Implementación Paralela del Método de Minimización de la Traza para el Problema de Valores Propios Generalizado Simétrico

Tesis de Máster

Máster en Computación Paralela y Distribuida

Autor: Eloy Romero Alcalde

Dirigido por José E. Román

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
22/9/2008

UNIVERSIDAD POLITECNICA DE VALENCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Índice

Resumen

Los problemas de valores propios son considerados de interés tanto desde el punto de vista teórico como práctico. Están presentes en la resolución de sistemas de ecuaciones diferenciales, análisis de modelos de crecimiento de una población, y cálculo de funciones de matrices, entre otras muchas aplicaciones. En multitud de áreas como física, sociología, biología, economía y estadística se está popularizando este tipo de herramientas y su resolución con el ordenador.

Los problemas de valores propios consisten en, dada una matriz A, hallar pares (λ, v), valor propio y vector propio, de tal forma que $Av = \lambda v$ (o en su versión más general $Av = \lambda Bv$). Es bastante frecuente que se deseen unos pocos valores propios en ciertas regiones, y que tanto A como B sean matrices dispersas de gran tamaño. En estos casos, los métodos iterativos suponen una gran ventaja frente a los métodos directos como la iteración QR o Divide y Vencerás.

En concreto, en este documento se analiza Minimización de la Traza, un método iterativo para resolver problemas generalizados simétricos y definidos positivos de valores propios. Comprobamos cómo ciertas técnicas como las estrategias de desplazamiento, de control de *breakdowns*, estimación del error y precondicionado, aceleran la convergencia y confieren más robustez al algoritmo.

Como resultado final del trabajo, se dispone de una implementación paralela que estará disponible en la librería SLEPc y cuyas prestaciones son analizadas en detalle en este documento.

Muchas de las técnicas que se aplican en Minimización de la Traza, pueden ser usadas en otros métodos iterativos más avanzados como los populares Jacobi-Davidson y LOBPCG.

Agradecimientos

En primer lugar, agradezco el esfuerzo que mis padres, Eloy y Felisa, han dedicado en mi educación. Ellos, mis hermanas y mis amigos me han apoyado incondicionalmente.

Agradezco especialmente a José E. Román y a Andrés Tomás los consejos que me han ofrecido y la oportunidad de trabajar con ellos en proyectos francamente muy interesantes y con una utilidad muy directa.

Gracias también a Vicente Hernández y su grupo GRyCAP por acogerme todo este tiempo y proporcionarme recursos indispensables para el desarrollo del proyecto.

Por último, me queda agradecer los recursos y asistencia proporcionados por el Centro Nacional de Supercomputación (BSC).

Capítulo 1

Introducción

Los problemas de valores propios son considerados de interés tanto desde el punto de vista teórico como práctico. Están presentes en la resolución de sistemas de ecuaciones diferenciales, análisis de modelos de crecimiento de una población, y cálculo de funciones de matrices, entre otras muchas aplicaciones. En multitud de áreas como física, sociología, biología, economía y estadística se está popularizando este tipo de herramientas y su resolución con el ordenador.

En este capítulo presentamos el problema de los valores propios, sus tipos y una perspectiva histórica de los métodos más destacados que se han ido desarrollando hasta nuestros días. Después se exponen los objetivos de la tesis de máster, se describen los detalles de notación empleados y, finalmente, se describe la estructura del resto del documento.

1.1. Problemas de valores propios

Dado un operador lineal T, es decir, que cumple

$$\begin{aligned} T(u+v) &= T(u)+T(v) \ , \text{ y} \\ T(ku) &= kT(u) \ , \end{aligned}$$

siendo u y v vectores y k un escalar, los *vectores propios* de T son aquellos vectores no nulos que al aplicarles el operador T no cambian su dirección, aunque sí pueden sufrir escalamiento en un factor que denominamos *valor propio*. Por ejemplo, los vectores propios asociados a un operador de rotación son aquellos vectores que se encuentran en el eje de rotación, ya que por definición son los únicos que no sufren cambio en su dirección.

De manera formal, se buscan aquellos pares (λ, v) tales que

$$T(v) = \lambda v \ . \tag{1.1}$$

Se observa que si v es un vector propio con el valor propio λ entonces cualquier múltiplo de v diferente de cero, es también un vector propio con el valor propio λ. De hecho, todos los vectores propios con el valor propio asociado λ junto con 0, forman un subespacio V, que recibe el nombre de *espacio propio* de λ. La *multiplicidad geométrica* de un valor propio es la dimensión del espacio propio asociado.

Dado que T corresponde con una transformación lineal, es posible construir una base en su espacio vectorial asociado. Entonces, T y λv pueden representarse en relación a esa base en forma de matriz A y vector columna λv, respectivamente, y la ecuación 1.1 quedaría en la forma habitual en que se presenta el problema de valores propios estándar:

$$Av = \lambda v \ . \tag{1.2}$$

En algunos casos es necesario resolver un problema de tipo

$$Av = \lambda Bv \ , \tag{1.3}$$

al que denominamos problema de valores propios generalizado. Por ejemplo, en el análisis vibratorio de estructuras se calculan las frecuencias más bajas que cumplen la ecuación $K\phi = \lambda M\phi$, donde K es la matriz de rigidez, M es la matriz de masas, y λ y ϕ están directamente relacionados con las frecuencias y los modos naturales de vibración, respectivamente.

En general, λ toma valores en el plano complejo, incluso en el caso en que la matrices sean reales. Sin embargo, si estamos ante un problema simétrico real (o hermitiano, en el caso complejo), como los que se tratan en este documento, λ toma exclusivamente valores reales. Ello simplifica su tratamiento teórico, y los métodos que lo resuelven.

1.2. Problemas grandes y dispersos

En la mayoría de las aplicaciones como pueden ser los problemas de dinámica de estructuras (por ejemplo [Bertolini, 1998]) y electromagnetismo (por ejemplo [Arbenz y Geus, 1999]), las matrices A y B son muy grandes, y *dispersas* (tienen muchos elementos nulos), y sólo son necesarios unos pocos valores y vectores propios, bien en los extremos del espectro (por ejemplo los más grandes o los más pequeños en valor absoluto) o cercanos a un punto en el plano complejo.

Si las matrices fueran de pequeña dimensión, pueden abordarse mediante métodos directos, como el algoritmo iterativo QR o el método *divide y vencerás*. Estos métodos se comentarán en la sección siguiente.

En el caso de matrices de gran dimensión y dispersas, los métodos iterativos son muy interesantes. Por un lado, se puede configurar la precisión de los resultados. Esto es útil cuando los datos de origen tienen un determinado error asociado (y no tiene sentido hallar soluciones de mayor precisión a dicho error), o sencillamente no se requiere toda la precisión que logra un método directo. Entonces, cuando se alcanza el grado de precisión deseado, el método se detiene devolviendo los pares propios convergidos.

Por otro lado, en muchas aplicaciones se dispone de ciertas operaciones que se pueden realizar de manera muy eficiente sobre sus matrices, como el producto matriz-vector en matrices Vandermonde o Toeplitz. En otras, las matrices no son explícitas y sólo están definidas ciertas operaciones, la más común de las cuales suele ser su producto por un vector. Existe una gran variedad de métodos iterativos que únicamente usan esta operación sobre matrices para resolver el problema.

De cualquier forma, en aplicaciones reales es habitual plantear problemas de orden 10^6 o superior. En estos casos, además de emplear los métodos más avanzados, es imprescindible el uso de paralelismo.

1.3. Perspectiva histórica de los métodos de resolución

Gran parte de la computación científica depende críticamente de alguna u otra manera en los algoritmos del álgebra lineal numérica. No solo los problemas de física o ingeniería hacen un uso extensivo de los núcleos computacionales de álgebra lineal, sino también muchas aplicaciones modernas, tales como la recuperación de información y el tratamiento digital de imágenes.

La obtención de valores y vectores propios es mucho más complicada que resolver sistemas lineales, razón por la cual el auge más significativo en este área coincidió con la introducción de los ordenadores sobre 1950.

Las soluciones que proponen la construcción explícita de la ecuación característica, $\det(A-\lambda I) = 0$, en general, no son viables ya que los coeficientes de dicha ecuación no pueden ser calculados de manera numéricamente estable. Y en el caso de que se pudieran, la obtención de sus raíces es un problema muy mal condicionado debido a que pequeñas perturbaciones en los coeficientes pueden generar grandes perturbaciones en las raíces del polinomio.

Aunque se distingue entre métodos directos e iterativos, todos los algoritmos que resuelven problemas de valores propios tienen un carácter inherentemente iterativo: según el teorema de Abel-Ruffini no existe ninguna fórmula para el cálculo de las raíces de un polinomio general de grado mayor que 4. Por tanto, nos centraremos en buscar algoritmos con rápidas tasas de convergencia y que ofrezcan resultados precisos.

La aproximación estándar de las soluciones numéricas de los problemas de valores propios consisten en reducir las matrices involucradas a alguna forma más sencilla que lleve de una manera más o menos directa a los pares propios, como por ejemplo una matriz diagonal. Se prefieren las transformaciones ortogonales tanto como sea posible con el objetivo de reducir los efectos de las perturbaciones.

El caso más sencillo es el simétrico en el cual existe una matriz ortogonal Q tal que $Q^T AQ = D$, donde D es una matriz diagonal. Los elementos en la diagonal de D son los valores propios de A y las columnas de Q corresponden con sus vectores propios asociados.

Existen otras aproximaciones en la literatura que explotan la naturaleza del problema, como por ejemplo las técnicas multi-malla en ecuaciones de derivadas parciales. Sin embargo, no serán tratados en este documento.

1.3.1. Método de Jacobi

El método de Jacobi (consultar [Golub y Van˜Loan, 1996]) fue originalmente propuesto en 1846. Reducía una matriz real y simétrica a una forma diagonal mediante una secuencia de rotaciones de plano. Jacobi, sin embargo, no usaba el método hasta que convergiera totalmente, sino que lo combinaba con el método iterativo de Jacobi para los componentes que faltaban del vector deseado. De hecho, las técnicas de Jacobi pueden ser vistas como formas de precondicionar las iteraciones de Jacobi, también usadas para resolver problemas lineales de mínimos cuadrados.

El éxito del método de Jacobi para diagonalizar matrices simétricas usando transformaciones de semejanza ortogonales inspiraron otras muchas técnicas para matrices no simétricas, como la descomposición de Schur. También la iteración QR puede verse como un método de Jacobi: reduce la matriz a una forma de Schur vía una secuencia de transformaciones de semejanza compuestas de rotaciones.

En los años sesenta empezaron a ser más populares los métodos de Givens y Householder que reducen las matrices a una forma tridiagonal, para luego emplear métodos eficientes que calculasen los pares propios (ver apartado 1.3.3).

Durante los setenta y parte de los ochenta volvió un cierto interés por el método de Jacobi con el auge de las máquinas paralelas [Sameh, 1971]. Fueron propuestas variantes del método para matrices normales [Paardekooper, 1971] y no-normales [Eberlein, 1970]. Además se propusieron extensiones a bloques para mejorar la localidad de datos en máquinas de memoria distribuida [Blackford et al., 1997].

1.3.2. Método de la potencia

El método de la potencia es el más simple de todos para resolver problemas del tipo tratado. La idea básica consiste en que al aplicar reiteradamente la matriz del sistema A a un vector inicialmente bien seleccionado, éste converge al vector propio correspondiente al valor propio dominante (el de mayor módulo).

La convergencia del método depende del ratio entre el primer y el segundo valor propio más grande (en valor absoluto). Su baja tasa de convergencia ha llevado a proponer alternativas, la más efectiva de las cuales es el método de la potencia inversa propuesto en 1944 por Wielandt, que trabaja con la matriz $(A - \mu I)^{-1}$ (μ desplaza los valores propios de A sobre la recta real, razón por la cual μ es llamada *desplazamiento*). Este autor propuso también técnicas de deflación implícitas (trasformar el operador de manera que no incluyera el vector propio recién convergido) para calcular más de un valor propio [Parlett, 1980].

Ninguno de los métodos (el de la potencia y la potencia inversa) es competitivos incluso si se quieren calcular sólo unos pocos pares propios. Sin embargo siguen siendo de interés ya que directa o indirectamente forman parte de métodos más modernos. En este sentido, la secuencia de vectores que genera el método de la potencia expande subespacios de Krylov que van aumentando de dimensión, subespacios que juegan un papel importante, por ejemplo, en métodos como Lanczos o Arnoldi (explicados en el apartado 1.3.4).

Otra importante mejora sobre la potencia inversa consiste en trabajar con desplazamientos adecuadamente actualizados. En concreto si empleamos el cociente de Rayleigh de los vectores

aproximados obtenemos el método de la Iteración del cociente de Rayleigh (*Rayleigh Quotient Iteration*, RQI). Ostrowski, estudió su convergencia y la estableció en cúbica tanto para sistemas simétricos como no simétricos (con una adecuada generalización del coeficiente de Rayleigh para este último caso). Estos resultados son esenciales para entender métodos iterativos más modernos que están basados en aproximaciones de estrategias de desplazamiento e inversión (*shift-and-invert*), como por ejemplo el método de Jacobi-Davidson.

Otra forma de obtener más de un valor propio consiste en iterar un conjunto de vectores en vez de un sólo vector. Para evitar que todos los vectores converjan al dominante, se mantiene linealmente independiente el conjunto de vectores. Si usamos la descomposición QR para mantener la ortogonalidad de los vectores en el método de la potencia, resulta el método llamado Iteración del Subespacio (que será comentado en el capítulo 2).

1.3.3. Métodos de reducción

En 1954 Givens se dio cuenta del hecho que las matrices se podrían reducir por transformaciones de semejanza ortogonales en un número finito de pasos a formas en las que se pudieran aplicar algoritmos más eficientes. En concreto, las matrices simétricas pueden ser reducidas a una forma tridiagonal empleando rotaciones de Jacobi (también llamadas a partir de ese momento rotaciones de Givens). Householder en 1958 propuso lo mismo, pero con una técnica más eficiente basada en actualizaciones de rango uno (llamadas reflectores de Householder).

El uso de la descomposición QR empieza con los trabajos de Francis (ver [Francis, 1961]) que en 1961 reconocían que la iteración QR transformaba las matrices Hessenberg en otras del mismo tipo, y esto hacía el proceso muy económico y estable, ya que los elementos nulos no debían de calcularse. Además aceleró el método con desplazamientos implícitos que se pueden aplicar a matrices Hessenberg de manera muy eficiente. Su convergencia (extraída de los trabajos de Ostrowski sobre el RQI) para matrices simétricas es cúbica, mientras que para las no simétricas es cuadrática.

Las implementaciones modernas de la iteración QR incorporan técnicas de múltiples desplazamientos (usan un desplazamiento diferente para cada valor propio) implícitos, desarrolladas en los años noventa. Los desplazamientos implícitos anteriormente se implementaban como actualizaciones de rango uno, pero técnicas más recientes como la llamada *bulge-chasing* permiten emplear rotaciones de Givens.

En 1970 se consideraban resueltos los problemas de valores propios de matrices densas y no muy grandes, y empezaron a plantearse métodos para abordar problemas generalizados y cuadráticos. En este caso las primeras aproximaciones consistieron en transformar el problema generalizado en otro estándar equivalente, bien invirtiendo la matriz A, la B, o aplicando transformaciones más complicadas como la de desplazamiento e inversión o la de Cayley.

Aunque no se esperaban mejoras significativas en problemas simétricos, Cuppen [1980] propuso una versión de divide y vencerás para matrices tridiagonales simétricas. Es considerado por Demmel como uno de los algoritmos más rápido para matrices que no sean pequeñas (en caso contrario recomienda emplear la iteración QR). También se están popularizando las implementaciones del método *Relative Robust Representation* [Parlett y Dhillon, 2000], que están superando en prestaciones a los anteriores métodos en matrices provenientes de aplicaciones industriales [Demmel et al., 2008].

Sin embargo, siguen habiendo nichos para otros algoritmos. Wilkinson aboga por el método de la bisección sobre matrices tridiagonales si sólo se requiere unos pocos valores propios. La iteración inversa puede ser empleada para obtener los correspondientes vectores propios.

1.3.4. Métodos iterativos

En 1950, Lanczos sugirió construir una base del subespacio de Krylov de manera estable ortogonalizando los vectores conforme se fueran obteniendo. Poco después Wilkinson mostró que seguía siendo altamente inestable y parecía que no hubiera otro remedio que reortogonalizar de nuevo los vectores generados. Comparó esta opción con Householder y Givens (métodos de los años cincuenta) y concluyó que eran más económicos estos últimos. Lanczos todavía era visto como un método directo.

Paige, años más tarde, en 1971 estudió profundamente el método y demostró que podría ser empleado de manera auténticamente iterativa para obtener aproximaciones buenas de unos pocos valores propios [Paige, 1971]. El hecho significativo que observó fue que la pérdida de ortogonalidad, el origen de todos los problemas del método, marca la convergencia de un par propio. Y lo que es más importante, esto no afectaba a la convergencia del resto. Esta pérdida de ortogonalidad debida a la introducción de nuevo de pares convergidos, conllevaba la duplicación de pares convergidos en la matriz tridiagonal reducida. En el resto de pares, el principal efecto era cierto retardo del proceso en computación exacta. Sus estudios motivaron la actividad del campo, y finalmente resultaron a partir de 1980 en un método de Lanczos competitivo para problemas grandes, simétricos y dispersos.

Además Van der Sluis y Van der Vorst [van˜der Sluis y van˜der Vorst, 1987] demostraron que la convergencia de los valores de Ritz en el método era superlineal. No importaba cómo de irregular fuera el ratio de convergencia, de media se volvía cada vez más rápido en cada iteración.

Arnoldi propuso un método parecido en 1951, pero para matrices no simétricas en el que como diferencia fundamental el vector generado tenía que ortogonalizarse contra todos los vectores de la base. Era más estable numéricamente, pero más costoso que la reducción de Householder, haciéndolo también menos atractivo como método directo para matrices grandes y densas.

Debió pasar algún tiempo hasta que algún método se hizo popular para el cálculo de matrices no simétricas. Destacaremos el método de Lanczos por los dos lados que tenía un coste en el peor de los casos similar a la reducción de Householder si se quisieran obtener todos los pares. También se propusieron mejoras al método de Arnoldi, como un precondicionamiento polinómico, y sobre todo en técnicas de reinicio más sofisticadas. De entre ellas cabe destacar la técnica de reinicio implícita de Sorensen [1992], que permitía continuar el proceso con una base de un subespacio reducido (a diferencia de los métodos clásicos de reinicio que continuaban con un único vector). En [Stewart, 2001] se presenta el método de Krylov-Schur, que usa una técnica simple de reinicio que logra el mismo efecto que el reinicio implícito, y de manera numéricamente estable.

En 1975, Davidson propuso un método iterativo que realizaba una proyección sobre un subespacio de manera parecida a Arnoldi, pero utilizando un subespacio diferente. Davidson obtenía los pares de Ritz (los pares propios aproximados), calculaba sus residuos y expandía el subespacio con el vector $(D_A - \theta I)^{-1} r$ (D_A es la diagonal de A, θ es un valor propio y r su residuo) tras la ortogonalización con respecto al subespacio actual. Justamente, para matrices diagonalmente dominantes (que eran muy frecuentes en química, su disciplina), esto se aproximaba a la iteración inversa con desplazamientos obtenidos del cociente de Rayleigh. Más allá de las matrices diagonalmente dominantes y una buena elección de los vectores de inicio, la convergencia estaba lejos de ser garantizada. El método no fue entendido desde el punto de vista del análisis numérico hasta los años noventa [Crouzeix et al., 1994] en los que sufrió importantes mejoras.

En 1996, Sleijpen y van der Vorst [Sleijpen et al., 1996] sugirieron restringir la expansión del subespacio actual al subespacio ortogonal a z, idea procedente de Jacobi. Jacobi resolvía la ecuación de corrección para la matriz A desplazada por el valor propio θ y restringido al espacio ortogonal al vector propio (ecuación de corrección). Esta ecuación era resuelta mediante iteraciones de Jacobi y tras dos iteraciones, Jacobi actualizaba el valor de θ. Sleijpen y van der Vorst sugirieron actualizar z, y de paso también, el subespacio. A esta combinación de Davidson junto a la idea de Jacobi de actualizar en el subespacio ortogonal a z fue bautizada como método de Jacobi-Davidson. La solución exacta de la ecuación de corrección, o una aproximación de mucha precisión llevan (si se seleccionan adecuadamente los desplazamientos) a una convergencia cúbica para sistemas simétricos y cuadrática para no simétricos.

Varios años después se propusieron técnicas eficientes para aplicar precondicionamiento (aunque todavía sigue como problema abierto identificar precondicionadores efectivos para muchas clases de matrices) y se extendió el método a problemas generalizados. Jacobi-Davidson es un método atractivo para sistemas grandes y dispersos en los que las operaciones de desplazamiento e inversión son costosas.

Volviendo a los años ochenta, para obtener los valores propios más pequeños y los correspondientes vectores propios en problemas generalizados, las generalizaciones de Lanczos a bloques y la Iteración del Subespacio necesitaban resolver en cada iteración un sistema lineal con la matriz B con mucha

precisión. En [Sameh y Wisniewski, 1982] se presenta el método de Minimización de la Traza que trataba de evitar este inconveniente. Años más tarde, A. Sameh y Z. Tong presentaron una versión del método enmarcada en los métodos de tipo Davidson [Sameh y Tong, 2000] que pretende competir con otros métodos de la misma familia como Jacobi-Davidson. Ambas versiones de Minimización de la Traza serán expuestas ampliamente en este documento.

Por último cabe destacar la labor de Knyazev que ofreció una excelente exposición de los trabajos rusos desarrollados sobre métodos precondicionados para problemas de valores propios (ver *Preconditioned eigensolvers* en [Bai et al., 2000]). Este estudio es relevante para la compresión de las formas inexactas o precondicionadas del desplazamiento e inversión en Lanczos, el método inexacto de Arnoldi y ciertas variantes de los métodos de Davidson, como Jacobi-Davidson. El método LOBPCG [Knyazev, 2001] propuesto por este autor se está popularizando e implementando en las librerías más modernas para problemas de valores propios dispersos. En la sección 6.5.1 se encuentra una descripción más detallada del mismo.

1.4. Objetivos de la tesis de máster y trayectoria

El objetivo principal de esta tesis consiste en el estudio e implementación del método de Minimización de la Traza de manera eficiente y robusta en entornos paralelos. Para alcanzarlo, se ha de llevar a cabo

- un estudio del método que tiene como objetivos, por un lado, plantear un diseño que permita una buena implementación, y por otro, detectar los principales inconvenientes del método y proponer soluciones;
- la implementación de las versiones más prometedoras del método; y finalmente,
- un estudio de las prestaciones y una comparación con otros métodos que se hallen en el estado del arte.

Cabe destacar que no existen implementaciones de Minimización de la Traza de este tipo, a excepción de la librería SVDPACK que implementa la versión clásica en secuencial.

Mi Proyecto Final de Carrera consistió en un estudio más aislado del método (en su versión clásica) y una implementación tanto en secuencial (en Matlab) como un prototipo en distribuido bajo la librería SLEPc. Los estudios y desarrollos del PFC sirvieron de base para esta tesis que abarca una visión del método dentro del estado del arte de la computación numérica. Sus aspectos más novedosos fueron presentados en la octava edición del congreso internacional VECPAR, y serán publicados en las actas en papel [Romero y Roman, 2008].

A su vez, estos desarrollos servirán de base a la tesis doctoral que abarcará otros métodos de gran importancia y popularidad para resolver problemas de valores propios grandes y dispersos, como los métodos de Davidson (comentados anteriormente) o los basados en Gradiente Conjugado, como el LOBPCG.

1.5. Notación y propiedades básicas de álgebra lineal

A lo largo del documento se usarán letras mayúsculas para las matrices: A, Λ, V; letras minúsculas y latinas para los vectores: x, v; y letras minúsculas y griegas para los escalares: λ, μ. Cuando una matriz o vector en un algoritmo varía en cada iteración, denotamos su estado con el subíndice de la iteración, V_k. Para indicar una columna de una matriz, se pone también como subíndice, separándola con coma si también se indica la iteración concreta $V_{k,i}$.

También se usarán las siguientes notaciones y abreviaciones de funciones:

- U^T es U traspuesta, y U^* es U traspuesta y conjugada.
- $u^*v = \langle u, v\rangle$, denota el producto escalar de u y v; y $u^*Av = \langle u, v\rangle_A$ denota el producto escalar de u y v con respecto a la matriz A.

- $u^*V = 0 \rightarrow u \perp V$, y $u^*AV = 0 \rightarrow u \perp_A V$
- I es la matriz identidad de tamaño adecuado, e I_p es una matriz identidad de tamaño p.
- $\mathcal{M}^{n\times m}$ es el conjunto de todas las matrices de n filas y m columnas.
- rango(A) indica el rango de la matriz A, es decir, el máximo número de filas o columnas linealmente independientes de A.
- $\lambda_i(A)$ denota el i-ésimo valor propio de A ordenados de menor a mayor.
- $\det(A)$ es el determinante de A.
- $\|u\|$ es la 2-norma del vector u, y $\|u\|_A = \sqrt{u^*Au}$.
- $|V|$ es el número de columnas de la matriz V.
- $\mathrm{tr}(A) = \sum a_{ii}$ es la traza de la matriz A, es decir, la suma de sus elementos en la diagonal, a_{ii}.
- $A/B = AB^{-1}$.
- $\lambda_{\text{máx}}(A)$ y $\lambda_{\text{mín}}(A)$ son el valor propio más grande y más pequeño de la matriz A, respectivamente.

Seguidamente se definen las propiedades más importante sobre las matrices:

- A es simétrica si $A = A^T$, y hermitiana si $A = A^*$.
- A es autoadjunta si para cualquier par de vectores u y v de tamaños adecuados se cumple que $\langle u, Av\rangle = \langle Au, v\rangle$.
- A es normal si $A^*A = AA^*$.
- A es ortogonal si $AA^T = A^TA = I$, y es unitaria si $A^*A = AA^* = I$.
- A es singular si existe su inversa, es decir, si todos sus valores propios son distintos de cero.
- A es definida positiva si (siendo hermitiana) todos sus valores propios son mayores que cero, y semidefinida positiva si todos sus valores propios son mayores o iguales que cero.

Por último, describimos dos teoremas que son referidos en el documento.

Teorema de Courant-Fischer Dada una matriz simétrica A de tamaño n con los valores propios ordenados $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ resulta que los valores propios son los valores estacionarios del cociente de Rayleigh:

$$\lambda_{n-k+1} = \max_{S\in\mathcal{M}^{n\times k}} \min_{0\neq y\in\mathrm{span}\{S\}} \frac{y^*Ay}{y^*y}$$

Desigualdad de Kantorovich Supongamos la lista de números $0 < \alpha_1 < \alpha_2 < \cdots < \alpha_n$, y $\lambda_1, \ldots, \lambda_n \geq 0$ cumpliendo $\sum_{j=1}^n \lambda_j = 1$. Entonces

$$\left(\sum_{j=1}^n \lambda_j\alpha_j\right)\left(\sum_{j=1}^n \lambda_j\alpha_j^{-1}\right) \leq \left(\frac{\alpha_1+\alpha_n}{2}\right)^2 (\alpha_1\alpha_n)^{-1} \quad .$$

1.6. Estructura de la memoria

El capítulo 2 se centra en el algoritmo de Minimización de la Traza en sus versiones clásica y de tipo Davidson. El capítulo 3 resume otros aspectos a considerar que no son propiamente del método como la ortogonalización, las estrategias de desplazamiento o el precondicionado del sistema lineal interno. Estos tres primeros capítulos encierran los aspectos más teóricos de tipo numérico.

Se sigue en el capítulo 4 con una introducción al campo de la computación de altas prestaciones y su aplicación a la resolución de problemas de valores propios. También se hace un pequeño estudio de las librerías más importantes para resolver estos problemas. El capítulo 5 resume los aspectos prácticos considerados durante la implementación.

Después, la descripción de los experimentos que por un lado comprueban la efectividad de las optimizaciones propuestas y por otro miden las prestaciones de la implementación, son recogidos en el capítulo 6 junto a los resultados.

Por último se encuentran las conclusiones de la tesis que son expuestas en el capítulo 7 y al final se adjunta la bibliografía referida durante el documento así como las referencias más importantes consultadas durante su elaboración.

Capítulo 2

Minimización de la traza

Minimización de la Traza es un método orientado a obtener los valores propios más pequeños en problemas simétricos generalizados. A continuación se describe de manera similar a como se hace en [Sameh y Wisniewski, 1982] y [Sameh y Tong, 2000].

2.1. Teorema de Minimización de la Traza

Tal y como se ha dicho en la introducción, se parte de un sistema propio generalizado con matrices A y B simétricas, y B semidefinida positiva. Además, en todo el documento se supondrá que A es definida positiva, requisito que no es muy estricto ya que, en el caso de que A no cumpliera esta condición, se puede resolver, en vez, el sistema equivalente:

$$(A - \mu B)x = (\lambda \ - \mu)Bx \ ,$$

con $\mu < \lambda_1 < 0$ (siendo λ_1 el menor valor propio del sistema original) haciendo que $A - \mu B$ sea definida positiva.

En [Beckenbach y Bellman, 1965] y [Sameh y Wisniewski, 1982] se presenta un teorema que se resume en la siguiente ecuación:

$$\min_{\substack{X \in \mathcal{M}^{n \times p} \\ X^T B X = I_p}} \operatorname{tr}(X^T A X) = \sum_{i=1}^{p} \lambda_i \ , \tag{2.1}$$

siendo $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ los valores propios del problema, y $1 \leq p \leq n$. La igualdad es únicamente alcanzada por aquellas matrices cuyas columnas generan el subespacio propio asociado a los p valores propios más pequeños.

También podemos reescribir la ecuación anterior de otra manera:

$$\min_{\substack{X \in \mathcal{M}^{n \times p} \\ \operatorname{rango}(X) = p}} \operatorname{tr}\left(\frac{X^T A X}{X^T B X}\right) = \sum_{i=1}^{p} \lambda_i \ . \tag{2.2}$$

El cociente $X^T A X (X^T B X)^{-1}$ es más conocido como cociente generalizado de Rayleigh, y muchos métodos se basan, implícita o explícitamente, en reducirlo paso a paso. Uno de los más sencillos es la Iteración del Subespacio: a partir de una aproximación X_k este método obtiene la siguiente aproximación como $X_{k+1} = A^{-1} B X_k$, que usando el teorema de Courant-Fischer y la desigualdad de Kantorovich se demuestra que

$$\lambda_i \left(\frac{X_{k+1}^T A X_{k+1}}{X_{k+1}^T B X_{k+1}}\right) \leq \lambda_i \left(\frac{X_k^T A X_k}{X_k^T B X_k}\right) \quad , \quad 1 \leq i \leq p \ . \tag{2.3}$$

La igualdad tiene lugar cuando las columnas de X_k generan el subespacio propio del problema. Por último, el algoritmo se completa realizando un proceso de Rayleigh-Ritz (explicado en la sección 2.2.2) sobre X_{k+1} logrando mejores aproximaciones que usando directamente X_{k+1} en la siguiente iteración. El algoritmo 1 muestra todo el proceso.

Algoritmo 1 (Iteración del subespacio)

Entrada: matrices A y B,
número de pares propios que se desean p
tamaño del subespacio m, $p \leq m \leq n$
Salida: pares propios obtenidos

Elegimos una matriz V_1 de $n \times m$ tal que $V_1^T B V_1 = I_m$
Para $k = 1, 2, ...$ hasta convergencia
1. Obtener $W_k \leftarrow AV_k$ y $H_k \leftarrow V_k^T W_k$.
2. Obtener los pares propios (Θ_k, Y_k) de H_k.
3. Obtener los vectores de Ritz $X_k \leftarrow V_k Y_k$.
4. Obtener los residuos $R_k \leftarrow W_k Y_k - B X_k \Theta_k$.
5. Comprobar convergencia.
6. Resolver el sistema lineal $AZ_{k+1} = BX_k$.
7. $V_{k+1} \leftarrow B$-ortogonalizar Z_{k+1} mediante Gram-Schmidt.

Fin para

Los pasos del 1 al 3 corresponden con el procedimiento de Rayleigh-Ritz, que será explicado en la sección 2.2.2. En el paso 2 se resuelve el problema de valores propios estándar denso $H_k Y_k = Y_k \Theta_k$. El algoritmo requiere que los valores propios (los elementos de la diagonal de la matriz Θ_k) estén ordenados de manera ascendente y que los vectores propios (las columnas de Y_k) sean ortogonales. Estas restricciones en el procedimiento de Rayleigh-Ritz son comunes en el resto de algoritmos detallados en el documento, y serán asumidas cuando se resuelva el correspondiente sistema propio.

Cabe destacar también que Gram-Schmidt es un procedimiento para ortogonalizar un conjunto de vectores y será explicado en la sección 3.4.

Continuando con el algoritmo, en cada iteración se requiere resolver p sistemas de ecuaciones lineales (uno por cada columna de X_k), y además han de resolverse con mucha precisión (de lo contrario se compromete la convergencia del algoritmo), lo cual es significativamente costoso. El algoritmo de Minimización de la Traza explota directamente el teorema 2.1, de tal forma que la convergencia no se comprometa aún cuando el sistema lineal no sea resuelto con mucha precisión.

2.2. Algoritmo básico

El esquema de Minimización de la Traza es similar a Iteración del Subespacio (algoritmo 1), excepto que las aproximaciones X_k tienden a reducir en cada iteración la expresión $\mathrm{tr}(X_k^T A X_k)$. Para ello, se obtiene Z_{k+1} como $X_k - \Delta_k$ de tal forma que

$$\begin{aligned} &\text{minimice } \mathrm{tr}\left[(X_k - \Delta_k)^T A (X_k - \Delta_k)\right] \ , \\ &\text{con } X_k^T B \Delta_k = 0 \ . \end{aligned} \tag{2.4}$$

De esta manera, se puede demostrar que Z_{k+1} y X_{k+1} satisfacen

$$\mathrm{tr}(Z_{k+1}^T A Z_{k+1}) \leq \mathrm{tr}(X_k^T A X_k) \ , \text{ y} \tag{2.5}$$

$$\mathrm{tr}(X_{k+1}^T A X_{k+1}) \leq \mathrm{tr}(X_k^T A X_k) \tag{2.6}$$

si se toma X_{k+1} como una base B-ortogonal del subespacio generado por las columnas de Z_{k+1}.

Se puede reescribir el problema de minimización con restricciones 2.4 como el problema de punto de silla (*saddle-point*)

$$\begin{bmatrix} A & BX_k \\ X_k^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta_k \\ L_k \end{bmatrix} = \begin{bmatrix} AX_k \\ 0 \end{bmatrix} \ , \tag{2.7}$$

donde $2L_k$ representa los multiplicadores de Lagrange. Este sistema a su vez se puede reducir a el sistema lineal semidefinido positivo (como se expone en la sección 2.2.1)

$$(P_k A P_k)\Delta_k = P_k A X_k \ , \quad \text{con } X_k^T B \Delta_k = 0 \ , \tag{2.8}$$

donde P_k es el proyector ortogonal

$$P_k = I - BX_k(X_k^T B^2 X_k)^{-1} X_k^T B \ . \tag{2.9}$$

En este documento nos centramos en emplear métodos de Krylov para resolver el sistema lineal ya que, además de sus buenas propiedades de convergencia, se cumple automáticamente la restricción $X_k^T B\Delta_k = 0$ si se parte de una solución inicial nula. El algoritmo 2 compila los pasos generales del método.

Algoritmo 2 (Minimización de la traza)

Entrada: matrices A y B de tamaño $n \times n$,
número de pares propios que se desean p,
tamaño del subespacio m, $p \leq m \leq n$
Salida: pares propios obtenidos

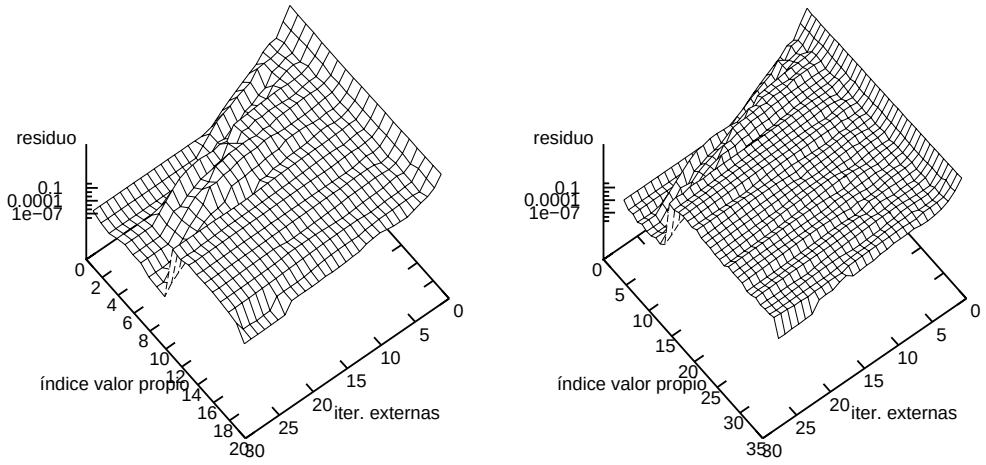
0. Elegir una matriz V_1 de $n \times m$ tal que $V_1^T B V_1 = I_m$.
Para $k = 1, 2, \ldots$ hasta convergencia
1. Obtener $W_k \leftarrow AV_k$ y $H_k \leftarrow V_k^T W_k$.
2. Obtener los pares propios (Θ_k, Y_k) de H_k.
3. Obtener los vectores de Ritz $X_k \leftarrow V_k Y_k$.
4. Obtener los residuos $R_k \leftarrow W_k Y_k - BX_k\Theta_k$.
5. Comprobar convergencia.
6. Resolver $(P_k A P_k)\Delta_k = P_k A X_k$ con $X_k^T B\Delta_k = 0$ aproximadamente.
7. $V_{k+1} \leftarrow B$-ortonormalizar $X_k - \Delta_k$ mediante Gram-Schmidt.
Fin para

La diferencia entre Iteración del Subespacio y Minimización de la Traza reside principalmente en el sistema a resolver en el paso 6. En la práctica, la resolución de los sistemas de ambos algoritmos consumen tiempos similares. El coste adicional de aplicar el proyector en cada iteración del método lineal, se amortiza si consideramos que $P_k A P_k$ está mejor condicionado que A (como muestra el teorema 2.3 en [Sameh y Tong, 2000]), y por tanto necesita menos iteraciones. A estas iteraciones las llamaremos *iteraciones internas*, y al sistema lineal del paso 7, *sistema interno*.

Cabe destacar que en el algoritmo 2 todos los vectores propios aproximados en V_k se actualizaban en cada iteración externa (iteración del bucle más externo). Sin embargo, sólo unos pocos pares propios de los más pequeños mejoran significativamente, como se puede apreciar en la figura 2.1, que muestra un ejemplo de cómo evoluciona el residuo de los pares propios. Por tanto, se puede considerar que es ineficiente llevar a cabo todos los pasos (sobre todo, resolver el sistema lineal en el paso 6 y la ortonormalización del paso 7, que son los pasos más costosos) para el resto de pares propios. Sería más adecuado un método en el cual el espacio de búsqueda V fuera creciendo conforme el algoritmo avanza. La sección 2.3 presenta un algoritmo con tales características.

Por último, a continuación se comentan varios teoremas sobre la convergencia del método. En primer lugar, si el sistema interno 2.8 fuera resuelto exactamente, es decir

$$\Delta_k \leftarrow X_k - A^{-1}BX_k(X_k^T BA^{-1}BX_k)^{-1} \ , \tag{2.10}$$



(a) $m = 20$ *(b)* $m = 35$

Figura 2.1: Norma de los residuos de las aproximaciones de los vectores propios del algoritmo 2 resolviendo el problema BCSST08, pidiendo los 10 pares propios más pequeños y empleando el precondicionador *boomeramg* de HYPRE.

entonces el algoritmo sería matemáticamente idéntico a Iteración del Subespacio. Como consecuencia, la convergencia global de Minimización de la Traza en aritmética exacta viene dada por la convergencia global de Iteración del Subespacio, tal y como se describe en [Rutishauser, 1970], [Parlett, 1980] y [Sameh y Wisniewski, 1982], que aseguran que el error

$$\phi = (x_{k,i} - x_i)^T A (x_{k,i} - x_i) \tag{2.11}$$

se reduce asintóticamente en un factor de $(\lambda_i/\lambda_{m+1})^2$, siendo x_i el vector propio exacto asociado al problema 1.3 y $x_{k,i}$ la columna i de la matriz X_k.

Por otro lado, en [Sameh y Tong, 2000] se prueba la convergencia del algoritmo considerando que el sistema interno es resuelto de manera inexacta, aunque asumiendo que el residuo será reducido siempre en un factor menor que 1. La demostración pasa, por un lado, por mostrar que en cada iteración las columnas de $X_k - \Delta_k$ son linealmente independiente; y por otro, que la corrección obtenida Δ_k satisface

$$\operatorname{tr}\left[(X_k - \Delta_k)^T A (X_k - \Delta_k)\right] \leq \operatorname{tr}(X_k^T A X_k) \ . \tag{2.12}$$

Esto asegura, sin importar cómo de prematuramente termine el *solver* lineal, que la traza de $X_k^T A X_k$ forme una secuencia decreciente con límite inferior $\sum_{i=1}^{m} \lambda_i$.

2.2.1. Reducción del problema del punto de silla

Como se ha comentado previamente, el método de Minimización de la Traza se fundamenta en reducir el problema de minimización con restricciones 2.4 en un sistema de ecuaciones lineales, 2.8. En esta sección se describe brevemente tal proceso de reducción como se muestra en [Sameh y Wisniewski, 1982].

El problema 2.4 es un caso apropiado para ser abordado mediante multiplicadores de Lagrange: es un problema de minimización, pero con una restricción adicional. Para facilitar la descripción se consideran las columnas separadamente, con lo que el problema se puede reescribir de esta manera:

$$\text{minimizar} \quad (x_j - d_j)^T A (x_j - d_j) \tag{2.13}$$

$$\text{con} \quad X_k^T B d_j = 0 \ , \tag{2.14}$$

donde $d_j = \Delta_k e_j$ y $x_j = X_k e_j$, es decir, las columna j-ésimas de sus respectivas matrices.

Entonces, la función lagrangiana queda así:

$$\Lambda(d_j, l) = (x_j - d_j)^T A(x_j - d_j) - l^T X_k^T B d_j \ . \tag{2.15}$$

Siguiendo el método, derivamos Λ respecto de d_j y l, e igualamos a 0:

$$\begin{aligned} \frac{\partial \Lambda}{\partial d_j} &= -2A(x_j - d_j) - BX_k l = 0 \rightarrow \\ &\rightarrow A(x_j - d_j) - BX_k l' = 0 \qquad (2.16) \\ \frac{\partial \Lambda}{\partial l} &= X_k^T B d_j = 0 \ , \qquad (2.17) \end{aligned}$$

con $-2l' = l$. Este sistema se puede reescribir como

$$\begin{bmatrix} A & BX_k \\ X^T B & 0 \end{bmatrix} \begin{bmatrix} d_j \\ l' \end{bmatrix} = \begin{bmatrix} Ax_j \\ 0 \end{bmatrix} \ . \tag{2.18}$$

Ahora, se calcula la factorización QR de BX_k,

$$BX_k = QR = [Q_1, Q_2]R \ , \tag{2.19}$$

donde $R^T = [R'^T, 0]$, con R' matriz cuadrada triangular superior de orden m, y Q ortogonal, siendo Q_1 de tamaño $n \times m$. Sustituyendo lo anterior en 2.18 resulta

$$\begin{bmatrix} Q^T AQ & R \\ R^T & 0 \end{bmatrix} \begin{bmatrix} g \\ l' \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \ , \tag{2.20}$$

siendo $g = Q^T d_j$ y $f = Q^T A x_j$. Si tomamos $g^T = [g'^T, \tilde{g}^T]$ con g' de orden m, entonces la ecuación $R^T g = 0$ y la no singularidad de R' implican que $g' = 0$. Por tanto,

$$Q^T AQ \begin{bmatrix} 0 \\ \tilde{g} \end{bmatrix} + \begin{bmatrix} R'l' \\ 0 \end{bmatrix} = f \ , \tag{2.21}$$

es decir,

$$\begin{aligned} Q_1^T AQ_2 \tilde{g} + R'l' &= Q_1^T A x_j \ , \qquad (2.22) \\ Q_2^T AQ_2 \tilde{g} &= Q_2^T A x_j \ . \qquad (2.23) \end{aligned}$$

Como se está interesado sólo en d_j, será suficiente con resolver la ecuación 2.23. Teniendo en cuenta que $P_k \equiv Q_2 Q_2^T$, $Q_2^T Q_2 = I_m$, y $d_j = Qg = Q_2 \tilde{g}$, se puede transformar 2.23 en 2.8:

$$\begin{array}{lrcl} & Q_2^T AQ_2 \tilde{g} &=& Q_2^T A x_j \\ [\text{premultiplicar por } Q_2] & Q_2 Q_2^T AQ_2 \tilde{g} &=& Q_2 Q_2^T A x_j \\ [\text{usar } P_k \equiv Q_2 Q_2^T] & P_k AQ_2 \tilde{g} &=& P_k A x_j \\ [\text{usar } Q_2^T Q_2 = I_s] & P_k AQ_2 Q_2^T Q_2 \tilde{g} &=& P_k A x_j \\ [\text{usar } P_k \equiv Q_2 Q_2^T] & P_k APQ_2 \tilde{g} &=& P_k A x_j \\ [\text{usar } d_j = Q_2 \tilde{g}] & P_k AP_k d_j &=& P_k A x_j \end{array}$$

2.2.2. Procedimiento de Rayleigh-Ritz

Este procedimiento, en principio, no es necesario para la convergencia del método pero la mejora a muy bajo coste. El procedimiento de Rayleigh-Ritz obtiene k pares propios aproximados $(\tilde{\lambda}_i, \tilde{x}_i)$, $1 \leq i \leq k$ de una matriz A, de tal forma que los vectores propios $\tilde{x}_i$ pertenezcan a un subespacio $\mathcal{K}$ de dimensión m dado. Además, los pares propios deben satisfacer la condición de Ritz-Galerkin

$$A\tilde{x}_i - \tilde{\lambda}_i \tilde{x}_i \perp \mathcal{K} \ , \tag{2.24}$$

es decir, que los residuos de los pares propios sean ortogonales al subespacio $\mathcal{K}$. Si se dispone de una base ortonormal $V = \{v_1, \ldots, v_m\}$ de $\mathcal{K}$, se puede escribir los vectores propios en función de la base

$$\tilde{x}_i = V y_i \ . \tag{2.25}$$

Entonces se puede reescribir la condición 2.28 de la forma

$$V^T(A\tilde{x}_i - \tilde{\lambda}_i \tilde{x}_i) = 0 \ . \tag{2.26}$$

Sustituyendo 2.25 en 2.26 resulta:

$$V^T(AVy_i - \tilde{\lambda}_i V y_i) = 0 \ \rightarrow \ V^T A V y_i = \tilde{\lambda}_i y_i \ . \tag{2.27}$$

Por tanto, es posible calcular los pares propios con la condición 2.28 si se hallan los pares propios de $V^T AV$, y se transforman los vectores propios según 2.25.

Este procedimiento es usado en muchos otros métodos (además de los comentados en el capítulo anterior) como por ejemplo en métodos de Krylov. Desde el punto de vista del coste, es un procedimiento muy interesante cuando se requiere obtener unos pocos (k) valores propios de una matriz grande ($n \times n$): consigue reducir el problema a otro en el que se calculan k valores propios de una matriz de $m \times m$, siendo m la dimensión del subespacio. Típicamente la dimensión del subespacio suele ser el doble de k.

El siguiente algoritmo resume lo anterior:

Algoritmo 3 (Procedimiento de Rayleigh-Ritz)

Entrada: A, matriz de la cual se quieren calcular k pares propios
V, base ortonormal de $\mathcal{K}$
Salida: k pares de Ritz

1. Calcular $C \leftarrow V^T AV$.
2. Calcular los pares propios de C, $(\tilde{\lambda}_i, y_i)$.
3. Seleccionar del paso 2 los k valores propios de interés $\tilde{\lambda}_j$.
4. Devolver los pares de Ritz $(\tilde{\lambda}_j, Vy_j)$.

En el problema generalizado, la condición de Ritz-Galerkin 2.28 se reescribe como

$$A\tilde{x}_i - \tilde{\lambda}_i B\tilde{x}_i \perp \mathcal{K} \ . \tag{2.28}$$

En este caso, si V es una base B-ortonormal de $\mathcal{K}$, $V^T BV = I$, el problema proyectado resulta ser también $V^T AVy_i = \tilde{\lambda}_i y_i$. Un caso más general de esta condición se tratará en la siguiente sección.

2.3. Versión de Davidson

En esta sección, se generaliza el método de Minimización de la Traza, descrito en la sección anterior, enmarcándolo en el contexto de los métodos de tipo Davidson. Para tal propósito, se introducirá el Método de Jacobi-Davidson, se resaltarán las relaciones con Minimización de la Traza, y se terminará con la descripción *tipo Davidson* del método.

2.3.1. Método de Jacobi-Davidson

El método de Lanczos inicialmente no recibía mucha atención ya que se veía como un método para tridiagonalizar una matriz, que no podía competir en precisión con los métodos de Givens y Householder, si no era con costosísimas modificaciones como la ortogonalización explícita de los vectores obtenidos (en principio, operación innecesaria en aritmética exacta). Años más tarde, los trabajos de Paige permitieron una serie de importantes contribuciones de muchos autores, resultando

en una mejor comprensión del método, implementaciones más estables, y por tanto, un aumento de su popularidad.

Este método fue pensado originalmente para manejar problemas de valores propios estándares. Las extensiones que le permiten resolver sistemas generalizados tienen como inconveniente la necesidad de resolver un sistema lineal con la matriz A o B en cada iteración (por ejemplo, por el empleo de la transformación espectral desplazamiento e inversión que sustituye la A por $(A-\sigma B)^{-1}B$), o la factorización de matrices de la forma $A-\sigma B$ durante el proceso, delegando la convergencia del método en la precisión con la cual se resuelvan los sistemas lineales o la precisión de los factores.

Los métodos de Davidson pueden ser vistos como métodos precondicionados de Lanczos. En un principio, el método propuesto por Davidson fue muy exitoso para resolver problemas estándares de valores propios en química cuántica, cuyas matrices suelen ser diagonalmente dominantes. Tiempo después, los métodos de Davidson fueron mejorando significativamente, como se puede apreciar en el método que vamos a presentar seguidamente.

El método de Jacobi-Davidson es una variante del esquema original de los métodos de Davidson y del método de Newton. Como en Minimización de la Traza, iterativamente calcula correcciones ortogonales sobre las aproximaciones de los vectores propios, siguiendo un esquema bastante parecido al que se ha descrito. Primero resuelve un sistema de valores propios proyectado mediante el procedimiento de Petrov-Galerkin (una generalización del procedimiento de Rayleigh-Ritz visto anteriormente). Y luego, resuelve la *ecuación de corrección de Jacobi* para expandir el subespacio de búsqueda.

De la misma forma que el procedimiento de Rayleigh-Ritz, el método de Petrov-Galerkin busca soluciones del problema 1.3 en el subespacio $\mathcal{K}$ de manera que el error sea ortogonal a un subespacio dado $\mathcal{L}$, siendo ambos subespacios de dimensión k:

$$AVs - \theta BVs \perp \mathcal{L} \ , \tag{2.29}$$

donde V es una base ortonormal del subespacio $\mathcal{K}$. Si consideramos W como una base ortonormal del subespacio $\mathcal{L}$, la ecuación anterior la podemos reescribir como

$$W^*AVs_i - \theta_i W^*BVs_i = 0 \ . \tag{2.30}$$

Esta ecuación tiene k soluciones (θ_i, s_i), con $1 \leq i \leq k$. Los correspondientes k pares (θ_i, Vs_i) se llaman esta vez pares de Petrov, compuestos por un valor y un vector de Petrov. Bajo ciertas elecciones de los subespacios $\mathcal{K}$ y $\mathcal{L}$ los pares de Petrov son aproximaciones de los pares propios del sistema original.

Según este método, el espacio de búsqueda $\mathcal{K}$ es expandido por la solución de la llamada ecuación de corrección de Jacobi-Davidson:

$$(I - q_i q_i^*)(A - \theta_i B)(I - u_i u_i^*)t = r_i \qquad t \perp u_i \ , \tag{2.31}$$

siendo $r_i = (A - \theta_i B)u_i$, y q_i y u_i pertenecientes a $\mathcal{L}$ y $\mathcal{K}$, respectivamente. Para una rápida convergencia cuadrática, q_i debe ser una combinación lineal de Au_i y Bu_i, y ortogonal al residuo r_i (ver [Fokkema et al., 1999]).

Si consideramos B no singular (lo cual es asumible, si hemos partido de un sistema con B definida positiva), podemos encontrar una versión en [Sleijpen et al., 1996] en que $W = BV$ y $u_i = q_i$, resultando el algoritmo que aparece a continuación.

Algoritmo 4 (Jacobi-Davidson)

Entrada: matrices A y B
número de pares propios que se desean p
tamaño de bloque s, $s \geq p$
dimensión máxima del subespacio m, $m \geq s$
Salida: pares propios obtenidos

0. Elegimos una matriz V_1 de tamaño $n \times s$ tal que $V_1^T B V_1 = I_s$.

Para $k = 1, 2, \ldots$ hasta convergencia
1. Obtener $W_k \leftarrow AV_k$ y $H_k \leftarrow V_k^T W_k$.
2. Obtener los s pares propios (Θ_k, Y_k) más pequeños de H_k.
3. Obtener los vectores de Ritz $X_k \leftarrow V_k Y_k$.
4. Obtener los residuos $R_k \leftarrow W_k Y_k - BX_k\Theta_k$.
5. Comprobar convergencia.
6. Resolver $[P_{k,i}(A - \sigma_{k,i}B)P_{k,i}]d_{k,i} = P_{k,i}r_{k,i}$ aproximadamente, con $x_{k,i} \perp_B d_{k,i}$, donde $P_{k,i} = I - Bx_{k,i}(x_{k,i}^T B^2 x_{k,i})^{-1}x_{k,i}^T B$.
7. Si $|V_k| \leq m - s$ entonces $V_{k+1} \leftarrow B$-ortonormalización de $[V_k, \Delta_k]$, si no $V_{k+1} \leftarrow B$-ortonormalización de $[X_k, \Delta_k]$.

Fin para

NOTA: $d_{k,i}$ y $r_{k,i}$ corresponden con la columna i de las matrices Δ_k y R_k, respectivamente.

Este algoritmo puede ser tomado como una versión de Minimización de la Traza con subespacios crecientes. El comportamiento del algoritmo depende de cómo de buena sea la solución inicial y cómo de eficiente y preciso es resuelto el sistema lineal interno.

Si se toma como solución del sistema interno su parte derecha, el algoritmo se reduce al método de Lanczos. Por otro lado, si el sistema interno es resuelto de manera muy precisa, el algoritmo se comporta como la Iteración del Cociente de Rayleigh (*Rayleigh Quotien Iteration*, RQI) a bloques con expansión del subespacio, que converge cúbicamente. Sin embargo, si el sistema interno es resuelto aproximadamente, su comportamiento se encuentra a mitad camino entre ambos métodos.

De la combinación del esquema de este algoritmo, junto a la resolución del sistema interno a la manera de Minimización de la Traza, surge el algoritmo que presentamos seguidamente.

2.3.2. Minimización de la Traza tipo Davidson

En [Sameh y Tong, 2000] propusieron una variante de Minimización de la Traza con expansión del subespacio, en la cual el número de columnas de V_k y las dimensiones del problema proyectado H_k crecían a medida que progresaba el algoritmo. Esta variante está relacionada con los métodos de Davidson Generalizados debido a que emplea el precondicionamiento de los residuos para expandir el espacio de búsqueda. Estos residuos remplazarán a las aproximaciones de los vectores propios que aparecen en el sistema lineal. La integración de estas ideas en el algoritmo 2 resulta en el Método de Minimización de la Traza tipo Davidson[1], resumido en el siguiente algoritmo:

Algoritmo 5 (Minimización de la Traza tipo Davidson)

Entrada: matrices A y B
número de pares propios que se desean p
tamaño de bloque s, $s \geq p$
dimensión máxima del subespacio m, $m \geq s$
Salida: pares propios obtenidos

0. Elegir una matriz V_1 de tamaño $n \times s$ tal que $V_1^T B V_1 = I_s$.
Para $k = 1, 2, \ldots$ hasta convergencia
1. Obtener $W_k \leftarrow AV_k$ y $H_k \leftarrow V_k^T W_k$.
2. Obtener los s pares propios (Θ_k, Y_k) más pequeños de H_k.
3. Obtener los vectores de Ritz $X_k \leftarrow V_k Y_k$.
4. Obtener los residuos $R_k \leftarrow W_k Y_k - BX_k\Theta_k$.
5. Comprobar convergencia.
6. Resolver $[P_k(A - \sigma_{k,i}B)P_k]d_{k,i} = P_k r_{k,i}$ aproximadamente, con $d_{k,i} \perp_B X_k$.
7. Si $|V_k| \leq m - s$

[1] En el artículo [Sameh y Tong, 2000] lo llaman *Davidson-type Trace Minimization.*

entonces $V_{k+1} \leftarrow B$-ortonormalización de $[V_k, \Delta_k]$,
si no $V_{k+1} \leftarrow B$-ortonormalización de $[X_k, \Delta_k]$.
Fin para

Cabe destacar como primera diferencia con respecto al algoritmo 2 que el número de columnas de V_k es inicialmente s, aunque va creciendo conforme avanzan las iteraciones, de la misma manera que la matriz proyectada del problema H_k. Sin embargo, en el paso 2 únicamente se obtienen los s pares propios más pequeños (y no todos como en el algoritmo 2). Por tanto, el número de columnas de X_k se mantiene constante a s. Luego, el sistema lineal del paso 6, también difiere del original, y el papel de los $\sigma_{k,i}$ será explicado con detalle en la sección 3.1. Finalmente, el paso 7 expande el subespacio de trabajo, excepto si alcanza su tamaño máximo m, que en tal caso se reinicia con los vectores de Ritz X_k y los actualizados Δ_k.

Los únicos pasos que *en apariencia* tienen costes importantes dependientes de m son el 1, 2 y 7. En la práctica se aprovecha que si no hay reinicio, el paso 7 no modifica los vectores existentes en V_{k-1} y sólo añade vectores nuevos. Por tanto, si $V_k = [V_{k-1}\ \tilde{V}_k]$ siendo $\tilde{V}_k$ los vectores añadidos en la iteración k, podemos implementar las operaciones del paso 1 como sigue:

$$\begin{aligned} W_k &= AV_k = A[V_{k-1}\ \tilde{V}_k] = [AV_{k-1}\ A\tilde{V}_k] = [W_{k-1}\ A\tilde{V}_k] = [W_{k-1}\ \tilde{W}_k] \\ H_k &= V_k^T W_k = \begin{bmatrix} V_{k-1}^T \\ \tilde{V}_k^T \end{bmatrix} [W_{k-1}\ \tilde{W}_k] = \begin{bmatrix} V_{k-1}^T W_{k-1} & V_{k-1}^T \tilde{W}_k \\ \tilde{V}_k^T W_{k-1} & \tilde{V}_k^T \tilde{W}_k \end{bmatrix} \\ &= \begin{bmatrix} H_{k-1} & V_{k-1}^T \tilde{W}_k \\ \tilde{W}_k^T V_{k-1} & \tilde{V}_k^T \tilde{W}_k \end{bmatrix} \end{aligned}$$

La actualización de H_k es el paso más costoso. Considerando que H_k es a lo sumo de tamaño m, se puede suponer que su coste es de orden $\mathcal{O}(nsm)$, aunque cabe decir que es una operación muy paralelizable (con una operación de comunicación colectiva sería suficiente).

En el paso 2 se resuelve directamente un problema estándar de valores propios de orden m a lo sumo, con coste $\mathcal{O}(m^3)$, pero no se paraleliza (como se verá en la sección 5.2.2). Por último, el paso 7 normalmente B-ortonormaliza Δ_k con respecto a V_k que tiene un coste computacional asintótico de $\mathcal{O}(ns(m-s))$, y tan sólo s operaciones de reducción colectivas (por cada iteración de ortogonalización que se llevara a cabo, ver sección 3.4).

Por tanto, y exceptuando la aplicación de matrices y precondicionadores, el coste en mensajes por iteración es $\mathcal{O}(sr)$, siendo r el número de mensajes necesarios para realizar una operación colectiva de reducción.

Capítulo 3

Consideraciones prácticas

En las siguientes secciones se describen los métodos empleados en Minimización de la Traza, así como ciertas técnicas y detalles que aceleran la convergencia y hacen más robusta su implementación.

3.1. Múltiples desplazamientos dinámicos

Una de las optimizaciones más importantes es el uso de desplazamientos (*shift*) de manera que aceleren la convergencia. El sistema original 2.8 es equivalente a este siguiente:

$$[P(A-\sigma_{k,i}B)P]d_{k,i} = PAx_{k,i}, \quad X^TBd_{k,i} = 0, \quad 0 \leq \sigma_{k,i} \leq \lambda_i \ , \tag{3.1}$$

donde $d_{k,i}$ es la columna i de la matriz Δ_k, $x_{k,i}$ es la columna i de la matriz X_k y $\sigma_{k,i}$ corresponde al desplazamiento seleccionado para valor propio λ_i. También el sistema interno de la versión de tipo Davidson tiene desplazamientos. En ambos casos, la implementación debe asignar un valor a los $\sigma_{k,i}$ basándose en los valores de Ritz. Para facilitar la notación, se considerará que en cada iteración externa ambas versiones actualizan s aproximaciones de pares propios. En el caso de la versión de Davidson, este parámetro forma parte de los argumentos de entrada, mientras que para la versión clásica s tendrá el valor de m.

3.1.1. Estrategia básica

La estrategia para establecer los desplazamientos que se comenta a continuación, se describe en [Sameh y Tong, 2000]. Se basa en el siguiente teorema enunciado en [Parlett, 1980]: dado un vector u y un escalar σ no nulos, existe un valor propio λ del sistema 1.3 que cumple la relación

$$|\lambda - \sigma| \leq \frac{\|(A-\sigma B)u\|_{B^{-1}}}{\|Bu\|_{B^{-1}}} \ . \tag{3.2}$$

Si empleamos los valores y los vectores de Ritz en el anterior teorema obtenemos unas cotas de los valores propios exactos,

$$|\lambda - \theta_{k,i}| \leq \frac{\|(A-\theta_{k,i}B)x_{k,i}\|_{B^{-1}}}{\|Bx_{k,i}\|_{B^{-1}}} = \|r_{k,i}\|_{B^{-1}} \ . \tag{3.3}$$

Si combinamos el anterior teorema con que el valor propio λ_i está siempre por debajo del valor de Ritz correspondiente $\theta_{k,i}$ (esto se puede demostrar mediante el teorema de Courant-Fischer), se obtiene que cuando el valor de Ritz está cerca de su valor propio objetivo, éste se puede acotar en el intervalo $[\theta_{k,i} - \|r_{k,i}\|_{B^{-1}}, \theta_{k,i}]$. A continuación se describe una estrategia de desplazamientos que tiene en cuenta esto último.

Sea i_0 el número de valores propios que han convergido, entonces,

- si
$$\theta_{k,i_0+1} + \|r_{k,i_0+1}\|_{B^{-1}} \leq \theta_{k,i_0+2} - \|r_{k,i_0+2}\|_{B^{-1}},$$
entonces, $\sigma_{k,i_0+1} \leftarrow \theta_{k,i_0+1}$; si no, se establece
$$\sigma_{k,i_0+1} \leftarrow \text{máx}(\theta_{k,i_0+1} - \|r_{k,i_0+1}\|_{B^{-1}}, \lambda_{i_0});$$
- sea j una columna, con $i_0 + 1 < j \leq p$, se elige el mayor $\theta_{k,l}$ tal que $\theta_{k,l} < \theta_{k,j} - \|r_{k,j}\|_{B^{-1}}$;
- si por el criterio anterior $\sigma_{k,i-1} = \theta_{k,i-1}$ y se cumple que $\theta_{k,i} < \theta_{k,i+1} - \|r_{k,i+1}\|_{B^{-1}}$, entonces asignamos $\sigma_{k,i} \leftarrow \theta_{k,i}$; y
- para el resto de columnas j, asignar $\sigma_{k,j} \leftarrow \sigma_{k,i_0+1}$.

Por razones prácticas, en [Sameh y Tong, 2000] se recomienda sustituir las normas en base B^{-1} (que requerirían la resolución de un sistema lineal con B) por 2-normas (que sólo requieren un producto escalar de vectores). Sin embargo, la implementación de esta estrategia con 2-normas no ofrece muy buenos resultados: en la figura 6.4 se observa que solamente es efectiva en 4 de 14 casos propuestos en la batería de test.

3.1.2. Aproximación de la B^{-1}-norma

Lanzamos como primera hipótesis que quizás la aproximación de la B^{-1}-norma como una 2-norma no produjera buenos desplazamientos, es decir, los desplazamientos o están muy alejados o son mayores que el valor propio exacto, y esto repercutiera en la efectividad de la estrategia. Teórica y experimentalmente se demuestra que ambos fenómenos se dan.

Queremos demostrar que existe una matriz simétrica y semidefinida positiva B y un vector residuo r tal que $\theta_j - \|r\|_{B^{-1}} \leq \theta_j - \|r\|_2$, es decir, que con la 2-norma se obtiene un desplazamiento que está por encima de la cota sugerida por 3.3. Si reescribimos la B^{-1}-norma de la manera

$$\|r\|_{B^{-1}} = \sqrt{r^T B^{-1} r} = \|r\|_2 \sqrt{z^T B^{-1} z} \leq \|r\|_2 \sqrt{\lambda_{\text{máx}}(B^{-1})} = \|r\|_2 \, \lambda_{\text{mín}}(B)^{-\frac{1}{2}} \ , \tag{3.4}$$

siendo $z = \|r\|^{-1} r$, entonces

$$\begin{aligned} \theta_j - \|r\|_{B^{-1}} \leq \theta_j - \|r\|_2 \quad &\longrightarrow \quad \|r\|_2 \leq \|r\|_{B^{-1}} \\ &\longrightarrow \quad \|r\|_2 \leq \|r\|_2 \, \lambda_{\text{mín}}(B)^{-\frac{1}{2}} \\ &\longrightarrow \quad 1 \leq \lambda_{\text{mín}}(B)^{-\frac{1}{2}} \\ &\longrightarrow \quad \lambda_{\text{mín}}(B) \leq 1 \ . \end{aligned}$$

Por tanto, los desplazamientos obtenidos con el empleo de la 2-norma en la estrategia pueden no ser seguros en aquellos problemas cuya matriz B tenga un valor propio menor que 1.

Por otro lado, vamos a analizar cuanto dista la 2-norma de la B^{-1}-norma. Para ello analizamos el máximo y el mínimo del cociente

$$\begin{aligned} \frac{\|r\|_2}{\|r\|_{B^{-1}}} &= \sqrt{\frac{r^T r}{r^T B^{-1} r}} = \sqrt{\frac{z^T z}{z^T B^{-1} z}} = \sqrt{\frac{1}{z^T B^{-1} z}} \ : \\ \max_r \frac{\|r\|_2}{\|r\|_{B^{-1}}} &= \max_z \sqrt{\frac{1}{z^T B^{-1} z}} = \lambda_{\text{máx}}(B)^{\frac{1}{2}} \ , \text{ y} \qquad (3.5) \\ \min_r \frac{\|r\|_2}{\|r\|_{B^{-1}}} &= \min_z \sqrt{\frac{1}{z^T B^{-1} z}} = \lambda_{\text{mín}}(B)^{\frac{1}{2}} \ . \qquad (3.6) \end{aligned}$$

Por tanto, el empleo de la 2-norma aleja a los desplazamientos más allá del límite inferior seguro en un factor que depende de los valores propios de B.

En conclusión, la sustitución de las normas funcionará mejor en problemas cuya matriz B tenga valores propios mayores que 1, pero próximos a 1.

Para paliar ambos efectos, proponemos sustituir $\|r\|_{B^{-1}}$ por $\alpha^{-\frac{1}{2}}\|r\|_2$ donde α es una aproximación por debajo del valor propio más pequeño de B, inspirada en la relación 3.4. En la figura 6.4 se observa cómo con la estimación de $\lambda_{\text{mín}}(B)$ proporcionada por el teorema de los discos de Gershgorin (explicado en la sección 5.2.1) se obtiene una mejora (respecto a no emplear desplazamientos) en 8 de los 14 casos, pudiendo llegar a 9 si se emplea una aproximación de $\lambda_{\text{mín}}(B)$ con mucha precisión (en este caso con 6 dígitos significativos obtenidos mediante el Krylov-Schur con la transformación espectral desplazamiento e inversión). Esta variante de las estrategias de desplazamientos será llamada desplazamientos corregidos.

3.1.3. Desplazamientos y la precisión de los valores de Ritz. Desplazamientos seguros

Experimentalmente se observó que en las primeras iteraciones se producían desplazamientos no seguros mucho más frecuentemente que en iteraciones posteriores, produciendo una ralentización de la convergencia del problema. La figura 3.1 muestra la convergencia de los valores propios del problema BCSST09 con diferentes estrategias de desplazamientos. Se observa en la figura 3.1b cómo se están empleando desplazamientos que son mayores que el valor propio al que corresponden, produciendo una *lenta* convergencia de las aproximaciones comparado con la versión sin desplazamientos (gráfica 3.1a).

La estrategia de desplazamientos corregidos evita en gran medida que se usen desplazamientos no seguros como se puede observar en la gráfica 3.1c, pero produce cotas tan holgadas que sólo emplea los valores propios más pequeños como desplazamientos, reduciendo la aceleración que esta técnica podría producir.

Sin embargo, la aplicación de los desplazamientos sólo cuando el error relativo del par propio es menor que una cierta cota, obtiene mejores resultados. La gráfica 3.1d muestra la evolución de los valores propios si se aplica el desplazamiento cuando el error relativo del par propio es menor que 10^{-4}. Se hará referencia a esta estrategia como desplazamientos seguros.

3.2. Precondicionado

Otra característica importante de Minimización de la Traza, es la naturalidad con la cual se incorporan los precondicionadores. En particular, son empleados para acelerar y mejorar la convergencia de la resolución del sistema interno.

3.2.1. Precondicionado directo

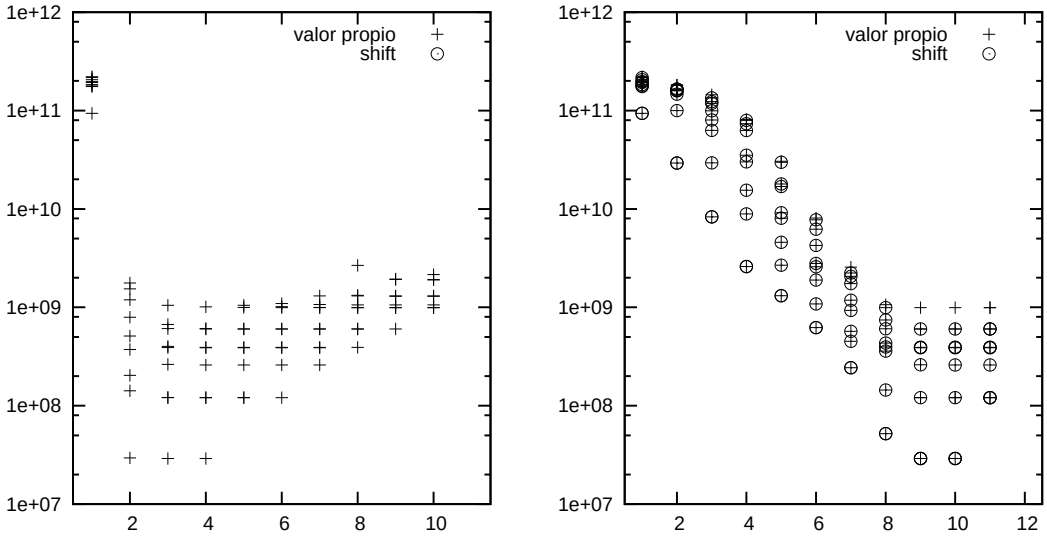
En [Sameh y Tong, 2000] se sugiere construir un precondicionador $\hat{A} = CC^T$ simétrico y definido positivo de $A - \sigma_{k,i}B$. Entonces el sistema 3.1 puede transformarse en

$$[\tilde{P}(\tilde{A} - \sigma_{k,i}\tilde{B})\tilde{P}]\tilde{d}_i = \tilde{P}\tilde{A}\tilde{x}_i, \quad \tilde{X}^T\tilde{B}\tilde{d}_i = 0, \tag{3.7}$$

siendo

$$\begin{aligned}
\tilde{A} &= C^{-1}AC^{-T}, & \tilde{B} &= C^{-1}BC^{-T}, \\
\tilde{d}_i &= C^T d_{k,i}\ , & \tilde{X} &= C^T X_k\ , \\
\tilde{x}_i &= C^T x_{k,i}\ , & \tilde{P} &= I - \tilde{Y}(\tilde{Y}^T\tilde{Y})^{-1}\tilde{Y}^T, \\
\tilde{Y} &= C^{-1}BX_k\ . & &
\end{aligned}$$

Es posible reescribir 3.7 de manera que pueda ser resuelto mediante el método lineal precondicio-



(a) sin desplazamientos

(b) con desplazamientos

(c) con desplazamientos corregidos

(d) con desplazamientos seguros

Figura 3.1: Evolución de los valores propios del problema BCSST09 empleando la versión de tipo Davidson y comparando entre las variantes sin desplazamientos, con desplazamientos, con desplazamientos corregidos y con desplazamientos seguros. Las gráficas únicamente muestran los pares que todavía no han sido bloqueados, es decir, los que no se consideran convergidos.

nado, si sustituimos

$$\begin{aligned}
\tilde{P} &= I - C^{-1}\overbrace{BX_k(\tilde{Y}^T\tilde{Y})^{-1}B^TX_k^T}^{P'}C^{-T}, && (3.8)\\
\tilde{A}-\sigma_{k,i}\tilde{B} &= C^{-1}(A-\sigma_{k,i}B)C^{-T}, && (3.9)\\
\tilde{P}\tilde{A}\tilde{x}_i &= \tilde{P}C^{-1}Ax_{k,i}\ , \quad \text{y} && (3.10)\\
\hat{A}^{-1} &= C^{-T}C^{-1} && (3.11)
\end{aligned}$$

en 3.7, resultando

$$C^{-1}P_1'(A-\sigma_{k,i}B)P_2'C^{-T}C^Td_{k,i} = C^{-1}P_1'Ax_{k,i} \tag{3.12}$$

donde

$$\begin{aligned}
P_1' &= I - P'\hat{A}^{-1}\ , \quad \text{y} && (3.13)\\
P_2' &= I - \hat{A}^{-1}P'\ . && (3.14)
\end{aligned}$$

De manera similar, se puede aplicar esta técnica al sistema interno de la versión tipo Davidson.

Siguiendo las recomendaciones en [Sameh y Tong, 2000], hemos conseguido que esta manera de precondicionar el sistema lineal funcionase únicamente en problemas pequeños y sencillos, resolviendo el sistema lineal con el método CG precondicionado y GMRES con la descomposición incompleta de Cholesky sin relleno ICC(0) sobre la matriz A.

Tras descartar errores en la resolución del sistema proyectado y en la ortonormalización, buscamos la forma en que se resolvían sistemas parecidos en otros métodos.

3.2.2. Adaptación de las soluciones propuestas para la ecuación de corrección de Jacobi-Davidson

Jacobi-Davidson, como ya ha sido comentado, es también un método de resolución de problemas de valores propios simétricos y definidos positivos, que se basa en la resolución de la *ecuación de corrección*, 2.31. Este sistema, de la misma manera que el sistema lineal con desplazamientos del método de Minimización de la Traza (ecuación 3.1), es un sistema proyectado con una restricción de ortogonalidad.

Nuestra propuesta es, pues, adaptar un método presente en la literatura para resolver la ecuación de corrección al sistema lineal del método de Minimización de la Traza. En concreto, se presentará la técnica descrita en [Sleijpen et al., 1998] y su adaptación al método en estudio.

Partiremos, esta vez, de una formulación un poco más general de la ecuación de corrección. Si se desea obtener más de un par propio, es necesario incorporar la deflación a Jacobi-Davidson, que modifica la ecuación de corrección quedando finalmente

$$(I-[\tilde{Q},q_i][\tilde{Q},q_i]^*)(A-\theta_iB)(I-[\tilde{U},u_i][\tilde{U},u_i]^*)t = r_i, \qquad t\perp[\tilde{U},u_i]\ , \tag{3.15}$$

siendo las columnas de $\tilde{Q}$ y $\tilde{U}$ las correspondientes a los pares convergidos. Para facilitar la notación, a partir de ahora $Q=[\tilde{Q},q_i]$ y $U=[\tilde{U},u_i]$.

En [Sleijpen et al., 1998] se sugiere el empleo de métodos de Krylov para solucionar 3.15, pues, entre otros motivos permiten en general la incorporación de un precondicionador. Supongamos que P es una aproximación de $A-\theta_iB$ que permite la resolución de sistemas lineales $Px=y$ muy eficientemente. El precondicionador también debe ser restringido de la misma manera que $A-\theta_iB$ en la ecuación de corrección:

$$\tilde{P} = (I-QQ^*)P(I-UU^*)\ . \tag{3.16}$$

La solución z para el sistema

$$\tilde{P}z = y, \qquad z\perp U,\ y\perp Q\ , \tag{3.17}$$

puede ser reescrita como

$$z = \left(I - \tilde{Q}\left(UP^{-1}Q\right)^{-1}U^*\right)\tilde{y}\ , \tag{3.18}$$

donde $\tilde{y} = P^{-1}y$, $\tilde{Q} = P^{-1}Q$. Nótese que para un vector arbitrario y el vector z obtenido de 3.18 satisface

$$\tilde{P}z = (I - QQ^*)y, \qquad z \perp Q\ . \tag{3.19}$$

Esta observación permite la simplificación del cálculo del precondicionado por la izquierda en un método de Krylov. La acción del operador precondicionado sobre un vector dado v consiste en su multiplicación por la matriz proyectada

$$y = (I - QQ^*)(A - \theta_i B)(I - UU^*)v\ , \tag{3.20}$$

seguido de la aplicación del precondicionador proyectado, ecuación 3.17. Cuando v es ortogonal a Q, las ecuaciones 3.19 y 3.20 muestran que esto es equivalente a multiplicar por $A - \theta_i B$ seguido del cómputo de z en 3.18. Como el vector resultante z es ortonormal a U, podemos observar que el subespacio de Krylov para el *solver* lineal y el operador precondicionado pueden ser construidos con multiplicaciones por la matriz $A - \theta_i B$ (sin proyección de ningún tipo) y la aplicación del precondicionador, siempre que se tome el vector de inicio del *solver* lineal uno ortogonal a las columnas de Q.

En resumidas cuentas, para resolver el sistema 3.15 mediante un método de Krylov, bastaría con que se cumpliera que

- el operador aplicara el precondicionador, $\tilde{y} = P^{-1}y$, seguido del proyector oblicuo

$$z = \left(I - \tilde{Q}\left(UP^{-1}Q\right)^{-1}U^*\right)\tilde{y}\ ;\ \text{y que} \tag{3.21}$$

- el vector inicial fuera ortogonal a las columnas de Q.

Entonces, retomando el objetivo inicial (resolver el sistema 3.1 con precondicionamiento), se observa que el sistema 3.1 es un caso particular de la ecuación de corrección en el que Q y U son iguales a una base ortonormal de BX_k, y el residuo está proyectado. Nótese que la restricción $t \perp U$ se sigue cumpliendo ya que $d \perp BX_k$.

El proyector oblicuo 3.18 queda pues de la siguiente manera:

$$z = \left(I - P^{-1}\tilde{X}\left(\tilde{X}^*P^{-1}\tilde{X}\right)^{-1}\tilde{X}^*\right)\tilde{y}\ , \tag{3.22}$$

siendo $\tilde{X}$ una base ortonormal de BX_k.

Como se verá en el capítulo 6, esta manera de precondicionar es efectiva en la totalidad de los problemas de las baterías 6.1 y 6.2 usando el *solver* lineal GMRES de PETSc y el precondicionador *boomeramg* de HYPRE. En la sección 6.3.2 se discute sobre otros métodos para resolver el sistema lineal y se muestran algunos resultados experimentales.

3.3. Resolución de sistemas lineales mediante métodos de Krylov

El método de Gradiente Conjugado es un método iterativo basado en subespacios de Krylov para resolver sistemas lineales simétricos y definidos positivos. Los métodos de Krylov construyen un subespacio de Krylov $\mathcal{K}_k(G, x)$, que se define como el subespacio generado por las sucesivas potencias de G aplicadas sobre x:

$$\mathcal{K}_k(G, x) \equiv \text{span}\{x, Gx, \ldots, G^{k-1}x\}. \tag{3.23}$$

El conjunto $x, Gx, \ldots, G^{k-1}x$ forma una base muy mal condicionada. Parte del esfuerzo de estos algoritmos consiste en construir una base mejor, concretamente una base ortonormal.

En la versión precondicionada del problema,

$$M^{-1}Ax = M^{-1}b, \tag{3.24}$$

M también debe ser simétrica y definida positiva, para que $M^{-1}A$ sea simétrica y definida positiva respecto del producto escalar $\langle x, y\rangle_M$:

$$\begin{aligned} \langle x, M^{-1}Ay\rangle_M &= x^T M(M^{-1}Ay) = x^T Ay & (3.25)\\ &= x^T AM^{-1}My = (M^{-1}Ax)^T My = \langle M^{-1}Ax, y\rangle_M & (3.26)\end{aligned}$$

Durante la resolución del sistema interno mediante un método de Krylov, puede que el *solver* termine prematuramente (*breakdown*). Para evitar esto en la medida de lo posible, en [Sameh y Wisniewski, 1982] se sugiere lo siguiente.

De la expresión 2.3, parece razonable terminar de iterar con el sistema lineal para $d_{k,i}$ (la columna i de Δ_k) cuando el error

$$\epsilon_{k,i}^{(l)} = [(d_{k,i}^{(l)} - d_{k,i})^T A(d_{k,i}^{(l)} - d_{k,i})]^{1/2} \tag{3.27}$$

se reduzca en un factor $\tau_i = \lambda_i/\lambda_{s+1}$, siendo $d_{k,i}^{(l)}$ la solución aproximada en la iteración l-ésima del método para la columna i de X_k, y $d_{k,i}$ la solución exacta.

Es posible estimar $\epsilon_{k,i}^{(l)}$ como

$$[(d_{k,i}^{(l)} - d_{k,i}^{(l+1)})^T A(d_{k,i}^{(l)} - d_{k,i}^{(l+1)})]^{1/2}, \tag{3.28}$$

que en muchos casos se puede obtener fácilmente de los cálculos intermedios del método. También podemos estimar τ_i como $\theta_{k,i}/\theta_{k-1,s}$, ya que $\theta_{k,i}$ y $\theta_{k-1,s}$ convergerán progresivamente hacia λ_k y λ_s respectivamente.

Esta estrategia de terminación se debe adaptar al caso en que se empleen desplazamientos. Entonces, el método se para cuando el error

$$\epsilon_{k,i}^{(l)} = [(d_{k,i}^{(l)} - d_{k,i}^{(l+1)})(A - \sigma_{k,i}B)(d_{k,i}^{(l)} - d_{k,i}^{(l+1)})]^{1/2} \tag{3.29}$$

se reduzca en un factor aproximado de

$$\tau_i = \begin{cases} (\theta_{k,i} - \sigma_{k,i})/(\theta_{k-1,s} - \sigma_{k,i}), & \text{si } \theta_{k,i} \neq \sigma_{k,i} \\ (\theta_{k-1,i} - \sigma_{k,i})/(\theta_{k-1,s} - \sigma_{k,i}), & \text{si } \theta_{k,i} = \sigma_{k,i} \end{cases}. \tag{3.30}$$

3.4. Ortogonalización

El método de Minimización de la Traza necesita que V sea B-ortogonal. En el paso 0 y en el 7, se realizan explícitamente ortogonalizaciones para conservar la anterior propiedad. Existen varios algoritmos para realizar dicha operación. Por ejemplo, la factorización QR basada en reflectores de Householder o en rotaciones de Givens sería interesante si sólo se fueran a calcular unos pocos productos matriz-vector con Q sin construir Q explícitamente. En nuestro caso, es más ventajoso utilizar el algoritmo de Gram-Schmidt porque se requiere que los vectores columna de Q formen una base ortonormal, y además con respecto a un producto interior dado $\langle x, y\rangle_B$.

Existen varias versiones del procedimiento Gram-Schmidt. La versión clásica para B-ortogonalizar se muestra en el algoritmo 6. La versión modificada (algoritmo 7) consiste en un pequeño cambio. En la versión clásica el bucle interno que calcula el sumatorio $\sum_{j=1}^{i-1}(v_j^T Bv_i)v_j$, siempre realiza el producto escalar con el mismo valor de v_j. En la versión modificada, se usa un v_j que ya es ortogonal a todo v_l con $l < j$, haciendo esta versión numéricamente más estable que la anterior. Pero tiene como inconveniente que resulta muy ineficiente realizar en paralelo las iteraciones del bucle j (ya que se necesita el valor de v_j de la iteración anterior), problema que no presenta la versión clásica.

Sin embargo, las anteriores versiones proporcionan una pobre ortogonalización en muchos casos. Por tal motivo, se idearon versiones iterativas que consisten en *reortogonalizar* una y otra vez hasta

que se cumpla un criterio de convergencia (en la práctica, 2 o 3 iteraciones suelen ser suficientes). Para determinados criterios, como

$$0{,}5\|\tilde{v}_j\|_2 \leq \|v_j\|_2, \tag{3.31}$$

siendo $\tilde{v}_j$ el valor de v_j previo a la última ortogonalización, resulta que ambas versiones (CGS y MGS iterativas) logran un nivel de ortogonalidad muy bueno. Por tanto, la versión clásica iterativa es una buena candidata para ser usada en un entorno paralelo.

Algoritmo 6 (Gram-Schmidt Clásico, CGS)

Entrada: $V = \{v_1, \ldots, v_k\}$
Salida: base B-ortonormal de V
Para $i = 1, \ldots, k$
$v_i \leftarrow v_i - \sum_{j=1}^{i-1}(v_j^T B v_i)v_j$
$v_i \leftarrow v_i / \|v_i\|_B$
Fin

Algoritmo 7 (Gram-Schmidt Modificado, MGS)

Entrada: $V = \{v_1, \ldots, v_k\}$
Salida: base B-ortonormal de V
Para $i = 1, \ldots, k$
Para $j = 1, \ldots, i-1$
$v_i \leftarrow v_i - (v_j^T B v_i)v_j$
Fin
$v_i \leftarrow v_i / \|v_i\|_B$
Fin

3.5. Bloqueo

Cuando un par propio ha convergido no es necesario que vuelva a calcularse una nueva aproximación, ni calcular su residuo, ni ortonormalizar de nuevo. En la práctica, la matriz V_k se divide en dos partes: $V_k = [V_1, V_2]$. Al principio del proceso V_1 está vacío y en V_2 están todos los vectores propios aproximados. Cuando el valor propio más pequeño de V_2 converge, se elimina de V_2 su vector propio asociado y se añade a la derecha de V_1. Por tanto, V_1 contiene los i_0 vectores propios convergidos hasta ese momento, y además se encontrarán en orden creciente (de izquierda a derecha) correspondiente a sus valores propios.

En los pasos del 1 al 6 del algoritmo 2 sólo se usa la parte V_2 de V. Pero en el paso 7, $X_k - \Delta_k$ (en la versión clásica) o Δ_k (en la versión de tipo Davidson) también se B-ortogonaliza con respecto a V_1. Si esto último no se hiciera, podría volver a aparecer en V_2 algún vector propio que ya hubiera convergido. A esta técnica se la conoce como *bloqueo* (*locking*), que es una forma de una técnica más general llamada *deflación*.

Capítulo 4

Modelos computacionales y librerías

Una vez descritos los métodos de Minimización de la Traza y sus fundamentos teóricos en el capítulo 2, y discutidos los detalles más importes del mismo en el capítulo 3, quedando suficientemente cubiertos los aspectos algorítmicos, en este capítulo junto con el que sigue se considerarán los aspectos más relevantes y específicos de la fase de implementación.

4.1. Superordenadores

Las ramas teóricas y aplicadas de la ciencia convergen en la computación científica: permite simular fenómenos demasiado complejos para hacer predicciones seguras de manera teórica o demasiado peligrosos o caros para ser reproducidos en laboratorios. El éxito de la computación científica reside en que la demanda de supercomputación se ha mantenido en alza en las últimas décadas. Hoy en día es una herramienta indispensable en muchos campos. Algunos requieren gran cantidad de cálculo y demandan lo último en recursos computacionales.

Cabe recordar que esta tesis se enmarca dentro de la *computación de altas prestaciones* (High Performance Computing) que tiene como uno de los objetivos abordar problemas de gran tamaño, bien por el gran volumen de datos y/o gran cantidad de cómputo que involucran. Para lograr *altas prestaciones* se debe considerar, irremediablemente, la arquitectura de las máquinas más potentes que existan. Según la lista TOP500[1], la máquina más potente en Junio de 2008 es el RoadRunner[2], un supercomputador construido por IBM en Los Alamos National Laboratory (EEUU), con la espectacular marca de 1 PetaFlop (10^{15} operaciones de coma flotante por segundo) en el Linpack Benchmark[3]. Éste consiste en la conexión jeraraquizada mediante una red Infiniband de 6120 TriBlades (figura 4.1), formados cada uno por dos placas de dos CPU PowerXCell 8i cada uno, y otra placa con un AMD Opteron 2210 de dos núcleos. Es pues a este nivel una máquina de memoria distribuida ya que cada placa tiene su memoria RAM propia y sólo se pueden comunicar mediante paso de mensajes.

Sin embargo, las arquitecturas de los procesadores están aumentando su complejidad para escalar nuevas marcas de rendimiento, cambiando la estrategia de años anteriores que consistía en aumentar la frecuencia de reloj y explotar el paralelismo a nivel de instrucciones de manera automática (pipelining, ejecución superescalar, ejecución fuera de orden, renombrado de registros, ejecución especulativa o predicción de saltos). Ahora replican las unidades de cómputo y dan soporte para que el usuario (o el compilador) las programe individualmente, ofreciendo en muchos casos un paradigma de memoria compartida.

[1] http://www.top500.org
[2] http://www.lanl.gov/orgs/hpc/roadrunner
[3] http://www.netlib.org/benchmark/hpl/

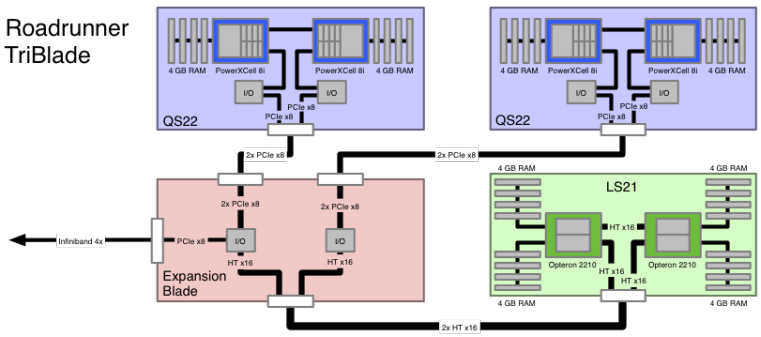


Figura 4.1: Esquema de los TriBlades del supercomputador BladeRunner de IBM.

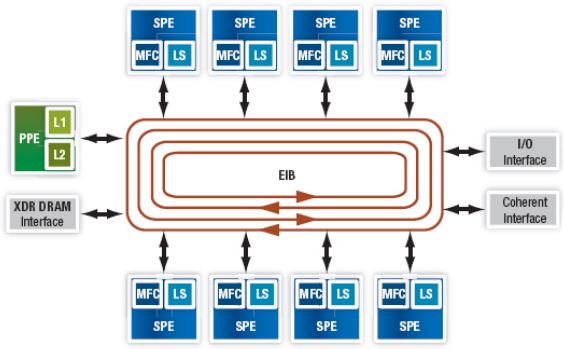


Figura 4.2: Arquitectura del Cell.

Por ejemplo, la arquitectura de la familia Cell Broadband Engine consiste en un núcleo IBM 64-bit Power Architecture, llamado *Power Processing Element* (PPE), y ocho coprocesadores especializados llamados *Synergistic Processing Elements* (SPEs) con arquitectura *Instrucción Única para Múltiples Datos* (Single Instruction Multiple Data, SIMD), los cuales desarrollan casi todo el potencial del procesador. Los nueve núcleos, junto a la memoria externa, están conectados vía el *Element Interconect Bus* (EIB) de altas prestaciones (ver figura 4.2).

También los procesadores de propósito general soportan instrucciones SIMD, como por ejemplo las tecnlogías 3DNow! y SSE de AMD e Intel, respectivamente.

Varios factores han conducido a desarrollar el paralelismo como fuente de potencia computacional. Por un lado, la velocidad de la luz y la capacidad de disipación de calor imponen limitaciones físicas en la velocidad de un procesador. Por otro, el ratio precio/prestaciones se vuelve muy desfavorable cuando se requieren altas prestaciones en un único procesador.

Sin embargo, la computación paralela presenta inconvenientes tanto de tipo software como de tipo hardware.

Las prestaciones de las redes de comunicación son una de las grandes limitaciones impuestas por el hardware. En los últimos años se han hecho grandes progresos, y la relación entre las prestaciones de los procesadores y de las redes se ha mejorado significativamente tanto en sistemas de memoria compartida (la red comunica los procesadores con la memoria) como distribuida (desde redes locales hasta incluso WAN). Aun así, sigue siendo el principal cuello de botella del rendimiento de muchas aplicaciones.

En muchos casos, la computación de altas prestaciones debe adaptarse a máquinas en las que confluyen los modelos de memoria compartida y distribuida, junto a entornos de computación heterogénea formados por procesadores con distintas características. Esta diversidad dificulta seriamente el desarrollo de códigos que cumplan con los preceptos de la ingeniería del software sin que ello afecte a las prestaciones.

Como consecuencia, es comprensible la falta de software (librerías, compiladores...) competente que realmente aproveche las capacidades del hardware, logrando prestaciones parecidas a las que prometen los fabricantes. Este problema es más acusado en entornos de memoria compartida (el hardware ofrece muchas más posibilidades de las que los compiladores, generalmente, suelen aprovechar).

De todas formas se han hecho grandes progresos que permiten desarrollar código portable y expresivo. En los últimos 15 años se han establecido una serie de librerías y APIs como estándares de facto por su uso extensivo:

- MPI [MPI Forum, 1994] como interfaz de paso de mensajes entre procesos, en el paradigma de memoria distribuida;
- hilos de POSIX[4] y OpenMP[5] como interfaces para la gestión de hilos de ejecución en el paradigma de memoria compartida;
- BLAS [Blackford et al., 2002] como modelo de operaciones básicas del álgebra de vectores y matrices densas;
- LAPACK [Anderson et al., 1992] como modelo de operaciones avanzadas del álgebra lineal: resolución de sistemas de ecuaciones lineales, de valores propios, SVD y mínimos cuadrados, para matrices densas.

Existen implementaciones de BLAS y LAPACK tanto para memoria compartida, como la AMD Core Math Library (AMCL)[6] y la Intel Math Kernel Library (MKL)[7], y memoria distribuida, como ScaLAPACK[8].

Sin embargo, para álgebra lineal dispersa no existe una hegemonía tan clara. Cabe destacar el paquete SPARSKIT[9] que tuvo mucha influencia y recoge la mayoría de los formatos de matrices dispersas así como implementaciones básicas para las mismas funcionalidades que BLAS. Las implementaciones de álgebra lineal dispersa en paralelo son muy reducidas, y con APIs muy diversas.

Uno de los últimos pasos en esta estandarización fue la extensión de BLAS al álgebra lineal dispersa: *Sparse BLAS*, que actualmente está implementada en la MKL (casi en su totalidad) y en la ACML (sólo el primer nivel).

La razón de que paquetes de uso tan amplio como BLAS y LAPACK no se hayan extendido al álgebra lineal dispersa hasta ahora no es porque ésta sea poco frecuente en aplicaciones reales, que no lo es, sino más bien, por un lado, por la dificultad que conlleva la optimización con datos dispersos, y por otro la variedad de formatos de almacenamiento de vectores y matrices dispersas que dificulta seriamente diseñar software aplicable a todas las áreas.

Las técnicas de planificación estática, que eran la base de las implementaciones en álgebra lineal densa, son ahora ineficientes en matrices que no sean estructuradas, al menos que los patrones de acceso sean conocidos a priori.

Sin embargo, sí son posibles ciertas optimizaciones en el campo del álgebra lineal dispersa. En muchos métodos iterativos una misma operación es llamada repetidamente. Por tanto, parece razonable emplear algún tiempo en calcular patrones de accesos más eficientes para la matriz y la operación dada. Dichas mejoras suelen venir dadas típicamente por el reordenamiento de los accesos para mejorar el paralelismo de grano fino, o mejorar el rendimiento de la caché. En este sentido, la reescritura de los métodos orientándolos a bloques ha obtenido buenos resultados.

Primero profundizaremos más en los diferentes modelos de paralelismo, para luego discutir el modo de funcionamiento de las librerías empleadas.

[4] Descrito en el estándar IEEE 1003.1c-1995.

[5] La especificación de la versión 3.0 fue aprobada por la OpenMP Architecture Review Board, y se puede encontrar en http://openmp.org/wp/openmp-specifications/

[6] http://developer.amd.com/cpu/Libraries/acml/

[7] http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/

[8] http://www.netlib.org/scalapack/

[9] http://www.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html

4.2. Memoria compartida y memoria distribuida

Existen diversas taxonomías de máquinas paralelas. Una de las más mencionadas en este campo es la taxonomía de Flynn, que las clasifica según el número de instrucciones concurrentes disponibles y a cuántos datos afecta. Este apartado se centrará en las máquinas de tipo Múltiples Instrucciones Múltiples Datos (MIMD), distinguiendo si la memoria está compartida o distribuida.

La *memoria compartida* es uno de los modelos más simples de paralelismo de control: aquél que no está determinado implícitamente por la independencia de los datos, sino que es el programador quien explícitamente lo especifica. En memoria compartida, las instrucciones de los procesadores tienen acceso a un único espacio de direccionamiento común a todos. Los accesos concurrentes a la memoria pueden coordinarse mediante mecanismos de sincronización, como los cerrojos.

Otra importante división de maquinas paralelas son las de *memoria distribuida*, aquellas que sólo tienen un espacio de direccionamiento privado, y se coordinan intercambiando mensajes en una red. El modelo de programación también suele recibir el nombre de *paso de mensajes*.

En memoria distribuida los usuarios no se preocupan de las condiciones de carrera (asunto que se debe tratar a conciencia en memoria compartida), pero, como desventaja, el programador asume la tarea de distribuir los datos, y ello suele producir profundas transformaciones en los algoritmos. No obstante, en el fondo puede que la tarea compense, ya que una buena distribución de los datos es la clave para obtener altas prestaciones, pues se ataca el principal cuello de botella desde el principio: la red.

4.3. Librerías de valores propios

Si se había comentado que el software de álgebra lineal dispersa estaba poco estandarizado, se puede comprender la situación de las librerías que implementan algoritmos para resolver problemas concretos, como en el caso que nos ocupa. Esta sección se centrará en los métodos iterativos para la resolución de problemas de valores propios.

Como estos métodos necesitan únicamente una serie de operaciones concretas sobre la matriz, sobre todo el producto matriz por vector, y dada la variabilidad de formatos de almacenamiento de las matrices dispersas, se optó por la abstracción. De esta manera, ARPACK[10] se convirtió, y sigue siendo, en una importante referencia, ya que empleaba la interfaz *reverse communication* para aplicar el producto matriz por vector. Desafortunadamente, tal mecanismo es engorroso, difícil de mantener y no permite la encapsulación de datos. Además ARPACK padece de un paradigma procedural en donde tales productos descansan en la implementación física de los datos manipulados (vectores), haciendo su reutilización limitada, y su mantenibilidad compleja y más engorrosa.

La tabla 4.1 lista los paquetes de valores propios para matrices dispersas más importantes así como los métodos que implementan. Se puede observar que las librerías más modernas abandonan el FORTRAN (aunque ofrecen una interfaz) por lenguajes que permiten implementar el *reverse communication* de manera más *natural*[11] (caso de PRIMME[12]), otras librerías se han decidido por un diseño orientado a objetos (caso de SLEPc) y otras por una implementación orientada a objetos empleando programación genérica (caso de ANASAZI).

De la tabla anterior también se puede deducir que los métodos más implementados son Arnoldi/-Lanczos (con reinicio implícito y la versión Krylov-Schur), Jacobi-Davidson y LOBPCG.

Tras esta radiografía de las librerías de valores propios, el resto del capítulo se centrará en las que han sido empleadas en las implementaciones del método de Minimización de la Traza: SLEPc y PETSc.

[10] http://www.caam.rice.edu/software/ARPACK/

[11] casi todos los lenguaje modernos soportan referencias a funciones.

[12] http://www.cs.wm.edu/~andreas/software/

Tabla 4.1: Lista del software más importante de valores propios dispersos. La columna llamada 'Par.' indica con un 'M' si usa MPI, 'O' si usa OpenMP y '-' si no esta paralelizada.

Librería	Métodos que implementa	Versión	Fecha	Lenguaje	Par.
ANASAZI	Block Krylov-Schur, block Davidson, LOBPCG	7.0.6	2007	C++	M
ARPACK	Arnoldi/Lanczos (reinicio implícito)	2.1	1995	F77	M
BLKLAN	Block Lanczos	-	2003	C/Matlab	-
BLOPEX	LOBPCG	-	2004	C/Matlab	M
BLZPACK	Block Lanczos	04/00	2000	F77	M
EIGIFP	Inverse-free Krylov subspace method	2.1.1	2004	Matlab	-
IETL	Power, RQI, Lanczos	2.2	2006	C++	-
INSYLAN	Lanczos Simétrico Indefinido	1.0	2000	Matlab	-
IRBLEIGS	Block Lanczos (reinicio implícito)	1.0	2002	Matlab	-
JDBSYM	Jacobi-Davidson (simétrico)	0.14	1999	C	-
JDCG	Jacobi-Davidson (simétrico)	-	2000	Matlab	-
JDQR/JDQZ	Jacobi-Davidson	-	1998	F77/Matlab	-
MPB	Conjugate Gradient, Davidson	1.4.2	2003	C	M
NA18	Block Davidson	-	1999	F77	-
PDACG	Deflation-accelerated Conjugate Gradient	-	2000	F77	M
PRIMME	Block Davidson, JDQMR, JDQR, LOBPCG	1.1	2006	C/F77	M
PROPACK	SVD via Lanczos	2.1/1.1	2005	F77/Matlab	O
PySPARSE	Jacobi-Davidson (simétrico)	1.0.1	2007	Python	-
SLEPc	Krylov-Schur, Arnoldi, Lanczos, *Trace Minimization*	2.3.4	2008?	C/F77	M
SPAM	Davidson con SPAM	-	2001	F90	-
TRLAN	Lanczos (reinicio grueso dinámico)	-	2006	F90	M

4.4. Las librerías SLEPc y PETsc

SLEPc[13] (*Scalable Library for Eigenvalue Problem Computation*) es una librería orientada a la resolución de problemas de valores propios y valores singulares grandes y dispersos en entornos paralelos. Soporta problemas estándares y generalizados, tanto hermitianos como no hermitianos, y en aritmética real o compleja.

La mayoría de los métodos que ofrece SLEPc son métodos de proyección como Iteración del Subespacio, RQI, Arnoldi, Lanczos o Krylov-Schur. Además, opcionalmente ofrece una interfaz hacia métodos de otras librerías como ARPACK, PRIMME, BLZPACK[14], TRLAN[15] y BLOPEX[16].

SLEPc se apoya en PETSc (*Portable, Extensible Toolkit for Scientific Computation*), proporcionándole la filosofía y las estructuras de datos básicas que fueron diseñadas para resolver problemas numéricos de ecuaciones en derivadas parciales. Cabe destacar los siguientes rasgos:

- usa un estilo de programación orientado a objetos, facilitando su extensión y mantenibilidad;
- la implementación de la estructura de datos es neutra permitiendo, por ejemplo, ser compilado para precisión simple, doble o aritmética compleja;
- es muy flexible en tiempo de ejecución, con el objetivo de poder ajustar ciertos parámetros (como el *solver* lineal a emplear, el precondicionador...) sin necesidad de recompilar;
- es portable a una gran variedad de plataformas paralelas, que soporten MPI;
- ofrece interfaz a FORTRAN y a otros lenguajes que puedan interoperar con C, como C++ y Python.

[13] http://www.grycap.upv.es/slepc
[14] http://crd.lbl.gov/~osni/
[15] http://crd.lbl.gov/~kewu/trlan.html
[16] http://www-math.cudenver.edu/~aknyazev/software/BLOPEX/

4.4.1. Descripción de la filosofía de orientación a objetos de PETSc y SLEPc

El código está escrito de manera independiente a las estructuras de datos para facilitar reutilización y flexibilidad. La librería está organizada jerárquicamente, permitiendo a los usuarios emplear el nivel de abstracción más apropiado para su problema particular (figura 4.3).

Gran parte del código consiste en crear objetos, llamar a funciones que usan esos objetos, y destruirlos. A modo de ejemplo, el siguiente código crea un vector distribuido y un objeto de números aleatorios, e inicializa el vector con valores aleatorios:

```
1  Vec         b;          /* vector */
2  PetscRandom rctx;       /* generador de aleatorios */
3  PetscInt    N;          /* tamaño del problema */
4  PetscErrorCode ierr;    /* código de error */
5
6  /* Crea un vector distribuido de longitud N */
7  ierr = VecCreateMPI(PETSC_COMM_WORLD, PETSC_DECIDE, N, &b); CHKERRQ(ierr);
8  ierr = VecSetFromOptions(b); CHKERRQ(ierr);
9
10 /* Crea un generador de número aleatorio distribuido */
11 ierr = PetscRandomCreate(PETSC_COMM_WORLD, &rctx); CHKERRQ(ierr);
12 ierr = PetscRandomSetType(rctx, PETSCRAND48);CHKERRQ(ierr);
13
14 /* Inicializa el vector b con valores aleatorios */
15 ierr = VecSetRandom(b, rctx); CHKERRQ(ierr);
16
17 /* Destruye el vector y el generador */
18 ierr = PetscRandomDestroy(rctx); CHKERRQ(ierr);
19 ierr = VecDestroy(b); CHKERRQ(ierr);
```

Normalmente el nombre de las funciones empieza por el tipo de objetos que referencia. Así `VecCreate` crea un `Vec` (objeto vector) y `PetscRandomDestroy` destruye un objeto `PetscRandom`.

A la creación de un objeto le suele seguir una serie de funciones que modifican sus atributos. PETSc permite que se puedan modificar los atributos por defecto de los objetos, incluso desde la línea de comandos, si se invoca a `*SetFromOptions` en el código.

La implementación concreta que utilizará un objeto se determina con el *tipo*. El tipo se puede indicar de dos maneras:

- hay funciones específicas de creación para determinados tipos. En el ejemplo, `VecCreateMPI` crea un vector distribuido basado en MPI. También se podría haber creado un vector local con la llamada a la función `VecCreateSeq`;
- si se desea realizar una implementación más genérica, se puede crear un objeto genérico. En el ejemplo anterior se creaba un generador aleatorio genérico invocando a `PetscRandomCreate`. Luego se especificaba el tipo con `PetscRandomSetType`, u opcionalmente desde la línea de comandos, si tras la creación se invoca a `PetscRandomSetFromOptions`.

PETSc soporta cierta gestión de errores. Con ese objetivo, casi todas la funciones devuelven un tipo `PetscErrorCode`, y el usuario debería comprobarlo, por ejemplo con la macro `CHKERRQ(ierr)` que aborta la ejecución y muestra un mensaje de error.

También trata de mejorar la portabilidad y abstracción de los tipos básicos en C, encapsulándolos en macros. Por ejemplo, `PetscInt` es un entero, y `PetscReal` puede ser un `double` o un `float` dependiendo de las opciones de compilación de la librería. Esto era especialmente importante en relación a los número de complejos, cuando ANSI C no daba soporte a los mismos.

La figura 4.3 muestra las principales clases de PETSc y SLEPc. Seguidamente se describe someramente la función de las clases que se han necesitado para implementar Minimización de la Traza:

`IS` conjuntos de índices para, por ejemplo, indexar vectores y matrices.

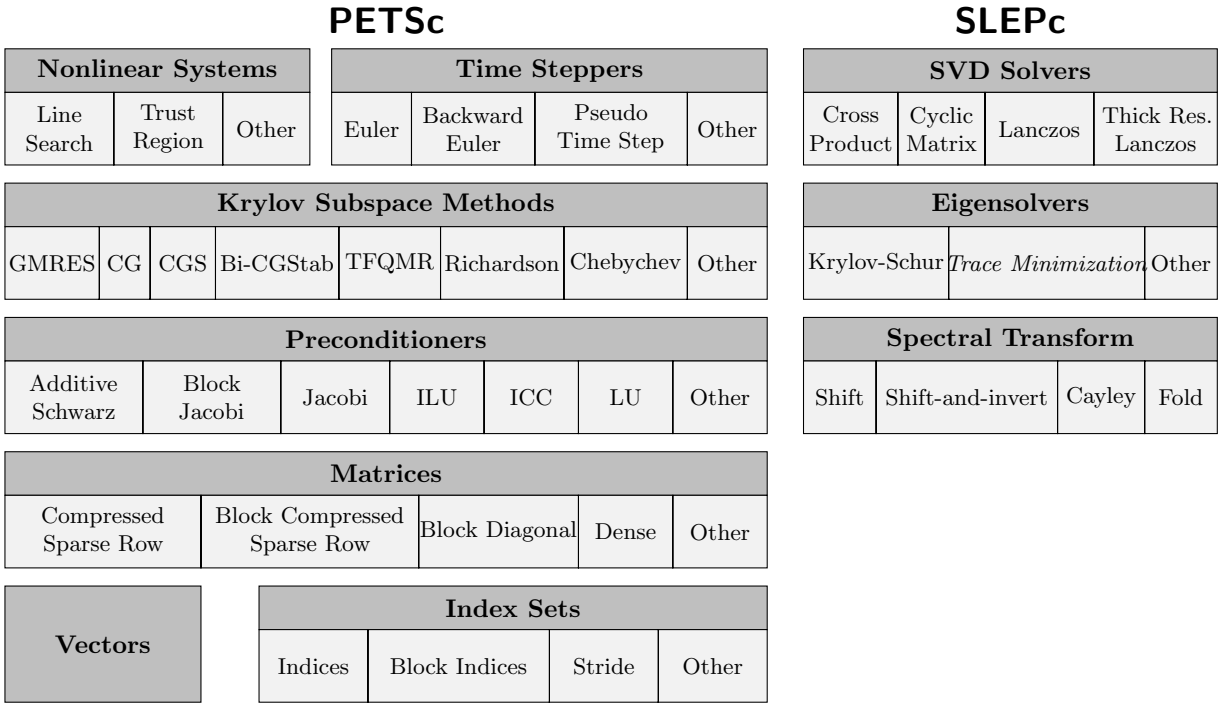


Figura 4.3: Organización de los paquetes PETSc y SLEPc.

`Vec` objetos que representan vectores algebraicos, y soportan las operaciones típicas sobre vectores: suma, producto, etc. Existen implementaciones secuenciales y paralelas.

`Mat` objetos que representan matrices algebraicas, y como los vectores, soportan operaciones típicas entre ellas y con vectores. Hay una gran variedad de implementaciones: densas, dispersas, dispersas simétricas, diagonal a bloques, dispersas a bloques, etc.

`KSP` métodos iterativos de resolución de sistemas lineales basados en subespacios de Krylov, tales como Gradiente Conjugado, GMRES, Bi-CGSTAB, etc.

`PC` encapsulan precondicionadores que pueden ser usados con los `KSP`. Actualmente hay implementaciones de ILU, ICC, Jacobi a bloques y Schwart aditivo, entre otros. Además, se tiene acceso a precondicionadores de otras librerías como HYPRE, que será usada en los experimentos.

`EPS` es el objeto principal de SLEPc, y proporciona una interfaz uniforme a diversos *solvers* de valores propios.

`ST` es un objeto de SLEPc que encapsula diversas técnicas de aceleración basadas en la transformación del espectro. Estas transformaciones son aplicadas de manera transparente a los objetos `EPS`. Su principales finalidades son:

- calcular zonas concretas del espectro (los valores propios más pequeños en magnitud, los más grandes en magnitud, los más a la derecha, los más a la izquierda, o los contenidos en un determinado intervalo),
- acelerar la convergencia (la convergencia de muchos algoritmos depende de cómo de cerca estén los valores propios en el espectro, y esto se puede modificar mediante una transformación espectral), y
- manejar algunas situaciones especiales, como que la matriz B sea singular.

$$\left[\begin{array}{c|cc|cc|c} A_1^d & \multicolumn{5}{c}{A_1^r} \\ \hline A_2^l & A_2^d & \multicolumn{4}{c}{A_2^r} \\ \hline \multicolumn{2}{c|}{A_3^l} & A_3^d & \multicolumn{3}{c}{A_3^r} \\ \hline \multicolumn{3}{c|}{A_4^l} & A_4^d & \multicolumn{2}{c}{A_4^r} \\ \hline \multicolumn{4}{c|}{A_5^l} & \multicolumn{2}{c}{A_5^d} \end{array}\right] \left[\begin{array}{c} v_1 \\ \hline v_2 \\ \hline v_3 \\ \hline v_4 \\ \hline v_5 \end{array}\right] = \left[\begin{array}{c} u_1 \\ \hline u_2 \\ \hline u_3 \\ \hline u_4 \\ \hline u_5 \end{array}\right]$$

Figura 4.4: Distribución de matrices y vectores entre los procesadores en PETSc. El procesador i tiene las filas A_i de la matriz y los elementos u_i y v_i de los vectores involucrados en el producto matriz-vector.

`IP` es un objeto auxiliar también de SLEPc que encapsula diferentes aplicaciones del producto escalar, y varias implementaciones del procedimiento de Gram-Schmidt (algunas de ellas comentadas en la sección 3.4).

Se puede obtener una descripción más detallada en los manuales de usuario de ambas librerías, [Balay et al., 2007] y [Hernandez et al., 2007].

En cuanto a la distribución de los datos, cabe destacar que la implementación de PETSc es bastante simple: se basa en una distribución orientada a bloques de filas consecutivas. Por tanto, si se tiene un vector de tamaño n y hay p procesadores, el procesador i (el primer procesador lo numeramos como 1) almacena los elementos de $n(i-1)/p+1$ (supóngase que n es divisible entre p y el primer elemento del vector también es 1) hasta ni/p. Las filas de las matrices se distribuyen de la misma manera entre los procesadores (ver figura 4.4). Para mejorar las prestaciones, el producto matriz-vector se realiza en dos fases:

1. con los datos locales, el procesador i realiza el producto de la submatriz diagonal con el vector local: $u_i \leftarrow A_i^d v_i$; y en paralelo envía los elementos locales del vector v_i que serán requeridos por otros procesadores.

2. cuando el procesador i ha terminado el producto local y le han llegado todos los elementos del vector v que poseían otros procesadores, se añade el resto de productos al resultado anterior: $u_i \leftarrow u_i + \left[A_i^l \; A_i^r\right] \left[\{v_j\}_{1,\ldots,i-1,i+1,\ldots,n}\right]$.

Es, pues, muy evidente la mejora que conllevan las técnicas de reordenación que tiendan a concentrar los valores no nulos de la matriz entorno a la diagonal, desplazando consecuentemente la carga computacional al producto matriz-vector local del primer paso, y reduciendo las comunicaciones necesarias.

Por último, y a modo de introducción del siguiente capítulo, describimos más detalladamente la estructura de SLEPc. Esta librería cuando fue diseñada sólo incluía métodos de proyección que buscaban los valores propios más grandes en problemas estándares, como el método de la potencia, Iteración del Subespacio, Arnoldi y Lanczos. Para buscar los valores propios más pequeños (o valores internos cercanos a un punto del plano complejo) o resolver problemas generalizados, se empleaban transformaciones espectrales. También son útiles para manejar situaciones especiales, como por ejemplo que B fuese una matriz singular.

En vistas del papel que iba a desempeñar la transformación espectral, se diseñó como una clase propia que contendría las matrices del problema: `ST`. La tabla 4.2 lista las transformaciones implementadas en SLEPc. La clase `ST` necesita resolver sistemas lineales en todos los casos, y esta es la razón de que tenga referencia a un objeto `KSP`, que puede ser configurado desde fuera para seleccionar el método y el precondicionador `PC` que se desee.

Como hemos dicho, la clase `EPS` contiene los métodos de resolución de valores propios, y fueron diseñados para soportar solamente problemas estándares, ya que la clase `ST` se encargaba de transformar todos los casos al problema estándar. Muchos métodos explotan la simetría de los

Transformación espectral	Operador T	Trans. de los valores propios θ
Shift	$B^{-1}A + \sigma I$	$\lambda - \sigma$
Fold	$(B^{-1}A - \sigma I)^2$	$(\lambda - \sigma)^2$
Shift-and-invert	$(A - \sigma B)^{-1}B$	$1/(\lambda - \sigma)$
Cayley	$(A - \sigma B)^{-1}(A + \tau B)$	$(\lambda + \tau)/(\lambda - \sigma)$

Tabla 4.2: Transformaciones implementadas en la clase `ST`. El sistema original $Av = \lambda Bv$ se transforma en $Tv = \theta v$.

problemas reduciendo almacenamiento y/o computación, pero al emplear las transformaciones espectrales, se pierde tal simetría en la mayoría de los casos (también es aplicable a la propiedad hermitiana). Como solución, SLEPc permite que los métodos trabajen en espacios isomorfos al euclídeo, en los que el nuevo operador T resultado de la transformación espectral sea autoadjunto. Así pues, los operadores shift, fold y shift-and-invert son autoadjuntos con respecto al producto interno con B, $\langle \cdot, \cdot \rangle_B$, y el operador de Cayley con respecto a $\langle \cdot, \cdot \rangle_{A+\tau B}$.

Esta abstracción la permite la clase `IP`, ya que las normas y la ortogonalización de vectores se hacen a través suyo. La clase `EPS` padre, conociendo el tipo de la transformación espectral, configura el objeto `IP` adecuadamente.

Finalmente, la figura 4.5 resume las relaciones entre las clases de SLEPc mediante un diagrama de clases de UML.

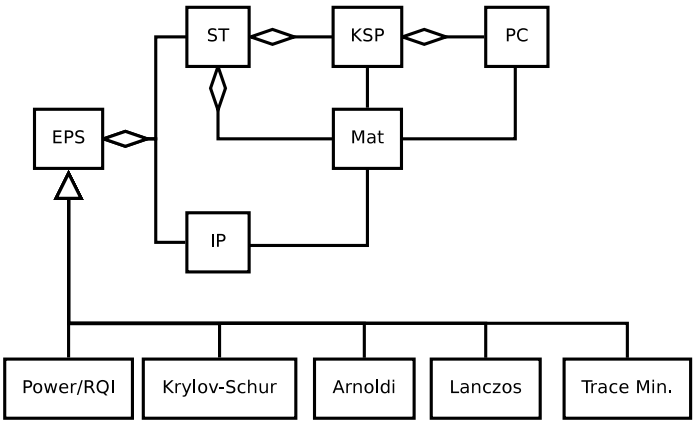


Figura 4.5: Diagrama de clases de SLEPc.

Capítulo 5

Descripción de la implementación en SLEPc

Una vez descrita la estructura básica de SLEPc y sus clases, en este capítulo se describen las opciones de interfaz y los detalles de implementación más interesantes.

5.1. Interfaz del solver

En el capítulo 3 fueron descritas las alternativas más interesantes que se planteaban al llevar a cabo una implementación robusta del método. Seguidamente se listan todas las opciones de la interfaz del solver:

`-eps_max_it` indica el máximo número de iteraciones externas.

`-eps_tol` indica el error relativo máximo de un par propio antes de ser considerado convergido. Es decir, el par propio $(\theta_{k,i}, x_{k,i})$ se considera convergido cuando la tolerancia aquí indicada es mayor que su error relativo $\epsilon_{k,i}$ definido como

$$\epsilon_{k,i} = \begin{cases} \dfrac{\|r_{k,i}\|_2}{|\theta_{k,i}|\|x_{k,i}\|_2} & \text{si } \|r_{k,i}\|_2 < |\theta_{k,i}| \\ \dfrac{\|r_{k,i}\|_2}{\|x_{k,i}\|_2} & \text{si } \|r_{k,i}\|_2 \geq |\theta_{k,i}| \end{cases}, \tag{5.1}$$

donde $r_i = Ax_{k,i} - \theta_{k,i}Bx_{k,i}$. Este criterio de convergencia es robusto incluso cuando el valor propio está muy próximo a cero.

`-eps_nev` número de valores propios que se desean.

`-eps_ncv` número de vectores en la base. En la versión clásica coincide con el número de columnas de V_k y en la de tipo Davidson es su número máximo de columnas.

`-eps_smallest_magnitude` **y** `-eps_smallest_real` para los problemas definidos positivos ambas opciones producen el mismo resultado. Pero en las baterías de problemas existen casos en los que las matrices son *ligeramente* indefinidas, y tienen por tanto algunos valores propios negativos. Esto no es un inconveniente *grave* para el algoritmo, porque la estrategia de desplazamientos es capaz de mantener en muchos casos la condición de definido positivo del sistema lineal. Entonces, si se desea obtener los valores propios más cercanos al origen se puede usar la opción `smallest_magnitude`.

`-eps_orthog_type` selecciona el algoritmo de ortogonalización que será usado, tal y como hemos visto en la sección 3.4.

`-eps_orthog_refinement` define la estrategia de parada de los algoritmos de ortonormalización:

- `none` sólo hacen una iteración;
- `ifneeded` tras cada iteración se sigue iterando si la norma del vector ortonormalizado, $\|q_j\|_2$, es menor que la norma que tenía antes por un factor, $\eta\|\tilde{q}_j\|$, hasta un límite de tres iteraciones;
- `always` siempre hace dos iteraciones.

`-eps_orthog_eta` parámetro empleado en el criterio de parada de la rutina de ortonormalización con la opción `ifneeded`, η.

`-eps_tracemin_shifting` activa la estrategia básica de desplazamientos descrita en la sección 3.1.1 para la resolución del sistema interno.

`-eps_tracemin_dyn_tol` activa la estrategia de tolerancia dinámica con la cual se termina el sistema lineal interno (ver sección 3.3). Si no se activa, el sistema termina según los parámetros predeterminados del `KSP`.

`-eps_tracemin_min_tol` en algunos casos la estrategia anterior obtiene tolerancias excesivamente permisivas retrasando la convergencia de los pares propios. Con este parámetro se indica la tolerancia mínima con la cual se configura la terminación del sistema lineal interno.

`-eps_tracemin_safe_shifting` indica el error relativo máximo que puede tener un par propio para activar la estrategia de desplazamientos (desplazamientos seguro), tal y como se explica en la sección 3.1.3.

`-eps_tracemin_min_eigv` obtiene una aproximación del valor propio más pequeño de B mediante el teorema de los discos Gershgorin. Esta opción sólo está permitida si la matriz B soporta explorar sus filas, es decir, para los casos en que la matriz sea explícita. Su utilidad fue explicada en la sección 3.1.2, y cómo se realiza el cálculo será explicado en la sección 5.2.1.

`-eps_tracemin_min_eigv_value` permite al usuario introducir una cota inferior del valor propio más pequeño de B.

`-eps_tracemin_bs` establece el valor del tamaño de bloque en la versión de tipo Davidson (parámetro s). De forma predeterminada tiene el mismo valor que el indicado en `-eps_nev`.

`-eps_tracemin_classical` activa la versión clásica (algoritmo 2). De lo contrario usa la versión de tipo Davidson (algoritmo 5).

`-st_type` es el tipo de transformación espectral. Por ahora no se permite ninguna.

`-st_ksp_type` se puede configurar el *solver* lineal empleado para resolver el sistema interno a través de las opciones que ofrece el `KSP` de `ST`. La más importante es la opción `type` que establece el método que será usado.

`-st_pc_type` de la misma manera que en la opción anterior, podemos configurar el precondicionador que empleará el *solver* lineal.

5.2. Detalles de implementación y decisiones de diseño

Por último, en esta sección se dibujan los aspectos de implementación de más bajo nivel que pueden ser interesantes y que aplican ciertas consideraciones sobre las arquitecturas y los modelos de programación que fueron descritos en el capítulo 4.

5.2.1. Discos de Gershgorin

Si activamos la opción `-eps_tracemin_min_eigv`, el *eigensolver* calculará una cota inferior del valor propio más pequeño de B teniendo en cuenta que todos sus valores propios λ deben cumplir

$$|\lambda - b_{ii}| \leq \sum_{i \neq j} |b_{ij}| \ . \tag{5.2}$$

Entonces, podemos obtener una cota inferior como la siguiente:

$$c = \min_i \left[b_{ii} - \sum_{i \neq j} |b_{ij}| \right] \leq \lambda \ . \tag{5.3}$$

El error que se comete, y que se emplea para estimar la calidad de la cota muy burdamente es el siguiente:

$$\frac{|\lambda - c|}{\lambda} \leq 2\frac{\sum_{i \neq j} |b_{ij}|}{\lambda} \leq 2\frac{\sum_{i \neq j} |b_{ij}|}{c} = \epsilon \ , \tag{5.4}$$

siendo i la fila que obtuvo el mínimo en el cálculo de c.

El cálculo de c y ϵ se realiza en paralelo de la siguiente manera:

1. cada procesador p realiza el cómputo de c, y su correspondiente error ϵ con sus filas locales;
2. todos los procesadores realizan un reducción empleando el siguiente operador f:

$$f(\{c_0, \epsilon_0\}, \{c_1, \epsilon_1\}) = \begin{cases} \{c_0, \epsilon_0\} & \text{si } c_0 \leq c_1 \\ \{c_1, \epsilon_1\} & \text{si } c_0 > c_1 \end{cases} \ . \tag{5.5}$$

MPI soporta reducciones con operadores definidos por el usuario, como es este caso. Además son tan flexibles como para permitir un operador con parámetros con estructura. Normalmente las reducciones se implementan en árbol, lo cual requiere tantas rondas de mensajes como el logaritmo del número de procesadores que intervengan.

5.2.2. Procedimiento de Rayleigh-Ritz

La implementación del procedimiento de Rayleigh-Ritz es también bastante directa. Tanto V_k, W_k, X_k como R_k son conjuntos de vectores de PETSc. No se implementan como matrices porque los métodos para acceder a submatrices en PETSc son *ineficientes* (como en general cualquier operación que requiera creación de objetos).

Entonces, el paso 1 de los algoritmos de Minimización de la Traza (algoritmos 2 y 5) se realiza mediante productos matriz-vector y productos escalares de un vector por varios vectores (de esta manera se reducen las comunicaciones). Estos últimos se van almacenando en la matriz H_k que es de pequeño tamaño (como el número de vectores en V_k) y está replicada en todos los procesadores. La resolución del sistema de valores propios proyectado (paso 2 de los algoritmos) también se resuelve individualmente en cada procesador siguiendo los siguientes pasos:

1. se reduce la matriz H_k a forma de Hessenberg superior T y se almacenan las transformaciones llevadas a cabo en U con las funciones `xGEHRD` y `xORGHR` de LAPACK;
2. se obtiene la forma de Schur mediante la función `xHSEQR` de LAPACK;
3. se ordena los bloques diagonales de T de menor a mayor con la función `xTREXC` también de LAPACK.

Por último, se actualizan los vectores V_k con los vectores de Schur, como se indica en el paso 3 (que no requiere ningún tipo de comunicación), y se calculan los residuos (que sí requiere un producto matriz-vector con la matriz B).

Por tanto, las comunicaciones se reducen a productos matriz por un conjunto de vectores, y productos internos entre conjuntos de vectores. Por ahora, PETSc no implementa *multivectores*, es decir, no permite operaciones con múltiples vectores que mejorarían bastante las prestaciones tanto en ahorro de comunicaciones (se compactarían muchos mensajes en uno sólo) como en prestaciones con la caché (el producto interno entre conjunto de vectores se podría implementar con BLAS de nivel 3). Las únicas excepciones que han sido empleadas son las funciones `VecMDot` que obtiene el producto escalar de un vector por varios vectores, y el par `VecNormBegin` y `VecNormEnd` que compacta los mensajes de reducción necesarios para calcular la norma de una serie de vectores.

5.2.3. Resolución del sistema interno

Como se describió en la sección 3.2.2, el operador del método iterativo para resolver sistemas lineales basado en los subespacios de Krylov tenía que aplicar sobre un vector entrante v la siguiente transformación:

$$z = Fv, \text{ donde } F = \left(I - P^{-1}\tilde{X}\left(\tilde{X}^* P^{-1}\tilde{X}\right)^{-1}\tilde{X}^*\right)P^{-1}(A - \sigma_{k,i}B) \ . \tag{5.6}$$

Sin embargo, los `KSP` permiten configurar tres operadores que se aplican en cada iteración interna:

- la matriz del sistema lineal A^{KSP},
- el precondicionador K^{KSP}, y
- el espacio nulo N^{KSP} que ortogonaliza los vectores con respecto al subespacio que expanden.

De todas formas los dos últimos operadores se aplican juntos y en el orden $N^{KSP}K^{KSP}$.

La cuestión está en repartir el operador descrito en la ecuación 5.6 en los tres operadores que se acaban de describir. Una opción podría ser la que sigue:

$$F = \overbrace{\left(I - P^{-1}\tilde{X}\left(\tilde{X}^* P^{-1}\tilde{X}\right)^{-1}\tilde{X}^*\right)}^{N^{\text{KSP}}}\overbrace{P^{-1}}^{K^{KSP}}\overbrace{(A - \sigma_{k,i}B)}^{A^{KSP}} \ , \tag{5.7}$$

pero tiene como inconveniente que N^{KSP} sería un proyector oblicuo, y no hay funciones directas para construir tal proyector. La solución que se ha adoptado es

$$F = \overbrace{\left(I - P^{-1}\tilde{X}\left(\tilde{X}^* P^{-1}\tilde{X}\right)^{-1}\tilde{X}^*\right)P^{-1}}^{N^{\text{KSP}}}\overbrace{I}^{K^{KSP}}\overbrace{(A - \sigma_{k,i}B)}^{A^{KSP}} \ , \tag{5.8}$$

porque se puede construir N^{KSP} de la siguiente manera:

$$\begin{aligned} N^{KSP} &= \left(I - P^{-1}\tilde{X}\left(\tilde{X}^* P^{-1}\tilde{X}\right)^{-1}\tilde{X}^*\right)P^{-1} && (5.9)\\ &= P^{-1} - P^{-1}\tilde{X}\left(\tilde{X}^* P^{-1}\tilde{X}\right)^{-1}\tilde{X}^* P^{-1} && (5.10)\\ &= P^{-1} - \tilde{J}\tilde{J}^* \ , && (5.11) \end{aligned}$$

donde $\tilde{J} = P^{-1}\tilde{X}$, y $\tilde{X}$ es una base P^{-1}-ortonormal de BX_k. Para hacer esto último, se crea un objeto `IP` que realiza los productos internos con respecto al precondicionador, $\langle \cdot, \cdot \rangle_{P^{-1}}$.

Capítulo 6

Experimentos y resultados

En este capítulo se describen los experimentos llevados a cabo y sus resultados con el objetivo de poner a prueba la efectividad de las diferentes técnicas desarrolladas en capítulos anteriores.

6.1. Máquinas y colecciones de problemas

Las matrices usadas en las pruebas pertenecen a las colecciones de *Matrix Market* y de la *University of Florida Sparse Matrix Collection*. Todos los problemas corresponden a sistemas de valores propios reales, simétricos y generalizados procedentes de aplicaciones reales. El cuadro 6.1 recoge todos los problemas empleados en pruebas secuenciales y sus características, mientras que el cuadro 6.2 describe tres matrices bastante más grandes empleadas en medir prestaciones en paralelo.

Los experimentos tanto secuenciales como paralelos que tenían como objetivo medir tiempos fueron ejecutados en dos clústers:

- *Odin*, compuesto por 55 nodos bi-procesadores Pentium Xeon a 2,80 GHz. Los nodos están conectados formando una topología de toro 2D con tecnología SCI.
- *MareNostrum*, que consiste en un supercomputador con 2.560 nodos de computación blade JS21, cada uno con 2 procesadores de doble núcleo IBM 64-bits PowerPC 970MP a 2,3 GHz, interconectados con tecnología Myrinet de baja latencia.

Aunque en [Sameh y Tong, 2000] el método CG es usado para resolver el sistema interno del Método de Minimización, sólo unos pocos problemas de las baterías de test anteriormente expuestas pueden ser resueltos con esta configuración. A menos que se indique lo contrario, todos los experimentos aquí expuestos emplearan el *solver* lineal GMRES con un precondicionador multimalla algebraico. Este precondicionador es uno de los que proporciona la librería HYPRE [Henson y Yang, 2002], bajo el nombre *boomeramg*. Con esta configuración, todos los casos de test de las tablas 6.1 y 6.2 son resueltos sin problemas de convergencia.

6.2. Medidas de prestaciones

Los experimentos recogen diferentes parámetros de las ejecuciones, que generalmente han sido el tiempo, el número de iteraciones externas y el número de iteraciones internas. Las evaluaciones en secuencial tienen como objetivo contrastar los costes de ambas versiones usando las distintas optimizaciones con diversas configuraciones, y por supuesto, probando todas las matrices de la batería de test. Esto supone un número alto de experimentos (sólo en secuencial). Si usamos el tiempo como medida del *coste* asociado a la ejecución, debemos además repetir el experimento varias veces y, mediante un estadístico robusto, obtener un valor representativo de la muestra.

Una manera de reducir el número de ejecuciones, es tomar el número de iteraciones internas como medida del coste asociado a la resolución del caso, ya que directamente es un parámetro muy robusto:

Tabla 6.1: Batería de test para experimentos secuenciales (p: definida positiva, s: semidefinida positiva, i: indefinida). La columna etiquetada *speed-up* muestra el factor de ganancia de la versión de tipo Davidson (algoritmo 5) con respecto a la versión clásica (algoritmo 2).

		A		B		
	tamaño	no nulos	definida	no nulos	definida	speed-up
BCSST02	66	2,211	p	66	p	0.81
BCSST08	1,074	7,017	p	1,074	p	2.80
BCSST09	1,083	9,760	p	1,083	p	2.21
BCSST10	1,086	11,578	p	11,589	i	1.31
BCSST11	1,473	17,857	p	1,473	p	0.90
BCSST12	1,473	17,857	p	10,566	s	1.54
BCSST13	2,003	42,943	p	11,973	s	1.47
BCSST19	817	3,835	i	817	p	–
BCSST21	3,600	15,100	i	3,600	p	–
BCSST22	138	417	i	138	p	–
BCSST23	3,134	24,156	i	3,134	p	–
BCSST24	3,562	81,736	i	3,562	s	–
BCSST25	15,439	133,840	i	15,439	p	–
BCSST26	1,922	16,129	i	1,922	p	–
BCSST27	1,224	28,675	p	28,675	i	1.26
BCSST38	8,032	355,460	p	10,485	i	–

Tabla 6.2: Problemas usados en pruebas de prestaciones en paralelo.

	tamaño	no nulos en A	no nulos en B
DIAMON5	19,200	9,347,698	3,115,256
BS01	127,224	6,715,152	2,238,384
GYRO	17,361	1,021,159	340,431

la misma implementación, configuración y caso de prueba realiza en general el mismo número de iteraciones internas. Las variaciones, que son despreciables, se deben a que la solución inicial es una base de vectores aleatorios (la influencia del *solver* lineal es limitada ya que, como hemos comentado, parte de una solución inicial nula).

El número de iteraciones externas también es un parámetro robusto, pero como podemos apreciar en la figura 6.1, existe una relación lineal más clara entre el tiempo y el número de iteraciones internas que externas.

Sin embargo, en los experimentos para medir prestaciones en paralelo seguimos usando el tiempo como referencia del coste del experimento, pero esta vez, del nodo más lento. Como estadístico robusto se usa el mínimo de todas las repeticiones (podemos asumir que el mínimo es robusto cuando se miden tiempos).

6.3. Evaluación en secuencial

6.3.1. Parámetros m y s

Hemos realizado un estudio estadístico sobre el impacto de los parámetros m y s sobre el número de iteraciones internas del Método de Minimización de la traza tipo Davidson. Se han resuelto los problemas listados en la tabla 6.1 con $m = \{20, 30, 40, 60, 80, 120, 160\}$ y $s = \{5, 10, 20, 40, 80\}$, buscando los 10 pares propios más pequeños. Excepto en los problemas BCSST{02,19}, la configuración con mejores prestaciones tenía $s = 10$.

La figura 6.2 muestra el número de iteraciones internas conforme aumenta el parámetro m para los distintos problemas, fijando el tamaño de bloque $s = 10$. Se observa que el aumento del tamaño

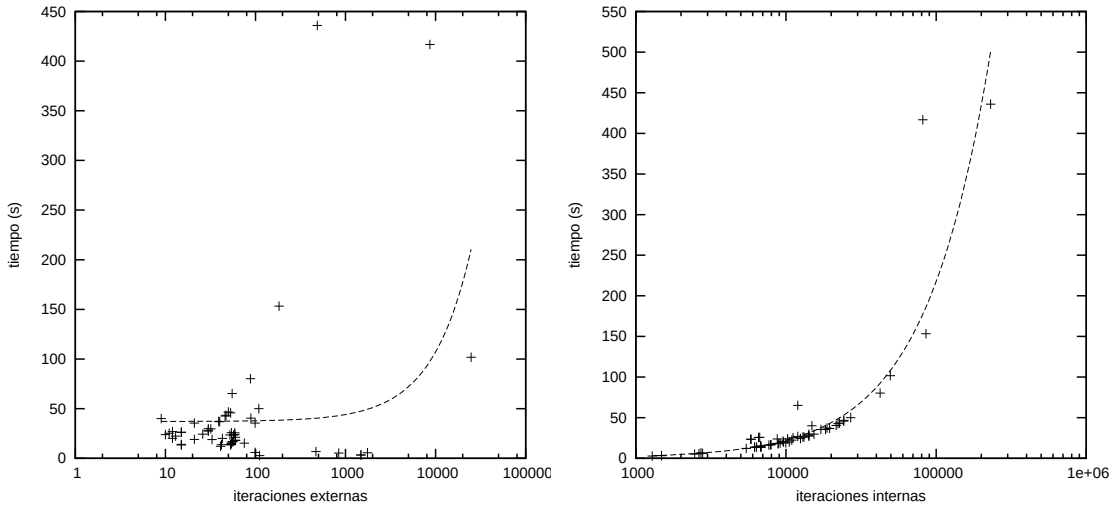


Figura 6.1: Las gráficas representan el número de iteraciones (externas en la figura de la izquierda e internas en la figura de la derecha) y el tiempo empleados en resolver el problema BCSST10 por la implementación de la versión de Davidson usando diversas configuraciones y optimizaciones.

máximo del subespacio, en general, supone una disminución del número de iteraciones internas hasta un cierto límite. Se observa en esta prueba que valores de m entre 40 y 80 ofrecen mejores resultados que con m fijada a 20.

6.3.2. Evaluación de los *solvers* lineales de PETSc

Se pueden encontrar distintos métodos recomendados para resolver sistemas lineales proyectados con restricciones, como el sistema 2.8, usando precondicionamiento. Así por ejemplo, en [Sameh y Wisniewski, 1982] (la referencia a la versión clásica de Minimización de la Traza) resuelven el sistema lineal con CG, dejando las consideraciones sobre el precondicionamiento en el artículo [Sameh y Tong, 2000], en el cual se muestran resultados usando CG y GMRES en el sistema lineal precondicionado a la manera descrita en la sección 3.2. Tal estrategia resultó muy poco efectiva en las baterías de problemas anteriormente expuestas. Por último, el artículo [Sleijpen et al., 1998] que describe un método de precondicionado para resolver la ecuación de corrección (un sistema lineal con restricción ortogonal) recomienda emplear un método de Krylov que soporte sistemas indefinidos, como por ejemplo GMRES o BiCGStab [van˜der Vorst, 1992].

La figura 6.3 muestra los resultados de aplicar diferentes *solvers* lineales para resolver el sistema lineal. Gracias a la flexibilidad de PETSc, esto es tan sencillo como lanzar el mismo programa variando un argumento. Podemos observar que el último artículo tenía razón: los mejores resultados son ofrecidos por los solvers *bcgs* y *bcgsl*, que corresponden a una implementación de BiCGStab y a una variante suya descrita en [Sleijpen y Fokkema, 1994], respectivamente.

6.3.3. Versión clásica respecto a versión tipo Davidson

Como fue señalado en la sección 2.3, la versión de tipo Davidson (algoritmo 5) no solamente reduce el coste de cada iteración interna (ya que X_k tiene menos vectores en esta versión), sino que también el número total de las mismas. Con el objetivo de aportar evidencias que apoyen esta última afirmación, el cuadro 6.1 lista una comparación de ambas versiones resolviendo la batería de problemas. La última columna de la tabla muestra el factor de ganancia de la versión de tipo Davidson frente a la clásica, excepto para aquellos problemas en los que la versión clásica no converge. Llama la atención que en la mayoría de los casos donde esta última versión no converge, la matriz A del sistema sea indefinida.

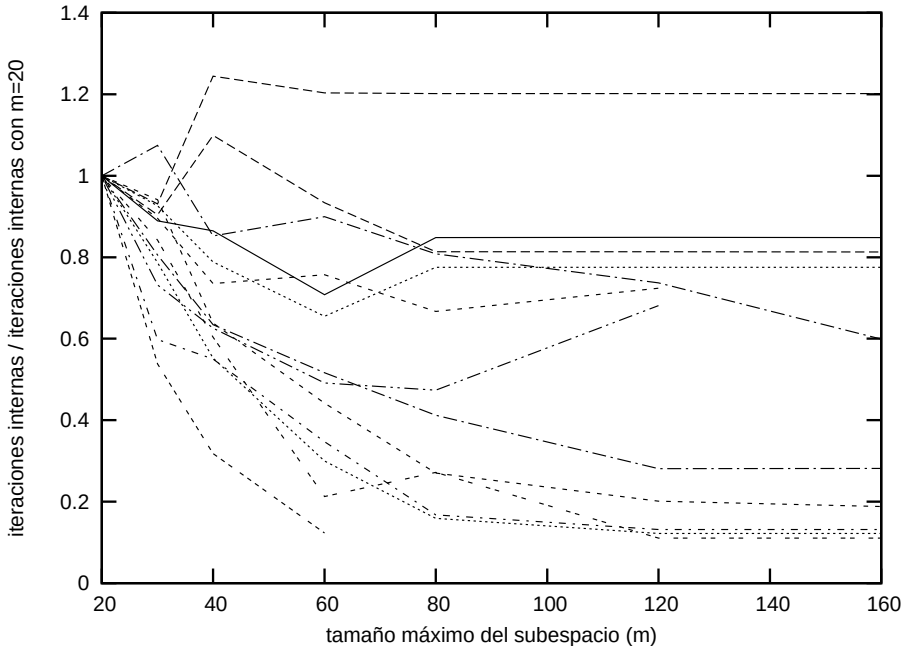


Figura 6.2: Gráfica de la evolución del número iteraciones internas para distintos tamaños máximos de subespacio m normalizadas con el caso $m = 20$, en la versión de tipo Davidson.

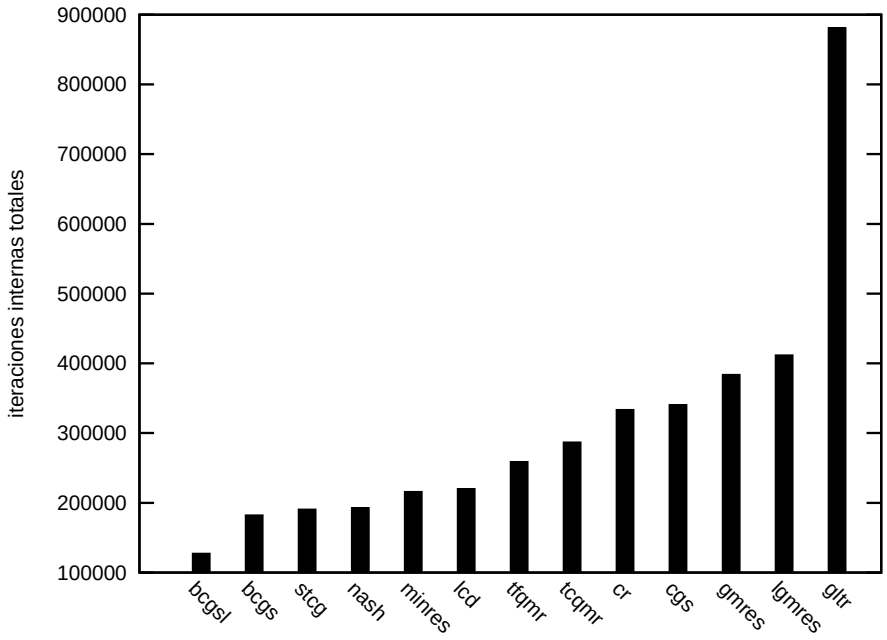


Figura 6.3: Total de iteraciones internas empleadas para resolver los problemas listados en la tabla 6.1 (excepto BCSST{19,24,38}) usando diferentes *solvers* lineales disponibles en PETSc.

Como se verá en las figuras 6.9 y 6.11, el comportamiento de ambas versiones en paralelo es muy similar, lo cual sugiere que tienen parecidos patrones de comunicación. A esta conclusión también se llega comparando directamente los algoritmos: sus estructuras son muy similares, y tienen las mismas operaciones que requieren comunicación en el mismo orden.

Teniendo en cuenta este último hecho y las limitaciones de recursos computacionales, el análisis de las optimizaciones y su prestaciones en paralelo se centrará en la versión de tipo Davidson.

6.3.4. Análisis de las optimizaciones

El el capítulo 3 se describen varias técnicas que presumen de acelerar o hacer más robusta la implementación. En concreto, este apartado se centra en la estrategia de desplazamiento y en el criterio de parada del *solver* lineal. En cuanto al resto de optimizaciones que se comentan en dicho capítulo, se destaca que el precondicionamiento es necesario para resolver todos los casos propuestos en las baterías, y sólo es efectiva la alternativa basada en las técnicas empleadas sobre la ecuación de corrección de Jacobi-Davidson (descrito en la sección 3.2.2). Además, de los métodos de ortogonalización descritos en 3.4 se usa la versión iterativa de Gram-Schmidt clásico, aquella que tiene mejores prestaciones en paralelo.

En este apartado presentaremos los resultados de aplicar las diferentes técnicas de desplazamientos, el criterio de parada del *solver* lineal, y el comportamiento de la combinación de varias técnicas, sobre la batería de test expuesta en el cuadro 6.1.

Como se señaló en el apartado 3.1.1, la estrategia básica de desplazamientos (ahí descrita) no ofrece buenos resultados. La figura 6.4 muestra que sólo es efectiva en 4 de los 14 casos. La primera mejora propuesta consistía en corregir la 2-norma del residuo por un factor de manera que se aproximara más a la B^{-1}-norma, a la cual nos referimos como desplazamiento corregido (descrito en la sección 3.1.2). Esta opción necesita una cota inferior del valor propio más pequeño de B, que se puede obtener mediante el teorema de los discos de Gershgorin o directamente calculando los valores propios más pequeños de B. Esta última figura muestra que esta mejora es efectiva en 8 de 14 casos, y que la cotas obtenidas por el teorema de Gershgorin son en general suficientes, pues obtiene casi tan buenos resultados como las otras.

También se observó que los peores desplazamientos se producían en las primeras iteraciones externas del método. En el apartado 3.1.3, se propone activar la estrategia sólo cuando el par de Ritz tenga un cierto error relativo (lo que se llamó desplazamiento seguro). La figura 6.5 muestra los resultados con diversas cotas. Se observa que no hay ninguna que sea efectiva en todos los casos, pero al menos, alguna es efectiva en 10 de los 14 casos. Se tomará la cota de 10^{-4} como predeterminada.

La figura 6.6 compara las técnicas de desplazamiento corregido y seguro, y la combinación de ambas. Se observa que ninguna de ellas es significativamente mejor que las otras.

El *solver* lineal decide que se ha alcanzado la convergencia cuando el residuo se reduce por debajo de un valor (al que nos referimos como tolerancia) que es calculado como se describe en 3.3. La figura 6.7 muestra que esta estrategia no es efectiva en ningún caso (comparada con fijar la tolerancia a 10^{-5}). Se observó que la causa de su mal funcionamiento era que a veces producía tolerancias muy altas y el sistema lineal realizaba muy pocas iteraciones. Como solución, la implementación controla que la tolerancia sea menor que una determinada cota. La anterior figura compara la efectividad de diferentes cotas. Con una cota de 0,1 se obtienen excelentes resultados.

Por último, en la figura 6.8 se compara la efectividad de la combinación del desplazamiento corregido y seguro con el control de la tolerancia, y una combinación de todo. Se observa que la tolerancia es, en general, más robusta que las otras dos. Aunque las estrategias de desplazamiento obtienen el mejor resultado en los problemas BCSST13 y BCSST19 que son bastante difíciles de resolver.

6.4. Evaluación de las prestaciones en paralelo

Empezamos el análisis de las prestaciones paralelas con un estudio de escalabilidad, es decir, midiendo el speed-up para diferente número de procesadores resolviendo un problema con un tamaño

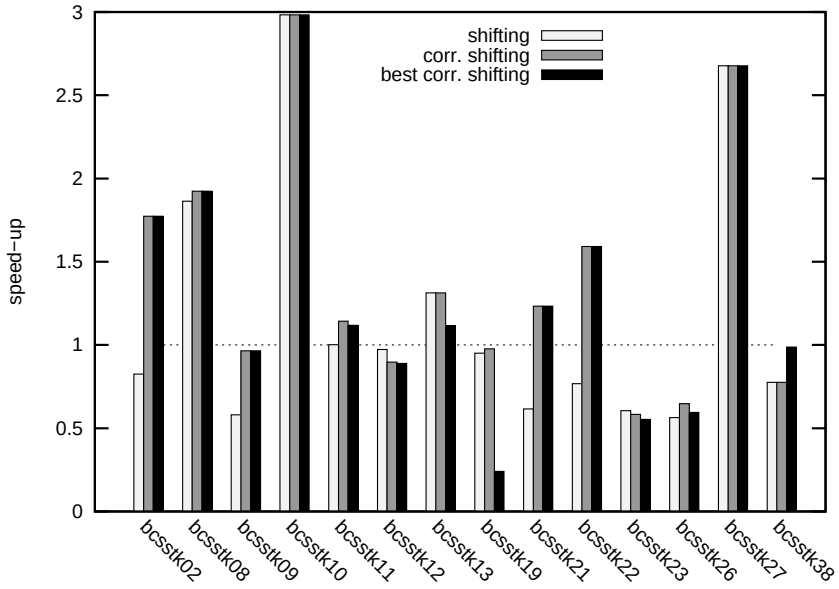


Figura 6.4: Factor de ganancia con respecto a no emplear ninguna optimización al activar la estrategia de desplazamientos (`shifting`), y la versión con las normas corregidas usando la cota del valor propio más pequeño de B obtenida, por un lado, mediante los discos de Gershgorin (`corr. shifting`), y por otro, mediante el *solver* Krylov-Schur (`best corr. shifting`). La ganancia usa el número de iteraciones internas.

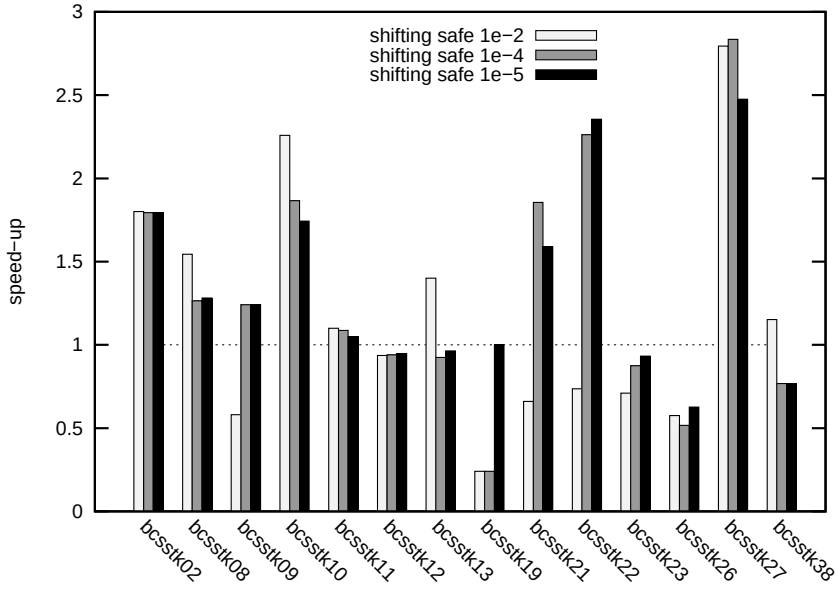


Figura 6.5: Factor de ganancia con respecto a no emplear ninguna optimización al activar la estrategia de desplazamientos cuando el error relativo del par propio es menor que 10^{-2} (`safe 1e-2 shifting`), 10^{-4} (`safe 1e-4 shifting`) y 10^{-5} (`safe 1e-5 shifting`). La ganancia usa el número de iteraciones internas.

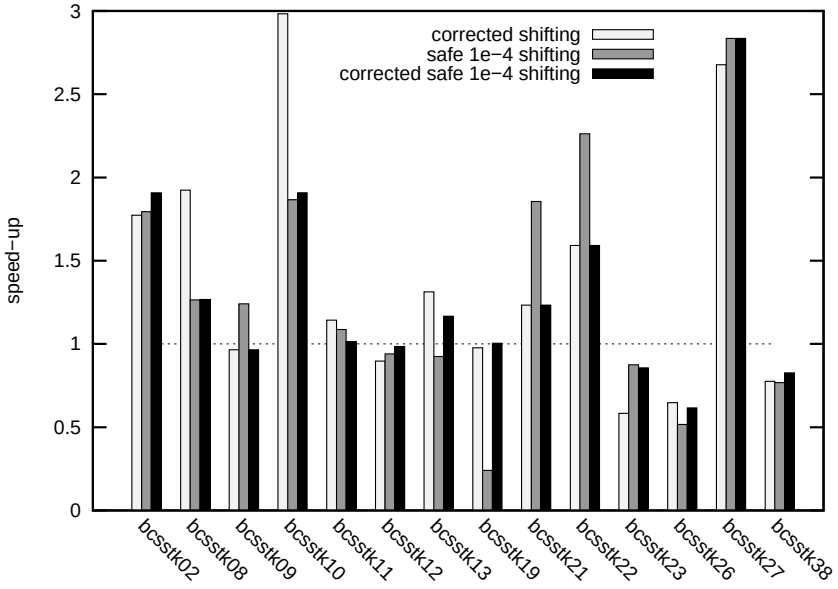


Figura 6.6: Factor de ganancia con respecto a no emplear ninguna optimización al activar la estrategia de desplazamientos, por un lado, con las normas corregidas usando la cota del valor propio más pequeño de B obtenida mediante los discos de Gershgorin (`safe 1e-4 shifting`), por otro, cuando el error relativo del par propio es menor que 10^{-4} (`safe 1e-4 shifting`), y con la combinación de ambas (`corrected safe 1e-4 shifting`). La ganancia usa el número de iteraciones internas.

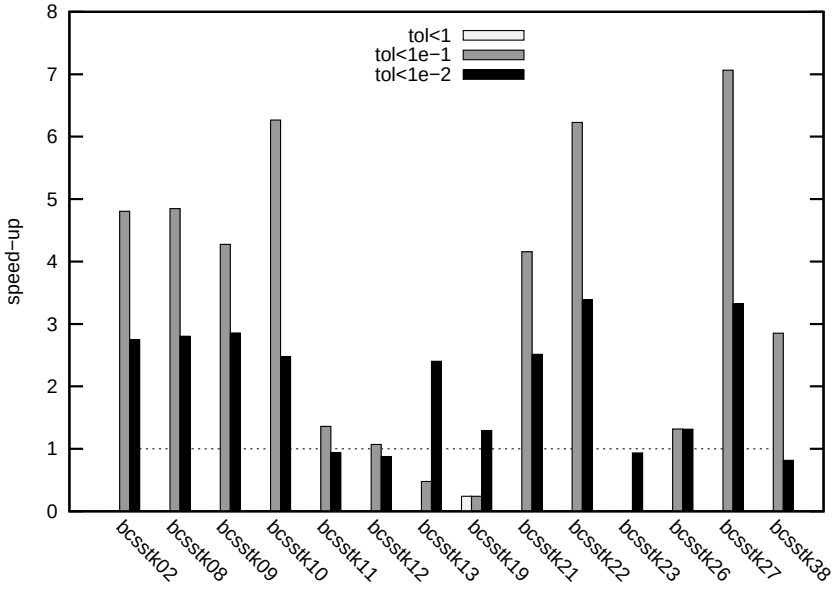


Figura 6.7: Factor de ganancia con respecto a no emplear ninguna optimización de la estrategia de tolerancia con tolerancia máxima 1 (`tol<1`), 0,1 (`tol<1e-1`) y 0,01 (`tol<1e-2`). La ganancia usa el número de iteraciones internas.

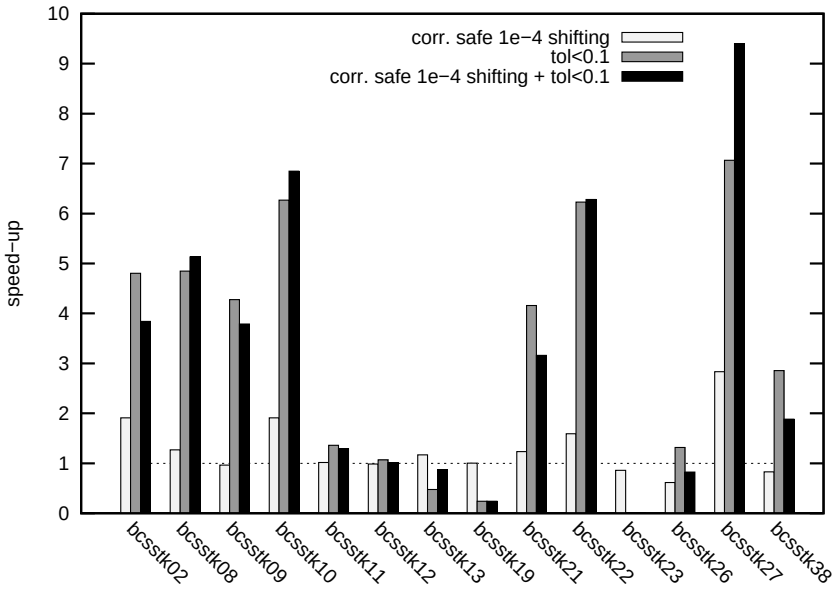


Figura 6.8: Factor de ganancia con respecto a no emplear ninguna optimización de la estrategia de desplazamiento corregido y seguro (`corr. safe 1e-4 shifting`), control de la tolerancia (`tol<0.1`) y combinación de ambas (`corr. safe 1e-4 shifting + tol<0.1`). La ganancia usa el número de iteraciones internas.

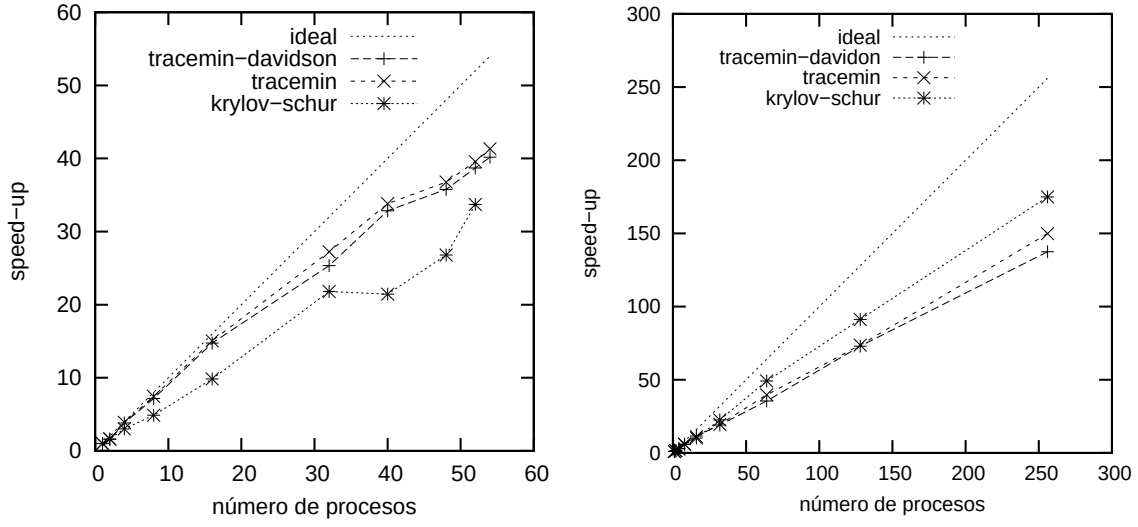


Figura 6.9: Escalabilidad de la implementación de Minimización de la Traza (`tracemin`), versión tipo Davidson (`tracemin-Davidson`) y Krylov-Schur (`krylov-schur`) resolviendo un problema de valores propios tridiagonal simétrico en las máquinas Odin (izquierda) y MareNostrum (derecha).

Tabla 6.3: Porcentaje de tiempo de ejecución correspondiente a las cuatro operaciones que más tiempo consumen resolviendo en problema de valores propios tridiagonal simétrico en MareNostrum, usando 128 procesadores.

	VecMAXPY	MatMult	VecMDot	VecNorm
Krylov-Schur	32 %	19 %	10 %	10 %
Minimización de la Traza (tipo Davidson)	29 %	26 %	24 %	8 %

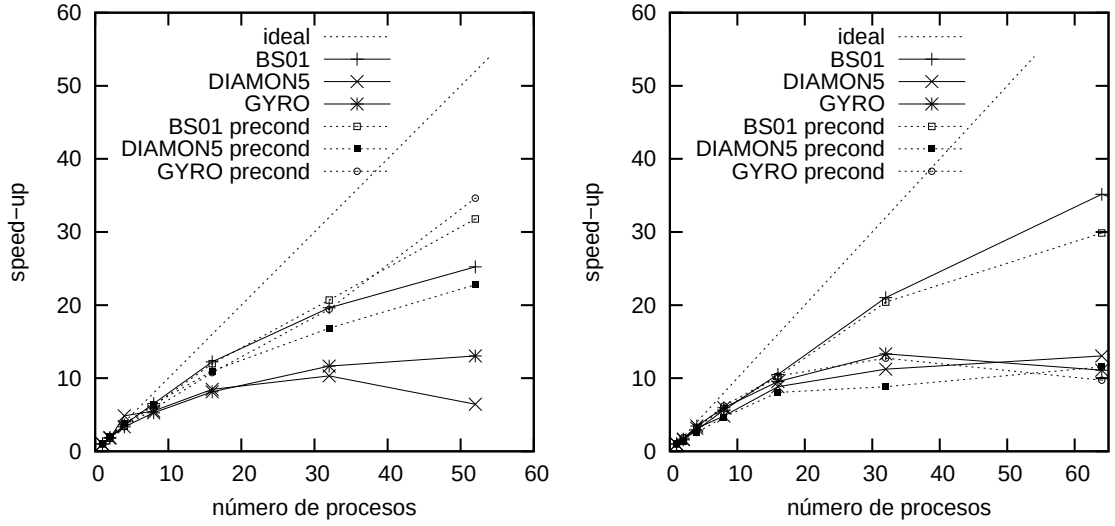


Figura 6.10: Speed-up de la versión de tipo Davidson resolviendo los problemas BS01, DIAMON5 y GYRO en Odin (izquierda) y MareNostrum (derecha). Se representa junto al speed-up de la aplicación del precondicionador.

adecuado para que se conserve la carga por procesador. La figura 6.9 presenta los resultados de escalabilidad en ambos clústers de las dos implementaciones de Minimización de la Traza, resolviendo un problema con matrices A y B tridiagonales, simétricas y aleatorias aunque diagonalmente dominantes. Además se compara con la implementación de Krylov-Schur disponible en SLEPc (usando la transformación espectral desplazamiento e inversión para que converjan los valores propios más pequeños). En este caso, como el problema es muy sencillo se usa un precondicionador de Jacobi que escala muy bien y es muy efectivo en matrices diagonalmente dominantes.

La figura muestra que ambas versiones de Minimización de la Traza escalan razonablemente igual de bien, lo cual sugiere que los patrones de comunicación de ambas implementaciones son muy parecidos (resultado que se deduce también de comparar las estructuras de ambos algoritmos). En Odin, las versiones de Minimización de la Traza escalan un poco mejor que Krylov-Schur, pero los papeles se intercambian en MareNostrum. Esta diferencia de comportamiento podría ser explicada por el hecho de que Minimización de la Traza emplea más tiempo en productos escalares de vectores (VecMDot y VecNorm) que en productos de una matriz por un vector (MatMult) y en adición de vectores (VecMAXPY), como se deduce de los resultados de la tabla 6.3. Los productos de vectores y las normas escalan peor, pues requieren una operación colectiva de reducción sobre todos los procesadores, mientras que las operaciones VecMAXPY su paralelización es trivial y los productos matriz dispersa por vector normalmente escalan bien. Entonces, se puede explicar el anterior cambio de papeles debido a una menor eficiencia de las operaciones de reducción colectivas en MareNostrum (por ejemplo, debido a las bajas prestaciones del hardware de comunicación o por problemas de configuración).

También se presentan resultados procedentes de problemas reales, en particular los listados en la

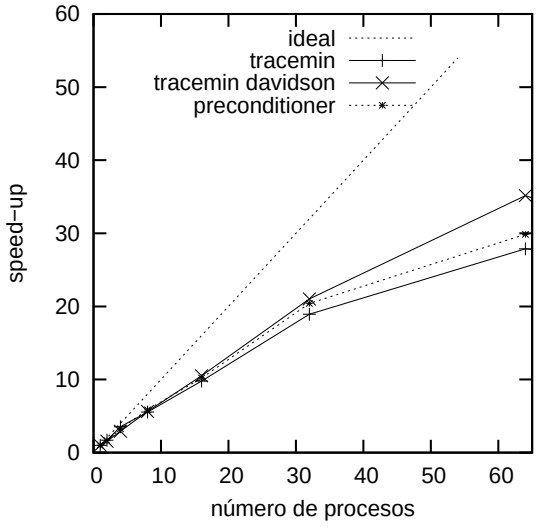


Figura 6.11: Speed-up de ambas versiones de Minimización de la Traza resolviendo el problema BS01 en MareNostrum. Se representan junto al speed-up de la aplicación del precondicionador.

batería de problemas del cuadro 6.2. El speed-up de la implementación de tipo Davidson resolviendo todos los problemas en ambas máquinas, se muestra en la figura 6.10. Además, la figura incluye el speed-up de la aplicación del precondicionador, el cual parece que está en fuerte correlación con el speed-up de la implementación. Podemos concluir que los bajos resultados de speed-up de las implementaciones se pueden atribuir que el precondicionador no escalan muy bien o el patrón de dispersión de las matrices no es muy bueno.

Finalmente, la figura 6.11 compara el speed-up de la implementación de tipo Davidson y la versión original de Minimización de la Traza con el problema más grande de la batería, mostrando un ligera ventaja del primero.

6.5. Comparación con LOBPCG

Para finalizar el estudio de prestaciones, se comparará la implementación de tipo Davidson de Minimización de la Traza aquí descrito con el método LOBPCG incluido en la librería BLOPEX. Este método está adquiriendo popularidad para resolver problemas simétricos, y está presente en las librerías más modernas, como se apuntó en la sección 4.3.

Seguidamente se adjunta una descripción del método LOBPCG, descrito en [Knyazev, 2001], así como de la implementación testeada llamada BLOPEX, descrita en [Knyazev et al., 2007].

6.5.1. Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG)

Primero formalizamos la *clase de los métodos precondicionados* de resolución de problemas de valores propios como un procedimiento iterativo de un sólo vector sobre el par de matrices (A, B) de tal forma que

$$x^{(k)} = p_k(TA, TB)x^{(0)} \ , \tag{6.1}$$

donde p_k es un polinomio de grado k de dos variables independientes (que el método selecciona bien previamente o durante el proceso), $x^{(0)}$ es una aproximación inicial y T es un precondicionador constante. También presentamos el siguiente espacio de Krylov:

$$K_k\left(TA, TB, x^{(0)}\right) = \left\{P_k(TA, TB)x^{(0)}\right\} \ , \tag{6.2}$$

donde P_k es el conjunto de todos los polinomio de grado k de dos variables independientes.

Por tanto, de las anteriores expresiones 6.1 y 6.2 cabe deducir que

$$x^{(k)} \in K_k\left(TA, TB, x^{(0)}\right) \quad . \tag{6.3}$$

Un método concreto de esta clase es el llamado *método de optimización global*, que busca el vector que minimiza el cociente generalizado de Rayleigh en el subespacio de Krylov 6.2:

$$x^{(k)} = \text{argmín}_{x \in K_k(TA,TB,x^{(0)})} \frac{x^T A x}{x^T B x} \quad . \tag{6.4}$$

El método de Gradiente Conjugado Precondicionado (PCG) es otro método de esta familia, y cumple la siguiente recurrencia de tres términos:

$$x^{(i+1)} = x^{(i)} + \alpha^{(i)} v^{(i)} + \beta^{(i)}(x^{(i)} - x^{(i-1)}), \;\; v^{(i)} = T(A - \lambda_1 B), \;\; \beta^{(0)} = 0 \;\; , \tag{6.5}$$

siendo λ_1 el valor propio más pequeño, y seleccionando $\alpha^{(i)}$ y $\beta^{(i)}$ de tal manera que minimice la $(A - \lambda_1 B)$-norma de $x^{(i+1)}$.

Knyazev describe en [Knyazev, 2001] un método similar al anterior en el que λ_1 es remplazado por una aproximación (el cociente de Rayleigh de $x^{(i)}$) y los parámetros $\alpha^{(i)}$ y $\beta^{(i)}$ se seleccionan, en su lugar, empleando el método de Rayleigh-Ritz sobre el subespacio

$$x^{(i+1)} \in \text{span}\left\{v^{(i)}, x^{(i)}, x^{(i-1)}\right\} \quad . \tag{6.6}$$

Conforme el método va convergiendo, $x^{(i)}$ y $x^{(i+1)}$ están más próximos y los sistemas proyectados que se resuelven en el procedimiento de Rayleigh-Ritz están peor condicionados. El autor propone una sencilla solución que consiste en emplear un esquema recursivo de tres términos que contenga el actual vector propio aproximado, el residuo precondicionado y un cómputo implícito de la diferencia entre el vector actual y el de la iteración anterior:

$$x^{(i+1)} = v^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} p^{(i)}, \tag{6.7}$$

$$p^{(i+1)} = v^{(i)} + \gamma^{(i)} p^{(i)}, \;\; p^{(0)} = 0, \tag{6.8}$$

$$v^{(i)} = T(A - \lambda^{(i)} B) x^{(i)}, \quad \lambda^{(i)} = ({x^{(i)}}^T A x^{(i)})/({x^{(i)}}^T B x^{(i)}) \;\; , \tag{6.9}$$

siendo seleccionados $\tau^{(i)}$ y $\gamma^{(i)}$ con la misma idea de *optimalidad local* descrita anteriormente, es decir, de tal forma que minimicen el cociente de Rayleigh de $x^{(i+1)}$. Este esquema expande el mismo subespacio que 6.6 ya que

$$x^{(i+1)} \in \text{span}\left\{v^{(i)}, x^{(i)}, p^{(i)}\right\} = \text{span}\left\{v^{(i)}, x^{(i)}, x^{(i-1)}\right\} \;\; , \tag{6.10}$$

puesto que $p^{(i+1)} = x^{(i+1)} - \tau^{(i)} x^{(i)}$.

El único método que se encuentra en la librería BLOPEX implementa justo el anterior esquema recursivo, aunque extendido a bloques y con soporte de restricciones:

Algoritmo 8 (LOBPCG)

Entrada: matrices A y B,
l restricciones linealmente independientes, $Y \in R^{n \times l}$
precondicionador T,
número de pares propios que se desean m
Salida: pares propios obtenidos

0. Elegir una matriz aleatoria $X \in R^{n \times m}$
1. Aplicamos las restricciones a X, $X \leftarrow X - Y(Y^T BY)^{-1} X^T BY$.
2. B-ortonormalizamos X.

3. Obtener vectores de Ritz iniciales mediante el procedimiento de Rayleigh-Ritz sobre el subespacio X, y almacenarlos en X.
4. I es una máscara sobre las columnas de las matrices y la iniciamos a completa.

Para $k = 1, 2, ...$ hasta convergencia

5. Obtener los residuos: $W \leftarrow AX_I - BX_I\Lambda_I$.
6. Eliminar de I los índices correspondientes a los pares convergidos. Si I está vacío, salir.
7. Calcular $W_I \leftarrow TW_I$.
8. Aplicar las restricciones a W_I: $W_I \leftarrow W_I - Y(Y^TBY)^{-1}W_I^TBY$.
9. B-ortonormalizar W_I.
10. Si $k > 1$, B-ortonormalizar P_I.
11. Calcular las matrices de Gram:

    Si $k > 1$

    $$\tilde{A} \leftarrow \begin{bmatrix} \Lambda & X^TAW_I & X^TAP_I \\ 0 & W_I^TAW_I & W_I^TAP_I \\ 0 & 0 & P_I^TAP_I \end{bmatrix}, \tilde{B} \leftarrow \begin{bmatrix} I & X^TBW_I & X^TBP_I \\ 0 & I & W_I^TBP_I \\ 0 & 0 & I \end{bmatrix}.$$

    Sino

    $$\tilde{A} \leftarrow \begin{bmatrix} \Lambda & X^TAW_I \\ 0 & W_I^TAW_I \end{bmatrix}, \tilde{B} \leftarrow \begin{bmatrix} I & X^TBW_I \\ 0 & I \end{bmatrix}.$$

12. Resolver el sistema de valores propios generalizado $\tilde{A}\tilde{Y} = \tilde{B}\tilde{Y}\tilde{\Lambda}$, donde los primeros m valores propios están ordenados de menor a mayor en la matriz $\tilde{\Lambda}$ y los vectores propios en $\tilde{Y}$ están $\tilde{B}$-ortonormalizados.
13. Obtenemos los vectores de Ritz:

    Si $k > 1$, siendo $\tilde{Y} = \begin{bmatrix} Y_X \\ Y_W \\ Y_P \end{bmatrix}$, $P_I \leftarrow W_IY_W + P_IY_P$, $X \leftarrow XY_X + P$.

    Si no, siendo $\tilde{Y} = \begin{bmatrix} Y_X \\ Y_W \end{bmatrix}$, $P_I \leftarrow W_IY_W$, $X \leftarrow XY_X + P$.

Fin para

BLOPEX usa la descomposición de Cholesky para la B-ortonormalización, seguramente la técnica más barata en cuanto a coste computacional, pero muy sensible a errores de redondeo. De hecho, al aumentar la $m = 4$ más de la mitad de los casos de prueba no pueden ser resueltos por esta librería, y en su totalidad son debidos a fallos en la B-ortonormalización.

6.5.2. Experimentos y resultados

La figura 6.12 muestra el speed-up en iteraciones de BLOPEX frente a nuestra implementación del Método de Minimización de la Traza tipo Davidson con diversas optimizaciones: desplazamientos seguro a 10^{-4}, tolerancia dinámica y cota mínima a 10^{-1}. En concreto se compara el número de iteraciones de LOBPCG con el número de iteraciones internas del Método de Minimización de la Traza, al buscar un sólo par propio. Cabe destacar que BLOPEX sólo es mejor en 3 de los 15 problemas, además de no ser capaz de resolver otros 3 problemas.

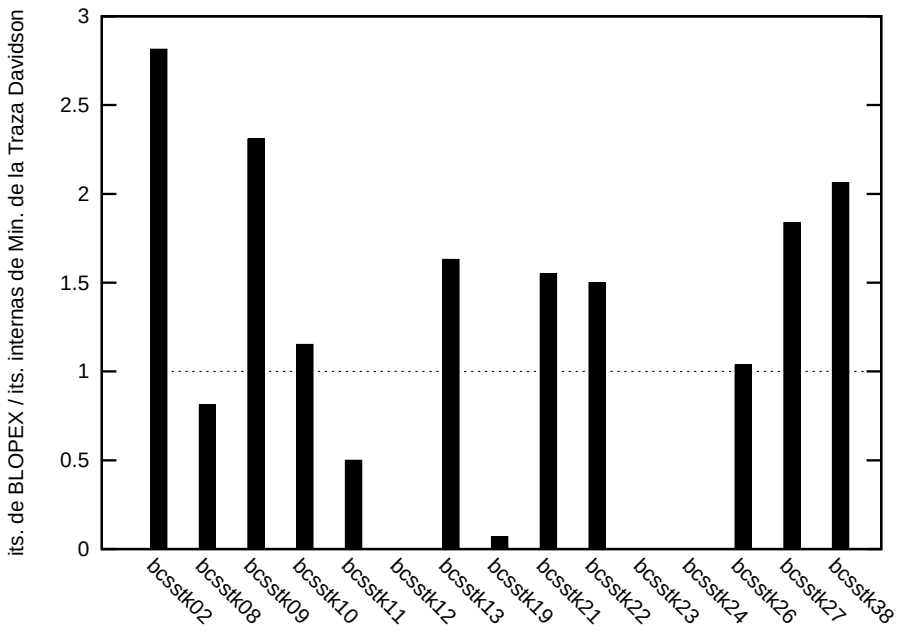


Figura 6.12: Iteraciones de BLOPEX dividido entre iteraciones internas de Minimización de la Traza tipo Davidson buscando el par propio más pequeño en los problemas de la tabla 6.1. BLOPEX no resuelve los problemas BCSST{12,23,24}.

Capítulo 7

Conclusiones y trabajo futuro

Tras una introducción al estado del arte de los métodos que resuelven problemas de valores propios, en este trabajo hemos expuesto en detalle el método de Minimización de la Traza aplicable a problemas simétricos generalizados, tanto en su versión clásica como su reescritura bajo el marco de trabajo de los métodos de tipo Davidson. Después nos hemos centrado en las alternativas de implementación y en las optimizaciones propuestas en la literatura más recientemente: estrategias de desplazamientos, estrategias de control de *breakdowns* y precondicionamiento, que aceleran la convergencia y confieren al algoritmo más robustez.

La implementación se ha desarrollado bajo los objetivos de la computación de altas prestaciones, motivo por el cual se expone una breve visión de su estado actual, y de las principales librerías que resuelven problemas de valores propios. Concretamente, nos centramos en la librería SLEPc, que incluirá en su próxima versión el *eigensolver* aquí desarrollado.

El trabajo termina con estudio de la eficacia de las optimizaciones presentadas en la exposición del método y de las prestaciones (sobre todo, en paralelo) de la implementación. También se incluye una comparativa con métodos muy populares en el estado del arte como Krylov-Schur y LOBPCG. Los resultados experimentales sobre la batería de problemas propuesta parecen prometedores. Por un lado, el *solver* es capaz incluso de resolver problemas que incumplen *ligeramente* las propiedades exigidas (la mitad de los problemas de la baterías de test tienen matrices A que no son definidas positivas o matrices B indefinidas). Por otro lado, ha resultado competitivo frente a métodos importantes como Krylov-Schur y LOBPCG.

Como trabajo futuro se plantean diferentes posibilidades. Por un lado, se podría extender el soporte de los multivectores en PETSc, y adaptar la implementación de Minimización de la Traza en la librería SLEPc para que los pudiera usar. Por otro, se podría plantear la implementación de otros métodos de Davidson, como Jacobi-Davidson, y tratar de aplicar las optimizaciones aquí expuestas. También sería muy interesante desarrollar heurísticas que eligieran el método y el precondicionador más apropiado a un problema dado. Esto simplificaría seriamente el uso de estas herramientas, y las haría más accesibles a los usuarios.

Bibliografía

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

P. Arbenz y R. Geus. A comparison of solvers for large eigenvalue problems occuring in the design of resonant cavities. *Numer. Linear Algebra Appl.*, 6(1):3–16, 1999.

P. Arbenz y W. Petersen. *Introduction to Parallel Computing (Oxford Texts in Applied and Engineering Mathematics)*. Oxford University Press, 2004. ISBN 0198515766.

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

S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith y H. Zhang. PETSc users manual. Informe Técnico ANL-95/11 - Revision 2.3.3, Argonne National Laboratory, 2007.

E. F. Beckenbach y R. Bellman. Inequalities. 1965.

A. F. Bertolini. Review of eigensolution procedures for linear dynamic finite element analysis. *Applied Mechanics Reviews*, 51(2):155–172, 1998.

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker y R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

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 y R. C. Whaley. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Software*, 28(2):135–151, 2002. ISSN 0098-3500.

M. Crouzeix, B. Philippe y M. Sadkane. The Davidson method. *SIAM J. Sci. Comput.*, 15(1):62–76, 1994. ISSN 1064-8275.

J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36(2):177–195, 1980.

J. W. Demmel, O. A. Marques, B. N. Parlett y C. Vomel. Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers. *SIAM J. Sci. Comput.*, 30(3):1508–1526, 2008.

P. J. Eberlein. Solution to the complex eigenproblem by a norm reducing jacobi type method. *Numer. Math.*, 14(3):232–245, 1970.

D. R. Fokkema, G. L. G. Sleijpen y H. A. V. der Vorst. Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1):94–125, 1999.

J. G. F. Francis. The QR Transformation A Unitary Analogue to the LR Transformation–Partes I y II. *Comput. J.*, 4(3):265–271, 1961.

G. H. Golub y H. A. van der Vorst. Eigenvalue computation in the 20th century. *J. Comput. Appl. Math.*, 123(1-2):35–65, 2000. ISSN 0377-0427.

G. H. Golub y C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, third edición, 1996.

J. L. Hennessy y D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. ISBN 1558607242.

V. E. Henson y U. M. Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics: Transactions of IMACS*, 41(1):155–177, 2002.

V. Hernandez, J. E. Roman, A. Tomas y V. Vidal. SLEPc users manual. Informe Técnico DSIC-II/24/02 - Revision 2.3.3, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2007. Disponible en http://www.grycap.upv.es/slepc.

A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 23(2):517–541, 2001.

A. V. Knyazev, M. E. Argentati, I. Lashuk y E. E. Ovtchinnikov. Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in HYPRE and PETSc. 2007.

MPI Forum. MPI: a message-passing interface standard. *Int. J. Supercomp. Applic. High Perf. Comp.*, 8(3/4):159–416, 1994.

M. H. C. Paardekooper. An eigenvalue algorithm for skew-symmetric matrices. *Numer. Math.*, 17(3):189–202, 1971.

C. C. Paige. *The computation of eigenvalues and eigenvectors of very large matrices*. Tesis Doctoral, University of London, 1971.

B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, NJ, 1980. Reissued with revisions by SIAM, Philadelphia, 1998.

B. N. Parlett y I. S. Dhillon. Relatively robust representations of symmetric tridiagonals. *Linear Algebra and Appl*, 309:121–151, 2000.

E. Romero y J. E. Roman. A parallel implementation of the trace minimization eigensolver. En *High Performance Computing for Computational Science - VECPAR 2008*, Lect. Notes Comp. Sci. 2008. (aceptado).

H. Rutishauser. Simultaneous iteration method for symmetric matrices. *Numer. Math.*, 16:205–223, 1970.

A. Sameh y Z. Tong. The trace minimization method for the symmetric generalized eigenvalue problem. *J. Comput. Appl. Math.*, 123(1-2):155–175, 2000.

A. H. Sameh. On jacobi and jacobi-like algorithms for a parallel computer. *Math. Comp.*, 25(115):579–590, 1971.

A. H. Sameh y J. A. Wisniewski. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM J. Numer. Anal.*, 19(6):1243–1259, 1982.

G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema y H. A. van der Vorst. Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36:595–633, 1996.

G. L. G. Sleijpen y D. R. Fokkema. BI-CGSTAB(L) for linear equation involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 1:11–32, 1994.

G. L. G. Sleijpen, H. A. van der Vorst y E. Meijerink. Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. *Electron. Trans. Numer. Anal.*, 7:75–89, 1998.

D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13:357–385, 1992.

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

A. van der Sluis y H. A. van der Vorst. The convergence behavior of ritz values in the presence of close eigenvalues. *Linear Algebra Appl.*, 88:651–694, 1987.

H. A. van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.