Universidad de Valladolid

Facultad de Ciencias

TRABAJO DE FIN DE GRADO

Grado en MATEMÁTICAS

MÉTODO DE LANCZOS

Autor: Alfonso Martín Mozo

Tutora: Begoña Cano Urdiales

Junio 2021

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a Begoña Cano la enorme labor de tutorización y toda la ayuda que me ha aportado en este trabajo de fin de grado. Su paciencia y dedicación a este trabajo han sido enormes, y sus numerosos consejos, correcciones, y direcciones fueron realmente valiososas y, gracias a ellos, he podido finalizar esta memoria.

En segundo lugar, también me gustaría agradecer a todos los profesores que he tenido a lo largo de mi formación, tanto universitarios como preuniversitarios, que hayan sabido despertar en mí una gran curiosidad y gusto por las matemáticas que han hecho posible que finalize mis estudios de grado.

Finalmente, me gustaría agradecer a todos los amigos, familiares y compañeros, con los que he tenido el placer de compartir mi etapa como estudiante, su apoyo incondicional a lo largo de estos años.

Valladolid, 25 de junio de 2021

Índice general

Lista de Algoritmos

Capítulo 1

Introducción

El *Álgebra Lineal Numérica*, también conocida como *Álgebra Lineal Aplicada*, es un campo de las matemáticas que se encarga del estudio de algoritmos utilizados para realizar cálculos de álgebra lineal, como pueden ser operaciones y factorizaciones matriciales, cálculos vectoriales, y aproximación de autovalores. Esta rama aprovecha las propiedades de vectores y matrices para desarrollar algoritmos de computación que minimicen el error inducido al utilizar una aritmérica de punto flotante en vez de una aritmética exacta, necesaria para resolver problemas de matemática continua.

El Álgebra Lineal Numérica es en numerosas ocasiones una parte fundamental de la Ingeniería y de las Ciencias de Computación. Algunas de sus muchísimas aplicaciones son el procesamiento y tratado de imágenes y señales, telecomunicaciones, física aplicada, ciencias de materiales o biología estructural. Es cierto que si bien es un campo relativamente pequeño comparado con otras de las grandes ramas de las matemáticas, sirve para muy diversas aplicaciones, posiblemente razón por la cual matemáticos como Nick Trefethen han llegado a argumentar que este campo es "*tan fundamental para las ciencias matemáticas como el cálculo y las ecuaciones diferenciales*".

Históricamente, fue desarollada por pioneros en el mundo de la computación como son -entre muchos otros - John von Neumann, Alan Turing, James H. Wilkinson o Alston Scott Householder, quienes intentaron utilizar los primeros ordenadores para resolver problemas continuos, como problemas balísticos, o sistemas de ecuaciones diferenciales en derivadas parciales. Este campo ha crecido de forma notable con la evolución de la tecnología: hoy en día los ordenadores son capaces de resolver sistemas complejos de matrices extremadamente grandes con gran precisión, y nuevos enfoques, como puede ser la computación paralela, permiten efectuar prácticamente procedimientos que hace unas décadas eran impensables.

El objetivo de este trabajo es exponer el método de Lanczos, un procedimiento que sirve para aproximar numéricamente los autovalores de matrices simétricas (o hermíticas en caso de tener coeficientes complejos), dispersas y de gran tamaño, y que puede ser aplicado para, entre otros, resolver sistemas lineales o de mínimos cuadrados. Para ello, se emplean resultados y conceptos básicos relacionados tanto

con el Álgebra como con el Cálculo Numérico, siendo fundamentales por ello las asignaturas de "Álgebra y Geometría Lineal I", "Cálculo Numérico", "Análisis Numérico" y "Ampliación de Análisis Numérico".

En el segundo capítulo, se deduce tanto el método de Lanczos con algunas de sus propiedades de convergencia (*Teoría de Kainel-Paige)*, así como variaciones del mismo (reortonormalización selectiva y completa) que buscan solventar algunos de los problemas que pueden surgir al implementar en aritmética finita el algoritmo. Para desarrollar el algoritmo de Lanczos se han seguido los capítulos 10 y 7 de [2] y [1] respectivamente mientras que, para presentar los esquemas de reortonormalización completa y selectiva, se han seguido los trabajos de diferentes matemáticos como Scott (ver [9]), Saad (ver [10]), Parlett (ver [9] y [8]), Simmons (ver [12]) y Paige (ver [7]).

En el tercer capítulo, se estudia cómo aplicar el algoritmo de Lanczos a la resolución de sistemas lineales cuya matriz de coeficientes es simétrica y dispersa. Posiblemente uno de los resultados más sorprendentes que se muestran en este capítulo es que resolver este tipo de sistemas mediante el método de Lanczos es equivalente a aplicar el método del gradiente conjugado. Por ello, será necesario revisar el método del gradiente conjugado, siguiendo para ello el esquema de deducción empleado en [2].

En el cuarto capítulo, se exponen los algoritmos de Golub-Kahan-Lanczos de bidiagonalización de matrices, y cómo a partir de ellos podemos desarrollar el algoritmo LSQR ("*least square QR factorization algorithm"*), un algoritmo de resolución de problemas de mínimos cuadrados. Para desarrollar este algoritmo, se seguirán algunos de los trabajos de C. Paige y A. Saunders (ver [5] y [6]).

Capítulo 2

Método de Lanczos

Supongamos que tenemos una matriz $A \in \mathbb{R}^{n\times n}$ que es simétrica, dispersa y de gran tamaño, y que necesitamos aproximar algunos de sus autovalores extremos (aquellos en ambos extremos de su espectro). El algoritmo de Lanczos busca resolver este problema, y genera para ello una secuencia de matrices tridiagonales $\{T_k\}$ con la propiedad de que los autovalores extremos de dichas matrices son estimaciones progresivamente mejores de los autovalores extremos de la matriz A.

Este algoritmo iterativo fue desarrollado en el año 1950 por Cornelius Lanczos. Inicialmente, no recibió mucha atención ya que fue concebido como un método para tridiagonalizar matrices, una tarea que puede ser resuelta de manera más simple mediante rotaciones de Givens o reflexiones de Householder. Sin embargo, tras equipar este método con una reortonormalización de los autovectores (necesaria al no poder trabajar en la práctica con aritmética exacta, sino flotante), y tras numerosas aportaciones de distintos matemáticos (entre los cuales se destaca el trabajo de Paige), este algoritmo se ha convertido en uno de los métodos más utilizados para aproximar algunos de los autovalores de matrices simétricas y dispersas (o hermíticas en caso de tener coeficientes complejos) de gran tamaño.

En este capítulo se pretende proporcionar una descripción detallada del método de Lanczos, así como de sus propiedades de convergencia, incluyendo las técnicas utilizadas para evitar la pérdida de ortogonalidad.

2.1. Deducción del método de Lanczos

El algoritmo de Lanczos puede ser deducido mediante diversos métodos. Posiblemente los dos más populares en la literatura son mediante la reducción directa a una matriz tridiagonal mediante ecuaciones de recurrencia, o mediante la optimización del cociente de Rayleigh.

Utilizaremos el segundo de los procedimientos mencionados, pues es posiblemente el que mejor ilustra las propiedades de convergencia hacia los autovalores extremos. La optimización de dicho cociente nos lleva - como ya veremos - al estudio de los

subespacios de Krylov.

Supongamos que $A \in \mathbb{R}^{n\times n}$ es la matriz simétrica, dispersa y de gran tamaño cuyos autovalores extremos queremos aproximar. Comencemos por considerar el cociente de Rayleigh de dicha matriz.

Definición 2.1 (**Cociente de Rayleigh**). *Dada una matriz simétrica, real y cuadrada $A \in \mathbb{R}^{n\times n}$ y un vector $x \in \mathbb{R}^n$ con $x \neq 0$, se define el cociente de Rayleigh de A en x como*

$$r(A, x) = \frac{x^t\, A\, x}{x^t\, x}. \tag{2.1}$$

De ahora en adelante, siempre que no haya posible confusión respecto a la matriz sobre la que se calcula el cociente de Rayleigh, denotaremos por $r(x)$ al cociente de Rayleigh de A sobre x.

El cociente de Rayleigh nos es útil a la hora de aproximar autovalores y autovectores, pues es evidente que si $x \in \mathbb{R}^n$ es un autovector aproximado de cierta matriz $A \in \mathbb{R}^{n\times n}$, entonces su cociente de Rayleigh $r(x)$ es una elección razonable de autovalor aproximado (de hecho, se tiene el autovalor exacto para autovectores exactos de A).

Sabemos que toda matriz simétrica tiene autovalores reales, es diagonalizable, y admite una base ortonormal de vectores propios. En las siguientes secciones, dada una matriz simétrica A $\in \mathbb{R}^{n\times n}$, denotaremos por $\lambda_k(A)$ a su k-ésimo autovalor de mayor tamaño; y en caso de que no haya confusión con ser el autovalor de otra posible matriz, simplemente utilizaremos λ_k. Utilizando esta notación, tenemos que

$$\lambda_n(A) \leq \cdots \leq \lambda_2(A) \leq \lambda_1(A).$$

El siguiente teorema nos será útil no solo para deducir en un primer momento el algoritmo de Lanczos, sino que será crucial para la demostración del teorema de autovalores entrelazados de Cauchy; y este último, nos ilustrará cómo evolucionan las sucesivas aproximaciones a los autovalores de la matriz A sobre la que aplicamos el algoritmo.

Teorema 2.1.1 (**Teorema Minimax de Courant-Fischer**). *Sean $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ los autovalores de una matriz simétrica $A\in \mathbb{R}^{n\times n}$, y $q_1, \ldots, q_n$ sus correspondientes autovectores unitarios. Entonces, para $j = 1, \ldots, n$ se tiene que:*

$$\lambda_j = \max_{dim(R)=j} \min_{0\neq y\in R} \frac{y^T Ay}{y^T y} = \min_{dim(S)=n-j+1} \max_{0\neq y\in S} \frac{y^T Ay}{y^T y}. \tag{2.2}$$

Nota:

- El máximo en la primera expresión de λ_j es sobre todos los subespacios j-dimensionales de $\mathbb{R}^n$, y el mínimo siguiente sobre todos los vectores no nulos de dicho subespacio.
- El mínimo en la segunda expresión de λ_j es sobre todos los subespacios (n-j+1)-dimensionales de $\mathbb{R}^n$, y el máximo siguiente sobre todos los vectores no nulos de dicho subespacio.

Demostración:

Consideremos R y S dos subespacios cualesquiera de $\mathbb{R}^n$ de dimensiones j y (n-j+1) respectivamente. Puesto que $\dim(R) + \dim(S) = j + (n - j + 1) = n + 1 > n = \dim(\mathbb{R}^n)$, y $R, S \subset \mathbb{R}^n$, se tiene que $R \cap S \neq \{0\}$ y por tanto existe $x \neq 0$ tal que $x \in R \cap S$. En particular, puesto que $x \in R$ y $x \in S$, es evidente que se tiene la siguente acotación para el cociente de Rayleigh de A sobre x

$$\min_{0\neq y\in R} \frac{y^TAy}{y^Ty} \leq \frac{x^TAx}{x^Tx} \leq \max_{0\neq y\in S} \frac{y^TAy}{y^Ty}.$$

Elijamos $\hat{R}$ un subespacio de $\mathbb{R}^n$ para maximizar la desigualdad de la izquierda, y $\hat{S}$ otro subespacio de $\mathbb{R}^n$ para minimizar la desigualdad de la derecha tal que

$$\min_{0\neq y\in \hat{R}} \frac{y^TAy}{y^Ty} = \max_{\dim(R)=j} \min_{0\neq y\in R} \frac{y^TAy}{y^Ty} \leq \frac{x^TAx}{x^Tx} \leq \min_{\dim(S)=n-j+1} \max_{0\neq y\in S} \frac{y^TAy}{y^Ty} = \max_{0\neq y\in \hat{S}} \frac{y^TAy}{y^Ty}.$$

Para ver que todas las desigualdades son en realidad igualdades, notemos que existen espacios particulares R y S de forma que la cota inferior de la desigualdad anterior coincide con la cota superior.

- Tomemos $R^j = \text{span}\{q_1, \dots q_j\}$, de forma que

$$\max_{\dim(R)=j} \min_{0\neq y\in R} \frac{y^TAy}{y^Ty} \geq \min_{0\neq y\in R^j} \frac{y^TAy}{y^Ty} = \min_{0\neq y=\sum_{i\leq j} a_iq_i} \frac{y^TAy}{y^Ty} = \min_{\text{algún } a_i\neq 0} \frac{\sum_{i\leq j} a_i^2\lambda_i}{\sum_{i\leq j} a_i^2} = \lambda_j,$$

  donde a_i son las coordenadas de y en la base $\mathbb{B}_{R^j} = \{q_1, \dots q_j\}$.

- Tomemos $S^{n-j+1} = \text{span}\{q_j, \dots q_n\}$, de forma que

$$\min_{\dim(S)=n-j+1} \max_{0\neq y\in S} \frac{y^TAy}{y^Ty} \leq \max_{0\neq y\in S^{n-j+1}} \frac{y^TAy}{y^Ty} = \max_{0\neq y=\sum_{i\geq j} b_iq_i} \frac{y^TAy}{y^Ty} = \max_{\text{algún } b_i\neq 0} \frac{\sum_{i\geq j} b_i^2\lambda_i}{\sum_{i\geq j} b_i^2} = \lambda_j,$$

  donde b_i son las coordenadas de y en la base $\mathbb{B}_{S^{n-j+1}} = \{q_j, \dots q_n\}$.

Por tanto, se tiene que

$$\lambda_j \leq \max_{\dim(R)=j} \min_{0\neq y\in R} \frac{y^TAy}{y^Ty} \leq \min_{\dim(S)=n-j+1} \max_{0\neq y\in S} \frac{y^TAy}{y^Ty} \leq \lambda_j,$$

con lo que queda probada la igualdad. □

Hemos encontrado dos formas de caracterizar a un autovalor λ_k de cierta matriz A. Generalmente, nos referimos a la **caracterización max-min** del autovalor λ_k cuando utilizamos la expresión de más a la izquierda en la igualdad (2.2), y **caracterización min-max** del autovalor λ_k cuando utilizamos la expresión de más a la derecha.

Tomando los valores $k = 1$ y $k = n$ en el teorema 2.1.1, observamos que, sobre cualquier vector $x \neq 0$, el cociente de Rayleigh para cierta matriz A está acotado por el menor y mayor autovalor de dicha matriz; y que de hecho se alcanzan dichos máximo y mínimo precisamente en los autovectores asociados a dichos autovalores.

Por tanto, sabemos que los valores máximos y mínimos de $r(x)$ son respectivamente λ_1 y λ_n. Supongamos ahora que $\{q_i\} \subseteq \mathbb{R}^n$ es una secuencia de vectores ortonormales, y definamos para cada $k = 1, \ldots, n$ la matriz ortogonal $Q_k = [q_1 \mid \ldots \mid q_k]$, con lo que se tiene que $Q_k^T A Q_k$ es una matriz de tamaño $k \times k$ a la que se pretende aplicar el resultado anterior.

Aplicando las caracterizaciones anteriores, definimos los siguientes escalares

$$M_k = \lambda_1(Q_k^T A Q_k) = \max_{y \neq 0} \frac{y^T \left(Q_k^T A Q_k\right) y}{y^T y} = \max_{\|y\|_2 1} r(Q_k y) \leq \lambda_1(A), \tag{2.3}$$

$$m_k = \lambda_k(Q_k^T A Q_k) = \min_{y \neq 0} \frac{y^T \left(Q_k^T A Q_k\right) y}{y^T y} = \min_{\|y\|_2 = 1} r(Q_k y) \geq \lambda_n(A), \tag{2.4}$$

donde en las últimas igualdades se ha utilizado que $Q_k^T Q_k = I_k$, siendo I_k la matriz identidad de tamaño k.

Puesto que

$$span\{q_1\} \subset span\{q_1, q_2\} \subset \cdots \subset span\{q_1, \ldots, q_n\} = \mathbb{R}^n,$$

se sigue que

$$M_1 \leq M_2 \leq \ldots \leq M_n = \lambda_1(A), \tag{2.5}$$

$$m_1 \geq m_2 \geq \ldots \geq m_n = \lambda_n(A). \tag{2.6}$$

Por tanto, utilizando el procedimiento anterior, tenemos garantizada la convergencia del autovalor máximo y mínimo de $Q_k^T A Q_k$ a los autovalores máximo y mínimo de A respectivamente cuando k crece. Sin embargo, nuestro objetivo es elegir los vectores q_i de forma que - además de ser ortonormales - aseguren que M_k y m_k sean buenas estimaciones de $\lambda_1(A)$ y $\lambda_n(A)$ mucho antes de que k sea igual a n.

El algoritmo de Lanczos aparece precisamente al buscar vectores q_k que hagan que $\{M_k\}$ y $\{m_k\}$ se acerquen rápidamente a λ_1 y λ_n respectivamente.

Supongamos que $u_k \in span\{q_1, \ldots q_k\}$ satisface que $M_k = r(u_k)$. Como r crece más rápidamente en la dirección del gradiente, con vistas a construir M_{k+1} tan grande como sea posible, tiene sentido elegir el siguiente vector q_{k+1} de forma que

$$\nabla r(u_k) \in span\{q_1, \ldots, q_{k+1}\}. \tag{2.7}$$

De la misma forma, si $v_k \in span\{q_1, \ldots q_k\}$, tiene sentido requerir que

$$\nabla r(v_k) \in span\{q_1, \ldots, q_{k+1}\}, \tag{2.8}$$

ya que $r(x)$ decrece de forma más rapida en la direccion de $-\nabla r(x)$.

Nuestra siguiente tarea consiste en ver cómo escoger $\{q_1, \ldots, q_n\}$ para que puedan cumplirse tanto (2.7) como (2.8). Nótese que para todo $x \in \mathbb{R}^n$ con $x \neq 0$, el gradiente del cociente de Rayleigh de A sobre x viene dado por

$$\nabla r(x) = \frac{2}{x^T x} \left(Ax - r(x)\, x\right),$$

luego

$$\nabla r(x) \in span\{x, Ax\}.$$

Puesto que los vectores u_k, $v_k \in span\{q_1, \ldots q_k\}$, las inclusiones (2.7) y (2.8) se satisfacen si

$$span\{q_1, \ldots q_k\} = span\{q_1, Aq_1, \ \ldots\ , A^{k-1}q_1\},$$

y se elige q_{k+1} de forma que

$$span\{q_1, \ldots q_{k+1}\} = span\{q_1, Aq_1, \ \ldots\ , A^{k-1}q_1, A^k q_1\}.$$

El problema presente consiste en encontrar una base ortonormal para el *subespacio de Krylov*

$$\mathcal{K}(A, q_1, k) = span(q_1, Aq_1, \ \ldots\ , A^{k-1}q_1). \tag{2.9}$$

Estos subespacios son precisamente los espacios columna de las matrices de Krylov

$$K(A, q_1, k) = \left[q_1 |\ Aq_1 \mid A^2 q_1 \mid \ \ldots \mid A^{k-1}q_1\right], \qquad A \in \mathbb{R}^{n\times n}, q_1 \in \mathbb{R}^n. \tag{2.10}$$

Notemos que exise cierta similitud con el método de la potencia para el mismo iterante inicial q_1. El método de la potencia busca el siguiente iterante en la dirección de $A^{k-1}q_1$, un subespacio contenido en el subespacio de Krylov $\mathcal{K}(A, q_1, k)$. En este sentido, suele decirse que el algoritmo de Lanczos utiliza la ventaja de "experiencias anteriores". En cada k-ésima iteración, el método de la potencia busca el autovector en la dirección determinada por $A^k q_1$, sin tener en cuenta ya las direcciones q_1, $Aq_1, \ldots, A^{k-1}q_1$, como hace el método de Lanczos. Posterior a la deducción de este método se detallará una comparación entre ambos métodos.

2.1.1. Tridiagonalización

Según hemos visto, nuestro objetivo de cara a obtener buenas aproximaciones M_k y m_k de $\lambda_1(A)$ y $\lambda_n(A)$ respectivamente, es obtener una base ortonormal de cierto subespacio de Krylov. En esta sección se pretende demostrar la relación entre la tridiagonalización de A y la factorización QR de $K(A, q_1, k)$. Debemos empezar por recordar dos definiciones básicas: la factorización QR de una matriz cuadrada, y la definición de matriz tridiagonal.

Definición 2.2 (**Factorización QR**). *Dada una matriz real cuadrada $A \in \mathbb{R}^{n\times n}$, se define su descomposición QR o factorización QR como $A = QR$ donde la matriz $Q \in \mathbb{R}^{n\times n}$ es ortogonal (es decir: $Q^T Q = I$), y $R \in \mathbb{R}^{n\times n}$ es una matriz triangular superior.*

Recordemos ahora lo que entendemos por una matriz tridiagonal real.

Definición 2.3 (**Matriz Tridiagonal real**). *Se dice que una matriz $T \in \mathbb{R}^{n\times n}$ es una matriz tridiagonal real de tamaño n si sus elementos son solo distintos de cero en la diagonal principal y las diagonales adyacentes por debajo y por encima de esta.*

Evidentemente, se define de forma análoga lo que se entiende por una matriz tridiagonal compleja, pero en nuestro caso, únicamente utilizaremos matrices tridiagonales con coeficientes reales.

Existe una relación muy estrecha entre la descomposición tridiagonal de una matriz simétrica y la factorización QR de la matriz $K(A, q_1, n)$. Supongamos que $Q^TAQ = T$ es una descomposición tridiagonal de $A \in \mathbb{R}^{n\times n}$. Consideremos ahora el producto de Q^T por la matriz de Krylov de A en dirección de $q_1 = Q(:, 1)$ de tamaño n, esto es, $K(A, q_1, n)$,

$$\begin{aligned} Q^TK(A, q_1, n) &= Q^T \cdot \left[q_1 \mid Aq_1 \mid \ldots \mid A^{n-1}q_1\right] \\ &= \left[Q^Tq_1 \mid (Q^TAQ)(Q^Tq_1) \mid \ldots \mid (Q^TAQ)^{n-1}(Q^Tq_1)\right] \\ &= \left[e_1 \mid Te_1 \mid \ldots \mid T^{n-1}e_1\right] = R, \end{aligned}$$

donde e_1 es el vector canónico de tamaño $k \times 1$, con un 1 en su primera componente.

Observamos que el producto anterior nos da una matriz triangular superior. Tenemos entonces que $Q^TK(A, q_1, n) = R$, y puesto que Q es una matriz ortogonal, despejando $K(A, q_1, n)$, observamos que $K(A, q_1, n) = QR$.

Habíamos deducido que, de cara a obtener buenas aproximaciones M_k y m_k de $\lambda_1(A)$ y $\lambda_n(A)$, no teníamos más que encontrar una base ortonormal para el subespacio de Krylov $\mathcal{K}(A, q_1, k)$. Vemos ahora que, para generar los vectores q_i que forman la base ortonormal de $\mathcal{K}(A, q_1, k)$, no tenemos más que tridiagonalizar la matriz A con transformaciones ortogonales.

Existen numerosos métodos de tridiagonalización de matrices. Posiblemente uno de los más populares es la tridiagonalización de Householder. Sin embargo, si A es una matriz simétrica de gran tamaño y dispersa, este procedimiento no es conveniente debido a que las transformaciones aplicadas sobre A tienden a romper la dispersión, dando lugar a matrices no solo de gran tamaño sino poco dispersas, aumentando notablemente el número de operaciones a realizar y tiempo de cómputo del algoritmo. En general, cualquier método que compute una matriz tridiagonal actualizando dicha matriz suele romper la dispersión, por lo que no es útil para el caso que nos ocupa.

Es cierto que se pueden utilizar otros métodos de tridiagonalización - como pueden ser las transformaciones de Givens - para evitar la destrucción de la dispersión de la matriz en cuestión; sin embargo, es mucho más rápido intentar calcular los elementos de la matriz tridiagonal directamente.

Supongamos que $T = Q^T AQ$, donde $Q = [q_1 \mid \ldots \mid q_n]$ y

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \ldots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \ldots & & \beta_{n-1} & \alpha_n \end{bmatrix}. \tag{2.11}$$

Igualando las columnas de $AQ = QT$ se deduce que

$$Aq_k = \beta_{k-1}q_{k-1} + \alpha_k q_k + \beta_k q_{k+1}, \qquad \beta_0 q_0 \equiv 0, \qquad \text{para k=1,} \ldots, \text{n-1.} \tag{2.12}$$

Multiplicando la primera expresión de (2.12) por q_k^T, y teniendo en cuenta la ortonormalidad de los vectores q_k, tenemos que $\alpha_k = q_k^T A q_k$ (otra forma de verlo es simplemente tener en cuenta que $T_{ij} = q_i^T A q_j$, luego las entradas de la diagonal son $T_{kk} = q_k^T A q_k$).

Por otro lado, en (2.12) obtenemos una expresión para calcular los sucesivos q_k. Definimos el siguiente vector (conocido por *vector de residuos*):

$$r_k = \beta_k q_{k+1} = (A - \alpha_k I)\, q_k - \beta_{k-1} q_{k-1}, \tag{2.13}$$

y en caso de que $r_k \neq 0$, encontramos nuestro siguiente vector de Lanczos sin más que dividir por su norma:

$$q_{k+1} = r_k/\beta_k \qquad \text{donde} \qquad \beta_k = \|r_k\|_2 .$$

Evidentemente, no hay pérdida de generalidad en haber elegido los β_k positivos. Por ello, podíamos haber tomado $\beta_k = -\|r_k\|_2$.

Si encontramos un $r_k = 0$, la iteración termina; pero como veremos en el teorema 2.2.1, no sin llegar a un subespacio invariante. En la práctica, rara vez encontramos un $\beta_k = 0$, pero igualmente se producen buenas aproximaciones incluso antes de encontrar un β_k pequeño. Lo que se suele hacer es pedir un número m de máximas iteraciones, de forma que si no se llega en m iteraciones a un subespacio invariante, finalizamos el algoritmo y habremos aproximado m autovalores de la matriz dada. A partir de los razonamientos anteriores, podemos definir la primera versión del algoritmo de tridiagonalización de Lanczos. Viene descrito en el en algoritmo 1.

2.2. Finalización del algoritmo

En la sección anterior hemos descrito una primera aproximación al algoritmo de Lanczos. Hemos mencionado que en la práctica solemos pedir un número máximo de iteraciones para que se ejecute el algoritmo. Sin embargo, en ocasiones el algoritmo puede llegar a terminar de forma prematura. El siguiente teorema nos muestra cuándo termina el teorema de forma autónoma (es decir; sin llegar al número máximo de iteraciones que hayamos seleccionado).

Algoritmo 1: Algoritmo de Lanczos

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$, un vector aleatorio $q \in \mathbb{R}^n$, y m el número de iteraciones

Salida : Matriz tridiagonal $T_k \in \mathbb{R}^{k\times k}$, y $Q_k = [q_1 \mid \ldots \mid q_n] \in \mathbb{R}^{n\times k}$ tal que $AQ_k = Q_k T_k$

$q_1 = q/\|q\|$, $\beta_0 = 1$, $q_0 = 0$.
for $k = 1$ **to** m **do**
  $\alpha_k = q_k^T A q_k$;
  $r_k = Aq_k - \alpha_k q_k - \beta_{k-1} q_{k-1}$
  $\beta_k = \|r_k\|$;
  if $\beta_k \neq 0$ **then**
    $q_{k+1} = r_k/\beta_k$
  end
end

Teorema 2.2.1. *La iteración de Lacnzos -descrita en el algoritmo 1 - continúa hasta que k=m, donde*

$$m = rank(K(A, q_1, n)).$$

Es más, para $k = 1, \ldots m$ *se tiene que*

$$AQ_k = Q_k T_k + r_k e_k^T \tag{2.14}$$

donde

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \ldots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{k-1} \\ 0 & \ldots & & \beta_{k-1} & \alpha_k \end{bmatrix},$$

e_k *es el vector canónico de tamaño* $k \times 1$ *con el 1 en su última componente, y* $Q = [q_1 \mid \ldots \mid q_k]$ *tiene columas ortonormales que generan* $\mathcal{K}(A, q_1, k)$.

Demostración:

Razonemos por inducción sobre k. Para k=1, el teorema es evidente por la expresión de r_1 en el algoritmo. Supongamos que, para cierto $k \geq 1$, el algoritmo 1 ha generado $Q = [q_1 \mid \ldots \mid q_k]$ con columnas ortogonales y de forma que

$$ran(Q_k) = span\{q_1, \ldots, q_k\} = \mathcal{K}(A, q_1, k).$$

De (2.13) se obtiene inmediatamente (2.14), que implica que, por la ortogonalidad de los vectores q_i, se tiene que

$$Q_k^T A Q_k = T_k + Q_k^T r_k e_k^T. \tag{2.15}$$

Por tanto, utilizando el algoritmo 1, hemos construido para $i = 1, \ldots k$

$$\alpha_i = q_i^T A q_i,$$

y para $i = 1, \dots k-1$, por la ortogonalidad de las columas de Q_k, se concluye que

$$q_{i+1}^T A q_i = q_{i+1}^T \left(A q_i - \alpha_i q_i - \beta_{i-1} q_{i-1} \right). \tag{2.16}$$

Ahora bien, el término entre paréntesis de (2.16) es $\beta_i q_{i+1}$ por cómo se ha definido q_{i+1}, y puesto que q_{i+1} tiene norma 1 por construcción, tenemos que

$$q_{i+1}^T A q_i = q_{i+1}^T \left(A q_i - \alpha_i q_i - \beta_{i-1} q_{i-1} \right) = q_{i+1}^T \left(\beta_i q_{i+1} \right) = \beta_i.$$

Por el mismo razonamiento, si $i + 2 \leq j \leq k$, entonces

$$q_j^T A Q_i = 0,$$

de donde se sigue que $Q_k^T A Q_k = T_k$, y de (2.15) que $Q_k^T r_k = 0$.

Si $r_k \neq 0$, $q_{k+1} = r_k / \left\| r_k \right\|_2$ es ortogonal a $q_1, \dots, q_k$. Luego se deduce que $q_{k+1} \notin \mathcal{K}(A, q_1, k)$ (ya que $q_1, \dots, q_k$ generan $\mathcal{K}(A, q_1, k)$) y que

$$q_{k+1} \in span\{A q_k, q_k, q_{k-1}\} \subseteq \mathcal{K}(A, q_1, k+1).$$

Entonces, $Q_{k+1}^T Q_{k+1} = I_{k+1}$ y

$$ran(Q_{k+1}) = \mathcal{K}(A, q_1, k+1).$$

Por otro lado, si $r_k = 0$, tenemos que $A Q_k = Q_k T_k$. Esto nos dice que $ran(Q_k) = \mathcal{K}(A, q_1, k)$ es un espacio invariante para A, luego

$$k = m = rank(K(A, q_1, k+1)) = rank(K(A, q_1, n)),$$

como queríamos demostrar. □

Se pueden aproximar los autovalores de la matriz sobre la que se aplica el algoritmo de Lanczos por los autovalores de las sucesivas matrices T_k. Durante la k-ésima iteración estamos realizando el cálculo de α_{k+1} y β_{k+1}, de forma que la matriz T_k no es más que una submatriz de T_{k+1}.

De cara a estudiar cómo se produce la convergencia de los autovalores mediante este método, es muy interesante la conclusión que se puede obtener del teorema de Cauchy de autovalores entrelazados.

Teorema 2.2.2 (**Teorema de Cauchy de autovalores entrelazados**). *Sea* $A = \begin{pmatrix} H & \boldsymbol{b} \\ \boldsymbol{b}^T & u \end{pmatrix}$ *una matriz simétrica de tamaño* n*, con* H *otra matriz de tamaño* $n-1$*. Sean* $\alpha_n \leq \cdots \leq \alpha_1$ *los autovalores de* A*, y* $\beta_{n-1} \leq \cdots \leq \beta_1$ *los autovalores de* H*. Entonces, los autovalores de* A *entrelazan a los autovalores de* H*, es decir,*

$$\alpha_n \leq \beta_{n-1} \leq \cdots \leq \beta_i \leq \alpha_i \leq \beta_{i-1} \leq \alpha_{i-1} \leq \cdots \leq \beta_1 \leq \alpha_1.$$

Demostración:
Para la demostración de este teorema utilizaremos el teorema minimax de Courant-Fischer.

Sea I_{n-1} la matriz identidad de tamaño $(n-1)$, y consideremos la matriz de tamaño $n \times (n-1)$ definida según $P = \left[\begin{smallmatrix} I_{n-1} \\ \mathbf{O} \end{smallmatrix} \right]$. Es evidente que

$$H = P^T A P. \tag{2.17}$$

Además, puesto que $P^T P = I_{n-1}$, se cumple que, para cualquier vector $\mathbf{x} \in \mathbb{R}^{n-1}$, se tiene que

$$\mathbf{x}^t \mathbf{x} = (P\mathbf{x})^T (P\mathbf{x}). \tag{2.18}$$

Por otro lado, si $\mathbf{x}_1, \dots \mathbf{x}_n \in \mathbb{R}^{n-1}$ son un conjunto de vectores linealmente independientes, $P\mathbf{x}_1, \dots P\mathbf{x}_n \in \mathbb{R}^n$ también son vectores linealmente independientes. Luego si S es un subespacio k-dimensional de $\mathbb{R}^{n-1}$, PS es un subespacio k-dimensional de $\mathbb{R}^n$. Ahora bien, utilizando la caracterización min-max para el autovalor β_i, y utilizando (2.17) y (2.18), tenemos que

$$\beta_i = \min_{\mathrm{S}^{n-i}} \max_{0 \neq \mathrm{y} \in \mathrm{S}^{n-i}} \frac{y^T H y}{y^T y} = \min_{\mathrm{S}^{n-i}} \max_{0 \neq \mathrm{y} \in \mathrm{S}^{n-i}} \frac{(Py)^T A (Py)}{(Py)^T (Py)},$$

y, teniendo en cuenta en la expresión de la derecha que el mínimo de un subconjunto es mayor o igual que el mínimo del conjunto entero, y reconociendo la caracterización min-max de α_{i+1}, concluimos que

$$\begin{aligned} \beta_i = \min_{\mathrm{S}^{n-i} \subseteq \mathbb{R}^{n-1}} \max_{0 \neq \mathrm{y} \in \mathrm{S}^{n-i}} \frac{y^T H y}{y^T y} &= \min_{\mathrm{S}^{n-i} \subseteq \mathbb{R}^{n-1}} \max_{0 \neq \mathrm{y} \in \mathrm{S}^{n-i}} \frac{(Py)^T A (Py)}{(Py)^T (Py)} \\ &\geq \min_{\mathrm{S}^{n-i} \subseteq \mathbb{R}^{n}} \max_{0 \neq \mathrm{z} \in \mathrm{S}^{n-i}} \frac{z^T A z}{z^T z} = \alpha_{i+1}. \end{aligned} \tag{2.19}$$

Por otro lado, utilizando la caracterización max-min de β_i, y teniendo en cuenta que el máximo de un subconjunto es menor o igual que el máximo del conjunto entero, llegamos a que

$$\begin{aligned} \beta_i = \max_{\mathrm{S}^{i} \subseteq \mathbb{R}^{n-1}} \min_{0 \neq \mathrm{y} \in \mathrm{S}^{i}} \frac{y^T H y}{y^T y} &= \max_{\mathrm{S}^{i} \subseteq \mathbb{R}^{n-1}} \min_{0 \neq \mathrm{y} \in \mathrm{S}^{i}} \frac{(Py)^T A (Py)}{(Py)^T (Py)} \\ &\leq \max_{\mathrm{S}^{i} \subseteq \mathbb{R}^{n}} \min_{0 \neq \mathrm{z} \in \mathrm{S}^{i}} \frac{z^T A z}{z^T z} = \alpha_{i}. \end{aligned} \tag{2.20}$$

Finalmente, combinando las desigualdades encontradas en (2.19) y (2.20,) tenemos que

$$\alpha_{i+1} \leq \beta_i \leq \alpha_i$$

con lo que queda demostrado el teorema. □

Ahora bien, consideremos el mayor autovalor de cada matriz T_k. Puesto que T_k se puede ver como una submatriz de T_{k+1} tal que

$$T_{k+1} = \left(\begin{array}{ccc:c} & & & 0 \\ & T_k & & \vdots \\ & & & 0 \\ & & & \beta_k \\ \hdashline 0 & \dots & 0 & \beta_k & \alpha_{k+1} \end{array}\right),$$

tomando $H = T_k$, $b^T = (0, \dots, 0, \beta_k)$ y $u = \alpha_{k+1}$ en el teorema de los autovalores entrelazados de Cauchy, se cumple que $\lambda_1(T_{k+1}) \geq \lambda_1(T_k)$. Deducimos, por tanto, que en cada iteración el mayor autovalor crece de forma monótona con k, como ya sabíamos por (2.5). El análogo ocurre para los autovalores de menor módulo, los autovalores aproximados decrecen de forma monótona con k, como ya sabíamos por (2.6). De hecho, el teorema de Cauchy nos dice algo más: los autovalores de la matriz T_k entrelazan a los autovalores de T_{k+1}, es decir;

$$\lambda_i(T_{k+1}) \geq \lambda_i(T_k) \geq \lambda_{i+1}(T_{k+1}) \geq \lambda_{i+1}(T_k).$$

De aquí se deduce que λ_i converge de forma monótona con k para cualquier i fijo, no solo para $i = 1$.

La siguiente pregunta a responder es hacia qué autovalor de A converge $\lambda_i(T_k)$ cuando k crece. Es evidente que, por construcción del método, $\lambda_1(T_k)$ debe converger hacia el mayor autovalor de A, es decir; $\lambda_1(A)$. De hecho, en caso de que el algoritmo se ejecute hasta $k = n$ sin encontrar ningún $\beta_k = 0$, se tiene que T_n es una matriz similar a A, y por tanto, $\lambda_1(T_n) = \lambda_1(A)$. De manera similar, el i-ésimo mayor autovalor $\lambda_i(T_k)$ debe crecer monótonamente y converger hacia el i-ésimo mayor autovalor $\lambda_i(A)$ de A; y el i-ésimo menor autovalor $\lambda_{k+1-i}(T_k)$ debe decrecer monótonamente y converger hacia el i-ésimo menor autovalor $\lambda_{n+1-i}(A)$ de A.

2.2.1. Ejemplos

En el siguiente par de ejemplos se pretende ilustrar cómo se produce la convergencia de los autovalores cuando crece el número de iteraciones. Para ello, vamos a tomar una matriz cuyos autovalores exactos conozcamos (de forma que podamos comparar con nuestras aproximaciones numéricas), y aplicaremos el algoritmo de Lanczos para dos vectores iniciales q_1 diferentes.

Consideremos A una matriz diagonal de tamaño 1000 con entradas (autovalores) tomados de una distribución $N(\mu = 0, \sigma^2 = 1)$, que han sido reordenadas posteriormente, de forma que $a_{1,1} \geq a_{2,2} \geq \cdots \geq a_{1000,1000}$ como matriz sobre la que aplicar el algoritmo 1. En la figura 2.1 se han representado los autovalores exactos de dicha matriz.

Ejemplo 2.1. *Tomamos como vector inicial $q_1 = [1, \dots, 1]^T$, es decir, el vector columna de 1000 entradas donde todas ellas son 1. No debería sorprendernos que, con esta elección particular de q_1, el primer autovalor aproximado sea cercano a cero,*

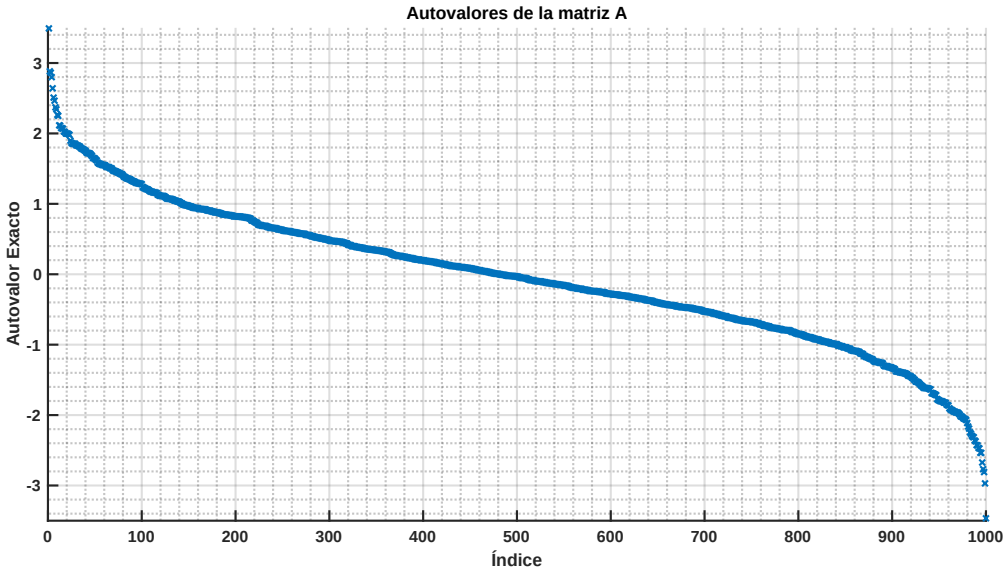

Figura 2.1: Autovalores exactos de la matriz A

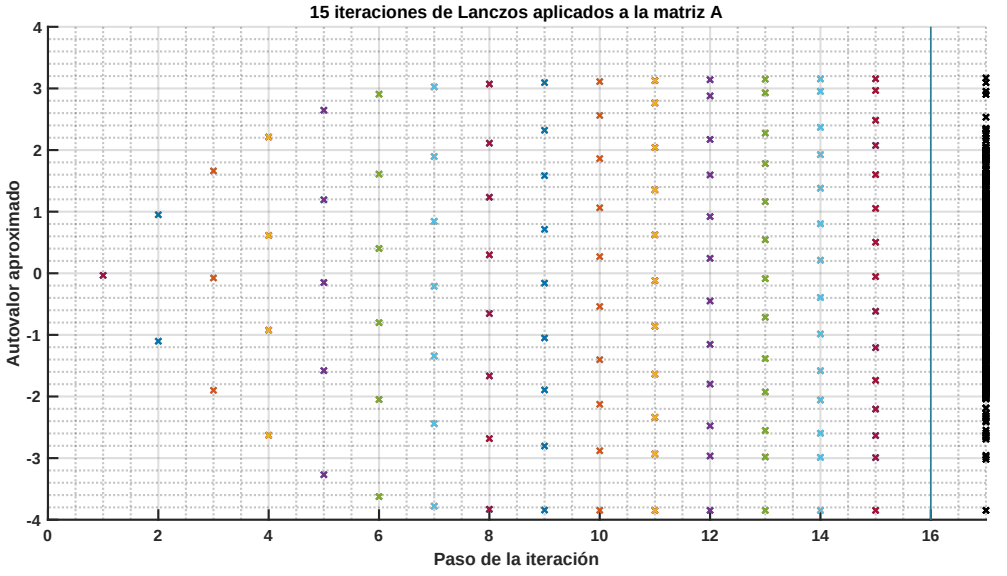

Figura 2.2: Autovalores aproximados por cada iteración de Lanczos frente a autovalores exactos para el vector inicial $q_1 = [1, 1, \ldots, 1]^T$

pues en la primera iteración calculamos $\alpha_1 = q_1^T A q_1$*, es decir, estamos sumando las entradas de la diagonal (tomadas de una distribución de media* $\mu = 0$*). Las sucesivas aproximaciones a los autovalores han sido representadas en la figura 2.2.*

Según hemos visto, podemos elegir como vector q_1 un vector aleatorio. En el siguiente ejemplo estudiaremos la aproximación de autovalores para la misma matriz A (cuyos autovalores han sido representados en la figura 2.1) para otra elección del vector inicial.

Ejemplo 2.2. *Tomamos ahora como vector inicial el vector* $q_2 = [1, 2, \ldots, 1000]^T$ *y A la matriz descrita anteriormente. Las sucesivas aproximaciones a los autovalores*

han sido representadas en la figura 2.3.

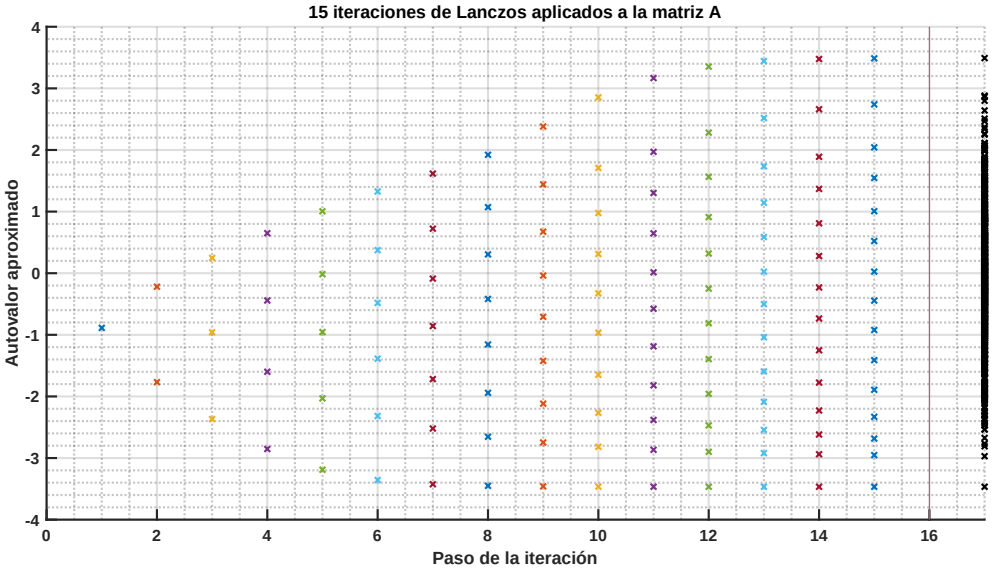

Figura 2.3: Autovalores aproximados por cada iteración de Lanczos frente a autovalores exactos para el vector inicial $q_2 = [1, 2, \ldots, 1000]^T$

En las figuras 2.2 y 2.3 podemos apreciar el comportamiento de la convergencia del método. En ambos casos, podemos observar cómo los autovalores de una iteración entrelazan los autovalores de la iteración anterior.

Notamos que, como hemos deducido de forma teórica, la convergencia es monótona. Es interesante remarcar que $\lambda_1(T_k) \leq \lambda_1(A)$, como ya sabíamos por (2.3), es decir, las sucesivas aproximaciones al autovalor máximo de la matriz A son menores o iguales que el mismo. Por otra parte, se tiene que en todo momento $\lambda_k(T_k) \geq \lambda_n(A)$, como ya sabíamos por (2.4), por lo que las sucesivas aproximaciones al autovalor mínimo son siempre mayores que el mismo.

2.3. Teoría de la convergencia

Como ya se ha visto en el teorema 2.2.1, encontrar un $\beta_k = 0$ es un buen evento en el sentido de que se ha calculado un espacio invariante exacto. En la práctica, rara vez ocurre esto, y sin embargo, los autovalores extremos de las sucesivas matrices T_k son aproximaciones sorprendentemente buenas de los autovalores extremos de A.

En esta sección se pretende estudiar la calidad de los autovalores aproximados, así como la convergencia hacia los autovalores exactos de la matriz A en función del número de iteraciones k.

En los siguientes teoremas se muestra la calidad de los autovalores aproximados mediante el algoritmo 1.

Teorema 2.3.1. *Supongamos que se han aplicado k iteraciones de Lanczos descritas en el algoritmo 1, y que* $S_k^T T_k S_k = diag(\theta_1, \ \dots \ , \theta_k)$ *es la descomposición de Schur de la matriz tridiagonal* T_k*. Si* $Y_k = [y_1, \dots, y_k] = Q_k S_k \in \mathbb{R}^{n\times k}$*, entonces para* $i = 1, \dots, k$ *se tiene que*

$$\|Ay_i - \theta_i y_i\|_2 = |\beta_k||s_{ki}|, \qquad \text{donde } S_k = (s_{pq}). \tag{2.21}$$

Demostración:

Supongamos que se han realizado k iteraciones de Lanczos. Entonces, sabemos por el teorema 2.2.1 que se tiene (2.14). Postmultiplicando dicha expresión por S_k, y teniendo en cuenta que $Q_k S_k = Y_k$, se concluye que

$$Q_k T_k S_k = Q_k S_k \text{diag}(\theta_1, \ \dots \ , \theta_k) = Y_k \, \text{diag}(\theta_1, \ \dots \ , \theta_k),$$

luego deducimos que

$$AY_k = Y_k \text{diag}(\theta_1, \ \dots \ , \theta_k) + r_k e_k^T S_k.$$

Por tanto, $Ay_i = \theta_i y_i + r_k \left(e_k^T S_k e_i\right)$, de donde se sigue

$$Ay_i - \theta_i y_i = r_k \left(e_k^T S_k e_i\right), \tag{2.22}$$

y tomando normas, como $\|r_k\|_2 = |\beta_k|$, se deduce la expresión (2.21). □

La expresión (2.22) nos da una idea de la calidad del autovalor aproximado, pues esperamos obtener una buena aproximación en caso de que la norma del vector de residuos r_k sea pequeña. En este sentido, se tiene el siguiente resultado.

Lema 2.3.2. *Sea A una matriz simétrica, y supongamos que*

$$Ax - \theta x = r \qquad \text{con } x \neq 0. \tag{2.23}$$

Entonces

$$\min_{\mu\in\lambda(A)} \quad |\mu - \theta| \leq \|r\|_2 \, / \, \|x\|_2. \tag{2.24}$$

Demostración:

Consideremos $Z^T A Z = \text{diag}(\lambda_1, \dots, \lambda_n)$ la descomposición de Schur de la matriz A. Sin más que sustituir $A = Z\text{diag}(\lambda_1, \dots, \lambda_n)Z^T$ en (2.23), tenemos que

$$Z \left(\text{diag}(\lambda_1, \dots, \lambda_n) - \theta I_n\right) Z^T x = r,$$

de donde se sigue que

$$\left(\text{diag}(\lambda_1, \dots, \lambda_n) - \theta I_n\right) \left(Z^T x\right) = Z^T r,$$

donde I_n denota a la matriz identidad de tamaño n. Notemos que $\text{diag}(\lambda_1, \dots, \lambda_n) - \theta I_n$ es una matriz diagonal, luego

$$\begin{aligned} \|r\|_2 = \left\|Z^T r\right\|_2 &= \left\|\left(\text{diag}(\lambda_1, \dots, \lambda_n) - \theta I_n\right) \left(Z^T x\right)\right\|_2 \\ &\geq \min_{\mu\in\lambda(A)} \quad |\mu - \theta| \left\|Z^T x\right\|_2 = \min_{\mu\in\lambda(A)} \quad |\mu - \theta| \, \|x\|_2, \end{aligned}$$

concluyendo con (2.24). □

Si aplicamos el lema anterior a la expresión (2.22), obtenemos una cota para las aproximaciones a los autovalores de la matriz A obtenidos a partir del método de Lanczos. Este resultado se recoge en el siguiente corolario.

Corolario 2.1. *Supongamos que se han aplicado k iteraciones del método de Lanczos. Si $\theta_i = \lambda_i(T_k)$, entonces*

$$\min_{\mu\in\lambda(A)} |\theta_i - \mu| \leq |\beta_k||s_{ki}|/\|Q_k S_k e_i\| \qquad \text{para } i = 1, \ldots, k, \tag{2.25}$$

donde $\lambda(A)$ representa el espectro de la matriz A.

En ausencia de errores de redondeo, $\|y\|_2 = \|Q_k S_k e_i\|_2 = 1$, por lo que sin más que sustituir en (2.25), tenemos que existe un autovalor μ de A tal que

$$|\mu - \theta_i| \leq |\beta_k||s_{ki}|.$$

Sin embargo, al implementar numéricamente este algoritmo, la pérdida de ortogonalidad de los vectores q_i puede destruir la cota anterior. Esta pérdida de ortogonalidad motiva, como ya veremos, el desarrollo de diversas técnicas de reortogonalización.

Los siguientes teoremas estudian cómo de bien los autovalores de T_k aproximan los autovalores de A en función del número de iteraciones k. Estos resultados se enmarcan en lo que se conoce por *Teoría de convergencia de Kaniel-Paige.*

Teorema 2.3.3. *Sea A una matriz $n \times n$ real, y consideremos su descomposición de Schur*

$$Z^T A Z = diag(\lambda_1, \ \ldots \ , \lambda_n), \qquad \lambda_1 \geq \cdots \geq \lambda_n, \qquad Z = [z_1 \mid \ \ldots \ \mid z_n] \, .$$

Supongamos que se han realizado k iteraciones de Lanczos (ver algoritmo 1), y que T_k es la matriz tridiagonal (3.6). Si $\theta_1 = \lambda_1(T_k)$, entonces

$$\lambda_1 \ \geq \ \theta_1 \ \geq \ \lambda_1 - (\lambda_1 - \lambda_n)\left(\frac{\tan(\phi_1)}{c_{k-1}(1+2\rho_1)}\right)^2, \tag{2.26}$$

donde $\cos(\phi_1) = \left|q_1^T z_1\right|$,

$$\rho_1 = \frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n}, \tag{2.27}$$

y $c_{k-1}(x)$ es el polinomio de Chebyshev de grado $k-1$.

Demostración:

Supongamos que el algoritmo de Lanczos ha realizado k iteraciones, y ha generado por tanto las matrices T_k y Q_k tales que $Q_k^T A Q_k = T_k$. Si aplicamos la caracterización min-max dada por el teorema 2.1.1 a $\theta_1(T_k)$, tenemos que

$$\theta_1 = \max_{0\neq y\in S} \frac{y^T T_k y}{y^T y} = \max_{0\neq y\in S} \frac{(Q_k y)^T A (Q_k y)}{(Q_k y)^T (Q_k y)} \quad = \max_{0\neq w\in \mathcal{K}(A,q_1,k)} \frac{w^T A w}{w^T w}.$$

Puesto que sabemos que el mayor autovalor de la matriz A viene dado por el máximo de $w^T Aw/w^T w$ sobre todos los vectores no nulos $w \in \mathbb{R}^n$, es evidente que $\theta_1 \leq \lambda_1$.

Para obtener una cota inferior para θ_1 notemos que, como $\{q_1, Aq_1, \ldots, A^{k-1}q_1\}$ genera $\mathcal{K}(A, q_1, k)$, se tiene que[1]

$$\theta_1 = \max_{0 \neq w \in \mathcal{K}(A,q_1,k)} \frac{w^T Aw}{w^T w} = \max_{p \in \mathcal{P}_{k-1}} \frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1}, \tag{2.28}$$

donde $\mathcal{P}_{k-1}$ es el conjunto de todos los polinomios de grado menor o igual que $k-1$, y $p(x)$ es un polinomio en dicho conjunto.

Considerando la expresión de q_1 en la base $\{z_1, \ldots, z_n\}$, es decir,

$$q_1 = d_1 z_1 + \cdots + d_n z_n, \tag{2.29}$$

donde $d_i = q_1^T z_i$, se sigue que

$$\frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1} = \frac{\sum_{i=1}^n d_i^2 p(\lambda_i)^2 \lambda_i}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2} \geq \frac{\lambda_1 d_1^2 p(\lambda_1)^2 + \lambda_n \delta^2}{d_1^2 p(\lambda_1)^2 + \delta^2} = \lambda_1 - \frac{(\lambda_1 - \lambda_n)\,\delta^2}{d_1^2 p(\lambda_1)^2 + \delta^2}, \tag{2.30}$$

donde se ha definido

$$\delta^2 = \sum_{i=2}^n d_i^2 p(\lambda_i)^2. \tag{2.31}$$

En vistas de la expresión (2.30), deducimos que si el polinomio $p(x)$ tiene la propiedad de ser grande en $x = \lambda_1$ comparado con sus valores en $\lambda_2, \ldots, \lambda_n$, entonces obtenemos una buena cota inferior para θ_1. Una forma de hacer esto es considerar el polinomio

$$p(x) = c_{k-1}\left(-1 + 2\frac{x - \lambda_n}{\lambda_2 - \lambda_n}\right),$$

donde $c_{k-1}(z)$ es el (k-1)-ésimo polinomio de Chebyshev generado a través de la recursión

$$c_k(z) = 2z c_{k-1}(z) - c_{k-2}(z), \qquad c_0 = 1, \quad c_1 = z.$$

Estos polinomios están acotados por la unidad en el intervalo $[-1, 1]$, pero crecen rápidamente fuera de este intervalo. Definiendo $p(x)$ de esta forma, tenemos que $|\, p(\lambda_i) \,| \leq 1$ para $i = 2, \ldots, n$ y $p(\lambda_1) = c_{k-1}(1 + 2\rho_1)$, donde ρ_1 se define como en (2.27). Por tanto,

$$\delta^2 \leq \sum_{i=2}^n d_i^2 = 1 - d_1^2,$$

donde se ha tenido en cuenta que q_1 tiene norma 1, y de aquí, usando (2.30) se tiene que

$$\theta_1 \geq \lambda_1 - (\lambda_1 - \lambda_n)\frac{1 - d_1^2}{d_1^2}\frac{1}{\left(c_{k-1}\left(1 + 2\rho_1\right)\right)^2}. \tag{2.32}$$

[1] notar que todo $w \in \mathcal{K}(A, q_1, k)$ se puede expresar como $w = p_w(A)q_1$, siendo $p_w(x)$ un polinomio de grado $k-1$

Puesto que $\cos(\phi_1) = \left|q_1^T z_1\right|$ y $d_1 = q_1^T z_1$, se tiene que $1 - d_1^2 = \sin^2(\phi_1)$; luego $(1 - d_1^2)/d_1^2 = \tan^2(\phi_1)$).

Finalmente, sustituyendo en (2.32), y teniendo en cuenta que $\theta_1 \leq \lambda_1$ tenemos que

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - (\lambda_1 - \lambda_n)\frac{\tan^2(\phi_1)}{(c_{k-1}(1+2\rho_1))^2},$$

como queríamos demostrar. □

Un resultado análogo con respecto al autovalor más pequeño de las matrices T_k puede ser demostrado como corolario del teorema anterior.

Corolario 2.2. *Usando la misma notación que en el teorema 2.3.3, si $\theta_k = \lambda_k(T_k)$, entonces*

$$\lambda_n \;\leq\; \theta_k \;\leq\; \lambda_n + (\lambda_1 - \lambda_n)\left(\frac{\tan(\phi_n)}{c_{k-1}(1+2\rho_n)}\right)^2,$$

donde $\cos(\phi_1) = \left|q_1^T z_n\right|$, y

$$\rho_1 = \frac{\lambda_{n-1} - \lambda_n}{\lambda_1 - \lambda_{n-1}}.$$

Demostración:
Notemos que el corolario 2.2 se demuestra sin más que aplicar el teorema 2.3.3 remplazando la matriz A por la matriz $-A$. □

El teorema 2.3.3 y el corolario 2.2 ofrecen información de la calidad de las aproximaciones al autovalor máximo y mínimo respectivamente en función del número de iteraciones k.

La idea principal detrás de la demostración del teorema 2.3.3 es seleccionar un polinomio amplificador de forma que $p(A)q_1$ amplifique la componente de q_1 en la dirección del autovector z_1. Esta misma idea se puede utilizar para obtener cotas para las sucesivas aproximaciones de los autovalores interiores (es decir, para los θ_i). Sin embargo, los resultados no son tan satisfactorios como para las aproximaciones de los autovalores máximo y mínimo de A, pues en este caso el polinomio amplificador involucra el producto de polinomios de Chebyshev c_{k-i} y del polinomio $(x - \lambda_1)\ldots(x - \lambda_{i-1})$.

El siguiente teorema muestra cotas de error para las aproximaciones de los autovalores interiores. Notemos que debido al factor κ_i, y al grado reducido del polinomio de Chebyshev, es claro que las cotas se deterioran cuando i crece.

Teorema 2.3.4. *Usando la misma notación que en el teorema 2.3.3, si $1 \leq i \leq k$ y $\theta_i = \lambda_i(T_k)$, entonces*

$$\lambda_i \;\geq\; \theta_i \;\geq\; \lambda_i - (\lambda_i - \lambda_n)\left(\frac{\kappa_i \tan(\phi_i)}{c_{k-i}(1+2\rho_i)}\right)^2,$$

donde

$$\rho_i = \frac{\lambda_i - \lambda_{i+1}}{\lambda_{i+1} - \lambda_n}, \qquad \kappa_i = \prod_{j=1}^{i-1} \frac{\theta_j - \lambda_n}{\theta_j - \lambda_i}, \qquad \cos(\phi_i) = \left|q_1^T z_i\right|.$$

Demostración:
Una demostración detallada puede ser encontrada en [10]. □

2.3.1. Comparación con el método de la potencia

El método de la potencia constituye posiblemente uno de los algoritmos numéricos más conocidos para aproximar el autovalor dominante de cierta matriz. Según se ha mencionado durante la deducción del método de Lanczos, existe cierta similitud entre este método y el método de la potencia. El objetivo de esta sección es comparar la convergencia hacia el autovalor maximal de cierta matriz simétrica que ofrecen estos métodos.

Supongamos que queremos aproximar numéricamente el autovalor maximal de cierta matriz simétrica A. Por simplicidad, supongamos que

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0, \tag{2.33}$$

y que para aproximar dichos autovalores, aplicamos tanto el método de la potencia como el método de Lanczos con cierto vector inicial q_1.

Notemos que, en general, el método de la potencia no converge hacia el autovalor mayor, sino hacia el autovalor dominante (mayor en valor absoluto), pero por (2.33) sabemos que este método convergerá (en caso de hacerlo) hacia λ_1. Tras $k-1$ iteraciones, el método de la potencia obtiene una nueva aproximación al autovector dominante en la dirección de

$$v = A^{k-1} q_1 = \sum_{i=1}^{n} d_i \lambda_i^{k-1} z_i,$$

donde se ha tenido en cuenta la expresión de q_1 dada en (2.29), junto con su estimación al autovalor dominante

$$\gamma_1 = \frac{v^T A v}{v^T v}.$$

De cara a obtener cotas para las aproximaciones obtenidas mediante el método de la potencia, podemos notar que, puesto que este método busca nuevos vectores únicamente en la dirección determinada por $A^{k-1}q_1$, siguiendo un razonamiento análogo al del teorema 2.3.3, el *polinomio amplificador* que aparece en la expresión (2.28) debe ser necesariamente $p(x) = x^{k-1}$.

Por tanto, la expresión (2.30) se traduce en

$$\gamma_1 \geq \frac{\lambda_1 d_1^2 p(\lambda_1)^2 + \lambda_n \delta^2}{d_1^2 p(\lambda_1)^2 + \delta^2} = \lambda_1 - \frac{(\lambda_1 - \lambda_n)\,\delta^2}{d_1^2 p(\lambda_1)^2 + \delta^2} = \lambda_1 - (\lambda_1 - \lambda_n) \cdot \frac{d_2^2 p(\lambda_2)^2 + \cdots + d_n^2 p(\lambda_n)^2}{d_1^2 p(\lambda_1)^2 + \cdots + d_n^2 p(\lambda_n)^2},$$

donde se ha definido δ^2 como en (2.31).

Mediante una simple acotación de la fracción de la expresión anterior, llegamos a que

$$\gamma_1 \geq \lambda_1 - (\lambda_1 - \lambda_n) \cdot \frac{p(\lambda_2)^2 \left(d_2^2 + \cdots + d_n^2\right)}{d_1^2 p(\lambda_1)^2} = \lambda_1 - (\lambda_1 - \lambda_n) \cdot \frac{1 - d_1^2}{d_1^2} \cdot \frac{p(\lambda_2)^2}{p(\lambda_1)^2} =$$
$$= \lambda_1 - (\lambda_1 - \lambda_n) \tan^2(\phi_1) \left(\frac{\lambda_2}{\lambda_1}\right)^{2(k-1)},$$

donde en la última igualdad se ha utilizado la expresión concreta de $p(x)$. Con todo esto, obtenemos la siguiente cota para la aproximación γ_1 a λ_1

$$\lambda_1 \;\geq\; \gamma_1 \;\geq\; \lambda_1 - (\lambda_1 - \lambda_n)\, \tan^2(\phi_1) \left(\frac{\lambda_2}{\lambda_1}\right)^{2(k-1)}. \tag{2.34}$$

Por tanto, en vistas de (2.26) y de (2.34), de cara a comparar las cotas inferiores de las aproximaciones que estos métodos ofrecen, no tenemos más que comparar

$$L_{k-1} \equiv \frac{1}{\left[c_{k-1}\left(2\frac{\lambda_1}{\lambda_2} - 1\right)\right]^2} \geq \frac{1}{\left[c_{k-1}\left(1 + 2\rho_1\right)\right]^2} \tag{2.35}$$

y

$$R_{k-1} \equiv \left(\frac{\lambda_2}{\lambda_1}\right)^{2(k-1)}. \tag{2.36}$$

La desigualdad de (2.35) se sigue de la monotonía de los polinomios de Chebyshev. Es evidente que $2\lambda_1/\lambda_2 - 1 \geq 1$, y que $1 + 2\rho_1 \geq 1$. Por otro lado, tenemos que $-\lambda_1\lambda_n \leq -\lambda_2\lambda_n$. Sumando a ambos lados de esta última igualdad $\lambda_1\lambda_2$, llegamos a $\lambda_1\left(\lambda_2 - \lambda_n\right) \leq \lambda_2\left(\lambda_1 - \lambda_n\right)$, por lo que

$$\frac{\lambda_1}{\lambda_2} \leq \frac{\lambda_2 + \lambda_1 - \lambda_n - \lambda_2}{\lambda_2 - \lambda_n} = 1 + \frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n}.$$

De aquí, se sigue que

$$2 \cdot \frac{\lambda_1}{\lambda_2} - 1 \leq 2\left(1 + \frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n}\right) = 1 + 2\frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n} = 1 + 2\rho_1. \tag{2.37}$$

Recordemos que el polinomio de Chebyshev de grado k-ésimo tiene k raíces en el intervalo $(-1, 1)$, concretamente, en los puntos $x_k = \cos(2(k-1)\pi/2n)$. Por tanto, en virtud del teorema de Rolle, deducimos que los $k-1$ ceros del polinomio $c_k'(x)$ están también contenidos en (-1,1). Además, es evidente que

$$c_k(x) \xrightarrow[x\to\infty]{} \infty,$$

ya que el coeficiente principal de $c_k(x)$ es 2^{k-1} para $k \geq 1$, luego se deduce que, si $k \geq 1$, $c_k'(x) > 0$ para $x > 1$. Por tanto, los polinomios de Chebyshev son crecientes en $x > 1$, luego, teniendo en cuenta (2.37), se sigue que $c_{k-1}(2\lambda_1/\lambda_2 - 1) \leq c_{k-1}(1+\rho_1)$, con lo que se demuestra la desigualdad de (2.35).

En la tabla 2.1 se comparan diferentes valores de L_{k-1}/R_{k-1} de cara a comparar ambos métodos.

λ_1/λ_2	$k = 5$	$k = 10$	$k = 15$	$k = 20$	$k = 25$	$k = 30$
2	$\frac{3.0 \times 10^{-6}}{3.9 \times 10^{-3}}$	$\frac{6.6 \times 10^{-14}}{3.8 \times 10^{-6}}$	$\frac{1.5 \times 10^{-21}}{3.7 \times 10^{-9}}$	$\frac{3.2 \times 10^{-29}}{3.6 \times 10^{-12}}$	$\frac{7.2 \times 10^{-37}}{3.6 \times 10^{-15}}$	$\frac{1.6 \times 10^{-44}}{3.5 \times 10^{-18}}$
1.5	$\frac{1.1 \times 10^{-4}}{3.9 \times 10^{-2}}$	$\frac{2.0 \times 10^{-10}}{6.8 \times 10^{-4}}$	$\frac{3.9 \times 10^{-16}}{1.2 \times 10^{-5}}$	$\frac{7.4 \times 10^{-22}}{2.0 \times 10^{-7}}$	$\frac{1.4 \times 10^{-27}}{3.5 \times 10^{-9}}$	$\frac{2.7 \times 10^{-33}}{6.1 \times 10^{-11}}$
1.3	$\frac{9.2 \times 10^{-4}}{1.2 \times 10^{-1}}$	$\frac{2.6 \times 10^{-8}}{8.9 \times 10^{-3}}$	$\frac{7.4 \times 10^{-13}}{6.4 \times 10^{-4}}$	$\frac{2.1 \times 10^{-17}}{4.7 \times 10^{-5}}$	$\frac{5.9 \times 10^{-22}}{3.4 \times 10^{-6}}$	$\frac{1.7 \times 10^{-26}}{2.5 \times 10^{-7}}$
1.1	$\frac{2.7 \times 10^{-2}}{4.7 \times 10^{-1}}$	$\frac{5.4 \times 10^{-5}}{1.7 \times 10^{-1}}$	$\frac{1.1 \times 10^{-7}}{6.9 \times 10^{-2}}$	$\frac{2.1 \times 10^{-10}}{2.6 \times 10^{-2}}$	$\frac{4.2 \times 10^{-12}}{1.0 \times 10^{-2}}$	$\frac{8.4 \times 10^{-16}}{4.0 \times 10^{-3}}$
1.01	$\frac{5.6 \times 10^{-1}}{9.2 \times 10^{-1}}$	$\frac{1.0 \times 10^{-1}}{8.3 \times 10^{-1}}$	$\frac{1.14 \times 10^{-2}}{7.5 \times 10^{-1}}$	$\frac{2.0 \times 10^{-3}}{6.8 \times 10^{-1}}$	$\frac{2.8 \times 10^{-4}}{6.2 \times 10^{-1}}$	$\frac{3.7 \times 10^{-5}}{5.6 \times 10^{-1}}$

Tabla 2.1: L_{k-1}/R_{k-1} para diferentes valores de k y de λ_1/λ_2

Viendo los valores que muestra la tabla 2.1, es evidente la superioridad en convergencia con el número de iteraciones del método de Lanczos frente al método de la potencia. Sin embargo, esto no debería sorprendernos en absoluto, pues θ_1 es el máximo de $r(A, x) = x^T Ax/x^T x$ sobre todos los vectores no nulos de $\mathcal{K}(A, q_1, k)$ mientras que γ_1 es simplemente $r(A, v)$ con $v = A^{k-1}q_1$, esto es, un vector particular de $\mathcal{K}(A, q_1, k)$.

2.4. Aritmética exacta

El uso de aritmética de precisión finita provoca grandes discrepancias con lo que cabría esperar al implementar el algoritmo 1 si se usase aritmética exacta. Teóricamente, los autovalores θ_i de las matrices T_k convergen muy rápidamente a los autovalores de la matriz A. Sin embargo, en la práctica, si se realizan suficientes iteraciones del algoritmo, suelen aparecer autovalores y autovectores aproximados repetidos en las matrices T_k, excediendo la multiplicidad del autovalor correspondiente en A. Además, en otras ocasiones, el algoritmo devuelve ciertos autovalores incorrectos como convergentes (*autovalores espurios).*

Un análisis detallado del método bajo aritmética finita muestra que este comportamiento no deseado aparece al mismo tiempo que los autovectores de Lanczos comienzan a perder ortogonalidad mutua. Lanczos ya era consciente de este problema, y propuso reortogonalizar explícitamente cada vector q_k con respecto a todos

los vectores anteriores en cada paso. Este proceso se conoce como *Algoritmo de Lanczos con reortogonalización completa.* Aunque este procedimiento es eficaz, la reortonormalización completa supone un aumento notable del costo computacional del algoritmo, por lo que únicamente se reserva para casos en los que se desea aproximar unos pocos autovalores. Estudios posteriores han propuesto otras alternativas para poder hacer frente a la pérdida de ortogonalidad a menor coste.

En esta sección, se ilustrará, mediante un ejemplo, las consecuencias que puede tener implementar el algoritmo 1 en aritmética finita. Después, se implementará el *Algoritmo de Lanczos con reortogonalización completa,* y finalmente, se estudiarán algunos de los procedimientos más habituales utilizados para reducir el número de operaciones necesarios para la reortonormalización completa.

Ejemplo 2.3. *Supongamos que queremos aproximar mediante el algoritmo de Lanczos los autovalores de la matriz* $A = diag\,(0.0001,\ 0.00025,\ 0.0005,\ 0.035,\ 0.6, 80)$*, utilizando como vector inicial* $q_1 = [1, \dots, 1]^T$*. Así definida, la matriz* A *tiene seis autovalores simples. Notemos que, en virtud del teorema 2.2.1, esperamos que el algoritmo 1 finalice tras* $m = rank(K(A, q_1, k))$ *iteraciones, donde* $K(A, q_1, k)$ *se define como en (2.10). Denotemos por* λ_i *al i-ésimo autovalor de* A *(i-ésima entrada diagonal). Notemos que para cada* $i = 2, \dots, 6$ *se tiene que* $A^{i-1}q_1 = \left[\lambda_1^{i-1}, \dots, \lambda_6^{i-1}\right]^T$*, luego*

$$K(A, q_1, k) = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^5 \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^5 \\ 1 & \lambda_3 & \lambda_3^2 & \dots & \lambda_3^5 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_6 & \lambda_6^2 & \dots & \lambda_6^5 \end{bmatrix}, \tag{2.38}$$

es decir, la matriz de Krylov $K(A, q_1, k)$ *es una matriz de Vandermonde. Puesto que* $\lambda_i \neq \lambda_j$ *si* $i \neq j$*, sabemos que tiene rango máximo, luego como*

$$rank(K(A, q_1, k)) = rank\left(\left[q_1 |\ Aq_1 \mid A^2q_1 \mid \dots \mid A^5q_1\right]\right) = 6 = dim(A), \tag{2.39}$$

cabría esperar que, al ejecutar el algoritmo de Lanczos con un total de $k = 6$ *iteraciones, obtengamos los autovalores exactos (hasta precisión de la máquina). Aplicando el algoritmo 1 a los datos mencionados, obtenemos que*

$$Q_6^T Q_6 = \begin{pmatrix} 1.000 & -8.9 \times 10^{-17} & 2.1 \times 10^{-17} & 9.3 \times 10^{-12} & 2.1 \times 10^{-6} & -0.3958 \\ -8.9 \times 10^{-17} & 1.000 & 1.0 \times 10^{-14} & 2.1 \times 10^{-11} & 4.8 \times 10^{-6} & 0.8850 \\ 2.1 \times 10^{-17} & 1.0 \times 10^{-14} & 1.000 & -1.8 \times 10^{-13} & 3.4 \times 10^{-8} & 0.0064 \\ 9.3 \times 10^{-12} & 2.1 \times 10^{-11} & -1.8 \times 10^{-13} & 1.000 & 1.7 \times 10^{-11} & 2.6 \times 10^{-6} \\ 2.1 \times 10^{-6} & 4.8 \times 10^{-6} & 3.4 \times 10^{-8} & 1.7 \times 10^{-11} & 1.000 & -1.8 \times 10^{-11} \\ -0.3958 & -0.8850 & 0.0064 & 2.6 \times 10^{-6} & -1.8 \times 10^{-11} & 1.000 \end{pmatrix},$$

donde vemos que los vectores q_i *pierden la ortogonalidad a partir de la cuarta iteración. Por otra parte, obtenemos que los autovalores aproximados*[2] *mediante este*

[2]Se obtienen estos valores hasta precisión de máquina.

algoritmo son los siguientes

$$\theta_1 = 1.405 \times 10^{-4}, \qquad \theta_2 = 4.740 \times 10^{-4}, \qquad \theta_3 = 0.0350,$$
$$\theta_4 = 0.6000, \qquad \theta_5 = 75.188, \qquad \theta_6 = 80.000.$$

Observamos que no se han obtenido aproximaciones a todos los autovalores, sino que, en su lugar, uno de ellos ha sido aproximado de forma múltiple (el autovalor $\lambda_1 = 80.000$*)*

El ejemplo anterior sirve para motivar la necesaria reortogonalización de los vectores q_i al aplicar el algoritmo de Lanczos. En las siguientes secciones veremos algunas de las técnicas más utilizadas para solventar la pérdida de ortogonalidad, y las ideas detrás de ellas.

2.4.1. Algoritmo de Lanczos con Reortogonalización Completa

Como ya se ha mencionado, Lanczos era consciente de la pérdida de ortogonalidad de los vectores q_i, y propuso para solventar este problema una reortogonalización completa. Sin embargo, aunque este procedimiento es eficaz, aumenta notablemente el número de operaciones en cada iteración del algoritmo. Este aumento del número de operaciones fue la causa de que este algoritmo no fuese prácticamente utilizado en la década de 1950 y 1960, y de que no se popularizase su uso hasta la década de 1970, gracias al trabajo de diversos matemáticos entre los que destacan Paige, Kahan, Parlett y Scott.

El algoritmo con reortonormalización completa puede ser implementado mediante el pseudocódigo mostrado en el algoritmo 2.

Algoritmo 2: Algoritmo de Lanczos con Reortonormalización Completa

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$, un vector aleatorio $q \in \mathbb{R}^n$, y m el número de iteraciones
Salida : Matriz tridiagonal $T_k \in \mathbb{R}^{k\times k}$, y $Q_k = [q_1 \mid \ldots \mid q_k] \in \mathbb{R}^{n\times k}$ tal que $AQ_k = Q_kT_k$

$q_1 = q/\|q\|$, $\beta_0 = 1$, $q_0 = 0$.
for $k = 1$ **to** m **do**
  $\alpha_k = q_k^T A q_k$;
  $r_k = Aq_k - \alpha_k q_k - \beta_{k-1} q_{k-1}$
  $r_k = r_k - Q_k(Q_k^T r_k)$
  $\beta_k = \|r_k\|$;
  if $\beta_k \neq 0$ **then**
    $q_{k+1} = r_k/\beta_k$
  end
end

En el siguiente ejemplo comparamos los resultados que obtenemos al implementar esta nueva versión del algoritmo de Lanczos con lo que obtuvimos en el ejemplo 2.3.

Ejemplo 2.4. *Supongamos que queremos aproximar mediante el algoritmo de Lanczos los autovalores de la matriz* $A = diag(0.0001, 0.00025, 0.0005, 0.035, 0.6, 80)$*, utilizando como vector inicial* $q_1 = [1, \dots, 1]^T$*, y aplicamos para ello el algoritmo 2.*

Sabemos por (2.39) que el algoritmo no terminará de forma prematura, luego al haber realizado la reortonormalización, esperamos obtener de forma exacta los autovalores de A. Aplicando dicho algoritmo a los datos mencionados, obtenemos que

$$Q_6^T Q_6 = \begin{bmatrix} 1.000 & -8.9\times10^{-17} & -1.1\times10^{-17} & -1.9\times10^{-17} & 2.3\times10^{-17} & 8.2\times10^{-18} \\ -8.9\times10^{-17} & 1.000 & 6.6\times10^{-18} & 1.1\times10^{-17} & -2.6\times10^{-17} & 3.5\times10^{-18} \\ -1.1\times10^{-17} & 6.6\times10^{-18} & 1.000 & -2.13\times10^{-17} & 3.5\times10^{-17} & 2.3\times10^{-17} \\ -1.9\times10^{-17} & 1.1\times10^{-17} & -2.1\times10^{-17} & 1.000 & -1.8\times10^{-17} & -1.3\times10^{-17} \\ 2.3\times10^{-17} & -2.6\times10^{-18} & 3.5\times10^{-17} & -1.8\times10^{-17} & 1.000 & -8.6\times10^{-18} \\ 8.2\times10^{-18} & 3.5\times10^{-18} & 2.3\times10^{-17} & -1.3\times10^{-17} & -8.6\times10^{-17} & 1.000 \end{bmatrix}.$$

Observamos que en este caso no se pierde ortogonalidad entre los vectores (puesto que $Q_6^T Q_6 \approx I_6$*). Ahora, los autovalores aproximados*[3] *son*

$$\theta_1 = 1.0\times10^{-4}, \quad \theta_2 = 2.5\times10^{-4}, \quad \theta_3 = 5\times10^{-4},$$
$$\theta_4 = 0.0350, \quad \theta_5 = 0.6000, \quad \theta_6 = 80.000.$$

2.4.2. Ortogonalización selectiva

Una consecuencia irónica de un análisis del error de Paige, que explicaremos en esta sección, es que la pérdida de ortogonalidad ocurre con la convergencia de los autovalores y autovectores aproximados por el método de Lanczos. Aunque en principio podría parecer no deseable, este hecho es la base de la técnica de ortogonalización selectiva, posiblemente uno de los esquemas más utilizados para minimizar el número de operaciones debidas a la reortonormalización de los vectores q_i que necesitamos realizar.

En esta sección, realizaremos en primer lugar un estudio de la convergencia cuando se implementa el algoritmo de Lanczos con aritmética finita, y, posteriormente, se explicará e implementará el *Algoritmo de Lanczos con ortonormalización selectiva.*

Supongamos que se han realizado k iteraciones del algoritmo de Lanczos implementado con aritmética finita sin que este termine de forma prematura. De ahora en adelante, y a lo largo de esta sección, Q_i, T_i, $R_i, \beta_i \dots$ denotan a los vectores, matrices o escalares que han sido calculados de forma numérica, y no los calculados mediante aritmética exacta. Además, diremos que el par de autovalor y autovector (θ_i, y_i) es un *buen par* (buen autovalor y autovector respectivamente) si satisface que

$$\|Ay_k - \theta_k y_k\| \approx \sqrt{\epsilon}\|A\|_2, \tag{2.40}$$

donde ϵ es el épsilon de la máquina, y nos referiremos a θ_i e y_i como autovalor y autovector de Ritz. Además, supondremos que el problema de autovalores de la

[3] Hasta precisión de máquina.

matriz tridiagonal puede ser resuelto de forma exacta. Por tanto, podemos encontrar matrices ortogonales S_j tales que

$$T_j = S_j \Theta_k S_j^T, \qquad \text{con } \Theta_j = \text{diag}(\theta_1, \ldots, \theta_j). \tag{2.41}$$

Con todo ello, para $j = 1, \ldots, k$ la ecuación (2.14) se escribe como

$$AQ_j = Q_j T_j + r_j e_j^T + F_j \tag{2.42}$$

donde e_j es el vector canónico de tamaño $j \times 1$ con el 1 en su última componente,y F_j da cuenta de los errores de redondeo. Premultiplicando la expresión (2.42) por Q_j^T se obtiene que

$$Q_j^T A Q_j = Q_j^T Q_j T_j + Q_j^T r_j e_j^T + Q_j^T F_j. \tag{2.43}$$

Puesto que $Q_j^T A Q_j$ es una matriz simétrica, restando a (2.43) su expresión traspuesta, llegamos a que

$$0 = Q_j^T Q_j T_j - T_j Q_j^T Q_j + Q_j^T r_j e_j^T - e_j r_j^T Q_j + Q_j^T F_j - F_j^T Q_j,$$

luego

$$\begin{aligned} Q_j^T r_j e_j^T &= T_j Q_j^T Q_j - Q_j^T Q_j T_j + F_j^T Q_j - Q_j^T F_j + e_j r_j^T Q_j \\ &= (I_j - Q_j^T Q_j) T_j - T_j (I_j - Q_j^T Q_j) + F_j^T Q_j - Q_j^T F_j + e_j r_j^T Q_j. \end{aligned} \tag{2.44}$$

Consideremos las matrices C_j y Δ_j tales que

$$I_j - Q_j^T Q_j = C_j^T + \Delta_j + C_j,$$

donde I_j es la matriz identidad de tamaño j, C_j es una matriz estrictamente triangular superior, y Δ_j es una matriz diagonal. Sin más que sustituir en (2.44), tenemos que

$$\begin{aligned} Q_j^T r_j e_j^T =& (C_j^T + \Delta_j + C_j) T_j - T_j (C_j^T + \Delta_j + C_j) + F_j^T Q_j \\ & - Q_j^T F_j + e_j r_j^T Q_j. \end{aligned} \tag{2.45}$$

De ahora en adelante, dada una matriz cuadrada M, denotaremos por $\nabla(M)$ a su parte triangular superior (incluyendo la diagonal). Es evidente, dadas dos matrices cuadradas R y S del mismo tamaño, que $\nabla(R + S) = \nabla(R) + \nabla(S)$, y que $\nabla(RS) \neq \nabla(R)\nabla(S)$ en general.

Si consideramos la parte triangular superior en ambos lados de la expresión (2.45), tenemos que

$$\begin{aligned} \nabla(Q_j^T r_j e_j^T) =& \nabla((C_j^T + \Delta_j + C_j) T_j) - \nabla(T_j (C_j^T + \Delta_j + C_j)) + \nabla(F_j^T Q_j) - \nabla(Q_j^T F_j) \\ & + \nabla(e_j r_j^T Q_j) \\ =& \nabla(C_j T_j - T_j C_j) + \nabla(\Delta_j T_j - T_j \Delta_j) + \nabla(C_j^T T_j - T_j C_j^T) \\ & + \nabla(F_j^T Q_j) - \nabla(Q_j^T F_j) + \nabla(e_j r_j^T Q_j). \end{aligned}$$

Definamos las matrices triangulares superiores

$$B_j = \nabla(\Delta_j T_j - T_j \Delta_j) + \nabla(C_j^T T_j - T_j C_j^T) + \nabla(e_j r_j^T Q_j)$$

y

$$E_j = \nabla(F_j^T Q_j - Q_j^T F_j),$$

de forma que la expresión anterior se convierte en

$$\nabla(Q_j^T r_j e_j^T) = \nabla(C_j T_j - T_j C_j) + B_j + E_j.$$

Ahora bien, la matriz $Q_j^T r_j e_j^T$ es idénticamente nula salvo su última columna, que es precisamente $Q_j^T r_j$, luego $\nabla(Q_j^T r_j e_j^T) = Q_j^T r_j e_j^T$. Por otra parte, puesto que T_j es una matriz tridiagonal, y C_j es una matriz estrictamente triangular superior, tenemos que $C_j T_j - T_j C_j$ es también triangular superior, por lo que $\nabla(C_j T_j - T_j C_j) = C_j T_j - T_j C_j$. Por tanto,

$$Q_j^T r_j e_j^T = C_j T_j - T_j C_j + B_j + E_j. \tag{2.46}$$

Considerando la ecuación (2.46) para $j = k$, y premultiplicando dicha expresión por por s_i^T, y postmultiplicando por s_i (donde s_i es una columna de la matriz S_k), llegamos a que

$$\underbrace{s_i^T Q_j^T}_{y_i^T} \underbrace{r_k}_{\beta_k q_{k+1}} \underbrace{e_k^T s_i}_{s_{ki}} = s_i^T (C_k T_k - T_k C_k) s_i + s_i^T (B_k + E_k) s_i, \tag{2.47}$$

que se simplifica a

$$\beta_k s_{ki} y_i^T q_{k+1} = (s_i^T C_k s_i)\theta_i - \theta_i (s_i^T C_k s_i) + s_i^T (B_k + E_k) s_i = s_i^T (B_k + E_k) s_i,$$

y, sin más que despejar en la igualdad anterior, concluimos que

$$y_i^T q_{k+1} = \frac{s_i^T (B_k + E_k) s_i}{\beta_k s_{ki}}. \tag{2.48}$$

Scott demuestra en el *Anexo A* de [11] que $\left\| s_i^T (B_k + E_k) s_i \right\|_2 \approx \epsilon \|A\|_2$. Esta demostración se basa en un enunciado auxiliar demostrado por Paige. Aunque la demostración no es compleja, requiere de varios resultados auxiliares, por lo que omitiremos su demostración. Tomando por cierto este resultado, tenemos que

$$\left| y_i^T q_{k+1} \right| \approx \frac{\epsilon \|A\|_2}{|\beta_k||s_{ki}|}. \tag{2.49}$$

En otras palabras, la componente del nuevo vector de Lanczos q_{k+1} en la dirección del autovector de Ritz y_i es inversamente proporcional a $|\beta_k||s_{ki}|$, que según vimos en el teorema 2.3.1, es una medida de lo bueno que es el autovalor de Ritz θ_i como aproximación a un autovalor de A. Más concretamente, cuando el par (θ_i, y_i) converge, $|\beta_k||s_{ki}|$ tiende a cero, y el nuevo vector de Lanczos q_{k+1} tiene una componente grande en la dirección del i-ésimo autovector de Ritz.

La *ortogonalización selectiva* utiliza este hecho: tan pronto como el nuevo vector de Lanczos q_{k+1} es calculado, es ortogonalizado respecto a cada uno de los buenos autovectores de Ritz. Esto es mucho menos costoso que una reortogonalización completa, pues en general hay muchos menos buenos vectores que vectores de Lanczos. Una forma de implementar la ortogonalización selectiva es diagonalizar en cada k-ésima iteración la matriz T_k, y ortogonalizar el nuevo vector de Lanczos q_{k+1} respecto a todos los buenos autovectores.

Cabe notar que, tanto al hablar de buenos vectores como en (2.49), consideramos $\|A\|_2$, que, evidentemente, es desconocida. Sin embargo, puesto que hemos visto que los autovalores extremos son los primeros en converger, y que además lo hacen de forma muy rápida, se tiene que $\|A\|_2 \approx \|T\|_2$. De hecho, si $\theta_1, \ldots, \theta_k$ son los autovalores aproximados en la k-ésima iteración por el algoritmo de Lanczos, y $\lambda_1, \ldots, \lambda_n$ los autovalores exactos de A, se tiene que $\theta_1 \leq \lambda_1$, y que $\theta_k \geq \lambda_n$, luego es evidente que $\rho(T_K) \leq \rho(A)$, de donde se sigue que $\|A\|_2 \geq \|T_k\|_2$.

Por tanto, se verifica la siguiente desigualdad

$$\sqrt{\epsilon}\|T_k\|_2 \leq \sqrt{\epsilon}\|A\|_2, \tag{2.50}$$

por lo que utilizando en cada k-ésima iteración la norma de T_k para comprobar los buenos vectores (definidos según (2.40)), tenemos que el algoritmo toma la siguiente forma:

Algoritmo 3: Algoritmo de Lanczos con Reortonormalización Selectiva. Versión 1

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$, un vector aleatorio $q \in \mathbb{R}^n$, y m el número de iteraciones
Salida : Matriz tridiagonal $T_k \in \mathbb{R}^{k\times k}$, y $Q_k = [q_1 \mid \ldots \mid q_n] \in \mathbb{R}^{n\times k}$ tal que $AQ_k = Q_kT_k$

$q_1 = q/\|q\|$, $\beta_0 = 1$, $q_0 = 0$.
for $k = 1$ **to** m **do**
  $\alpha_k = q_k^T A q_k$;
  $r_k = Aq_k - \alpha_k q_k - \beta_{k-1}q_{k-1}$
  Calcular la factorización $T_k = S_k\Theta_k S_k^T$
  for $j = 1$ **to** k **do**
    if $|\beta_k s_{kj}| \leq \sqrt{\epsilon}\|T_k\|_2$ **then**
      $v = Q_k s_j$
      $r_k = r_k - (v^T r_k)v$
    end
  end
  $\beta_k = \|r_k\|_2$
  if $\beta_k \neq 0$ **then**
    $q_{k+1} = r_k/\beta_k$
  end
end

Comparemos los resultados obtenidos al aplicar el algoritmo de reortonormalización selectiva al ejemplo anteriormente estudiado.

Ejemplo 2.5. *Supongamos que queremos aproximar mediante el algoritmo de Lanczos los autovalores de la matriz* $A = diag(0.0001, 0.00025, 0.0005, 0.035, 0.6, 80)$*, utilizando como vector inicial* $q_1 = [1, \ldots, 1]^T$*, y aplicamos para ello el algoritmo 3.*

Aplicando el algoritmo de Lanczos con reortonormalización selectiva obtenemos

$$Q_6^T Q_6 = \begin{bmatrix} 1.000 & -8.9 \times 10^{-17} & 2.1 \times 10^{-17} & 9.3 \times 10^{-12} & 2.2 \times 10^{-10} & 3.5 \times 10^{-12} \\ -8.9 \times 10^{-17} & 1.000 & 1.1 \times 10^{-14} & 2.1 \times 10^{-11} & -9.7 \times 10^{-11} & -1.6 \times 10^{-12} \\ 2.1 \times 10^{-17} & 1.1 \times 10^{-14} & 1.000 & -1.8 \times 10^{-13} & -4.8 \times 10^{-10} & 2.1 \times 10^{-12} \\ 9,3 \times 10^{-12} & 2.1 \times 10^{-11} & -1.8 \times 10^{-13} & 1.000 & 3.1 \times 10^{-12} & -2.3 \times 10^{-12} \\ 2.2 \times 10^{-10} & -9.6 \times 10^{-11} & -4.7 \times 10^{-10} & 3.1 \times 10^{-12} & 1.000 & -7.2 \times 10^{-12} \\ 3.5 \times 10^{-12} & -1.6 \times 10^{-12} & 2.1 \times 10^{-12} & -2.3 \times 10^{-12} & -7.1 \times 10^{-12} & 1.000 \end{bmatrix},$$

mientras que los autovalores aproximados[4] *son*

$$\theta_1 = 1.0 \times 10^{-4}, \quad \theta_2 = 2.5 \times 10^{-4}, \quad \theta_3 = 5 \times 10^{-4},$$
$$\theta_4 = 0.0350, \quad \theta_5 = 0.6000, \quad \theta_6 = 80.000.$$

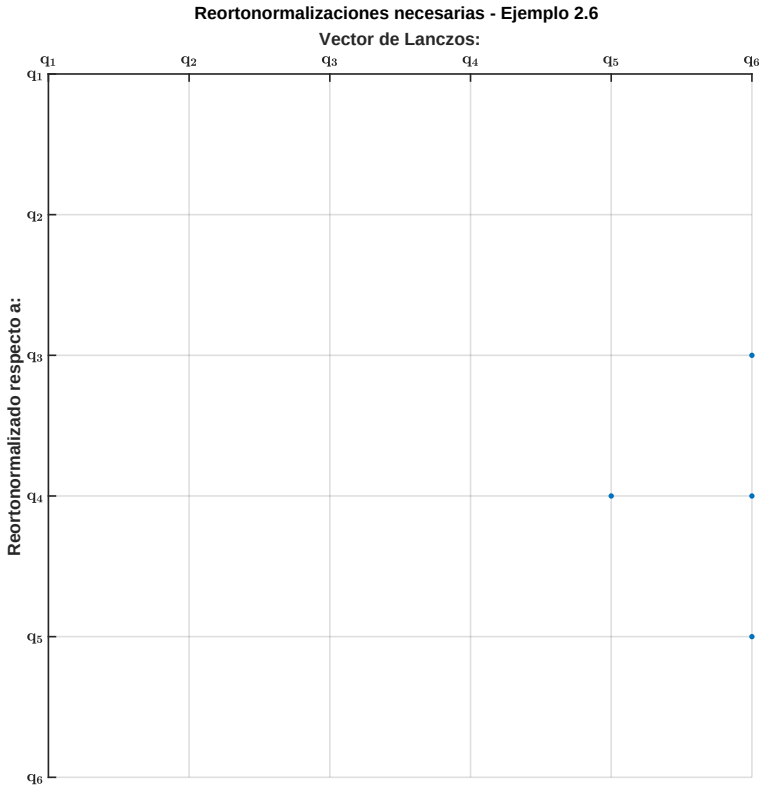

Figura 2.4: Localización de las reortonormalizaciones necesarias para el Ejemplo 2.5.

Es interesante estudiar en qué momento y cuántas veces ha sido necesario realizar la

[4] Hasta octava cifra signiticativa.

reortogonalización selectiva. En la figura 2.4 se ha representado dicho estudio. Ahí podemos observar que la pérdida de ortogonalidad se produce con el quinto de los vectores de Lanczos (q_5 utilizando la notación de dicha figura), y que este únicamente debe ser reortogonalizado con q_4. Posteriormente, q_6 debe ser reortogonalizado con respecto a q_3, q_4 y q_5. De esta forma, frente a las 15 reortonormalizaciones que se realizan con el esquema de reortogonalización completa, usando la reortogonalización selectiva únicamente debemos reortonormalizar 4 veces.

Veamos un ejemplo con una matriz de mayor tamaño.

Ejemplo 2.6. *Consideremos una matriz A dispersa, simétrica y definida positiva de tamaño $n = 80$. Generaremos una matriz de manera aleatoria con estas características, pero indicando la semilla utilizada en el proceso, de forma que A pueda ser recuperada en todo momento. Para estos propósitos, podemos utilizar el siguiente código de Matlab*

```
rng(2,'philox');      %semilla utilizada
n=80;
A=sprandsym(n,0.3,0.04,1);
%sprandsym(n,d,rc,1) genera una matriz tamaño nxn, con ↩
   densidad de ceros d, e inverso de número de condición rc.
```

que genera una matriz simétrica con aproximadamente $0.3 \cdot 80 \cdot 80$ ceros, sobre la que aplicaremos el algoritmo 3, tomando como vector inicial $q_1 = [1, \ldots, 1]^T$.

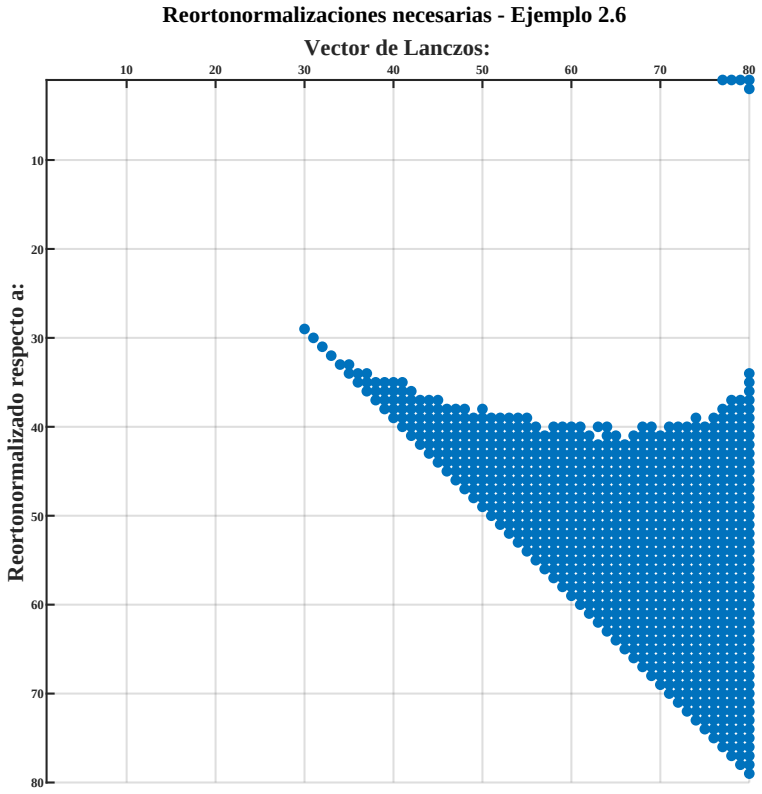

Figura 2.5: Localización de las reortonormalizaciones necesarias para el Ejemplo 2.6.

Evidentemente, en la práctica rara vez iteramos el algoritmo el número máximo de veces pero, puesto que esta matriz es relativamente pequeña, podemos hacerlo de cara a estudiar este procedimiento. En la figura 2.5 se han representado las reortogonalizaciones que han sido necesarias. Por una parte, vemos que en caso de iterar 80 *veces, únicamente han sido necesarias* 889 *reortogonalizaciones, frente a las* $n\,(n-1)/2 = 3160$ *que serían necesarias al aplicar el algoritmo de reortogonalización completa.*

Por otra parte, para estudiar la calidad de las reortogonalizaciones, pongamos

$$S = Q_{80}^T Q_{80} = (s_{ij})_{\substack{1\leq i\leq 80\\ 1\leq j\leq 80}}.$$

Entonces, se puede comprobar que

$$\max_{\substack{1\leq i,j\leq 80\\ i\neq j}} |s_{i,j}| = 2.801 \times 10^{-9},$$

por lo que en la práctica, los vectores q_i *son ortogonales cercanos hasta precisión de máquina.*

La ortogonalización selectiva no es el último paso que se puede dar en el método de Lanczos. Existen otros esquemas, como puede ser el método de Lanczos con reiniciado o el método de Lanczos con s pasos, que mejoran algunas de las características de los métodos hasta ahora desarrollados, pero que no serán desarrollados en este trabajo, pues su complejidad escapa de los objetivos del mismo.

En su lugar, se ha preferido enfatizar en las consecuencias que se derivan de aplicar este algoritmo en aritmética finita, y proponer algunos de los métodos que se pueden utilizar para solventar estos problemas. Es cierto que se podrían seguir estudiando muchos de los aspectos prácticos del método de Lanczos, pero para nuestros propósitos, nos sirve con los algoritmos y la teoría hasta ahora desarrollada.

Capítulo 3

Resolución de Sistemas Lineales

En el capítulo anterior hemos visto cómo utilizar diferentes variantes del método de Lanczos para aproximar autovalores de matrices simétricas, dispersas y de gran tamaño. El algoritmo de Lanczos también puede ser utilizado para resolver sistemas lineales donde la matriz de coeficientes es simétrica, dispersa y de gran tamaño. Como veremos a lo largo del capítulo, utilizar el método de Lanczos con este propósito equivale a utilizar el método del gradiente conjugado.

3.1. Sistemas simétricos definidos positivos

Supongamos que $A \in \mathbb{R}^{n\times n}$ es una matriz simétrica y definida positiva, y supongamos que queremos resolver el sistema de ecuaciones lineales $Ax = b$, donde $b \in \mathbb{R}^n$. Para dicho vector de términos independientes, consideremos el operador $\phi(x)$ definido por

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b. \tag{3.1}$$

Sin más que calcular el gradiente de dicho operador, tenemos que $\nabla\phi(x) = Ax-b$, luego $x = A^{-1}b$ es el único minimizador de ϕ. Por tanto, podemos ver un minimizador aproximado de ϕ como una solución aproximada del sistema $Ax = b$.

Supongamos que $x_0 \in \mathbb{R}^n$ es una aproximación inicial al mínimo de ϕ, y que queremos generar una secuencia de vectores $\{x_k\}$ que converja a x. Una forma de hacer esto es generar una secuencia de vectores ortonormales $\{q_k\}$, y hacer que x_k minimice ϕ sobre el conjunto

$$S := x_0+\text{span}\{q_1,\dots,q_k\} = \{x_0+a_1q_1+\cdots+a_kq_k \ : a_k \in \mathbb{R}\} \qquad \text{para } k = 1,\dots,n.$$

Evidentemente, se tiene que $x_n = x$, ya que $\text{span}\{q_1,\dots,q_n\} = \mathbb{R}^n$, pero esperamos que x_k sea una buena aproximación a x mucho antes de que $k = n$. Notemos que todo vector $x \in S$ se puede expresar mediante

$$x = x_0 + Q_k y_k, \qquad \text{donde } Q_k = [q_1,\dots,q_k] \text{ e } y_k \in \mathbb{R}^k. \tag{3.2}$$

Por tanto, de cara a minimizar ϕ sobre S, tenemos que buscar un vector y_k que minimice

$$\begin{aligned}\phi(x_0 + Q_k y_k) &= \frac{1}{2}(x_0 + Q_k y_k)^T A(x_0 + Q_k y_k) - (x_0 + Q_k y_k)^T b \\ &= \frac{1}{2} y_k^T (Q_k^T A Q_k) y_k - y_k^T Q_k^T (b - A x_0) + \frac{1}{2} x_0^T A x_0 - x_0^T b \\ &= \frac{1}{2} y_k^T (Q_k^T A Q_k) y_k - y_k^T Q_k^T (b - A x_0) + \phi(x_0).\end{aligned} \tag{3.3}$$

El gradiente de la expresión anterior respecto de y_k viene dado por

$$\nabla_{y_k} \phi(x_0 + Q_k y_k) = (Q_k^T A Q_k) y_k - Q_k^T (b - A x_0),$$

luego $\nabla_{y_k} \phi(x_0 + Q_k y) = 0$ si se cumple que

$$(Q_k^T A Q_k) y_k = Q_k^T (b - A x_0). \tag{3.4}$$

De esta forma, para encontrar sucesivas mejores estimaciones al minimizador x, no tenemos más que resolver el sistema (3.4) para obtener y_k, y poder así calcular $x = x_0 + Q_k y_k$.

El Algoritmo de Lanczos anteriormente estudiado puede ayudarnos en esta tarea. Supongamos que se han realizado k iteraciones del algoritmo de Lanczos, de forma que hemos obtenido la factorización

$$A Q_k = Q_k T_k + r_k e_k^T, \tag{3.5}$$

donde

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \dots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{k-1} \\ 0 & \dots & & \beta_{k-1} & \alpha_k \end{bmatrix}, \tag{3.6}$$

y pongamos $q_1 = r_0/\beta_0$, donde $r_0 = b - A x_0$ y $\beta_0 = \|r_0\|_2$. Entonces, $\beta_0 Q_k(:, 1) = r_0$, de forma que el sistema lineal (3.4), que debemos resolver de cara a obtener y_k, se puede escribir como

$$T_k y_k = Q_k^T r_0 = \beta_0 e_1, \tag{3.7}$$

donde e_1 es el vector canónico de tamaño k con el 1 en su primera componente.

Con todo ello, el siguiente algoritmo recoge una versión preliminar de cómo el algoritmo de Lanczos puede ser utilizado para resolver sistemas lineales simétricos definidos positivos. Podemos observar que el algoritmo anterior requiere de un procedimiento para resolver sistemas lineales. La ventaja es que reducimos notablemente la dimensión del sistema a resolver. Sin embargo, si bien el algoritmo recientemente descrito es válido, su formulación no es adecuada para problemas de gran tamaño, pues x_k es calculado de forma explícita como un producto de una matriz de tamaño

Algoritmo 4: Resolución de sistemas lineales mediante el algoritmo de Lanczos. Versión 1

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, un vector $b \in \mathbb{R}^n$ y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$, $\beta_0 = \|r_0\|_2$ y $q_0 = 0$.

while $\beta_k \neq 0$ **do**

$q_{k+1} = r_k/\beta_k$

$k = k + 1$

$\alpha_k = q_k^T A q_k$

Resolver el sistema $T_k y_k = \beta_0 e_1$

$x_k = Q_k y_k$

$r_k = (A - \alpha_k I)q_k - \beta_{k-1} q_{k-1}$

$\beta_k = \|r_k\|_2$

end

$x = x_0 + x_k$

$(n \times k)$ por un vector, y esto requiere almacenar y emplear en cada iteración todos los vectores de Lanczos q_i previamente calculados. De cara a desarrollar un algoritmo eficiente, necesitamos un proceso que resuelva fácilmente el problema lineal (3.7), y que sea capaz de calcular $x_k = x_0 + Q_k y_k$ sin tener que referirse de forma explícita a $q_1, \dots q_k$ como sugiere (3.2).

Según hemos visto, utilizando el algoritmo de Lanczos, el sistema (3.4) se transforma en (3.7), un sistema lineal cuya matriz de coeficientes es simétrica, tridiagonal, y definida positiva. Una forma de resolver estos sistemas lineales de forma eficiente es mediande la factorización LDL^T. Recordemos que toda matriz simétrica definida positiva admite una factorización LDL^T.

Definición 3.1 (Factorización LDT$^{\mathbf{T}}$). *Dada una matriz simétrica y definida positiva T, su factorización LDL^T es $T = LDL^T$, donde L es una matriz triangular inferior con 1 en todas sus entradas diagonales, y D una matriz diagonal.*

Para el caso que nos ocupa, podemos calcular la factorización $L_k D_k L_k^T$ de T_k por comparación directa de los coeficientes. Supongamos que

$$L_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_1 & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \ell_{k-1} & 1 \end{bmatrix}, \qquad D_k = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & d_k \end{bmatrix}. \tag{3.8}$$

Entonces, tenemos que

$$L_k D_k L_k^T = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ \ell_1 d_1 & d_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & \ell_{k-1}d_{k-1} & d_k \end{bmatrix} L_k^T = \begin{bmatrix} d_1 & \ell_1 d_1 & & \cdots & 0 \\ \ell_1 d_1 & \ell_1^2 d_1 + d_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \ell_{k-1}d_{k-1} \\ 0 & \cdots & & \ell_{k-1}d_{k-1} & \ell_k^2 d_{k-1} + d_k \end{bmatrix}.$$

Comparando las entradas del producto matricial anterior con las de la matriz (3.6), se deduce que

$$\begin{aligned} d_1 &= \alpha_1 \\ \ell_i d_i &= \beta_i \qquad \text{para } i = 1, \ldots, k-1, \\ \ell_{i-1}^2 d_{i-1} + d_i &= \alpha_i \qquad \text{para } i = 2, \ldots, k, \end{aligned} \tag{3.9}$$

luego podemos calcular la factorización LDL^T mediante el algoritmo 5.

Algoritmo 5: Factorización LDL^T de una matriz tridiagonal $T = \text{tridiag}(\alpha, \beta, \alpha)$

Entrada: Matriz tridiagonal simétrica $T_k \in \mathbb{R}^{k\times k}$
Salida : Matrices L_k y D_k, con L_k triangular inferior con todo 1 en la diagonal, y D_k diagonal, tales que $T_k = L_k D_k L_k^T$

$d_1 = \alpha_1$
for $i = 1$ **to** k **do**
  $\ell_{i-1} = \beta_{i-1}/d_{i-1}$
  $d_i = \alpha_i - \beta_{i-1}\ell_{i-1}$
end

Notemos que, al ir calculando las sucesivas matrices T_k, no tenemos que iterar el algoritmo anterior completamente para calcular cada una de las factorizaciones LDL^T, sino que únicamente necesitamos calcular

$$\begin{aligned} \ell_{k-1} &= \beta_{k-1}/d_{k-1}, \\ d_k &= \alpha_k - \beta_{k-1}\ell_{k-1}, \end{aligned} \tag{3.10}$$

para obtener L_k y D_k a partir de L_{k-1} y D_{k-1}.

Como ya se ha mencionado, además de querer resolver el sistema (3.7) de manera eficiente, queremos poder calcular las sucesivas aproximaciones x_k sin tener que referir de forma explícita a los $q_1, \ldots, q_k$. Con este fin, podemos definir $C_k \in \mathbb{R}^{n\times k}$ y $p_k \in \mathbb{R}^k$ por las ecuaciones

$$\begin{aligned} C_k L_k^T &= Q_k, \\ L_k D_k p_k &= Q_k^T (b - Ax_0). \end{aligned} \tag{3.11}$$

Observemos que, tomando de nuevo $r_0 = b - Ax_0$ y sustituyendo la factorización LDL^T de T_k en (3.7), tenemos que

$$y_k = (L_k D_k L_k^T)^{-1} Q_k^T r_0,$$

de donde, sin más que sustituir en (3.2), deducimos que

$$\begin{aligned} x_k = x_0 + Q_k y_k &= x_0 + Q_k (L_k D_k L_k^T)^{-1} Q_k^T r_0 \\ &= x_0 + C_k L_k^T (L_k D_k L_k^T)^{-1} L_k D_k p_k \\ &= x_0 + C_k p_k. \end{aligned} \tag{3.12}$$

Denotemos por c_i, para $i = 1, \dots k$, a las columnas de C_k, de forma que $C_k = [c_1 \mid \dots \mid c_k]$. De la primera expresión de (3.11), se sigue que

$$[c_1 \mid \ell_1 c_1 + c_2 \mid \dots \mid \ell_{k-1} c_{k-1} + c_k] = [q_1 \mid \dots \mid q_k],$$

luego $C_k = [C_{k-1} \mid c_k]$ donde

$$c_k = q_k - \ell_{k-1} c_{k-1}. \tag{3.13}$$

Teniendo en cuenta la estructura de L_k y D_k, tenemos la siguiente igualdad

$$L_k D_k = \left[\begin{array}{ccc:c} & L_{k-1} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hdashline 0 & \dots & \ell_{k-1} & 1 \end{array}\right] \left[\begin{array}{ccc:c} & D_{k-1} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hdashline 0 & \dots & 0 & d_k \end{array}\right] = \left[\begin{array}{ccc:c} & L_{k-1}D_{k-1} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hdashline 0 & \dots & \ell_{k-1}d_{k-1} & d_k \end{array}\right].$$

Además, si ponemos $p_k = [\rho_1, \dots, \rho_k]^T$, entonces, la segunda de las ecuaciones de (3.11) nos dice que

$$\left[\begin{array}{ccc:c} & L_{k-1}D_{k-1} & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hdashline 0 & \dots & \ell_{k-1}d_{k-1} & d_k \end{array}\right] \left[\begin{array}{c} \rho_1 \\ \vdots \\ \hdashline \rho_{k-1} \\ \hdashline \rho_k \end{array}\right] = \left[\begin{array}{c} q_1^T r_0 \\ \vdots \\ \hdashline q_{k-1}^T r_0 \\ \hdashline q_k^T r_0 \end{array}\right]. \tag{3.14}$$

Puesto que $L_{k-1} D_{k-1} p_{k-1} = Q_{k-1}^T r_0$, se sigue que

$$p_k = \begin{bmatrix} p_{k-1} \\ \rho_k \end{bmatrix}, \qquad \text{donde} \qquad \rho_k = (q_k^T r_0 - \ell_{k-1} d_{k-1} \rho_{k-1})/d_k, \tag{3.15}$$

y, sustituyendo en (3.12), se deduce que

$$x_k = x_0 + C_k p_k = x_0 + C_{k-1} p_{k-1} + \rho_k c_k = x_{k-1} + \rho_k c_k. \tag{3.16}$$

Este es el tipo de recursión que necesitamos para x_k, pues las ecuaciones (3.10), (3.13) y (3.15) nos permiten calcular $(\ell_{k-1}, d_k, q_k, c_k, x_k)$ a partir de $(\ell_{k-2}, d_k, q_{k-1}, c_{k-1}, x_{k-1})$ mediante una serie de pocos productos, sumas y divisiones.

Por otra parte, existe una simplificación adicional si en vez de considerar q_1 un vector aleatorio unitario, como propone originariamente el método de Lanczos, tomamos q_1 un vector unitario en la dirección del vector residual inicial $r_0 = b - Ax_0$ como propone el algoritmo 5. . Con esta elección del vector inicial de Lanczos, tenemos que $q_k^T r_0 = 0$ para $k \geq 2$. Además, utilizando (3.2), (3.5) y (3.7) en la primera, tercera y quinta igualdad respectivamente, se tiene que

$$\begin{aligned} b - Ax_k &= b - A(x_0 + Q_k y_k) = (b - Ax_0) - (AQ_k)y_k \\ &= r_0 - (Q_k T_k + r_k e_k^T)y_k = r_0 - Q_k(T_k y_k) - r_k e_k^T y_k \\ &= r_0 - Q_k Q_k^T r_0 - r_k e_k^T y_k = -r_k e_k^T y_k. \end{aligned} \tag{3.17}$$

Por tanto, si $\beta_k = \|r_k\|_2 = 0$ en alguna iteración de Lanczos, entonces $Ax_k = b$. Además, $\|Ax_k - b\|_2 = \beta_k \left| e_k^T y_k \right|$, de forma que podemos calcular la calidad de nuestra aproximación x_k como solución del problema $Ax = b$.

Con todo ello, tenemos el algoritmo 6.

Algoritmo 6: Resolución de sistemas lineales mediante el algoritmo de Lanczos. Versión 2

Entrada: Matriz $A \in \mathbb{R}^{n \times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$, $\beta_0 = \|r_0\|_2$, $q_0 = 0$, y $k = 0$

while $\beta_k \neq 0$ **do**

- $q_{k+1} = r_k / \beta_k$
- $k = k + 1$
- $\alpha_k = q_k^T A q_k$
- $r_k = (A - \alpha_k I_k) q_k - \beta_{k-1} q_{k-1}$
- $\beta_k = \|r_k\|_2$
- **if** $k = 1$ **then**
  - $d_1 = \alpha_1$
  - $c_1 = q_1$
  - $\rho_1 = \beta_0 / \alpha_1$
  - $x_1 = \rho_1 q_1$
- **else**
  - $\ell_{k-1} = \beta_{k-1} / d_k$
  - $d_k = \alpha_k - \beta_{k-1} \ell_{k-1}$
  - $c_k = q_k - \ell_{k-1} c_{k-1}$
  - $\rho_k = -\ell_{k-1} d_{k-1} \rho_{k-1} / d_k$
  - $x_k = x_{k-1} + \rho_k c_k$
- **end**

end

$x = x_k$

El algoritmo anterior no se suele ejecutar hasta que $\beta_k = 0$, pues en la práctica,

debido a errores numéricos y de redondeo, rara vez ocurre $\beta_k = 0$. Lo que se suele hacer es, o bien indicar un número máximo de iteraciones m, o bien indicar una tolerancia, de forma que el bucle se ejecuta hasta que $\|Ax_k - b\|_\infty < tol$.

En los siguientes ejemplos se pretende aplicar el algoritmo anterior a sistemas lineales cuya solución exacta conozcamos a priori, y podamos comparar las aproximaciones numéricas que produce el algoritmo anteriormente expuesto en las sucesivas iteraciones. Más adelante, en (3.70), se verá una cota del error para este método, luego revisaremos estos mismos ejemplos.

Ejemplo 3.1. *Supongamos que queremos resolver el sistema lineal*

$$\begin{bmatrix} 1 & & & & \\ & 2 & & & \\ & & \ddots & & \\ & & & 39 & \\ & & & & 40 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{39} \\ x_{40} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}. \tag{3.18}$$

Denotemos al sistema anterior por $Ax = b$. Evidentemente, la solución exacta viene dada por

$$x = \left[1, \frac{1}{2}, \frac{1}{3}, \quad \dots \quad , \frac{1}{38}, \frac{1}{39}, \frac{1}{40}\right]^T. \tag{3.19}$$

Utilicemos como aproximación inicial el vector

$$x_0^T = \left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \quad \dots \quad , \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right]^T, \tag{3.20}$$

y ejecutemos el algoritmo un total de $m = 15$ iteraciones. Es evidente que el vector (3.20) es un mal aproximante inicial, pero aun así, el algoritmo terminará por producir buenas aproximaciones.

En la figura 3.1 se ha representado para cada una de las $m = 15$ iteraciones las sucesivas aproximaciones a la solución del sistema, frente a la solución exacta dada por (3.19). De cara a estudiar la calidad de la aproximación, podemos representar $\|Ax_k - b\|_\infty$, donde x_k representa el aproximante a la solución en la k-ésima iteración. Haciendo esto, obtenemos la figura 3.2.

El error cometido es relativamente alto al principio de las iteraciones. Esto sin duda se debe a que hemos utilizado una mala aproximación inicial a la solución del sistema. En la siguiente tabla, se recoge el error máximo cometido en función del número de iteraciones. Cabe notar que, para $m = 40$, es decir, el número máximo de iteraciones que podemos realizar, obtenemos una solución exacta hasta precisión de la máquina.

Iteración	*Error*	*Iteración*	*Error*	*Iteración*	*Error*
$m = 0$	0.5000	$m = 15$	0.0075	$m = 30$	$5.5564 \cdot 10^{-8}$
$m = 5$	0.4335	$m = 20$	$3.0941 \cdot 10^{-4}$	$m = 35$	$1.0051 \cdot 10^{-10}$
$m = 10$	0.1407	$m = 25$	$6.9134 \cdot 10^{-6}$	$m = 40$	$6.6613 \cdot 10^{-16}$

Tabla 3.1: Error máximo cometido en cada iteración

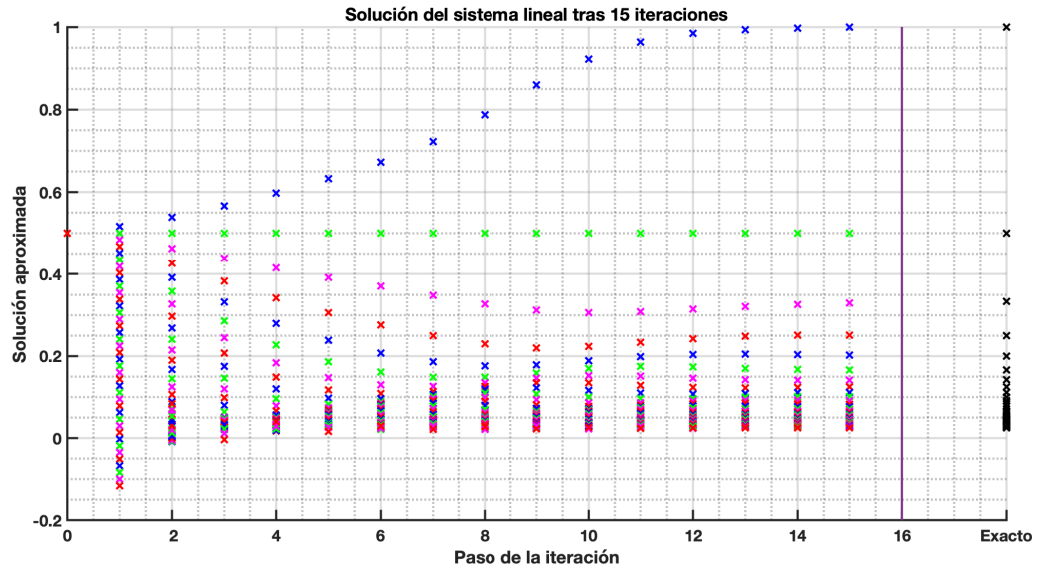

Figura 3.1: Soluciones aproximadas del sistema $Ax = b$ para $m = 15$ iteraciones

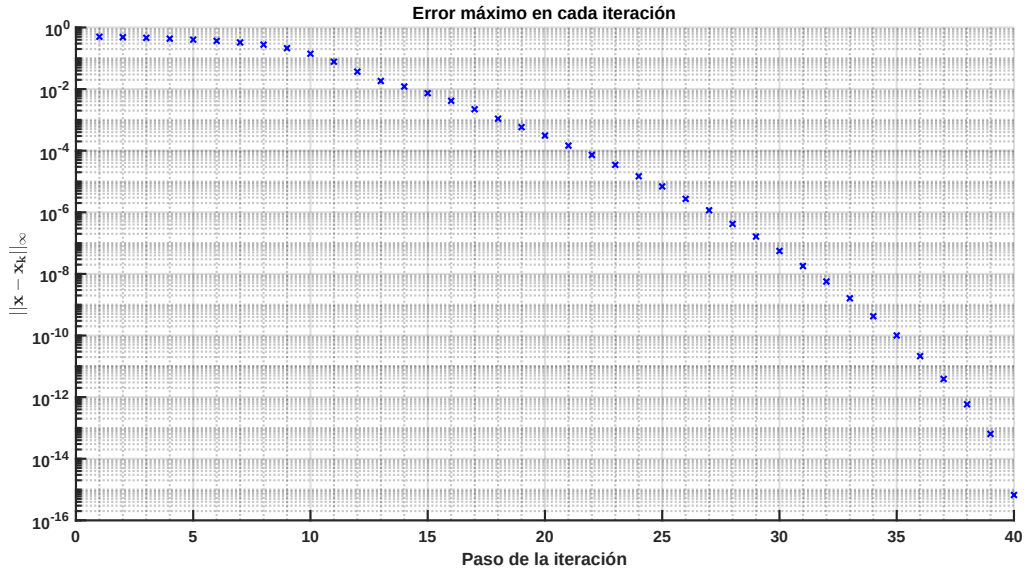

Figura 3.2: Error máximo en la solución aproximada del sistema (6).

Veamos otro ejemplo donde la matriz de coeficientes del sistema es dispersa y simétrica, pero no necesariamente diagonal.

Ejemplo 3.2. *Para no tener que escribir de manera explícita un sistema de gran tamaño, generaremos uno aleatorio con Matlab, indicando la semilla que se ha utilizado, de forma que se pueda consultar el sistema y el iterante inicial en todo momento. Consideremos el sistema $Ax = b$, utilizando como iterante inicial x_0, generado a través de los comandos*

```
n=40;
rng(2,'philox');      A=sprandsym(n,0.3,0.04,1);      b=3*rand(n,1);
rng(3,'philox');      x0=2*rand(n,1);
```

En la figura 3.3 se han representado las sucesivas aproximaciones a la solución exacta (representadas en negro) del problema propuesto. Observamos que, a pesar de que el iterante inicial es de muy mala calidad, el método converge con gran velocidad a partir de un cierto número de iteración.

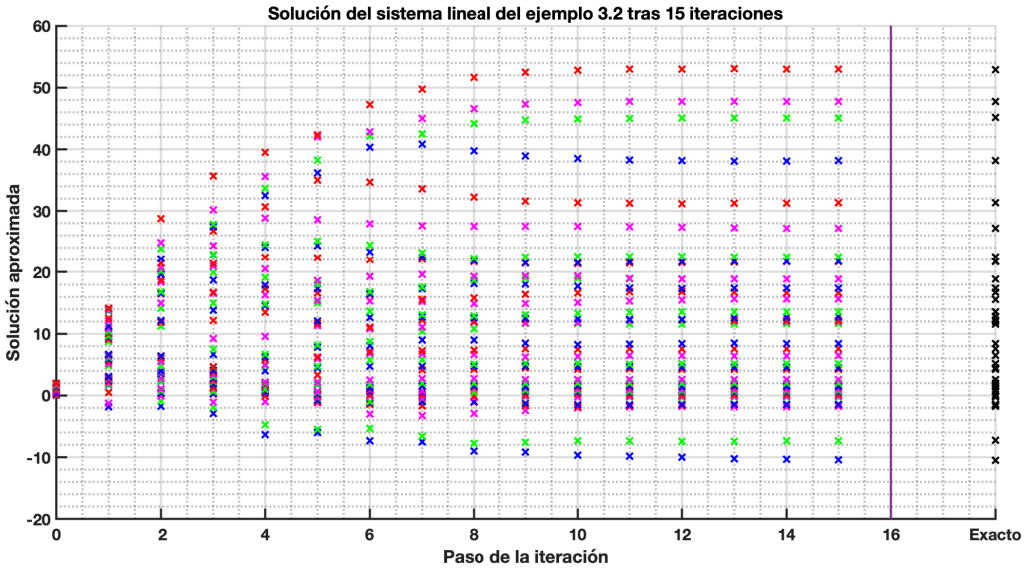

Figura 3.3: Soluciones aproximadas por el método de Lanczos del sistema lineal 3.2 para $m = 15$ iteraciones

Si representamos ahora el error máximo en cada iteración, utilizando como antes la norma infinito, obtenemos la figura 3.4, donde se aprecia la convergencia del método.

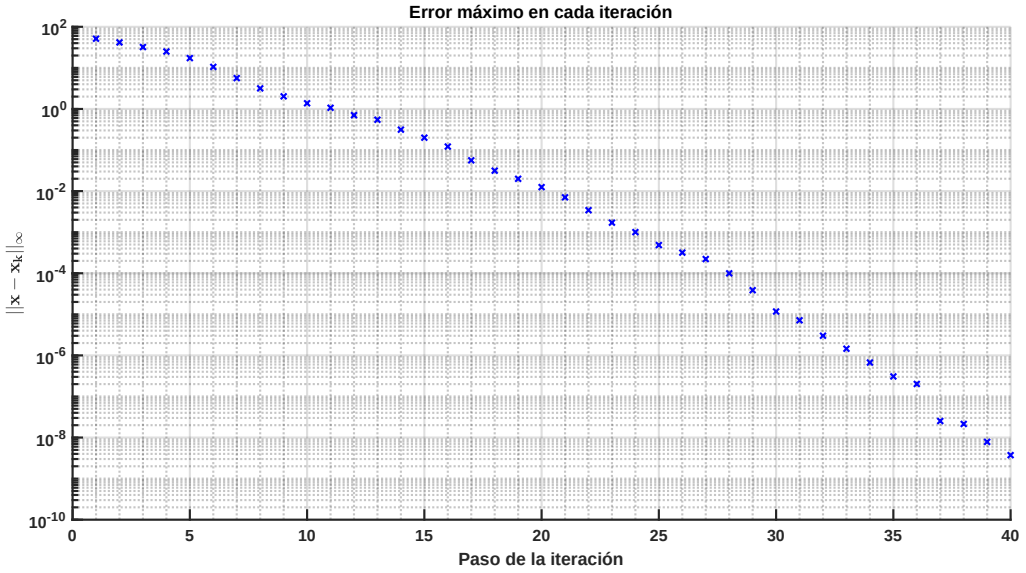

Figura 3.4: Error máximo en la solución aproximada del Ejemplo 3.2.

3.2. Método del gradiente conjugado

En la sección anterior se ha desarrollado un procedimiento con el cual podemos emplear el método de Lanczos para resolver sistemas lineales de gran tamaño cuya matriz de coeficientes sea dispersa y definida positiva. Se ha mencionado que este procedimiento es equivalente a aplicar el método del gradiente conjugado a este mismo sistema lineal. El objetivo de esta sección es derivar el *método del gradiente conjugado* para poder, posteriormente, identificarlo con el método de Lanczos. Para ello, necesitaremos derivar dos algoritmos previos, el *algoritmo del máximo descenso* y el *algoritmo de las direcciones conjugadas*.

En esta sección, derivaremos dichos métodos y, una vez obtenido el conocido método del gradiente conjugado, demostraremos que es equivalente a aplicar el método de Lanczos para resolver sistemas lineales dispersos y definidos positivos.

3.2.1. Método del máximo descenso

Supongamos que, como en la sección anterior, queremos resolver el sistema de ecuaciones lineales $Ax = b$, donde $A \in \mathbb{R}^{n\times n}$ es una matriz simétrica y definida positiva y $b \in \mathbb{R}^n$. Consideremos de nuevo el operador ϕ de (3.1)

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b.$$

Según vimos en la sección 3.1, el único mínizador de $\phi(x)$ es $x = A^{-1}b$, luego su valor mínimo es $b^T A^{-1}b/2$.

En resumen, resolver el sistema $Ax = b$ es equivalente a minimizar ϕ si A es simétrica y definida positiva. En la sección anterior, no hicimos más que desarrollar un método que permite minimizar el operador ϕ. De igual manera, el *método del máximo descenso* surge de otro planteamiento por el cual se puede minimizar ϕ. En el punto x_k, el operador ϕ decrece más rápidamente en la dirección del gradiente negativo $-\nabla\phi(x_k) = b - Ax_k$. Definamos por r_k al vector

$$r_k = b - Ax_k, \tag{3.21}$$

denotado por *vector de residuos en* x_k. Si este vector de residuos es no nulo, de cara a buscar el mínimo de ϕ, buscamos un nuevo iterante en la dirección definida por el gradiente negativo, es decir, consideramos $x_{k+1} = x_k + \alpha r_k$, donde α es una constante que satisface

$$\min_{\alpha\in\mathbb{R}} \phi\left(x_k + \alpha r_k\right). \tag{3.22}$$

Para definir α, podríamos considerar el gradiente de $\phi\left(x_k + \alpha r_k\right)$ respecto de α e

igualarlo a cero. Sin embargo, notemos que

$$\begin{aligned}\phi\left(x_k+\alpha r_k\right) &= \frac{1}{2}\left(x_k^T+\alpha r_k^T\right)A\left(x_k+\alpha r_k\right)-\left(x_k^Tb+\alpha r_k^Tb\right)\\ &= \frac{1}{2}x_k^TAx_k+\frac{\alpha}{2}x_k^TAr_k+\frac{\alpha}{2}r_k^TAx_k+\frac{\alpha^2}{2}r_k^TAr_k-x_k^Tb-\alpha r_k^Tb\\ &= \left(\frac{r_k^TAr_k}{2}\right)\alpha^2+\left(r_k^T\left(Ax_k-b\right)\right)\alpha+\frac{1}{2}x_k^TAx_k-x_k^Tb\\ &= \left(\frac{r_k^TAr_k}{2}\right)\alpha^2-r_k^Tr_k\alpha+\phi(x_k),\end{aligned} \tag{3.23}$$

luego visto como función de α, tenemos un polinomio de segundo grado en α. Además, puesto que A es por hipótesis definida positiva, $r_k^TAr_k/2 > 0$, por lo que sabemos que alcanza su mínimo precisamente en

$$\alpha_k = \frac{r_k^Tr_k}{r_k^TAr_k}. \tag{3.24}$$

Con todo esto, se puee generar el algoritmo 7. En la práctica, como de costumbre, suele pedirse o bien un número máximo de iteraciones, o bien que la diferencia entre dos iterantes sucesivos sea menor que cierta tolerancia.

Algoritmo 7: Resolución de sistemas lineales mediante el método del máximo descenso

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$

while $r_k \neq 0$ **do**

  $k = k + 1$

  $\alpha_k = r_{k-1}^Tr_{k-1}/r_{k-1}^TAr_{k-1}$

  $x_k = x_{k-1} + \alpha_k r_{k-1}$

  $r_k = b - Ax_k$

end

El objetivo de esta sección no es realizar un estudio detallado del método del máximo descenso. Sin embargo, de cara a exponer por qué no es un método siempre conveniente, y poder así realizar las modificaciones necesarias que conducen al método del gradiente conjugado, es necesario realizar un análisis del error global del método. Para ello, nos es conveniente definir la siguiente norma matricial.

Definición 3.2 (**A-norma**). *Dada una matriz $A \in \mathbb{R}^{n\times n}$ definida positiva, definimos la A-norma, o norma asociada a la matriz A, por*

$$\|v\|_A = \sqrt{v^TAv} \qquad \text{para } v \in \mathbb{R}^n. \tag{3.25}$$

Se puede probar fácilmente que (3.25) define una norma. Denotemos por x_* a la solución exacta del sistema $Ax = b$. Notemos que

$$\begin{aligned}\phi(x_k) &= \frac{1}{2}x_k^T A x_k - x_k^T b = \frac{1}{2}x_k^T A x_k - \frac{1}{2}x_k^T b - \frac{1}{2}b^T x_k \\ &= \frac{1}{2}x_k^T A x_k - \frac{1}{2}x_k^T \underbrace{A x_*}_{b} - \frac{1}{2}\underbrace{x_*^T A}_{b^T} x_k + \frac{1}{2}\underbrace{x_*^T A}_{b^T} x_* - \frac{1}{2}b^T x_* \\ &= \frac{1}{2}\left(x_k - x_*\right)^T A \left(x_k - x_*\right) - \frac{1}{2}b^T \underbrace{A^{-1}b}_{x_*},\end{aligned} \tag{3.26}$$

y que $\phi(x_*) = -b^T A^{-1} b/2$, luego podemos escribir $\phi(x_k)$ en función de la A-norma definida en la definición 3.2 como

$$\phi(x_k) = \frac{1}{2}\|x_k - x_*\|_A^2 + \phi(x_*). \tag{3.27}$$

Notemos ahora que, sustituyendo α_k definido como en (3.24) en (3.23), se sigue que

$$\phi(x_{k+1}) = \phi(x_k) - \frac{1}{2}\frac{(r_k^T r_k)^2}{r_k^T A r_k}. \tag{3.28}$$

De cara a estudiar la convergencia global del método, definamos

$$\kappa_k = \frac{r_k^T A r_k}{r_k^T r_k} \cdot \frac{r_k^T A^{-1} r_k}{r_k^T r_k}, \tag{3.29}$$

de forma que, introduciendo (3.29) en (3.28), tenemos que

$$\phi(x_{k+1}) = \phi(x_k) - \frac{1}{2}\frac{1}{\kappa_k} r_k^T A^{-1} r_k. \tag{3.30}$$

Por otra parte, notemos que

$$r_k^T A^{-1} r_k = (b^T - x_k^T A)(A^{-1}b - x_k) = b^T A^{-1} b - b^T x_k - x_k^T b + x_k^T A x_k = 2\phi(x_k) + b^T A^{-1} b,$$

luego sustituyendo esta expresión en (3.28), deducimos que

$$\phi(x_{k+1}) = \phi(x_k) - \frac{1}{\kappa_k}\left(\phi(x_k) + \frac{1}{2}b^T A^{-1} b\right). \tag{3.31}$$

Denotemos, al igual que hicimos en el capítulo anterior, por $\lambda_1(A)$ y $\lambda_n(A)$ al mayor y menor autovalor de A respectivamente, de forma que se tiene

$$\kappa_k = \frac{r_k^T A r_k}{r_k^T r_k} \cdot \frac{r_k^T A^{-1} r_k}{r_k^T r_k} \leq \frac{\lambda_1(A)}{\lambda_n(A)} = \kappa_2(A), \tag{3.32}$$

donde $\kappa_2(A)$ denota el número de condición euclíde0 de la matriz A.

Restando ahora $\phi(x_*) = -(b^T A^{-1} b)/2$ de ambos lados de la igualdad (3.31), vemos que

$$\phi(x_{k+1}) - \phi(x_*) = \left(1 - \frac{1}{\kappa_k}\right)\Big[\phi(x_k) - \phi(x_*)\Big], \tag{3.33}$$

de donde, sin más que utilizar (3.27) y (3.32), deducimos que

$$\|x_{k+1} - x_*\|_A^2 = \left(1 - \frac{1}{\kappa_k}\right) \|x_k - x_*\|_A^2 \leq \left(1 - \frac{1}{\kappa_2(A)}\right) \|x_k - x_*\|_A^2. \tag{3.34}$$

Se sigue por inducción que este método es globalmente convergente. Sin embargo, la velocidad de convergencia viene dictada, en esencia, por $(1 - 1/\kappa_2(A))^{k/2}$, luego salvo que $\kappa_2(A) = \lambda_1(A)/\lambda_n(A)$ sea cercano a 1, la convergencia es en general lenta. Claramente, esto supone un problema para la convergencia del método.

Tratemos de resolver el ejemplo 3.1 mediante este nuevo algoritmo.

Ejemplo 3.3. *Consideremos el sistema lineal descrito en el ejemplo 3.1, y apliquemos el algoritmo del máximo descenso. En la figura 3.5 se han representado las sucesivas iteraciones, si utilizamos como aproximación inicial el vector x_0^T descrito en ese mismo ejemplo. Notemos que, a diferencia de lo que ocurría para el algoritmo*

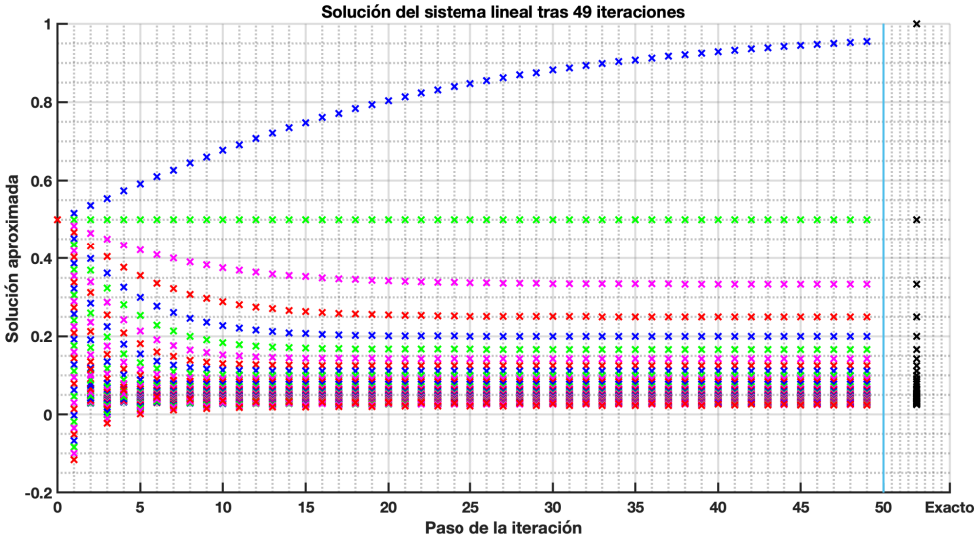

Figura 3.5: Soluciones aproximadas del sistema $Ax = b$ para $m = 49$ iteraciones, con el método del máximo descenso

6, en este caso no tenemos garantizada la convergencia tras un total de m iteraciones, siendo m el tamaño de la matriz de coeficientes del sistema lineal que queremos resolver.

Si representamos de nuevo el error $\|Ax_k - b\|_\infty$, donde x_k representa el aproximante a la solución en la k-ésima iteración, obtenemos la figura 3.6.

De cara a poder comparar con el algoritmo de resolución lineal basado en el método de Lanczos, es interesante recoger estos errores, de forma que puedan ser comparados con los de la tabla 3.1.

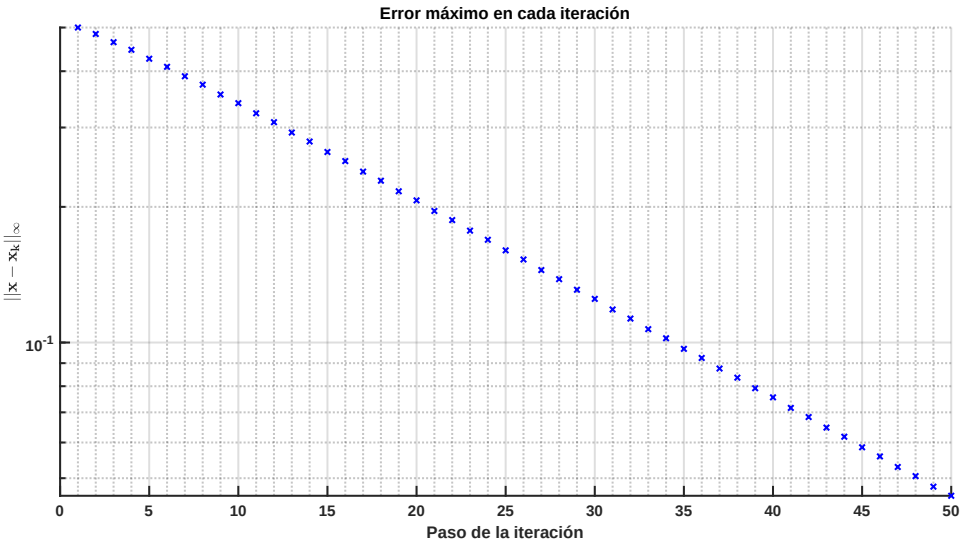

Figura 3.6: Error máximo en la solución aproximada del sistema $Ax = b$ para $m = 49$ iteraciones. Método del máximo descenso

Iteración	*Error*	*Iteración*	*Error*	*Iteración*	*Error*
$m = 0$	0.5000	$m = 20$	0.2067	$m = 40$	0.0716
$m = 5$	0.4263	$m = 25$	0.1602	$m = 45$	0.0586
$m = 10$	0.3397	$m = 30$	0.1250	$m = 47$	0.0529
$m = 15$	0.2647	$m = 35$	0.0968	$m = 49$	0.0457

Tabla 3.2: Error máximo cometido en cada iteración

Podemos observar que, este método produce mediante el mismo número de iteraciones, aproximaciones de peor calidad que las que obtuvimos al resolver este mismo sistema mediante el algoritmo 6, si bien es cierto que este último emplea en cada iteración un mayor número de operaciones.

3.2.2. Direcciones de búsqueda generales

Según hemos visto, aunque el método del máximo descenso es globalmente convergente, esta convergencia puede ser en ocasiones realmente lenta. De cara a evitar este problema, podemos considerar la sucesiva minimización de ϕ a lo largo del conjunto de direcciones $\{p_1, p_2, \dots\}$, de forma que estas no sean necesariamente $\{r_1, r_2, \dots\}$.

En este caso, consideraremos iterantes de la forma $x_k = x_{k-1} + \alpha p_k$, siendo α una constante. Para minimizar $\phi(x_{k-1} + \alpha p_k)$, podemos razonar de manera análoga

a como hicimos en (3.23), es decir, puesto que

$$\begin{aligned}\phi\left(x_{k-1}+\alpha p_k\right) &= \frac{1}{2}\left(x_{k-1}^T+\alpha p_k^T\right) A\left(x_{k-1}+\alpha p_k\right) - \left(x_{k-1}^T b+\alpha p_k^T b\right)\\ &= \frac{1}{2}x_{k-1}^T A x_{k-1} + \frac{\alpha}{2}x_{k-1}^T A p_k + \frac{\alpha}{2}p_k^T A x_{k-1} + \frac{\alpha^2}{2}p_k^T A p_k - x_{k-1}^T b - \alpha p_k^T b\\ &= \left(\frac{p_k^T A p_k}{2}\right)\alpha^2 + \left(p_k^T\left(A x_{k-1}-b\right)\right)\alpha + \frac{1}{2}x_{k-1}^T A x_{k-1} - x_{k-1}^T b\\ &= \left(\frac{p_k^T A p_k}{2}\right)\alpha^2 - p_k^T r_{k-1}\,\alpha + \phi(x_{k-1}),\end{aligned} \tag{3.35}$$

tenemos que $\phi\left(x_{k-1}+\alpha p_k\right)$ se minimiza tomando

$$\alpha_k = \frac{p_k^T r_{k-1}}{p_k^T A p_k}. \tag{3.36}$$

Sustituyendo α_k en $\phi(x_k)$, vemos que

$$\phi(x_k) = \phi\left(x_{k-1}+\alpha p_k\right) = \phi(x_{k-1}) - \frac{1}{2}\frac{(p_k^T r_{k-1})^2}{p_k^T A p_k}. \tag{3.37}$$

Notemos que, para asegurar que ϕ es minimizado en la k-ésima iteración, debemos asegurar que p_k no sea ortogonal a r_{k-1}. Esto nos lleva al siguiente planteamiento para un posible algoritmo que evite los problemas del método del máximo descenso.

Algoritmo 8: Resolución de sistemas lineales mediante búsqueda en direcciones generales

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$

while $r_k \neq 0$ **do**

  $k = k + 1$

  Elegir una dirección p_k tal que $p_k^T r_{k-1} \neq 0$.

  $\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$

  $x_k = x_{k-1} + \alpha_k p_k$

  $r_k = b - Ax_k$

end

Los iterantes x_k generados a partir del nuevo algoritmo 8 verifican que

$$x_k \in x_0 + \operatorname{span}\{p_1,\ldots,p_k\} \equiv \{x_0 + a_1 p_1 + \ldots a_k p_k \ : a_i \in \mathbb{R}\}. \tag{3.38}$$

Si las sucesivas direcciones de búsqueda p_k son linealmente independientes , puesto que los iterantes x_k se generan de forma que se cumple que

$$x_k = \min_{x\in x_0+\operatorname{span}\{p_1,\ldots,p_k\}} \phi\left(x\right), \qquad \text{para } k = 1, 2, \ldots \tag{3.39}$$

entonces la convergencia está garantizada a lo sumo en n iteraciones, pues x_n minimiza ϕ en $x_0 + \text{span}\{p_1, \ldots, p_n\} \equiv \mathbb{R}^n$, y por tanto satisface que $Ax_n = b$.

3.2.3. Búsqueda en direcciones conjugadas de A

Para que el procedimiento de búsqueda en direcciones generales sea viable de forma práctica, nos gustaría que el método, al igual que el algoritmo 6, tenga la propiedad de que x_k pueda ser fácilmente calculado a partir de x_{k-1}. Veamos qué implicaciones puede tener dicha propiedad sobre la elección de los vectores p_k. Sin más que escribir el iterante x_k como

$$x_k = x_0 + P_{k-1}y + \alpha p_k \tag{3.40}$$

donde $\mathcal{P}_{k-1} = [p_1, \ldots, p_{k-1}]$, $y \in \mathbb{R}^{k-1}$ y $\alpha \in \mathbb{R}$, tenemos que

$$\begin{aligned}\phi(x_k) &= \frac{1}{2}\left(x_0^T + y^T\mathcal{P}_{k-1}^T + \alpha p_k^T\right)\left(Ax_0 + A\mathcal{P}_{k-1}y + \alpha Ap_k\right) - \left(x_0^T + y^T\mathcal{P}_{k-1}^T + \alpha p_k^T\right)b \\ &= \phi(x_0 + \mathcal{P}_{k-1}y) - \alpha p_k^T\left(b - Ax_0\right) + \alpha y^T\mathcal{P}_{k-1}^T Ap_k + \frac{\alpha^2}{2}p_k^T Ap_k \\ &= \phi(x_0 + \mathcal{P}_{k-1}y) - \alpha p_k^T r_0 + \alpha y^T\mathcal{P}_{k-1}^T Ap_k + \frac{\alpha^2}{2}p_k^T Ap_k.\end{aligned} \tag{3.41}$$

Notemos que si

$$p_k \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^{\perp}, \tag{3.42}$$

entonces $\alpha y^T\mathcal{P}_{k-1}^T Ap_k = 0$, y esto simplifica notablemente la expresión anterior, pues

$$\begin{aligned}\min_{x_k \in x_0 + \text{span}\{p_1,\ldots,p_k\}} \phi\left(x_k\right) &= \min_{y,\alpha} \phi\left(x_0 + \mathcal{P}_{k-1}y + \alpha p_k\right) \\ &= \min_{y,\alpha}\left(\phi\left(x_0 + \mathcal{P}_{k-1}y\right) + \frac{\alpha^2}{2}p_k^T Ap_k - \alpha p_k^T r_0\right) \\ &= \min_{y}\phi\left(x_0 + \mathcal{P}_{k-1}y\right) + \min_{\alpha}\left(\frac{\alpha^2}{2}p_k^T Ap_k - \alpha p_k^T r_0\right),\end{aligned} \tag{3.43}$$

es decir, minimizar $\phi(x_k)$ equivale a realizar dos minimizaciones independientes, una para y y otra para α.

Es evidente que si y_{k-1} resuelve el primero de los problemas, entonces $x_{k-1} = x_0 + \mathcal{P}_{k-1}y_{k-1}$ minimiza ϕ sobre $x_0 + \text{span}\{p_1, \ldots, p_{k-1}\}$. Por otra parte, la solución del segundo de los problemas, un problema de minimización de un polinomio de segundo grado respecto de α con coeficiente principal positivo, viene dada por $\alpha_k = p_k^T r_0 / p_k^T Ap_k$. Además, en vistas de (3.42), se tiene que

$$\begin{aligned}p_k^T r_{k-1} &= p_k^T(b - Ax_{k-1}) \\ &= p_k^T(b - A(x_0 + \mathcal{P}_{k-1}y_{k-1})) = p_k^T r_0.\end{aligned} \tag{3.44}$$

Si tomamos las direcciones generales p_k del algoritmo 8 de forma que verifiquen (3.42), tenemos que este algoritmo se transforma en el *algoritmo de búsqueda mediante direcciones conjugadas de A.*

Algoritmo 9: Resolución de sistemas lineales mediante búsqueda en direcciones conjugadas de A

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$

while $r_k \neq 0$ **do**

$k = k + 1$

Elegir una dirección $p_k \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$ tal que $p_k^T r_{k-1} \neq 0$.

$\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$

$x_k = x_{k-1} + \alpha_k p_k$

$r_k = b - Ax_k$

end

El desarrollo del algoritmo 9 se ha basado en la existencia de sucesivos vectores p_k que satisfagan (3.42) y tales que $p_k^T r_{k-1} \neq 0$. El siguiente lema justifica que es posible encontrar direcciones de búsqueda que satisfacen dichas propiedades.

Lema 3.2.1 (Existencia de direcciones de búsqueda conjugadas de A). *Dado un vector de residuos r_{k-1} tal que $r_{k-1} \neq 0$, existe una dirección de búsqueda p_k que verifica que $p_k \in span\{Ap_1, \ldots, Ap_{k-1}\}^\perp$ y que $p_k^T r_{k-1} \neq 0$.*

Demostración:

Razonemos por inducción. Para el caso $k = 1$ la demostración del lema es trivial sin más que tomar $p_1 = r_0$.

Supongamos que el algoritmo 9 ha realizado $k > 1$ número de iteraciones, y que $r_{k-1} \neq 0$. Hasta ahora, el algoritmo ha utilizado las direcciones de búsqueda determinadas por $p_1, \ldots, p_{k-1}$, mediante las cuales ha generado los sucesivos iterantes $x_1, \ldots, x_{k-1}$ y sus respectivos vectores de residuos $r_1, \ldots, r_{k-1}$. Queremos probar que existe una nueva dirección de búsqueda no nula p_k, tal que $p_k \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$ y tal que $p_k^T r_{k-1} \neq 0$.

Puesto que $r_{k-1} = b - Ax_{k-1} \neq 0$, tenemos que $A^{-1}b \neq x_{k-1}$, y en vistas de la definición recursiva de x_{k-1} respecto de los demás iterantes x_i, es evidente que

$$A^{-1}b \notin x_0 + \text{span}\{p_1, \ldots, p_{k-1}\}, \tag{3.45}$$

luego, pasando el término x_0 al lado izquierdo y multiplicar por A por la izquierda, se sigue que

$$r_0 \equiv b - Ax_0 \notin \text{span}\{Ap_1, \ldots, Ap_{k-1}\}. \tag{3.46}$$

Por tanto, existe p_k verificando que

$$p_k \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp, \qquad p_k^T r_0 \neq 0. \tag{3.47}$$

Además, según vimos en (3.44), $p_k^T r_{k-1} = p_k^T r_0$, de donde teniendo en cuenta (3.47), confirmamos la existencia del vector p_k verificando las propiedades requeridas por el teorema. □

Se dice que las sucesivas direcciones de búsqueda p_i que utiliza este algoritmo son *A-Conjugadas* debido a que $p_i^T A p_j = 0$ para todo $i \neq j$. Notemos además que si $\mathcal{P}_k = [p_1, \ldots, p_k]$ es la matriz que tiene por columnas a estos vectores, entonces

$$\mathcal{P}_k^T A \mathcal{P}_k = \text{diag}\,(p_1^T A p_1, \ldots, p_k^T A p_k)$$

es una matriz invertible, ya que A es definida positiva, por lo que $p_i^T A p_i \neq 0$ para todo $i = 1, \ldots, k$.

Por tanto, el espacio generado por las columnas de $\mathcal{P}_k$, esto es, $ran(\mathcal{P}_k)$, tiene rango máximo. Esto es de suma importancia, pues garantiza la convergencia en a lo sumo n iteraciones, pues en caso de iterar el algoritmo n veces, tenemos que x_n minimiza $\phi(x)$ sobre $ran(\mathcal{P}_k) = \mathbb{R}^n$, luego necesariamente $x_n = x_* = A^{-1}b$, y habremos llegado a la solución exacta.

3.2.4. Elección de la mejor dirección de búsqueda

Hasta ahora, hemos desarrollado el método del máximo descenso y el método de las direcciones de búsqueda conjugadas. Una forma de combinar los aspectos positivos de ambos métodos es tomar como vector p_k del algoritmo 9 el vector más cercano a r_{k-1} que sea A-conjugado a los vectores $p_1, \ldots, p_{k-1}$. Esta es la base del *método del gradiente conjugado*, el algoritmo que se pretende detallar en esta sección y cuya versión preliminar podemos escribir como el algoritmo 10.

Algoritmo 10: Método del Gradiente Conjugado. Idea inicial

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

```
k = 0, r_0 = b − Ax_0
while r_k ≠ 0 do
    if k=0 then
        p_1 = r_0;
    else
        Tomar p_k minimizando ‖p − r_{k−1}‖_2 sobre todos los vectores
          p ∈ span{Ap_1, ..., Ap_{k−1}}^⊥;
    end
    α_k = p_k^T r_{k−1} / p_k^T A p_k
    x_k = x_{k−1} + α_k p_k
    r_k = b − Ax_k
end
```

Nota 3.1:

- Cuando elegimos el vector p_k, denominado *vector de dirección de búsqueda*, nos interesa propiamente la dirección del mismo, y no su norma. Notemos que si hubiésemos consideramos $\tilde{p}_k = \lambda_k p_k$, con $\lambda \in \mathbb{R}/\{0\}$, como dirección de búsqueda, entonces $\tilde{\alpha} = \tilde{p}_k^T r_{k-1}/\tilde{p}_k^T A\tilde{p}_k = \alpha_k/\lambda$, de donde se sigue que $\tilde{x}_k = x_{k-1} + \tilde{\alpha}_k p_k = x_{k-1} + \alpha_k p_k = x_k$. Es decir, al elegir el vector p_k, nos interesa su dirección y no su norma, pues independientemente de su norma producirá el mismo nuevo iterante x_k.

Para que este algoritmo pueda ser implementado de forma práctica, necesitamos un método eficiente para calcular los p_k. El siguiente teorema no solo muestra una propiedad interesante de los vectores p_k, sino que además, como ya veremos, establece la primera conexión con los subespacios de Krylov, a partir de los cuales derivamos el método de Lanczos.

Teorema 3.2.2. *Para $k \geq 2$, los vectores p_k generados a partir del algoritmo 10 satisfacen que*

$$p_k = r_{k-1} - A\mathcal{P}_{k-1} z_{k-1}, \tag{3.48}$$

donde $\mathcal{P}_{k-1} = [p_1, \ldots, p_{k-1}]$, y z_{k-1} resuelve el problema

$$\min_{z \in \mathbb{R}^{k-1}} \|r_{k-1} - A\mathcal{P}_{k-1} z\|_2 . \tag{3.49}$$

Demostración:
Consideremos el vector z_{k-1} que resuelve el problema de mínimos cuadrados (3.49), y p su vector residual asociado, esto es,

$$p = r_{k-1} - A\mathcal{P}_{k-1} z_{k-1}. \tag{3.50}$$

Nuestro objetivo es ver que el vector p es precisamente el vector p_k definido en el algoritmo 10.

Notemos que $p \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$, ya que $A\mathcal{P}_{k-1} z_{k-1}$ es la proyección ortogonal de r_{k-1} sobre el espacio $\text{span}\{Ap_1, \ldots Ap_{k-1}\}$, y que por tanto $p^T A\mathcal{P}_{k-1} = 0$.

Para comprobar que p minimiza $\|p - r_{k-1}\|_2$ sobre $\text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$ basta con probar que $(p - r_{k-1}) \perp \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$, pero por definición, $(p - r_{k-1}) \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$, concluyendo así la demostración. □

Con el siguiente resultado, podemos establecer relaciones importantes entre los vectores residuales r_k, las direcciones de búsqueda p_k, y los subespacios de Krylov definidos en (2.9), esto es,

$$\mathcal{K}(A, r_0, k) = span(r_0, Ar_0, \ \ldots \ , A^{k-1} r_0).$$

Teorema 3.2.3. *Después de k iteraciones del algoritmo 10, tenemos que*

$$r_k = r_{k-1} - \alpha_k A p_k, \tag{3.51}$$

$$\mathcal{P}_k^T r_k = 0, \tag{3.52}$$

$$\text{span}(p_1, \ldots, p_k) = \text{span}(r_0, \ldots, r_{k-1}) = \mathcal{K}(A, r_0, k), \tag{3.53}$$

y los vectores residuales $r_0, \ldots, r_k$ son ortogonales entre sí.

Demostración:
Supongamos que se han aplicado k iteraciones del algoritmo 10. Entonces, tenemos que $x_k = x_{k-1} + \alpha_k p_{k-1}$. Premultiplicando por A la ecuación anterior, vemos que $Ax_k = Ax_{k-1} + \alpha_k Ap_{k-1}$ y, restando b en ambos lados, concluimos que

$$(Ax_k - b) = (Ax_{k-1} - b) + \alpha_k Ap_{k-1},$$

de donde, teniendo en cuenta que los términos entre paréntesis del lado izquierdo y derecho son por definición, salvo el signo, r_k y r_{k-1} respectivamente, se deduce (3.51).

De cara a probar la expresión (3.52), recordemos que $x_k = x_0 + \mathcal{P}_k y_k$, donde y_k es el minimizador de

$$\phi(x_0 + P_k y) = \phi(x_0) + \frac{1}{2} y^T \left(P_k^T A P_k\right) y - y^T P_k^T (b - Ax_0).$$

Pero si y_k minimiza la expresión anterior, entonces es solución del sistema lineal $(\mathcal{P}_k^T A \mathcal{P}_k) y = \mathcal{P}_k^T (b - Ax_0)$, luego

$$0 = P_k^T (b - Ax_0) - P_k^T A P_k y_k = P_k^T (b - A(x_0 + P_k y_k)) = P_k^T r_k, \tag{3.54}$$

de donde se deduce (3.52).

Para probar (3.53), notemos que de (3.51) se deduce que

$$\{Ap_1, \ldots, Ap_{k-1}\} \subseteq \text{span}\{r_0, \ldots, r_{k-1}\},$$

por lo que a continuación del teorema 3.2.2 se sigue que

$$p_k = r_{k-1} - [Ap_1, \ldots, Ap_{k-1}] z_{k-1} \in \text{span}\{r_0, \ldots, r_{k-1}\}. \tag{3.55}$$

Por tanto, existe una matriz triangular superior T tal que

$$[p_1, \ldots, p_k] = [r_0, \ldots, r_{k-1}] T.$$

Además, puesto que las direcciones de búsqueda p_i son linealmente independientes, se deduce que la matriz T es no singular, de donde deducimos que

$$\text{span}\{p_1, \ldots, p_k\} = \text{span}\{r_0, \ldots, r_{k-1}\}. \tag{3.56}$$

Notemos que, utilizando (3.51), tenemos que

$$r_k \in \text{span}\{r_{k-1}, Ap_k\} \subseteq \text{span}\{r_{k-1}, Ar_0, \ldots, Ar_{k-1}\}, \tag{3.57}$$

y aplicando inducción sobre la expresión anterior, se deduce la última igualdad de (3.53).

Finalmente, para probar que los vectores residuales $r_0, \ldots, r_k$ son ortogonales entre sí, notemos que de (3.52) se deduce que r_k es ortogonal a cualquier vector en $ran(\mathcal{P}_k)$, pero de (3.53) se deduce que este subespacio contiene a los vectores $r_0, \ldots, r_{k-1}$, con lo que queda probado el teorema. □

Los resultados del teorema anterior pueden ser utilizados para demostrar que cada una de las direcciones de búsqueda p_k se puede escribir como una combinación lineal de la dirección de búsqueda anterior p_{k-1} y del residuo r_{k-1} generado en esta misma iteración.

Teorema 3.2.4. *Los residuos y las direcciones de búsqueda del algoritmo 10 tienen la propiedad de que* $p_k \in span\{p_{k-1}, r_{k-1}\}$ *para* $k \geq 2$.

Demostración:

Para $k = 2$ el resultado es evidente, pues de (3.53) se deduce que $p_2 \in \text{span}\{r_0, r_1\}$, pero $p_1 = r_0$ por construcción del algoritmo, luego tenemos que, efectivamente, p_2 es una combinación lineal de p_1 y r_1.

Si $k > 2$, consideremos la partición del vector z_{k-1} del teorema 3.2.2 dada por

$$z_{k-1} = \begin{bmatrix} w \\ \mu \end{bmatrix}, \tag{3.58}$$

donde $w \in \mathbb{R}^{k-2}$ y $\mu \in \mathbb{R}$. Por otra parte, de (3.51) se deduce que

$$-Ap_{k-1} = \frac{1}{\alpha_{k-1}} (r_{k-1} - r_{k-2}). \tag{3.59}$$

Puesto que $\mathcal{P}_{k-1} = [\mathcal{P}_{k-2}, p_k]$, sin más que introducir la descomposición (3.58) en (3.48) y sustituir la expresión (3.59), tenemos que

$$\begin{aligned} p_k &= r_{k-1} - A\mathcal{P}_{k-1} z_{k-1} = r_{k-1} - A\mathcal{P}_{k-2} w - \mu A p_{k-1} \\ &= \left(1 + \frac{\mu}{\alpha_{k-1}}\right) r_{k-1} + \left[-\frac{\mu}{\alpha_{k-1}} r_{k-2} - A\mathcal{P}_{k-2} w\right]. \end{aligned} \tag{3.60}$$

Denotemos por s_{k-1} al término entre corchetes de la expresión anterior, de forma que

$$\begin{aligned} s_{k-1} &= -\frac{\mu}{\alpha_{k-1}} r_{k-2} - A\mathcal{P}_{k-2} w \in \text{span}\{r_{k-2}, A\mathcal{P}_{k-2} w\} \\ &\subseteq \text{span}\{r_{k-2}, Ap_1, \ldots, Ap_{k-2}\} \subseteq \text{span}\{r_0, \ldots, r_{k-2}\}, \end{aligned} \tag{3.61}$$

donde para la última inclusión se ha tenido en cuenta (3.59) para los subíndices desde 1 hasta $k - 2$.

Por tanto, $s_{k-1} = \sum_{i=0}^{k-2} a_i r_i$. Puesto que, según vimos en el teorema 3.2.3, los vectores residuales r_i son ortogonales entre sí, deducimos de (3.56) y (3.51) que s_{k-1} y r_{k-1} también son ortogonales entre sí. Teniendo en cuenta (3.60) y el hecho de que ambos vectores son ortogonales, el problema de mínimos cuadrados (3.49) se traduce en encontrar μ y w que minimicen

$$\begin{aligned} \|p_k\|_2^2 &= \left(1 + \frac{\mu}{\alpha_{k-1}}\right)^2 \|r_{k-1}\|_2^2 + \|s_{k-1}\|_2^2 \\ &= \left(1 + \frac{\mu}{\alpha_{k-1}}\right)^2 \|r_{k-1}\|_2^2 + \frac{\mu^2}{\alpha_{k-1}^2} \left\| r_{k-2} - A\mathcal{P}_{k-2} \frac{\alpha_{k-1}}{\mu} w \right\|_2^2 \end{aligned} \tag{3.62}$$

una función cuadrática en μ con el coeficiente de μ^2 positivo, luego tiene un mínimo. Derivando respecto de μ, igualando a cero y simplificando, teniendo en cuenta que la segunda de las normas es fija, se tiene que

$$\frac{1}{\alpha_{k-1}} \left(1 + \frac{\mu}{\alpha_{k-1}}\right)^2 \|r_{k-1}\|_2^2 + \frac{\mu}{\alpha_{k-1}} \|\cdot\|_2^2 = 0,$$

de donde se obtiene que el mínimo se alcanza para

$$\mu^* = -\frac{\|r_{k-1}\|_2^2 \alpha_{k-1}}{\|r_{k-1}\|_2^2 + \|\cdot\|_2^2}.$$

LLevando este valor a la segunda igualdad de (3.62), tenemos que

$$\|p_k\|_2^2 = \left(1 - \frac{\|r_{k-1}\|_2^2}{\|r_{k-1}\|_2^2 + \|\cdot\|_2^2}\right)^2 \|r_{k-1}\|_2^2 + \frac{\|r_{k-1}\|_2^4}{(\|r_{k-1}\|_2^2 + \|\cdot\|_2^2)^2}\|\cdot\|_2^2,$$

y en vistas de esta expresión, se deduce que $\|p_k\|_2^2$ es mínimo cuanto más pequeño sea $\|\cdot\|_2^2$, pues notemos que si $f(x) = x/(c+x)$ con $c, x > 0$, entonces $f'(x) = c/(c+x)^2 > 0$.

Puesto que la norma-2 $\|\cdot\|_2^2$ se minimiza con $z = z_{k-2}$, de forma que $r_{k-2} - A\mathcal{P}_{k-2}z_{k-2} = p_{k-1}$, deducimos que s_{k-1} es un múltiplo de p_{k-1}, de donde finalmente se deduce que $p_k \in \text{span}\{r_{k-1}, p_{k-1}\}$, como queríamos demostrar. □

El teorema anterior nos sirve para deducir una expresión sencilla para calcular las sucesivas direcciones de búsqueda p_k. Sin pérdida de generalidad, y debido a la nota 3.1, del teorema anterior deducimos que existe $\beta_k \in \mathbb{R}$ tal que

$$p_k = r_{k-1} + \beta_k p_{k-1}. \tag{3.63}$$

Multiplicando por $p_{k-1}^T A$ por la izquierda a la expresión anterior, observamos que que

$$p_{k-1}^T A p_k = p_{k-1}^T A r_{k-1} + \beta_k p_{k-1}^T A p_{k-1},$$

y puesto que anteriormente vimos que las direcciones de búsqueda p_i son A-conjugadas, tenemos que $p_{k-1}^T A p_k = 0$, luego sin más que despejar β_k deducimos que

$$\beta_k = -\frac{p_{k-1}^T A r_{k-1}}{p_{k-1}^T A p_{k-1}}. \tag{3.64}$$

Esto nos lleva a la primera versión del método del gradiente conjugado.

Aunque este método puede ser implementado tal y como se ha descrito, podemos mejorar notablemente la eficiencia del algoritmo 11. En esta implementación del método del gradiente conjugado, empleamos en cada iteración tres multiplicaciones matriz-vector (concretamente Ar_{k-1}, Ar_k y Ax_k), luego si A es una matriz de gran tamaño, tendremos un costo computacional para resolver el sistema $Ax = b$ relativamente alto. Sin embargo, podemos mejorar notablemente la eficiencia de este algoritmo sin más que aplicar las conclusiones a las que llegamos en el teorema 3.2.3, pues de (3.51) tenemos que

$$r_{k-1} = r_{k-2} - \alpha_{k-1} A p_{k-1}, \tag{3.65}$$

luego multiplicando (3.65) por r_{k-1}^T por la izquierda, se concluye que

$$r_{k-1}^T r_{k-1} = -\alpha_{k-1} r_{k-1}^T A p_{k-1}, \tag{3.66}$$

Algoritmo 11: Método del Gradiente Conjugado. Versión 1.

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$

while $r_k \neq 0$ **do**

  $k = k + 1$,

  if *k=1* **then**

    $p_1 = r_0$;

  else

    $\beta_k = -p_{k-1}^T Ar_{k-1}/p_{k-1}^T Ap_{k-1}$

    $p_k = r_{k-1} + \beta_k p_{k-1}$

  end

  $\alpha_k = p_k^T r_{k-1}/p_k^T Ap_k$

  $x_k = x_{k-1} + \alpha_k p_k$

  $r_k = b - Ax_k$

end

y multiplicando ahora (3.65) por p_{k-1}^T por la izquierda, teniendo en cuenta (3.51) y despejando $r_{k-2}^T r_{k-2}$, se deduce que

$$p_{k-1}^T r_{k-2} = \alpha_{k-1} p_{k-1}^T Ap_{k-1}. \tag{3.67}$$

Ahora bien, puesto que $p_{k-1} = r_{k-2} + \beta_{k-2} p_{k-2}$ por definición, se comprueba que

$$p_{k-1}^T r_{k-2} = r_{k-2}^T r_{k-2} + \beta_{k-2} \underbrace{p_{k-2}^T r_{k-2}}_{0},$$

donde la llave sale de aplicar (3.51). Por tanto, sustituyendo esta última igualdad en (3.67) se concluye que

$$r_{k-2}^T r_{k-2} = \alpha_{k-1} p_{k-1}^T Ap_{k-1}. \tag{3.68}$$

Dividiendo ahora (3.66) entre (3.68), tenemos que

$$\frac{r_{k-1}^T r_{k-1}}{r_{k-2}^T r_{k-2}} = \frac{-\alpha_{k-1} r_{k-1}^T Ap_{k-1}}{\alpha_{k-1} p_{k-1}^T Ap_{k-1}} = -\frac{p_{k-1}^T Ar_{k-1}}{p_{k-1}^T Ap_{k-1}}, \tag{3.69}$$

que es precisamente la definición de β_k que dimos en (3.64). Por tanto, podemos evitar realizar dos de las tres multiplicaciones matriz-vector que aparecen en el algoritmo 11 sustituyendo la expresión de β_k por la nueva expresión (3.69).

Evidentemente, el criterio de terminación del algoritmo 12 no es realista desde el punto de vista numérico. Al igual que otras veces, suele pedirse cierto criterio de tolerancia o bien un número máximo de iteraciones.

Algoritmo 12: Método del Gradiente Conjugado. Versión 2.

Entrada: Matriz $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva, y un vector $b \in \mathbb{R}^n$, y x_0 un aproximante inicial a la solución del sistema lineal a resolver

Salida : Un vector x, solución aproximada del sistema $Ax = b$

$k = 0$, $r_0 = b - Ax_0$

while $r_k \neq 0$ **do**

  $k = k + 1$

  if *k=1* **then**

    $p_1 = r_0$;

  else

    $\beta_k = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$

    $p_k = r_{k-1} + \beta_k p_{k-1}$

  end

  $\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$

  $x_k = x_{k-1} + \alpha_k p_k$

  $r_k = b - Ax_k$

end

Respecto a las propiedades de convergencia de este método, un estudio detallado supondría demostrar diversos resultados auxiliares, por lo que únicamente se enunciará el teorema principal de convergencia que ofrece este algoritmo.

Teorema 3.2.5. *Supongamos que $A \in \mathbb{R}^{n\times n}$ es una matriz simétrica y definida positiva, que $b \in \mathbb{R}^n$, y que se quiere aproximar numéricamente la solución del sistema lineal $Ax = b$ mediante el algoritmo 12. Si $\{x_k\}$ son las sucesivas aproximaciones a la solución que devuelve este algoritmo, y $\kappa_2(A)$ denota el número de condición euclídea de la matriz A, entonces*

$$\|x - x_k\|_A \leq 2 \|x - x_0\|_A \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \tag{3.70}$$

Demostración:

Una prueba detallada de este teorema puede ser encontrada en el capítulo 2 de [4]. □

Ejemplo 3.4. *Supongamos que queremos resolver de nuevo el sistema lineal (3.18), y aplicamos para ello el algoritmo 12, utilizando como aproximación inicial x_0^T el vector (3.20).*

Si representamos las sucesivas aproximaciones a la solución, obtenemos la figura 3.7

Observamos que dicha gráfica es idéntica a la figura 3.1. Esto puede sorprendernos, pero en la siguiente sección demostraremos que dada una aproximación inicial de la solución de un sistema lineal, aplicar el método de Lanczos y el método del gradiente conjugado son equivalentes.

Puesto que, como detallaremos en la siguiente sección, estos métodos son equivalentes, la cota de error (3.70) es válida también para las soluciones obtenidas al

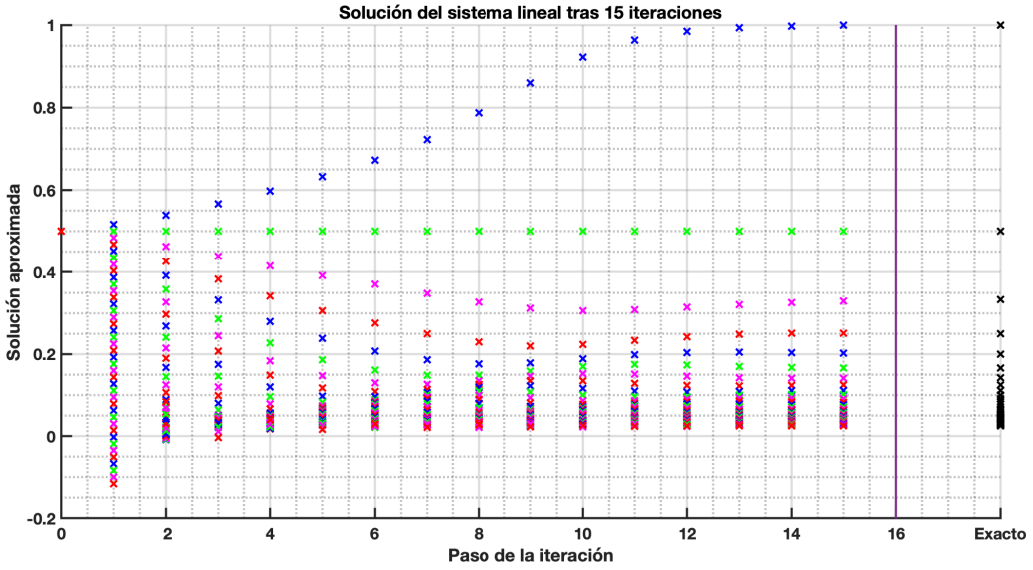

Figura 3.7: Soluciones aproximadas del sistema $Ax = b$ para m=15 iteraciones del método del gradiente conjugado.

aplicar el algoritmo de resolución de sistemas lineales de Lanczos (algoritmo 5). Revisemos los errorres cometidos al aplicar este método a los ejemplos 3.1 y 3.2.

Notemos que, tomando logaritmos en ambos lados de la expresión (3.70), se sigue que

$$\log_{10}\left(\|x - x_k\|_A\right) \leq \log_{10}\left(2\|x - x_0\|_A\right) + k \cdot \log_{10}\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right),$$

expresión a partir de la cual deducimos que el error cometido es lineal con k, y con pendiente $log_{10}\left(\sqrt{\kappa_2(A)} - 1)/(\sqrt{\kappa_2(A)} + 1)\right)$

Ejemplo 3.1. *Retomemos el ejemplo visto anteriormente. Puesto que conocemos la solución exacta, podemos representar el error dado por la expresión (3.70). En la figura 3.8, mediante cruces se ha representado el error* $\log_{10}\left(\|x - x_k\|_A\right)$*, mientras que la recta dibujada corresponde a una recta de ordenada en el origen n y pendiente m dadas por*

$$m = log_{10}\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right) = -0.1384 \qquad n = \log_{10}\left(2\|x - x_0\|_A\right) = 1.4569$$

Ejemplo 3.2. *Retomando ahora el ejemplo 3.2, y repitiendo el procedimiento anterior, tenemos la figura 3.9.*
donde ahora se tiene que

$$m = log_{10}\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right) = -0.1761 \qquad n = \log_{10}\left(2\|x - x_0\|_A\right) = 1.7927$$

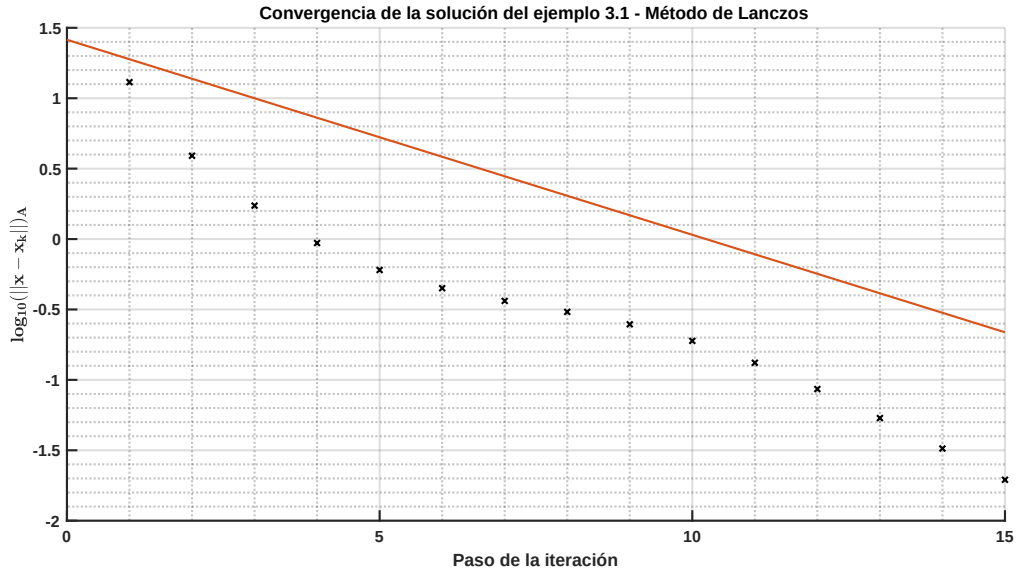

Figura 3.8: Error al resolver el ejemplo 3.1 tras m=15 iteraciones con el método de Lanczos.

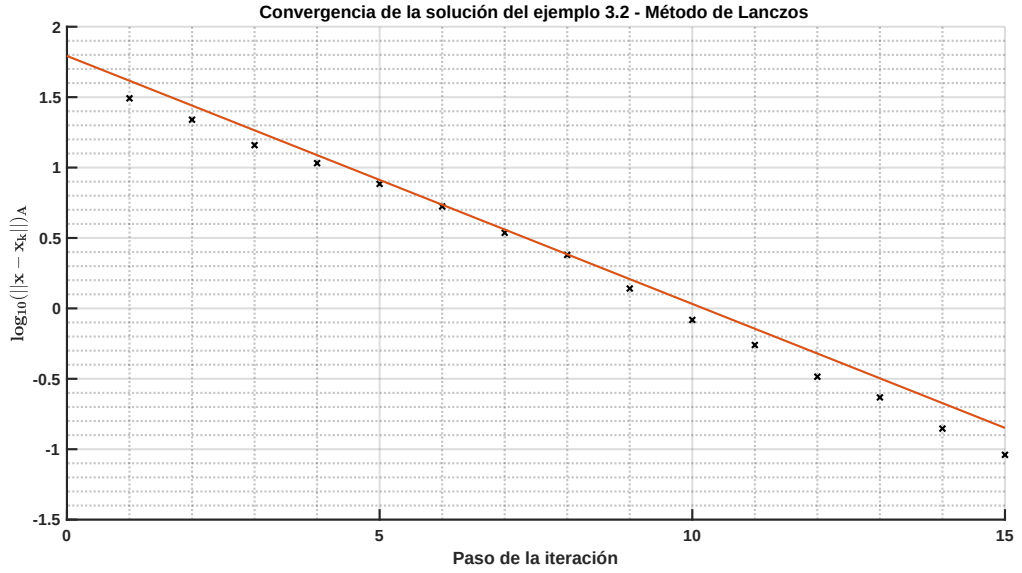

Figura 3.9: Error al resolver el ejemplo 3.2 tras m=15 iteraciones con el método de Lanczos.

De las gráficas de los errores se deduce que la solución aproximada a los ejemplos considerados converge más rápido para el algoritmo de resolución de sistemas lineales de Lanczos (o del gradiente conjugado) que para el método del máximo descenso. Hemos visto que, en esencia, la velocidad de covergencia del método de Lanczos viene dictada por

$$E_{GC} \equiv \frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1},$$

mientras que para el método del máximo descenso viene dada por

$$E_D \equiv \left(1 - \frac{1}{\kappa_2(A)}\right)^{\frac{1}{2}}.$$

Teniendo en cuenta que para el ejemplo 3.1 se tiene una matriz de coeficientes cuyo número de condición euclídea es $\kappa_2(A) = 40$, y para el ejemplo 3.2 $\kappa_2(A) = 20$, tenemos la siguiente tabla

	E_{GC}	E_{D}
Ejemplo 3.1	0.7629	0.9874
Ejemplo 3.2	0.6107	0.9798

de donde queda justificado que el error disminuye más rápidamente en cada iteración del algoritmo de Lanczos/Gradiente Conjugado, que en el algoritmo del máximo descenso.

3.3. Conexión entre el método de Lanczos y el del gradiente conjugado

En las secciones anteriores hemos derivado el conocido método del gradiente conjugado para resolver sistemas lineales de la forma $Ax = b$, donde A es una matriz simétrica definida positiva. Por otra parte, en la sección 3.1 se derivó una forma de aplicar el método de Lanczos para resolver sistemas lineales de este mismo tipo. El objetivo de esta sección es demostrar que, si bien es sorprendente, estos dos métodos son equivalentes.

Para ver la conexión entre estos métodos, realizaremos el camino inverso al realizado en la sección 3.1, es decir, partiremos del método del gradiente conjugado, que hemos deducido a lo largo de la sección 3.2, para obtener de nuevo el método de Lanczos.

Supongamos que, para resolver numéricamente cierto sistema lineal definido positivo $Ax = b$, se han aplicado k iteraciones del método del gradiente conjugado enunciado en el algoritmo 12, y definamos la *matriz de residuos* como la matriz que tiene como columnas los vectores residuales generados hasta ese momento, es decir, $\mathcal{R}_k \in \mathbb{R}^{n\times k}$ definida por

$$\mathcal{R}_k = [r_0, \ldots, r_{k-1}]\,, \tag{3.71}$$

así como la matriz triangular superior $\mathcal{B}_k \in \mathbb{R}^{n\times k}$ definida por

$$\mathcal{B}_k = \begin{bmatrix} 1 & -\beta_2 & 0 & \cdots & 0 \\ 0 & 1 & -\beta_3 & & \vdots \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\beta_k \\ 0 & \cdots & & 0 & 1 \end{bmatrix}, \tag{3.72}$$

donde los escalares β_i son los generados a partir del algoritmo 12.

Notemos que, según vimos en (3.63), $p_i = r_{i-1} + \beta_i p_{i-1}$ para $i = 2, \ldots, k$ y $p_1 = r_0$, luego las matrices $\mathcal{R}_k$ y $\mathcal{B}_k$, definidas en (3.71) y (3.72) respectivamente, satisfacen que $\mathcal{R}_k = \mathcal{P}_k \mathcal{B}_k$, donde $\mathcal{P}_k$ es la matriz definida en el teorema 3.2.2.

Puesto que las columnas de $\mathcal{P}_k = [p_1, \ldots, p_k]$ son A-conjugadas, deducimos que la matriz

$$\mathcal{R}_k^T A \mathcal{R}_k = (\mathcal{P}_k \mathcal{B}_k)^T A (\mathcal{P}_k \mathcal{B}_k) = \mathcal{B}_k^T \operatorname{diag}\left(p_1^T A p_1, \ldots, p_k^T A p_k\right) \mathcal{B}_k, \tag{3.73}$$

es una matriz tridiagonal. De hecho,

$$\mathcal{R}_k^T A \mathcal{R}_k = \begin{bmatrix} p_1^T A p_1 & -\beta_2 p_1^T A p_1 & 0 & \cdots & 0 \\ -\beta_2 p_1^T A p_1 & p_2^T A p_2 + \beta_2^2 p_1^T A p_1 & -\beta_3 p_2^T A p_2 & & \vdots \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\beta_k p_{k-1}^T A p_{k-1} \\ 0 & \cdots & & -\beta_k p_{k-1}^T A p_{k-1} & p_k^T A p_k + \beta_2^2 p_{k-1}^T A p_{k-1} \end{bmatrix}.$$

De (3.53) se sigue que si

$$\Delta = \operatorname{diag}(\rho_0, \ldots, \rho_{k-1}), \qquad \rho_i = \|r_i\|_2,$$

entonces las columnas de $\mathcal{R}_k \Delta^{-1}$ no son más que las columnas de $\mathcal{R}_k$ normalizadas, luego constituyen una base ortonormal del subespacio $\operatorname{span}\{r_0, Ar_0, \ldots, A^{k-1} r_0\} = \mathcal{K}(A, r_0, k)$. Por tanto, las columnas de esta matriz son en esencia los vectores de Lanczos q_i del algoritmo 1, es decir,

$$q_i = \pm r_{i-1}/\rho_{i-1}, \qquad i = 1, \ldots, k.$$

Por otra parte, la matriz tridiagonal asociada con los vectores de Lanczos calculados por el algoritmo 1 se puede obtener, a partir de productos que están disponibles en la k-ésima iteracion del gradiente conjugado, como

$$T_k = \Delta^{-1} R_k^T A R_k \Delta^{-1},$$

y, usando (3.73), se concluye que

$$T_k = \Delta^{-1} \mathcal{B}_k^T \operatorname{diag}\left(p_1^T A p_1, \ldots, p_k^T A p_k\right) \mathcal{B}_k \Delta^{-1}.$$

Por tanto, empleando el método del gradiente conjugado podemos obtener aproximaciones de los autovalores extremos de la matriz A, pero para ello, debemos almacenar las sucesivas direcciones de búsqueda p_i, así como los sucesivos vectores residuales r_i, con el costo de memoria que esto conlleva.

Capítulo 4

Problema de mínimos cuadrados

En el capítulo anterior se ha visto cómo se puede utilizar el método de Lanczos para resolver de forma eficiente sistemas lineales de ecuaciones donde la matriz de coeficientes es una matriz definida positiva y dispersa. Otra de las posibles aplicaciones del método de Lanczos es resolver problemas de mínimos cuadrados. Precisamente, el objetivo de este capítulo es desarrollar un algoritmo para resolver el problema

$$\text{mín} \left\| Ax - b \right\|_2 , \tag{4.1}$$

siendo $A \in \mathbb{R}^{m \times n}$ una matriz rectangular y dispersa. Para ello, será necesario en primer lugar desarrollar el *algoritmo de Lanczos de bidiagonalización* de matrices, un algoritmo similar al algoritmo clásico de Lanczos, pero cuya salida es, en vez de una matriz tridiagonal, una matriz bidiagonal.

4.1. Algoritmos de bidiagonalización

Consideremos una matriz rectangular $A \in \mathbb{R}^{m \times n}$. Antes de comenzar a desarrollar un algoritmo para resolver el problema de mínimos cuadrados (4.1), es necesario definir lo que entendemos por *descomposición* o *factorización bidiagonal completa*.

Definición 4.1 (**Factorización bidiagonal completa**). *Dada $A \in \mathbb{R}^{m \times n}$ una matriz rectangular, las descomposiciones*

$$U^T A S = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \alpha_3 & \\ & & \ddots & \ddots \end{bmatrix} = B \in \mathbb{R}^{m \times n}, \quad P^T A V = \begin{bmatrix} \rho_1 & \theta_2 & & \\ & \rho_2 & \theta_3 & \\ & & \rho_3 & \theta_4 \\ & & & \ddots & \ddots \end{bmatrix} = R \in \mathbb{R}^{m \times n},$$

donde $U, P \in \mathbb{R}^{m \times m}$ y $S, V \in \mathbb{R}^{n \times n}$ tienen columnas mutuamente ortonormales, se conocen respectivamente por bidiagonalización inferior y superior completa respectivamente.

En el desarrollo de este capítulo no solo aparecerá la bidiagonalización completa de una matriz, sino que también lo harán la k-ésima bidiagonalización parcial y la $(k+)$-ésima bidiagonalización parcial de la matriz A.

Definición 4.2 (**k-ésima bidiagonalización parcial**). *Dada una matriz rectangular $A \in \mathbb{R}^{m\times n}$ con factorización bidiagonal inferior completa dada por $U^T AS = B$, donde $U = [\,u_1\,,\,u_2\,,\,\ldots\,,\,u_m]$ y $S = [\,s_1\,,\,s_2\,,\,\ldots\,,\,s_n]$, y un entero k tal que $k \leq min\{n, m\}$, se define como k-ésima factorización bidiagonal parcial inferior de A a la descomposición*

$$U_k^T AS_k = B_k \in \mathbb{R}^{k\times k}, \quad \text{con } U_k \in \mathbb{R}^{m\times k},\ S_k \in \mathbb{R}^{n\times k},$$

donde B_k denota a la submatriz principal superior izquierda de B, y U_k y S_k las matrices formadas por las primeras k columnas de U y S respectivamente.

De manera análoga, si $P^T AV = R$ es una factorización bidiagonal completa de la matriz rectangular A, entonces su k-ésima bidiagonalización parcial superior es la descomposición

$$P_k^T AV_k = R_k \in \mathbb{R}^{k\times k}, \quad \text{con } P_k \in \mathbb{R}^{m\times k},\ V_k \in \mathbb{R}^{n\times k},$$

donde R_k denota a la submatriz principal superior izquierda de R, y P_k y V_k las matrices formadas por las primeras k columnas de P y V respectivamente.

Notemos que, utilizando las matrices U_k, S_k y B_k de la definición anterior se cumple que

$$B = \left[\begin{array}{c|ccc} B_k & & & \\ \hline \beta_{k+1} & \alpha_{k+1} & & \\ & \beta_{k+2} & \alpha_{k+2} & \\ & & \ddots & \ddots \end{array}\right], \qquad \begin{aligned} U &= [U_k, u_{k+1}, \ldots, u_m]\,, \\ S &= [S_k, s_{k+1}, \ldots, s_n]\,,, \end{aligned}$$

Por otro lado, utilizando las matrices P_k, V_k y R_k, es evidente que se cumple que

$$R = \left[\begin{array}{c|ccc} R_k & \theta_{k+1} & 0 & \ldots \\ \hline & \rho_{k+1} & \theta_{k+2} & \\ & & \rho_{k+2} & \theta_{k+3} \\ & & & \ddots \end{array}\right], \qquad \begin{aligned} P &= [P_k, p_{k+1}, \ldots, p_m]\,, \\ V &= [V_k, v_{k+1}, \ldots, v_n]\,. \end{aligned}$$

Definición 4.3 ((**k+**)**-ésima bidiagonalización parcial inferior**). *Utilizando la notación de la definición 4.2, dado un entero k tal que $k < min\{n, m\}$, llamamos $(k+)$-ésima factorización bidiagonal parcial inferior de la matriz A a la descomposición*

$$U_{k+1}^T AS_k = \left[\begin{array}{c} B_k \\ \hline e_k^T \beta_{k+1} \end{array}\right] = B_{k+} \in \mathbb{R}^{(k+1)\times k}, \quad \text{con } U_{k+1} \in \mathbb{R}^{m\times(k+1)},\ S_k \in \mathbb{R}^{n\times k}.$$

De manera similar, se puede definir la $(k+)$-ésima bidiagonalización parcial superior, pero esta definición no la usaremos a lo largo de este capítulo, por lo que omitimos escribir de forma explícita esta factorización.

Notemos que, dependiendo de la relación entre m y n, las bidiagonalizaciones completas pueden contener o no bloques enteros de ceros. Es evidente que dichas

transformaciones están estrechamente conectadas, pues una bidiagonalización superior completa de A se puede expresar como una bidiagonalización inferior completa de A^T.

La tarea de bidiagonalización matricial puede ser llevada a cabo mediante procesos de Householder (ver 5.4.3 de [2]). Sin embargo, si la matriz A es de gran tamaño y dispersa, podemos esperar que aparezcan matrices de gran tamaño y densas en el proceso de bidiagonalización. Por ello, sería conveniente encontrar un algoritmo que compute las matrices bidiagonal superior e inferior B y R sin actualizar la matriz A en los pasos intermedios.

En las siguientes secciones se pretende desarrollar algoritmos eficientes para la bidiagonalización superior e inferior completa de matrices rectangulares dispersas y de gran tamaño.

4.1.1. Bidiagonalización superior

Supongamos que $P^T AV = R$ representa la bidiagonalización superior completa de $A \in \mathbb{R}^{m\times n}$, tal que

$$P^T AV = \begin{bmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \rho_3 & \theta_4 & \\ & & & \ddots & \ddots \end{bmatrix} = R \tag{4.2}$$

El objetivo de esta sección es encontrar un algoritmo eficiente que realice esta bidiagonalización sin actualizar la matriz A, pues en tal caso, estropearíamos su dispersión.

El algoritmo que desarrollaremos se conoce como *algoritmo de Lanczos de bidiagonalización superior*, también conocido por método de Golub-Kahan-Lanczos.

Podemos plantear un algoritmo con estas características de la misma forma que razonamos durante el desarrollo del algoritmo de Lanczos, es decir, igualando directamente las columnas de las ecuaciones $AV = PR$ y $A^T P = VR^T$. Denotemos por p_i y v_i a las i-esimas columnas de las matrices P y V respectivamente, de forma que $P = [\, p_1 \,,\, p_2 \,,\, \ldots \,,\, p_m]$ y $V = [\, v_1 \,,\, v_2 \,,\, \ldots \,,\, v_n]$.

Igualando la primera y la $(k+1)$-ésimas columnas de la igualdad $AV = PR$, poniendo $p_0 \equiv 0$, deducimos que para $k = 1, 2, \ldots n-1$ se tiene que

$$\begin{aligned} Av_1 &= \rho_1 p_1, \\ Av_{k+1} &= \theta_{k+1} p_k + \rho_{k+1} p_{k+1}. \end{aligned} \tag{4.3}$$

Por otra parte, igualando ahora las k-ésimas columnas de $A^T P = VR^T$, se sigue que para $k = 1, \ldots m$ se tiene que

$$A^T p_k = \rho_k v_k + \theta_{k+1} v_{k+1}, \tag{4.4}$$

Fijemos los escalares $\rho_k \geq 0$ y $\theta_k \geq 0$ para $k = 1, 2, \ldots$ de forma que en todo momento se verifique $\|p_k\|_2 = \|v_k\|_2 = 1$, es decir, que se verifique $PP^T = I_m$ y $VV^T = I_n$ hasta precisión de máquina. Tomemos como v_1 el vector columna unitario $v_1 = A^T b/\left\|A^T b\right\|$, que, utilizando una notación acorde con el cálculo de los demás vectores v_k, se escribe como $\theta_1 v_1 = A^T b$, siendo $\theta_1 = \left\|A^T b\right\|$.

De esta forma, de (4.3) y (4.4), se deduce que los vectores columna necesarios para la bidiagonalización superior de la matriz A se pueden generar a partir de de las siguientes ecuaciones:

$$\begin{aligned} \theta_1 v_1 &= A^T b, \\ \rho_1 p_1 &= A v_1, \\ \theta_{k+1} v_{k+1} &= A^T p_k - \rho_k v_k, \qquad k = 1, 2, \ldots \\ \rho_{k+1} p_{k+1} &= A v_{k+1} - \theta_{k+1} p_k, \qquad k = 1, 2, \ldots \end{aligned} \tag{4.5}$$

Con ellas, y sin más que definir los vectores r_k y s_k, tenemos el algoritmo 13, un algoritmo de bidiagonalización superior de matrices.

Algoritmo 13: Reducción de A a la forma bidiagonal superior.

Entrada: Matriz $A \in \mathbb{R}^{m\times n}$ y un vector $b \in \mathbb{R}^n$
Salida : Matrices P, V y R tales que $P^T AV = R$, siendo R bidiagonal superior.

$\theta_1 = \left\|A^T b\right\|$, $v_1 = A^T b/\theta_1$, $\rho_1 = \left\|A^T v_1\right\|$, $p_1 = A v_1/\rho_1$, $k = 1$.
while $\theta_k \neq 0$ **do**
  $r_{k+1} = A^T p_k - p_k v_k$
  $k = k + 1$
  $\theta_k = \|r_k\|$
  $v_k = r_k/\theta_k$
  $s_k = A v_k - \theta_k p_{k-1}$
  $\rho_k = \|s_k\|$
  $p_k = s_k/\rho_k$
end

Notemos que tras k iteraciones, hemos construido la k-ésima bidiagonalización parcial superior, es decir, tenemos las matrices

$$P_k = [p_1, p_2, \ldots, p_k], \qquad V_k = [v_1, v_2, \ldots, v_k], \qquad R_k = \begin{bmatrix} \rho_1 & \theta_2 & & \cdots & 0 \\ 0 & \rho_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \theta_k \\ 0 & \cdots & & 0 & \rho_k \end{bmatrix}, \tag{4.6}$$

mediante las cuales podemos reescribir las ecuaciones (4.5) como

$$V_k\left(\theta_1 e_1\right) = A^{\mathrm{T}} b, \tag{4.7}$$

$$A V_k = P_k R_k, \tag{4.8}$$

$$A^{\mathrm{T}} P_k = V_k R_k^T + \theta_{k+1} v_{k+1} e_k^{\mathrm{T}}, \tag{4.9}$$

y, en caso de utilizar aritmética exacta, también se verificaría que $P_k^T P_k = V_k^T V_k = I_k$. El algoritmo 13 termina cuando encuentra $\theta_{k+1} = 0$. Sin embargo, cuando esto ocurre, se deduce de (4.8) y (4.9) que

$$A\,[v_1, v_2, \ldots, v_k] = [p_1, p_2, \ldots, p_k]\,B_k,$$
$$A^T\,[p_1, p_2, \ldots, p_k] = [v_1, v_2, \ldots, v_k]\,B_k^T.$$

4.1.2. Bidiagonalización inferior

Supongamos ahora que $U^T AS = B$ representa la bidiagonalización inferior de $A \in \mathbb{R}^{m\times n}$ tal que

$$U^T AS = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \alpha_3 & \\ & & \ddots & \ddots \end{bmatrix} = B. \tag{4.10}$$

De cara a encontrar un algoritmo de bidiagonalización inferior eficiente, podemos razonar de manera análoga a la sección anterior. Igualando las columnas de $AS = UB$, y denotando por s_i y u_i a las i-ésimas columnas de S y U respectivamente, deducimos que

$$[As_1,\, As_2, \ldots,\, As_n] = [\alpha_1 u_1 + \beta_2 u_2,\, \alpha_2 u_2 + \beta_3 u_3,\, \ldots,\, \alpha_n u_n]\,,$$

y de igualar las columnas de $A^T U = SB^T$

$$\left[A^T u_1,\, A^T u_2, \ldots,\, A^T u_n\right] = [\alpha_1 s_1, \beta_2 s_2 + \alpha_2 s_2,\, \beta_3 s_3 + \alpha_3 s_3,\, \ldots,\, \beta_n s_{n-1} + \alpha_n s_n]\,,$$

de donde obtenemos las siguientes ecuaciones

$$\begin{aligned} A^T u_1 &= \alpha_1 s_1, & & \\ \beta_{k+1} u_{k+1} &= A s_k - \alpha_k u_k & & k = 1, 2, \ldots \\ \alpha_{k+1} s_{k+1} &= A^T u_{k+1} - \beta_{k+1} s_k & & k = 1, 2, \ldots \end{aligned} \tag{4.11}$$

Notemos que si fijamos $\beta_1 u_1 = b$, es decir, $u_1 = b/\|b\|$, entonces s_1 es un múltiplo escalar de $A^T b$, y puesto que de nuevo utilizamos los parámetros α_i y β_i de forma que $\|s_k\|_2 = \|u_k\|_2 = 1$, tenemos que $s_1 = v_1$, siendo v_1 el vector inicial definido en algoritmo 13. De esta forma, comparando la recurrencia de v_{k+1} en (4.5), y la de s_{k+1} en (4.11), se deduce que la matriz S es precisamente la matriz V del algoritmo anterior.

Por consistencia con la notación del algoritmo 13, de ahora en adelante $s_i = v_i$. Sin más que utilizar esta nueva notación, estamos en condiciones de escribir un algoritmo de bidiagonalización inferior. Si fijamos $u_1 = b/\beta_1$, tenemos que el algoritmo de reducción a forma bidiagonal superior se puede escribir como en el algoritmo 14.

De nuevo, si denotamos por U_{k+1} y V_k a las matrices cuyas columnas son los $k+1$ y k primeros vectores u_i y v_i respectivamente, es decir,

$$U_{k+1} = [u_1, u_2, \ldots, u_{k+1}]\,, \qquad V_k = [v_1, v_2, \ldots, v_k]\,, \tag{4.12}$$

Algoritmo 14: Reducción de A a la forma bidiagonal inferior.

Entrada: Matriz $A \in \mathbb{R}^{m\times n}$ y un vector $b \in \mathbb{R}^n$
Salida : Matrices U, V y B tales que $U^T AV = B$, siendo B bidiagonal inferior.

$\beta_1 = \|b\|$, $u_1 = b/\beta_1$, $\alpha_1 = \left\|A^T u_1\right\|$, $v_1 = A^T u_1/\alpha_1$. $k = 1$,
while $\beta_k \neq 0$ **do**
  $r_{k+1} = Av_k - \alpha_k u_k$
  $k = k + 1$
  $\beta_k = \|r_k\|$
  $u_k = r_k/\beta_k$
  $v_k = A^T u_k - \beta_k v_{k-1}$
  $\alpha_k = \|v_k\|$
  $v_k = v_k/\alpha_k$
end

tenemos que

$$U_{k+1}^T AV_k = B_{k+} = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ 0 & \dots & 0 & \beta_{k+1} \end{bmatrix} \in \mathbb{R}^{(k+1)\times k}, \tag{4.13}$$

luego $U_{k+1}^T AV_k$ es la $(k+)$-ésima bidiagonalización parcial inferior de la matriz A.

Utilizando las matrices definidas en (4.12), podemos escribir las ecuaciones (4.11) como

$$U_{k+1}\left(\beta_1 e_1\right) = b, \tag{4.14}$$
$$AV_k = U_{k+1}B_{k+}, \qquad k = 1, 2, \dots \tag{4.15}$$
$$A^{\mathrm{T}}U_{k+1} = V_k B_{k+}^{\mathrm{T}} + \alpha_{k+1}v_{k+1}e_{k+1}^{\mathrm{T}}, \qquad k = 1, 2, \dots \tag{4.16}$$

y en aritmética exacta tendríamos que $U_{k+1}^T U_{k+1} = I$ y $V_k^T V_k^T = I$.

4.1.3. Relación entre las bidiagonalizaciones inferior y superior

En la sección anterior se han desarrollado los algoritmos de Golub-Kahan-Lanczos. Según se ha visto, en estos algoritmos hemos tenido libre la elección de los vectores v_1 y s_1. Sin embargo, se han fijado elecciones concretas en vistas de que las matrices V y S fuesen idénticas. Por tanto, tras k iteraciones de cualquiera de los dos algoritmos, obtenemos la matriz V_k.

Considerar dicha elección de los vectores iniciales tiene la ventaja de que proporciona la igualdad

$$B_{k+}^T B_{k+} = R_k^T R_k, \tag{4.17}$$

siendo B_{k+} y R_k las matrices definidas anteriormente. Que se tenga la igualdad (4.17) no debería sorpendernos en absoluto, pues V_k es el resultado de aplicar k iteraciones del algoritmo de Lanczos[1] a la matriz $B = A^T A$, que es cuadrada y simétrica.

Para verlo, simplemente tenemos que darnos cuenta de que a través de dichas bidiagonalizaciones se llega a la tridiagonalización de Lanczos de $A^T A$. Notemos que

$$\begin{aligned} A^T A V_k &= A^T P_k R_k = \left[V_k R_k^T + \theta_{k+1} v_{k+1} e_k^T\right] R_k \\ &= V_k \underbrace{R_k^T R_k}_{T_k} + \theta_{k+1} v_{k+1} \left[0, \ldots, 0, \rho_k\right], \end{aligned} \tag{4.18}$$

donde en la primera de las igualdades se ha utilizado la expresión de AV_k dada en (4.8), y en la segunda la expresión de A_k^T dada en (4.9). Razonando de forma similar se deduce que

$$\begin{aligned} A^T A V_k &= A^T U_{k+1} B_{k+} = \left[V_k B_{k+}^T + \alpha_{k+1} v_{k+1} e_{k+1}^T\right] B_{k+} \\ &= V_k \underbrace{B_{k+}^T B_{k+}}_{T_k} + \alpha_{k+1} v_{k+1} \left[0, \ldots, 0, \beta_{k+1}\right], \end{aligned} \tag{4.19}$$

donde en este caso se ha utilizado (4.15) y (4.16) en la primera y segunda de las igualdades respectivamente.

De las expresiones (4.18) y (4.19) se deduce que necesariamente se tiene (4.17), y como

$$B_{k+}^T B_{k+} = \begin{bmatrix} \alpha_1^2 + \beta_2^2 & \beta_2\alpha_2 & & & \\ \beta_2\alpha_2 & \alpha_2^2 + \beta_3^2 & \beta_3\alpha_3 & & \\ & \beta_3\alpha_3 & \beta_4^2 + \alpha_3^2 & \ddots & \\ & & \ddots & \ddots & \beta_k\alpha_k \\ & & & \beta_k\alpha_k & \beta_{k+1}^2 + \alpha_k^2 \end{bmatrix},$$

y

$$R_k^T R_k = \begin{bmatrix} \rho_1^2 & \rho_1\theta_2 & & & \\ \rho_1\theta_2 & \theta_2^2 + \rho_2^2 & \rho_2\theta_3 & & \\ & \rho_2\theta_3 & \rho_3^2 + \theta_3^2 & \ddots & \\ & & \ddots & \ddots & \rho_{k-1}\theta_k \\ & & & \rho_{k-1}\theta_k & \rho_k^2 + \theta_k^2 \end{bmatrix},$$

concluimos las siguientes relaciones

$$\begin{aligned} &\alpha_1^2 + \beta_2^2 = \rho_1^2, \\ &\alpha_k^2 + \beta_{k+1}^2 = \rho_k^2 + \theta_k^2, \quad \alpha_k\beta_k = \rho_{k-1}\theta_k, \quad \text{para } k > 1. \end{aligned} \tag{4.20}$$

[1] Para verlo, no tenemos más que tomar la expresión (2.14), donde usando la notación de dicha expresión, Q_k es la matriz V_k.

La siguiente propiedad sorprendente es que R_k es la matriz que uno obtendría al aplicar el algoritmo de factorización QR convencional a la matriz B_k, de forma que

$$Q_k B_{k+} = \begin{bmatrix} R_k \\ 0 \end{bmatrix} \in \mathbb{R}^{(k+1)\times k}, \tag{4.21}$$

Esto es equivalente a demostrar que

$$Q_k^T \begin{bmatrix} R_k \\ 0 \end{bmatrix} = B_{k+} \in \mathbb{R}^{(k+1)\times k}. \tag{4.22}$$

Supongamos que se tiene la igualdad

$$Q_k^T \begin{bmatrix} \tilde{R}_k \\ 0 \end{bmatrix} = B_{k+} \in \mathbb{R}^{(k+1)\times k}, \tag{4.23}$$

para cierta matriz $\tilde{R}_k \in \mathbb{R}^{k\times k}$ triangular superior, y veamos que, necesariamente, $\tilde{R}_k \equiv R_k$. Considerando el producto de B_{k+} por su matriz transpuesta, y aplicando la igualdad anterior, se sigue que

$$B_{k+}^T B_{k+} = \begin{bmatrix} \tilde{R}_k & 0 \end{bmatrix} Q_k Q_k^T \begin{bmatrix} \tilde{R}_k \\ 0 \end{bmatrix} = \begin{bmatrix} \tilde{R}_k^T & 0 \end{bmatrix} \begin{bmatrix} \tilde{R}_k \\ 0 \end{bmatrix} = \tilde{R}_k^T \tilde{R}_k,$$

donde se ha utilizado que $Q_k^T Q_k = I_{k+1}$ por ser Q_k ortogonal. Por tanto,

$$B_{k+}^T B_{k+} = \tilde{R}_k^T \tilde{R}_k,$$

y, sin más que sustituir (4.17) en el término izquierdo de la expresión anterior, se concluye que

$$R_k^T R_k = \tilde{R}_k^T \tilde{R}_k.$$

Multiplicando por $\tilde{R}_k^{-1}$ por la derecha, y luego por $(R_k^{-1})^T$ por la izquierda, se tiene que

$$\underbrace{(R_k^{-1})^T \tilde{R}_k^T}_{\text{triang. inferior}} = \underbrace{R_k \tilde{R}_k^{-1}}_{\text{triang. superior}},$$

de donde se deduce que $R_k = D\tilde{R}_k$, siendo D una matriz diagonal.

Volvamos sobre la igualdad (4.23). De ahí se deduce que

$$\begin{bmatrix} \alpha_1 \\ \beta_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \tilde{r}_{11}\, q_1, \tag{4.24}$$

donde $\tilde{r}_{11}$ es la primera entrada de la diagonal de $\tilde{R}_k$. Puesto que q_1 tiene norma 1, se tiene que

$$\tilde{r}_{11}^2 = \alpha_1^2 + \beta_2^2,$$

pero por (4.20) sabemos que $\tilde{r}_{11}^2 = \rho_1^2$, de donde se sigue que $|\tilde{r}_{11}| = r_{11}$ Razonando de forma similar para el resto de columnas, se deduce que la diagonal de R_k es la misma que la diagonal de $\tilde{R}_k$.

4.2. Algoritmo LSQR

Los algoritmos de bidiagonalización desarrollados en la sección anterior serán utilizados para implementar un método de resolución eficiente de problemas de mínimos cuadrados. Este método se conoce como *algoritmo LSQR*, donde *LSQR* son las siglas de *Least Square problem with QR factorization*, en inglés, problema de mínimos cuadrados con factorización QR, haciendo referencia a que este algoritmo emplea, como veremos, la relación mediante la factorización QR de las bidiagonalizaciones anteriormente descritas.

Supongamos que queremos resolver el problema (4.1), y consideremos la matriz U procedente de la bidiagonalización inferior de A dada por (4.10). Puesto que esta matriz es ortogonal, se tiene que

$$\min_x \|Ax - b\|_2 = \min_x \left\|U^T(Ax - b)\right\|_2 , \tag{4.25}$$

y, definiendo y tal que $x = Vy$, puesto que V también es ortogonal, $\|x\| = \|y\|$, luego

$$\min_x \|Ax - b\|_2 = \min_y \left\|U^T AVy - U^T b\right\|_2 = \min_y \|By - \beta_1 e_1\|_2 , \tag{4.26}$$

con lo que nuestro problema inicial se reduce a resolver

$$\min_y \|By - \beta_1 e_1\|_2 . \tag{4.27}$$

Computacionalmente, es más sencillo encontrar la solución del problema (4.27) que el problema inicial, pues para resolver este segundo podemos utilizar la factorización QR (4.21) que conecta la bidiagonalización parcial superior e inferior que hemos considerado anteriormente.

Si repetimos el mismo razonamiento anterior, pero considerando en este caso la (k+)-ésima descomposición bidiagonal inferior de $A \in \mathbb{R}^{m\times n}$ dada por el algoritmo 14 en vez de la descomposición bidiagonal inferior total, estaremos definiendo para cada vector $y_k \in \mathbb{R}^{k\times 1}$

$$x_k = V_k y_k, \tag{4.28}$$
$$r_k = b - Ax_k, \tag{4.29}$$
$$t_{k+1} = \beta_1 e_1 - B_{k+} y_k, \tag{4.30}$$

donde β_1 se define como en este mismo algoritmo, es decir, $\beta_1 = \|b\|$.

Utilizando las definiciones anteriores y las igualdades vistas en (4.14) y (4.15), tenemos que $U_{k+1}t_{k+1} = r_k$, pues

$$\begin{aligned} U_{k+1}\, t_{k+1} = U_{k+1}\left(\beta_1 e_1 - B_{k+} y_k\right) &= \underbrace{U_{k+1}\beta_1 e_1}_{b} - \underbrace{U_{k+1}B_{k+}}_{AV_k} y_k = b - A\underbrace{V_k y_k}_{x_k} \\ &= b - Ax_k = r_k. \end{aligned} \tag{4.31}$$

Para resolver el problema de mínimos cuadrados, construiremos una sucesión $\{x_k\}$ de soluciones aproximadas del problema (4.1). Por tanto, si x_k es una buena aproximación a la solución del problema, entonces $\|r_k\|$ es pequeño. Puesto que U_{k+1} es (teóricamente) ortonormal por construcción, tiene sentido elegir y_k de forma que se minimice $\|t_{k+1}\|$. Con todo esto, nuestro problema inicial se reduce a resolver el problema de mínimos cuadrados

$$\text{mín} \, \|\beta_1 e_1 - B_k y_k\| \, . \tag{4.32}$$

Considerando la matriz Q_k de la factorización QR anteriormente descrita, se tiene que

$$Q_k \, [B_k \mid \beta_1 e_1] = \begin{bmatrix} R_k & f_k \\ & \phi_{k+1} \end{bmatrix} = \left[\begin{array}{ccccc:c} \rho_1 & \theta_2 & & & & \phi_1 \\ & \rho_2 & \ddots & & & \phi_2 \\ & & \ddots & \ddots & & \vdots \\ & & & \ddots & \theta_k & \vdots \\ & & & & \rho_k & \phi_{k-1} \\ & & & & & \phi_k \\ \hdashline & & & & & \phi_{k+1}, \end{array}\right] \tag{4.33}$$

donde $Q_k = Q_{k,k+1} \, \ldots \, Q_{2,3} \, Q_{1,2}$ es un producto de rotaciones utilizado para eliminar $\beta_2, \beta_3, \ldots$ de la matriz B_k, y $f_k = [\phi_1, \ldots, \phi_k]^T$ es un vector columna de tamaño k. Notemos que el vector f_k se ha definido según

$$f_k = R_k y_k. \tag{4.34}$$

Sustituyendo la factorización (4.22) en (4.30), se observa que

$$t_{k+1} = \beta_1 - Q_k^T \begin{bmatrix} R_k \\ 0 \end{bmatrix} y_k = \beta_1 - Q_k^T \begin{bmatrix} f_k \\ 0 \end{bmatrix}, \tag{4.35}$$

donde en la última igualdad se ha introducido la definición de (4.34).

Por otra parte, de (4.33), se deduce que

$$Q_k \beta_1 e_1 = \begin{bmatrix} f_k \\ \phi_k \end{bmatrix},$$

luego

$$\beta_1 e_1 = Q_k^T \begin{bmatrix} f_k \\ \phi_k \end{bmatrix}.$$

Sustituyendo esta última igualdad en (4.35), se sigue que

$$t_{k+1} = Q_k^T \begin{bmatrix} 0 \\ \phi_{k+1} \end{bmatrix},$$

donde la segunda de las matrices es una matriz con $k+1$ filas y k columnas cuya única entrada no nula es la última, siendo esta ϕ_{k+1}.

Generalmente, y_k no se podrá expresar el términos de y_{k-1}, y una relación de este tipo simplificaría notablemente el algoritmo. Sin embargo, $[R_k f_k]$ es la misma matriz que $[R_{k-1} f_{k-1}]$ con una columna y una fila adicional. De (4.34) se deduce que $y_k = R_k^{-1} f_k$, luego sustituyendo en (4.28) encontramos la relación

$$x_k = V_k R_k^{-1} f_k \equiv D_k f_k, \tag{4.36}$$

donde las columnas de $D_k = [d_1, d_2, \ldots, d_k]$ se pueden encontrar de forma sucesiva mediante el sistema $R_k^T D_k^T = V_k^T$ mediante sustitución progresiva. Poniendo $x_0 = d_0 = 0$, tenemos que

$$d_k = \frac{1}{\rho_k}(v_k - \theta_k d_{k-1}), \tag{4.37}$$

$$x_k = x_{k-1} + \phi_k d_k. \tag{4.38}$$

Además, notemos que únicamente necesitamos almacenar los últimos iterantes, con las ventajas de memoria que esto conlleva.

Nuestro siguiente objetivo es encontrar relaciones de recurrencia para que la idea anterior pueda ser implementada en forma de algoritmo. Volvamos sobre la factorización (4.33). Esta factorización se determina costruyendo la k-ésima rotacion $Q_{k,k+1}$ para operar en la fila k y la columna $k+1$ de $[B_k \ \beta_1 e_1]$ para eliminar β_{k+1}. Esto nos lleva, sin más que aplicar el algoritmo 5.2.2 de [2] al caso presente, a las relaciones

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \bar{\rho}_k & 0 & \bar{\phi}_k \\ \beta_{k+1} & \alpha_{k+1} & 0 \end{bmatrix} = \begin{bmatrix} \rho_k & \theta_{k+1} & \phi_k \\ 0 & \bar{\rho}_{k+1} & \bar{\phi}_{k+1} \end{bmatrix}, \tag{4.39}$$

donde $\bar{\phi}_1 \equiv \alpha_1$, $\bar{\phi}_1 \equiv \beta_1$, y los escalares c_k y s_k son los elementos no triviales de la matriz de rotación $Q_{k,k+1}$.

Las rotaciones $Q_{k,k+1}$ son eliminadas una vez han sido utilizadas en (4.39), pues no requerimos almacenar la matríz Q_k de la factorización QR. Además, podemos ahorrar una división en la expresión (4.38) utilizando los vectores $w_k \equiv \rho_k d_k$.

Con todo ello, estamos en condiciones de poder escribir el algoritmo LQSR. Al igual que en todos los algoritmos descritos en este capítulo, los escalares α_i y β_i son tomados positivos, y se eligen para normalizar los correspondientes vectores, de forma que, por ejemplo, $\alpha_1 v_1 = A^T u_1$ implica el cálculo de $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|$, y $v_1 = \bar{v}_1/\alpha_1$.

Algoritmo 15: LSQR para el problema de mínimos cuadrados

Entrada: Matriz $A \in \mathbb{R}^{m\times n}$, un vector $b \in \mathbb{R}^n$ y una tolerancia máxima *tol*
Salida : Solución aproximada del problema de mínimos cuadrados (4.1)

$\beta_1 = \|b\|$, $u_1 = b/\beta_1$, $\alpha_1 = \left\|A^T u_1\right\|$, $v_1 = A^T u_1/\alpha_1$, $w_1 = v_1$, x_0.
$\bar{\phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1$, $k = 1$, $t = 1$.
while $t \leq tol$ **do**
Bidiagonalización
$\beta_{k+1}u_{k+1} = Av_k - \alpha_k u_k$ con $\beta_{k+1} = 1/\|u_{k+1}\|$
$\alpha_{k+1}v_{k+1} = A^T u_{k+1} - \beta_{k+1}v_k$ con $\alpha_{k+1} = 1/\|v_{k+1}\|$
Transformaciones Ortogonales
$\rho_k = \left(\bar{\rho}_k^2 + \beta_{k+1}^2\right)^{1/2}$ $c_k = \bar{\rho}_k/\rho_k$
$s_k = \beta_{k+1}/\rho_k$
$\theta_{k+1} = s_k\alpha_{k+1}$
$\bar{\rho}_{k+1} = -c_k\alpha_{k+1}$
$\phi_k = c_k\bar{\phi}_k$
$\bar{\phi}_{k+1} = s_k\bar{\phi}_k$
Actualización de la solución
$x_k = x_{k-1} + (\phi_k/\rho_k)w_k$
$w_{k+1} = v_{k+1} - (\theta_{k+1}/\rho_k)w_k$
$k = k + 1$
end

4.2.1. Relación con el método de Tridiagonalización de Lanczos

El objetivo de esta sección es evidenciar la conexión del algoritmo LSQR y el método de Lanczos. El algoritmo LSQR se basa en los procedimientos de bidiagonalización anteriormente descritos. Se puede ver la relacción entre estos métodos de diferentes maneras. Una de ellas es mostrar que el algoritmo de bidiagonalización superior de una matriz rectangular A es equivalente a aplicar la tridiagonalización de Lanczos a la matriz A^TA, o bien a la matriz AA^T. Para verlo, supongamos que se han aplicado k iteraciones del algoritmo de bidiagonalización superior, de forma que se tiene (4.8) y (4.9), es decir,

$$AV_k = P_k R_k, \tag{4.40}$$
$$A^{\mathrm{T}}P_k = V_k R_k^T + \theta_{k+1}v_{k+1}e_k^{\mathrm{T}}. \tag{4.41}$$

Premultiplicando la ecuación (4.40) por A^T se tiene que

$$A^TAV_k = A^TP_kR_k,$$

donde sin más que sustituir (4.41) se sigue que

$$A^TAV_k = V_k + \theta_{k+1}\rho_k v_{k+1}e_k^T. \tag{4.42}$$

Recordemos que la situación tras k iteraciones del método de Lanczos viene dada, utilizando la notación del capítulo 2, por

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

luego comparando la expresión anterior con (4.42), se deduce que el algoritmo de bidiagonalización superior es equivalente a aplicar el algoritmo de Lanczos a la matriz $A^T A$. En particular, los vectores v_i generados a partir del algoritmo 13 forman una base ortonormal para el espacio de Krylov

$$\mathcal{K}(A^T A, v_1, k) = span(v_1, A^T A v_1, \ \dots \ , (A^T A)^{k-1} v_1).$$

Otra forma de relacionar las ecuaciones (4.40) y (4.41) es premultiplicar la segunda de ellas por A^T, de forma que

$$AA^T P_k = AV_k R_k^T + \theta_{k+1} A v_{k+1} e_k^T,$$

y, sustituyendo (4.40),

$$AA^T P_k = P_k R_k R_k^T + \theta_{k+1} A v_{k+1} e_k^T.$$

Esto podría parecer que no es una descomposición tridiagonal de Lanczos, pues en general el vector Av_{k+1} no tiene por qué ser ortogonal a P_k. Sin embargo, notemos que en la k-ésima iteración del algoritmo 13 se calcula $AV_{k+1} = \theta_{k+1} p_k + \rho_{k+1} p_{k+1}$ luego sustituyendo en la expresión anterior, tenemos que

$$AA^T P_k = P_k R_k R_k^T + \theta_{k+1} \left(\theta_{k+1} p_k + \rho_{k+1} p_{k+1}\right) e_k^T \tag{4.43}$$

$$= P_k \left(R_k R_k^T + \theta_{k+1}^2 e_k e_k^T\right) + \theta_{k+1} \rho_{k+1} p_{k+1} e_k^T. \tag{4.44}$$

Ahora, $R_k R_k^T + \theta_{k+1}^2 e_k e_k^T$ es una matriz simétrica definida positiva y tridiagonal, y que p_{k+1} es ortogonal a P_k, por lo que tenemos de nuevo una descomposición de Lanczos. Por tanto, los vectores p_i generados a partir del algoritmo 13 forman una base ortonormal del subespacio de Krylov

$$\mathcal{K}(AA^T, p_1, k) = span(p_1, AA^T p_1, \ \dots \ , (AA^T)^{k-1} p_1).$$

De hecho, como se enuncia tanto en la sección 9.3.3 de [2] como en [3], la relación del proceso de bidiagonalización y el método de Lanczos se puede poner de manifiesto de otras maneras. La idea es aplicar el Método de Lanczos a la matriz

$$C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}, \tag{4.45}$$

tomando como vector inicial

$$z_1 = \begin{bmatrix} 0 \\ v_1 \end{bmatrix}. \tag{4.46}$$

Se puede demostrar que los vectores de Lanczos que se generan en este proceso son precisamente

$$z_{2j-1} = \begin{bmatrix} 0 \\ v_j \end{bmatrix} \quad \text{y} \quad z_{2j} = \begin{bmatrix} p_j \\ 0 \end{bmatrix}, \tag{4.47}$$

y que después de $2k$ iteraciones del método de Lanczos, la matriz tridiagonal T_{2k} es

$$T_{2k} = \begin{bmatrix} 0 & \rho_1 & & & & & \\ \rho_1 & 0 & \theta_2 & & & & \\ & \theta_2 & 0 & \rho_2 & & & \\ & & \rho_2 & 0 & \theta_3 & & \\ & & & \theta_3 & 0 & \ddots & \\ & & & & \ddots & \ddots & \rho_k \\ & & & & & \rho_k & 0 \end{bmatrix}. \tag{4.48}$$

Por tanto, se deduce que dos iteraciones del método de Lanczos proporcionan la misma información que una iteración del algoritmo 13.

4.2.2. Generación de problemas

El objetivo de esta sección es proponer un procedimiento que genere problemas de mínimos cuadrados, cuya solución sea conocida, y que puedan ser utilizados como ejemplos sobre los que aplicar el algoritmo LSQR. Es evidente que este algoritmo puede ser también utilizado para proporcionar sistemas lineales de solución conocida, pues en caso de que A sea cuadrada, y $Ax = b$ un sistema compatible, el residuo será cero. En el capítulo 3 se ha presentado cómo se puede utilizar el método de Lanczos para resolver sistemas simétricos de matriz de coeficientes definida positiva. Utilizando este procedimiento, podemos resolver también sistemas cuadrados que no tengan matriz de coeficientes simétrica, completando así el estudio anterior.

En la sección 8 de [7] se proporciona el procedimiento de generación de problemas test que utilizaremos. Para ello, si $A \in \mathbb{R}^{m\times n}$ con $m \geq n$, debemos realizar los siguientes pasos

1. Elegir vectores $x, z \in \mathbb{R}^{n\times 1}$, $y \in \mathbb{R}^{m\times 1}$ y $c \in \mathbb{R}^{(m-n)\times 1}$, tales que $\|x\| = \|y\|$.

2. Definir las siguientes matrices

$$Y = I_m - 2yy^T \in \mathbb{R}^{m\times m}, \quad Z = I_n - 2zz^T \in \mathbb{R}^{n\times n},$$

   donde I_k representa la matriz identidad de tamaño k.

3. Definir $D \in \mathbb{R}^{n\times n}$ una matriz diagonal, y considerar D' y c' las matriz y vector completados definidos según

$$D' = \begin{bmatrix} D \\ 0 \end{bmatrix} \in \mathbb{R}^{m\times n}, \qquad c' = \begin{bmatrix} 0 \\ c \end{bmatrix} \in \mathbb{R}^{m\times 1},$$

   es decir, D' es la matriz diagonal D seguida de $(m-n)$ filas de ceros, mientras que c' es el vector c precedido de n filas de ceros.

4. Tomar como matriz A, y vectores r y b

$$A = YD'Z = Y \begin{bmatrix} D \\ 0 \end{bmatrix} Z, \qquad r = Yc' = Y \begin{bmatrix} 0 \\ c \end{bmatrix}, \qquad b = Ax + r.$$

Con este procedimiento, se puede mostrar que x es la solución del problema de mínimos cuadrados, luego además de conocer la solución exacta del problema, sabemos a priori que la mínima norma residual posible es $\|r\| = \|c\|$.

Una vez proporcionado una manera de obtener ejemplos con solución conocida, estamos en condiciones de aplicar el algoritmo LSQR a estos problemas. Para ahorrar la escritura implícita del problema a resolver, utilizaremos procedimientos aleatorios en Matlab para generar los problemas, pero indicaremos la semilla utilizada, de forma que se puedan recuperar la matriz A y el vector b del problema considerado.

Ejemplo 4.1. *Consideremos el siguiente problema de mínimos cuadrados*

$$\text{mín} \|Ax - b\|_2 ,$$

donde $A \in \mathbb{R}^{40\times 15}$ *y* $b \in \mathbb{R}^{40}$ *han sido generados a través del siguiente procedimiento en Matlab*

```
m=40; n=15;          rng(27,'philox'); % Fijamos la semilla
x=rand(n,1);
y= rand(m,1);          y=y/norm(y);
z= rand(n,1);          z=z/norm(z);
c=rand(m-n,1);         C= [zeros(n,1) ; c];
Y= eye(m)-2*y*y';      Z=eye(n)-2*z*z';
D= [diag(rand(n,1)); zeros(m-n,n)];
A= Y*D*Z;              r=Y*C;      b=A*x+r;
filename='lsqr.mat'; save(filename)
```

La solución exacta del problema viene dada por el vector x*, mientras que el residuo mínimo resulta ser* $\|r\| = \|c\| = 3.113606738907739$.

Estudiemos la diferencia obtenida entre la norma residual $\|r_k\|$ *obtenida en la* k*-ésima iteración del algoritmo 15 y la norma del residuo mínimo alcanzable (es decir, el residuo de la solución óptima del problema considerado). Para enfatizar dicha diferencia, se ha considerado el logaritmo en base 10 de dicha diferencia, de forma que estamos considerando*

$$Error = \log_{10}(|\ \|r_k\| - \|r\|\ |). \tag{4.49}$$

Iteración	*Error*	*Iteración*	*Error*
$m = 1$	-1.5550	$m = 9$	-4.3393
$m = 3$	-2.1148	$m = 11$	-4.3787
$m = 5$	-3.4318	$m = 13$	-4.4268
$m = 7$	-4.0780	$m = 14$	-6.2185

Tabla 4.1: Error (4.49) para el problema considerado.

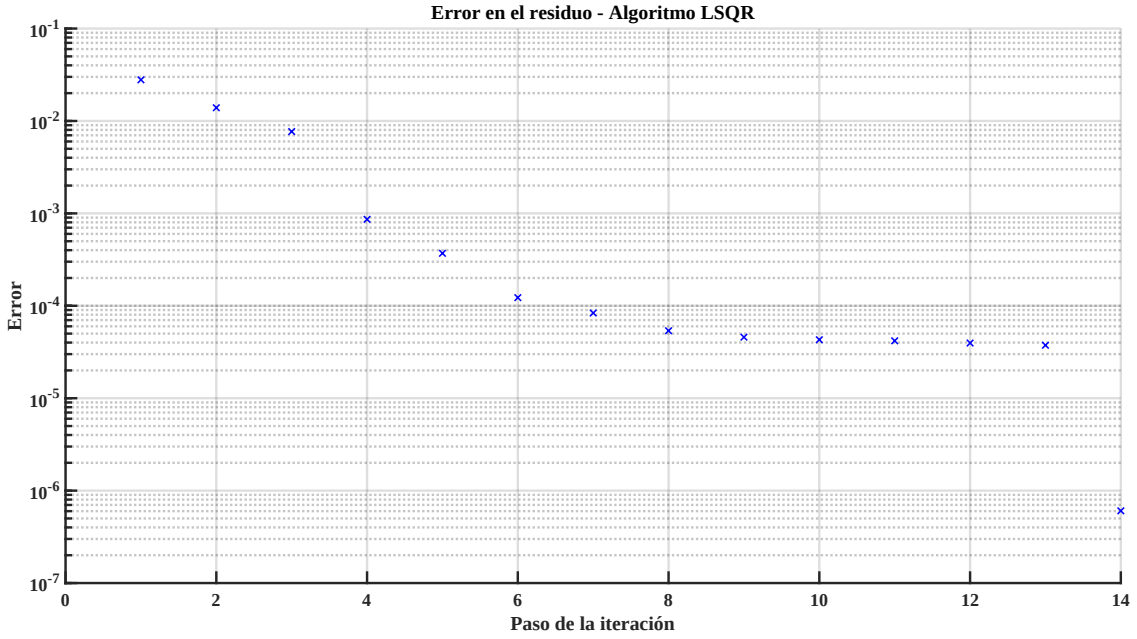

Figura 4.1: Soluciones aproximadas del sistema $Ax = b$ para m=15 iteraciones del método del gradiente conjugado.

Si representamos dicho error en ejes semilogarítmos, podemos obtener la figura 4.1. Se ha finalizado tras un total de 14 iteraciones, pues se alcanzó la tolerancia máxima especificada tol $= 10^{-16}$.

Apéndice A

Códigos de Matlab Utilizados

Este Apéndice contiene los programas y algoritmos implementados en Matlab que se han utilizado a lo largo del estudio del algoritmo de Lanczos.

Capítulo 2

Introducción al Método de Lanczos

En esta sección se recogen los programas utilizados a lo largo del capítulo 2.

Sección 2.2

Algoritmo de Lanzos

```
function [alpha,beta,Q]=Lanczos(A,q,m)
% A matriz , vector aleatorio q, número de interaciones m
    Q(:,1) = q/norm(q); % Normalizamos el vector q
    for k=1:m
        v= A * Q(:,k);
        alpha(k) = Q(:,k)'*v;
        v=v-alpha(k)*Q(:,k);
            if k>1 % equivalente a poner q_0=0
               v=v-beta(k-1)*Q(:,k-1);
            end
            if k< m % para no calcular q_(m+1)
                beta(k)=norm(v);
                Q(:,k+1) = v/beta(k);
            end
    end
    % Creamos la matriz trigiagonal T con alpha y beta
    T=full(gallery('tridiag',beta(1:end),alpha(1:end),beta(1:↩
       end)));
    clear k; clear v;  clear q; % Eliminamos var. que no ↩
       usaremos más
```

```
    filename = 'resultados.mat'; %Guardo A, Q, alpha,beta y T
    save(filename) %Guardamos el resultado en un archivo .mat

  %FIGURA

end
```

Evidentemente, el algoritmo anterior ha sido programado de forma que se almacenen todos los vectores q_i, pues los requerimos para un posterior análisis del resultado. Sin embargo, en cada iteración únicamente son necesarios q_k y q_{k-1}, luego se puede sustituir $Q(:, k-1)$ y $Q(;, k)$ por $q1$ y $q2$ respectivamente si no se desea almacenar todos los q_i. Lo mismo ocurre con los valores de α y β, una vez han sido añadidos a la matriz T_k, pueden ser eliminados, de forma que ponemos sustituir α y β como vectores por escalares.

Si queremos crear gráficos similares a las figuras 2.2 o 2.3, no tenemos más que insertar el siguiente fragmento de código en el programa anterior, donde *exacta* es un vector que contiene los autovalores exactos de la matriz en cuestión

```
for i=1:m %Calcular ahora los autovalores aproximados
        [V,D] = eig(T(1:i,1:i)); %Da la matriz D diagonal y V
                                 %tal que V^-1 * T * V = D
        [Ds,Is] = sort(-diag(D)); %Creamos una lista indexada ↩
           de los autovalores Ds
        Autoval(1:i,i) = -Ds; %Guardo los autov. aproximados ↩
           en cada interación en las columnas de Autoval
    end

    figure(1) %Creamos la gráfica
    hold on
    for i=1:m  %i representa la interación realizada
        s=(i*ones([1,i]))'; %Creamos un vector de dimension i
        scatter(s,Autoval(1:i,i),'x'); %En x=i dibujamos los i↩
            autov.(col. i de Autoval)
    end
    %Añadimos una recta vertical que separe los aproximados de↩
        exactos
    x=[m+1,m+1];
    y=[-4,4]; %Ajustar para cubrir el espectro de A
    plot(x,y) %Recta vertical en x=m+1

    %Añadimos con cruces los autovalores exactos en x=m+2
    i=((m+2)*ones([1,size(A)])); %Creo un vector de 1000 en
    scatter(i,exacta,'x','k');
    hold off
    grid on
    grid minor
    axis([0 inf -4 4]) %Ajustar a ejes apropiaos
```

```
    title([int2str(m),' iteraciones de Lanczos aplicados a la ↩
       matriz A'])
    xlabel('Paso de la iteración')
    ylabel('Autovalor aproximado')
```

Sección 2.4

Tabla 2.4

El siguiente código ha sido utilizado para generar la tabla 2.1.

```
function []=comparar()
K=[5,10,15,20,25,30];
cociente_lambda=[2,1.5,1.3,1.1,1.01];
%format longG
L=zeros(length(cociente_lambda)+1,length(K)+1);
R=zeros(length(cociente_lambda)+1,length(K)+1);
for i=1:length(cociente_lambda)
    L(i+1,1) = cociente_lambda(i);
    R(i+1,1) = cociente_lambda(i);
end

for j=1:length(K)
    L(1,j+1) = K(j);
    R(1,j+1) = K(j);
end

for i=1:length(K)
    for j=1:length(cociente_lambda)
        L(j+1,i+1) = 1/( chebyshev(K(i)-1,2*cociente_lambda(j)↩
           -1))^2;
    end
end

for i=1:length(K)
    for j=1:length(cociente_lambda)
        R(j+1,i+1) = (1/cociente_lambda(j))^(2*(K(i)-1));
    end
end

filename = 'comparacion_potencia.mat';
save(filename)
end
function T=chebyshev(n,x) %n=0,1,2,3,...
    m=length(x);
    T=zeros(1,m);
    T1=ones(1,m);
    T2=x;
```

```
    if n==1
        T=T2;
    else
        for j=2:n
            T=2*x.*T2-T1;
            T1=T2;
            T2=T;
        end
    end
end
```

Sección 2.4

Reortogonalización Completa

```
function []=reorto_completa(A,q,m) %Introducimos la matriz
%  matriz A , un vector q y el número de interaciones m
    Q(:,1) = q/norm(q); %Normalizamos el vector dado, y pasa a↩
        ser q1
    for k=1:m
        v= A * Q(:,k);
        alpha(k) = Q(:,k)'*v;
        v=v-alpha(k)*Q(:,k);
            if k>1 %equivalente a poner q_0=0
               v=v-beta(k-1)*Q(:,k-1);
               v=v-Q*Q'*v;
            end
            if k< m %para no calcular q_(m+1)
                beta(k)=norm(v);
                Q(:,k+1) = v/beta(k);
            end
    end
    %Creamos la matriz trigiagonal T con alpha y beta
    T=full(gallery('tridiag',beta(1:end),alpha(1:end),beta(1:↩
       end)));
    [V,D] = eig(T);
    S=Q'*Q; %mide si hay pérdida de ortogonalidad
    filename = 'reortonorm.mat'; %Guardo los resultados
    save(filename)
    %load('reortonorm.mat')
end
```

Sección 2.4

Reortogonalización Selectiva

```
function []=orto_selectiva(A,q,m) %Introducimos la matriz ,
% A , un vector aleatorio q y el número de interaciones m
    Q(:,1) = q/norm(q); %Normalizamos el vector dado, y pasa a↩
        ser q1
    w=0; %contamos el n° de ortogonalizaciones
    for k=1:m
        v= A * Q(:,k);
        alpha(k) = Q(:,k)'*v; %es el elpha k
        v=v-alpha(k)*Q(:,k);
            if k>1 %
               v=v-beta(k-1)*Q(:,k-1); %es el rk
            end

       if k>1
       T=full(gallery('tridiag',beta(1:end),alpha(1:end),beta↩
          (1:end)));
       [S,theta] = eig(T);
       beta(k)=norm(v);
       for j=1:k
            if abs(beta(k)*S(k,j)) < sqrt(eps(1))*norm(T)
                w=w+1;
                l= Q*S(:,j);
                v=v-(l'*v)*l;
            end
       end
       end
       if k< m %para no calcular q_(m+1)
                beta(k)=norm(v);
                Q(:,k+1) = v/beta(k);
       end
    end
    [V,D] = eig(T);
    S=Q'*Q; %comprobamos si S \approx I
    filename = 'reortono_selectiva.mat'; %Guardamos A, Q, ↩
       alpha,beta y T
    save(filename)
end
```

Capítulo 3

Resolución de Sistemas Lineales

En esa parte del anexo recopilaremos los programas utilizados a lo largo del capítulo 3

Sección 3.1

Resolución de Sistemas Lineales. Versión 2.

```
function [x]=lanc_sist(A,y,b,m)
% y= x0, m nuúmero máximo de iteraciones
% programa que resuelve el sistema lineal Ax=b
x(:,1)=y;
r0=b-A*y;
beta0=norm(r0);
Q(:,1)=r0/beta0;
    for k=1:m
        v= A * Q(:,k);
        alpha(k) = Q(:,k)'*v;
        v=v-alpha(k)*Q(:,k);
            if k>1 % equivalente a poner q_0=0
               v=v-beta(k-1)*Q(:,k-1);
            end
            beta(k)=norm(v);
            if k< m % para no calcular q_(m+1)
                if k==1
                    d(1)=alpha(1);
                    c(:,1)=Q(:,1);
                    rho(1)= beta0/alpha(1);
                    x(:,2)=y+rho(1)*Q(:,1);
                else
                    l(k-1)= beta(k-1)/d(k-1);
                    d(k)=alpha(k)-beta(k-1)*l(k-1);
                    c(:,k)=Q(:,k)-l(k-1)*c(:,k-1);
                    rho(k)=-l(k-1)*d(k-1)*rho(k-1)/d(k);
                    x(:,k+1)=x(:,k)+rho(k)*c(:,k);
                    end
             Q(:,k+1) = v/beta(k);
             end
    end
    figure(1) % Creamos la gráfica
    hold on
     for i=1:m % i representa la interación realizada
        for j=1:length(b)
            if (rem(j,4)==1), % para ir dibujando en cada
                scatter(i-1,x(j,i),'bx'); % iteración los x_{i}
            elseif (rem(j,4)==2), % del mismo color
                scatter(i-1,x(j,i),'gx');
            elseif (rem(j,4)==3),
                scatter(i-1,x(j,i),'mx');
            elseif (rem(j,4)==0),
                scatter(i-1,x(j,i),'rx');
            end
```

```
        end
    end
    %Añadimos una recta vertical que separe los aproximados ↩
        de exactos
    x1=[m,m];
    y1=[-0.2,1];
    plot(x1,y1) %Recta vertical en x=m+1
    %Dibujo la solución exacta
    scatter((m+2)*ones(length(b),1),exacta, 'kx')
    xticks([0 2 4 6 8 10 12 14 16 18])
    xticklabels({'0',' 2', '4', '6', '8' , '10' , '12' , '14' ,↩
        '16', 'Exacto' })
    grid on
    grid minor
    title(['Solución del sistema lineal tras ' int2str(m-1),' ↩
       iteraciones'])
    xlabel('Paso de la iteración')
    ylabel('Solución aproximada')
    hold off

    %Errores cometidos
    figure(2)
    hold on
    e=repmat(exacta',1,size(x,2)); %copiamos la solucion exacta↩
        m veces
    error=x-e;
    maxe=max(abs(error),[],1);
    in=1:(m-1);
    scatter(in(1:m-1),maxe(1:m-1),'x');
    grid on
    grid minor
    title(['Error máximo en cada iteración'])
    xlabel('Paso de la iteración')
    ylabel('Error máximo')
    hold off


    filename = 'sis_lineal.mat';
    save(filename)
end
```

Sección 3.2

Resolución de Sistemas Lineales. Máximo descenso.

```
function [x]=max_descenso(A,y,b,m)
```

```
x(:,1)=y;
r(:,1)=b-A*y;
 for k=1:m
        alpha(k) = (r(:,k)'*r(:,k))/(r(:,k)'*A*r(:,k));
        x(:,k+1)=x(:,k)+alpha(k)*r(:,k);
        r(:,k+1)=b-A*x(:,k+1);
 end

    figure(1) %Creamos la gráfica
    hold on
     for i=1:m %i representa la interación realizada
        for j=1:length(b)
            if (rem(j,4)==1), %para ir dibujando de forma que ↩
               en cada
                scatter(i-1,x(j,i),'bx'); %iteración los x_{i}↩
                    estén
            elseif (rem(j,4)==2), %del mismo color
                scatter(i-1,x(j,i),'gx');
            elseif (rem(j,4)==3),
                scatter(i-1,x(j,i),'mx');
            elseif (rem(j,4)==0),
                scatter(i-1,x(j,i),'rx');
            end
        end                                  %autov.(col. i de ↩
           Autoval)
     end
     %Añadimos una recta vertical que separe los aproximados ↩
        de exactos
    x1=[m,m];
    y1=[-0.2,1];
    plot(x1,y1) %Recta vertical en x=m+1
    %Dibujo la solución exacta
    for i=1:length(b) exacta(i)=1/(i); end
    scatter((m+2)*ones(length(b),1),exacta, 'kx')
    xticks([0 5 10 15 20 25 30 35 40 45 50 54])
    xticklabels({'0','5', '10', '15', '20' , '25' , '30' , '35'↩
        , '40', '45', '50', 'Exacto' })
    xlim([0 54])
    grid on
    grid minor
    title(['Solución del sistema lineal tras ' int2str(m-1),' ↩
       iteraciones'])
    xlabel('Paso de la iteración')
    ylabel('Solución aproximada')
    hold off


    figure(2)
    hold on
```

```
    e=repmat(exacta',1,size(x,2)); %copiamos la solucion exacta↩
        m veces
    error=x-e;
    maxe=max(abs(error),[],1);
    in=1:(m-1);
    scatter(in(1:m-1),maxe(1:m-1),'x');
    grid on
    grid minor
    title(['Error máximo en cada iteración'])
    xlabel('Paso de la iteración')
    ylabel('Error máximo')
    hold off

end
```

Sección 3.3

Cota de error del Método de Lanczos

En esta sección se pretende proporcionar un programa que realiza las gráficas 3.8 y 3.9. En él se utiliza un programa llamado *lanc_sist2* que es en esencia el código anteriormente escrito, eliminado la parte referente a las gráficas, y donde pedimos por entrada además la solución exacta.

```
function [err_k]=error_lineal(A,x0,b,k,exacta)
x=lanc_sist2(A,x0,b,k,exacta);
%x0 iterante inicial, b A*x=b, k num.iter
kappa=cond(A);
dif=repmat(exacta,1,k+1)-x;
for i=1:k err_k(i)=log10(normA(dif(:,i),A)); end
figure(1)
hold on
scatter(1:k,err_k,60, 'kx','LineWidth',2)

%Dibujo la línea teórica
a=log10(abs(2*normA(exacta(:,1)-x(:,1),A)));
m=log10((sqrt(kappa)-1)/(sqrt(kappa)+1));
w=0:0.01:k;
y=a+m*w;
plot(w,y,'LineWidth',2)
grid on
    grid minor
    set(gca,'FontSize',20)
    title('Convergencia de la solución del ejemplo 3.1 - Método↩
        de Lanczos');
    xlabel('Paso de la iteración')
    ylabel('$$\bf{log_{10}(|| x-x_{k} || )_{A}}$$','interpreter↩
        ','latex')
```

```
hold off
set(gca,'TickLabelInterpreter','none');
set(gca,'fontweight','bold','fontsize',12);
set(gca,'FontSize',15);
H=gca;
H.LineWidth=2;
x0=10;
y0=10;
width=800;
height=200;
set(gcf,'position',[x0,y0,width,height])
set(gcf, 'PaperPosition', [0 0 40 20]);
set(gcf, 'PaperSize', [40 20]);
    saveas(gcf, 'error_31', 'pdf')
    hold off
end

function [norm]=normA(v,A)
norm=sqrt(v'*A*v);
 end
```

Capítulo 4

Problemas de mínimos cuadrados

En esa parte del anexo recopilaremos los programas utilizados a lo largo del capítulo 4.

Sección 4.2

Algoritmo LSQR

```
function [x]=LSQR(A,b,niter)
format long;
[m,n] = size(A); %m número de filas , y n columnas
u= b(1:m); %Problema min{Ax-b} con A es m x n
beta = norm(u);
tol=1e-16;
alfa= 0;
if beta > 0
   u = (1/beta) * u;
   v = A'*u;
   alfa = norm(v);
end
if alfa > 0
   v = (1/alfa)*v;
```

```
   w = v;
end
phibar = beta;
rhobar = alfa;
x= zeros(n,1); %Aquí guardaremos la solución
j=0;
while j < niter
    j=j+1;
    %bidiagonalizacion
      u     = A*v  -  alfa*u;
      beta  = norm(u);
      if beta > 0
         u     = (1/beta) * u; %divido entre beta_{i+1}
         v     = A'*u  -  beta*v;
         alfa  = norm(v);
         if alfa > 0,  v = (1/alfa) * v;end %divido entre ↩
            alpha_{i+1}
      end
    %Elimino beta_{k+1}
      rho = sqrt(rhobar^2 + beta^2);
      c      =   rhobar/ rho;
      s = beta / rho;
      theta   =   s * alfa;
      rhobar  = - c * alfa;
      phi     =   c * phibar;
      phibar  =   s * phibar;

    %Actualizo solución
      x       = x      +  (phi  /rho)*w;
      w       = v      - (theta/rho)*w;
   %Comprobamos si debemos parar
   RES=b-A*x;

   %Para evitar sistemas compatibles
   if norm(RES)< tol
       fprintf('%d Tolerancia alcanzada en iteraciones\n',j);
   return;
   end

   if norm(A'*RES) < tol*norm(A)*norm(RES)
       fprintf('%d Tolerancia alcanzada en iteraciones\n',j);
   return;
   end

end
filename='solucion_lsqr.mat';
save(filename);
end
```

Bibliografía

[1] James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[2] Gene H Golub y Charles F Van Loan. *Matrix computations*. Vol. 3. JHU press, 2013.

[3] Vicente Hernandez y col. "Restarted Lanczos bidiagonalization for the SVD in SLEPc". En: *STR-8, Tech. Rep.* (2007).

[4] David G Luenberger. *Introduction to linear and nonlinear programming*. Vol. 28. Addison-wesley Reading, MA, 1973.

[5] Christopher C Paige y Michael A Saunders. "LSQR: An algorithm for sparse linear equations and sparse least squares". En: *ACM Transactions on Mathematical Software (TOMS)* 8.1 (1982), págs. 43-71.

[6] Christopher C Paige y Michael A Saunders. "LSQR: Sparse linear equations and least squares problems". En: *ACM Transactions on Mathematical Software (TOMS)* 8.2 (1982), págs. 195-209.

[7] Christopher Conway Paige. "The computation of eigenvalues and eigenvectors of very large sparse matrices." Tesis doct. University of London, 1971.

[8] Beresford N Parlett. *The symmetric eigenvalue problem*. SIAM, 1998.

[9] Beresford N Parlett y David S Scott. "The Lanczos algorithm with selective orthogonalization". En: *Mathematics of computation* 33.145 (1979), págs. 217-238.

[10] Yousef Saad. "On the rates of convergence of the Lanczos and the block-Lanczos methods". En: *SIAM Journal on Numerical Analysis* 17.5 (1980), págs. 687-706.

[11] David St Clair Scott. *Analysis of the Symmetric Lanczos Process*. Inf. téc. CALIFORNIA UNIV BERKELEY ELECTRONICS RESEARCH LAB, 1978.

[12] Horst D Simon. "Analysis of the symmetric Lanczos algorithm with reorthogonalization methods". En: *Linear algebra and its applications* 61 (1984), págs. 101-131.