(X_i, H_i)$
4: Obtención de $(\Delta X, \Delta H)$ tal que $\mathbb{L}(X_i, H_i)(\Delta X, \Delta H) = (R, 0)$
5: $\tilde{X}_{i+1} \leftarrow X_i - \Delta X, \quad \tilde{H}_{i+1} \leftarrow H_i - \Delta H$
6: Cálculo de la descomposición QR compacta $V_d(\tilde{X}_{i+1}, \tilde{H}_{i+1}) = WT$
7: $X_{i+1} \leftarrow \tilde{X}_{i+1} T^{-1}, \quad H_{i+1} \leftarrow T\tilde{H}_{i+1} T^{-1}$
8: Comprobación de la convergencia, fin si satisfecha
9: **fin para**

El Algoritmo 4.1, extraído de [51], muestra la iteración de Newton para, a partir de una aproximación a un par invariante simple de (1.9), calcular iterativamente una mejor aproximación. Para evitar trabajar con bases mal condicionadas, al inicio de cada iteración $j \geq 0$ de dicho algoritmo, la matriz W se fija de forma que $W^* V_d(X_j, H_j) = I$. Para ello, en la iteración previa (paso 6), se calcula una descomposición QR compacta de $V_d(\tilde{X}_j, \tilde{H}_j)$ y se actualiza la aproximación $(\tilde{X}_j, \tilde{H}_j)$ de forma conveniente (paso 7). Se hace notar que no es necesario calcular explícitamente la factorización QR de $V_d(X_j, H_j)$, de tamaño $(dn \times k)$, sino que la matriz T de la actualización puede obtenerse, con menor coste computacional, descomponiendo X_j en la forma, $X_j = UG_j$, con $U \in \mathbb{C}^{n\times k}$ tal que $U^*U = I$ y $G_j \in \mathbb{C}^{k\times k}$. Con ello se obtiene la expresión

$$V_d(X_j, H_j) = (I_d \otimes U) V_d(G_j, H_j), \tag{4.9}$$

donde

$$V_d(G_j, H_j) := \begin{bmatrix} G_j \, \Phi_0(H_j) \\ \vdots \\ G_j \, \Phi_{d-1}(H_j) \end{bmatrix}, \tag{4.10}$$

lo que permite obtener T por medio de la descomposición QR de menor tamaño, $V_d(G_j, H_j) = \tilde{U}T$.

El paso 4 del Algoritmo 4.1 se corresponde con la resolución del sistema de ecuaciones lineales

$$\begin{cases} \mathrm{D}\,\mathbb{P}(X_i, H_i)(\Delta X, \Delta H) = \mathbb{P}(X_i, H_i) \\ \mathrm{D}\,\mathbb{V}(X_i, H_i)(\Delta X, \Delta H) = 0, \end{cases} \tag{4.11}$$

dado de forma más explicita en (4.2). Para resolver dicho sistema se utiliza la técnica de sustitución progresiva (o *forward substitution*) descrita en [15, 51]. Asumiendo que la matriz H_i es triangular superior (siempre posible trabajando con la forma compleja de Schur de H_i), las distintas columnas de ΔX y ΔH se calculan sucesivamente resolviendo k sistemas lineales de dimensión $n+k$, para los que se actualiza el lado derecho a cada paso, teniendo en cuenta las columnas de la solución obtenidas en los pasos previos. A continuación se detalla cómo sería el proceso, por ejemplo, para obtener la primera columna de la solución de (4.11) mediante dicha técnica.

Al post-multiplicar (4.2) por e_1 se obtiene el sistema lineal

$$\begin{cases} P(h_{11})\Delta x_1 + \sum_{j=0}^{d} A_j X_i \,\mathrm{D}\,\Phi_j(H_i)(\Delta H)e_1 = r_1 \\ (W^0)^*\Delta x_1 + \sum_{j=1}^{d-1}(W^j)^*\left(\Delta x_1 \left[\Phi_j(H_i)\right]_{11} + X_i \,\mathrm{D}\,\Phi_j(H_i)(\Delta H)e_1\right) = f_1, \end{cases} \tag{4.12}$$

donde Δx_1 y Δh_1 son las primeras columnas de ΔX y ΔH, respectivamente; h_{11} y $[\Phi_j(H_i)]_{11}$ denotan el elemento $(1,1)$ de H_i y $\Phi_j(H_i)$, respectivamente, las cuales son matrices triangulares superiores; r_1 es la primera columna del residuo $\mathbb{P}(X_i, H_i)$; y $f_1 = 0$. Entonces, definiendo $\{\mathrm{D}\,\Phi_j(H_i)\}_{11}$, para $j = 1, \ldots,$ como la matriz triangular que verifica

$$\mathrm{D}\,\Phi_j(H_i)(C)e_1 = \{\mathrm{D}\,\Phi_j(H_i)\}_{11} Ce_1, \quad \forall C \in \mathbb{C}^{k\times k}, \tag{4.13}$$

se obtiene un sistema lineal a partir del cual es posible obtener la primera columna de ΔX y ΔH,

$$\begin{bmatrix} P(h_{11}) & \sum_{j=0}^{d} A_j X_i \{\mathrm{D}\,\Phi_j(H_i)\}_{11} \\ \sum_{j=0}^{d-1} \Phi_j(h_{11})(W^j)^* & \sum_{j=1}^{d-1}(W^j)^* X_i \{\mathrm{D}\,\Phi_j(H_i)\}_{11} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta h_1 \end{bmatrix} = \begin{bmatrix} r_1 \\ f_1 \end{bmatrix}. \tag{4.14}$$

Las matrices (4.13) se pueden obtener recursivamente a partir de (4.1):

$$\begin{aligned} &\{\mathrm{D}\,\Phi_0(H_i)\}_{11} = 0, \quad \{\mathrm{D}\,\Phi_1(H_i)\}_{11} = \alpha_0^{-1} I_k, \quad \text{y para } j > 0, \\ &\{\mathrm{D}\,\Phi_{j+1}(H_i)\}_{11} = \alpha_j^{-1}\left((h_{11} - \beta_j)\{\mathrm{D}\,\Phi_j(H_i)\}_{11} + \Phi_j(H_i) - \gamma_j \{\mathrm{D}\,\Phi_{j-1}(H_i)\}_{11}\right). \end{aligned} \tag{4.15}$$

Para el cálculo de las sucesivas columnas Δx_p y Δh_p, $p = 2, \ldots, k$, se forma el sistema correspondiente, análogo a (4.14), sustituyendo h_{11} por h_{pp} (también en (4.15)), pero teniendo en cuenta que el correspondiente lado derecho, $\left[\begin{smallmatrix} r_p \\ f_p \end{smallmatrix}\right]$, debe ser recalculado, aplicando sucesivas actualizaciones con las columnas previamente calculadas, según la expresión

$$\begin{aligned} r_p &= \mathbb{P}(X_i, H_i)e_p - \sum_{q=1}^{p-1}\left(\sum_{j=0}^{d} A_j\left(\Delta x_q \left[\Phi_j(H_i)\right]_{qp} + X_i \left[\mathrm{D}\,\Phi_j(H_i)(Z_q)\right]_p\right)\right), \\ f_p &= -\sum_{q=1}^{p-1}\left(\sum_{j=1}^{d-1}(W^j)^*\left(\Delta x_q \left[\Phi_j(H_i)\right]_{qp} + X_i \left[\mathrm{D}\,\Phi_j(H_i)(Z_q)\right]_p\right)\right), \end{aligned} \tag{4.16}$$

donde $[\Phi_j(H_i)]_{qp}$ denota el elemento (q,p) de $\Phi_j(H_i)$, y $[\mathrm{D}\,\Phi_j(H_i)(Z_q)]_p$ la p-ésima columna of $\mathrm{D}\,\Phi_j(H_i)(Z_q)$, siendo $Z_q := \Delta x_q e_q^T$.

Se hace notar que el método de refinamiento iterativo descrito, es válido cuando se quiere refinar de forma individual pares propios simples, los cuales pueden ser considerados pares invariantes simples de dimensión $k = 1$. Esto ha motivado la consideración de dos variantes de refinamiento, cuando se calcula un conjunto de q pares propios. Por un lado, la *variante múltiple* refina el par invariante de tamaño $k = q$, generando un par invariante de la misma dimensión. Por otro lado, la *variante simple* refina cada par propio individualmente, resolviendo q sistemas lineales de la forma (4.2) con $k = 1$. Ambas variantes han sido incluidas en la implementación del solver realizada.

En términos computacionales, la parte más costosa del procedimiento descrito en el Algoritmo 4.1 es el paso 4, correspondiente a la resolución del sistema de ecuaciones lineales. Con el fin de minimizar el tiempo requerido para la resolución de los sistemas que la iteración de Newton genera, se han estudiado y evaluado diversos métodos para llevar a cabo tales resoluciones. Las diversas opciones consideradas se desarrollan en detalle en la siguiente sección.

4.2. Resolución de la ecuación de corrección

En cada iteración del método de Newton expuesto en la sección 4.1.2, la resolución de la ecuación de corrección (4.11), mediante el método de sustitución progresiva, conlleva la resolución de k sistemas de ecuaciones lineales de dimensión $n+k$, siendo n la dimensión de las matrices del problema polinómico y k la dimensión del par invariante aproximado que se pretende refinar. El coste computacional de la resolución de estos sistemas de ecuaciones puede suponer un porcentaje elevado de la computación total, pudiendo ser incluso mayor que el tiempo requerido para la obtención del par invariante inicial. Por ello, se han estudiado diversas alternativas que buscan una resolución eficiente de dichos sistemas, explotando cuando es posible, más de un nivel de paralelismo.

Los sistemas lineales, (4.14), generados por el método de sustitución progresiva, tienen la forma

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \tag{4.17}$$

donde la matriz de coeficientes es una *matriz con borde* en la que los bloques tienen dimensiones, $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times k}$, $C \in \mathbb{C}^{k\times n}$ y $D \in \mathbb{C}^{k\times k}$, con $k \ll n$. En la Figura 4.1 se representa la distribución paralela de las cuatro submatrices. La matriz D se almacena de forma secuencial replicada en cada proceso, mientras que los bordes B y C se almacenan como arrays de vectores paralelos representados en sendos objetos de la clase `BV` (ver sección 1.4.2). Respecto al bloque principal, éste es de la forma $A = P(h)$ y se almacena como una matriz estándar de PETSc, en la que cada proceso es propietario de un rango contiguo de filas. Al refinar un par invariante aproximado, (X, H), se tiene que $A = P(h_{ii})$ siendo h_{ii} un elemento de la

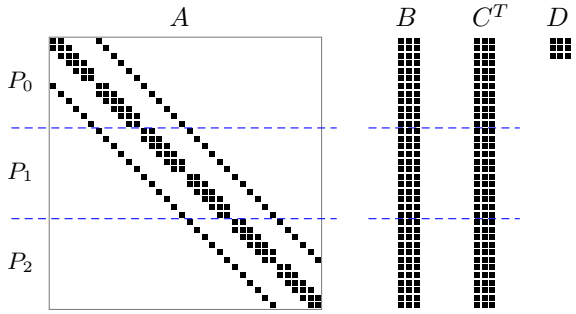

Figura 4.1: Distribución paralela de matrices y vectores.

diagonal de H, que está en forma de Schur. Este hecho hace que dicho bloque esté próximo a ser singular (más cuanto mejor sea la aproximación del par invariante), y condiciona en gran medida el tipo de métodos que pueden ser utilizados para la resolución de estos sistemas.

4.2.1. Formación explícita del sistema

La alternativa más inmediata y que consideramos en primer lugar, es la que construye explícitamente la matriz involucrada en el sistema (4.14), como una matriz PETSc. Este enfoque, al que nos referiremos como el de la *matriz explicita*, tiene una gran ventaja en cuanto a que permite el uso de cualquier método para la resolución de sistemas lineales disponibles en PETSc, los que incluyen tanto métodos de Krylov precondicionados, como métodos directos. Sin embargo, este enfoque en principio sencillo, presenta también varios inconvenientes. Por un lado, la matriz de coeficientes del sistema pierde, en cierta medida, parte de la dispersión presente en las matrices del problema polinómico, ya que $P(\cdot)$ se bordea con franjas densas. Esto afecta, por ejemplo, al rendimiento en paralelo de la multiplicación matriz-vector, ya que cualquier proceso que posea una fila densa de la matriz necesita una copia del vector distribuido completo para llevar a cabo la parte computacional de la operación que le corresponde. Por otro lado, al construir la matriz del sistema como una matriz PETSc que se distribuye por bloques de filas consecutivas entre el grupo de procesos, se produce un desequilibrio en la carga que soporta cada proceso, hecho que puede penalizar de forma severa el rendimiento global.

Con el fin de reducir el desequilibrio de la carga que se produce entre los distintos procesos al utilizar este enfoque, en lugar de distribuir la matriz con borde tal cual se genera, se distribuye una adecuada permutación simétrica de la misma. Dicha permutación se escoge de forma que las filas densas correspondientes al bloque C se distribuyen equitativamente, situándolas en cada proceso, inmediatamente después de las filas locales del bloque principal A asignadas al mismo. La Figura 4.2 muestra la matriz, antes y después de aplicar la permutación descrita. Al trabajar con este enfoque, los vectores resultantes de una multiplicación matriz-vector deben ser

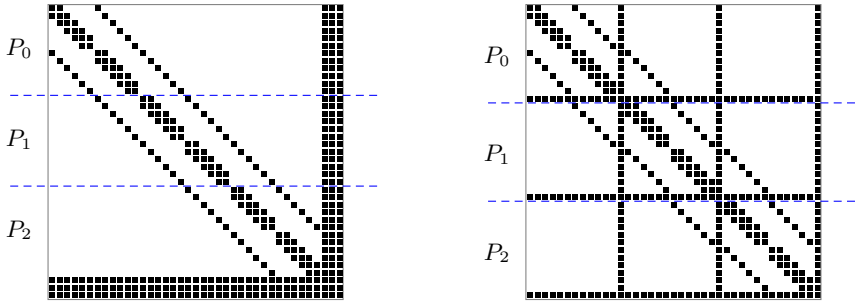

Figura 4.2: Ilustración de la distribución paralela de la matriz creada al utilizar el enfoque de matriz explicita, antes (izquierda) y después (derecha) de aplicar la permutación simétrica.

actualizados con la misma permutación.

4.2.2. Resolución mediante el complemento de Schur

La segunda opción que se ha considerado, utiliza el *complemento de Schur* del bloque diagonal inferior en (4.17), D, dado por la expresión

$$S := A - BD^{-1}C, \tag{4.18}$$

para calcular x_1 y x_2 a partir de las ecuaciones

$$Sx_1 = y_1 - BD^{-1}y_2, \tag{4.19}$$

$$x_2 = D^{-1}(y_2 - Cx_1). \tag{4.20}$$

Se hace notar que la utilización de este esquema no sería adecuada en el caso de que se utilice el complemento de Schur del bloque diagonal principal A, ya que ello supondría la resolución de sistemas lineales con una matriz cercana a ser singular.

Esta alternativa presenta una limitación en cuanto a los métodos que se pueden utilizar para la resolución del sistema (4.19) que genera. El hecho de que la matiz S sea densa hace que no deba ser explícitamente formada, y que no pueda calcularse su factorización, por lo que no pueden utilizarse métodos directos para la resolución de dicho sistema. Éste sistema puede, sin embargo, ser resuelto por métodos iterativos como GMRES o cualquiera de los métodos de Krylov disponibles en PETSc. En el curso de la resolución de un sistema de ecuaciones lineales, los métodos de Krylov realizan operaciones de multiplicación matriz-vector con la matriz del sistema (S en forma implícita en nuestro caso), y resoluciones de sistemas con una aproximación de la matriz del sistema (precondicionador). Para la construcción de dicho precondicionador, requerido por estos métodos iterativos, se utiliza la aproximación de S dada por $P := A - \text{diag}(BD^{-1}C)$, donde $\text{diag}(\cdot)$ denota la operación que, realizada sobre una matriz, hace nulos los elementos no pertenecientes a la diagonal.

4.2.3. Método de eliminación a bloques mixta

El tercer enfoque considerado permite la utilización de métodos directos para la resolución de (4.14). Utiliza el método de *eliminación a bloques mixta*, con siglas MBE (*mixed block elimination*), descrito en [33, 34], el cual está diseñado para la resolución de sistemas lineales con borde en la forma (4.17), realizando resoluciones con la matriz del bloque diagonal principal y su traspuesta, aun siendo éstas casi singulares. Los métodos de Krylov no son adecuados para la resolución de dichos sistemas, por lo que siempre se llevan a cabo utilizando métodos directos. Esto supone una limitación al tipo de problemas en los que se puede utilizar este enfoque.

Algoritmo 4.2 Método de eliminación a bloques mixta (MBE)

Entrada: $A \in \mathbb{C}^{n\times n}$, $b, y_1 \in \mathbb{C}^{n\times 1}$, $c \in \mathbb{C}^{1\times n}$ y $d, y_2 \in \mathbb{C}$ que definen el sistema lineal (4.21)
Salida: $x \in \mathbb{C}^{n+1}$ solución de (4.21) ($x = \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]$)
1: Resolver $A^T v = c^T$
2: $\delta \leftarrow d - v^T b$
3: Resolver $Aw = b$
4: $\rho \leftarrow d - cw$
5: $p_2 \leftarrow (y_2 - v^T y_1)/\delta$
6: $g_1 \leftarrow y_1 - bp_2$
7: $g_2 \leftarrow y_2 - dp_2$
8: Resolver $Az = g_1$
9: $q_2 \leftarrow (g_2 - cz)/\rho$
10: $x_1 \leftarrow z - wq_2$
11: $x_2 \leftarrow p_2 + q_2$

El algoritmo 4.2 muestra el proceso seguido por el método MBE para la resolución de un sistema lineal con un borde de tamaño uno,

$$\begin{bmatrix} A & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{4.21}$$

En él se puede comprobar, que para resolver un sistema de dimensión $n+1$ (con borde uno), el método realiza, aparte de otros cálculos de menor coste, la resolución de tres sistemas de dimensión n (pasos 1, 3 y 8). En el caso de querer resolver un sistema lineal con un borde de mayor tamaño (por ejemplo k), se trabajaría de forma recursiva, utilizando el proceso descrito en el algoritmo 4.2 para resolver, en un primer paso, un sistema de tamaño $n+k$, con bloque principal de dimensión $n+k-1$ y borde 1. Para la resolución de los tres sistemas de tamaño $n+k-1$ que el proceso genera en este caso, se utilizaría, a su vez, el mismo algoritmo para resolver, en un segundo paso, sistemas con bloque principal de tamaño $n+k-2$ (y borde 1). Siguiendo con este procedimiento recursivo se va reduciendo en uno el tamaño del borde de los sistemas a resolver, hasta llegar a un caso base de dimensión $n+1$ (con borde 1) que puede ya ser abordado directamente haciendo resoluciones con

la matriz, A, del bloque diagonal principal. A continuación se dan detalles de un ejemplo en el que se resuelve un sistema lineal (4.17) con un borde de tamaño $k = 2$,

$$\begin{array}{c} \\ n \\ 1 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} \overset{n}{} & \overset{1}{} & \overset{1}{} \end{array} \\ \left[\begin{array}{cc|c} A & b_1 & b_2 \\ c_1 & d_{11} & d_{12} \\ \hline c_2 & d_{21} & d_{22} \end{array}\right] \end{array} \left[\begin{array}{c} x_{1:n} \\ \hline x_{n+1} \\ \hline x_{n+2} \end{array}\right] = \left[\begin{array}{c} y_{1:n} \\ \hline y_{n+1} \\ \hline y_{n+2} \end{array}\right]. \tag{4.22}$$

Al aplicar el Algoritmo 4.2 para la resolución de (4.22), se resuelven tres sistemas con la submatriz de dimensión $n+1$,

$$\begin{bmatrix} A & b_1 \\ c_1 & d_{11} \end{bmatrix}, \tag{4.23}$$

y lados derechos respectivos, los vectores

$$b := \left[\begin{smallmatrix} b_2 \\ d_{12} \end{smallmatrix}\right], \quad c := \left[\begin{smallmatrix} c_2 & d_{21} \end{smallmatrix}\right], \quad g_1 \in \mathbb{C}^{n+1}, \tag{4.24}$$

con g_1 calculado a partir de y con las actualizaciones de los pasos 5 y 6 del Algoritmo 4.2. En la resolución de cada uno de los tres sistema con la matriz (4.23) (mediante el Algoritmo 4.2), se resuelven a su vez tres sistemas, dos de los cuales (pasos 1 y 3) son independientes de los lados derechos correspondientes (vectores b y c), y sólo dependen de la propia matriz (4.23), por lo que dichos cálculos son compartidos por las resoluciones de los tres sistemas requeridos por (4.22).

Aun suponiendo que se almacenan cálculos intermedios para evitar la resolución de sistemas repetidos, a la hora de resolver un sistema de dimensión $n+k$ con un borde de tamaño k, el método MBE requiere la resolución de $2k+1$ sistemas de dimensión n. Además, esta cantidad se multiplica por k cuando dicho método se utiliza, dentro del proceso de sustitución progresiva, para refinar un par invariante de dimensión k. A pesar de la cantidad de resoluciones de sistemas que supone, tal como se verá en la sección 4.4 de resultados numéricos, el método de eliminación a bloques mixta presenta un buen comportamiento frente a otros enfoques cuando se utiliza para el refinamiento de pares invariantes de dimensión moderada. Para valores más elevados de k, puede también obtener ventaja frente a los otros enfoques cuando se añade un segundo nivel de paralelismo.

Al resolver con MBE un sistema de borde k-dimensional, la mayor parte de los cálculos que se realizan ($2k$ de los $2k+1$ sistemas generados) se llevan a cabo involucrando únicamente valores de la matriz del sistema, siendo por ello independientes del lado derecho del sistema. Por otro lado, cuando se resuelve la ecuación de corrección (4.12) mediante el método de sustitución progresiva, para refinar un par invariante de orden k, la secuencialidad en la resolución de los k sistemas que se generan es debida a la actualización de los lados derechos (4.16). Por ello, el uso del método MBE para la resolución de los sistemas (4.12) reduce considerablemente la secuencialidad inherente al método de sustitución progresiva, y obtiene beneficio de la introducción de un nuevo nivel de paralelismo en la realización de los cálculos que dependen únicamente de las matrices de los sistemas a resolver, y no de los correspondientes lados derechos. Otros enfoques, como el de la matriz explícita, pese a

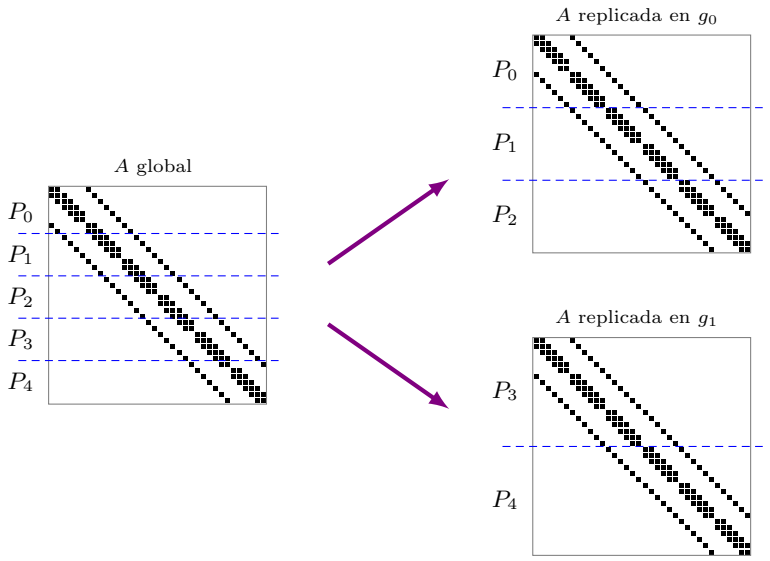

Figura 4.3: Distribución paralela de matrices replicadas en dos subcomunicadores g_0 y g_1, provenientes de un comunicador padre con 5 procesos.

no poseer tan buenas características para su paralelización, también pueden obtener beneficio de la introducción de un segundo nivel de paralelismo en la resolución de la ecuación de corrección. En concreto, si se utilizan métodos directos para la resolución de los k sistemas lineales requeridos por el procedimiento de sustitución progresiva, la limitada escalabilidad de las factorizaciones tipo LU puede ser mejorada mediante la realización en paralelo de las mismas.

4.2.4. Paralelismo jerárquico

Las consideraciones anteriores han motivado la introducción del citado nivel de paralelismo en la fase de resolución de la ecuación de corrección como una opción, tanto para los esquemas de refinamiento simple y múltiple, como para los distintos enfoques dentro de éstos. En tal caso, el conjunto de procesos MPI se divide en varios subgrupos, de forma que cada nuevo grupo es responsable de la resolución de algunos de los sistemas requeridos por el procedimiento de sustitución progresiva. Dicha división conlleva la replicación redundante en cada subgrupo de las matrices que definen el problema polinómico, y del conjunto de vectores distribuidos que forman parte del par invariante a refinar X (la matriz H está siempre replicada en cada proceso). La Figura 4.3 muestra cómo sería la distribución dentro de cada subgrupo de una matriz replicada. En este ejemplo se supone una ejecución con un grupo inicial de cinco procesos que se subdivide dando lugar a dos nuevos grupos.

En el caso de que el refinamiento se lleve a cabo utilizando el esquema simple, es decir, refinando k pares propios individualmente, se obtienen k sistemas (4.14)

(con borde de tamaño 1) cuyas resoluciones son independientes unas de otras. Esto hace que la paralelización de esta fase, en este caso, sea especialmente sencilla, ya que las resoluciones de los sistemas a realizar se pueden distribuir libremente entre los distintos subgrupos de procesos. Tal como se ha apuntado con anterioridad, la situación, en caso de realizar el refinamiento siguiendo el esquema múltiple, no es ya tan sencilla, debido a la secuencialidad introducida por el procedimiento de sustitución progresiva en la actualización de los lados derechos en (4.16). El Algoritmo 4.3 muestra los pasos que se realizan para la paralelización del proceso de sustitución progresiva mediante la división en subcomunicadores. La replicación de las matrices

Algoritmo 4.3 Segundo nivel de paralelismo en el proceso de sustitución progresiva

Entrada: Número de subcomunicadores MPI n_g

1: Crear n_g subgrupos identificados por $id_g = 0, \ldots, n_g - 1$
2: Duplicar matrices $\{A_j\}_{j=0}^{d}$ y vectores X_i en cada subgrupo
3: **para** $l = 0, \ldots, (k-1)/n_g$ **hacer**
4: **para todo** subgrupo en paralelo **hacer**
5: $p \leftarrow l * n_g + id_g$
6: **si** $p < k$ **entonces**
7: Calcular bloques de matriz asociada a (4.14) para columna p de X_i y H_i
8: Factorizar bloque principal $P(h_{pp})$
9: [**Sólo si MBE**] Resolver los $2k$ sistemas independientes del lado derecho
10: **fin si**
11: **fin para**
12: **para** $g = 0, 1, \ldots, n_g - 1$ **hacer**
13: $q \leftarrow l * n_g + g$
14: Calcular lado derecho $\begin{bmatrix} r_q \\ f_q \end{bmatrix}$ en comunicador padre y enviar al subgrupo g
15: Subgrupo g resuelve un sistema triangular con lado derecho $\begin{bmatrix} r_q \\ f_q \end{bmatrix}$
16: Subgrupo g redistribuye la columna calculada $(\Delta x_q, \Delta h_q)$ en comunicador padre
17: **fin para**
18: **fin para**

en el paso 2 se lleva a cabo según el esquema que se muestra en la Figura 4.3. Esta operación, junto con las copias de vectores paralelos en los pasos 2, 14 y 16, supone una redistribución de datos que requieren operaciones de comunicación global implicando a la totalidad de los procesos.

Los pasos 4–11 del Algoritmo 4.3, reflejan las operaciones que se realizan en paralelo por los distintos subgrupos durante la ejecución del procedimiento de sustitución progresiva. En esta fase, a cada subgrupo se le asigna el cálculo de una columna diferente de ΔX y ΔH, tras lo cual, cada uno calcula la factorización de una matriz que será el bloque diagonal principal, A, de (4.17) en caso de utilizar el enfoque de eliminación progresiva mixta, o bien la matriz con borde completa para el enfoque de la matriz explícita. Después de esto, la dependencia entre los lados derechos en los sistemas implicados, fuerza una fase secuencial para la resolución de los sistemas

correspondientes (pasos 12–17 en el Algoritmo 4.3). Después de que una columna de ΔX y ΔH sea calculada, esta es redistribuida desde el subcomunicador responsable de su cálculo al comunicador global original. La secuencia de etapas descritas es general, aunque en el caso de utilizar el enfoque MBE, las características de este método permiten, además, que parte de los cálculos que se realizan para la resolución de los sistemas asociados a la sección secuencial, puedan ser movidos a la parte concurrente, mejorando así el grado de paralelismo para este enfoque. Los cálculos susceptibles de ser movidos a la sección concurrente son los de la resolución de los $2k$ sistemas lineales con el bloque diagonal principal A, que son independientes del lado derecho $\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]$. De esta forma, sólo la resolución de un único sistema, con matriz A y lado derecho dependiente de $\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]$, se deja en la sección secuencial (paso 15).

Se hace notar que, en general, la activación del segundo nivel de paralelismo descrito no se utiliza en combinación con el enfoque del complemento de Schur, ya que dicha activación tiene relevancia principalmente en el caso de utilizar métodos directos para la resolución de los sistemas lineales. Dado que dicho enfoque utiliza métodos de Krylov que tienen buena escalabilidad, no habrá, en general, ganancia al dividir el conjunto de procesos en varios subgrupos.

Otra observación que se quiere reflejar, es que la ganancia que se puede obtener al activar este segundo nivel de paralelismo está limitada por el tamaño del par invariante que se quiere refinar (o por el número de autovalores en el caso de utilizar el esquema de refinamiento simple). A pesar de dicha limitación, tal como se verá en la sección 4.4, el nuevo nivel de paralelismo consigue mitigar la escasa escalabilidad de los *solvers* basados en métodos directos.

4.3. Solver Jacobi–Davidson

En esta sección se describe el solver polinómico de SLEPc que implementa el método Jacobi–Davidson. La implementación realizada sigue la descripción de dicho método incluida en [27] para problemas no lineales, la cual incluye un esquema de deflación adecuado para dichos problemas. En esta sección se describen los elementos principales de dicha propuesta, que son, por un lado, el método de Jacobi–Davison no lineal [91], y por otro lado, la técnica de deflación que se describe en términos de ampliación de pares invariantes [27]. El primero de estos elementos se describe brevemente en la sección 4.3.1, mientras el segundo se aborda en la sección 4.3.2, en la que se dan expresiones similares a las que aparecen en [27], pero particularizadas para el problema polinómico (1.9) y la definición (2.6) de minimalidad. Dichas expresiones son utilizadas para generar un problema proyectado polinómico, que estará entonces definido en función de la misma base que el PEP a resolver. En la sección 4.3.2 también se dan detalles de implementación relativos a minimizar el sobrecoste que supone la técnica de deflación para una ejecución en paralelo.

4.3.1. Jacobi–Davidson polinómico

El método de Jacobi–Davidson, introducido en la sección 1.2.2, obtiene aproximaciones a vectores propios en un cierto subespacio de búsqueda, que mejoran, iterativamente, a medida que dicho subespacio se amplía. Éste método tiene dos fases principales, por un lado, la fase de *extracción*, en la que se obtienen pares propios aproximados utilizando una técnica de proyección, y por otro lado, la fase de *expansión*, en la que el subespacio de búsqueda se amplía utilizando correcciones obtenidas de una iteración de Newton aplicada al problema del cálculo de valores propios. Los pasos seguidos por este método para la obtención de un par propio (aproximado) asociado a un PEP, se resumen en el Algoritmo 4.4. En éste se han incluido elementos relativos a la resolución de la ecuación de corrección (1.20), mediante un método de Krylov precondicionado, del que daremos algunos detalles a continuación. También, se ha considerado que la técnica de proyección que se utiliza es oblicua (en paso 4 del Algoritmo 4.4), y se muestra la forma en que se inicializa y expande el subespacio de test (pasos 2 y 16). Dicha estrategia de extracción es la propuesta en [27] y la que se ha utilizado en la implementación realizada.

Algoritmo 4.4 Cálculo de un par propio mediante Jacobi–Davidson polinómico

Entrada: Base Φ y matrices $\{A_i\}_{i=0}^{d} \subset \mathbb{C}^{n\times n}$ que definen el PEP, entorno de búsqueda $\sigma \in \mathbb{C}$, tolerancia tol, tamaño máximo de las bases m.
Salida: Par propio (x, λ) aproximado, con $\|P(\lambda)x\| \leq tol$.

1: $v \in \mathbb{C}^n$ aleatorio, $\dot{r} = P'(\sigma)v$
2: $V_0 = v/\|v\|$, $W_0 = \dot{r}/\|\dot{r}\|$ (*Inicialización*)
3: **para** j=1,... **hacer**
4: $(u, \mu) = (Vy, \mu)$ con $W_j^* P(\mu) V_j y = 0$ para μ próximo a σ (*Extracción*)
5: $r = P(\mu)u$
6: **si** $\|r\| \leq tol$ **entonces**
7: $(x, \lambda) = (u, \mu)$, **Parar** (*Finalización*)
8: **fin si**
9: **si** $\dim V > m$ **entonces**
10: Comprimir V, W, extraer (u, μ), fijar r (*Reinicio*)
11: **fin si**
12: Crear precondicionador K de $P(\mu)$
13: $\hat{K} \leftarrow \{K, \dot{r} = P'(\mu)u,\ z = K^{-1}\dot{r},\ \gamma = u^* z\}$ (*Precondicionador*)
14: $t = \text{solve}(P(\mu), \hat{K}, -r)$ (*Ecuación de corrección*)
15: $v = (I - VV^*)t$, $w = (I - WW^*)r$ (*Ortogonalización*)
16: $V \leftarrow [V, \frac{v}{\|v\|}]$, $W \leftarrow [W, \frac{w}{\|w\|}]$ (*Expansión de subespacios*)
17: **fin para**

La resolución de la ecuación de corrección (1.20) del método de Jacobi–Davidson se realiza de forma aproximada, utilizando por ejemplo, un método de Krylov precondicionado [76] como GMRES. Estos métodos iterativos de resolución de sistemas lineales realizan repetidamente operaciones del tipo $K^{-1}Mv$, donde v es un vector, M es la matriz que define el sistema lineal y K es un precondicionador de M, es

decir, una matriz cuya inversa es relativamente fácil de aplicar sobre un vector, y aproxima a la de M.

La ecuación (1.20) que viene dada por

$$M_p t = -r, \quad M_p = \pi_1 P(\mu)\pi_2, \quad t \perp u, \tag{4.25}$$

con

$$\pi_1 := \left(I - \frac{P'(\mu)uw^*}{w^*P'(\mu)u}\right) \text{ y } \pi_2 := (I - uu^*), \tag{4.26}$$

está definida por M_p que es una restricción de $M := P(\mu)$ a dos subespacios de codimensión uno. Si K es un precondicionador de M, entonces, su restricción $K_p = \pi_1 K \pi_2$, es el precondicionador adecuado para M_p [91]. Al aplicar un método iterativo para la resolución de (4.25) se realizan operaciones del tipo $y = {K_p}^{-1} M_p v$, las cuales se comprueba, siguiendo el razonamiento descrito en [91], que son equivalentes a $y = \hat{K} M v$, siendo

$$\hat{K} := \left(I - \frac{K^{-1}\dot{r}u^*}{u^*K^{-1}\dot{r}}\right) K^{-1}, \tag{4.27}$$

donde $r = P(\mu)u$ y $\dot{r} = P'(\mu)u$. Como consecuencia, se tiene que es equivalente utilizar un método iterativo para resolver un sistema con las restricciones M_p y K_p, que aplicar dicho método para la resolución de un sistema con la matriz M, utilizando un precondicionador cuya acción viene dada por $\hat{K}$. Éste último no se crea explícitamente sino que se calculan los elementos necesarios para su aplicación. Para ello se requiere, por un lado, la obtención de un precondicionador K para $M = P(\mu)$ (paso 12 del Algoritmo 4.4), y por otro lado los elementos que intervienen en el proyector que se aplica tras K^{-1} en (4.27), que son, $z = K^{-1}\dot{r}$ y $\gamma = u^*z$ (obtenidos en el paso 13). La operación indicada en el paso 14 del Algoritmo 4.4, se corresponde con la aplicación del método de Krylov para la resolución de la ecuación de corrección, utilizando la matriz $P(\mu)$ y el precondicionador definido por $\hat{K}$. Este precondicionador no se crea explícitamente, sino que, a nivel de implementación, se define la rutina con la que obtener el producto de $\hat{K}$ por un vector, considerando que los elementos necesarios están almacenados en una estructura de datos.

Otro elemento que aparece en el Algoritmo 4.4 es el reinicio, en el paso 10. Dicha operación se hace necesaria, al igual que en otros solvers que implementan métodos de proyección, para controlar el tamaño máximo de los subespacios de búsqueda y test. Si bien la técnica de proyección produce mejores aproximaciones a medida que dichos subespacios se amplían, el crecimiento de los mismos también conlleva el aumento del coste computacional de la ortogonalización, por un lado, y de los requerimientos de memoria por otro. Así, cuando la dimensión de dichos subespacios ha alcanzado un determinado límite, que dependerá de la aplicación y del entorno de ejecución, se reduce el tamaño de los mismos, manteniendo en éstos las direcciones asociadas a los valores de Ritz de mayor interés.

El proceso descrito en el Algoritmo 4.4 genera un par propio para un problema polinómico de valores propios. Cuando se quieren calcular más de un par propio se añade un bucle exterior a dicho proceso, que itera hasta conseguir la cantidad deseada. Sin embargo, es necesario añadir algún mecanismo para evitar obtener

pares propios previamente calculados o, al menos, una forma de detectar si ello se produce. Hay varias estrategias posibles:

- Para asegurar que se obtiene una cantidad determinada de pares propios distintos, se puede, por ejemplo, escoger (paso 4 del Algoritmo 4.4) valores de Ritz que no estén próximos (según una tolerancia) de un autovalor calculado. Este esquema puede ser suficiente cuando se resuelven problemas en los que los autovalores de interés están muy bien separados.

- Otro enfoque con el que se evita la reaparición de pares propios ya calculados es utilizar la misma estrategia que en el problema lineal, que es mantener el subespacio de búsqueda ortogonal a los autovectores obtenidos, lo que se consigue, por ejemplo, ortogonalizando cada vector con el que se amplía el subespacio de búsqueda, contra dichos autovectores. En el contexto de los problemas no lineales, este esquema plantea el inconveniente de que no permite obtención de autovalores cuyo autovector asociado sea combinación lineal de los ya calculados.

- Effenberger [27] propone una estrategia de deflación que es adecuada para ser aplicada en el contexto de los problemas de valores propios no lineales. Dicho esquema, que se describe en el siguiente apartado, es el que se ha escogido para la implementación realizada.

4.3.2. Deflación de pares invariantes

En esta sección se aborda la incorporación, en el solver que implementa el método Jacobi–Davison, de la técnica de deflación desarrollada en [27], la cual va destinada a evitar la reconvergencia hacia pares previamente calculados cuando se desea obtener más de un autovalor. Dicha técnica se formula en términos de extensión de pares invariantes asociados a problemas de valores propios no lineales. A continuación se resume el enfoque seguido en dicho trabajo, y se dan detalles de las adaptaciones realizadas, la cuales han venido motivadas, principalmente por el interés en generar, en la etapa de extracción, un problema proyectado de las mismas características que el PEP original. También se abordarán algunas cuestiones de implementación relativas a minimizar el sobrecoste que supone la técnica de deflación incorporada.

Effenberger [27] estudia la forma, una vez se tiene calculado un par invariante minimal $(X, H) \in \mathbb{C}^{n\times k} \times \mathbb{C}^{k\times k}$ de un problema no lineal (1.8), de obtener un par ampliado

$$(\tilde{X}, \tilde{H}) = \left(\begin{bmatrix} X & x \end{bmatrix}, \begin{bmatrix} H & t \\ & \lambda \end{bmatrix} \right), \tag{4.28}$$

que sea a su vez un par invariante minimal del mismo problema. En el contexto de problemas no lineales (1.8), la condición de par invariante se formula mediante una integral de contorno,

$$\mathbb{T}(X, H) := \frac{1}{2\pi i} \int_\Gamma T(z)X(zI - H)^{-1}dz = 0, \tag{4.29}$$

siendo Γ una curva en Ω que engloba los autovalores de H. Imponiendo sobre el par ampliado (4.28) las condiciones de par invariante (4.29) por un lado, y la de minimalidad (2.76) por otro, se obtiene un sistema de ecuaciones

$$\begin{bmatrix} P(\lambda) & U(\lambda) \\ A(\lambda) & B(\lambda) \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} = 0 \tag{4.30}$$

cuya solución proporciona los datos para la ampliación (4.28) deseada. El sistema (4.30) representa un nuevo problema de valores propios no lineal extendido (dimensión $n+1$).

En [27] también se proporcionan detalles relativos al modo de incluir esta técnica de deflación en el entorno del método Jacobi–Davidson no lineal. En particular, en dicho trabajo se da la expresión explícita para un posible precondicionador que puede ser utilizado en la resolución de la ecuación de corrección asociada al problema extendido (4.30),

$$\tilde{M}^{-1} = \begin{bmatrix} M^{-1} & -X(\sigma I - H)^{\dagger}(B - AX(\sigma I - H)^{\dagger})^{-1} \\ 0 & (B - AX(\sigma I - H)^{\dagger})^{-1} \end{bmatrix}, \tag{4.31}$$

donde se ha utilizado el símbolo † para denotar la matriz pseudo-inversa. El precondicionador extendido (4.31) se define a partir de un precondicionador M formado para una matriz $T(\mu)$ asociada al problema no lineal original.

Durante la fase de extracción del método de Jacobi–Davidson descrito en [27] se genera un problema proyectado no lineal que se define por medio de las expresiones obtenidas para las funciones $U(\lambda)$, $A(\lambda)$ y $B(\lambda)$. Nuestro interés en obtener un problema proyectado polinómico, que se defina en función de la misma base de polinomios que el PEP a resolver, ha motivado la obtención de expresiones específicas para problemas polinómicos del tipo (1.9).

Para obtener las fórmulas explícitas que se han utilizado en la implementación realizada, imponemos para (4.28) las condiciones de par invariante Φ-minimal, para el caso particular de un problema polinómico (1.9). Esto nos permitirá obtener un problema extendido que será también un PEP y generará a su vez un problema proyectado del mismo tipo. En primer lugar, al imponer la condición de par invariante obtenemos la expresión,

$$0 = \sum_{i=0}^{d} A_i \begin{bmatrix} X & x \end{bmatrix} \Phi_i \left(\begin{bmatrix} H & t \\ & \lambda \end{bmatrix} \right) = \sum_{i=0}^{d} A_i \begin{bmatrix} X & x \end{bmatrix} \begin{bmatrix} \Phi_i(H) & \hat{\Phi}_i(\lambda)t \\ & \Phi_i(\lambda) \end{bmatrix}, \tag{4.32}$$

donde $\hat{\Phi}_i(\lambda)t$ puede ser obtenido de forma recursiva. En efecto, utilizando la recurrencia (2.20) aplicada a $\tilde{H}$ se tiene $\Phi_{-1}(\tilde{H}) = 0$, $\Phi_0(\tilde{H}) = I$ y, para $j \geq 0$,

$$\Phi_{j+1}(\tilde{H}) = \frac{1}{\alpha_j}(\tilde{H}\,\Phi_j(\tilde{H}) - \beta_j\,\Phi_j(\tilde{H}) - \gamma_j\,\Phi_{j-1}(\tilde{H})), \tag{4.33}$$

de donde se deriva la recurrencia buscada,

$$\hat{\Phi}_{-1}(\lambda)t = 0, \quad \hat{\Phi}_0(\lambda)t = 0,$$

$$\begin{aligned}\hat{\Phi}_{j+1}(\lambda)t &= \begin{bmatrix} I_k & 0 \end{bmatrix} \Phi_{j+1}(\tilde{H})e_{k+1} = \frac{1}{\alpha_j}\left(\begin{bmatrix} H & t \end{bmatrix} \begin{bmatrix} \hat{\Phi}_j(\lambda)t \\ \Phi_j(\lambda) \end{bmatrix} - \beta_j \hat{\Phi}_j(\lambda)t - \gamma_j \hat{\Phi}_{j-1}(\lambda)t \right) \\ &= \frac{1}{\alpha_j}\left(\Phi_j(\lambda) I_k + (H - \beta_j I_k)\hat{\Phi}_j(\lambda) - \gamma_j \hat{\Phi}_{j-1}(\lambda) \right) t, \quad j \geq 0. \end{aligned} \tag{4.34}$$

Así, definiendo la función matricial

$$U(\lambda) := \sum_{i=0}^{d} A_i X \hat{\Phi}_i(\lambda), \tag{4.35}$$

la condición (4.32) da lugar a la igualdad $0 = \begin{bmatrix} \mathbb{P}(X, H) & P(\lambda)x + U(\lambda)t \end{bmatrix}$, de donde se sigue (utilizando que (X, H) es un par invariante) que la ecuación

$$P(\lambda)x + U(\lambda)t = 0, \tag{4.36}$$

determina una condición de par invariante para (4.28). Para tratar el requisito de minimalidad consideramos la Definición 2.6, que nos permitirá utilizar, también para esta condición, la misma base de polinomios con la que se define el PEP. Denotando $Z := V_d(\tilde{X}, \tilde{H})$, cada bloque Z^i, para $i = 0, 1, \ldots, d-1$, tiene la forma

$$Z^i = \begin{bmatrix} X & x \end{bmatrix} \Phi_i \left(\begin{bmatrix} H & t \\ & \lambda \end{bmatrix} \right) = \begin{bmatrix} X\, \Phi_i(H) & \hat{\Phi}_i(\lambda)t + x\, \Phi_i(\lambda) \end{bmatrix}. \tag{4.37}$$

Para asegurar que $V_d(\tilde{X}, \tilde{H})$ es de rango por columnas completo, al igual que en [27], se impone la condición de que la última columna de dicha matriz sea ortogonal al resto de columnas, $V_d(X, H)$, que son linealmente independientes por ser (X, H) minimal. Ello da lugar a la ecuación

$$0 = \sum_{i=0}^{d-1} (X\, \Phi_i(H))^* (X \hat{\Phi}_i(\lambda)t + x\, \Phi_i(\lambda)) = A(\lambda)x + B(\lambda)t, \tag{4.38}$$

donde las funciones matriciales A y B vienen dadas por

$$A(\lambda) := \sum_{i=0}^{d-1} \Phi_i(\lambda)\, \Phi_i(H)^* X^*, \quad B(\lambda) := \sum_{i=0}^{d-1} \Phi_i(H)^* X^* X \hat{\Phi}_i(\lambda). \tag{4.39}$$

Las expresiones (4.39) y (4.35) determinan el problema de valores propios extendido (4.30) con el que se amplía un par invariante Φ-minimal asociado a un PEP en la forma (1.9). Cada uno de los cuatro bloques de la matriz (4.30) es un polinomio matricial en la base de polinomios $\{\Phi_j(\lambda)\}_{j \geq 0}$. Como consecuencia, se tiene que el nuevo problema (4.30) es un PEP definido por un polinomio matricial $\tilde{P}(\lambda)$ del tipo (1.9), similar al inicial $P(\lambda)$, aunque de mayor dimensión, que puede ser resuelto por el método de Jacobi–Davidson descrito en la sección 4.3.1.

En la fase de extracción del método de Jacobi–Davidson (paso 4 del Algoritmo 4.4) aplicado sobre el problema extendido, se quiere generar un problema proyectado $W^*\tilde{P}(\lambda)V$ que se exprese como un problema polinómico en la base Φ. Para ello, es necesario conocer los coeficientes del polinomio $\tilde{P}(\lambda)$ en dicha base. Los bloques $P(\lambda)$ y $A(\lambda)$ ya están, por definición, expresados en la forma deseada. Sin embargo, las matrices $U(\lambda)$ y $B(\lambda)$ se han definido en (4.35) y (4.39), respectivamente, en función de las funciones matriciales $\hat{\Phi}_j(\lambda)$ y tenemos interés en poderlas expresar como polinomios matriciales en la base Φ_j. El siguiente resultado proporciona una recurrencia con la que poder realizar dicha transformación.

Proposición 4.3. *Dadas $\{M_j\}_{j=0}^d \subset \mathbb{C}^{m\times k}$, las matrices $\{N_j\}_{j=0}^d$ definidas por*

$$\begin{aligned} N_d &= 0, \quad N_{d-1} = \frac{1}{\alpha_{d-1}} M_d, \\ N_{j-1} &= \frac{1}{\alpha_{j-1}} \left(M_j + N_j(H - \beta_j I_k) - \gamma_{j+1} N_{j+1} \right), \quad j < d-1, \end{aligned} \tag{4.40}$$

verifican que $\sum_{j=0}^d M_j \hat{\Phi}_j(\lambda) = \sum_{j=0}^d N_j\, \Phi_j(\lambda)$, para $\hat{\Phi}_j$ definidas en (4.34).

Demostración. La prueba se realiza por inducción sobre d. Para $d = 0$ se tiene de forma inmediata que $N_0 = 0$. Para $d = 1$, haciendo uso de las relaciones (4.34) se tiene $N_1\, \Phi_1(\lambda) + N_0\, \Phi_0(\lambda) = M_1 \hat{\Phi}_1(\lambda) = (M_1/\alpha_0)\, \Phi_0(\lambda)$, de donde se sigue que $N_1 = 0$ y $N_0 = M_1/\alpha_0$. Suponemos ahora que el resultado es cierto hasta $d-1$ y comprobemos que se verifica también hasta orden d.

Definiendo las matrices $\tilde{M}_{d-1} := M_{d-1} + \frac{1}{\alpha_{d-1}} M_d (H - \beta_{d-1} I_k)$, $\tilde{M}_{d-2} := M_{d-2} - \frac{\gamma_{d-1}}{\alpha_{d-1}} M_d$ y $\tilde{M}_j := M_j$, para $j < d-2$, por hipótesis de inducción se tiene, para matrices $\{\tilde{N}_j\}_{j=0}^{d-1}$ verificando (4.40),

$$\begin{aligned} \sum_{j=0}^{d} M_j \hat{\Phi}_j(\lambda) &= M_d \hat{\Phi}_d(\lambda) + \sum_{j=0}^{d-1} M_j \hat{\Phi}_j(\lambda) = \frac{M_d}{\alpha_{d-1}}\, \Phi_{d-1}(\lambda) + \sum_{j=0}^{d-1} \tilde{M}_j \hat{\Phi}_j(\lambda) \\ &= \frac{M_d}{\alpha_{d-1}}\, \Phi_{d-1}(\lambda) + \sum_{j=0}^{d-1} \tilde{N}_j\, \Phi_j(\lambda) = N_{d-1}\, \Phi_{d-1}(\lambda) + \sum_{j=0}^{d-2} \tilde{N}_j\, \Phi_j(\lambda). \end{aligned} \tag{4.41}$$

Por otro lado,

$$\begin{aligned} \tilde{N}_{d-2} &= \frac{\tilde{M}_{d-1}}{\alpha_{d-2}} = \frac{1}{\alpha_{d-2}} (M_{d-1} + N_{d-1}(H - \beta_{d-1} I_k)) = N_{d-2}, \\ \tilde{N}_{d-3} &= \frac{1}{\alpha_{d-3}} (\tilde{M}_{d-2} + \tilde{N}_{d-2}(H - \beta_{d-2} I_k) - \gamma_{d-1} \tilde{N}_{d-1}) \\ &= \frac{1}{\alpha_{d-3}} (M_{d-2} - \frac{\gamma_{d-1} M_d}{\alpha_{d-1}} + N_{d-2}(H - \beta_{d-2} I_k)) = N_{d-3}, \\ \tilde{N}_j &= \frac{1}{\alpha_j} \left(\tilde{M}_{j+1} + \tilde{N}_{j+1}(H - \beta_{j+1} I_k) - \gamma_{j+2} \tilde{N}_{j+2} \right) = N_j, \quad j < d-3. \end{aligned} \tag{4.42}$$

Sustituyendo (4.42) en (4.41) se tiene el resultado buscado. □

Para la fase de expansión (paso 14 del Algoritmo 4.4), se utiliza la expresión obtenida para $\tilde{P}(\lambda)$ y el precondicionador extendido (4.31) propuesto en [27]. A continuación pasamos a detallar algunos aspectos de implementación referentes al modo de trabajar con dichos operadores extendidos, y a la forma en que se ha intentado reducir el sobrecoste que conlleva el aumento de la dimensión del problema extendido, en una ejecución en paralelo.

El problema extendido (4.30) no se crea explícitamente, sino que todas las operaciones en las que está involucrada la matriz extendida, o su precondicionador (4.31), se realizan de forma implícita. Los elementos que intervienen en la definición del problema extendido son: las matrices (distribuidas) del PEP inicial; los vectores paralelos X, que se almacenan en un objeto de la clase `BV`; y la matriz H, que está replicada en todos los procesos. Las bases de los subespacios de búsqueda y test se almacenan en objetos de la clase `BV` que tiene la funcionalidad necesaria para la ortogonalización. Los vectores de dichos subespacios se consideran, desde el inicio, con dimensión $n+k$, siendo k el número de pares propios a calcular. Las k últimas posiciones, las cuales se asignan a uno de los procesos, serán cero inicialmente y dejarán de serlo a medida que se produce la convergencia de los pares propios, y se amplía el problema extendido. Las principales operaciones paralelas que intervienen en la ejecución del método Jacobi–Davidson, mostrado en el Algoritmo 4.4, son:

Multiplicación. Cada vector $y = \left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right] \in \mathbb{C}^{n+k}$ que interviene en esta operación, se construye como un vector paralelo PETSc, en el que y_1 (de dimensión n) se distribuye de forma acorde a las matrices del PEP original, e y_2 está asociado a uno de los procesos. La multiplicación $y = \tilde{P}(\mu)z$ se lleva a cabo mediante los cálculos,

$$y_1 = P(\mu)z_1 + \sum_{i=0}^{d} A_i X \hat{\Phi}_i(\mu) z_2, \tag{4.43}$$

$$y_2 = \sum_{i=0}^{d-1} \Phi_i(\mu)\, \Phi_i(H)^* X^* z_1 + \sum_{i=0}^{d-1} \Phi_i(H)^* X^* X \hat{\Phi}_i(\mu) z_2, \tag{4.44}$$

en los que además de la multiplicación matriz-vector $P(\mu)z_1$ con el problema original, se realizan otros cálculos que precisan comunicación entre los procesos. La operación de multiplicación se repite muchas veces en el transcurso del método iterativo que se utiliza para la resolución de la ecuación de corrección (paso 14 del Algoritmo 4.4). Por ello, y con el fin de minimizar la comunicación extra requerida por esta operación, se utilizan estructuras distribuidas de datos (de la clase `BV`), que se actualizan cada vez que se obtiene un nuevo par propio y se amplía el par invariante (X, H), en las que se almacenan los productos $A_i X$, para $i = 0, \ldots, d$. También se mantienen almacenados en cada proceso los productos X^*X (de dimensión $k \times k$). De esta forma, los segundos sumandos que aparecen en las expresiones (4.43) y (4.44) requieren comunicación únicamente para el envío de z_2 (dimensión k) desde el proceso propietario al resto. Por su parte, en el primer sumando de (4.44) se realiza el producto

escalar de k pares de vectores, lo que se hace mediante una única operación de comunicación utilizando la funcionalidad de `BV`.

Precondicionador. Para minimizar la comunicación extra que se realiza al aplicar el precondicionador extendido (4.31) respecto del de dimensión n, cada vez que aquel se actualiza, se precalculan las matrices

$$S := (\sigma I - H)^{\dagger}, \quad J := (B - AXS)^{-1}, \tag{4.45}$$

de dimensión reducida, que quedan almacenadas en todos los procesos. Con ello, una operación con el precondicionador (4.31), que entonces toma la forma

$$\tilde{M}^{-1} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} M^{-1} z_1 - XSJ z_2 \\ J z_2 \end{bmatrix}, \tag{4.46}$$

sólo requiere comunicación extra, respecto de la operación $M^{-1} z_1$, para el envío del segmento z_2.

Ortogonalización. La ortogonalización en los espacios de búsqueda y test de dimensión extendida se realiza por medio de las operaciones disponibles en la clase `BV` y no supone comunicación adicional respecto de la de dimensión n.

Proyección. El problema proyectado que se resuelve en la fase de extracción tiene una expresión de la forma

$$W^* \tilde{P}(\mu) V = \sum_{i=0}^{d} Q_i \, \Phi_i(\mu). \tag{4.47}$$

Las matrices coeficiente Q_i asociadas a la base $\{\Phi_i\}$ se almacenan y son ampliadas a medida que los subespacios V y W se extienden. Para ello, se mantienen estructuras distribuidas de datos U_i que almacenan, para $i = 0, \ldots, d$, conjuntos de vectores de forma que $\tilde{P}(\mu)V$ puede ser expresado mediante

$$\tilde{P}(\mu) V = \sum_{i=0}^{d} U_i \, \Phi_i(\mu), \tag{4.48}$$

para todo $\mu \in \mathbb{C}$. De este modo, al añadir una columna a V y W (en la posición p), e incrementar la columna correspondiente de las estructuras U_i, las matrices Q_i en (4.47) se amplían de forma sencilla, añadiendo una fila $e_p^T Q_i = w_p^* U_i$ y una columna $Q_i e_p = W^* U_i e_p$. Para actualizar los conjuntos de vectores $\{U_i\}_{i=0}^{d}$, ampliándolos al extender la base V con una columna v, se utiliza la expresión de la operación de multiplicación

$$\tilde{P}(\lambda) \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} P(\lambda) v_1 + U(\lambda) v_2 \\ A(\lambda) v_1 + B(\lambda) v_2 \end{bmatrix} = \sum_{i=0}^{d} \begin{bmatrix} A_i v_1 + N_i v_2 \\ \Phi_i(H)^* X^* v_1 + \hat{N}_i v_2 \end{bmatrix} \Phi_i(\lambda), \tag{4.49}$$

donde las matrices N_i y $\hat{N}_i$ se obtienen por medio de la recurrencia (4.40) utilizando, respectivamente, las matrices $M_i = A_i X$ y $M_i = \Phi_i(H)^* X^* X$. Cada matriz U_i se amplía con el correspondiente coeficiente asociado a $\Phi_i(\lambda)$ en (4.49).

Se quiere puntualizar que la implementación del solver Jacobi–Davidson, disponible actualmente en SLEPc, solo funciona para PEPs definidos según la base de monomios. El tratamiento para problemas en la forma (1.9), que está pendiente de implementación, requiere una ampliación de funcionalidad en la subclase `DSPEP`, encargada de la resolución del problema polinómico proyectado. Dicha subclase sólo da cobertura, por el momento, a PEPs (de dimensión reducida) definidos mediante la base de monomios.

4.4. Evaluación computacional de los solvers basados en el método de Newton

En esta sección se presentan resultados de diversos experimentos computacionales para la evaluación de los solvers descritos en este capítulo. En especial se muestran resultados del solver de refinamiento iterativo, en los que se comparan las distintas variantes de resolución propuestas, centrándonos principalmente en comparar la escalabilidad de los diferentes enfoques considerados, aunque evaluando también el correcto funcionamiento de los mismos.

El entorno de ejecución es el descrito en la sección 2.6.1. Las ejecuciones se han llevado a cabo utilizando aritmética compleja, aunque la implementación realizada permite la ejecución en aritmética real, en el caso del refinamiento iterativo, siempre que los valores propios asociados al par invariante sean reales, y en el caso del solver Jacobi–Davidson, siempre que en la ejecución no aparezcan valores de Ritz complejos. La información de los problemas de test utilizados se resume en la Tabla 4.1. En dicha tabla se muestra información del grado del problema polinómico, la dimensión de las matrices que definen el problema, el número de valores propios (tamaño del par invariante) y del desplazamiento en torno al cual se han calculado los valores propios. El primer problema considerado surge del cálculo de la estructura electrónica de puntos cuánticos por medio de la discretización de la ecuación de Schrödinger [48]. El resto de problemas considerados pertenecen a la colección de problemas no lineales NLEVP [14]. El último test de la tabla (loaded_string) se corresponde con un problema racional que ha sido resuelto mediante interpolación polinómica utilizando polinomios de Chebyshev (ver sección 5.1), por lo que el problema polinómico resultante está expresado en función de dichos polinomios. El resto de problemas considerados están definidos utilizando la base de monomios.

Tabla 4.1: Descripción de los problemas de test utilizados para el análisis de rendimiento. Se indica el grado (*grad*) del polinomio matricial, la dimensión (*dim*) de las matrices de coeficientes, el número de autovalores solicitado (*nev*) y el desplazamiento (σ) en torno al cual se han calculado los autovalores.

nombre	grad	dim	nev	σ
qd_cylinder	3	751,900	6	0.1
qd_pyramid-186k	5	186,543	8	0.4
qd_pyramid-1.5m	5	1.5 mill	8	0.4
sleeper	2	1 mill	8	-0.9
spring	2	1 mill	5	-10
pdde_stability	2	640,000	32	-1
acoustic_wave_2d	2	999,000	16	0
loaded_string	10	1 mill	8	0

4.4.1. Resultados sobre el refinamiento iterativo de pares invariantes

Los primeros resultados que se muestran son los realizados para la evaluación de la fiabilidad y exactitud de los distintos enfoques de resolución propuestos. La calidad de las soluciones obtenidas se mide utilizando el error hacia atrás relativo definido en la sección 2.6.2. En la Tabla 4.2 se muestra el máximo de dicho error, antes y después de realizar un único paso de refinamiento iterativo sobre cada uno de los problemas test de la Tabla 4.1. Se puede observar que en todos los casos, el refinamiento iterativo produce una mejora significativa, respecto a la aproximación inicial, en la precisión de la solución que se obtiene. En la Tabla 4.2 también se incluyen, a modo de muestra, los tiempos de ejecución obtenidos al realizar una iteración de Newton sobre un par invariante aproximado inicial, calculado utilizando el método por defecto en SLEPc para la solución de problemas polinómicos (TOAR, sección 2.2.3), con una tolerancia *tol* y utilizando 8 procesos MPI (distribuidos en *sub* subcomunicadores). Las variantes del refinamiento escogidas para cada problema son las que luego se utilizan en las pruebas de rendimiento. En estos tiempos podemos observar que hay casos en los que el tiempo necesario para el cálculo de las aproximaciones iniciales es mucho mayor que el del refinamiento, mientras que en otros casos la situación es la opuesta. En general, el refinamiento es un proceso de elevado coste computacional, por lo que éste se recomienda sólo si las aproximaciones iniciales no tienen la precisión deseada. En la mayoría de los experimentos realizados se ha utilizado una tolerancia mayor de lo habitual (10^{-4}) para la resolución inicial del problema polinómico, con el fin de provocar una baja precisión en las aproximaciones iniciales y poder así apreciar la mejora obtenida con el refinamiento.

A continuación se muestran los resultados de las pruebas realizadas para la evaluación de las prestaciones paralelas de los métodos propuestos. En todos los casos se presenta un análisis de escalabilidad fuerte, es decir, el tamaño del problema no

Tabla 4.2: Resultados computacionales del refinamiento iterativo. Las aproximaciones iniciales están calculadas verificando una tolerancia *tol*. Se muestra el error hacia atrás relativo de los autovalores, antes (η_{KS}) y después (η_{NR}) del refinamiento, junto con el tiempo de ejecución (en segundos) tanto del cálculo de la aproximación inicial (t_{KS}) como del refinamiento (t_{NR}). El numero de subcomunicadores es *sub*.

nombre	tol	t_{KS}	η_{KS}	ref. método	sub	t_{NR}	η_{NR}
qd_cylinder	10^{-8}	1837	9×10^{-12}	ninguno	1	-	-
qd_cylinder	10^{-4}	1033	2×10^{-6}	multiple-schur	1	169	2×10^{-13}
qd_pyramid-186k	10^{-4}	50	3×10^{-8}	simple-schur	1	25	8×10^{-14}
qd_pyramid-1.5m	10^{-4}	591	4×10^{-6}	multiple-schur	1	152	7×10^{-12}
sleeper	10^{-6}	42	8×10^{-10}	multiple-mbe	8	29	5×10^{-17}
pdde_stability	10^{-4}	185	1×10^{-6}	simple-mbe	1	277	3×10^{-13}
acoustic_wave_2d	10^{-6}	111	1×10^{-8}	multiple-exp-lu	1	837	3×10^{-14}
loaded_string	10^{-4}	40	2×10^{-6}	simple-mbe	8	36	6×10^{-17}

varía al aumentar el número de procesos.

Los problemas de test provenientes de la simulación de puntos cuánticos son: el test qd_cylinder que se corresponde con un problema polinómico de grado 3, generado a partir de un punto cuántico cilíndrico, discretizado mediante volúmenes finitos sobre una malla uniforme, y el test qd_pyramid, el cual se corresponde con un problema polinómico de grado 5, generado a partir de un punto cuántico piramidal. En pruebas realizadas con estos problemas, se ha comprobado que la resolución de sistemas lineales utilizando métodos de Krylov iterativos precondicionados, muestran, en general, un buen comportamiento, por lo que se ha escogido el método Bi-CGStab (proporcionado por PETSc), con un precondicionador de Jacobi a bloques (que utiliza una LU incompleta con llenado cero en cada subdominio).

La Figura 4.4 muestra el tiempo de ejecución en paralelo, para un número creciente de procesos, del test qd_cylinder. En ella se comparan tiempos totales de cálculo para dos situaciones en las que se calculan 6 autovalores con un solver polinómico (TOAR, sección 2.2.3). En el primer caso los valores son inicialmente calculados con una tolerancia de 10^{-8} y no se realiza refinamiento alguno sobre ellos. En el segundo caso los autovalores iniciales se calculan con una tolerancia de $tol = 10^{-4}$, y se realiza un paso de refinamiento iterativo para refinar el par invariante asociado (esquema múltiple), utilizando el enfoque del complemento de Schur. Tal como puede verse también en la Tabla 4.2, en este caso, el solver polinómico necesita mucho tiempo (muchas iteraciones) para alcanzar la tolerancia de 10^{-8}. Por ello, en este caso, es preferible realizar el cálculo de una aproximación inicial con poca precisión y luego aplicar un paso de refinamiento de Newton para alcanzar la precisión deseada. En la Figura 4.4 se comprueba que los dos casos analizados presentan similar escalabilidad.

En los resultados que se presentan a continuación se compara la escalabilidad de los distintos enfoques de refinamiento, cuando se resuelve el problema qd_pyramid.

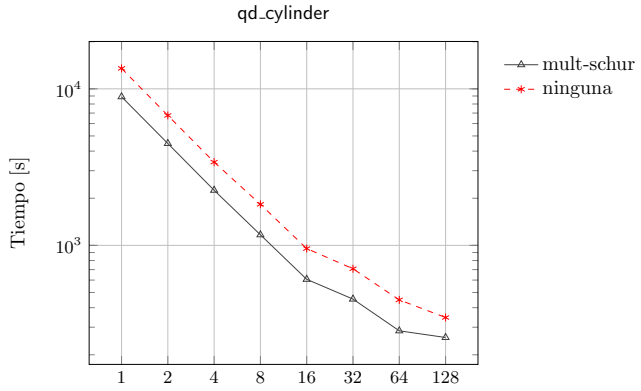

Figura 4.4: Escalabilidad paralela (hasta 128 procesos) en el cálculo de 6 autovalores para el problema qd_cylinder con y sin refinamiento.

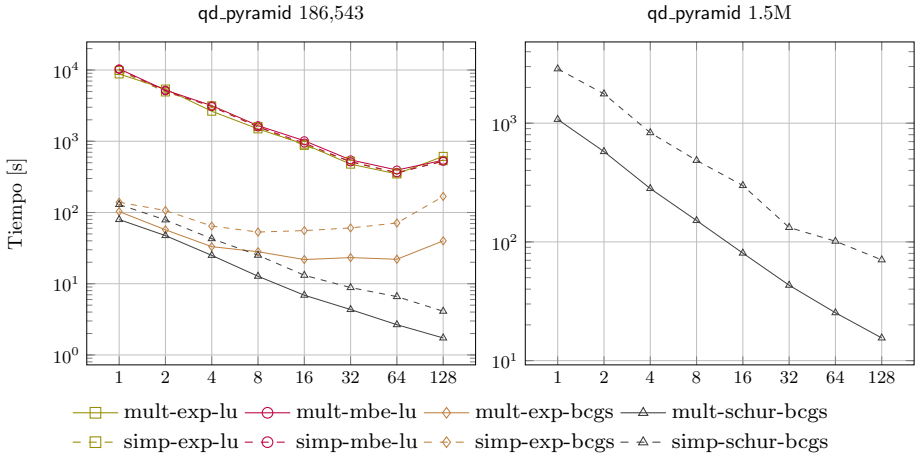

Figura 4.5: Escalabilidad paralela (hasta 128 procesos) de diferentes enfoques de refinamiento iterativo para el test qd_pyramid de dimensión 186,543 (izquierda) y 1.5 millones (derecha). Leyenda: *mult*/*simp* hace referencia al esquema (múltiple o simple); sistemas lineales via matriz explícita (*exp*), MBE (*mbe*) o complemento de Schur (*schur*) con solver directo (*lu*) o iterativo (*bcgs*).

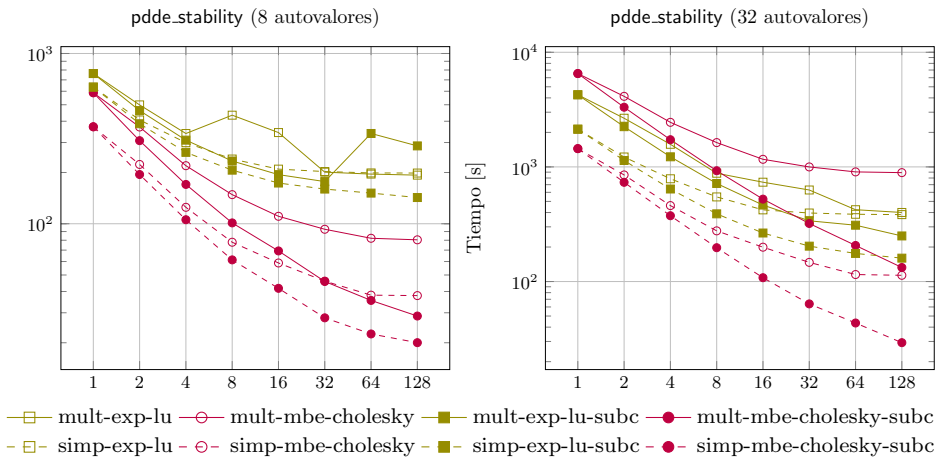

Figura 4.6: Escalabilidad paralela (hasta 128 procesos) de las diferentes variantes de refinamiento iterativo al refinar 8 (izquierda) o 32 (derecha) autovalores del problema pdde_stability. Leyenda: *mult*/*simp* hace referencia al esquema (múltiple o simple); sistemas lineales via matriz explícita (*exp*) o MBE (*mbe*), con descomposición LU o Cholesky; *subc* indica que se ha utilizado más de un subcomunicador.

La Figura 4.5 muestra tiempos de ejecución para dos tamaños de problema. En el panel izquierdo, se puede observar que para este problema (de tamaño moderado) todas las ejecuciones que utilizan un método directo para la resolución de los sistemas lineales se solapan y no presentan un buen comportamiento paralelo; esto es debido a que el tiempo de ejecución está dominado por el de la factorización. Respecto a las alternativas que realizan las resoluciones de sistemas lineales mediante un método iterativo, las basadas en el enfoque del complemento de Schur escalan considerablemente mejor que las basadas en el enfoque de la matriz explícita, en las que el número de iteraciones requeridas por el solver iterativo aumenta con el número de procesos. La buena escalabilidad que presenta para el test qd_pyramid la variante que utiliza el complemento de Schur puede apreciarse mejor en el problema de 1.5 millones, el cual se muestra en el panel derecho de la Figura 4.5.

Para la evaluación de rendimiento del enfoque basado en el método de eliminación a bloques mixta se muestran, en la Figura 4.6, ejecuciones con el problema pdde_stability en las que se compara dicha estrategia con respecto a la de la matriz explícita. Para este problema no se muestran resultados con la variante del complemento de Schur porque no se ha encontrado ningún solver iterativo que converja para este problema. Las ejecuciones realizadas con el esquema simple se han representado mediante líneas discontinuas, mientras que las que se corresponden con el esquema múltiple se representan por lineas continuas. Las ejecuciones con sólo un nivel de paralelismo y un único comunicador se dibujan utilizando círculos o cuadrados huecos, mientras que las ejecuciones con varios subcomunicadores se dibujan

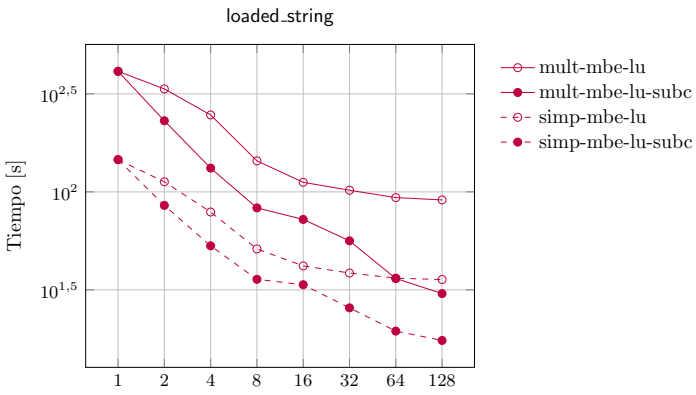

Figura 4.7: Escalabilidad paralela (hasta 128 procesos) de diferentes enfoques de refinamiento iterativo operando en el test loaded_string. Leyenda: *mult*/*simp* hace referencia al esquema (múltiple o simple); sistemas lineales vía eliminación a bloques mixta (*mbe*) con descomposición LU; *subc* indica que se ha utilizado más de un subcomunicador.

con puntos o cuadrados densos. Los tiempos representados en este último caso se han obtenido ejecutando con p procesos y $min(nev, p)$ subcomunicadores (es decir, se utilizan subcomunicadores de un único proceso hasta que el número de procesos es mayor que la cantidad de autovalores a refinar).

En el panel izquierdo de la Figura 4.6 se puede comprobar que al refinar 8 autovalores, la variante MBE es siempre más rápida que la de la matriz explícita para cualquier número de procesos (y cualquiera de los esquemas). Sin embargo, y coincidiendo con lo señalado en la sección 4.2, en el panel derecho se puede apreciar cómo el rendimiento del enfoque basado en MBE se ve penalizado al incrementar el tamaño del par invariante a refinar. En dicha ejecución, en la que se refinan 32 autovalores se observa que el enfoque MBE junto con el esquema múltiple es más lento que la variante de la matriz explícita. También se puede comprobar que si se opta por el esquema simple, el rendimiento en el enfoque basado en MBE no se ve afectado por el aumento de autovalores y mantiene el tiempo de ejecución por debajo de la variante de la matriz explícita.

En la Figura 4.6 se muestra también el comportamiento de los enfoques basados en MBE o matriz explícita al activar el segundo nivel de paralelismo. Se comprueba, que aunque la variante basada en MBE puede ser, en algunos casos, más lenta, es posible mejorar su escalabilidad mediante la utilización de subcomunicadores según el Algoritmo 4.3. En los dos paneles de la figura se puede ver que la utilización de subcomunicadores mejora significativamente la escalabilidad de MBE, con una reducción de tiempo casi proporcional al número de procesos.

Por último se muestran resultados para el problema loaded_string, cuyo polinomio asociado tiene grado 10 y está representado mediante la base de polinomios de Chebyshev. Al igual que en el caso anterior, este ejemplo también requiere la utili-

zación de métodos directos para la resolución de los sistemas lineales (los métodos iterativos y precondicionadores probados tuvieron problemas de convergencia). En la Figura 4.7 se muestra el tiempo de ejecución para la variante MBE, tanto para el esquema de refinamiento múltiple (líneas continuas) como para el simple (líneas discontinuas). También se muestran sendas ejecuciones para los esquemas múltiple y simple en las que se ha activado el segundo nivel de paralelismo, que utilizará un máximo de 8 subcomunicadores (número de autovalores que se refinan). Se observa una evidencia clara de que la utilización de subcomunicadores mejora el grado de escalabilidad en cualquiera de los dos esquemas, aunque se comprueba que el número de autovalores que se refinan limita la mejora de escalabilidad que cabe esperar con el nivel de paralelismo adicional.

4.4.2. Resultados para el solver Jacobi–Davidson

En esta sección se muestran resultados de varias pruebas realizadas para la evaluación del solver polinómico JD, que implementa el método Jacobi–Davidson. Dicho solver, para el que aún está prevista la incorporación de algunas extensiones que se encuentran todavía en fase de estudio e implementación, no tiene, por el momento, distintas opciones de resolución que puedan ser comparadas, por lo que en su evaluación nos limitamos a comprobar su correcto funcionamiento y escalabilidad.

En la Tabla 4.3 se muestran diversas ejecuciones con algunos de los problemas de la Tabla 4.1, para los que se ha variado alguno de los parámetros de ejecución (reflejados en la Tabla 4.3). En dicha tabla se proporciona información del número de autovalores solicitados y del máximo error hacia atrás $\eta_P(\lambda)$ obtenido, que nos permite comprobar que el solver JD funciona correctamente en cuanto a la exactitud de los resultados que proporciona. Cada una de las ejecuciones con el solver JD se acompaña de otra, con similares parámetros de ejecución, pero utilizando el solver polinómico TOAR. Para la resolución de los sistemas lineales, en JD siempre se ha utilizado el método Bi-CGStab conjuntamente con un precondicionador de Jacobi a bloques, utilizando una tolerancia relativa igual a 10^{-4}. En el caso de TOAR, la tolerancia utilizada es 10^{-8} y a veces ha requerido la utilización de un método directo (indicado en la Tabla 4.3). Comparando ambos solvers podemos comprobar que, en general, el método TOAR es más eficiente, especialmente si se solicita un número elevado de autovalores. Aún así, también es posible encontrar algunos problemas o condiciones de ejecución para los que JD presenta un mejor comportamiento.

Para las pruebas de rendimiento paralelo, se han escogido estos casos. En la Figura 4.8 se muestra una comparativa con el solver TOAR para dos casos de prueba, los tests qd_cylinder y qd_pyramid, en la que se incluyen los tiempos requeridos para la resolución de los sistemas lineales en el método de Jacobi–Davidson. En estos problemas se calculan 4 autovalores cercanos al desplazamiento, y se utiliza un tamaño máximo de base igual a 30 vectores (tanto para TOAR como JD). En ambos casos de prueba podemos observar el buen comportamiento paralelo que presenta el solver JD, el cual mantiene la buena escalabilidad de la resolución de los sistemas lineales.

Tabla 4.3: Resultados de varias ejecuciones con 8 procesos MPI con los solvers polinómicos Jacobi–Davidson (JD) y TOAR. Se muestra la cantidad de autovalores solicitados *nev*, el precondicionador utilizado *PC* (LU método directo), el tamaño máximo del subespacio de proyección *ncv*, el número de iteraciones del solver *iter*, el error hacia atrás máximo $\eta_P(\lambda)$ y el tiempo de ejecución en segundos. Las aproximaciones se han calculado con una tolerancia de 10^{-8}, la tolerancia utilizada para la resolución de los sistemas lineales es 10^{-4} en Jacobi–Davidson y 10^{-8} en TOAR.

problema	nev	método	PC	ncv	iter	$\eta_P(\lambda)$	tiempo
qd_cylinder	4	JD	BJacobi	30	84	9×10^{-13}	385
	4	TOAR	BJacobi	30	6	2×10^{-10}	941
qd_pyramid	4	JD	BJacobi	30	32	8×10^{-11}	416
	4	TOAR	BJacobi	30	2	3×10^{-11}	597
	8	JD	BJacobi	40	108	7×10^{-11}	3537
	8	TOAR	BJacobi	40	4	1×10^{-10}	1248
sleeper	20	JD	BJacobi	40	276	2×10^{-10}	2140
	20	TOAR	LU	40	4	9×10^{-12}	101
spring	5	JD	BJacobi	20	28	1×10^{-10}	70
	5	TOAR	LU	20	2	1×10^{-13}	43

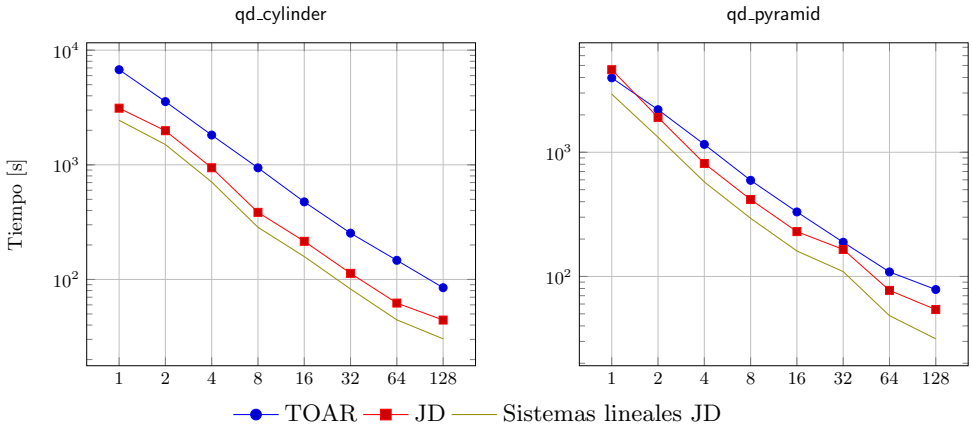

Figura 4.8: Tiempo de ejecución (en segundos) con los tests qd_cylinder (izquierda) y qd_pyramid (derecha), mostrados en la Tabla 4.3, para los métodos TOAR y Jacobi–Davidson, utilizando hasta 128 procesos MPI y solicitando 4 autovalores. Se incluye el tiempo requerido para la resolución de los sistemas lineales en JD.

4.5. Conclusiones

En este capítulo se han descrito dos solvers polinómicos, que han sido agrupados por el motivo de que ambos se basan en la iteración de Newton y plantean la ejecución en términos de pares invariantes. Ambos solvers han sido incorporados en SLEPc y están disponibles desde la última versión de ésta biblioteca (versión 3.7).

El primero de los dos solvers considerados, que es el de refinamiento de Newton, se utiliza como complemento a otros solvers polinómicos disponibles en SLEPc, especialmente los basados en linealización. Éste está disponible para ser utilizado como un postprocesado que permite realizar el refinamiento de los pares invariantes calculados, cuando se desea mejorar la exactitud de los mismos. La utilización iterativa de dicho refinamiento permite obtener muy buenas aproximaciones (a menudo es suficiente con una sola iteración). En este solver se han implementado dos variantes, una que opera directamente sobre el par invariante (o variante *múltiple*) y otra que realiza el refinamiento de los pares propios individualmente (variante *simple*). Para aliviar el elevado sobrecoste que supone su utilización, en este solver se han incluido varias opciones de resolución para la parte más costosa de su ejecución, que es la resolución de los sistemas lineales generados por la iteración de Newton. Dichas opciones son, utilización del *complemento de Schur*, formación *explícita* del sistema y, por último, utilización del método de eliminación a bloques mixta, *MBE*. Para esta dos últimas se ha incluido también la posibilidad de activar un nivel de paralelismo adicional (división en *subcomunicadores*).

Las pruebas realizadas han permitido, por un lado, comprobar el buen comportamiento del solver en cuanto a la exactitud de los resultados que proporciona, y por otro, comparar los distintos rendimientos en paralelo que se obtienen con los respectivos esquemas de resolución. Ello nos ha permitido establecer que el esquema basado en el complemento de Schur escala muy bien y debería ser la opción utilizada cuando es posible la utilización de un método iterativo. En el caso de que se requiera un método directo de resolución, el esquema basado en MBE presenta una escalabilidad aceptable, tanto para las opciones múltiple como simple, la cual mejora considerablemente al incluir la opción de subdivisión en subcomunicadores.

El segundo solver considerado en este capítulo implementa el método Jacobi–Davidson polinómico. Con su incorporación en SLEPc se incide el propósito de disponer de una variedad de solvers para ser utilizados en problemas con características diversas. El solver desarrollado incluye un esquema de deflación efectiva que permite la obtención de varios pares propios en el contexto de problemas no lineales de valores propios. La implementación de dicho esquema se ha realizado de forma que el problema polinómico proyectado que genera se expresa en la misma base de polinomios que el PEP original. Los resultados obtenidos han permitido comprobar, también en este caso, el buen comportamiento del solver, en cuanto a la exactitud de los resultados que proporciona y la escalabilidad de su ejecución en paralelo.

Capítulo 5

Extensiones para la resolución de problemas no lineales de valores propios

Este capítulo se centra en el problema no lineal de valores propios NEP (1.8), por el que existe un creciente interés en los últimos años. En un trabajo muy reciente de Güttel y Tisseur [35] se ofrece una visión general sobre el conocimiento actual en dicho campo. En esta tesis se aborda, aunque de una forma parcial, la resolución numérica de este problema, llevando a cabo varios desarrollos software que han sido incluidos en la clase `NEP` de SLEPc, la cual da soporte a la resolución del problema no lineal de valores propios. Los métodos implementados en dichos desarrollos están, de alguna forma, relacionados con algunos de los métodos ya descritos en esta tesis, en el contexto de problemas polinómicos de valores propios.

Se presentan dos solvers para la resolución de problemas no lineales que están basados en interpolación. El primero de ellos, que se describe en la sección 5.1, realiza una interpolación polinómica de Chebyshev, en la forma descrita en [28], del operador no lineal T, y resuelve el PEP resultante. El polinomio asociado puede tener un grado elevado por lo que, para estos problemas, tiene especial relevancia la técnica de representar la base de Krylov de forma eficiente (ver TOAR, sección 2.2.3). El segundo de los solvers que se describen en este capítulo (en la sección 5.2), implementa el método NLEIGS [36], en el que se realiza una interpolación racional del operador T, que se resuelve por medio de un problema lineal que tiene los mismos valores propios que la aproximación racional (una linealización). Al igual que en el problema polinómico, la dimensión de la linealización crece con el número de términos racionales en la aproximación. Este hecho ha motivado el desarrollo, con base en el enfoque seguido en el solver TOAR, de un esquema eficiente de resolución para la linealización del método NLEIGS.

En métodos como los que se exponen en este capítulo, en los que se resuelve un problema que aproxima al original, puede suceder (por ejemplo si la aproximación

no es suficientemente buena), que los autovalores calculados cumplan con un buen residuo para la aproximación, pero no para al problema original. Especialmente para estos casos, se ha incluido la posibilidad de realizar el refinamiento iterativo con el que mejorar las aproximaciones de pares propios obtenidas. Dicho refinamiento, que se expone en la sección 5.3 sigue el esquema utilizado en [51], y es similar al expuesto en la sección 4.1 para problemas polinómicos.

5.1. Resolución vía interpolación polinómica

El solver basado en interpolación polinómica, que se describe en esta sección, permite la obtención de los autovalores del problema no lineal definido por una función analítica $T : \Omega \subset \mathbb{C} \to \mathbb{C}^{n\times n}$, próximos a un intervalo real acotado $[a, b] \subset \Omega$. Éste es un solver bastante básico, que no incluye el desarrollo de características especiales. Se basa en aproximar el operador no lineal por medio de un polinomio de interpolación y resolver el problema polinómico asociado, aprovechando la propiedad de la clase `PEP`, de poder definir el problema polinómico utilizando una base genérica distinta de la de monomios.

Para su implementación se ha seguido el enfoque descrito en [28], donde se propone la utilización de una interpolación polinómica del operador T utilizando los *nodos de Chebyshev*. Se considera que la función T está definida en el intervalo $[-1, 1]$, ya que otro caso puede ser reducido a éste mediante el cambio de variable

$$\lambda = \frac{(b-a)}{2}\theta + \frac{(b+a)}{2}, \tag{5.1}$$

considerando el problema $\tilde{T}(\theta) := T(\lambda)$. Los nodos de Chebyshev $\{\nu_j\}_{j=0}^{d} \subset [-1, 1]$, se definen por la expresión

$$\nu_j := \cos\left(\left(j + \frac{1}{2}\right)\frac{\pi}{d+1}\right) \text{ para } j = 0, \ldots, d. \tag{5.2}$$

Éstos se corresponden con las raíces del polinomio de Chebyshev de grado $d+1$,

$$\tau_{d+1}(\lambda) = \cos\left((d+1)\cos^{-1}(\lambda)\right), \text{ para } \lambda \in [-1, 1]. \tag{5.3}$$

El polinomio de grado d que interpola T en los nodos de Chebyshev se expresa, según la base de polinomios de Chebyshev (de orden d), como

$$P_d(\lambda) = \frac{a_0}{2}\tau_0(\lambda) + \sum_{k=1}^{d} a_k\tau_k(\lambda), \tag{5.4}$$

con $a_k = \frac{2}{d+1}\sum_{i=0}^{d} T(\nu_i)\tau_k(\nu_i)$, para ν_i definidos en (5.2), o equivalentemente,

$$a_k = \frac{2}{d+1}\sum_{i=0}^{d} T(\cos\omega_i)\cos(k\omega_i), \quad \omega_i = \left(i + \frac{1}{2}\right)\frac{\pi}{d+1}. \tag{5.5}$$

En [28] se prueba que el error de aproximación $\|T(\lambda)-P_d(\lambda)\|_F$, que se obtiene al aproximar T por el polinomio (5.4), decrece de forma exponencial, y uniformemente en $[-1,1]$, al aumentar el grado de la interpolación. También se prueba que, para d suficientemente grande, los autovalores que se obtengan de resolver el PEP asociado a P serán buenas aproximaciones a los autovalores de T siempre que éstos estén cercanos al eje real.

En la implementación realizada, de la que se presentan resultados en la sección 5.4, se utilizan las expresiones (5.4) y (5.5) para construir el polinomio P que aproxima el operador no lineal T. El grado de la interpolación puede ser fijado mediante la interfaz de usuario, aunque también se está estudiando la posibilidad de incluir una opción con la que éste sea determinado de forma automática, en el caso de que el problema no lineal esté definido mediante un operador de la forma

$$T(\lambda) = f_0(\lambda)A_0 + \cdots + f_t(\lambda)A_t, \tag{5.6}$$

con matrices $\{A_i\}_{i=0}^t$ constantes. En dicho caso, la norma del coeficiente a_d dado en (5.5) puede ser estimada mediante la cota $\|a_d\|_\infty \leq \sum_{j=0}^t |a_d^j| \|A_j\|_\infty$, donde a_d^j representa el correspondiente coeficiente de interpolación de la función f_j, para $j = 1, \ldots, t$. La idea es determinar una expresión que utilice las estimaciones $\|a_d\|_\infty$, como por ejemplo $\|a_d\|_\infty / \|a_0\|_\infty$, y determinar el grado del polinomio de interpolación como el valor d mínimo necesario para que dicha expresión esté por debajo de una determinada tolerancia (por ejemplo, la utilizada como cota del residuo para los pares propios del NEP a resolver). Por el momento no se dispone de una expresión adecuada que nos haya permitido la obtención del grado del polinomio interpolador en todos los casos considerados.

Se quiere hace notar, que aunque mediante una adecuada parametrización, se podría extender el enfoque de interpolación descrito, a curvas diferenciables en el plano complejo (no solo un intervalo del eje real) dicha opción no ha sido incorporada en el solver implementado.

5.2. Interpolación racional: Método NLEIGS

El esquema de resolución de un NEP mediante interpolación polinómica presenta, en general, un buen comportamiento si la función asociada al problema no lineal es entera, o incluso sin serlo, si los puntos de interpolación están suficientemente alejados de las singularidades de ésta [28, 36]. Cuando no se dan dichas condiciones, el error en la aproximación supone una limitación en la exactitud de los autovalores que se pueden obtener mediante dicho enfoque. Para estos casos, es preferible plantear la aproximación de la función T por medio de una interpolación racional, como la propuesta en [36]. En dicho trabajo se aborda la resolución de NEPs utilizando dos elementos fundamentales; por un lado, una interpolación racional de la función no lineal, que se construye teniendo en cuenta el conjunto de singularidades de la misma; y por otro lado, una linealización del problema racional, que se resuelve mediante

un método de Krylov para obtener las aproximaciones a pares propios del problema no lineal.

En esta sección se describe el método NLEIGS desarrollado por Güttel y coautores en [36], el cual ha sido implementado en el solver para problemas no lineales que lleva el mismo nombre. En la implementación realizada, se ha incluido un esquema de tratamiento eficiente de la memoria en la resolución de la linealización, al estilo del utilizado en el método TOAR polinómico, descrita en la sección 2.2.3. En el apartado 5.2.1 se dan detalles de las relaciones, derivadas a partir de la linealización, que han sido utilizadas para incluir dicho esquema.

El método NLEIGS plantea la obtención de autovalores en un conjunto compacto $\Sigma \subset \mathbb{C}$, denominado *conjunto objetivo*, en el que la función $T(\lambda)$ es analítica. Sigue el enfoque de aproximar T mediante una función racional con polos situados en el *conjunto de singularidades* de T, denotado por Ξ. Dicho método utiliza las funciones racionales

$$b_j(\lambda) := \frac{1}{\beta_0} \prod_{k=1}^{j} \frac{\lambda - \sigma_{k-1}}{\beta_k(1 - \lambda/\xi_k)}, \quad j = 0, 1, \ldots, \tag{5.7}$$

para construir una función racional con polos en $\xi_k \in \Xi$,

$$Q_d(\lambda) := \sum_{j=0}^{d} b_j(\lambda) D_j, \tag{5.8}$$

que interpole T en los nodos $\sigma_k \in \partial\Sigma$ (contorno de Σ). Los números β_k en (5.7) son factores de escalado que se obtienen de forma que $\max_{\lambda \in \partial\Sigma} |b_k(\lambda)| = 1$. Las funciones $b_j(\lambda)$, definidas en (5.7), satisfacen una relación de recurrencia dada por

$$\begin{aligned} b_0(\lambda) &= 1, \\ b_j(\lambda) &= \frac{\lambda - \sigma_{j-1}}{\beta_j(1 - \lambda/\xi_j)} b_{j-1}(\lambda), \quad j = 1, 2, \ldots. \end{aligned} \tag{5.9}$$

Si no se utiliza información sobre las singularidades (tomando $\xi_k = \infty$), dichas funciones se corresponden con la base de polinomios de Newton asociada a los nodos de interpolación σ_k. Las matrices D_j de (5.8) son *diferencias divididas racionales* que se obtienen, al imponer las condiciones de interpolación $Q_j(\sigma_j) = T(\sigma_j)$, mediante la recurrencia

$$\begin{aligned} D_0 &= \beta_0 T(\sigma_0), \\ D_j &= \frac{T(\sigma_j) - Q_{j-1}(\sigma_j)}{b_j(\sigma_j)}, \quad j = 1, 2, \ldots. \end{aligned} \tag{5.10}$$

Para los casos en que la función T esté expresada en la forma (5.6), denotando por d_i^j, para $j = 0, \ldots, t$, la diferencia dividida racional j-ésima correspondiente a la función f_i, se verifica que

$$D_j = \sum_{i=0}^{t} d_i^j A_i, \quad j \geq 0. \tag{5.11}$$

Tal como se prueba en [36, Teorema 2.1], las diferencias divididas racionales de las funciones f_i, utilizadas en (5.11), se pueden obtener de forma estable, utilizando funciones de matrices, mediante la expresión

$$d_i^j = \beta_0 e_j^T f_i(H_{m+1}K_{m+1}^{-1})e_1, \tag{5.12}$$

donde las matrices H_{m+1} y K_{m+1} vienen dadas por

$$H_{m+1} := \begin{bmatrix} \sigma_0 & & & \\ \beta_1 & \sigma_1 & & \\ & \ddots & \ddots & \\ & & \beta_m & \sigma_m \end{bmatrix}, \quad K_{m+1} := \begin{bmatrix} 1 & & & \\ \beta_1/\xi_1 & 1 & & \\ & \ddots & \ddots & \\ & & \beta_m/\xi_m & 1 \end{bmatrix}. \tag{5.13}$$

Respecto de la implementación del método incluida en SLEPc, si el problema ha sido definido en la forma (5.6), se utiliza la expresión (5.12) para el cálculo de las diferencias divididas racionales, siempre que las funciones f_i en las que el objeto `FN` asociado soporte su evaluación a nivel de matrices (ver sección 1.4.2). En caso contrario se calculan según (5.10). Se hace notar que, en la implementación realizada, las diferencias divididas matriciales no se crean explícitamente en el caso de problemas del tipo (5.6), sino que se opera con ellas implícitamente, tal como se explica más adelante, accediendo únicamente a las matrices que intervienen en la definición del operador T. Para determinar el grado d en la interpolación (5.8), se evalúa la norma de las diferencias divididas racionales, a medida que éstas se generan, hasta que se verifica $\|D_d\|/\|D_0\| < tol$, para una determinada tolerancia tol. En problemas del tipo (5.6), para los que las diferencias divididas no se generan explícitamente, se utiliza la estimación $\delta^d := \text{máx}\{|d_i^d| : i = 0 \dots t\}$, propuesta en [36], que hace uso de las diferencias divididas de las funciones f_i que definen el NEP.

Los nodos de interpolación y los polos que determinan la aproximación $Q_d(\lambda)$ dada en (5.8), se calculan según [36, Sección 4], como una secuencia de puntos de *Leja–Bagby* para $(\partial\Sigma, \Xi)$, los cuales se obtienen recursivamente como

$$\max_{z\in\partial\Sigma} |s_j(z)| = |s_j(\sigma_{j+1})| \quad \text{y} \quad \min_{z\in\Xi} |s_j(z)| = |s_j(\xi_{j+1})|, \tag{5.14}$$

para

$$s_j(\lambda) := \frac{\prod_{k=0}^{j}(\lambda - \sigma_k)}{\prod_{k=1}^{j}(1 - \lambda/\xi_k)}, \quad j = 0, 1, \dots. \tag{5.15}$$

Para la evaluación en (5.14) se utilizan puntos $\{\sigma_j\}$ y $\{\xi_j\}$ obtenidos mediante discretización de los conjuntos $\partial\Sigma$ y Ξ, respectivamente.

En el solver implementado, se realiza la discretización automática del conjunto objetivo, el cual se define mediante la interfaz del objeto región `RG` (ver Sección 1.4.2). Sin embargo, el conjunto de singularidades, en general, debe ser definido por el usuario como una función que proporciona una secuencia de puntos que determina su discretización. Si no se proporciona dicha función, el solver considera que las singularidades toman un valor *infinito*, y el método deriva en una interpolación

polinómica. No obstante, para problemas definidos en la forma (5.6), en los que las funciones f_i sean racionales, definidas como una combinación sencilla (suma o producto) de cocientes de polinomios, se ha incluido la posibilidad de cálculo automático del conjunto de singularidades. Los polos de un cociente de polinomios $p(\lambda)/q(\lambda)$ se obtienen calculando las raíces del denominador $q(\lambda) = c_0 + c_1\lambda + \cdots + c_l\lambda^l$, mediante la obtención de los valores propios de la matriz compañera asociada a dicho polinomio,

$$\begin{bmatrix} 0 & & & c_0/c_l \\ 1 & \ddots & & c_1/c_l \\ & \ddots & \ddots & \vdots \\ & & 1 & c_{l-1}/c_l \end{bmatrix},$$

cuyo polinomio característico coincide con q.

Para la obtención de los autovalores de la aproximación racional (5.8), en [36] se propone la formación y resolución de un problema lineal,

$$Ay = \lambda By, \tag{5.16}$$

que tiene los mismos autovalores que el problema racional, y cuyos autovectores tienen la forma

$$y = \begin{bmatrix} b_0(\lambda)x \\ \vdots \\ b_d(\lambda)x \end{bmatrix}, \tag{5.17}$$

siendo x un autovector de $Q(\lambda)x = 0$. Las matrices A y B de la linealización (5.16), una vez se ha realizado la simplificación propuesta en [36] de fijar el último polo $\xi_d = \infty$, vienen dadas por

$$\underbrace{\begin{bmatrix} D_0 & D_1 & \dots & D_{d-2} & (D_{d-1} - \frac{\sigma_{d-1}}{\beta_d}D_d) \\ \sigma_0 I & \beta_1 I & & & \\ & \ddots & \ddots & & \\ & & \ddots & \beta_{d-2}I & \\ & & & \sigma_{d-2}I & \beta_{d-1}I \end{bmatrix}}_{A}, \quad \underbrace{\begin{bmatrix} 0 & 0 & \dots & 0 & -\frac{D_d}{\beta_d} \\ I & \frac{\beta_1}{\xi_1}I & & & \\ & \ddots & \ddots & & \\ & & \ddots & \frac{\beta_{d-2}}{\xi_{d-2}}I & \\ & & & I & \frac{\beta_{d-1}}{\xi_{d-1}}I \end{bmatrix}}_{B}. \tag{5.18}$$

Para la resolución de dicha linealización, en [36] se proponen tres variantes del método *rational Krylov* [75]. En una de ellas la aproximación y la linealización se construyen dinámicamente a medida que se expande el subespacio de Krylov (variante dinámica), en otra se crea la linealización en un paso inicial y posteriormente se resuelve (variante estática), la tercera es una combinación de las anteriores.

En la implementación realizada en SLEPc se sigue un enfoque estático en el que, en un paso inicial, se crea tanto la aproximación como la linealización de ésta, la cual se resuelve mediante una variante del método de Krylov–Schur, que es similar a

la utilizada en el solver TOAR polinómico. En el siguiente apartado se dan detalles de la variante utilizada.

5.2.1. Variante tipo TOAR para NLEIGS

De forma similar a lo que se ha visto en la Sección 2.2, que trata la resolución vía linealización del problema polinómico, también en este caso ocurre que al utilizar el método de Arnoldi para la resolución del problema lineal (5.16), la estructura a bloques de las matrices A y B produce relaciones de dependencia entre los d bloques de los vectores de Krylov (de dimensión dn) generados por dicho método. Esto hace que sea posible definir una base de vectores de dimensión n, a partir de la cual obtener los distintos bloques de los vectores de Krylov. A continuación se detallan los elementos que se han introducido para posibilitar que el solver lineal Krylov–Schur opere implícitamente sin formar las matrices de la linealización ni los vectores de Krylov. En la exposición solo consideramos el caso en el que se realiza una transformación espectral de desplazamiento e inversión.

Para la extensión de la base de Krylov se requiere la operación de multiplicación matriz-vector con la matriz

$$S = (A - \sigma B)^{-1} B, \tag{5.19}$$

con A y B dadas en (5.18). Para derivar las recurrencias con las que operar implícitamente con la matriz S, se considera la descomposición $(A-\sigma B)\Pi = U_\sigma L_\sigma$, donde Π es una matriz de permutación,

$$\Pi = \begin{bmatrix} 0 & I_{(d-1)n} \\ I_n & 0 \end{bmatrix}, \quad L_\sigma = \begin{bmatrix} \frac{1}{b_{d-1}(\sigma)} I & & & \\ -\frac{b_0(\sigma)}{b_{d-1}(\sigma)} I & I & & \\ \vdots & & \ddots & \\ -\frac{b_{d-2}(\sigma)}{b_{d-1}(\sigma)} I & & & I \end{bmatrix}, \text{ y}$$

$$U_\sigma = \begin{bmatrix} Q_d(\sigma) & D_0 & D_1 & \cdots & D_{d-2} \\ & (\sigma_0 - \sigma) I & \beta_1(1 - \frac{\sigma}{\xi_1}) I & & \\ & & (\sigma_1 - \sigma) I & \ddots & \\ & & & \ddots & \beta_{d-2}(1 - \frac{\sigma}{\xi_{d-2}}) I \\ & & & & (\sigma_{d-2} - \sigma) I \end{bmatrix}. \tag{5.20}$$

Utilizando esta descomposición, el cálculo $w = Sx = \Pi L_\sigma^{-1} U_\sigma^{-1} B x$ se realiza resolviendo, en primer lugar, el sistema $U_\sigma y = Bx$, utilizando la recurrencia

$$\begin{aligned} y^{d-1} &= \frac{1}{\sigma_{d-2} - \sigma} x^{d-2} + \frac{\beta_{d-1}}{(\sigma_{d-2} - \sigma)\xi_{d-1}} x^{d-1}, \\ y^j &= \frac{1}{\sigma_{j-1} - \sigma} x^{j-1} + \frac{\beta_j}{(\sigma_{j-1} - \sigma)\xi_j} x^j - \frac{\beta_j}{\sigma_{j-1} - \sigma}(1 - \frac{\sigma}{\xi_j}) y^{j+1}, \\ & \quad j = d-2, \ldots, 1, \\ y^0 &= Q_d(\sigma)^{-1} \left(-D_0 y^1 - D_1 y^2 - \cdots - D_{d-2} y^{d-1} - \frac{1}{\beta_d} D_d x^{d-1} \right). \end{aligned} \tag{5.21}$$

En segundo lugar, se resuelve el sistema $L_\sigma \tilde{w} = y$ mediante las expresiones

$$\begin{aligned} \tilde{w}^0 &= b_{d-1}(\sigma) y^0, \\ \tilde{w}^j &= y^j + b_{j-1}(\sigma) y^0, \quad j = 1, \ldots, d-1. \end{aligned} \tag{5.22}$$

Finalmente, el vector solución se obtiene aplicando la permutación $w = \Pi \tilde{w}$.

Se hace notar que, en la implementación realizada, cuando el problema se ha definido mediante un operador del tipo (5.6), se trabaja con las diferencias divididas sin que éstas sean formadas explícitamente. Al ser combinaciones lineales de las matrices del problema (5.11), las operaciones donde éstas aparecen, que son, el cálculo de $Q(\sigma)$ y la formación del lado derecho en el sistema para la obtención de y^0 en (5.21), se reorganizan de forma que éstas se expresan directamente en función de las matrices que definen el problema inicial.

Al utilizar el método Krylov–Schur en la resolución del problema asociado a la linealización (5.18), se obtiene una relación de Arnoldi (2.5), de orden k, para la matriz S definida en (5.19). A partir de dicha relación, considerando V_k dividido en la forma (2.2), en d bloques de n filas consecutivas, e igualando los $d-1$ últimos bloques de filas en

$$BV_k = (A - \sigma B)(V_k H_k + h_{k+1,k} v_{k+1} e_k^*), \tag{5.23}$$

se obtienen $(d-1)$ relaciones que ponen de manifiesto la dependencia lineal entre los vectores del conjunto formado por las columnas de los bloques $\{V_k^i\}_{i=0}^{d-1}$ y los vectores $\{v_{k+1}^i\}_{i=0}^{d-1}$. Ello da lugar a una igualdad entre subespacios como la obtenida en (2.38) al considerar la resolución de problemas polinómicos, que demuestra que, también en este caso, es posible la utilización de una base al estilo de TOAR polinómico para representar los distintos bloques de la base de Krylov.

En el Algoritmo 2.2, en el que se detallan los distintos pasos del método TOAR polinómico para la resolución de PEPs, las operaciones dependientes de la linealización se han mantenido encapsuladas en la función denominada *expand* de la línea 5. Este hecho implica que dicho algoritmo pueda ser ahora utilizado para la resolución del problema racional, sin más que redefinir dicha función, la cual recibirá como entrada, en este caso, las funciones D_j que definen $Q_d(\lambda)$. Las recurrencias obtenidas en (5.21) y (5.22) se utilizan ahora para llevar a cabo la multiplicación asociada a la expansión del subespacio de Krylov. El modo en que trabaja dicha función en la implementación del método NLEIGS realizada se detalla a continuación.

Suponiendo que se tiene una representación tipo TOAR,

$$\begin{bmatrix} V_k^i & v_{k+1}^i \end{bmatrix} = U_{k+d} \begin{bmatrix} G_k^i & g_{k+1}^i \end{bmatrix}, \quad i = 0, \ldots, d-1, \tag{5.24}$$

de los vectores de Krylov de una relación de Arnoldi de orden k, para la matriz S definida en (5.19), la función

$$[u_{k+d+1}, g] = \text{expand}(\{D_i\}_{i=0}^{d}, U_{k+d}, g_{k+1}), \tag{5.25}$$

genera un vector u_{j+d+1} con el que se extiende la base de TOAR (si es necesario), y los coeficientes g con los que se representa el vector de expansión $w = S v_{k+1}$ en

función de dicha base mediante los siguientes pasos. En primer lugar, se definen los coeficientes

$$\begin{cases} \tilde{g}^{d-1} = \dfrac{1}{\sigma_{d-2}-\sigma} g_{k+1}^{d-2} + \dfrac{\beta_{d-1}}{(\sigma_{d-2}-\sigma)\xi_{d-1}} g_{k+1}^{d-1}, \\ \tilde{g}^{j} = \dfrac{1}{\sigma_{j-1}-\sigma} g_{k+1}^{j-1} + \dfrac{\beta_{j}}{(\sigma_{j-1}-\sigma)\xi_{j}} g_{k+1}^{j} - \dfrac{\beta_{j}}{\sigma_{j-1}-\sigma}(1-\dfrac{\sigma}{\xi_j})\tilde{g}^{j+1}, \\ \qquad j = d-2, \ldots, 1. \end{cases}$$

En segundo lugar, utilizando las expresiones, $y^i = U_{j+d}\tilde{g}^i$, para $i = 1, \ldots, d-1$, y $v_{k+1}^{d-1} = U_{j+d} g_{k+1}^{d-1}$, se calcula el vector

$$y^0 = Q_d(\sigma)^{-1}\left(-D_0 y^1 - D_1 y^2 - \cdots - D_{d-2} y^{d-1} - \frac{1}{\beta_d} D_d v_{k+1}^{d-1}\right), \tag{5.26}$$

que se utiliza para generar el vector u_{j+d+1} (si no hay dependencia lineal) con el que se extiende la base representada en U_{j+d}. En tercer lugar, se definen las coordenadas $\tilde{g}^0$ que expresan y^0 respecto de la base extendida

$$y^0 = U_{j+d+1}\tilde{g}^0. \tag{5.27}$$

Los coeficientes g^i de (5.25) se obtienen a partir de (5.22), utilizando los coeficientes $\tilde{g}$ obtenidos,

$$\begin{cases} g^j = \tilde{g}^{j+1} + b_j(\sigma)\tilde{g}^0, \quad j = 0, \ldots, d-2 \\ g^{d-1} = b_{d-1}(\sigma)\tilde{g}^0. \end{cases} \tag{5.28}$$

Como ya se ha comentado anteriormente, una vez se han derivado las ecuaciones (dependientes de la linealización) que permiten la expansión del subespacio de Krylov, el resto de elementos de la variante TOAR del método de Krylov–Schur expuestos en la sección 2.2.3 se mantienen sin cambios.

Se hace notar que la estructura de la linealización (5.16), con las matrices (5.18), se ajusta al tipo de problemas que pueden ser resueltos mediante el método CORK [90]. Dicho método aborda, de un forma general, la resolución de problemas (polinómicos) de valores propios que se resuelven planteando una linealización estructurada como las utilizadas en los capítulos 2 y 3. El enfoque seguido en CORK, combina el método *rational Krylov* junto con una representación tensorial (tipo TOAR) de la base del subespacio de Krylov.

5.3. Refinamiento de pares propios

En la resolución de un NEP mediante interpolación, puede ocurrir que las aproximaciones de pares propios obtenidas no cumplan con la tolerancia deseada. Esto sucedería, por ejemplo, en caso de utilizar un grado insuficiente para la aproximación. Con el fin de disponer de herramientas con las que mejorar en dichos casos los pares propios calculados, se ha incluido en SLEPc la opción de realizar el refinamiento de

los mismos mediante el método de Newton. La implementación realizada está basada en el trabajo de Kressner [51], particularizado para el caso de pares invariantes de un único vector. En ella se sigue un esquema similar al planteado en la sección 4.1 para el problema polinómico, incluyendo varias posibilidades de resolución para el sistema lineal que se genera a cada iteración, y la posibilidad de añadir un segundo nivel de paralelismo, útil principalmente si se quiere refinar un elevado número de pares propios.

Para llevar a cabo el refinamiento de una aproximación $(\tilde{x}, \tilde{\lambda})$ de un determinado par propio asociado a un autovalor simple de un problema no lineal (1.8), se toma dicha aproximación como punto de partida del método de Newton aplicado al sistema de ecuaciones

$$\begin{cases} T(\lambda)x = 0 \\ w^*x - 1 = 0, \end{cases} \tag{5.29}$$

donde w representa un vector de normalización (fijo). En cada iteración, se inicializa w con $\tilde{x}$ normalizado, y obtenemos una nueva aproximación mejorada del par propio mediante la actualización $(\tilde{\lambda}, \tilde{x}) \leftarrow (\tilde{\lambda}, \tilde{x}) - (\Delta\lambda, \Delta x)$, con la corrección $(\Delta\lambda, \Delta x)$ obtenida mediante la resolución del sistema de ecuaciones lineales

$$L \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{5.30}$$

donde $R := T(\tilde{\lambda})\tilde{x}$ es el residuo asociado a $(\tilde{\lambda}, \tilde{x})$ y L es la matriz jacobiana correspondiente a la función del sistema (5.29), dada por

$$L = \begin{bmatrix} T(\tilde{\lambda}) & T'(\tilde{\lambda})\tilde{x} \\ \tilde{x}^* & 0 \end{bmatrix}. \tag{5.31}$$

De forma análoga al enfoque seguido en la implementación del refinamiento de pares invariantes en PEP, también en este caso se han considerado varias posibilidades a la hora de plantear la resolución del sistema (5.30) con la matriz (5.31). Los esquemas implementados son:

Explicito. Con este esquema se crea la matriz (5.31) explícitamente. Permite el uso de los solvers y precondicionadores paralelos para la resolución de sistemas lineales, disponibles a través de la interfaz de PETSc. Éstos incluyen tanto métodos iterativos como directos.

MBE. El método de eliminación a bloques mixta, descrito en el Algoritmo 4.2, es un método diseñado para la resolución de sistemas con matrices en la forma (5.31) en las que el bloque principal, dado en este caso por $T(\tilde{\lambda})$, siendo $\tilde{\lambda}$ un autovalor aproximado, es casi singular. Con este método los sistemas que se resuelven son de la misma dimensión que el problema de valores propios del que se quieren refinar algunos de sus pares propios. Este método tiene ciertas limitaciones en cuanto a la utilización de métodos iterativos para las resoluciones de sistemas con la matriz $T(\tilde{\lambda})$, por ser ésta casi singular. Por ello, para este esquema, en general se utilizarán métodos directos.

Schur. Para utilizar un enfoque basado en el complemento de Schur en la forma explicada en la sección 4.2.2, es necesario cambiar la ecuación relativa a la normalización en (5.29) para obtener una matriz jacobiana similar a (5.31), de forma que el último elemento diagonal sea no nulo. Para ello, consideramos la condición de normalización dada por

$$w^* \begin{bmatrix} x \\ \lambda x \end{bmatrix} - 1 = 0, \tag{5.32}$$

donde w representa un vector fijo, que puede ser actualizado al inicio de cada iteración para que tome el valor

$$\begin{bmatrix} \tilde{x} \\ \tilde{\lambda}\tilde{x} \end{bmatrix},$$

donde $(\tilde{\lambda}, \tilde{x})$ es el par propio aproximado, con $\|\tilde{x}\| = 1$.

El complemento de Schur asociado a la matriz jacobiana, que viene dado por

$$S := T(\tilde{\lambda}) - T'(\tilde{\lambda})\tilde{x}(\tilde{\lambda}^*)^{-1}(1 + |\tilde{\lambda}|^2)\tilde{x}^*, \tag{5.33}$$

no se forma explícitamente y no se factoriza. Por ello, los métodos de resolución de sistemas que se utilizarán, en este caso, serán siempre de tipo iterativo.

El esquema a utilizar dependerá de las características del problema a resolver.

Dado que no hay dependencia alguna en el proceso de refinamiento de los distintos pares propios, la incorporación de un segundo nivel de paralelismo se implementa de forma trivial. No obstante, la división en subgrupos del conjunto total de procesos, únicamente es recomendable cuando el método de resolución de sistemas utilizado no escale bien hasta el número de procesos considerado. Este caso se suele dar, por ejemplo, al utilizar métodos directos de resolución y un número elevado de procesos.

5.4. Resultados

En esta sección se muestran resultados obtenidos en varias pruebas realizadas para la evaluación de los solvers no lineales descritos en este capítulo, los que denominamos *Interpol*, para el solver que realiza la interpolación polinómica, y *NLEIGS*, para el que implementa el método del mismo nombre. El entorno de ejecución utilizado es el descrito en la sección 2.6.1. La versión de SLEPc utilizada en esta ocasión ha sido la de desarrollo, ya que alguna de las opciones utilizadas, en concreto, la de obtención automática de las singularidades en problemas racionales (descrita en la sección 5.2), no está disponible en la última versión de esta biblioteca (versión 3.7). Los problemas de test utilizados son los denominados delay, loaded_string y gun, provenientes de la colección NLEVP [14], donde se puede encontrar una descripción de los mismos. No obstante, para la formación del test asociado al problema delay se ha utilizado la descripción del mismo incluida en [27], que nos ha permitido generar

problemas de dimensión elevada para las pruebas de escalabilidad. En las ejecuciones realizadas se ha utilizado aritmética real, excepto para el problema gun en el que los parámetros de definición son complejos. A continuación se especifican las funciones matriciales no lineales asociadas a cada uno de los problemas de test considerados, y la región de búsqueda utilizada para cada uno de ellos.

Time_delay. El problema considerado, descrito en [27] y [49], se expresa como

$$T(\lambda) = -\lambda + A_0 + e^{-\tau\lambda}A_1, \tag{5.34}$$

con matrices reales A_0 y A_1, tridiagonal y diagonal, respectivamente, y retardo $\tau = 0{,}05$. La región considerada, extraída de [49], es $\Sigma = [-70, 20]$, en la que hay 12 autovalores para el problema (5.34). En este caso se trata de una función holomorfa en $\mathbb{C}$ por lo que $\Xi = \emptyset$.

Loaded_string. Es un problema racional real, dado por la función

$$T(\lambda) = A - \lambda B + \frac{\lambda}{\lambda - \gamma}C, \tag{5.35}$$

con matrices tridiagonales definidas en [14], y parámetro $\gamma = 1$, para el que se desean calcular los menores autovalores reales. La región de búsqueda considerada ha sido $\Sigma = [4, 800]$, en la que hay 9 autovalores. El conjunto de singularidades es $\Xi = \{1\}$.

Gun. Este problema, extraído de [14] y [36] tiene la forma

$$T(\lambda) = K - \lambda M + i\sqrt{\lambda - \kappa_1^2}W_1 + i\sqrt{\lambda - \kappa_2^2}W_2, \tag{5.36}$$

con matrices K, M y W reales de dimensión $n = 9956$, y parámetros $\kappa_1 = 0$ y $\kappa_2 = 108{,}8774$. Se utiliza la región de búsqueda de autovalores dada por la envoltura convexa $\Sigma = \mathrm{Co}(\{12500 - i, 120500 - i, 120500 + 30000i, 70000 + 30000i\})$, que es una aproximación a la región utilizada en [36]. El conjunto de singularidades, $\Xi = (-\infty, \kappa_2^2)$, está determinado por las de la raíz cuadrada principal.

Tabla 5.1: Descripción de los problemas de test utilizados para la evaluación de los solvers no lineales basados en interpolación. Se indica la dimensión *dim* de las matrices de coeficientes, la función no lineal involucrada (al margen de términos polinómicos), la región donde se calculan los autovalores, el conjunto de singularidades, el número de autovalores *nev* en la región y el desplazamiento en torno al cual se calculan σ. La región Σ en el problema gun es la dada en la Figura 5.1.

nombre	dim	función	región	singularidades	nev	σ
delay	2^{20}	exponencial	$[-70, 20]$	$\emptyset$	12	0
loaded_string	10^6	racional	$[4, 800]$	$\{1\}$	9	10
gun	9956	raíz cuadrada	Σ	$(-\infty, \kappa^2)$	24	65000

Tabla 5.2: Resultados de varias ejecuciones con 16 procesos utilizando los solvers no lineales basados en interpolación. Se muestra el solver utilizado, el número de términos en la interpolación *term*, la tolerancia utilizada, las iteraciones realizadas por el método de Krylov, si se ha llevado a cabo refinamiento (un máximo de 10 iteraciones) de los pares propios *ref*, el residuo relativo máximo y el tiempo de ejecución en segundos.

problema	método	term	tol	iter	ref	$\|T(\lambda)x\|/(\|\lambda\|\|x\|)$	tiempo
delay	Interpol	14	10^{-9}	31	No	0,4	167
	Interpol			31	Sí	6×10^{-5}	357
	NLEIGS	14	10^{-9}	15	No	3	71
	NLEIGS			15	Sí	3×10^{-5}	260
loaded_string	Interpol	10	10^{-12}	15	No	3×10^{-5}	57
	Interpol			14	Sí	6×10^{-11}	334
	NLEIGS	3	10^{-12}	5	No	5×10^{-10}	23
gun	NLEIGS	17	10^{-9}	5	No	2×10^{-10}	5.38

La información de los casos de test utilizados se ha resumido en la Tabla 5.1, en la que se incluyen algunas características de las ejecuciones realizadas con estos problemas. En la Tabla 5.2 se muestran datos de dichas ejecuciones. En el caso del problema delay, para el que la función que lo define no tiene singularidades, y ambos métodos realizan una interpolación polinómica, podemos observar que los dos solvers se comportan de forma similar en cuanto a la exactitud que proporcionan. Sin embargo, en el problema loaded_string hay una clara diferencia en la precisión obtenida por ambos solvers, debido al hecho de que, dado que la función que define dicho problema es racional, el método NLEIGS no realiza una interpolación propiamente dicha, sino que reescribe la función racional y resuelve una linealización de la misma. En las ejecuciones con este problema se ha utilizado la opción, recientemente incluida en la versión de desarrollo de SLEPc, de cálculo automático de las singularidades en problemas racionales.

Referente al refinamiento iterativo de pares propios, podemos comprobar que su utilización (realizada hasta un máximo de 10 iteraciones) consigue mejorar, en todos los casos, las aproximaciones obtenidas. En dicha tabla podemos también observar que el solver NLEIGS resuelve satisfactoriamente el problema gun. En este caso, no se incluye la resolución con el solver Interpol porque se ha utilizado una región en el plano complejo.

En la Figura 5.1 se muestra la distribución de autovalores obtenidos, dentro de la región Σ de búsqueda, para el problema gun. En dicha figura también se muestran tiempos de la ejecución en paralelo con dicho problema. La reducida dimensión de las matrices, y la utilización de un método directo para resolución de los sistemas lineales hace que no sea posible escalar a un número elevado de procesos en este caso. Las pruebas de escalabilidad realizadas con los problemas delay y loaded_string muestran,

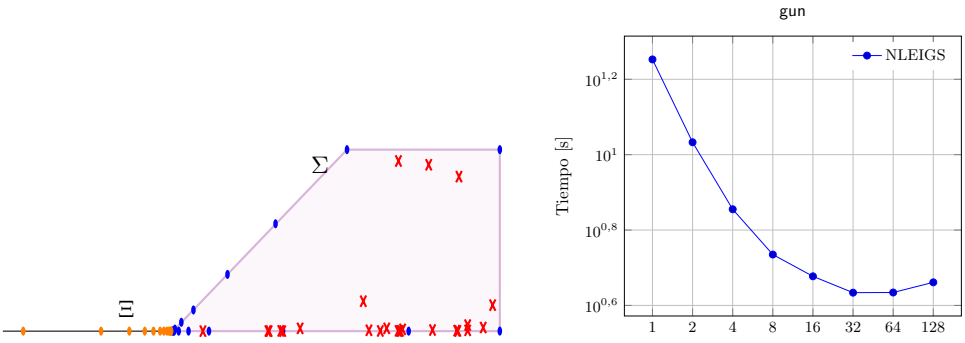

Figura 5.1: Representación de la región de búsqueda Σ y del conjunto de singularidades Ξ, para el problema gun (izquierda). Se muestra la distribución de los nodos (azul) y polos (naranja) de interpolación, así como los 24 autovalores obtenidos en la región de búsqueda. A la derecha, se muestra el tiempo de ejecución (en segundos) utilizando hasta 128 procesos MPI con el solver NLEIGS.

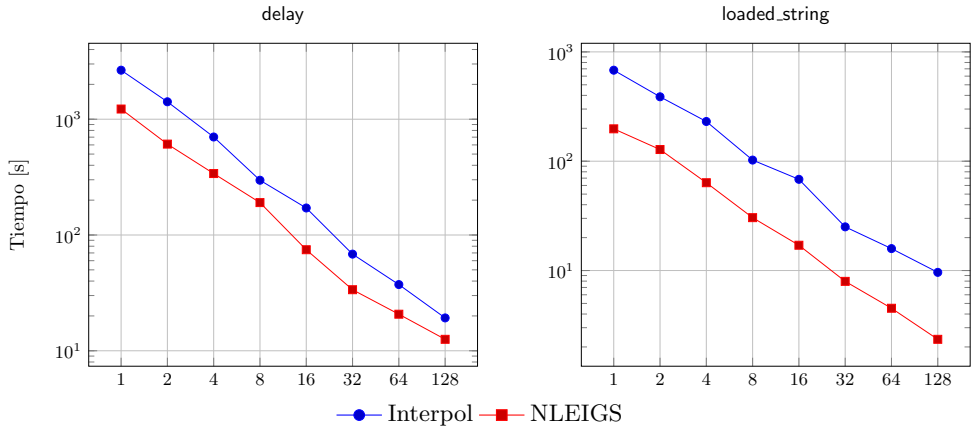

Figura 5.2: Tiempo de ejecución (en segundos) utilizando hasta 128 procesos MPI con los problemas delay (izquierda) y loaded_string (derecha) de la colección NLEVP, resueltos con los solvers *Interpol* y *NLEIGS*. Los parámetros de ejecución se muestran en la Tabla 5.2.

en la Figura 5.2, el buen comportamiento paralelo de ambos solvers. Al comparar con otras ejecuciones similares, mostradas en capítulos anteriores, con el problema loaded_string y el solver de interpolación polinómica (ver por ejemplo la Figura 2.3 donde se calculan 7 autovalores para dicho problema) observamos una diferencia considerable en la escalabilidad que presentan. Ello es debido a que entonces se utilizó un método directo para la resolución de los sistemas lineales, mientras que en esta ocasión se ha utilizado el método iterativo Bi-CGStab, conjuntamente con un precondicionador proporcionado por la biblioteca externa HYPRE[1], para la que PETSc incluye una interfaz.

5.5. Conclusiones

En este capítulo se ha abordado la resolución numérica del problema no lineal de valores propios mediante el desarrollo de dos solvers específicos para dichos problemas. Aparte del interés general existente en la resolución de NEPs, para los que SLEPc ya dispone de una clase específica, el principal motivo por el que en esta tesis se ha incidido en este tipo de problemas, es poder aplicar en este contexto algunos de los esquemas estudiados para el problema polinómico de valores propios.

Los dos solvers no lineales desarrollados en este capítulo están basados en la interpolación del operador no lineal que define el NEP. El primero de ellos realiza una interpolación polinómica y resuelve el PEP resultante mediante cualquiera de los solvers polinómicos descritos en los capítulos anteriores. El segundo realiza una interpolación racional que se resuelve vía linealización. Para dicha resolución se ha incluido, como aportación de esta tesis, un esquema de tratamiento eficiente de la memoria similar al realizado en el capítulo 2 en el contexto de problemas polinómicos resueltos vía linealización. Para los casos en los que el NEP está definido mediante una función racional, se ha incluido en este último solver un tratamiento especial en el que las singularidades son obtenidas de forma automática por el solver.

Además del desarrollo de los dos solvers mencionados, se ha estudiado la incorporación, dentro del contexto del problema no lineal, de la posibilidad de realizar el refinamiento iterativo de los pares propios calculados, de forma similar al esquema descrito en el capítulo 4 para problemas polinómicos.

Las pruebas realizadas con los solvers descritos en este capítulo, han permitido comprobar que éstos se comportan de forma satisfactoria, con un buen rendimiento paralelo. Como consecuencia, ambos han pasado a formar parte de los solvers disponibles en la clase `NEP` de SLEPc.

[1] `http://www.llnl.gov/casc/hypre`

Capítulo 6

Conclusiones

El trabajo presentado en esta tesis se ha centrado principalmente en la resolución numérica de problemas polinómicos de valores propios, en los que las matrices coeficiente que definen el polinomio matricial son dispersas y de gran dimensión. Para su resolución se han realizado diversos desarrollos software en el marco de la biblioteca SLEPc. Como fruto de esta tesis, dicha biblioteca dispone de varios solvers con los que realizar la resolución, en paralelo, del problema polinómico de valores propios de forma robusta y eficiente.

En primer lugar, en el capítulo 2 se han descrito los solvers basados en linealización. Los denominados TOAR y Arnoldi Básico han sido desarrollados para problemas polinómicos de grado arbitrario, definidos de forma genérica en función de bases polinómicas distintas de la de monomios, y han sido dotados de varias opciones que les proporcionan mayor robustez y flexibilidad, tales como soporte para transformaciones espectrales, así como distintas estrategias de escalado y extracción de los pares propios. El solver TOAR, que utiliza un tratamiento eficiente de la memoria, se ha implementado como respuesta a la problemática que genera el aumento de la dimensión del problema a resolver en los esquemas basados en linealización. Para dicho solver se ha diseñado una forma efectiva del bloqueo de los vectores convergidos, de modo que éstos no vuelvan a modificarse durante la ejecución. El solver TOAR ha resultado ser el más eficiente en términos computacionales y se ha determinado como el solver polinómico por defecto en SLEPc.

En el capítulo 3 se han desarrollado dos solvers específicos para el caso particular de problemas cuadráticos simétricos. Por un lado, el solver que implementa el método pseudo-Lanczos utiliza un enfoque basado en linealización y mantiene la simetría del problema inicial a lo largo de la ejecución, aunque a costa de no asegurar la estabilidad del proceso. El solver STOAR se ha diseñado como una variante del anterior, incorporando un esquema de computación eficiente al estilo del solver TOAR. En ambos solvers se ha incluido una técnica de reinicio grueso que ha requerido la implementación de métodos específicos para la diagonalización de matrices pseudo-simétricas de tamaño reducido, para los que se han estudiado varias estrategias con las que minimizar el riesgo de inestabilidad que dichos métodos presentan.

Para asegurar que los solvers simétricos, potencialmente inestables, proporcionan siempre resultados correctos, se ha incorporado un mecanismo para la detección de inestabilidad, que permite detener la ejecución si ésta se presenta.

En el capítulo 4 se ha descrito la implementación de dos métodos que se plantean en términos del cálculo de pares invariantes de problemas polinómicos, y que están basados en el método de Newton. Dichos desarrollos son: el refinamiento iterativo de pares invariantes, que se utiliza como un complemento para mejorar la robustez de otros solvers como los basados en linealización; y el solver polinómico que implementa el método Jacobi–Davidson, para el que se ha incluido una técnica eficaz de deflación. Estos métodos han sido adaptados para el caso de problemas polinómicos definidos en términos de una base genérica de polinomios, deduciendo, cuando ha sido necesario, nuevas expresiones adaptadas a este caso. En la implementación realizada del refinamiento iterativo, se han incluido varias opciones para la resolución de los sistemas lineales asociados a la ecuación de corrección de Newton. Esto permite que para cada tipo de problema sea posible escoger la opción más eficaz para minimizar el sobrecoste que supone la utilización de dicho método.

Finalmente, el capítulo 5 se ha dedicado, a diferencia del resto de la tesis, a la resolución numérica del problema no lineal de valores propios. En éste se describen dos solvers para la resolución de dichos problemas, que guardan relación con los desarrollados para problemas polinómicos. El primero de ellos realiza la interpolación polinómica del operador no lineal y utiliza un solver polinómico para su resolución, mientras que el segundo implementa el método NLEIGS, que realiza una interpolación racional que se resuelve vía linealización. Para este último solver se ha incluido un esquema de resolución eficiente similar al utilizado en el solver TOAR. En el caso de problemas racionales, este solver incluye la opción de cálculo automático de las singularidades. En este contexto de problemas no lineales de valores propios, también se ha incorporado la opción de refinamiento iterativo de pares propios.

Las distintas pruebas llevadas a cabo con las implementaciones realizadas, han permitido comprobar el buen comportamiento numérico y las adecuadas prestaciones paralelas de los solvers desarrollados.

Como resultado de esta tesis, la biblioteca SLEPc dispone de un nuevo módulo denominado `PEP` que engloba la funcionalidad específica para la resolución del problema polinómico de valores propios, en el que están incluidos todos los desarrollos descritos en esta tesis. De acuerdo con el objetivo principal planteado, se ha incorporado un número significativo de solvers variados que proporcionan flexibilidad para escoger el más adecuado a cada aplicación específica. Varios trabajos recientes [19, 57, 58] muestran la utilización, por parte de la comunidad científica, de la funcionalidad incorporada a la biblioteca SLEPc.

Publicaciones

Gran parte de los contenidos expuestos en esta tesis han sido objeto de publicaciones en revistas científicas. En concreto, los capítulos 2, 3 y la mayor parte del 4 se corresponden, respectivamente, con los siguientes artículos publicados:

[22] C. Campos and J. E. Roman. Parallel Krylov solvers for the polynomial eigenvalue problem in SLEPc. *SIAM J. Sci. Comput.*, 38(5):S385–S411, 2016.

[23] C. Campos and J. E. Roman. Restarted Q-Arnoldi-type methods exploiting symmetry in quadratic eigenvalue problems. *BIT*, 56(4):1213–1236, 2016.

[21] C. Campos and J. E. Roman. Parallel iterative refinement in polynomial eigenvalue problems. *Numer. Linear Algebra Appl.*, 23(4):730–745, 2016.

Extensiones de la tesis

Se plantean varias líneas de extensión, en el futuro, para el trabajo desarrollado en esta tesis. En primer lugar, para los solvers basados en linealización, descritos en el capítulo 2, está previsto estudiar la posibilidad de incluir soporte para la utilización de otras linealizaciones diferentes de la considerada. Por otro lado, utilizando linealizaciones simétricas como las descritas en [43], se pretende ampliar la funcionalidad de los solvers simétricos, descritos en el capítulo 3, para que éstos den cobertura a la resolución de problemas polinómicos simétricos con grado mayor que 2. En la línea de los desarrollos realizados en el capítulo 3, sería interesante la incorporación de solvers específicos que preserven otras estructuras como la hamiltoniana [11].

En el contexto de problemas cuadráticos simétricos hiperbólicos [45], está previsto la incorporación de un solver, similar al existente en SLEPc para el problema lineal simétrico definido [20], que utilice una técnica de disección del espectro (*spectrum slicing*) para la obtención de todos los autovalores contenidos en un intervalo real. En este caso, el solver de disección del espectro utilizaría la implementación del método pseudo-Lanczos descrita en esta tesis y el cálculo de la inercia para problemas hiperbólicos estudiado en [50].

Para el método de refinamiento iterativo de pares invariantes (capítulo 4) se podría estudiar la posibilidad de que dicha técnica sea utilizada también como un solver en sí, para su uso en aplicaciones que generan una secuencia de problemas polinómicos donde la solución de un problema varíe poco respecto a la del siguiente.

El solver Jacobi–Davidson polinómico admite todavía la inclusión de algunas características de utilidad, como por ejemplo la posibilidad de efectuar la computación utilizando aritmética real siempre que las matrices que definen el problema sean reales. También está prevista la adaptación del solver para que éste pueda ser utilizado en problemas polinómicos definidos mediante bases distintas de la de monomios, incluyendo la ampliación de la funcionalidad de la subclase `DSPEP` que ello requiere.

En el contexto del problema no lineal de valores propios sería interesante la incorporación del refinamiento de pares invariantes propuesto en [51], similar al descrito en el capítulo 4. De forma más general, la resolución numérica del problema no lineal de valores propios plantea una línea clara de extensión mediante la incorporación de nuevos solvers para dichos problemas.

Actualmente se han iniciado colaboraciones para la utilización del solver NLEIGS en la resolución de dos problemas de aplicación en el área de fotónica. En una de

ellas se pretende utilizar dicho método en la resolución del problema que en [19] se aborda planteando y resolviendo un PEP. En la otra, se colabora con los autores de [5] para la resolución de un problema similar utilizando también dicho solver.

Apéndice A

Interfaz de usuario para PEP en SLEPc

La interfaz de usuario definida para cada uno de los módulos principales de SLEPc (ver Figura 1.4) mantiene una estructura homogénea que facilita la utilización de los mismos. A continuación se describen los elementos más relevantes de la interfaz de usuario para el módulo `PEP`, que contiene la funcionalidad asociada a la resolución del problema polinómico de valores propios.

La Figura A.1 proporciona el código básico para la resolución de un problema polinómico de valores propios. El ejemplo muestra cómo se crea (y se destruye al final) una na instancia, `pep`, de la clase PEP, a la que se le asocia un comunicador MPI en el que se llevan a cabo las distintas operaciones paralelas. El problema a resolver se especifica proporcionando las matrices coeficiente con `PEPSetOperators` (se omite el código de generación de las matrices), y opcionalmente indicando la base de polinomios a utilizar con `PEPSetBasis` (por defecto se utiliza la base de monomios). La llamada a la rutina `PEPSetFromOptions` permite la utilización, por parte del usuario, de varias opciones a través de la línea de comandos, tal como se ilustra en algunos ejemplos más adelante. La invocación al solver propiamente dicho se hace a través de la llamada `PEPSolve`. Seguidamente, la solución puede ser recuperada con `PEPGetConverged` y `PEPGetEigenpair`. Se hace notar que la definición tanto del autovalor como del autovector se hace mediante dos variables, ello permite la obtención de pares propios complejos aun en el caso de que los cálculos se realicen en aritmética real; en otro caso, `PetscScalar` representa un tipo de datos en coma flotante complejo, el resultado se retorna en la primera variable y la segunda no se utiliza.

El solver polinómico a utilizar puede ser fijado en el código mediante la llamada `PEPSetType` con el nombre, por ejemplo, `PEPJD`, o alternativamente en tiempo de ejecución en la forma:

```
$ ./example -pep_type jd -pep_nev 6 -pep_ncv 24 -pep_tol 1e-9
```

```
1   #define NMAT 5
2   PEP         pep;           /*  eigensolver context  */
3   Mat         A[NMAT];       /*  coefficient matrices */
4   Vec         xr, xi;        /*  eigenvector, x       */
5   PetscScalar kr, ki;        /*  eigenvalue, k        */
6   PetscInt    j, nconv;
7   PetscReal   error;
8
9   PEPCreate( PETSC_COMM_WORLD, &pep );
10  PEPSetOperators( pep, NMAT, A );
11  PEPSetFromOptions( pep );
12  PEPSolve( pep );
13  PEPGetConverged( pep, &nconv );
14  for (j=0; j<nconv; j++) {
15    PEPGetEigenpair( pep, j, &kr, &ki, xr, xi );
16    PEPComputeError( pep, j, PEP_ERROR_BACKWARD, &error );
17  }
18  PEPDestroy( &pep );
```

Figura A.1: Ejemplo de código básico para la resolución de un problema utilizando un objeto de la clase `PEP`.

Las otras opciones indican que se deben calcular 6 autovalores con el solver Jacobi–Davidson, utilizando 24 vectores para almacenar las bases de los subespacios de búsqueda y test, y una tolerancia de 10^{-9} para el criterio de convergencia. Cada una de estas opciones pueden ser también dadas por código utilizando la correspondiente función de interfaz. Los detalles para cada una de ellas se pueden encontrar en la guía de usuario [72] o en la documentación online disponible en la web de SLEPc.

El usuario puede especificar si los autovalores deseados son los de mayor (o menor) magnitud, parte real, o parte imaginaria. Alternativamente, los autovalores buscados se pueden definir mediante la distancia a un desplazamiento τ, o por medio de la inclusión (o exclusión) de una región en el plano complejo (utilizando la clase `RG`). En cualquier caso, los autovalores interiores se obtienen de forma más eficiente si el solver se utiliza en combinación con la transformación espectral de desplazamiento e inversión (gestionada por la clase `ST`). Tal como se ha indicado en la sección 2.2.1 hay dos formas de llevar a cabo la transformación espectral cuando ésta se utiliza conjuntamente con un solver basado en linealización: (i) transformando el problema polinómico, o (ii) dejando que el solver polinómico maneje internamente dicha transformación. La primera de las dos formas debe ser explícitamente activada con `STSetTransform`. Un ejemplo de linea de comandos sería:

```
$ ./sleeper -pep_type stoar -pep_target -10 -st_type sinvert
 -st_transform -pep_hermitian -st_ksp_type preonly -st_pc_type lu
 -st_pc_factor_mat_solver_package mumps
```

El ejemplo calcula un par propio próximo a -10 con STOAR utilizando la transformación de desplazamiento e inversión. Las opciones `-st_transform -pep_hermitian` son requeridas por el solver STOAR, que es exclusivo para problemas simétricos y

solo da soporte a la realización de la transformación espectral a nivel del problema polinómico. En dicho ejemplo se han incluido también opciones para la utilización de un método directo para la resolución de los sistemas lineales con la matriz $P(-10)$, llevados a cabo por la biblioteca MUMPS.

Los distintos solvers y opciones descritos en esta tesis pueden ser indicados por código o línea de comandos, por ejemplo; `PEPSetScale` activa el escalado sobre el parámetro, diagonal o ambos (ver sección 2.4); `PEPSetExtract` selecciona el tipo de extracción descrita en la sección 2.5.2; `PEPSetRefine` se utiliza para activar el refinamiento iterativo (sección 4.1.2), tanto para la variante múltiple como simple, así como el esquema a utilizar en la resolución de los sistemas lineales (sección 4.2), o el número de subcomunicadores utilizados. Estas opciones también pueden ser especificadas por línea de comandos, por ejemplo,

```
$ ./qd_cylinder -pep_type toar -pep_target 0.1 -pep_nev 10
-st_type sinvert -pep_refine multiple -pep_refine_scheme schur
```

Mediante las especificaciones anteriores se realizaría el refinamiento iterativo de un par invariante de dimensión 10, utilizando el esquema *Schur* en la resolución de los sistemas lineales.

Bibliografía

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

[2] A. Amiraslani, D. A. Aruliah, and R. M. Corless. Block LU factors of generalized companion matrix pencils. *Theor. Comput. Sci.*, 381(1–3):134–147, 2007.

[3] A. Amiraslani, R. M. Corless, and P. Lancaster. Linearization of matrix polynomials expressed in polynomial bases. *IMA J. Numer. Anal.*, 29(1):141–157, 2009.

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

[5] J. C. Araujo-Cabarcas and C. Engström. On spurious solutions in finite element approximations of resonances in open systems. Technical Report arXiv:1606.09635, 2017.

[6] J. Asakura, T. Sakurai, H. Tadano, T. Ikegami, and K. Kimura. A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Letters*, 1:52–55, 2009.

[7] Z. Bai, D. Day, and Q. Ye. ABLE: an adaptive block Lanczos method for non-Hermitian eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 20(4):1060–1082, 1999.

[8] 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.

[9] Z. Bai, T. Ericsson, and T. Kowalski. Symmetric indefinite Lanczos 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 249–260. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[10] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, K. Rupp, B. Smith, S. Zampini, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

[11] P. Benner, H. Faßbender, and M. Stoll. Solving large-scale quadratic eigenvalue problems with Hamiltonian eigenstructure using a structure-preserving Krylov subspace method. *Electron. Trans. Numer. Anal.*, 29:212–229, 2008.

[12] M. M. Betcke and H. Voss. Restarting iterative projection methods for Hermitian nonlinear eigenvalue problems with minmax property. *Numer. Math.*, 135:397–430, 2017.

[13] T. Betcke. Optimal scaling of generalized and polynomial eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 30(4):1320–1338, 2008.

[14] T. Betcke, N. J. Higham, V. Mehrmann, C. Schröder, and F. Tisseur. NLEVP: a collection of nonlinear eigenvalue problems. *ACM Trans. Math. Software*, 39(2):7:1–7:28, 2013.

[15] T. Betcke and D. Kressner. Perturbation, extraction and refinement of invariant pairs for matrix polynomials. *Linear Algebra Appl.*, 435(3):514–536, 2011.

[16] T. Betcke and H. Voss. A Jacobi–Davidson-type projection method for nonlinear eigenvalue problems. *Future Gener. Comp. Sy.*, 20(3):363–372, 2004.

[17] W.-J. Beyn. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra Appl.*, 436(10):3839–3863, 2012.

[18] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Software*, 28(2):135–151, June 2002.

[19] Y. Brûlé, B. Gralak, and G. Demésy. Calculation and analysis of the complex band structure of dispersive and dissipative two-dimensional photonic crystals. *J. Opt. Soc. Am. B*, 33(4):691–702, 2016.

[20] C. Campos and J. E. Roman. Strategies for spectrum slicing based on restarted Lanczos methods. *Numer. Algorithms*, 60(2):279–295, 2012.

[21] C. Campos and J. E. Roman. Parallel iterative refinement in polynomial eigenvalue problems. *Numer. Linear Algebra Appl.*, 23(4):730–745, 2016.

[22] C. Campos and J. E. Roman. Parallel Krylov solvers for the polynomial eigenvalue problem in SLEPc. *SIAM J. Sci. Comput.*, 38(5):S385–S411, 2016.

[23] C. Campos and J. E. Roman. Restarted Q-Arnoldi-type methods exploiting symmetry in quadratic eigenvalue problems. *BIT*, 56(4):1213–1236, 2016.

[24] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo. Parallel distributed computing using Python. *Adv. Water Resour.*, 34(9):1124–1139, 2011.

[25] D. Day. An efficient implementation of the nonsymmetric Lanczos algorithm. *SIAM J. Matrix Anal. Appl.*, 18(3):566–589, 1997.

[26] G. de Samblanx and A. Bultheel. Nested Lanczos: Implicitly restarting an unsymmetric Lanczos algorithm. *Numer. Algorithms*, 18(1):31–50, 1998.

[27] C. Effenberger. Robust successive computation of eigenpairs for nonlinear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 34(3):1231–1256, 2013.

[28] C. Effenberger and D. Kressner. Chebyshev interpolation for nonlinear eigenvalue problems. *BIT*, 52(4):933–951, 2012.

[29] 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.

[30] H. Fan, W. Lin, and P. van Dooren. Normwise scaling of second order polynomial matrices. *SIAM J. Matrix Anal. Appl.*, 26(1):252–256, 2004.

[31] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. Academic Press, New York, 1982.

[32] I. Gohberg, P. Lancaster, and L. Rodman. *Indefinite Linear Algebra and Applications*. Birkhaüser, Basel, Switzerland, 2005.

[33] W. Govaerts. Stable solvers and block elimination for bordered systems. *SIAM J. Matrix Anal. Appl.*, 12(3):469–483, 1991.

[34] W. Govaerts and J. D. Pryce. Mixed block elimination for linear systems with wider borders. *IMA J. Numer. Anal.*, 13(2):161–180, 1993.

[35] S. Güttel and F. Tisseur. The nonlinear eigenvalue problem. Technical Report MIMS 2017.7, School of Mathematics, The University of Manchester, 2017.

[36] S. Güttel, R. van Beeumen, K. Meerbergen, and W. Michiels. NLEIGS: A class of fully rational Krylov methods for nonlinear eigenvalue problems. *SIAM J. Sci. Comput.*, 36(6):A2842–A2864, 2014.

[37] S. Hammarling, C. J. Munro, and F. Tisseur. An algorithm for the complete solution of quadratic eigenvalue problems. *ACM Trans. Math. Software*, 39(3):18:1–18:19, 2013.

[38] 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.

[39] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[40] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2002.

[41] N. J. Higham and A. H. Al-Mohy. Computing matrix functions. *Acta Numerica*, 19:159–208, 2010.

[42] N. J. Higham, R.-C. Li, and F. Tisseur. Backward error of polynomial eigenproblems solved by linearization. *SIAM J. Matrix Anal. Appl.*, 29(4):1218–1241, 2007.

[43] N. J. Higham, D. S. Mackey, N. Mackey, and F. Tisseur. Symmetric linearizations for matrix polynomials. *SIAM J. Matrix Anal. Appl.*, 29(1):143–159, 2006.

[44] N. J. Higham, D. S. Mackey, and F. Tisseur. The conditioning of linearizations of matrix polynomials. *SIAM J. Matrix Anal. Appl.*, 28(4):1005–1028, 2006.

[45] N. J. Higham, F. Tisseur, and P. M. van Dooren. Detecting a definite Hermitian pair and a hyperbolic or elliptic quadratic eigenvalue problem, and associated nearness problems. *Linear Algebra Appl.*, 351:455–474, 2002.

[46] M. Hochbruck and D. Lochel. A multilevel Jacobi-Davidson method for polynomial PDE eigenvalue problems arising in plasma physics. *SIAM J. Sci. Comput.*, 32(6):3151–3169, 2010.

[47] T.-M. Huang, W.-W. Lin, and V. Mehrmann. A Newton-type method with nonequivalence deflation for nonlinear eigenvalue problems arising in photonic crystal modeling. *SIAM J. Sci. Comput.*, 38(2):B191–B218, 2016.

[48] F.-N. Hwang, Z.-H. Wei, T.-M. Huang, and W. Wang. A parallel additive Schwarz preconditioned Jacobi-Davidson algorithm for polynomial eigenvalue problems in quantum dot simulation. *J. Comput. Phys.*, 229(8):2932–2947, 2010.

[49] E. Jarlebring. *The spectrum of delay-differential equations: numerical methods, stability and perturbation*. PhD thesis, TU Carolo-Wilhelmina zu Braunschweig, 2008.

[50] A. Kostić and H. Voss. On Sylvester's law of inertia for nonlinear eigenvalue problems. *Electron. Trans. Numer. Anal.*, 40:82–93, 2013.

[51] D. Kressner. A block Newton method for nonlinear eigenvalue problems. *Numer. Math.*, 114:355–372, 2009.

[52] D. Kressner, M. M. Pandur, and M. Shao. An indefinite variant of LOBPCG for definite matrix pencils. *Numer. Algorithms*, 66(4):681–703, 2014.

[53] D. Kressner and J. E. Roman. Memory-efficient Arnoldi algorithms for linearizations of matrix polynomials in Chebyshev basis. *Numer. Linear Algebra Appl.*, 21(4):569–588, 2014.

[54] P. Lancaster. *Lambda-matrices and vibrating systems*. Pergamon Press, Oxford, 1966.

[55] P. Lancaster. Linearization of regular matrix polynomials. *Electron. J. Linear Algebra*, 17:21–27, 2008.

[56] P. Lancaster and Q. Ye. Rayleigh-Ritz and Lanczos methods for symmetric matrix pencils. *Linear Algebra Appl.*, 185:173–201, 1993.

[57] C. Lestringant and B. Audoly. Elastic rods with incompatible strain: Macroscopic versus microscopic buckling. *J. Mech. Phys. Solids*, 103:40–71, 2017.

[58] C. Lestringant, C. Maurini, A. Lazarus, and B. Audoly. Buckling of an elastic ridge: competition between wrinkles and creases. *Phys. Rev. Lett.*, 118(16):165501, 2017.

[59] D. Lu and Y. Su. Two-level orthogonal Arnoldi process for the solution of quadratic eigenvalue problems. Manuscript, 2012.

[60] D. Lu, Y. Su, and Z. Bai. Stability analysis of the two-level orthogonal Arnoldi procedure. *SIAM J. Matrix Anal. Appl.*, 37(1):195–214, 2016.

[61] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann. Structured polynomial eigenvalue problems: good vibrations from good linearizations. *SIAM J. Matrix Anal. Appl.*, 28(4):1029–1051, 2006.

[62] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann. Vector spaces of linearizations for matrix polynomials. *SIAM J. Matrix Anal. Appl.*, 28(4):971–1004, 2006.

[63] Y. Maeda, Y. Futamura, and T. Sakurai. Stochastic estimation method of eigenvalue density for nonlinear eigenvalue problem on the complex plane. *JSIAM Letters*, 3:61–64, 2011.

[64] Y. Matsuo, H. Guo, and P. Arbenz. Experiments on a parallel nonlinear Jacobi–Davidson algorithm. *Procedia Comp. Sci.*, 29:565–575, 2014.

[65] K. Meerbergen. The Lanczos method with semi-definite inner product. *BIT*, 41(5):1069–1078, 2001.

[66] K. Meerbergen. The Quadratic Arnoldi method for the solution of the quadratic eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 30(4):1463–1482, 2008.

[67] V. Mehrmann and D. Watkins. Structure-preserving methods for computing eigenpairs of large sparse skew-Hamiltonian/Hamiltonian pencils. *SIAM J. Sci. Comput.*, 22(6):1905–1925, 2001.

[68] MPI Forum. MPI: a message-passing interface standard. *Int. J. Supercomp. Applic. High Perf. Comp.*, 8(3/4):159–416, 1994.

[69] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, NJ, 1980. Reissued with revisions by SIAM, Philadelphia, 1998.

[70] B. N. Parlett and H. C. Chen. Use of indefinite pencils for computing damped natural modes. *Linear Algebra Appl.*, 140(1):53–88, 1990.

[71] B. N. Parlett, D. R. Taylor, and Z. A. Liu. A look-ahead Lánczos algorithm for unsymmetric matrices. *Math. Comp.*, 44(169):105–124, 1985.

[72] J. E. Roman, C. Campos, E. Romero, and A. Tomas. SLEPc users manual. Technical Report DSIC-II/24/02–Revision 3.6, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2015.

[73] E. Romero and J. E. Roman. A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc. *ACM Trans. Math. Software*, 40(2):13:1–13:29, 2014.

[74] M. Rozložník, F. Okulicka-Dluzewska, and A. Smoktunowicz. Cholesky-like factorization of symmetric indefinite matrices and orthogonalization with respect to bilinear forms. *SIAM J. Matrix Anal. Appl.*, 36(2):727–751, 2015.

[75] A. Ruhe. Rational Krylov sequence methods for eigenvalue computation. *Linear Algebra Appl.*, 58:391–405, 1984.

[76] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM Publications, 2nd edition, 2003.

[77] Y. Saad. *Numerical Methods for Large Eigenvalue Problems, Revised Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.

[78] T. Sakurai and H. Sugiura. A projection method for generalized eigenvalue problems using numerical integration. *J. Comput. Appl. Math.*, 159(1):119–128, 2003.

[79] T. Sakurai and H. Tadano. CIRR: a Rayleigh-Ritz type method with contour integral for generalized eigenvalue problems. *Hokkaido Math. J.*, 36(4):745–757, 2007.

[80] K. Schreiber. *Nonlinear Eigenvalue Problems: Newton-type Methods and Nonlinear Rayleigh Functionals*. PhD thesis, Technische Universität Berlin, 2008.

[81] G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst. Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36(3):595–633, 1996.

[82] G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.

[83] G. L. G. Sleijpen, H. A. van der Vorst, and E. Meijerink. Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. *Electron. Trans. Numer. Anal.*, 7:75–89, 1998.

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

[85] Y. Su, J. Zhang, and Z. Bai. A compact Arnoldi algorithm for polynomial eigenvalue problems. talk presented at RANMEP, 2008.

[86] F. Tisseur. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra Appl.*, 309(1–3):339–361, 2000.

[87] F. Tisseur. Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 22(4):1038–1057, 2001.

[88] F. Tisseur. Tridiagonal-diagonal reduction of symmetric indefinite pairs. *SIAM J. Matrix Anal. Appl.*, 26(1):215–232, 2004.

[89] F. Tisseur and K. Meerbergen. The quadratic eigenvalue problem. *SIAM Rev.*, 43(2):235–286, 2001.

[90] R. van Beeumen, K. Meerbergen, and W. Michiels. Compact rational Krylov methods for nonlinear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 36(2):820–838, 2015.

[91] H. Voss. A Jacobi-Davidson method for nonlinear and nonsymmetric eigenproblems. *Comput. & Structures*, 85(17-18):1284–1292, 2007.

[92] D. S. Watkins. *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*. Society for Industrial and Applied Mathematics, 2007.

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

[94] S. Yokota and T. Sakurai. A projection method for nonlinear eigenvalue problems using contour integrals. *JSIAM Letters*, 5:41–44, 2013.