UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICA

Algunos Métodos Numéricos para la Descomposición en Valores Singulares de Matrices Reales

Trabajo Especial de Grado presentado ante la ilustre Universidad Central de Venezuela por el **Br. Ronaldys Y. Rosario M.** para optar al título de Licenciado en Matemática.
Tutor: Dra. Brígida Molina

Caracas, Venezuela

Mayo 2011

Nosotros, los abajo firmantes, designados por la Universidad Central de Venezuela como integrantes del Jurado Examinador del Trabajo Especial de Grado titulado "**Algunos Métodos Numéricos para la Descomposición en Valores Singulares de Matrices Reales**", presentado por el **Br. Ronaldys Y. Rosario M.**, titular de la Cédula de Identidad **18.323.995**, certificamos que este trabajo cumple con los requisitos exigidos por nuestra Magna Casa de Estudios para optar al título de **Licenciado en Matemática**.

Dra. Brígida Molina
Tutor

Dra. Zenaida Castillo
Jurado

Dr. Juan Guevara
Jurado

Dedicatoria

Es un gran honor presentar este trabajo
en dedicación a nuestro Creador Jehová Dios,
a mis Padres Pedro y Norma, y demás familiares.

Agradecimientos

"¡Oh la profundidad de las riquezas y de la sabiduría y del conocimiento de Dios![...]"(Romanos 11:33)

Primeramente vayan las gracias a Jehová, quien es la fuente de toda sabiduría y entendimiento.

A mis padres, Pedro Domingo Rosario y Norma de Rosario, quienes le dieron origen a mi vida, amorosamente me han dado su apoyo y han dirigido todos sus esfuerzos con el fin de que alcanzara el éxito en mi carrera.

A mi tutora Brígida Molina, quien fue un soporte en la realización de este trabajo.

Al Prof. Juan Guevara, quien formó parte del jurado examinador de esta investigación y que, con sus palabras reconfortantes, me dió ánimos en seguir adelante.

A la Prof. Zenaida Castillo por todo el apoyo y disponibilidad.

Al Prof. Mauricio Ángel, quien con su esfuerzo permitió que la defensa se realizara satisfactoriamente.

A mi gran amiga Rosa Díaz, quien me ha regalado su amistad, cariño incondicional; me dió fuerzas y estímulo hasta el último instante de este

camino recorrido.

A mis grandes compañeros de clases y amigos: Odalis Mejía, Zorely Jesús, Iris Lozano, Dugliany, Karelys Medina, Roberto Morillo, Gari Roa, Pablo Fory, Tomás Hernandez y Thaiser Barrantes, ni se imaginan lo profundamente agradecido que estoy con ustedes por todo el apoyo que me ofrecieron de manera incondicional durante todo este período. Así mismo, a Alejandra Granadillo, Ysamar Meza, Freysimar Solano, Carlos Gonzalez, Ángel Padilla, Frank Prieto, Amanda Escorche y Hovsep Pernalette, gracias por la ayuda impartida, por el interés que, de una u otra forma, me demostraron. A Aida Fagundez, por haberme proporcionado estímulo en esos momentos de angustia. Tambien a Miriam Prieto por su positivismo y cariño hacia mi. A Aleida Poleo, Neysa Martinez, Vanessa Hernandez, Yolanda Parga, Jannette Iglesias, Carlos Iglesias y todas aquellas personas quienes oraron a Jehová con la intención de que todo culminara gratamente.

A todos ustedes y a los que no mencioné tambien ¡Gracias!.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

RESUMEN

En el siguiente trabajo se presentan diversos métodos numéricos para la obtención de la Descomposición en Valores Singulares (SVD) de Matrices Reales. Todos son basados en el problema simétrico de autovalores. Para cada uno de ellos se describe detalladamente su estructura y correspondiente algoritmo en lenguaje pseudoformal. Además, se muestran teoremas que avalan la convergencia de algunos de estos métodos. Posteriormente, se exhiben ciertos resultados numéricos para el análisis de la eficiencia en tiempo de ejecución y alta precisión. Finalmente, se muestra una aplicación de este tipo de descomposición para la compresión digital de imágenes.

INTRODUCCIÓN

En el área del álgebra lineal numérica existe una forma de descomposición de matrices a valores complejos o reales que ha logrado aportar diversos beneficios en el momento de resolver, de manera eficaz, un problema de índole numérico. Dicha forma es llamada Descomposición en Valores Singulares (SVD, sus siglas en inglés). Entre los problemas en los cuales la SVD gana importancia se pueden mencionar: el cálculo del rango y el número de condición de una matriz, la obtención de la pseudoinversa, la solución a un sistema de ecuaciones lineales, la solución a un problema de mínimos cuadrados lineales, el cálculo de la distancia de una matriz al conjunto de matrices de rango menor que ella, por citar algunas. Por otro lado, la SVD permite ampliar el conjunto de las matrices a las cuales son admitidas una descomposición en autovalores.

Como es costumbre en matemáticas, no fue la necesidad práctica sino la necesidad de profundizar en el conocimiento lo que dió pie al desarrollo de los valores singulares. En efecto, fue en la segunda parte del siglo XIX cuando algunos geómetras se preguntaron por la posibilidad de reducir unitariamente la forma cuadrática $f(x, y) = x^t Ay$ a una forma diagonal, donde A es una matriz real de orden n, entre los que se destacan: Eugene Beltrami, Camille Jordan, James Sylvester, entre otros. Para aquella época, Carl Jacobi introdujo un método para el problema simétrico de autovalores que luego fue adaptado para

crear un mecanismo acorde al problema de valores singulares. Posteriormente, Wallace Givens, en 1953 inició por primera vez el método Bisección para el cálculo de autovalores, con el que nuevamente permitió que se desarrollara una estrategia para la SVD, pero en este caso, basado en ese método. En 1961, John Francis dió el primer aporte a la técnica Iteración QR, el cual sirvió de medio para que Gene Golub, William Kahan, y Christian Reinsch implementaran, en 1970, un algoritmo standard para la obtención de la SVD. Para 1981, Cuppen introdujo por primera vez el método Divide y Vencerás, para el cual se planteó un algoritmo SVD basado en dicha estrategia. Luego, para 1994, K. V. Fernando y B. N. Parlett plantearon un método eficiente para el cálculo de los valores singulares, este fue una variante del método de Iteración LR propuesto por Rutishauser en 1954. Algunos de estos métodos numéricos para la obtención de la SVD se presentarán en la siguiente investigación, y otro para el cálculo de los valores singulares, de manera eficiente y con alta precisión.

El contenido de este trabajo especial de grado está estructurado de esta manera: en el **capítulo 1** se presentan algunas definiciones, proposiciones, conceptos y algoritmos necesarios para el entendimiento de los capítulos siguientes. En el **capítulo 2** se exhiben diversos aspectos teóricos asociados con la SVD, entre las que se destacan: una breve reseña histórica, su interpretación geométrica, definición formal, la garantía de su existencia y unicidad, algunas propiedades relevantes, su importancia sobre el problema del cálculo de la descomposición en autovalores, y un teorema acerca de la sensibilidad de los valores singulares ante perturbaciones. En el **capítulo 3** se muestran tres métodos para el cálculo de la descomposición en autovalores, y otro únicamente de los valores propios, para matrices tridiagonales simétricas. Posteriormente, en el **capítulo 4** se detallan seis mecanismos para la obtención de la SVD, y otro solamente para el cálculo de los valores singulares, algunos de los cuales son basados en los métodos del capítulo 3. El **capítulo 5** corresponde a los resultados numéricos que avalan la eficiencia y precisión de los métodos descritos en el capítulo 4, en donde se muestran diversos cuadros y gráficos compara-

tivos para su mejor análisis. Además, exalta la importancia de la SVD en la computación gráfica, para la compresión digital de imágenes. Finalmente, se presentan las conclusiones respectivas a los resultados númericos obtenidos.

CAPÍTULO 1

PRELIMINARES

En este capítulo se presentan diversas definiciones, proposiciones y algoritmos básicos que servirán como herramientas útiles para la comprensión de los temas que se desarrollarán en los capítulos posteriores.

Definición 1.1. *Una matriz $U \in \mathbb{C}^{n\times n}$ es unitaria si sus columnas forman una base ortonormal de vectores de $\mathbb{C}^n$.*

Definición 1.2. *Una matriz no-singular $P \in \mathbb{R}^{n\times n}$ es una matriz ortogonal si*

$$P^{-1} = P^t.$$

Proposición 1.3. *Para $U \in \mathbb{C}^{n\times n}$ las siguientes condiciones son equivalentes:*

1. *U es unitaria.*
2. *U es no-singular y $U^* = U^{-1}$.*
3. *$UU^* = I_n$, donde I_n es la matriz identidad de dimensión n.*
4. *U^* es unitaria.*
5. *Las filas de U forman un sistema ortonormal de vectores de $\mathbb{C}^n$.*
6. *Para todo $x \in \mathbb{C}^n$ se tiene $\|x\|_2 = \|Ux\|_2$.*

Para detalles de la demostración véase [21, p.84].

Definición 1.4. *Una norma $\|\cdot\|$ en $\mathbb{C}^{m\times n}$ se dice que es unitariamente invariante si para todo $A \in \mathbb{C}^{m\times n}$ y para todo par de matrices unitarias $U \in \mathbb{C}^{m\times m}$ y $V \in \mathbb{C}^{n\times n}$ se cumple que $\|UAV\| = \|A\|$.*

Definición 1.5. *Se define la norma Frobenius de una matriz A de $n\times n$ como*

$$\|A\|_F = \left[\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2\right]^{1/2}.$$

Proposición 1.6. *Las normas $\|\cdot\|_2$ y $\|\cdot\|_F$ definidas en $\mathbb{C}^{m\times n}$ son unitariamente invariantes.*

Para detalles de la demostración refiérase a [30].

Definición 1.7. *Sea E un espacio vectorial Euclídeo o unitario de dimensión finita y S un subconjunto de E. Se define como un subespacio ortogonal de S al subconjunto*

$$S^{\perp} = \{v \in E \quad / \quad u\cdot v = 0 \ \textit{para todo}\ u \in S\}.$$

Definición 1.8. *Sea S una matriz no-singular. Entonces A y $B = S^{-1}AS$ son llamadas matrices similares y S es una transformación de similaridad.*

Definición 1.9. *Sea $A \in \mathbb{R}^{n\times n}$. Se dice que un escalar $\lambda \in \mathbb{R}$ es un autovalor de A si existe un vector $v \in \mathbb{R}^n$, $v \neq 0$, tal que $Av = \lambda v$, en cuyo caso se dice que v es un autovector de A asociado al autovalor λ.*

Definición 1.10. *(Convergencia de orden α). Sea $\{\vec{x}_n\}_{n\in\mathbb{N}}$ una sucesión de vectores convergente a $\vec{x}$. Para $\alpha \geq 1$, $\alpha \in \mathbb{R}$, se dice que la sucesión converge a $\vec{x}$ con orden α, si existen $C > 0$, $K \in \mathbb{N}$ tales que*

$$\|\vec{x}_{n+1} - \vec{x}\| \leq C\|\vec{x}_n - \vec{x}\|^{\alpha}, \ \textit{para}\ n \geq K,$$

siendo $C < 1$ cuando $\alpha = 1$; en este último caso se dice que la convergencia es lineal. Si $\alpha = 2$ se dice que la convergencia es cuadrática y si $\alpha = 3$ la sucesión converge cúbicamente.

1.1 Matriz de Rotación

Definición 1.11. *Una matriz de la forma*

$$G(j,k,\theta) = \begin{array}{c} \\ j \\ \\ k \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & j & & k & \end{array} \\ \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \end{array}$$

es denominada una Rotación de Givens o Matriz de Rotación, donde $c = cos(\theta)$ *y* $s = sen(\theta)$*, para algún* θ *y* $s^2 + c^2 = 1$*.*

Observación 1.12. *El hecho de que* $s^2 + c^2 = 1$ *implica que* $G(j,k,\theta) \cdot G(j,k,\theta)^t = I$*, y así* $G(j,k,\theta)$ *es una matriz ortogonal.*

Sea $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$*, entonces, se puede verificar que si*

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \tag{1.1}$$

y

$$s = \frac{x_2}{\sqrt{x_1^2 + x_2^2}} \tag{1.2}$$

entonces la Rotación de Givens $G(1,2,\theta) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ *es tal que* $G(1,2,\theta)x = \begin{pmatrix} * \\ 0 \end{pmatrix}$*, donde* $*$ *es un elemento no nulo.*

En el capítulo 4 se mostrará un tipo de matriz ortogonal, esencialmente útil para introducir una gran cantidad de ceros en un vector. Sin embargo, cuando sólo es necesario hacer ceros unas pocas entradas, las *Rotaciones de Givens* resultan ser las transformaciones más convenientes a utilizar. Por lo tanto,

cuando un vector $x = (x_1, \ldots, x_n)^t \in \mathbb{R}^n$ es premultiplicado por $G(j, k, \theta)$, sólo las componentes j-ésima y k-ésima de x son afectadas; las $n - 2$ componentes restantes se mantienen inalteradas.

Para calcular c y s, las fórmulas (1.1) y (1.2) podrían causar algún cero numérico o división por cero. Sin embargo, el siguiente algoritmo permite evitar tal eventualidad:

Algoritmo 1.1. Calculando los parámetros de Givens

Entrada: *vector* $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^{2\times 1}$.

Salida: *Parámetros* c *y* s.

Si $|x_2| \geq |x_1|$

$t = \dfrac{x_1}{x_2}$

$s = \dfrac{1}{\sqrt{1+t^2}}$

$c = s \cdot t$

Fin Si

Si $|x_2| < |x_1|$

$t = \dfrac{x_2}{x_1}$

$c = \dfrac{1}{\sqrt{1+t^2}}$

$s = c \cdot t$

Fin Si

Este algoritmo requiere de 4 operaciones y una raíz cuadrada.

A continuación se muestra cómo premultiplicar una matriz A por una Rotación de Givens ($A = G(j, k, \theta) \cdot A$) sin formar, esta última, explícitamente:

Algoritmo 1.2. Premultiplicación por una Rotación de Givens

Entrada: *matriz* $A \in \mathbb{R}^{m\times n}$, *parámetros* c *y* s *e índices* j *y* k.

Salida: *matriz* $A \in \mathbb{R}^{m\times n}$

Para $i = 1, \ldots, n$ **Hacer**

$a = a_{ji}$

$b = a_{ki}$

$a_{ji} = a \cdot c + b \cdot s$

$a_{ki} = -a \cdot s + b \cdot c$

Fin Para

Este algoritmo requiere de $4n$ operaciones. Además, se puede notar que la matriz A resultante es igual a la matriz A de entrada, salvo la fila j y la fila k, es por ello que esta rutina sólo actualiza dos filas de la matriz A. Por otro lado, se puede implementar de manera análoga, una rutina similar que permita postmultiplicar una matriz A por una matriz de Givens ($A = A \cdot G(j, k, \theta)$):

Algoritmo 1.3. Postmultiplicación por una matriz de Givens

Entrada: *matriz* $A \in \mathbb{R}^{m \times n}$*, parámetros* c *y* s *e índices* j *y* k*.*

Salida: *matriz* $A \in \mathbb{R}^{m \times n}$

Para $i = 1, \ldots, n$ **Hacer**

$a = a_{ij}$

$b = a_{ik}$

$a_{ij} = a \cdot c + b \cdot s$

$a_{ik} = -a \cdot s + b \cdot c$

Fin Para

Este algoritmo tiene la misma cantidad de operaciones que el Algoritmo 2 y actualiza sólo las columnas j y k de la matriz A.

CAPÍTULO 2

DESCOMPOSICIÓN EN VALORES SINGULARES (SVD)

En este capítulo se desarrollan algunos aspectos concernientes a la Descomposición en Valores Singulares (SVD, sus siglas en inglés) de una matriz general A de tamaño $m \times n$, con $m \geq n$. Se presenta una breve reseña histórica, su interpretación geométrica, definición, estructura, y propiedades, así como también cómo varían los valores singulares ante perturbaciones de la matriz y, finalmente, se muestra la importancia de la SVD en relación al problema de autovalores. Para mayores detalles véase [7, 8, 12, 13, 21, 23, 27].

2.1 Reseña Histórica

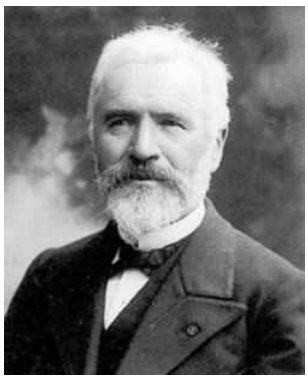
El origen de los valores singulares se encuentra en el intento de los geómetras del siglo XIX por conseguir, en lenguaje actual, la reducción de una forma cuadrática a forma diagonal mediante cambios ortogonales. La primera contribución en este sentido parece ser de Eugene Beltrami (1835-1899), un geómetra diferencial italiano. Él comenzó con una forma bilineal $f(x, y) = x^t Ay$, donde A es una matriz real de orden n, y demostró que existen matrices ortogonales Q_1 y Q_2 de $n \times n$, tales que

$$Q_1^t A Q_2 = \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \tag{2.1}$$

es una matriz diagonal de números reales positivos, los cuales son las raíces cuadradas de los autovalores de las matrices simétricas semejantes A^tA y AA^t. En realidad, Beltrami propuso, sin demostrar, la descomposición (2.1) cuando los elementos diagonales son todos distintos entre sí (véase [5]).

Eugene Beltrami
(1835-1899)

Más tarde, Camille Jordan (1831-1921) publicó un par de trabajos sobre formas bilineales [14, 15], por lo que se le puede considerar codescubridor de los valores singulares.



Camille Jordan
(1831-1921)

Su aproximación es completamente diferente. De hecho, en el primero de los artículos citados, su preocupación fue la de buscar el máximo y mínimo de la forma bilineal $P = x^tAy$ condicionados a que $\|x\|_2 = \|y\|_2 = 1$. Jordan probó que dicho máximo es la raíz cuadrada del mayor autovalor de la matriz $\begin{bmatrix} 0 & A \\ A^t & 0 \end{bmatrix}$.

Desconocedor, aparentemente, de los trabajos de Beltrami y Jordan, John Joseph Sylvester (1814-1897), escribió una nota a pie de página en un artículo de la revista *The Messenger of Mathematics* [25], en la cual propuso un algoritmo para reducir una forma cuadrática a una forma diagonal.

John Joseph Sylvester
(1814-1897)

Allí señaló que una implementación similar es posible para diagonalizar una forma bilineal e indicó que lo envió para su publicación en las *Comptes Rendues* [26]. Esto da una idea de que él no conocía el artículo original de Jordan publicado más de una década antes. En cualquier caso, Sylvester fue el primero en dar nombre a los números reales positivos que aparecen en la forma canónica diagonal: *multiplicadores canónicos*.


Erhard Schmidt (1876-1959)

Por otro lado, durante las primeras décadas del siglo XX, los valores singulares fueron redescubiertos en el área de las ecuaciones integrales, rama muy distinta a la que se había trabajado hasta entonces. En 1907, Erhard Schmidt (1876-1959) publicó una teoría general de ecuaciones integrales reales con núcleos simétricos y no simétricos [22].

Introdujo los conceptos de *valor propio* y de *función propia* de un núcleo $K(x, y)$ continuo en $[a, b]\times[a, b]$ mediante las ecuaciones integrales

$$u(x) = \lambda \int_a^b K(x, y)v(y)dy \quad \text{y} \quad v(y) = \int_a^b K(x, y)u(x)dx.$$

Luego, demostró que λ debe ser real porque λ^2 es un valor propio del núcleo simétrico definido positivo

$$H(x, y) = \int_a^b K(x, t)K(t, y)dt.$$

Si se piensa en $K(x, y)$ como lo análogo a la matriz A, entonces $H(x, y)$ es análogo a A^tA.

Finalmente, como otro de los muchos matemáticos notables que han trabajado en las propiedades de los valores singulares, cabe mencionar a Hermann Weyl (1885-1955). Este dió importantes contribuciones a la teoría de perturba-

ción de los mismos, pero en cuanto a nomenclatura se refiere, en su artículo, [28], citó "dos clases de valores propios".

Hermann Weyl
(1885-1955)

2.2 Interpretación geométrica

La SVD está motivada por el siguiente hecho geométrico:

La imagen de la esfera unitaria bajo cualquier matriz de $m \times n$ es una hiperelipse.

La SVD es aplicable tanto a matrices reales como complejas. Sin embargo, describiendo la interpretación geométrica, asúmase que la matriz es real.

El término "hiperelipse" puede no ser familiar, pero esto es sólo la generalización m-dimensional de una elipse. Se puede definir una hiperelipse en $\mathbb{R}^m$ como la superficie obtenida por estiramiento de la esfera unitaria en $\mathbb{R}^m$ por algunos factores $\sigma_1, \ldots, \sigma_m$ (posiblemente ceros) en algunas direcciones $u_1, \ldots, u_m \in \mathbb{R}^m$. Por conveniencia, tomése los u_i, con $i = 1, \ldots, m$, como vectores unitarios, es decir, $\|u_i\|_2 = 1$. Los vectores $\sigma_i u_i$ son los *semiejes principales* de la hiperelipse, con longitudes $\sigma_1, \ldots, \sigma_m$. Si A tiene rango r, exactamente r de las longitudes σ_i resultarán ser no nulas, y en particular, si $m \geq n$, a lo sumo n de ellas no serán ceros. Refiérase a la esfera unitaria como a la esfera usual del espacio Euclideano n-dimensional, la cual es denotada por S. Entonces AS, la imagen de S bajo la transformación A, es la hiperelipse que justamente se ha definido.

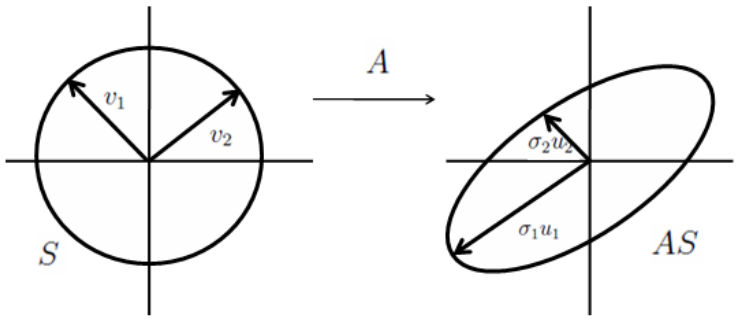


Figura 2.1: SVD de una matriz 2×2

Sea S la esfera unitaria en $\mathbb{R}^n$, y tómese cualquier $A \in \mathbb{R}^{m\times n}$ con $m \geq n$. Por simplicidad, supóngase que A tiene rango n. La imagen AS es una hiperelipse en $\mathbb{R}^m$. Ahora defínase algunas propiedades de A en términos de AS. Ver Figura 2.1.

En primer lugar, se denomina a los n *valores singulares* como las longitudes de los n semiejes principales de AS, escritos como $\sigma_1, \sigma_2, \ldots, \sigma_n$. Por convención, asúmase que los valores singulares están enumerados en orden decreciente, es decir, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$.

En segundo lugar, se denomina a los n *vectores singulares izquierdos* de A como los vectores unitarios $u_1, u_2, \ldots, u_n$ orientados en las direcciones de los semiejes principales de AS, enumerados con correspondencia a los valores singulares definidos anteriormente. Así, el vector $\sigma_i u_i$ es el i-ésimo semieje principal de AS.

Finalmente, se denomina a los n *vectores singulares derechos* de A como los vectores unitarios $v_1, v_2, \ldots, v_n \in S$ preimágenes de los semiejes principales de AS, enumerados tal que $Av_j = \sigma_j u_j$. Para mayores detalles acerca de la interpretación geométrica de la SVD puede verse [7, 8, 12, 27].

2.3 Descomposición en Valores Singulares

La condición $Av_j = \sigma_j u_j$, $j = 1, \ldots, n$, mostrada anteriormente, se puede escribir en forma matricial de la siguiente manera: Considérese $\hat{U} = [u_1 \cdots u_n]$ y $\hat{V} = [v_1 \cdots v_n]$, entonces se tiene que

$$A\hat{V} = \hat{\Sigma}\hat{U}, \quad \hat{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n)$$

siendo $\hat{U} \in \mathbb{R}^{m\times n}$ y $\hat{V} \in \mathbb{R}^{n\times n}$ matrices cuyas columnas son vectores ortonormales. Por consiguiente,

$$A = \hat{U}\hat{\Sigma}\hat{V}^t.$$

A esta factorización de A se le denomina *Descomposición en Valores Singulares Reducida* o *Económica de A*. Véase la figura 2.2.

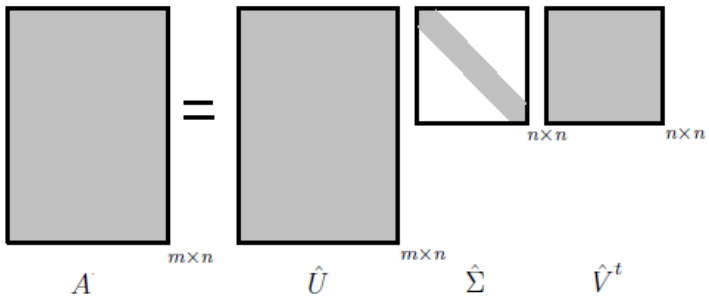


Figura 2.2: Representación de la Descomposición en Valores Singulares Reducida de A

Por otra parte, si $m \geq n$, $\hat{U}$ no es una matriz ortogonal y $\hat{\Sigma}$ no tiene el mismo tamaño de A. Sin embargo, se pudiese ampliar el sistema de vectores ortonormales $\{u_1, \ldots, u_n\}$ hasta una base ortonormal de $\mathbb{R}^m$. Tal estrategia siempre es posible porque los vectores $u_1, \ldots, u_n$ de $\mathbb{R}^m$ son linealmente independientes. Basta aplicar el método de Gram-Schmidt (se expone en el capítulo 3) para así obtener la base ortonormal deseada. Sea entonces $\{u_1, \ldots, u_n, u_{n+1}, \ldots, u_m\}$ una base ortonormal de $\mathbb{R}^m$ y considérese

$$U = \begin{bmatrix} u_1 & \cdots & u_n & u_{n+1} & \cdots & u_m \end{bmatrix} \quad \text{y} \quad \Sigma = \begin{bmatrix} \hat{\Sigma} \\ 0_{(m-n)\times n} \end{bmatrix}$$

por lo tanto

$$U\Sigma V^t = \begin{bmatrix} \hat{U} & \tilde{U} \end{bmatrix} \begin{bmatrix} \hat{\Sigma} \\ 0_{(m-n)\times n} \end{bmatrix} V^t = \hat{U}\hat{\Sigma}V^t = A.$$

A la expresión $A = U\Sigma V^t$ se le denomina *Descomposición en Valores Singulares Completa de* A (véase Figura 2.3).

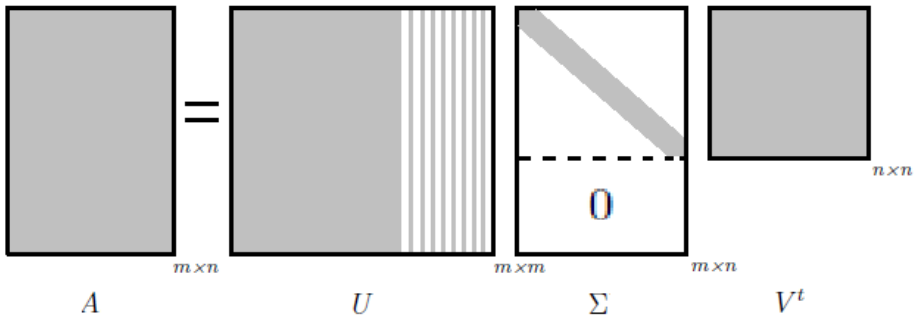


Figura 2.3: Representación de la Descomposición en Valores Singulares Completa de A

Nótese que de una Descomposición en Valores Singulares Completa de A se obtiene una Descomposición en Valores Singulares Reducida de A: $\hat{\Sigma}$ se obtiene suprimiendo las $m-n$ filas nulas de Σ y $\hat{U}$ resulta de eliminar las últimas $m-n$ columnas de U. Véase Figura 2.2 y 2.3.

Definición 2.1. *Sean* m *y* n *enteros positivos y* $A \in \mathbb{R}^{m\times n}$. *Una Descomposición en Valores Singulares (SVD) completa de* A *es una factorización de la forma*

$$A = U\Sigma V^t$$

donde

$$U \in \mathbb{R}^{m\times m} \text{ es ortogonal,}$$

$$V \in \mathbb{R}^{n\times n} \text{ es ortogonal,}$$

$$\Sigma \in \mathbb{R}^{m\times n} \text{ es diagonal.}$$

Además,

$$\Sigma = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix},$$

con $\Sigma_1 = \text{diag}(\sigma_1, \ldots, \sigma_p)$ *y* $p = \text{mín}(m, n)$.

Las entradas $\sigma_1, \ldots, \sigma_p$ *de la matriz diagonal* Σ_1 *son llamadas valores singulares de* A*. Estas entradas son no negativas y están en orden decreciente; esto es,* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$*, donde* $p = \text{mín}(m, n)$.

Las columnas de U *son llamadas vectores singulares izquierdos de* A *y las columnas de* V *o las filas de* V^t *son llamadas vectores singulares derechos de* A.

Observe que la matriz diagonal Σ preserva el mismo tamaño de la matriz A original, pero U y V son siempre matrices ortogonales cuadradas.

Es claro que la imagen de la esfera unitaria en $\mathbb{R}^n$ bajo una aplicación $A = U\Sigma V^t$ debe ser una hiperelipse en $\mathbb{R}^m$. La aplicación ortogonal V^t preserva la esfera, la matriz diagonal Σ contrae la esfera en una hiperelipse alineada con la base canónica, y la aplicación ortogonal U rota o refleja la hiperelipse sin alterar su forma. Así, si se prueba que toda matriz admite una SVD, entonces la imagen de la esfera unitaria bajo alguna aplicación lineal es una hiperelipse. Por ello, queda por establecer que tal descomposición es siempre posible y que los valores singulares están determinados de forma única por A.

2.4 Existencia y Unicidad de la SVD

Teorema 2.2. *(**Teorema SVD**) Toda matriz* $A \in \mathbb{R}^{m \times n}$ *admite una descomposición en valores singulares. Además, los valores singulares están determinados de forma única, y, si* A *es cuadrada y sus valores singulares son distintos,*

entonces los vectores singulares están también determinados de forma única salvo producto por un número complejo de módulo 1.

Demostración. Primero se demostrará la existencia de la SVD para una matriz $A \in \mathbb{R}^{m\times n}$. Si $A = 0$ entonces es inmediato la prueba ya que $I_{m\times m}0_{m\times n}I_{n\times n} = A$. Supóngase que $A \neq 0$, se procederá por inducción sobre n, el número de columnas de A, y además sea $m \geq n$. Si fuese $n \geq m$, y una vez demostrado el teorema con $m \geq n$, se pudiese aplicar a A^t. Así, existen matrices ortogonales U y V tales que $A^t = U\Sigma V^t$. Entonces $A = (A^t)^t = V\Sigma^t U^t$. Como los valores singulares son números reales, $\Sigma^t = \Sigma$ y $A = V\Sigma U^t$ con U y V ortogonales.

Sea entonces $n = 1$ y $m \geq 1$. Considérese $\hat{U} = \frac{1}{\|A\|_2}A$, $\hat{\Sigma} = \|A\|_2$ y $V = 1$. Así

$$\hat{U}\hat{\Sigma}V = \frac{1}{\|A\|_2}A\cdot\|A\|_2\cdot 1 = A.$$

Para $n = 1$, $A \in \mathbb{R}^{m\times 1}$ es un vector columna y por lo tanto $\hat{U}$ es un vector columna ortogonal. Así, $A = \hat{U}\hat{\Sigma}V$ es una descomposición reducida de A que puede extenderse a una descomposición completa tal como se mencionó anteriormente.

Considérese ahora que el teorema ha sido demostrado para matrices de tamaño $m\times p$ $(p \leq n-1)$. Sea $A \in \mathbb{R}^{m\times n}$ y $\sigma_1 = \|A\|_2$. Como $\|A\|_2 = \max_{\|x\|_2=1}\|Ax\|_2$ existe un vector unitario $v_1 \in \mathbb{R}^n$, $\|v_1\|_2 = 1$, tal que $\sigma_1 = \|A\|_2 = \|Av_1\|_2$. Sea $u_1 = \frac{1}{\|Av_1\|_2}Av_1$. Así, $\|u_1\|_2 = 1$ y $\|Av_1\| = \sigma_1 u_1$. Extendiendo u_1 y v_1 hasta bases ortonormales de $\mathbb{R}^m$ y $\mathbb{R}^n$, respectivamente, y sean U_1 y V_1 las matrices ortogonales cuyas columnas son los vectores de esas bases. Expresando

$$U_1 = \begin{bmatrix} u_1 & \hat{U}_1 \end{bmatrix}, \; V_1 = \begin{bmatrix} u_1 & \hat{V}_1, \end{bmatrix}.$$

entonces

$$U_1^t A V_1 = \begin{bmatrix} u_1^t \\ \hat{U}_1^t \end{bmatrix} A \begin{bmatrix} v_1 & \hat{V}_1 \end{bmatrix} = \begin{bmatrix} u_1^t A v_1 & u_1^t A \hat{V}_1 \\ \hat{U}_1^t A v_1 & \hat{U}_1^t A \hat{V}_1 \end{bmatrix}.$$

Por una parte, $Av_1 = \sigma_1 u_1$ implica que $u_1^t A v_1 = \sigma_1$ (recuerde que $u_1^t u_1 = 1$ porque u_1 es un vector unitario). Además, $\hat{U}_1^t A v_1 = \sigma_1 \hat{U}_1^t u_1$. Pero las columnas de $\hat{U}_1$ son ortogonales a u_1 y esto equivale a $\hat{U}_1^t u_1 = 0$. Así pues

$$U_1^t A V_1 = \begin{bmatrix} u_1^t A v_1 & u_1^t A \hat{V}_1 \\ 0 & \hat{U}_1^t A \hat{V}_1 \end{bmatrix}.$$

Veamos que también $u_1^t A \hat{V}_1 = 0$. Considérese $w^t = u_1^t A \hat{V}_1$ y $B = \hat{U}_1^t A \hat{V}_1$, $S = \hat{U}_1^t A V_1$ y $z = \begin{bmatrix} \sigma_1 \\ w \end{bmatrix}$. Como la norma 2 es consistente con la norma Euclídea

$$\begin{aligned} \|S\|_2 \|z\|_2 &\geq \|Sz\|_2 = \left\| \begin{bmatrix} \sigma_1 & w^t \\ 0 & B \end{bmatrix} \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \sigma_1^2 + w^t w \\ Bw \end{bmatrix} \right\|_2 \geq (\sigma_1^2 + w^t w) \\ &= (\sigma_1^2 + w^t w)^{\frac{1}{2}} \left\| \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2 = (\sigma_1^2 + w^t w)^{\frac{1}{2}} \|z\|_2. \end{aligned}$$

Así pues, $\|S\|_2 \geq (\sigma_1^2 + w^t w)^{\frac{1}{2}}$. Pero la norma espectral es unitariamente invariante; por lo tanto $\sigma_1 = \|A\|_2 = \|S\|_2 \geq (\sigma_1^2 + w^t w)^{\frac{1}{2}}$; lo cual implica que $w = 0$ tal y como se quería demostrar.

En consecuencia

$$U_1^t A V_1 = \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix}.$$

Debe notarse que B es la restricción de A al subespacio ortogonal a u_1; es decir, $<u_1>^{\perp}$. Además, $B \in \mathbb{R}^{(m-1)\times(n-1)}$. Por hipótesis inductiva, B admite una descomposición en valores singulares: $B = U_2 \Sigma_2 V_2^t$ con $U_2 \in \mathbb{R}^{(m-1)\times(m-1)}$

y $V_2 \in \mathbb{R}^{(n-1)\times(n-1)}$ ortogonales y $\Sigma_2 = \begin{bmatrix} \text{diag}(\sigma_2, \ldots, \sigma_n) \\ 0 \end{bmatrix}$. Así,

$$\begin{bmatrix} 1 & 0 \\ 0 & U_2^t \end{bmatrix} U_1^t A V_1 \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & U_2^t \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix} = \begin{bmatrix} \text{diag}(\sigma_2, \ldots, \sigma_n) \\ 0 \end{bmatrix}.$$

Si se considera

$$U^t = \begin{bmatrix} 1 & 0 \\ 0 & U_2^t \end{bmatrix} U_1^t \text{ y } V = V_1 \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix},$$

se tiene que $U^t A V = \Sigma$ y $A = U \Sigma V^t$. Esto prueba la existencia de la descomposición de A en valores singulares, excepto el ordenamiento de los valores singulares. Según la hipótesis de inducción, los valores singulares de B están ordenados de mayor a menor. Basta entonces demostrar que $\sigma_1(A) \geq \sigma_1(B)$. Es decir, $\|A\|_2 \geq \|B\|_2$, o bien, $\max\limits_{\|x\|_2=1} \|A\|_2 \geq \max\limits_{\|x\|_2=1} \|B\|_2$. Además, como la norma espectral es unitariamente invariante entonces se puede suponer que

$$A = \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix}.$$

Sea $x_0 \in \mathbb{R}^{n-1}$ un vector unitario para el que $\|Bx_0\|_2 = \max\limits_{\|x\|_2=1} \|Bx\|_2$ y sea

$$y = \begin{bmatrix} 0 \\ x_0 \end{bmatrix} \in \mathbb{R}^n.$$

Claramente, $y^t y = x_0^t x_0 = 1$, de modo que

$$\max_{\|x\|_2=1} \|Ax\|_2 \geq \|Ay\| = y^t A^t A y = x_0^t B^t x_0 = \|Bx_0\| = \max_{\|x\|_2=1} \|Bx\|_2,$$

tal y como se deseaba demostrar.

Ahora para demostrar que si A es cuadrada y sus valores singulares son todos distintos, entonces los vectores singulares están también determinados

de forma única salvo producto por un número complejo de módulo 1, se debe recordar lo siguiente sobre los valores propios de una matriz: Si $M \in \mathbb{C}^{n\times n}$ y sus valores propios son distintos dos a dos, entonces M admite un sistema completo de vectores propios linealmente independientes. Esto es una consecuencia de que a valores propios distintos corresponden vectores propios linealmente independientes. Si M tiene n valores propios distintos, hay n vectores propios linealmente independientes; y como están en un espacio de dimensión n, deben ser una base. Ahora bien, si v_i es un vector propio asociado al valor propio λ_i entonces $Mv_i = \lambda_i v_i$, y cualquier otro vector propio w_i asociado al mismo valor debe ser proporcional a v_i; es decir, existe $\alpha \in \mathbb{C}$ tal que $w_i = \alpha v_i$. Ahora, si $T = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$ entonces $T \in \mathbb{C}^{n\times n}$ es invertible y

$$T^{-1}MT = \operatorname{diag}(\lambda_1, \ldots, \lambda_n). \tag{2.2}$$

Recíprocamente, si $T \in \mathbb{C}^{n\times n}$ es una matriz invertible que verifica (2.2) con $\lambda_i \neq \lambda_j$, entonces la $i-$ésima columna de T es un vector propio asociado al valor propio λ_i.

Aplicando lo anterior a la matriz A^*A y teniendo en cuenta la demostración de la Proposición 2.13 se tiene que

$$V^*A^*AV = \operatorname{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2),$$

y también

$$U^*AA^*U = \operatorname{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2).$$

Esto indica que las columnas de V son una base ortonormal de vectores propios de $\mathbb{C}^n$ respecto de A^*A; y las de U son una base ortonormal de vectores propios de $\mathbb{C}^n$ respecto de AA^*. Además, si $A = U_1\Sigma V_1^*$ es otra descomposición de A en valores singulares, entonces $v_i = \lambda v_i^1$ (i-ésimas columnas de V y V_1). Como en este caso son, además, vectores unitarios, se tiene que $1 = v_i^* v_i = |\alpha|^2 v_i^{1*} v_i^1 = |\alpha|$. Es decir, α es un escalar de módulo 1. Para las columnas de U

sirve un razonamiento similar. Véase [3, 4, 13, 24, 30].

■

2.5 Propiedades de la SVD

La importancia de la SVD se hace evidente cuando se comienza a relacionar con temas fundamentales del álgebra lineal. A continuación se analizan algunas propiedades que se derivan del Teorema SVD:

Proposición 2.3. *Si r es el número de valores singulares de A distintos de cero, entonces* $\text{rang}(A) = r$.

La demostración es una consecuencia inmediata de que el rango de una matriz no varía si se multiplica dicha matriz por matrices invertibles. Ver [30].

Proposición 2.4. *Si $A = U\Sigma V^t$ es una descomposición en valores singulares de $A \in \mathbb{R}^{m\times n}$, $r = \text{rang}(A)$, y $U = [u_1\ u_2\ \cdots\ u_m]$ y $V = [v_1\ v_2\ \cdots\ v_n]$ entonces* $\text{Im}(A) = <u_1, \ldots, u_r>$ *y* $\text{Ker}(A) = <v_{r+1}, \ldots, v_m>$.

Demostración. Sobre la base de que V y U son invertibles, es inmediato que

$$\text{Im}(AV) = \text{Im}(A) \quad y \quad \text{Ker}(U^tA) = \text{Ker}(A).$$

Ahora bien,

$$\text{Im}(AV) = \text{Im}(U\Sigma) = <\sigma_1 u_1, \ldots, \sigma_r u_r>.$$

Por otra parte, como $\{v_1, \ldots, v_m\}$ es una base ortonormal de $\mathbb{R}^n$, si $x \in \mathbb{R}^n$ entonces $x = \sum_{i=1}^{m} c_i v_i = Vc$, con $c = (c_1, \ldots, c_m)$. Así,

$$\begin{aligned} x \in Ker(A) \ &\Leftrightarrow\ Ax = 0 \Leftrightarrow AVc = 0 \Leftrightarrow U^tAVc = 0 \Leftrightarrow \Sigma c = 0 \\ &\Leftrightarrow\ \sigma_i c_i = 0, 1 \le i \le r \Leftrightarrow x = \sum_{i=r+1}^{m} c_i v_i. \end{aligned}$$

Esto significa que $\text{Ker}(A) = <v_{r+1}, \ldots, v_m>$. Véase [24].

■

Proposición 2.5. *Si* $A = U\Sigma V^t$ *es una descomposición en valores singulares de* $A \in \mathbb{R}^{m\times n}$, $r = \text{rang}(A)$, *y* $U = [u_1\ u_2\ \cdots\ u_m]$ *y* $V = [v_1\ v_2\ \cdots\ v_n]$ *entonces* $\text{Im}(A^t) = <v_1, \ldots, v_r>$ *y* $\text{Ker}(A^t) = <u_{r+1}, \ldots, u_m>$.

Esta proposición puede ser apreciada como una consecuencia inmediata de la proposición anterior teniendo en cuenta las siguientes relaciones:

$$(\text{Im}(A))^\perp = \text{Ker}(A^t) \text{ y } (\text{Ker}(A))^\perp = \text{Im}(A^t). \tag{2.3}$$

Véase [6] para detalles de la demostración acerca de (2.3).

Proposición 2.6. *Sean* $A \in \mathbb{R}^{m\times n}$ *y* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ *sus valores singulares, con* $p = \text{mín}(m, n)$, *entonces* $\|A\|_2 = \sigma_1$ *y* $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_p^2}$.

Demostración. En efecto, si $A = U\Sigma V^t$ es una descomposición en valores singulares de A, como las normas $\|\cdot\|_2$ y $\|\cdot\|_F$ son unitariamente invariantes, se tiene que

$$\|A\|_2 = \|\Sigma\|_2 \text{ y } \|A\|_F = \|\Sigma\|_F.$$

Basta probar que $\|\Sigma\|_2 = \sigma_1$ y $\|\Sigma\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_p^2}$. La segunda expresión es inmediata por la definición de la norma de Frobenius. En cuanto a la primera expresión, supóngase que $m \geq n$ y sea $x \in \mathbb{R}^n$ un vector arbitrario tal que $\|x\| = 1$. Entonces

$$\|\Sigma x\|_2 = \sqrt{\sigma_1^2|x_1|^2 + \cdots + \sigma_n^2|x_n|^2} \leq \sqrt{|x_1|^2 + \cdots + |x_n|^2} = \sigma_1\|x\|_2 = \sigma_1,$$

donde se ha utilizado que $\sigma_1 \geq \cdots \geq \sigma_n$ y que $\|x\| = 1$. Además, si $e_1 = (1, 0, \ldots, 0) \in \mathbb{R}^n$ entonces $\|e_1\|_2 = 1$ y $\|\Sigma e_1\|_2 = \sigma_1$. Esto prueba que

$$\sigma_1 = \max_{\|x\|_2=1} \|\Sigma x\|_2 = \|\Sigma\|_2. \text{ Ver [30].}$$

■

Proposición 2.7. *Si $A \in \mathbb{R}^{n\times n}$ y $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ son sus valores singulares, entonces* $|\det(A)| = \prod_{i=1}^{n} \sigma_i$.

Demostración.- Si $A = U\Sigma V^t$ es una descomposición en valores singulares,

$$\det(A) = \det(U)\cdot \det(\Sigma)\cdot \det(V^t).$$

Pero U y V son ortogonales. Entonces, por una parte, $UU^t = I_{n\times n}$ y por otra, $\det(U^t) = \det(U)$. Así,

$$1 = \det(I_{n\times n}) = \det(U)\cdot \det(U^t) = \det(U)\cdot \det(U) = \det(U)^2$$

.

Finalmente,

$$|\det(U)| = |\det(V)| = 1,$$

y

$$|\det(A)| = |\det(\Sigma)| = \prod_{i=1}^{n} \sigma_i. \text{ Ver [30].}$$

■

Proposición 2.8. *Sean $A \in \mathbb{R}^{n\times n}$ invertible y $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ sus valores singulares, entonces los valores singulares de A^{-1} son $\frac{1}{\sigma_n} \geq \cdots \geq \frac{1}{\sigma_1}$. En particular, $\|A^{-1}\|_2 = \frac{1}{\sigma_n}$.*

Demostración. Si $A = U\Sigma V^t$ es una descomposición en valores singulares de A y es invertible, entonces $A^{-1} = V\Sigma^{-1}U^t$. Note que

$$\Sigma^{-1} = \operatorname{diag}\left(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_n}\right)$$

y que $\frac{1}{\sigma_1} \leq \cdots \leq \frac{1}{\sigma_n}$. Además, existe una matriz de permutación

$$P = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix}$$

tal que $P\Sigma^{-1}P^t = \text{diag}\left(\frac{1}{\sigma_n}, \ldots, \frac{1}{\sigma_1}\right)$. Considérese $V_1 = VP^t$ y $U_1 = UP^t$, entonces V_1 y U_1 son ortogonales, porque el producto de matrices ortogonales es una matriz ortogonal, y $A^{-1} = V_1 P\Sigma^{-1}P^t U_1^t$ es una descomposición en valores singulares de A^{-1}.

Como $\|A^{-1}\|_2$ es el mayor valor singular de A^{-1} entonces la conclusión es inmediata. Refiérase a [30].

■

Proposición 2.9. *Sean $A \in \mathbb{R}^{n\times n}$ invertible y $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ sus valores singulares, entonces*

$$\text{cond}(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}.$$

La prueba de esta proposición es consecuencia directa de la Proposición 2.6 y la Proposición 2.8.

Proposición 2.10. *Si $A = U\Sigma V^t \in \mathbb{R}^{m\times n}$ es una descomposición en valores singulares y $\text{rang}(A) = r$, entonces*

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^t \tag{2.4}$$

donde $U = [u_1 \ u_2 \ \cdots \ u_m]$, $V = [v_1 \ v_2 \ \cdots \ v_n]$ y $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ son los valores singulares de A.

Demostración. Basta considerar

$$\Sigma = \Sigma_1 + \Sigma_2 + \cdots + \Sigma_r, \quad \Sigma_i = \begin{bmatrix} \text{diag}(0, \ldots, \sigma_i, \ldots, 0) & 0 \\ 0 & 0 \end{bmatrix}$$

donde $\text{diag}(0, \ldots, \sigma_i, \ldots, 0) \in \mathbb{R}^{r\times r}$ y σ_i aparece en la $i-$ésima posición.

Es claro que $A = \sum_{i=1}^{r} U\Sigma_i V^t$ y que $U\Sigma_i V^t = \sigma_i u_i v_i^t$. Véase [30].

■

Observación 2.11. *La expresión (2.4) indica que se puede expresar a A como una suma de matrices de rango 1.*

Observación 2.12. *Note que*

$$\sum_{i=1}^{r} \sigma_i u_i v_i^t = U_r \Sigma_r V_r^t,$$

con $U_r = [u_1 \ u_2 \ \cdots \ u_r]$, $V_r = [v_1 \ v_2 \ \cdots \ v_r]$ y $\Sigma_r = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ es una descomposición reducida en valores singulares de A.

La siguiente proposición suministra una forma práctica de calcular los valores singulares de una matriz. Incluso, los métodos numéricos que se describen en el próximo capítulo están basados en este hecho.

Proposición 2.13. *Los valores singulares de $A \in \mathbb{R}^{m\times n}$ distintos de cero son las raíces cuadradas positivas de los valores propios distintos de cero de la matriz simétrica A^tA y también de los de la matriz simétrica AA^t. En otras palabras, si $A = U\Sigma V^t$ es la descomposición en valores singulares de A, r el rango de la matriz A y si se supone que $m \geq n$, entonces*

$$V^t(A^tA)V = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, 0, \ldots, 0)_{n\times n},$$

$$U^t(AA^t)U = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, 0, \ldots, 0)_{m\times m}.$$

Demostración. Se demostrará que los valores singulares de A son las raíces cuadradas positivas de los valores propios de A^tA. Para el caso de AA^t se demuestra de manera análoga. También es consecuencia de la siguiente propiedad: *Si $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times m}$ entonces los valores propios distintos de cero de AB y BA son los mismos.* La explicación de esta propiedad está contenida en la siguiente ecuación:

$$\begin{bmatrix} I_{m\times m} & -A \\ 0 & I_{n\times n} \end{bmatrix} \begin{bmatrix} AB & 0 \\ B & 0 \end{bmatrix} \begin{bmatrix} I_{m\times m} & A \\ 0 & I_{n\times n} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ B & BA \end{bmatrix}.$$

Como $\begin{bmatrix} I_{m\times m} & A \\ 0 & I_{n\times n} \end{bmatrix}^{-1} = \begin{bmatrix} I_{m\times m} & -A \\ 0 & I_{n\times n} \end{bmatrix}$, las matrices $\begin{bmatrix} AB & 0 \\ B & 0 \end{bmatrix}$ y $\begin{bmatrix} 0 & 0 \\ B & BA \end{bmatrix}$ son semejantes; es decir, tienen los mismos valores propios. Además, $\det \begin{bmatrix} \lambda I_{m\times m} - AB & 0 \\ -B & \lambda I_{n\times n} \end{bmatrix} = \lambda^n \det(\lambda I_{m\times m} - AB)$ y $\det \begin{bmatrix} \lambda I_{m\times m} & 0 \\ -B & \lambda I_{n\times n} - BA \end{bmatrix} = \lambda^m \det(\lambda I_{n\times n} - BA)$. Por lo tanto, las matrices AB y BA tienen los mismos valores propios distintos de cero.

Si $A = U\Sigma V^t$ es una descomposición en valores singulares de A entonces

$$A^tA = V\Sigma^tU^tU\Sigma V^t = V\Sigma^t\Sigma V^t$$

porque Σ es una matriz de números reales. Como V es ortogonal $V^t = V^{-1}$, por lo que A^tA y $\Sigma^t\Sigma$ son semejantes. Es decir, tienen los mismos valores propios. Pero

$$\Sigma^t\Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_r^2, 0, \ldots, 0) \in \mathbb{R}^{n\times n}$$

con $r = \text{rang}(A)$. Por lo tanto, $\sigma_1^2 \geq \cdots \geq \sigma_r^2$ son los valores propios de $\Sigma^t\Sigma$ y de A^tA. Véase [24].

■

Proposición 2.14. *Sea* $H = \begin{pmatrix} 0 & A^t \\ A & 0 \end{pmatrix}$, *donde* A *es de* $m\times n$ *y* $A = U\Sigma V^t$ *es la SVD de* A. *Sea* $p = \text{mín}(m, n)$, $\Sigma = diag(\sigma_1, \ldots, \sigma_p)$, $U = [u_1, \ldots, u_m]$, *y* $V = [v_1, \ldots, v_n]$. *Entonces los* $m + n$ *autovalores de* H *son los* $\pm\sigma_i$, *con correspondientes autovectores unitarios* $\frac{1}{\sqrt{2}} \begin{pmatrix} v_i \\ \pm u_i \end{pmatrix}$.

Corolario 2.15. *Si* $A = A^t$ *(simétrica), entonces los valores singulares de* A *son los autovalores de* A *en valor absoluto.*

Refiérase a [27] para detalles de la demostración.

Corolario 2.16. *Una matriz A de $n \times n$ es no-singular si y sólo si todos sus valores singulares son diferentes de cero.*

Una de las aplicaciones interesante del Teorema SVD (Teorema 2.2) es la de calcular el rango de una matriz con alta precisión. De hecho, dicho teorema proporciona una medida de esa precisión. Esto es consecuencia del siguiente teorema que suministra una cota de la distancia que hay de una matriz al conjunto de matrices de rango menor que ella.

Proposición 2.17. *Sea $A \in \mathbb{R}^{m \times n}$ una matriz de rango r cuya SVD es $A = U\Sigma V^t = \sum_{i=1}^{r} \sigma_i u_i v_i^t$. Considérese la matriz A_k como en Proposición 2.10: $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^t$. Entonces se cumple que la matriz de rango $k < r$ que mejor aproxima a A en norma espectral es A_k, es decir:*

$$\min_{rango(\tilde{A})=k} \|A - \tilde{A}\|_2 = \|A - A_k\|_2 = \sigma_{k+1},$$

y además el error en la aproximación es el valor singular σ_{k+1}, que es el mayor valor singular no incluido en A_k.

Véase [27] para una demostración detallada.

La última proposición mencionada hasta ahora, proporciona, como corolario, la distancia de una matriz no-singular a la matriz singular más próxima en la norma espectral: el valor singular más pequeño de la matriz no-singular.

Corolario 2.18. *Si $A \in \mathbb{C}^{n \times n}$ es una matriz no-singular y $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ son sus valores singulares, entonces*

$$\min_{det(B)=0} \|A - B\|_2 = \sigma_n.$$

Refiérase a [30] para una demostración.

2.6 SVD vs. Descomposición en Autovalores

Definición 2.19. *Una descomposición en autovalores de una matriz cuadrada A, es una factorización de la forma*

$$A = X\Lambda X^{-1}$$

donde X es no-singular y Λ es diagonal.

Existen diferencias fundamentales entre la SVD y la descomposición en autovalores. Primero, la SVD usa dos bases diferentes (los conjuntos de vectores singulares izquierdos y derechos), mientras que la descomposición en autovalores solo usa una base (el conjunto de autovectores). Segundo, la SVD usa bases ortonormales, mientras que la descomposición en autovalores usa una base que, generalmente, no es ortogonal. Tercero, no toda matriz (incluso, cuadrada) tiene una descomposición en autovalores, pero toda matriz (incluso, rectangular) tiene una descomposición en valores singulares como establece el Teorema SVD (Teorema 2.2).

2.7 Sensibilidad de los Valores Singulares

Una de las razones de la amplia aplicabilidad de los valores singulares en estudios prácticos, es el hecho de que ellos son bien estables numéricamente. A continuación se muestra un resultado al respecto:

Teorema 2.20. *La perturbación en los valores singulares de una matriz ante una perturbación E en la matriz A está acotada mediante la expresión:*

$$|\sigma_k(A+E) - \sigma_k(A)| \leq \|E\|_2.$$

Esto quiere decir que los valores singulares son muy estables ante perturbaciones de la matriz.

Demostración. En función de los valores y vectores singulares de la matriz A, puede considerarse:

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^t, \;\; A_{k-1} = \sum_{j=1}^{k-1} \sigma_j u_j v_j^t, \;\; \text{rang}(A_{k-1}) = k-1.$$

En virtud de la Proposición 2.17:

$$\min_{\text{rang}(B)\leq k-1} \|(A+E)-B\|_2 = \sigma_k(A+E). \tag{2.5}$$

Sustituyendo $B = A_{k-1}$ en la expresión (2.5), se obtiene:

$$\sigma_k(A+E) \leq \|A+E-A_{k-1}\|_2 \leq \|A-A_{k-1}\|_2 + \|E\|_2 = \sigma_k(A) + \|E\|_2.$$

A partir de este resultado:

$$\sigma_k(A+E) - \sigma_k(A) \leq \|E\|_2.$$

Así concluye la primera parte de la demostración. Ahora, sea $\bar{\sigma}_j$, $\bar{u_j}$ y $\bar{v_j}$ los valores y vectores singulares, respectivamente, de la matriz $A+E$, entonces

$$A+E = \sum_{j=1}^{\bar{r}} \bar{\sigma}_j \bar{u_j} \bar{v_j^t}, \;\; (A+E)_{k-1} = \sum_{j=1}^{k-1} \bar{\sigma}_j \bar{u_j} \bar{v_j^t}, \;\; \text{rang}(A+E)_{k-1} = k-1.$$

Aplicando ahora la Proposición 2.17 y tomando $B = (A+E)_{k-1}$ se obtiene:

$$\min_{\text{rang}(B)\leq k-1} \|A-B\|_2 = \sigma_k(A),$$

$$\sigma_k(A) \leq \|A+E-E-(A+E)_{k-1}\|_2 \leq \|A+E-(A+E)_{k-1}\|_2 + \|E\|_2 = \sigma_k(A+E) + \|E\|_2.$$

De esta expresión se concluye que

$$\sigma_k(A) - \sigma_k(A+E) \leq \|E\|_2.$$

■

Se ha podido notar la habilidad de los valores singulares para calcular eficientemente el rango numérico, el número de condición de una matriz y, a su vez, la distancia a una matriz de rango menor, entre otros aspectos. Esto hace

de la SVD una descomposición atractiva en muchas aplicaciones prácticas. Por ello, es importante saber cómo puede ser calculada numéricamente de manera estable. En el próximo capítulo se describen algunos métodos relacionados con la descomposición en autovalores de matrices simétricas para, posteriormente, presentar algunos algoritmos que permiten obtener la SVD de una matriz real, tomando como base las proposiciones 2.13 y 2.14 descritas en este capítulo.

CAPÍTULO 3

PROBLEMA DE AUTOVALORES PARA MATRICES TRIDIAGONALES SIMÉTRICAS

En este capítulo se describen algunos métodos que permiten obtener una descomposición en autovalores de una matriz real T tridiagonal simétrica, así como otro donde únicamente se presenta el cálculo de los autovalores. De entre los métodos existentes, aquí se muestran los siguientes: Iteración QR Implícito Simétrico con Desplazamiento Wilkinson [7, 8, 13, 27], Divide y Vencerás [7, 8, 13], Bisección e Iteración Inversa [7, 13] y DQDS [1, 2, 8]. Por último, se exhibe el método de Jacobi que, aunque no comienza con una matriz tridiagonal, sí trabaja inicialmente con una matriz densa simétrica [7, 8, 13].

3.1 Iteración QR Implícito Simétrico con Desplazamiento Wilkinson

Antes de implementar este método, es importante saber cómo se comportan y en qué consiste los antecesores a este. Es por ello que, primeramente, se plantea el método de Iteración QR Básico, luego, el método de Iteración QR con un Desplazamiento. Posteriormente, se analiza este último para el caso simétrico de manera implícita para matrices tridiagonales, donde el desplazamiento es

uno propuesto por el inglés James Wilkinson en 1965.

3.1.1 Iteración QR Básico

En 1961, el inglés John Francis comenzó con la implementación de esta técnica para el cálculo de autovalores y autovectores de matrices arbitrarias.

Algoritmo 3.1. Iteración QR Básico

A matriz arbitraria
Para $k = 0, 1, \ldots$
$A_k = UR$ *(Factorización QR)*
$A_{k+1} = RU$
Fin Para

Teorema 3.1. *(Un teorema de convergencia para Iteración QR Básico). Sean* $A \in \mathbb{R}^{n\times n}$, $\lambda_1, \ldots, \lambda_n$ *sus autovalores tal que* $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ *y la matriz* Y *de autovectores izquierdos tal que las sub-matrices principales de* A *son distintas de la matriz nula, entonces la sucesión* $A_{k\{k\in\mathbb{N}\}}$ *converge a una matriz triangular superior.*

Para detalles acerca de la convergencia de Iteración QR básico véase [29].

Cada matriz es ortogonalmente similar a la matriz original $A = A_0$. Esto implica que, todas las matrices preservan los mismos autovalores de A. Además, como la sucesión $\{A_k\}$ converge a una matriz triangular superior, entonces se tiene que los autovalores de la matriz original A están en la diagonal de la matriz límite de la sucesión.

3.1.2 Iteración QR Explícito Simétrico con Desplazamiento

El Desplazamiento surgió de la necesidad de acelerar la convergencia del algoritmo QR Básico. Para implementar este método con dicha estrategia es

necesario tener un aproximado del autovalor λ_i de la matriz original. En la práctica, la entrada (n, n) se toma como la aproximación inicial y entonces, la iteración se continúa, usando la entrada (n, n) de la matriz actualizada como el nuevo traslado. Es importante acotar que este mecanismo de traslado es útil, ya que permite acelerar la convergencia efectuada por el método Iteración QR Básico. He aquí los pasos del método para una matriz tridiagonal simétrica T:

Algoritmo 3.2. Iteración QR Explícito Simétrico con Desplazamiento

T Tridiagonal simétrica
Para $k = 0, 1, \ldots$
Determinar un desplazamiento real μ.
$T - \mu I = UR$ (Factorización QR)
$T = RU + \mu I$
Fin Para

3.1.3 Iteración QR Implícito Simétrico con Desplazamiento Wilkinson para una Matriz Tridiagonal

La idea de realizar Iteración QR con desplazamiento de manera implícita permite no calcular la matriz $T - \mu I$ tridiagonal simétrica, explícitamente, en cada iteración. De esta forma, se puede realizar una implementación más efectiva.

Sea

$$T = \begin{pmatrix} a_1 & b_1 & & \cdots & 0 \\ b_1 & a_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & & b_{n-1} & a_n \end{pmatrix}$$

entonces, como ya se mencionó, una elección razonable para el traslado es tomar

$\mu = a_n$. Sin embargo, una alternativa más práctica es trasladar por el autovalor de

$$T(n-1:n, n-1:n) = \begin{pmatrix} a_{n-1} & b_{n-1} \\ b_{n-1} & a_n \end{pmatrix}$$

más cercano a a_n. Esto es conocido como *traslado Wilkinson* y está dado por

$$\mu = a_n + d - \text{sign}(d)\sqrt{d^2 + b_{n-1}^2}, \tag{3.1}$$

donde $d = (a_{n-1} - a_n)/2$. Wilkinson demostró en 1968 que el Algoritmo 3.2 es convergente cúbicamente con o sin desplazamiento, pero por razones de conveniencia, el traslado de la expresión (3.1) es preferido.

Teorema 3.2. *La Iteración QR con Desplazamiento Wilkinson es cúbicamente convergente para casi todas las matrices.*

Para detalles de la demostración véase [20].

Dado $T \in \mathbb{R}^{n\times n}$ tridiagonal simétrica, el siguiente algoritmo transforma T a Z^tTZ, donde $Z = G_1 \cdots G_{n-1}$ es un producto de rotaciones de Givens y μ es el autovalor de la submatriz principal de 2×2 de T más cercano a $t_{n,n}$:

Algoritmo 3.3. Un paso QR Implícito Simétrico con Desplazamiento Wilkinson

Entrada: *matriz* $T \in \mathbb{R}^{n\times n}$ *tridiagonal simétrica*

Salida: *matriz* $T \in \mathbb{R}^{n\times n}$ *tridiagonal simétrica y matriz* $Z \in \mathbb{R}^{n\times n}$ *ortogonal*

$d = (t_{n-1,n-1} - t_{nn})/2$

$\mu = t_{n,n} - t_{n,n-1}^2/(d + \text{sign}(d)\sqrt{d^2 + t_{n,n-1}^2})$ *(desplaz. Wilkinson)*

$x = t_{11} - \mu$

$z = t_{21}$

Para $j = 1, \ldots, n-1$ **Hacer**

Obtener parámetros de Givens c_j y s_j usando Algoritmo 1.1 para $\mathbf{x} = (x \;\; z)^t$

Para $k = j + 1$ **Hacer**

Calcular $T = G^t T G$, *donde* $G = G(j, k, \theta)$

Obtener T ejecutando Algoritmo 1.2 (Premult. Givens) a T tridiag. con parámetros c_j y s_j e índices j y k.

Obtener T ejecutando Algoritmo 1.3 (Postmult. Givens) a T actualiz. con parámetros c_j y s_j e índices j y k.

Si $j < n - 1$ **Hacer**

$x = t_{j+1,k-1}$

$z = t_{j+2,k-1}$

Fin Si

Aplicar Algoritmo 1.3 (Postmult. Givens) a $I \in \mathbb{R}^{n\times n}$ con parámetros c_j y s_j e índices j y k para obtener el producto acumulado

$Z = IG_1 \ldots G_{n-1}$

Fin Para

Fin Para

Este algoritmo requiere cerca de $30n$ operaciones y n raíces cuadradas. Si una matriz ortogonal Q es transformada a $QG_1 \cdots G_{n-1}$, entonces $6n^2$ operaciones adicionales son necesitadas. Por otra parte, es importante acotar que dicho algoritmo no forma explícitamente la matriz $T - \mu I$. Es por ello que, en el próximo capítulo se podrá apreciar un método basado en este algoritmo para el cálculo de la SVD de una matriz A general, pero donde, además, la matriz tridiagonal simétrica T tampoco será formada explícitamente.

Teorema 3.3 (Q Implícito). *Sean $Q = [q_1, \ldots, q_n]$ y $V = [v_1, \ldots, v_n]$ matrices ortogonales con la propiedad de que tanto $Q^t A Q = T$ como $V^t A V = S$ son matrices tridiagonales, donde $A \in \mathbb{R}^{n\times n}$. Sea k el entero positivo más pequeño para el cual $t_{k+1,k} = 0$, con la convención que $k = n$ si T es irreducible. Si $v_1 = q_1$, entonces $v_i = \pm q_i$ y $|t_{i,i-1}| = |s_{i,i-1}|$, para $i = 2, \ldots, n$. Además, si $k < n$, entonces $s_{k+1,k} = 0$.*

Para detalles de la demostración véase [13].

Este teorema es fundamental para probar que el Algoritmo 3.3 es equivalente a otro que se expondrá en el capítulo 4 en la que no se construye explícitamente la matriz T tridiagonal simétrica.

Algoritmo 3.4. QR Implícito Simétrico con Traslado Wilkinson para la descomposición en autovalores

Entrada: $T \in \mathbb{R}^{n\times n}$ *matriz tridiagonal simétrica*

Salida: $Q \in \mathbb{R}^{n\times n}$ *matriz ortogonal y* $\Lambda \in \mathbb{R}^{n\times n}$ *matriz diagonal*

$W = \mathbb{I}_{n\times n}$

Para $j = 1, \ldots, n-1$

Mientras $|t_{n-j,n+1-j}| = |t_{n+1-j,n-j}| \geq \epsilon(|t_{n+1-j,n+1-j}| + |t_{n-j,n-j}|)$ **Hacer**

Calcular $T_{(1:n+1-j,1:n+1-j)} = Z^t T_{(1:n+1-j,1:n+1-j)} Z$

Obtener Z *y* $T \in \mathbb{R}^{(n+1-j)\times(n+1-j)}$ *aplicando Algoritmo 3.3*

(Un paso QR Implícito) a $T \in \mathbb{R}^{(n+1-j)\times(n+1-j)}$

$$Z = \begin{pmatrix} Z & 0 \\ 0 & \mathbb{I}_{(j-1)\times(j-1)} \end{pmatrix}$$

Acumulando Z

$W = ZW$

Fin Mientras

Fin Para

$Q = W$

Para $i = 1, \ldots, n$

$\Lambda_{(i,i)} = T_{(i,i)}$

Fin Para

3.2 Divide y Vencerás

Su nombre se debe al hecho de dividir, de manera recursiva, el problema grande de autovalores en una sucesión finita de sub-problemas, de la misma especie, fáciles de resolver. Este método es eficiente para conseguir todos los

autovalores y autovectores de matrices con órdenes mayores a $n = 25$, ya que, no garantiza que pequeños autovalores sean calculados con alta precisión. A continuación se describe la estrategia de este método para el problema de autovalores de una matriz tridiagonal simétrica T:

$$
\begin{aligned}
T &= \left[\begin{array}{ccccc|ccccc}
\alpha_1 & \beta_1 & & & & & & & & \\
\beta_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & \alpha_{m-1} & \beta_{m-1} & & & & & & \\
 & & \beta_{m-1} & \alpha_m & \beta_m & & & & & \\
\hline
 & & & \beta_m & \alpha_{m+1} & \beta_{m+1} & & & & \\
 & & & & \beta_{m+1} & \ddots & & & & \\
 & & & & & & \ddots & \beta_{n-1} & & \\
 & & & & & & \beta_{n-1} & \alpha_n & &
\end{array}\right] \\
&= \left[\begin{array}{ccccc|ccccc}
\alpha_1 & \beta_1 & & & & & & & & \\
\beta_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & \alpha_{m-1} & \beta_{m-1} & & & & & & \\
 & & \beta_{m-1} & \alpha_m - \beta_m & & & & & & \\
\hline
 & & & & \alpha_{m+1} - \beta_m & \beta_{m+1} & & & & \\
 & & & & \beta_{m+1} & \ddots & & & & \\
 & & & & & & \ddots & \beta_{n-1} & & \\
 & & & & & & \beta_{n-1} & \alpha_n & &
\end{array}\right] + \left[\begin{array}{ccccc|ccccc}
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & \beta_m & \beta_m & & \\
\hline
 & & & & & & \beta_m & \beta_m & &
\end{array}\right] \\
&= \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + \beta_m \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1_m \\ 1_{m+1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & 0 & 1_m & 1_{m+1} & 0 & \cdots & 0 \end{bmatrix} \\
&\equiv \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + \beta_m v v^t.
\end{aligned}
$$

Supóngase que se tiene la descomposición en autovalores de T_1 y T_2: $T_i = Q_i \Lambda_i Q_i^t$, para $i = 1, 2$. Éstas se calculan de forma recursiva haciendo uso

de este mismo algoritmo. Se relaciona la descomposición en autovalores de T a los de T_1 y T_2 como sigue:

$$\begin{aligned} T &= \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \beta_m v v^t \\ &= \begin{bmatrix} Q_1 \Lambda_1 Q_1^t & 0 \\ 0 & Q_2 \Lambda_2 Q_2^t \end{bmatrix} + \beta_m v v^t \\ &= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} + \beta_m u u^t \right) \begin{bmatrix} Q_1^t & 0 \\ 0 & Q_2^t \end{bmatrix}, \end{aligned}$$

donde

$$u = \begin{bmatrix} Q_1^t & 0 \\ 0 & Q_2^t \end{bmatrix}$$

$$v = \begin{bmatrix} \text{última columna de } Q_1^t \\ \text{primera columna de } Q_2^t \end{bmatrix}.$$

Por lo tanto, los autovalores de T son los mismos de los de la matriz similar $D + \rho u u^t$, donde $D = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}$ es diagonal, $\rho = \beta_m$ es un escalar, y u es un vector. Por consiguiente, se presenta el siguiente algoritmo que consigue los autovalores y autovectores de una matriz tridiagonal simétrica T usando el método Divide y Vencerás:

Algoritmo 3.5. Descomposición en Autovalores de una Matriz Tridiagonal Simétrica vía Divide y Vencerás

Entrada: *matriz $T \in \mathbb{R}^{n \times n}$ tridiagonal simétrica*

Salida: *matriz $Q \in \mathbb{R}^{n \times n}$ ortogonal y matriz $\Lambda \in \mathbb{R}^{n \times n}$ diagonal*

Si $T \in \mathbb{R}^{1 \times 1}$

Terminar $Q = 1,\ \Lambda = T$

Si no

Formar $T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \beta_m v v^t$

Obtener Q_1 *y* Λ_1 *aplicando Algoritmo 3.5 (Divide y Vencerás) a* T_1

Obtener Q_2 *y* Λ_2 *aplicando Algoritmo 3.5 (Divide y Vencerás) a* T_2

Formar $D + \rho u u^t$ *a partir de* $\Lambda_1,\ \Lambda_2,\ Q_1,\ Q_2$

Obtener la matriz Λ *de autovalores y la matriz* Q^* *de autovectores de* $D + \rho u u^t$.

Formar $Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \cdot Q^* =$ *matriz de autovectores de* T

Terminar $Q\ y\ \Lambda$

Fin Si

3.3 Bisección e Iteración Inversa

Este método consiste en hallar los valores propios o autovalores de matrices tridiagonales simétricas en un intervalo real dado. Sin embargo, este pudiera ejecutarse de manera eficiente si sólo se determinaran unos pocos autovalores.

El método de Bisección es usado, a menudo, junto al método de Iteración Inversa para poder obtener, de manera eficiente, los autovectores correspondientes a los autovalores que se hallan con el primero.

3.3.1 Bisección

Definición 3.4. *La inercia de una matriz simétrica* A *es la terna de enteros* $\text{Inercia}(A) \equiv (\upsilon, \zeta, \pi)$*, donde* υ *es el número de autovalores negativos de* A*,* ζ *es el número de autovalores nulos de* A*, y* π *es el número de autovalores positivos de* A*.*

El algoritmo basado en Bisección usa el teorema de Inercia de Sylvester, el cual se enuncia a continuación:

Teorema 3.5. ***Inercia de Sylvester****. Sea A una matriz simétrica y X una matriz no-singular. Entonces A y X^tAX tienen la misma inercia.*

Véase [8] para detalles de la demostración.

Ahora supóngase que se ha usado Eliminación Gaussiana para factorizar $A-zI = LDL^t$, donde L es no-singular y D es diagonal. Entonces, por Teorema 3.5, Inercia$(A - zI)$ =Inercia(D). Como D es diagonal, su inercia es fácil de calcular. (En lo que sigue, se usará la notación "$\#\ d_{ii} < 0$", la cual se lee como "el número de valores de d_{ii} que son menores que cero.")

$$\begin{aligned}
\text{Inercia}(A - zI) &= (\#\ d_{ii} < 0,\ \#\ d_{ii} = 0,\ \#\ d_{ii} > 0) \\
&= (\#\ \textit{autovalores negativos de}\ A - zI, \\
&\quad \#\ \textit{autovalores nulos de}\ A - zI, \\
&\quad \#\ \textit{autovalores positivos de}\ A - zI) \\
&= (\#\ \textit{autovalores de}\ A < z, \\
&\quad \#\ \textit{autovalores de}\ A = z, \\
&\quad \#\ \textit{autovalores de}\ A > z).
\end{aligned}$$

Supóngase $z_1 < z_2$ y calcúlese Inercia$(A-z_1I)$ e Inercia$(A-z_2I)$. Entonces el número de autovalores en el intervalo $[z_1, z_2)$ equivale a (# autovalores de $A < z_2$)$-$(# autovalores de $A < z_1$). Para tomar en consideración esta observación en un algoritmo, defínase

$$\text{Negcount}(A, z) = \#\ \textit{autovalores de}\ A < z.$$

Teorema 3.6 (**Gershgorin**). *Sea $A \in \mathbb{R}^{n\times n}$ arbitraria. Entonces los autovalores λ de A están localizados en la unión de los n discos*

$$|\lambda - a_{kk}| \leq \sum_{j=k} |a_{kj}|.$$

Refiérase a [13, 30] para detalles de la demostración.

Sea

$$T = \begin{pmatrix} a_1 & b_1 & & \dots & 0 \\ b_1 & a_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \dots & & b_{n-1} & a_n \end{pmatrix}.$$

Supóngase que se desea calcular $\lambda_k(T)$, para algún $k \in \{1, \dots, n\}$. Del teorema de Gershgorin (Teorema 3.6) se sigue que $\lambda_k(T) \in [a, b]$, donde

$$a = \min_{1\leq i\leq n} a_i - |b_i| - |b_{i-1}| \tag{3.2}$$

y

$$b = \max_{1\leq i\leq n} a_i + |b_i| + |b_{i-1}|. \tag{3.3}$$

Las expresiones (3.2) y (3.3) permiten hallar los extremos, inferior y superior, respectivamente, del intervalo que contiene todos los autovalores de la matriz T tridiagonal simétrica. A continuación se describe un algoritmo basado en la técnica de bisección del intervalo $[a, b]$ para el cálculo de autovalores:

Algoritmo 3.6. Bisección

Entrada: *$T \in \mathbb{R}^{n\times n}$ tridiagonal simétrica, $a, b \in \mathbb{Z}$ tal que $\lambda_k(T) \in [a, b)$, y tol*

Salida: *autovalores de $T \in \mathbb{R}^{n\times n}$*

$n_a = \text{Negcount}(T, a)$

$n_b = \text{Negcount}(T, b)$

Si $n_a = n_b$

Salir *porque no hay autovalores en* $[a, b)$

Colocar $[a, n_a, b, n_b]$ *en Worklist*

Worklist contiene una lista de intervalos $[a, b)$ *conteniendo autovalores desde el número* $n - n_a + 1$ *hasta el número* $n - n_b$*, los cuales son divididos repetidas veces por el algoritmo hasta que sean de longitud menor que tol.*

Mientras *Worklist no está vacío* **hacer**

Quitar $[low, n_{low}, up, n_{up}]$ *de Worklist*

Si $(up - low) < tol$ **entonces**

Imprimir "hay $n_{up} - n_{low}$ *autovalores en* $[low, up)$*"*

Si no

$mid = (low + up)/2$

$n_{mid} = \text{Negcount}(T, mid)$

Si $n_{mid} > n_{low}$ **entonces**

Imprimir "hay autovalores en $[low, mid)$*"*

Colocar $[low, n_{low}, mid, n_{mid}]$ *en Worklist*

Fin Si

Si $n_{up} > n_{mid}$ **entonces**

Imprimir "hay autovalores en $[mid, up)$*"*

Colocar $[mid, n_{mid}, up, n_{up}]$ *en Worklist*

Fin Si

Fin Si

Fin Mientras

Si T es densa, se pudiera implementar Negcount(T, z) mediante el método de Eliminación Gaussiana simétrico con pivoteo como está descrito en [8, p.79], pero este lograría ser no muy eficiente. Por lo que, Negcount(T, z) es muy fácil de calcular con poco costo computacional cuando T es tridiagonal simétrica, siempre que no se haga pivoteo:

$$T - zI = \begin{pmatrix} a_1 - z & b_1 & & \dots & 0 \\ b_1 & a_2 - z & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \dots & & b_{n-1} & a_n - z \end{pmatrix} = LDL^t$$

$$\equiv \begin{pmatrix} 1 & & \\ l_1 & \ddots & \\ & \ddots & \ddots \\ & & l_{n-1} \quad 1 \end{pmatrix} \cdot \begin{pmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} \cdot \begin{pmatrix} 1 & l_1 & & \\ & \ddots & \ddots & \\ & & \ddots & l_{n-1} \\ & & & 1 \end{pmatrix},$$

así, $a_1 - z = d_1$, $d_1 l_1 = b_1$ y posteriormente, $l_{i-1}^2 d_{i-1} + d_i = a_i - z$, $d_i l_i = b_i$. Sustituyendo $l_i = b_i/d_i$ en $l_{i-1}^2 d_{i-1} + d_i = a_i - z$ se obtiene la recurrencia siguiente:

$$d_i = (a_i - z) - \frac{b_{i-1}^2}{d_{i-1}}. \tag{3.4}$$

Note que, no se aplica pivoteo, por lo que se podría pensar que (3.4) es considerablemente inestable, especialmente cuando d_{i-1} es pequeño. Como la matriz T es tridiagonal, se puede demostrar que, para este caso, la relación (3.4) es muy estable [9, 16].

Lema 3.7. *El d_i calculado de la ecuación (3.4) tiene el mismo signo (y la misma inercia) del $\hat{d}_i$ calculado exactamente de la matriz $\hat{T}$, donde $\hat{T}$ es muy cercano a T:*

$$(\hat{T})_{ii} \equiv \hat{a}_i = a_i \;\; y \;\; (\hat{T})_{i,i+1} \equiv \hat{b}_i = b_i(1 + \epsilon_i), \;\; |\epsilon_i| \leq 2{,}5\varepsilon + O(\varepsilon^2).$$

Refiérase a [8] para detalles de la demostración.

El costo de una única llamada a Negcount para una matriz tridiagonal es de $4n$ operaciones.

Note que Bisección converge linealmente con mucha más precisión, para cada bisección de un intervalo dado. Existen algunas maneras de acelerar la convergencia usando algoritmos basados en el método de Newton o en cualquiera de sus variantes para conseguir los ceros del polinomio característico correspondiente a la matriz tridiagonal dada.

Para calcular los autovectores, una vez teniendo calculado los autovalores, se hará uso del método Iteración Inversa.

3.3.2 Iteración Inversa

Existe un método llamado Método de las Potencias [8] el cual permite calcular el autovalor de mayor módulo y su correspondiente autovector de una matriz $A \in \mathbb{R}^{n\times n}$ basándose en la construcción de una sucesión de vectores $\{u^k\}$, con u^0 inicial arbitrario, en la forma $u^{k+1} = Au^k$, para $k = 1, 2, \ldots$. Por ello, surgió la necesidad de plantear una estrategia que pudiera calcular, no únicamente el autovalor de mayor módulo, sino cualquier otro autovalor deseado bajo un parámetro σ y que, además, proporcionara una mayor velocidad de convergencia. Este método es denominado Iteración Inversa y consiste en aplicar el Método de las Potencias a la matriz $(A - \sigma I)^{-1}$, donde σ es una estrategia de traslado. Dicha iteración converge al autovalor más cercano a σ y a su autovector asociado (para detalles véase [8]). A continuación se presenta el algoritmo de este método para una matriz T tridiagonal simétrica:

Algoritmo 3.7. Iteración Inversa

Dado x_0 un vector inicial

$i = 0$

Repetir

Calcular $y_{i+1} = (T - \sigma I)^{-1}x_i$ usando una factorización LU de $T - \sigma I$

Resolver el sistema $Lz_i = x_i$, para z_i

Resolver el sistema $Uy_{i+1} = x_i$, para y_{i+1}

$x_{i+1} = y_{i+1}/\|y_{i+1}\|_2$ *(autovector aproximado)*

$\hat{\lambda}_{i+1} = x_{i+1}^t T x_{i+1}$ *(autovalor aproximado)*

$i = i + 1$

Hasta la convergencia

Este método es particularmente eficaz cuando se tiene una buena aproximación de un autovalor y se desea solamente calcular su autovector correspondiente. Esto indica que, si se considera a σ como un autovalor (hallado con el método de Bisección), el método de Iteración Inversa converge rápidamente a su autovector asociado. En este caso, el costo computacional es de $O(n)$ operaciones por autovector, ya que, un paso de Iteración Inversa requiere de resolver un sistema tridiagonal de ecuaciones, el cual se resuelve de manera eficiente usando una factorización LU de la matriz $T - \sigma I$ aprovechando su estructura en banda (refiérase a [8]). Cuando algunos autovalores $\hat{\lambda}_i, \ldots, \hat{\lambda}_j$, para ciertos $i, j \in \mathbb{N}$, con $i < j$, son muy cercanos entre sí, sus correspondientes autovectores calculados $\hat{q}_i, \ldots, \hat{q}_j$ podrían no ser ortogonales. Para evitar tal eventualidad, es conveniente hacer uso de un algoritmo que permita reortogonalizar tales autovectores y esto se logra calculando una factorización QR de $\hat{q}_i, \ldots, \hat{q}_j$ y reemplazando cada $\hat{q}_k$, con $k \in \{i, \ldots, j\}$, por la k-ésima columna de Q; esto garantiza que los $\hat{q}_k$ son ortonormales. Dicha factorización QR es usualmente calculada usando el proceso de ortogonalización Gram-Schmidt Modificado, el cual se presenta a continuación:

Algoritmo 3.8. Gram-Schmidt Modificado

Dado $T = [t_1 \cdots t_n] \in \mathbb{R}^{n \times n}$ *una matriz tridiagonal simétrica*

Para $i = 1, \ldots, n$

  $q_i = t_i$

  Para $j = 1, \ldots, i - 1$

    $r_{ji} = q_j^t q_i$

    $q_i = q_i - r_{ji} q_j$

  Fin Para

$r_{ii} = \|q_i\|_2$

$q_i = q_i / r_{ii}$

Fin Para

Finalmente, se muestra el algoritmo que fusiona el método de Bisección con el método de Iteración Inversa para la descomposición en autovalores de una matriz T tridiagonal simétrica:

Algoritmo 3.9. Bisección e Iteración Inversa

Entrada: $T \in \mathbb{R}^{n \times n}$ *matriz tridiagonal simétrica*

Salida: $D \in \mathbb{R}^{n \times n}$ *matriz diagonal de autovalores*

$Q \in \mathbb{R}^{n \times n}$ *matriz ortogonal de autovectores*

Hallar $\lambda_1, \ldots, \lambda_n$ *autovalores de* T *aplicando Algoritmo 3.6 (Bisección)*

Para $k = 1, \ldots, n$

$\sigma_k = \lambda_k$

Calculando el autovector $\hat{q}_k$ *de* T

Usar σ_k *como estrategia de traslado en Algoritmo 3.7 (Iteración Inversa) para obtener* $\hat{q}_k$

Fin Para

Obtener los q_k *reortogonalizando los* $\hat{q}_k$*, para ciertos* k *que lo requieran, usando Algoritmo 3.8 (Gram-Schmidt Modificado)*

$D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$

$Q = [q_1 \quad q_2 \quad \cdots \quad q_n]$

3.4 DQDS

En 1994, Fernando K.V. y Parlett B.N. [19, 20] descubrieron un algoritmo, el cual es llamado DQDS (*Differential quotient-difference algorithm with shifts*, por razones históricas). El algoritmo DQDS ha logrado un gran interés por parte de muchos investigadores por su alta precisión, rapidez y estabilidad

numérica.

Para derivar DQDS, se comienza con un algoritmo que antecede a Iteración QR, llamado Iteración LR, especializado para matrices simétricas definidas positivas, en este caso, se adapta para matrices tridiagonales.

Algoritmo 3.10. Iteración LR: *Sea T_0 una matriz tridiagonal simétrica. El siguiente algoritmo produce una sucesión de matrices tridiagonales simétricas T_i:*

$i = 0$

Repetir

Obtener un desplaz. τ_i^2 más pequeño que el autovalor más pequeño de T_i.

Calcular la factorización de Cholesky $T_i - \tau_i^2 I = B_i^t B_i$

(B_i es una matriz bidiagonal superior con diagonal positiva.)

$T_{i+1} = B_i B_i^t + \tau_i^2 I$

$i = i + 1$

Hasta la convergencia

La Iteración LR es muy similar en estructura a la Iteración QR: se calcula una factorización y se multiplica los factores en orden reverso para obtener el próximo iterado T_{i+1}. Es fácil ver que T_{i+1} y T_i son similares:

$$T_{i+1} = B_i B_i^t + \tau_i^2 I = B_i^{-t} B_i^t B_i B_i^t + \tau_i^2 B_i^{-t} B_i^t = B_i^{-t} T_i B_i^t.$$

De hecho, cuando el traslado $\tau_i^2 = 0$, se puede probar que dos pasos de Iteración LR produce el mismo T_2 que produce iteración QR con un paso.

Lema 3.8. *Sea T_2 la matriz producida por dos pasos del Algoritmo 3.10 (Iteración LR) usando $\tau_i^2 = 0$, y sea $\hat{T}$ la matriz producida por un paso de la Iteración QR. Entonces $T_2 = \hat{T}$.*

Para detalles de la demostración refiérase a [8].

El algoritmo DQDS que se implementa más adelante, es matemáticamente equivalente a la Iteración LR pero sin formar explícitamente $T_{i+1} = B_i B_i^t + \tau_i^2 I$. La estrategia es formar B_{i+1} directamente a partir de B_i, sin formar cada matriz intermedia T_{i+1}.

Para simplificación de notación, sea B_i la matriz bidiagonal con diagonal principal $\alpha_1, \ldots, \alpha_n$ y superdiagonal $\beta_1, \ldots, \beta_{n-1}$ y sea B_{i+1} la matriz bidiagonal con diagonal principal $\hat{\alpha}_1, \ldots, \hat{\alpha}_n$ y superdiagonal $\hat{\beta}_1, \ldots, \hat{\beta}_{n-1}$. Se usa la convención

$$\beta_0 = \hat{\beta}_0 = \beta_n = \hat{\beta}_n.$$

Se relaciona B_i con B_{i+1} por

$$B_{i+1}^t B_{i+1} + \tau_{i+1}^2 I = T_{i+1} = B_i B_i^t + \tau_i^2 I. \tag{3.5}$$

Igualando la entrada (j, j) del lado izquierdo de la ecuación (3.5) con su lado derecho, se obtiene, para $j < n$,

$$\hat{\alpha}_j^2 + \hat{\beta}_{j-1}^2 + \tau_{i+1}^2 = \alpha_j^2 + \beta_j^2 + \tau_i^2 \quad \text{ó} \quad \hat{\alpha}_j^2 = \alpha_j^2 + \beta_j^2 - \hat{\beta}_{j-1}^2 - \delta, \tag{3.6}$$

donde $\delta = \tau_{i+1}^2 - \tau_i^2 \geq 0$. τ_i^2 debe ser escogido de tal manera que sea cercano al autovalor mas pequeño de T. Igualando el cuadrado de la entrada $(j, j+1)$ del lado izquierdo de la ecuación (3.5) con su lado derecho, se obtiene

$$\hat{\alpha}_j^2 \hat{\beta}_j^2 = \alpha_{j+1}^2 \beta_j^2 \quad \text{ó} \quad \hat{\beta}_j^2 = \alpha_{j+1}^2 \beta_j^2 / \hat{\alpha}_j^2. \tag{3.7}$$

Combinando las ecuaciones (3.6) y (3.7) se obtiene el algoritmo, que aún no es definitivo:

Algoritmo 3.11.

Para $j = 1, \ldots, n-1$

$\hat{\alpha}_j^2 = \alpha_j^2 + \beta_j^2 - \hat{\beta}_{j-1}^2 - \delta$

$\hat{\beta}_j^2 = \beta_j^2 \cdot (\alpha_{j+1}^2 / \hat{\alpha}_j^2)$

Fin Para

$\hat{\alpha}_n^2 = \alpha_n^2 - \hat{\beta}_{n-1}^2 - \delta$

Esta versión del algoritmo tiene sólo 5 operaciones en el ciclo interno, que es menos costoso computacionalmente. Esto relaciona el cuadrado de la entrada de B_i con el cuadrado de la entrada de B_{i+1}. Por ello, es conveniente trabajar con adiciones, sustraciones o multiplicaciones, en vez de aplicar raíces cuadradas o divisiones. En efecto, definiendo las variables $q_j \equiv \alpha_j^2$ y $e_j \equiv \beta_j^2$ se obtiene el algoritmo QDS (de nuevo, el nombre es dado por razones históricas).

Algoritmo 3.12. Un paso del algoritmo QDS

Para $j = 1, \ldots, n-1$

$\hat{q}_j = q_j + e_j - \hat{e}_{j-1} - \delta$

$\hat{e}_j = e_j \cdot (q_{j+1} / \hat{q}_j)$

Fin Para

$\hat{q}_n = q_n - \hat{e}_{n-1} - \delta$

En el algoritmo final DQDS se realizará casi el mismo trabajo que en QDS pero será significativamente más preciso. Tómese la subexpresión $q_j - \hat{e}_{j-1} - \delta$ de la primera línea del Algoritmo 3.12 y reescríbase esto como sigue:

$$\begin{aligned} d_j &\equiv q_j - \hat{e}_{j-1} - \delta \\ &= q_j - \frac{q_j e_{j-1}}{\hat{q}_{j-1}} - \delta \qquad \text{de} \quad (3.7) \\ &= q_j \cdot \left[\frac{\hat{q}_{j-1} - e_{j-1}}{\hat{q}_{j-1}} \right] - \delta \\ &= q_j \cdot \left[\frac{q_{j-1} - \hat{e}_{j-2} - \delta}{\hat{q}_{j-1}} \right] - \delta \quad \text{de} \quad (3.6) \\ &= \frac{q_j}{\hat{q}_{j-1}} \cdot d_{j-1} - \delta. \end{aligned}$$

Esto reescribe el ciclo interno del Algoritmo 3.12 como

$$\begin{aligned} \hat{q}_j &= d_j + e_j \\ \hat{e}_j &= e_j \cdot (q_{j+1}/\hat{q}_j) \\ d_{j+1} &= d_j \cdot (q_{j+1}/\hat{q}_j) - \delta. \end{aligned}$$

Finalmente, nótese que d_{j+1} puede sobreescribir a d_j y que $t = q_{j+1}/\hat{q}_j$ necesita ser calculada sólo una vez y así, obtener el algoritmo final DQDS.

Algoritmo 3.13. DQDS

Inicialización: $q_k^{(0)} = (\alpha_k)^2 \ \ (k = 1, \ldots, n)$; $\ e_k^{(0)} = (\beta_k)^2 \ \ (k = 1, \ldots, n-1)$

$t^{(0)} = 0$

Para $m = 0, 1, \ldots$ **Hacer**

Escoger traslado $s^{(m)} \geq 0$

$d_1^{(m+1)} = q_1^{(m)} - s^{(m)}$

Para $k = 1, \ldots, n-1$ **Hacer**

$q_k^{(m+1)} = d_k^{(m+1)} + e_k^{(m)}$

$t^{(m)} = q_{k+1}^{(m)}/q_k^{(m+1)}$

$e_k^{(m+1)} = e_k^{(m)} \cdot t^{(m)}$

$d_{k+1}^{(m+1)} = d_k^{(m+1)} \cdot t^{(m)} - s^{(m)}$

Fin Para

$q_n^{(m+1)} = d_n^{(m+1)}$

$t^{(m+1)} = t^{(m)} + s^{(m)}$

Fin Para

3.4.1 Teorema de Convergencia Global del Algoritmo DQDS

El siguiente teorema, dado por Aishima et al. [1, 2], establece que, si $0 \leq s^{(m)} < \lambda_{min}^{(m)}$ para cada iteración m, donde $\lambda_{min}^{(m)}$ denota el autovalor más pequeño

de la matriz tridiagonal simétrica $(B^{(m)})^t B^{(m)}$, entonces las variables en el algoritmo 3.13 (DQDS) converge para una matriz inicial B bidiagonal:

Teorema 3.9. *(Convergencia Global del Algoritmo DQDS). Sean una matriz B bidiagonal y una estrategia de traslado $s^{(m)}$ tal que*

$$0 \leq s^{(m)} < \lambda_{min}^{(m)} \qquad (m = 0, 1, 2, \ldots). \tag{3.8}$$

Entonces

$$\begin{aligned} \lim_{m\to\infty} e_k^{(m)} &= 0 \qquad (k = 1, 2, \ldots, n-1), \\ \lim_{m\to\infty} q_k^{(m)} + t^{(m)} &= \lambda_k \qquad (k = 1, 2, \ldots, n). \end{aligned}$$

En forma matricial, se tiene que

$$\lim_{m\to\infty} (B^{(m)})^t B^{(m)} = \mathrm{diag}(\lambda_1 - t^{(m)}, \ldots, \lambda_n - t^{(m)}).$$

Para detalles de la demostración, véase [1, 2].

A continuación, se presenta un algoritmo para obtener un desplazamiento que garantiza la convergencia del método DQDS:

Algoritmo 3.14. Estrategia de Desplazamiento para el Algoritmo DQDS

$e_0^{(m)} = 0$, $\hat{d}_0^{(m)} = 1$

Para $k = 1, \ldots, n-1$

$$\hat{d}_k^{(m+1)} = \frac{\hat{d}_{k-1}^{(m+1)} q_k^{(m)}}{\hat{d}_{k-1}^{(m+1)} + e_{k-1}^{(m)}}$$

Si $\hat{d}_k^{(m+1)} \leq 0$ **entonces**

Escoger traslado $s^{(m)} = 0$

Terminar

Fin Si

Fin Para

$$\textit{Escoger traslado } s^{(m)} = \frac{\hat{d}_{n-1}^{(m+1)} q_n^{(m)}}{\hat{d}_{n-1}^{(m+1)} + e_{n-1}^{(m)}}$$

Terminar

El desplazamiento que se obtiene del algoritmo anterior satisface la condición (3.8) del teorema de convergencia global (Teorema 3.9). Los autores de la publicación [1] (quienes crearon esta estrategia), se basaron en la propuesta de un matemático suizo llamado Heinz Rutishauser, quien estableció un desplazamiento en el contexto del método de la Iteración LR.

Teorema 3.10. *(Convergencia Cúbica con el Desplazamiento que se obtiene del Algoritmo 3.14). Para el Algoritmo DQDS (Algoritmo 3.13) con desplazamiento del Algoritmo 3.14, se tiene que*

$$\lim_{m\to\infty} \frac{e_{n-1}^{m+1}}{(e_{n-1}^{m})^3} = \frac{1}{(\lambda_{n-1}-\lambda_n)^2}. \tag{3.9}$$

Por lo tanto, la convergencia es asintóticamente cúbica.

Para detalles de la demostración, véase [1].

3.5 Método de Jacobi

Este método no comienza considerando una matriz A tridiagonal simétrica como los métodos planteados anteriormente en este capítulo sino, más bien, como una matriz densa simétrica. He aquí su implementación básica:

Dada una matriz simétrica $A = A_0$, el Método de Jacobi produce una sucesión $A_1, A_2, \ldots$ de matrices ortogonalmente similares, la cual eventualmente, converge a una matriz diagonal con los autovalores en dicha diagonal. A_{i+1} es obtenida a partir de A_i por la fórmula $A_{i+1} = J_i^t A_i J_i$, donde J_i es una matriz ortogonal llamada *rotación de Jacobi*. Así

$$\begin{aligned} A_m &= J_{m-1}^t A_{m-1} J_{m-1} \\ &= J_{m-1}^t J_{m-2}^t A_{m-2} J_{m-2} J_{m-1} = \cdots \\ &= J_{m-1}^t \cdots J_0^t A_0 J_0 \cdots J_{m-1} \\ &\equiv J^t A J. \end{aligned}$$

Si se elige cada J_i de manera apropiada, A_m tenderá a la matriz diagonal Λ, para m grande. Por lo tanto, las columnas de J serían autovectores aproximados. La idea es considerar J_i de tal manera que dos entradas fuera de la diagonal de $A_{i+1} = J_i^t A_i J_i$ se hagan cero simultáneamente. Para ello, se elige a J_i como una matriz de Givens:

$$J_i = G(j,k,\theta) \equiv \begin{array}{c} \\ j \\ \\ \\ k \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & j & & k & & \end{array} \\ \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & cos(\theta) & \cdots & -sen(\theta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & sen(\theta) & \cdots & cos(\theta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \end{array},$$

donde θ es elegido para hacer cero las entradas j,k y k,j de A_{i+1}. Para determinar $cos(\theta)$ y $sen(\theta)$, se escribe

$$\begin{aligned} \begin{bmatrix} a_{jj}^{(i+1)} & a_{jk}^{(i+1)} \\ a_{kj}^{(i+1)} & a_{kk}^{(i+1)} \end{bmatrix} &= \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}^t \begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix} \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \end{aligned}$$

donde λ_1 y λ_2 son los autovalores de $\begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix}$.

Multiplicando la última expresión, usando simetría, abreviando $c \equiv cos(\theta)$ y $s \equiv sen(\theta)$, y eliminando el supra-índice i por simplicidad, se obtiene

$$\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} a_{jj}c^2 + a_{kk}s^2 + 2sca_{jk} & sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2) \\ sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2) & a_{jj}s^2 + a_{kk}c^2 + 2sca_{jk} \end{bmatrix}.$$

Estableciendo la entrada fuera de la diagonal igual a cero y resolviendo para θ, se obtiene que $0 = sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)$, o

$$\frac{a_{jj} - a_{kk}}{2a_{jk}} = \frac{c^2 - s^2}{2sc} = \frac{cos(2\theta)}{sen(2\theta)} = \cot(2\theta) \equiv \tau.$$

Ahora, sea $t = \dfrac{s}{c} = \tan(\theta)$ y nótese que $t^2 + 2\tau t - 1 = 0$. La solución de la fórmula cuadrática es $t = \dfrac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}$ y así, $c = \dfrac{1}{\sqrt{1+t^2}}$ y $s = t \cdot c$. Se resume este procedimiento en el siguiente algoritmo:

Algoritmo 3.15. Calculando y aplicando una rotación de Jacobi a A en coordenadas j, k

Entrada: *matriz* $A \in \mathbb{R}^{n \times n}$

Salida: *matriz* $A \in \mathbb{R}^{n \times n}$ *y matriz ortogonal* $J \in \mathbb{R}^{n \times n}$

$J = \mathbb{I}_{n \times n}$

Si $|a_{jk}|_2$ *no es tan pequeño*

$\tau = \dfrac{(a_{jj} - a_{kk})}{2a_{jk}}$

$t = \dfrac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}$

$c = \dfrac{1}{\sqrt{1+t^2}}$

$s = c \cdot t$

$A = G^t(j,k,\theta) \cdot A \cdot G(j,k,\theta)$ *donde* $c = \cos(\theta)$ *y* $s = \sin(\theta)$

Si *se desea obtener los autovectores*

$J = J \cdot G(j,k,\theta)$

Fin Si

Fin Si

El costo de aplicar $G(j,k,\theta)$ a A (ó a J) es de $O(n)$ operaciones, porque sólo las filas y columnas j y k de A (y columnas j y k de J) son modificadas (véase Algoritmo 1.2 y Algoritmo 1.3 planteados en el capítulo 1). He aquí el

algoritmo general de Jacobi (refiérase a [29] para detalles acerca de su convergencia asintóticamente cuadrática):

Algoritmo 3.16. Método de Jacobi para la descomposición en autovalores de una matriz simétrica A

Entrada: *matriz* $A \in \mathbb{R}^{n\times n}$

Salida: *matriz* $A \in \mathbb{R}^{n\times n}$ *diagonal y matriz* $V \in \mathbb{R}^{n\times n}$ *ortogonal*

$V = \mathbb{I}_{n\times n}$

Repetir

Elegir un par j, k

Calcular A *y* J *aplicando Algoritmo 3.15 (Rotación Jacobi) a* A

Si *se desea obtener los autovectores*

$V = V \cdot J$

Hasta *que* A *sea suficientemente diagonal*

CAPÍTULO 4

MÉTODOS NUMÉRICOS PARA LA SVD DE MATRICES

En este capítulo se presentan algunos métodos numéricos que permiten la obtención de la SVD de matrices (cuadradas y rectangulares) a valores reales. Estos son variantes de los algoritmos para el cálculo de autovalores que se mostraron en el capítulo anterior. Se describe en forma estándar y general, la característica estructural que puede tener un método para el cálculo de la SVD. Seguido a esto, se detalla el proceso de Bidiagonalización, basado en reflectores de Householder, propuesto por Gene Golub y William Kahan en el año 1965. En primera instancia, se muestra el método SVD que publicaron Gene Golub, William Kahan y Christian Reinsch en el año 1970, el cual se basa en el método de Iteración QR Implícito con Desplazamiento Wilkinson (ver Sección 3.1.3), para luego, ofrecer dos de sus variantes: SVD Chan, publicado por Tony Chan en el año 1982; y otro presentado en el año 1990 por James Demmel y William Kahan, el cual se basa en el método de Golub-Kahan-Reinsch pero con desplazamiento igual a cero. Por otro lado, se expone un lema de alta importancia que relaciona la descomposición en autovalores de matrices tridiagonales simétricas con el problema SVD de una matriz bidiagonal. Usando dicho lema, se proponen tres métodos para la SVD de matrices rectangulares a valores reales: Divide y Vencerás, Bisección e Iteración Inversa, y DQDS. Este

último está propuesto sólo para el cálculo de valores singulares. Finalmente, se plantea el Método de Jacobi para matrices cuadradas, que data de finales del siglo XIX, cuya estructura no sigue el patrón estándar de los métodos mencionados anteriores. Para mayores detalles acerca de estos métodos véase [1, 2, 7, 8, 9, 10, 13, 27].

4.1 Estructura General para la Obtención de la SVD de una Matriz A Cualquiera

A continuación se presenta la estructura estándar que puede seguir un método para el cálculo de la SVD de una matriz A general:

- Reducir A a la forma bidiagonal B usando matrices ortogonales U_1 y V_1 : $A = U_1 B V_1^t$.
- Obtener la SVD de B: $B = U_2 \Sigma V_2^t$, donde Σ es la matriz diagonal de valores singulares y, U_2 y V_2 son matrices ortogonales cuyas columnas son los vectores singulares izquierdos y derechos de B, respectivamente.
- Combinar estas descomposiciones para obtener $A = (U_1 U_2)\Sigma(V_1 V_2)^t$. Las columnas de $U = U_1 U_2$ y $V = V_1 V_2$ son los vectores singulares izquierdos y derechos de A, respectivamente.

4.2 Bidiagonalización

Definición 4.1. *Sea $A \in \mathbb{R}^{m\times n}$ tal que $m \geq n$. Definimos la representación bidiagonal de A como*

$$U^t A V = B = \begin{pmatrix} B_{1(n\times n)} \\ 0 \end{pmatrix}$$

donde

$$U = [u_1, \ldots, u_m] \qquad U^t U = I_{m\times m}$$

$$V = [v_1, \ldots, v_n] \qquad V^t V = I_{n\times n}$$

y

$$B_1 = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & & 0 & \alpha_n \end{pmatrix}.$$

4.2.1 Bidiagonalización vía Reflectores de Householder

Definición 4.2. *Una matriz de la forma*

$$H = I - \frac{2uu^t}{u^t u}$$

donde u es un vector no nulo, es llamado matriz Householder o reflector elemental. El vector u es llamado vector Householder.

Algunas propiedades

- H no altera la longitud de un vector, es decir, $\|Hx\|_2 = \|x\|_2$, para cada $x \in \mathbb{R}^n$.
- H es una matriz ortogonal. Consecuencia de la propiedad anterior.
- $H^2 = I$

La importancia de las matrices Householder viene del hecho de que éstas pueden ser usadas para crear ceros, a gran escala, en un vector.

Lema 4.3. *Dado un vector no nulo $x \neq e_1$, entonces existe siempre una matriz Householder H tal que Hx es un múltiplo de e_1.*

Véase [7] para detalles de la demostración.

Del lema anterior, basta considerar $u = x + \text{sign}(x_1)\|x\|_2 e_1$. Con esto, se describe el siguiente algoritmo que calcula un vector u y un escalar σ tal que $Hx = (I - \frac{2uu^t}{u^t u})x = (\sigma, 0, \ldots, 0)^t$:

Algoritmo 4.1. Creando ceros en un vector con una matriz Householder

Entrada: *vector* $x \in \mathbb{R}^n$

Salida: *vector* $u \in \mathbb{R}^n$ *y escalar* σ

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

$m = \text{máx}(|x_i|)$

$u_i = \dfrac{x_i}{m}$

$\sigma = \text{sign}(u_1)\sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}$

$u_1 = u_1 + \sigma$

$\sigma = -m \cdot \sigma$

Fin Para

Este algoritmo tiene un costo de $2(n + 1)$ operaciones mas el cálculo de una raíz cuadrada. Ahora se muestra el siguiente algoritmo eficiente, el cual consiste en calcular el producto HA sin formar explícitamente la matriz de Householder H:

Algoritmo 4.2. Pre-Multiplicación por una matriz Householder

Entrada: *matriz* $A \in \mathbb{R}^{m \times n}$ *y vector* $u \in \mathbb{R}^m$

Salida: *matriz* $A \in \mathbb{R}^{m \times n}$

Calcular $\beta = \dfrac{2}{u^t u}$

Para $j = 1, 2, \ldots, n$ **Hacer**

$\alpha = u_1 a_{1j} + u_2 a_{2j} + \cdots + u_m a_{mj}$

$\alpha = \beta \cdot \alpha$

Para $i = 1, 2, \ldots, m$ **Hacer**

$a_{ij} = a_{ij} - \alpha \cdot u_i$

Fin Para

Fin Para

Este algoritmo tiene un costo de $(m + 1) + 2mn + n$ operaciones. En particular, si $m = n$, entonces toma aproximadamente $2n^2$ operaciones,

comparados con los n^3 que se realizan formando la matriz de Householder explícitamente.

Análogamente, es posible implementar una rutina que permita calcular el producto AH sin formar explícitamente la matriz H:

Algoritmo 4.3. Post-Multiplicación por una matriz Householder

Entrada: *matriz* $A \in \mathbb{R}^{m\times n}$ *y vector* $u \in \mathbb{R}^n$
Salida: *matriz* $A \in \mathbb{R}^{m\times n}$
  Calcular $\beta = \dfrac{2}{u^t u}$
  Para $j = 1, 2, \ldots, m$ **Hacer**
    $\alpha = u_1 a_{j1} + u_2 a_{j2} + \cdots + u_n a_{jn}$
    $\alpha = \beta \cdot \alpha$
    Para $i = 1, 2, \ldots, n$ **Hacer**
      $a_{ji} = a_{ji} - \alpha \cdot u_i^t$
    Fin Para
  Fin Para

En el proceso de Bidiagonalización vía Householder, las matrices U y V que establece la definición 4.1, están dadas por $U = U_1 \cdots U_n$ y $V = V_1 \cdots V_{n-2}$, donde U_k, para $k = 1, \ldots, n$, y V_k, para $k = 1, \ldots, n-2$, son matrices de Householder. En general, U_k introduce ceros en la k-ésima columna de A, mientras que V_k los aplica a la k-ésima fila de A. Por ello, Dado $A \in \mathbb{R}^{m\times n}$, con $m \geq n$, el siguiente algoritmo transforma A a $U^t A V = B$, donde B es bidiagonal superior, $U = U_1 \cdots U_n$ y $V = V_1 \cdots V_{n-2}$. La parte esencial de los vectores Householder de los U_j están almacenados en $A(j+1 : m, j)$ y la parte esencial de los vectores Householder de los V_j están almacenados en $A(j, j+2 : n)$.

Algoritmo 4.4. Bidiagonalización Householder

Entrada: *matriz* $A \in \mathbb{R}^{m\times n}$

Salida: $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ *y* $A \in \mathbb{R}^{m\times n}$

Para $j = 1, \ldots, n$

Obtener u *y* σ *aplicando el Algoritmo 4.1 (ceros con Householder)*

al vector $A(j : m, j)$

Calcular $A = HA$ *aplicando el Algoritmo 4.2 (Premultipl. Householder)*

a $A(j : m, j : n)$ $A(j + 1 : m, j) = u(2 : m - j + 1)$

Acumular el producto de matrices Householder U_j

Calcular $U = UH$ *usando el Algoritmo 4.3 (Postmultipl. Householder)*

Si $j \leq n - 2$

Obtener u *y* σ *aplicando el Algoritmo 4.1 (ceros con Householder)*

al vector $A(j, j + 1 : n)^t$

Calcular $A = AH$ *aplicando el Algoritmo 4.3 (Postmultipl. Householder)*

a $A(j : m, j + 1 : n)$

$A(j, j + 2 : n) = u(2 : n - j)^t$

Acumular el producto de matrices Householder V_j

Calcular $V = VH$ *usando el Algoritmo 4.3 (Postmultipl. Householder)*

Fin Si

Fin Para

Este algoritmo requiere de $4mn^2 - 4n^3/3$ operaciones para conseguir únicamente la matriz bidiagonal. Si las matrices U y V se calculan explícitamente, entonces estos pueden ser acumuladas en $4m^2n - 4n^3/3$ y $4n^3/3$ operaciones, respectivamente. Por otro lado, es importante mencionar que, a parte de la Bidiagonalización vía Householder, existe otra manera de bidiagonalizar una matriz A general: vía Lanczos (véase [13] para detalles de este método), pero por razones de inestabilidad numérica este último no es planteado.

4.3 Método SVD Golub-Kahan-Reinsch (1970)

El siguiente algoritmo es hoy, un algoritmo modelo para calcular los valores singulares y vectores singulares de una matriz general A. Está estructurado en dos fases: Fase I como proceso finito y Fase II como proceso iterativo:

Fase I. La matriz A de $m \times n$ $(m \geq n)$, es transformada a una matriz bidiagonal superior B por equivalencia ortogonal.

Fase II. La matriz bidiagonal B es, además, reducida por equivalencia ortogonal a una matriz diagonal Σ usando el método Iteración QR Implícito simétrico con Desplazamiento Wilkinson.

La Fase I correspondiente al proceso de Bidiagonalización ha sido descrito en la sección 4.2.1. A continuación, se describe el proceso para hallar la SVD de una matriz bidiagonal asociado a la Fase II:

4.3.1 Consiguiendo la SVD de una Matriz Bidiagonal

El método es una variante de la Iteración QR. Sea $B \in \mathbb{R}^{n \times n}$ una matriz inicial bidiagonal obtenida en la Fase I, el proceso consiste en construir una sucesión de matrices bidiagonales $\{B_i\}_{i \in \mathbb{N}}$, tal que, cada B_i tenga las entradas de la superdiagonal más pequeñas, en valor absoluto, que las matrices bidiagonales anteriores de la sucesión. Por el Teorema Q Implícito (teorema 3.3) se tiene que la i-ésima iteración es equivalente a aplicar el método de Iteración QR Implícito Simétrico con Desplazamiento Wilkinson a la matriz tridiagonal simétrica $B_i^t B_i$ (Algoritmo 3.3) [1], descrito en la sección 3.1.3, pero sin formar explícitamente la matriz $B_i^t B_i$. Véase [13] para esto último.

[1] Si $B \in \mathbb{R}^{n \times n}$ es una matriz bidiagonal entonces la matriz $T = B^t B \in \mathbb{R}^{n \times n}$ es una matriz tridiagonal simétrica.

Algoritmo 4.5. Un paso SVD Bidiagonal

Entrada: $B \in \mathbb{R}^{n\times n}$ *bidiagonal*

Salida: $B \in \mathbb{R}^{n\times n}$ *bidiagonal y,* Z_1*,* Z_2 *ortogonales*

$Z_1 = Z_2 \in \mathbb{I}^{n\times n}$

$d = (B_{(n-1,n-1)} - B_{(n,n)})/2$

$\mu = B_{(n,n)} + d - \text{sign}(d)\sqrt{d^2 + B^2_{(n-1,n)}}$ *(traslado Wilkinson)*

$x = B^2_{(1,1)} - \mu$

$z = B_{(1,1)}B_{(2,1)}$

Para $k = 1, \ldots, n-1$

Obtener parámetros de Givens c_k *y* s_k *usando Algoritmo 1.1 para* $\mathbf{x} = (x \;\; z)^t$

Calcular $B = BJ_k(k, k+1, \theta)$

Aplicar Algoritmo 1.3 (Post. Givens) a B *usando* c_k *y* s_k*, e índices* k *y* $j = k+1$

Acumulando los $J_k(k, k+1, \theta)$ *aplicando Algoritmo 1.3 (Post. Givens)*

$Z_1 = Z_1 J_k(k, k+1, \theta)$

$x = B_{k,k}$

$z = B_{k+1,k}$

Obtener parámetros de Givens c_k *y* s_k *usando Algoritmo 1.1 para* $\mathbf{x} = (x \;\; z)^t$

Calcular $B = J_{2k}(k, k+1, \theta)B$

Aplicar Algoritmo 1.2 (Prem. Givens) a B *con parámetros* c_k *y* s_k*, e índices* k *y* $j = k+1$

Acumulando los $J_{2k}(k, k+1, \theta)$ *aplicando Algoritmo 1.2 (Prem. Givens)*

$Z_2 = J_{2k}(k, k+1, \theta)Z_2$

Si $k < n-1$ **Hacer**

$x = B_{k,k+1}$

$z = B_{k,k+2}$

Fin Si

Fin Para

Deflación y Convergencia

El algoritmo anterior representa un paso iterativo del algoritmo para el cálculo de la SVD de una matriz bidiagonal propuesto por Golub, Kahan y Reinsch. Al repetir dicho algoritmo, se logra construir una sucesión de matrices bidiagonales $\{B_k\}$ con entradas diagonal $\alpha_1^{\{k\}}, \ldots, \alpha_n^{\{k\}}$ y superdiagonal $\beta_2^{\{k\}}, \ldots, \beta_n^{\{k\}}$. Por la convergencia cúbica del algoritmo de Iteración QR implícito con traslado Wilkinson se sigue que $\beta_n^{\{k\}}$ converge a cero cúbicamente. Teniendo $\beta_n^{\{k\}}$ cercano a cero (bajo cierto criterio de parada definido en el próximo algoritmo), a la matriz resultante se le aplica una *deflación*, la cual consiste en ejecutar, repetidamente, el algoritmo en una o más matrices de orden $(n-1)$ o inferior a la original. El proceso se continúa hasta que B sea suficientemente diagonal.

Algoritmo 4.6. SVD Bidiagonal

Entrada: $B \in \mathbb{R}^{n\times n}$ *matriz bidiagonal con entrada diagonal* $\alpha_1, \ldots, \alpha_n$, *y superdiagonal* $\beta_2, \ldots, \beta_n$

Salida: $U_2 \in \mathbb{R}^{n\times n}$ *matriz ortogonal,* $\Sigma \in \mathbb{R}^{n\times n}$ *matriz diagonal y* $V_2 \in \mathbb{R}^{n\times n}$ *matriz ortogonal*

$W_1 = \mathbb{I}_{n\times n}$

$W_2 = \mathbb{I}_{n\times n}$

Para $j = 1, \ldots, n-1$

Mientras $|\beta_{n+1-j}| \geq \epsilon(|\alpha_{n+1-j}| + |\alpha_{n-j}|)$ **Hacer**

Calcular $B_{(1:n+1-j,1:n+1-j)} = Z_1^t B_{(1:n+1-j,1:n+1-j)} Z_2$

Obtener Z_1, Z_2, *y* $B \in \mathbb{R}^{(n+1-j)\times(n+1-j)}$ *aplicando Algoritmo 4.5 (Un paso SVD Bidiagonal) a* $B \in \mathbb{R}^{(n+1-j)\times(n+1-j)}$

$$Z_1 = \begin{pmatrix} Z_1 & 0 \\ 0 & \mathbb{I}_{(j-1)\times(j-1)} \end{pmatrix}$$

$$Z_2 = \begin{pmatrix} Z_2 & 0 \\ 0 & \mathbb{I}_{(j-1)\times(j-1)} \end{pmatrix}$$

Acumulando Z_1 *y* Z_2

$W_1 = Z_1 W_1$

$W_2 = W_2 Z_2$

Fin Mientras

Fin Para

$U_1 = W_1$

$V_2 = W_2$

Para $i = 1, \ldots, n$

$\sigma_i = B_{i,i}$

Fin Para

$\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$

A continuación se plantea el algoritmo definitivo para el cálculo de la SVD de una matriz general A de $m \times n$ propuesto por Golub, Kahan y Reinsch:

Algoritmo 4.7. SVD Golub-Kahan-Reinsch

Entrada: *matriz* $A \in \mathbb{R}^{m \times n}$*, con* $m \geq n$

Salida: *matriz* $U \in \mathbb{R}^{m \times m}$ *ortogonal, matriz* $\Sigma \in \mathbb{R}^{m \times n}$ *diagonal y matriz* $V \in \mathbb{R}^{n \times n}$ *ortogonal*

Fase I: Bidiagonalización de A

Obtener matrices $U_1 \in \mathbb{R}^{m \times m}$ *y* $V_1 \in \mathbb{R}^{n \times n}$ *ortogonales y matriz bidiagonal superior* $B = \begin{pmatrix} \hat{B} \\ 0_{m-n} \end{pmatrix} \in \mathbb{R}^{m \times n}$ *aplicando Algoritmo 4.4 (Bidiag. Householder) a la matriz* A

Fase II: Obteniendo la SVD de $\hat{B}$*:* $\hat{B} = U_2 \hat{\Sigma} V_2^t$

Obtener matrices $U_2 \in \mathbb{R}^{n \times n}$ *y* $V_2 \in \mathbb{R}^{n \times n}$ *ortogonales y matriz diagonal* $\hat{\Sigma} \in \mathbb{R}^{n \times n}$ *aplicando Algoritmo 4.6 (SVD Bidiagonal) a matriz bidiagonal* $\hat{B}$

$U = U_1 \text{diag}(U_2, \mathbb{I}_{(m-n)})$

$V = V_1 V_2$

$\Sigma = \begin{pmatrix} \hat{\Sigma} \\ 0_{m-n} \end{pmatrix}$

El costo del algoritmo es dominado por el costo de la Fase I. El estimado total es de $2m^2n + 4mn^2 + \frac{9}{2}n^3$ operaciones, con $m \geq n$. Este conteo incluye el costo de obtener U, Σ, y V. Si sólo se necesitan los valores singulares, entonces el estimado de las operaciones es de $2mn^2 - 2\frac{n^3}{3}$. Además, estos valores singulares, producidos por el Algoritmo SVD Golub-Kahan-Reinsch (Algoritmo 4.7), son los valores singulares exactos de la matriz $A + E$, donde $\|E\|_2 \leq c(m,n)\|A\|_2$. Allí, $c(m,n)$ es una constante que depende de m y n. El algoritmo es, por tanto, estable numéricamente. Véase [7] para detalles.

4.4 Método SVD Chan

Tony Chan en 1982 observó que el algoritmo SVD Golub-Kahan-Reinsch, descrito anteriormente, puede ser mejorado en el caso $m \gg n$ si a la matriz A original, primeramente, se le aplica una factorización QR y luego, a la matriz triangular superior R resultante de dicha factorización, se le bidiagonaliza. La mejoría, naturalmente, viene del hecho de que el trabajo requerido para bidiagonalizar la matriz triangular R es mucho menos costoso computacionalmente que el trabajo requerido para bidiagonalizar la matriz A original. Por supuesto, una vez obtenida la SVD de R, se puede fácilmente recuperar la SVD de A. Es por ello que, a continuación se presenta el proceso SVD Chan:

1. Calcular la factorización QR de A:

$$Q^t A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}.$$

2. Conseguir la SVD de R usando el algoritmo SVD Golub-Kahan-Reinsch:

$$R_1 = X\Sigma Y^t.$$

Entonces, los valores singulares de A son justamente los valores singulares de R_1. Las matrices U y V de los vectores singulares izquierdos y derechos de A, respectivamente, están dadas por

$$U = Q \cdot \operatorname{diag}(X, I_{m-n}) \quad \text{y} \quad V = Y.$$

Conteo de operaciones. El proceso SVD Chan requiere cerca de $2m^2n+11n^3$ operaciones para calcular Σ, U, y V, comparado con los $2m^2n + 4mn^2 + \frac{9}{2}n^3$ requeridos por el algoritmo SVD Golub-Kahan-Reinsch.

4.5 Método Demmel-Kahan

En 1990, Demmel y Kahan presentaron un nuevo algoritmo que calcula todos los valores singulares de una matriz bidiagonal con alta precisión. Ellos llamaron este algoritmo Iteración QR con Desplazamiento Cero. La razón de este nombre se debe a que el nuevo algoritmo corresponde al algoritmo SVD Golub-Kahan-Reinsch pero cuando el traslado es cero. Este es basado en la observación de que, cuando el desplazamiento es cero, no ocurre cancelación debido a la sustración, por lo que, los valores singulares muy pequeños, se pueden obtener con mayor precisión. El efecto del desplazamiento cero es notable. Refiérase a [7] para detalles. A continuación se presenta el algoritmo SVD Demmel-Kahan, una variante del algoritmo SVD Golub-Kahan-Reinsch:

Algoritmo 4.8. SVD Demmel-Kahan

Entrada: *matriz* $A \in \mathbb{R}^{m\times n}$*, con* $m \geq n$

Salida: *matriz* $U \in \mathbb{R}^{m\times m}$ *ortogonal, matriz* $\Sigma \in \mathbb{R}^{m\times n}$ *diagonal y*
matriz $V \in \mathbb{R}^{n\times n}$ *ortogonal*

Fase I: Bidiagonalización de A

Obtener matrices $U_1 \in \mathbb{R}^{m\times m}$ *y* $V_1 \in \mathbb{R}^{n\times n}$ *ortogonales y matriz bidiagonal superior* $B = \begin{pmatrix} \hat{B} \\ 0_{m-n} \end{pmatrix} \in \mathbb{R}^{m\times n}$ *aplicando Algoritmo 4.4 (Bidiag. Householder) a la matriz A*

Fase II: Obteniendo la SVD de $\hat{B}$*:* $\hat{B} = U_2 \hat{\Sigma} V_2^t$

Obtener matrices $U_2 \in \mathbb{R}^{n\times n}$ *y* $V_2 \in \mathbb{R}^{n\times n}$ *ortogonales y matriz diagonal* $\hat{\Sigma} \in \mathbb{R}^{n\times n}$ *aplicando Algoritmo 4.5 (Un paso SVD Bidiagonal) para* $\mu = 0$ *(desplaz. cero) y luego Algoritmo 4.6 (SVD Bidiagonal) a matriz bidiagonal* $\hat{B}$

$U = U_1 \text{diag}(U_2, \mathbb{I}_{(m-n)})$

$V = V_1 V_2$

$$\Sigma = \begin{pmatrix} \hat{\Sigma} \\ 0_{m-n} \end{pmatrix}$$

4.6 Métodos para el Cálculo de la SVD de Matrices usando una Descomposición en Autovalores de Matrices Tridiagonales Simétricas

En las proposiciones 2.13 y 2.14, presentadas en el capítulo 2, se pudo apreciar la estrecha relación existente entre la SVD de una matriz general y la descomposición en autovalores de las matrices simétricas A^tA, AA^t y $\begin{pmatrix} 0 & A^t \\ A & 0 \end{pmatrix}$. Por lo tanto, los algoritmos que se implementan en esta sección están basados en algunas de estas relaciones. El siguiente lema muestra otra manera de resolver el problema de conseguir la SVD de una matriz bidiagonal B, pero ahora, por medio de una descomposición en autovalores de matrices tridiagonales simétricas.

Lema 4.4. *Sea* B *una matriz bidiagonal de* $n\times n$*, con diagonal* $\alpha_1, \ldots, \alpha_n$ *y superdiagonal* $\beta_1, \ldots, \beta_{n-1}$*. Entonces existen tres maneras de convertir el problema de conseguir la SVD de* B *consiguiendo los autovalores y autovectores de una matriz tridiagonal simétrica.*

- *Sea* $A = \begin{pmatrix} 0 & B^t \\ B & 0 \end{pmatrix}$*. Sea* P *la matriz de permutación* $P = [e_1, e_{n+1}, e_2, e_{n+2}, \ldots, e_n, e_{2n}]$*, donde* e_i *es la* i*-ésima columna de la matriz*

identidad de $2n \times 2n$. *Entonces* $T_{ps} \equiv P^t A P$ *es tridiagonal simétrica. Se puede probar que la diagonal principal de* T_{ps} *todos son ceros y su respectiva superdiagonal y subdiagonal son de la forma* $\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \beta_{n-1}, \alpha_n$. *Si* $T_{ps} x_i = \lambda_i x_i$ *es un autopar para* T_{ps}, *con* x_i *un vector unitario, entonces* $\lambda_i = \pm\sigma_i$, *donde* σ_i *es un valor singular de* B, *y* $Px_i = \frac{1}{\sqrt{2}} \begin{pmatrix} v_i \\ \pm u_i \end{pmatrix}$, *donde* u_i *y* v_i *son los vectores singulares izquierdos y derechos de* B, *respectivamente.*

- *Sea* $T_{BB^t} \equiv BB^t$. *Entonces* T_{BB^t} *es tridiagonal simétrica con diagonal* $\alpha_1^2 + \beta_1^2, \alpha_2^2 + \beta_2^2, \ldots, \alpha_{n-1}^2 + \beta_{n-1}^2, \alpha_n^2$ *y, superdiagonal y subdiagonal* $\alpha_2\beta_1, \alpha_3\beta_2, \ldots, \alpha_n\beta_{n-1}$. *Los valores singulares de* B *son las raíces cuadradas de los autovalores de* T_{BB^t}, *y los vectores singulares izquierdos de* B *son los autovectores de* T_{BB^t}. T_{BB^t} *no contiene información acerca de los vectores singulares derechos de* B.

- *Sea* $T_{B^tB} \equiv B^tB$. *Entonces* T_{B^tB} *es tridiagonal simétrica con diagonal* $\alpha_1^2, \alpha_2^2 + \beta_1^2, \alpha_3^2 + \beta_2^2, \ldots, \alpha_n^2 + \beta_{n-1}^2$ *y, superdiagonal y subdiagonal* $\alpha_1\beta_1, \alpha_2\beta_2, \ldots, \alpha_{n-1}\beta_{n-1}$. *Los valores singulares de* B *son las raíces cuadradas de los autovalores de* T_{B^tB}, *y los vectores singulares derechos de* B *son los autovectores de* T_{B^tB}. T_{B^tB} *no contiene información acerca de los vectores singulares izquierdos de* B.

Véase Proposición 2.13 y Proposición 2.14.

4.6.1 SVD Divide y Vencerás

El método para hallar todos los valores singulares y vectores singulares de una matriz $A \in \mathbb{R}^{m \times n}$ usando el algoritmo recursivo Divide y Vencerás (Algoritmo 3.5), descrito en la sección 3.2, suele ser eficiente cuando la dimensión de la matriz es considerablemente grande. Sin embargo, este método no garantiza que pequeños valores singulares sean calculados con alta precisión.

A continuación, se expone el algoritmo asociado a la SVD de una matriz general A haciendo uso de dicho método recursivo:

Algoritmo 4.9. SVD Divide y Vencerás

Entrada: $A \in \mathbb{R}^{m\times n}$, $m \geq n$

Salida: $U \in \mathbb{R}^{m\times m}$ *ortogonal,* $\Sigma \in \mathbb{R}^{m\times n}$ *diagonal y* $V \in \mathbb{R}^{n\times n}$ *ortogonal*

Bidiagonalización de A

Obtener $U_1 \in \mathbb{R}^{m\times m}$ *y* $V_1 \in \mathbb{R}^{n\times n}$ *ortogonales y* $B = \begin{pmatrix} \hat{B} \\ 0_{m-n} \end{pmatrix} \in \mathbb{R}^{m\times n}$

bidiagonal superior aplicando Algoritmo 4.4 (Bidiag. Househ.) a A

Obteniendo la SVD de $\hat{B}$*:* $\hat{B} = U_2\hat{\Sigma}V_2^t$

Formación tridiagonal

Obtener $T = B^tB \in \mathbb{R}^{n\times n}$ *usando información de parte 3. del Lema 4.4*

Obtener $\Lambda \in \mathbb{R}^{n\times n}$ *y* $Q_2 \in \mathbb{R}^{n\times n}$ *tal que* $Q_2^tTQ_2 = \Lambda$ *sea diagonal*

Aplicar Algoritmo 3.5 (Divide y Vencerás) a T

$V_2 = Q_2$ *(Ver Lema 4.4) parte 3.*

Para $i = 1, \ldots, n$

$\sigma_i = \sqrt{\Lambda(i,i)}$

Fin Para

$\hat{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$

$\Sigma = \begin{pmatrix} \hat{\Sigma} \\ 0_{m-n} \end{pmatrix}$

Calculando las primeras n columnas ortogonales ($\in \mathbb{R}^m$*) de* U_2

usando la relación $B = U_2\Sigma V_2^t$

Para $j = 1, \ldots, n$

$U_{2j} = \dfrac{1}{\sigma_i}BV_{2j}$

Fin Para

Completando las últimas $m-n$ *columnas ortogonales (*$\in \mathbb{R}^m$*) de* U_2

usando el proceso Gram-Schmidt Modificado

Aplicar Algoritmo 3.8 (G-S. Mod.) a una matriz cualquiera de $m\times(m-n)$ *para*

obtener las $m-n$ *columnas restantes de* U_2 *ortogonales a las otras n columnas*

$U = U_1U_2$

$V = V_1 V_2$

4.6.2 SVD Bisección e Iteración Inversa

Este método calcula y garantiza los valores singulares con alta precisión si se aplica el Algoritmo 3.9 (Bisección e Iteración Inversa) a la matriz tridiagonal simétrica T_{ps} referida en la parte 1. del Lema 4.4, mientras que los vectores singulares podrían sufrir ocasionalmente alguna pérdida de ortogonalidad. Sin embargo, estos inconvenientes pueden ser resueltos si se reortogonalizan dichos vectores usando el proceso Gram-Schmidt Modificado. Es posible implementar el algoritmo para las matrices tridiagonales B^tB (análogo al algoritmo SVD Divide y Vencerás) o BB^t, pero ahora se muestra para la matriz T_{ps}:

Algoritmo 4.10. SVD Bisección e Iteración inversa

Entrada: $A \in \mathbb{R}^{m\times n}$, $m \geq n$

Salida: $U \in \mathbb{R}^{m\times m}$ *ortogonal,* $\Sigma \in \mathbb{R}^{m\times n}$ *diagonal y* $V \in \mathbb{R}^{n\times n}$ *ortogonal*

Bidiagonalización de A

Obtener $U_1 \in \mathbb{R}^{m\times m}$ *y* $V_1 \in \mathbb{R}^{n\times n}$ *ortogonales y* $B \in \mathbb{R}^{m\times n}$

bidiagonal superior aplicando Algoritmo 4.4 (Bidiag. Hous.) a A

Obteniendo la SVD de B: $B = U_2 \Sigma V_2^t$

Formación tridiagonal

Obtener $T = P^t \begin{pmatrix} 0 & B^t \\ B & 0 \end{pmatrix} P \in \mathbb{R}^{(m+n)\times(m+n)}$ *usando información de parte 1.*

del Lema 4.4

Obtener $\Lambda \in \mathbb{R}^{(m+n)\times(m+n)}$ *y* $Q_2 \in \mathbb{R}^{(m+n)\times(m+n)}$ *tal que* $Q_2^t T Q_2 = \Lambda$

sea diagonal

Aplicar Algoritmo 3.9 (Bisección e Iteración Inversa) a T

$W = \sqrt{2} P Q_2$

Para $i = 1, \ldots, n$

$V_{2i} = W(1:n, 1)$

$U_{2i} = W(n+1:m, 1)$

Fin Para

Para $i = 1, \ldots, n$

$\sigma_i = \sqrt{\Lambda(i,i)}$ *(asumiendo que los* $\Lambda(i,i)$ *están ordenados en forma decreciente)*

Fin Para

$\hat{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n)$

$$\Sigma = \begin{pmatrix} \hat{\Sigma} \\ 0_{m-n} \end{pmatrix}$$

Completando las últimas $m - n$ *columnas ortogonales* ($\in \mathbb{R}^m$) *de* U_2 *usando el proceso Gram-Schmidt Modificado*

Aplicar Algoritmo 3.8 (G-S. Modif.) a una matriz cualquiera de $m \times (m-n)$ *para obtener las* $m - n$ *columnas restantes de* U_2 *ortogonales a las otras* n *columnas*

$U = U_1 U_2$

$V = V_1 V_2$

4.6.3 SVD DQDS

Este método está diseñado para calcular únicamente los valores singulares de una matriz A de $m \geq n$. En la sección 3.4, se analizó la importancia de este proceso en relación a la Iteración LR, la cual permite construir, iterativamente, una sucesión de matrices bidiagonales B_i tendiendo, en la diagonal, a los autovalores de la matriz tridiagonal simétrica $B_0^t B_0$ (véase Teorema 3.9), y sin formar explícitamente, la sucesión de matrices tridiagonales $B_i^t B_i$. He aquí el algoritmo SVD DQDS para el cálculo de los valores singulares:

Algoritmo 4.11. SVD DQDS

Entrada: *matriz* $A \in \mathbb{R}^{m \times n}$, $m \geq n$

Salida: *vector* $\Sigma \in \mathbb{R}^n$

Bidiagonalización de A

Obtener matriz bidiagonal superior $B \in \mathbb{R}^{m \times n}$ *aplicando*

Algoritmo 4.4 (Bidiag. Hous.) a la matriz A

Obteniendo los valores singulares de B: $\Sigma = (\sigma_1, \ldots, \sigma_n)$

Obtener matriz diagonal B aplicando Algoritmo 3.13 (DQDS) a la matriz bidiagonal B

Para $i = 1, \ldots, n$

$\sigma_i = \sqrt{B(i,i)}$

Fin Para

$\Sigma = (\sigma_1, \ldots, \sigma_n)$

4.7 Método de Jacobi

Este método opera sobre la matriz original sin comenzar reduciéndolo a su forma bidiagonal como el resto de los algoritmos para la SVD planteados en este capítulo. En esta sección se muestra cómo aplicar dicho método para conseguir la SVD de una matriz G cualquiera de $n \times n$ aplicando Algoritmo 3.16 del capítulo anterior, a la matriz simétrica $A = G^tG$, de manera implícita. En otras palabras, en cada paso se calcula una rotación de Jacobi y se actualiza implícitamente G^tG a J^tG^tGJ, donde J es escogido de tal manera que dos entradas fuera de la diagonal de G^tG sean ceros en J^tG^tGJ. Pero en vez de calcular G^tG o J^tG^tGJ explícitamente, sólo se calcula GJ. Por esta razón, el siguiente algoritmo es llamado *Rotación Jacobi a un lado*. Además, para finalizar el proceso iterativo se propone considerar como criterio de parada el hecho de que $a_{ij}/(a_{ii}a_{jj})^{1/2}$ sea bien pequeño (véase [8, 10] para detalles acerca de este método):

Algoritmo 4.12. SVD Rotación Jacobi a un lado

Entrada: $A \in \mathbb{R}^{n \times n}$ *y tolerancia tol*

Salida: $U \in \mathbb{R}^{n \times n}$ *ortogonal,* $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}$ *diagonal*

y $V \in \mathbb{R}^{n \times n}$ *ortogonal*

$V = \mathbb{I}_{n \times n}$

Repetir

Para *todo par* $i < j$

Calcular $\begin{bmatrix} a & c \\ c & b \end{bmatrix} \equiv$ *la submatriz* (i, j) *de* $G^t G$

$$a = \sum_{k=1}^{n} G_{ki}^2$$

$$b = \sum_{k=1}^{n} G_{kj}^2$$

$$c = \sum_{k=1}^{n} G_{ki} \cdot G_{kj}$$

Calcular la Rotación de Jacobi la cual diagonaliza $\begin{bmatrix} a & c \\ c & b \end{bmatrix}$

$$\zeta = \frac{b - a}{2c}$$

$$t = \frac{\text{sign}(\zeta)}{|\zeta| + \sqrt{1 + \zeta^2}}$$

$$c = \frac{1}{\sqrt{1 + t^2}}$$

$$s = c \cdot t$$

Actualizar columnas i *y* j *de* G

Para $k = 1, \ldots, n$

$$tmp = G_{ki}$$

$$G_{ki} = c \cdot tmp - s \cdot G_{kj}$$

$$G_{kj} = s \cdot tmp + c \cdot G_{kj}$$

Fin Para

Actualizar la matriz V *de vectores singulares derechos*

Para $k = 1, \ldots, n$

$$tmp = V_{ki}$$

$$V_{ki} = c \cdot tmp - s \cdot V_{kj}$$

$$V_{kj} = s \cdot tmp + c \cdot V_{kj}$$

Fin Para

Fin Para

Hasta *la convergencia* $(|c|/\sqrt{a \cdot b} \leq tol)$

Para $k = 1, \ldots, n$

$$\sigma_k = \|G(:, k)\|_2$$

$U(:,k) = G(:,k)/\sigma_k$

Fin Para

$\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$.

CAPÍTULO 5

RESULTADOS NUMÉRICOS

En este capítulo se presentan diversos resultados numéricos correspondientes a la ejecución de los algoritmos planteados en el capítulo anterior para el cálculo de los valores singulares y de la SVD completa de algunas matrices de entradas reales. Mediante tablas y gráficas de barras comparativas se muestra el desempeño de cada algoritmo con respecto a cada matriz considerada, exponiendo el tiempo de ejecución de los mismos y el número de iteraciones realizadas; para aquellas matrices de prueba que no se tienen los valores singulares exactos se comparan los resultados obtenidos con los generados por la función svd.m de Matlab. Todos los algoritmos señalados en este trabajo han sido implementados en Matlab 7.8.0.

Cuadro 5.1: Información básica del computador

Sistema operativo	Microsoft Windows XP Professional
Tipo de sistema	Sistema operativo de 32 bits
Procesador	Pentium(R) 4 CPU 3,20GHz
Memoria RAM	1,024MB
Memoria RAM disponible	340,12MB
Capacidad del disco duro	80GB
Capacidad disponible del disco duro	455MB

En el cuadro 5.1 se describen las características básicas del computador que se utilizó para ejecutar los diferentes algoritmos estudiados en el capítulo 4.

Cuadro 5.2: Información básica de las matrices de prueba consideradas

Matriz	**Tamaño**	**Condición**	**Características**	**Procedencia**
NOS4	100×100	$1{,}5787\times10^{3}$	Sparse-No simét.	Colección Harwell-Boeing
RW136	136×136	$2{,}5221\times10^{5}$	Sparse-No simét.	Colección NEP
A_1	150×150	$4{,}9767\times10^{3}$	Densa-No simét.	Matriz random
$AR1$	165×165	$1{,}0760\times10^{2}$	Densa-No simét.	Creada para este trabajo
A_2	200×200	$1{,}0536\times10^{4}$	Densa-No simét.	Matriz random
$AR2$	219×219	$1{,}0029\times10^{4}$	Densa-No simét.	Creada para este trabajo
PDE225	225×225	39,0638	Sparse-No simét.	Colección NEP
$AR3$	250×240	1,2510	Densa-No simét.	Creada para este trabajo
A_3	300×180	94,3257	Densa-No simét.	Matriz random
A_4	400×284	$1{,}7833\times10^{2}$	Densa-No simét.	Matriz random
ILLC1033	1033×320	$1{,}8888\times10^{4}$	Sparse-No simét.	Colección Harwell-Boeing
WELL1033	1033×320	166,1333	Sparse-No simét.	Colección Harwell-Boeing

El cuadro 5.2 detalla algunas características de las doce matrices de prueba que fueron consideradas para los resultados numéricos, destacando sus dimensiones (se consideraron matrices desde 100×100 hasta 1033×320, donde siete son matrices cuadradas y cinco son matrices rectangulares), número de condición (desde 1,2510 hasta $2{,}5221\times10^{5}$), características estructurales (densas, sparse y no simétricas) y orígenes (tres de la colección Harwell-Boeing, dos de la colección NEP, cuatro son random obtenidas de Matlab y tres creadas específicamente para este trabajo). A continuación se detalla cómo se construyeron, en lenguaje Matlab, estas últimas tres matrices mencionadas:

```
>> x=1:0.65:1.0760e2;
% size(x)=165×1
>> S=sort(diag(x),'descend');
>> [U,R]=qr(magic(165));
>> [V,R]=qr(rand(165));
>> AR1=U*S*V';
```

```
>> x=1:46:1.0029e4;
% size(x)=219×1
>> S=sort(diag(x),'descend');
>> [U,R]=qr(magic(219));
>> [V,R]=qr(rand(219));
>> AR2=U*S*V';
```

```
>> x=1:1.0500e-3:1.2510;
% size(x)=240×1
>> S=sort(diag(x),'descend');
>> S=[S;zeros(10,240)];
% size(S)=250×240
>> [U,R]=qr(magic(250));
>> [V,R]=qr(rand(240));
>> AR3=U*S*V';
```

Estas tres implementaciones permitieron obtener, a priori, los valores singulares exactos (diag(S)) de $AR1$, $AR2$ y $AR3$ que sirvieron de datos para el análisis de la precisión de los valores singulares calculados por cinco de los algoritmos presentados.

Cuadro 5.3: Aplicaciones desde donde surgen algunas matrices tomadas de *http://math.nist.gov/MatrixMarket/*

Matriz	**Aplicación**
NOS4	Ecuaciones lineales en ingeniería estructural. Aproximación por elemento finito a una estructura de vigas
RW136	Teoría de la probabilidad y sus aplicaciones. Matriz de transición de la cadena de Markov
PDE225	Ecuación en derivadas parciales. Discretización en diferencias finitas centrada de 5 puntos.
ILLC1033	Problema de mínimos cuadrados en Topografía
WELL1033	Problema de mínimos cuadrados en Topografía

El cuadro 5.3 muestra las aplicaciones y/o disciplinas de donde surgieron las matrices de prueba seleccionadas de las dos colecciones NEP y Harwell-Boeing, las cuales pueden ser consultadas en la página web ***http://math.nist.gov/MatrixMarket/***.

Cuadro 5.4: Comparación del tiempo de ejecución y número de iteraciones entre tres métodos numéricos (Golub-Kahan-Reinsch y dos de sus variantes, los algoritmos de Chan y Demmel-Kahan)

Matriz		**Golub-Kahan-Reinsch**	**Chan**	**Demmel-Kahan**
NOS4	*tiempo* (*seg.*)	$2{,}17\times10^{2}$	$2{,}30\times10^{2}$	31,2031
100×100	*iter*	66.383	73.300	39.361
RW136	*tiempo* (*seg.*)	$2{,}16\times10^{2}$	$1{,}33\times10^{2}$	29,8438
136×136	*iter*	20.475	19.955	17.533
A_1	*tiempo* (*seg.*)	$3{,}44\times10^{2}$	$2{,}07\times10^{2}$	35,7813
150×150	*iter*	36.623	31.231	17.377
A_2	*tiempo* (*seg.*)	$7{,}83\times10^{2}$	$5{,}28\times10^{2}$	78,8281
200×200	*iter*	46.753	42.800	34.265
PDE225	*tiempo* (*seg.*)	$2{,}74\times10^{3}$	$1{,}24\times10^{3}$	$2{,}10\times10^{2}$
225×225	*iter*	170.344	124.132	56.814
A_3	*tiempo* (*seg.*)	$9{,}57\times10^{3}$	$1{,}56\times10^{2}$	57,2188
300×180	*iter*	121.748	13.363	7.945
A_4	*tiempo* (*seg.*)	$3{,}32\times10^{3}$	$1{,}04\times10^{3}$	$1{,}10\times10^{2}$
400×284	*iter*	166.254	42.176	24.327

En el cuadro 5.4 se observan los resultados obtenidos del tiempo de ejecución y del número de iteraciones requeridas de los algoritmos de Golub-Kahan-Reinsch (G-K-R) y dos de sus variantes, (Chan (Ch) y Demmel-Kahan (D-K)) para el cálculo de los valores singulares de siete matrices reales. Se establecieron dos criterios de parada: primero, el hecho de que todos los elementos en la diagonal superior de la matriz bidiagonal B obtenida en la fase I no excediera en módulo a la tolerancia exigida $tol = 100 * eps$, donde $eps = 2{,}2204\times10^{-16}$ y segundo, el hecho de que el número máximo de iteraciones no excediera al

parámetro $maxit = 500 \cdot n^2$, donde n es la cantidad de columnas de cada matriz.

Cuadro 5.5: Comparación de la norma $\|\Sigma - \Sigma_{Matlab}\|_2$ entre tres métodos numéricos (Golub-Kahan-Reinsch y dos de sus variantes, los algoritmos de Chan y Demmel-Kahan)

Matriz		Golub-Kahan-Reinsch	Chan	Demmel-Kahan
NOS4	$\|\Sigma - \Sigma_{Matlab}\|_2$	$1{,}2647\times10^{-9}$	$4{,}6557\times10^{-13}$	$4{,}5197\times10^{-13}$
RW136	$\|\Sigma - \Sigma_{Matlab}\|_2$	$3{,}8058\times10^{-13}$	$3{,}9735\times10^{-13}$	$3{,}7825\times10^{-13}$
A_1	$\|\Sigma - \Sigma_{Matlab}\|_2$	$2{,}7005\times10^{-10}$	$2{,}6947\times10^{-12}$	$2{,}6379\times10^{-12}$
A_2	$\|\Sigma - \Sigma_{Matlab}\|_2$	$5{,}6861\times10^{-12}$	$5{,}7074\times10^{-12}$	$5{,}1568\times10^{-12}$
PDE225	$\|\Sigma - \Sigma_{Matlab}\|_2$	$1{,}4232\times10^{-11}$	$8{,}4324\times10^{-12}$	$9{,}2673\times10^{-12}$
A_3	$\|\Sigma - \Sigma_{Matlab}\|_2$	$1{,}9185\times10^{-11}$	$2{,}2773\times10^{-12}$	$1{,}9309\times10^{-12}$
A_4	$\|\Sigma - \Sigma_{Matlab}\|_2$	$2{,}5150\times10^{-11}$	$1{,}0422\times10^{-11}$	$9{,}0292\times10^{-12}$

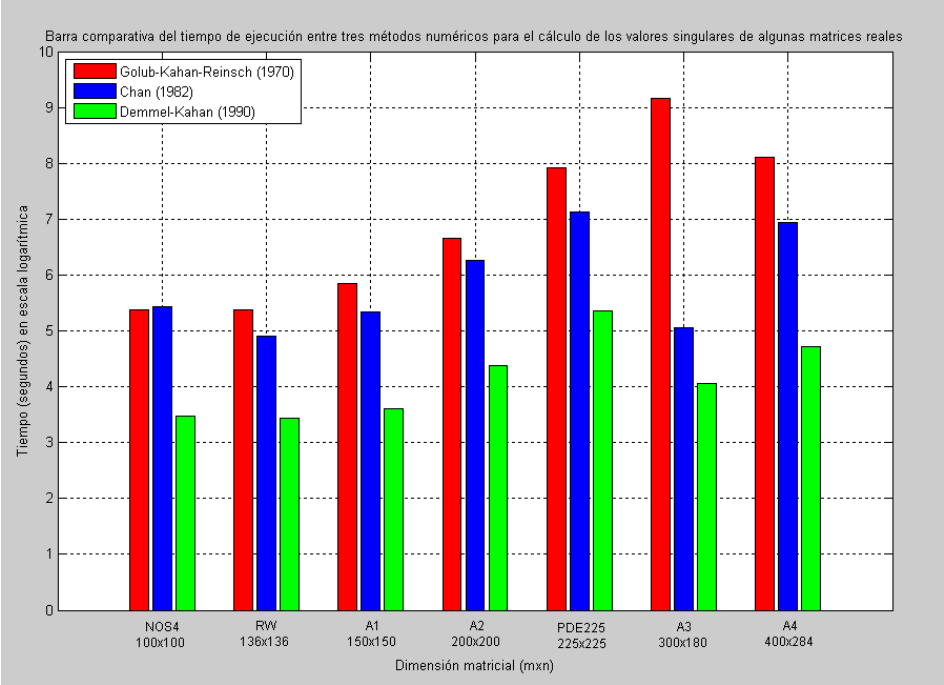


Figura 5.1: Gráfico de barras comparativas del tiempo de ejecución entre tres métodos numéricos (Golub-Kahan-Reinsch y dos de sus variantes) para el cálculo de los valores singulares de algunas matrices reales

La figura 5.1 muestra un gráfico donde se distingue el desempeño de estos tres métodos en cuanto al tiempo. Se observa que en todos los casos, los mejores tiempos de ejecución fueron obtenidos con el método D-K, representado por la barra de color verde. Para las dos matrices pequeñas, Ch y G-K-R (indicados por las barras de color azul y rojo, respectivamente) compiten mientras que para las matrices más grandes, Ch le gana en tiempo a G-K-R. Por ejemplo, este último logró 61,35 veces mayor tiempo que Ch, para la matriz A_3. Es importante acotar que, estos resultados muestran estrecha relación con lo expuesto en la sección 4.4 del capítulo anterior, ya que Ch y D-K son versiones de G-K-R que mejoran su eficiencia.

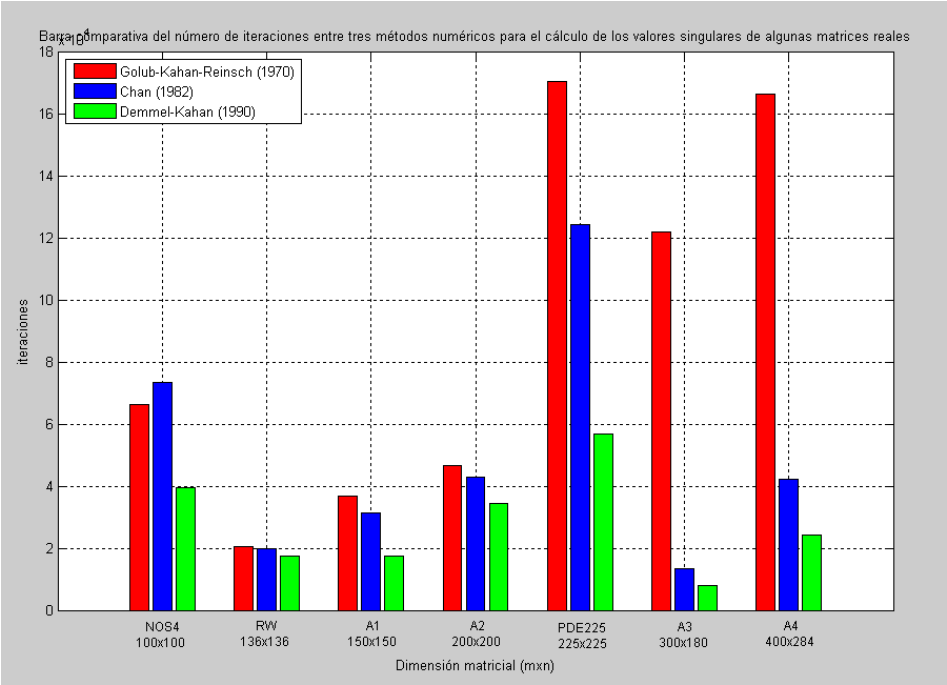


Figura 5.2: Gráfico de barras comparativas del número de iteraciones entre tres métodos numéricos (Golub-Kahan-Reinsch y dos de sus variantes) para el cálculo de los valores singulares de algunas matrices

La figura 5.2 es la representación gráfica para G-K-R, Ch, y D-K; en este caso, se consideró el número de iteraciones realizadas. Similarmente a la figura 5.1, se aprecia nuevamente cómo D-K requiere menos iteraciones que los otros dos métodos para satisfacer la tolerancia exigida para las siete matrices

consideradas, seguido por Ch y luego G-K-R.

Es importante resaltar que para estas siete matrices de prueba no se tienen sus valores singulares exactos y por tanto, no se puede indicar con qué precisión se alcanzaron los resultados. Sin embargo, para verificar el buen funcionamiento de nuestros códigos, se compararon los resultados con aquellos que genera el código svd.m de Matlab (usando las expresiones Σ y Σ_{Matlab} para nuestros resultados y los de Matlab, respectivamente). Se observa en el cuadro 5.5 que, en general, los resultados obtenidos en este trabajo son bastante similares a los obtenidos con el código de Matlab. Sin embargo, cabe resaltar que los resultados más parecidos en todos los casos, salvo en una (PDE225), fueron los generados por D-K. Más adelante, en el cuadro 5.8 se expondrán los errores absolutos correspondientes a tres matrices de prueba en las que, a priori, sí se conocen sus valores singulares exactos.

Cuadro 5.6: Comparación del tiempo de ejecución y $\|\Sigma - \Sigma_{Matlab}\|_2$ entre dos métodos numéricos basados en el Método de Bisección

Matriz		**Bisección con B^tB**	**Bisección con $P^t \begin{pmatrix} 0 & B^t \\ B & 0 \end{pmatrix} P$**
NOS4	*tiempo (seg.)*	7,91	58,45
100×100	$\|\Sigma - \Sigma_{Matlab}\|_2$	$2{,}8866\times10^{-15}$	$2{,}8866\times10^{-15}$
RW136	*tiempo (seg.)*	15,53	$1{,}18\times10^{2}$
136×136	$\|\Sigma - \Sigma_{Matlab}\|_2$	$4{,}2265\times10^{-15}$	$5{,}1070\times10^{-15}$
A_1	*tiempo (seg.)*	23,22	$1{,}47\times10^{2}$
150×150	$\|\Sigma - \Sigma_{Matlab}\|_2$	$4{,}2633\times10^{-14}$	$5{,}6843\times10^{-14}$
A_2	*tiempo (seg.)*	39,53	$2{,}92\times10^{2}$
200×200	$\|\Sigma - \Sigma_{Matlab}\|_2$	$3{,}4639\times10^{-14}$	$3{,}4639\times10^{-14}$
PDE225	*tiempo (seg.)*	51,13	$2{,}14\times10^{2}$
225×225	$\|\Sigma - \Sigma_{Matlab}\|_2$	$6{,}9278\times10^{-14}$	$6{,}7502\times10^{-14}$
A_3	*tiempo (seg.)*	36,09	$1{,}17\times10^{2}$
300×180	$\|\Sigma - \Sigma_{Matlab}\|_2$	$6{,}5725\times10^{-14}$	$6{,}5725\times10^{-14}$
A_4	*tiempo (seg.)*	$1{,}57\times10^{2}$	$3{,}91\times10^{2}$
400×284	$\|\Sigma - \Sigma_{Matlab}\|_2$	$6{,}9278\times10^{-14}$	$7{,}1054\times10^{-14}$
ILLC1033	*tiempo (seg.)*	$1{,}28\times10^{3}$	$1{,}67\times10^{3}$
1033×320	$\|\Sigma - \Sigma_{Matlab}\|_2$	$1{,}3843\times10^{-7}$	$1{,}6900\times10^{-8}$
WELL1033	*tiempo (seg.)*	$1{,}59\times10^{3}$	$7{,}15\times10^{3}$
1033×320	$\|\Sigma - \Sigma_{Matlab}\|_2$	$5{,}8318\times10^{-9}$	$7{,}9965\times10^{-9}$

Por otra parte, el cuadro 5.6 expone los resultados obtenidos de dos algoritmos basados en el método de Bisección. El primero usa la matriz tridiagonal simétrica $T = B^tB$ y el segundo utiliza la matriz tridiagonal simétrica $T_{ps} = P^t \begin{pmatrix} 0 & B^t \\ B & 0 \end{pmatrix} P$, donde P es una matriz de permutación. En cuanto a los resultados de los valores singulares obtenidos por estos dos algoritmos comparados con los generados por el código svd.m, se puede notar que son bastante similares coincidiendo entre 9 y 15 dígitos para las dos versiones y para las nueve matrices de prueba consideradas, a pesar de lo mal condicionadas que están algunas de ellas (por ejemplo, RW136, A_2 e ILLC1033).

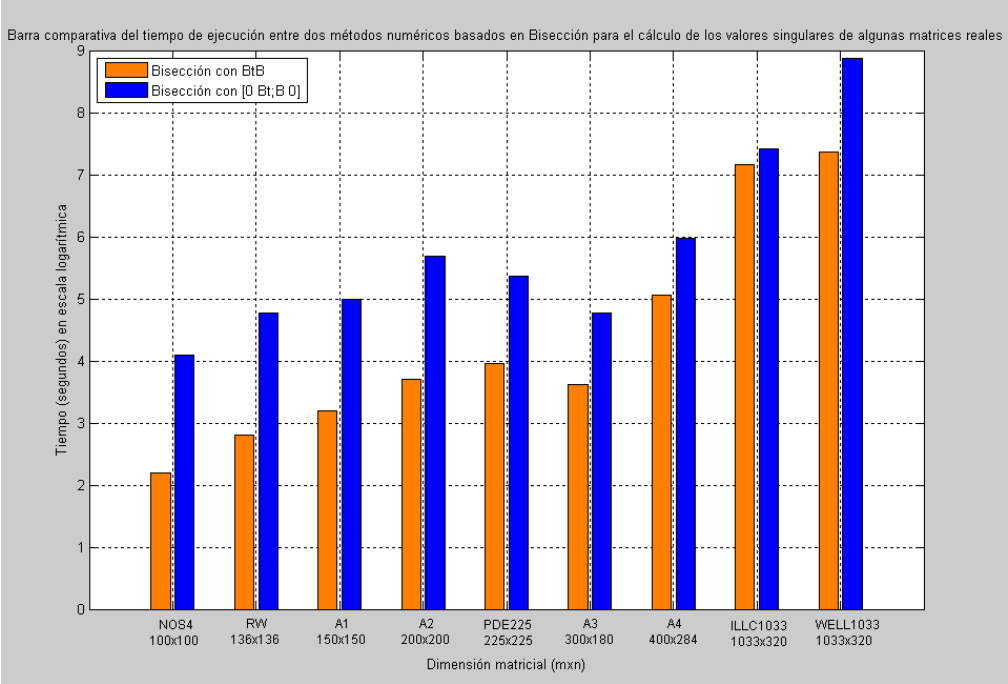


Figura 5.3: Gráfico de barras comparativas del tiempo en ejecución entre dos métodos numéricos basados en Bisección para el cálculo de los valores singulares de algunas matrices reales

En el cuadro 5.6 se muestra el tiempo de ejecución requerido por las dos versiones del método Bisección para las nueve matrices de prueba indicadas, así como las normas $\|\Sigma - \Sigma_{Matlab}\|_2$. Se observa en Figura 5.3 que el comportamiento, en cuanto a tiempo, del algoritmo basado en T_{ps} (barra de color azul) es superior al del algoritmo basado en $T = B^tB$ (barra de color naranja) para

nueve de las matrices de estudio. Por ejemplo, para RW136, Bis. T_{ps} resultó ser 7,60 veces más lento que Bis. B^tB mientras que para PDE225 fue 4,19 veces más lento también. Para WELL1033, Bis. B^tB fue 4,50 veces más rápido que Bis. T_{ps} mientras que para A_1 fue 6,33 veces más rápido también. Este resultado es esperado puesto que la matriz T_{ps} es de dimensión $(m+n)\times(m+n)$, mientras que $T = B^tB$ es de dimensión $n\times n$. Para WELL1033 es notorio la diferencia, ya que, además, en este caso, $m \gg n$. Por lo tanto, fue apropiado elegir al método adaptado a $T = B^tB$ como el método de Bisección para competir con otras de las propuestas presentadas en este trabajo para el cálculo de los valores singulares y de la SVD completa. Como criterio de parada se estableció el hecho de que las instrucciones se realizaran mientras que los subintervalos $[a, b]$ generados por las bisecciones tuviesen longitudes mayores a la tolerancia exigida ($tol = 1\times10^{-8}$).

Cuadro 5.7: Comparación del tiempo de ejecución y $\|\Sigma - \Sigma_{Matlab}\|_2$ entre cinco métodos numéricos

Matriz		De-Ka	Div-Ven	Bis B^tB	DQDS	Jacobi
NOS4	*tiempo (seg.)*	31,20	2,52	7,91	1,28	2,69
100×100	$\|\Sigma - \Sigma_{Matlab}\|_2$	$4{,}5197\times10^{-13}$	$7{,}4625\times10^{-15}$	$2{,}8866\times10^{-15}$	$9{,}9849\times10^{-8}$	$1{,}2323\times10^{-14}$
RW136	*tiempo (seg.)*	29,84	6,16	15,53	5,20	5,64
136×136	$\|\Sigma - \Sigma_{Matlab}\|_2$	$3{,}7825\times10^{-13}$	$1{,}1349\times10^{-13}$	$4{,}2265\times10^{-15}$	$8{,}5597\times10^{-7}$	$2{,}6423\times10^{-14}$
A_1	*tiempo (seg.)*	35,78	4,41	23,22	4,88	6,89
150×150	$\|\Sigma - \Sigma_{Matlab}\|_2$	$2{,}6379\times10^{-12}$	$1{,}0869\times10^{-13}$	$4{,}2633\times10^{-14}$	$1{,}2406\times10^{-8}$	$7{,}8160\times10^{-13}$
A_2	*tiempo (seg.)*	78,83	14,78	39,53	18,53	16,66
200×200	$\|\Sigma - \Sigma_{Matlab}\|_2$	$5{,}1568\times10^{-12}$	$1{,}9666\times10^{-12}$	$3{,}4639\times10^{-14}$	$1{,}7070\times10^{-8}$	$1{,}1937\times10^{-12}$
PDE225	*tiempo (seg.)*	$2{,}10\times10^{2}$	21,39	51,13	86,30	18,33
225×225	$\|\Sigma - \Sigma_{Matlab}\|_2$	$6{,}9278\times10^{-14}$	$6{,}7502\times10^{-14}$	$6{,}9278\times10^{-14}$	$1{,}7926\times10^{-7}$	$3{,}1264\times10^{-13}$
A_3	*tiempo (seg.)*	57,22	16,09	36,09	42,09	– – – –
300×180	$\|\Sigma - \Sigma_{Matlab}\|_2$	$1{,}9309\times10^{-12}$	$8{,}1712\times10^{-14}$	$6{,}5725\times10^{-14}$	$3{,}8701\times10^{-9}$	– – – –
A_4	*tiempo (seg.)*	$2{,}09\times10^{2}$	$1{,}10\times10^{2}$	$1{,}57\times10^{2}$	$2{,}67\times10^{2}$	– – – –
400×284	$\|\Sigma - \Sigma_{Matlab}\|_2$	$9{,}0292\times10^{-12}$	$1{,}2434\times10^{-13}$	$6{,}9278\times10^{-14}$	$4{,}3149\times10^{-9}$	– – – –
ILLC1033	*tiempo (seg.)*	– – – –	$6{,}75\times10^{3}$	$1{,}28\times10^{3}$	$8{,}63\times10^{2}$	– – – –
1033×320	$\|\Sigma - \Sigma_{Matlab}\|_2$	– – – –	$2{,}8161\times10^{-13}$	$1{,}3843\times10^{-7}$	$2{,}0880\times10^{-5}$	– – – –
WELL1033	*tiempo (seg.)*	– – – –	$1{,}92\times10^{3}$	$1{,}59\times10^{3}$	$1{,}21\times10^{3}$	– – – –
1033×320	$\|\Sigma - \Sigma_{Matlab}\|_2$	– – – –	$1{,}3989\times10^{-14}$	$5{,}8318\times10^{-9}$	$2{,}6550\times10^{-7}$	– – – –

En Cuadro 5.7 se presenta los resultados en cuanto a tiempo de ejecución que requirieron los algoritmos Demmel-Kahan (De-Ka), Divide y Vencerás (Div-Ven), Bisección con B^tB (Bis. B^tB), DQDS y Jacobi para calcular los valores singulares de algunas matrices de prueba reales. Además, se incluye la

norma de la diferencia de los valores singulares obtenidos por los cinco métodos numéricos y los valores singulares que se extraen del código para la SVD de Matlab. La notación "– – ––", en De-Ka, indica que para ILLC1033 y WELL1033, el algoritmo superó al máximo de iteraciones fijado, alcanzando no converger, y en Jacobi, indica que no fue posible obtener algunos resultados para matrices rectangulares, ya que este algoritmo trabajó únicamente con matrices cuadradas. El criterio de parada empleado en el algoritmo ya mencionado fue que las ejecuciones se realizaran hasta que $|c|/\sqrt{a \cdot b} \leq tol$, donde $tol = 1 \times 10^{-8}$ (véase Algoritmo 4.12). Cabe resaltar que los resultados son bastante similares a los obtenidos con Matlab, en general coinciden entre 11 y 14 decimales, salvo los correspondientes al algoritmo DQDS donde se tienen sólo entre 4 y 7 decimales iguales. Para este último, se consideraron los dos criterios de parada de G-K-R y sus dos variantes, pero en este caso, se fijaron los parámetros $tol = 1 \times 10^{-8}$ y $maxit = 400$.

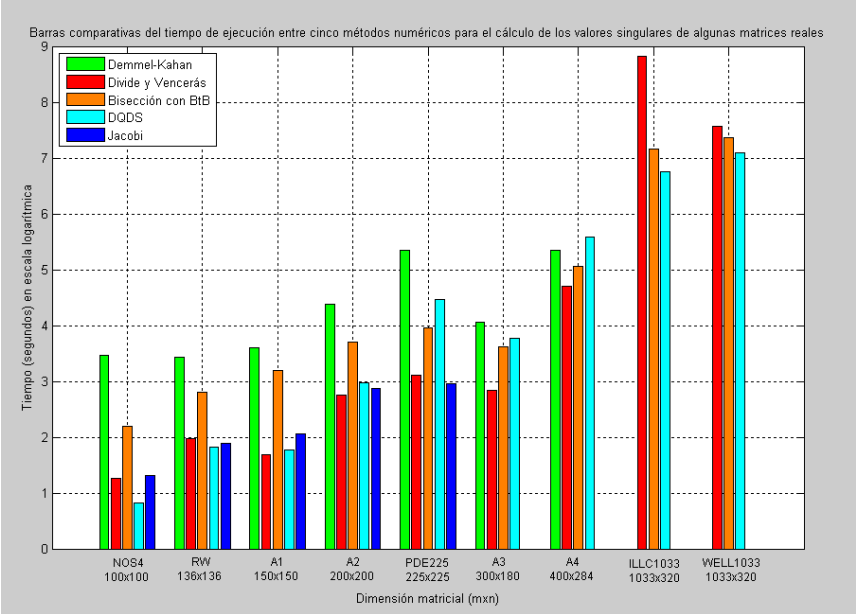


Figura 5.4: Gráfico de barras comparativas del tiempo en ejecución entre cinco métodos numéricos para el cálculo de los valores singulares de algunas matrices reales

En cuanto a la figura 5.4, se percibe el comportamiento asociado a la du-

ración de cada uno de los cinco algoritmos. Div-Ven (rojo), Bis. B^tB (naranja), DQDS (celeste), y Jacobi (azul), tuvieron mayor eficiencia en tiempo que D-K (verde), para las matrices NOS4, RW136, A_1, A_2, PDE225 y A_3, a pesar de que D-K fue el más efectivo en relación a G-K-R y Ch. Sin embargo, para A_4, D-K fue 1,28 veces más rápido que DQDS. Div-Ven volvió a ser el más eficaz para esta última matriz mencionada. ILLC1003 y WELL1033 resultaron ser las matrices que permitieron una mayor duración a tres de los métodos (Div-Ven, Bis. B^tB y DQDS), donde DQDS alcanzó la tolerancia exigida en menos tiempo. En este caso, el más lento fue Div-Ven.

Cuadro 5.8: Comparación del tiempo en ejecución y precisión en la SVD completa entre tres métodos numéricos

Matriz		**Div-Ven**	**Bis** B^tB	**Jacobi**
NOS4	*tiempo* (*seg.*)	2,52	7,91	2,69
100×100	$\|A - U\Sigma V^t\|_2$	$9{,}7481\times10^{-14}$	$2{,}9283\times10^{-15}$	$1{,}9818\times10^{-14}$
RW136	*tiempo* (*seg.*)	6,16	15,53	5,64
136×136	$\|A - U\Sigma V^t\|_2$	$1{,}3885\times10^{-13}$	$4{,}2265\times10^{-15}$	$5{,}2182\times10^{-14}$
A_1	*tiempo* (*seg.*)	4,41	23,22	6,89
150×150	$\|A - U\Sigma V^t\|_2$	$5{,}7673\times10^{-13}$	$1{,}9494\times10^{-13}$	$1{,}6552\times10^{-12}$
A_2	*tiempo* (*seg.*)	14,78	39,53	16,66
200×200	$\|A - U\Sigma V^t\|_2$	$5{,}1568\times10^{-12}$	$1{,}9666\times10^{-12}$	$3{,}4639\times10^{-14}$
PDE225	*tiempo* (*seg.*)	21,40	51,13	18,33
225×225	$\|A - U\Sigma V^t\|_2$	$3{,}8178\times10^{-12}$	$4{,}9460\times10^{-14}$	$5{,}9310\times10^{-13}$
A_3	*tiempo* (*seg.*)	16,09	36,09	– – ––
300×180	$\|A - U\Sigma V^t\|_2$	$1{,}3431\times10^{-12}$	$5{,}3641\times10^{-13}$	– – ––
A_4	*tiempo* (*seg.*)	$1{,}10\times10^{2}$	$1{,}57\times10^{2}$	– – ––
400×284	$\|A - U\Sigma V^t\|_2$	$2{,}8507\times10^{-12}$	$1{,}2029\times10^{-12}$	– – ––
ILLC1033	*tiempo* (*seg.*)	$6{,}75\times10^{3}$	$1{,}28\times10^{3}$	– – ––
1033×320	$\|A - U\Sigma V^t\|_2$	5,2141	1,6645	– – ––
WELL1033	*tiempo* (*seg.*)	$1{,}92\times10^{3}$	$1{,}59\times10^{3}$	– – ––
1033×320	$\|A - U\Sigma V^t\|_2$	3,3756	2,0771	– – ––

Ahora bien, el cuadro 5.8 señala los tiempos de Div-Ven, Bis. B^tB, y Jacobi

correspondientes al cálculo de la SVD completa de nueve matrices reales. En este caso, se muestra la norma $\|A - U\Sigma V^t\|_2$ para analizar qué tan precisos son los resultados obtenidos. Para las matrices pequeñas, se puede observar que los resultados tienen una alta precisión, entre 11 y 14 decimales exactos para los tres algoritmos. Es importante resaltar que para las matrices de tamaño mayor a 600, los vectores columnas de las matrices U y V pierden la ortogonalidad y los resultados suelen no ser confiables (en particular, sucedió con ILLC1033 y WELL1033). Sin embargo, la matriz diagonal de valores singulares Σ para esos problemas son similares a los obtenidos por Matlab como se pudo notar en los cuadros 5.6 y 5.7.

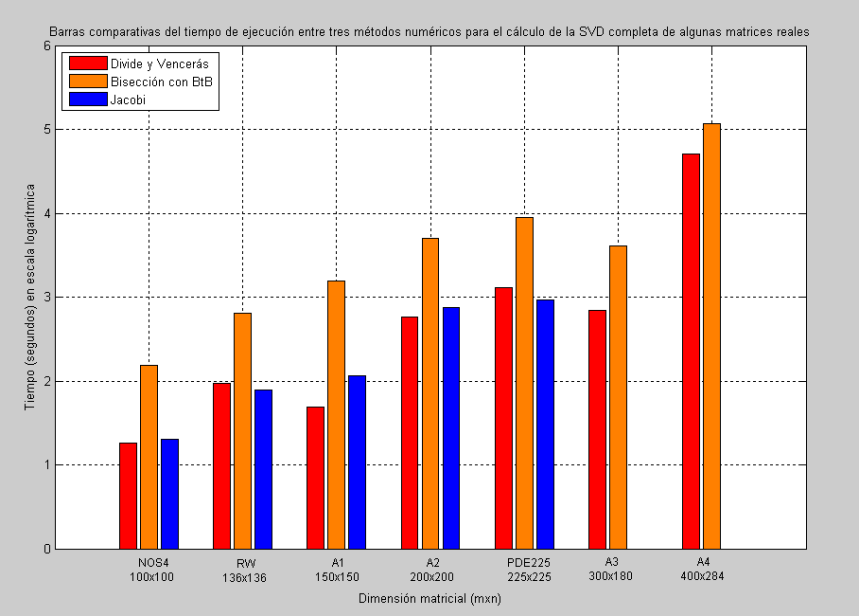


Figura 5.5: Gráfico de barras comparativas del tiempo en ejecución entre tres métodos numéricos para el cálculo de la SVD completa de algunas matrices reales

De la figura 5.5 se evidencia que, Div-Ven (barra de color rojo) fue el que obtuvo, en general, el mejor resultado en cuanto a tiempo. En todos los casos, Bis. B^tB (barra de color naranja) obtuvo el peor tiempo. En efecto, para NOS4, Div-Ven tuvo 3,14 veces mayor aceleración que Bis. B^tB mientras que, para A_1, fue de 5,27 veces más. Así mismo, para A_2, se consiguió que Bis. B^tB tuvo

2,68 veces más lentitud que Div-Ven. Jacobi (barra de color azul), aunque el tiempo que requirió fue bastante cercano al de Div-Ven no funcionó para todas las matrices de prueba.

Cuadro 5.9: Comparación del tiempo de ejecución y error absoluto en los valores singulares entre cinco métodos numéricos

Matriz		De-Ka	Div-Ven	Bis B^tB	DQDS	Jacobi
$AR1$	$tiempo\ (seg.)$	20,38	6,56	23,02	10,66	9,55
	$\|\Sigma - \Sigma_{Exacto}\|_2$	$9{,}1660\times10^{-12}$	$1{,}7160\times10^{-12}$	$1{,}1369\times10^{-13}$	$6{,}6433\times10^{-10}$	$2{,}2027\times10^{-12}$
$AR2$	$tiempo\ (seg.)$	46,80	17,64	46,91	32,53	22,91
	$\|\Sigma - \Sigma_{Exacto}\|_2$	$2{,}5402\times10^{-13}$	$5{,}0770\times10^{-13}$	$5{,}2636\times10^{-14}$	$3{,}2512\times10^{-8}$	$5{,}7732\times10^{-14}$
$AR3$	$tiempo\ (seg.)$	$1{,}93\times10^{2}$	55,17	$1{,}11\times10^{2}$	63,31	$-\ -\ -\ -$
	$\|\Sigma - \Sigma_{Exacto}\|_2$	$2{,}9701\times10^{-12}$	$2{,}5757\times10^{-14}$	$3{,}5527\times10^{-15}$	$3{,}6797\times10^{-9}$	$-\ -\ -\ -$

El cuadro 5.9 presenta, el tiempo de ejecución que necesitó cada algoritmo y la norma de la diferencia entre la matriz de valores singulares aproximados y la matriz de valores singulares exactos ($\|\Sigma - \Sigma_{Exacto}\|_2$) para tres matrices de prueba ($AR1$, $AR2$ y $AR3$). Estas matrices fueron construidas de tal manera de que se conociera exactamente sus valores singulares. Se observa que el algoritmo que obtuvo mejor precisión fue Bis. B^tB, con cifras entre $O(10^{-15})$ y $O(10^{-8})$ en todos los casos, seguido de Div-Ven.

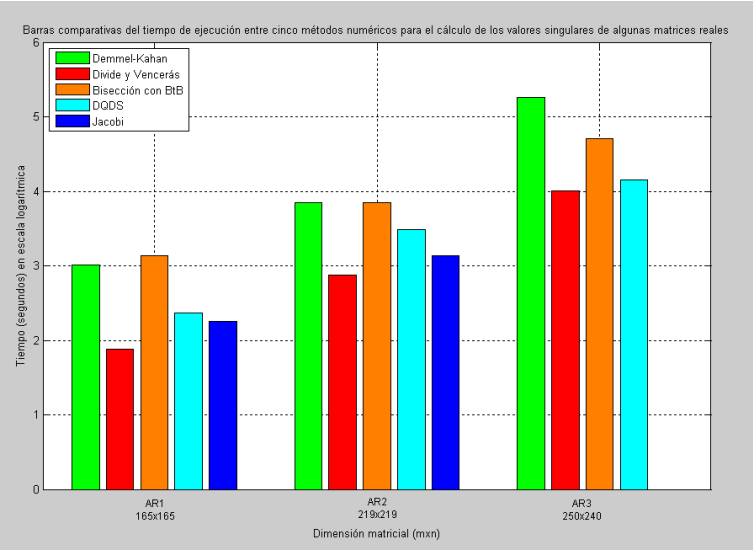


Figura 5.6: Gráfico de barras comparativas del tiempo en ejecución entre cinco métodos numéricos para el cálculo de los valores singulares de algunas matrices reales

La figura 5.6 exhibe el tiempo de ejecución de los cinco algoritmos ya mencionados para el cálculo de los valores singulares de $AR1$, $AR2$ y $AR3$. En ella se percibe la rapidez de convergencia de Div-Ven sobre el resto de los algoritmos para las matrices en cuestión, seguido de Jacobi y DQDS. A pesar del alto número de condición de $AR2$, con $1{,}0029\times10^4$, las barras correspondientes a los cinco métodos no sobrepasaron los 47 segundos. Los que resultaron ser más lentos, fueron Bis. B^tB y D-K.

5.0.1 SVD y la Compresión de Imágenes Digitales

A continuación se presenta una gran utilidad de la SVD en la técnica de la compresión de imágenes digitales. El campo de dicha aplicación es ampliamente conocido e importante. Por ello, se cuenta con múltiples herramientas para llevar a cabo dicha compresión y que, además, son fáciles de reconocer ya que vienen asociadas a la extensión de las imágenes, entre la que se destaca, la compresión JPEG. Sin embargo, la SVD juega un rol muy importante en esta técnica. Véase [8, 27].

Una imagen digital se compone de píxeles que se pueden imaginar como puntos de una pantalla. En el caso general, una imagen es de resolución $m\times n$ si se compone de m píxeles en la dirección vertical y n en la horizontal, por lo que, es posible asociarlo a una matriz de entrada real de $m\times n$. Por consiguiente, la compresión de imágenes está definida como el proceso de eliminar datos redundantes que se puedan encontrar en una imagen digital registradas en su matriz asociada. A continuación, se muestra, mediante una imagen digital dada, un experimento realizado en Matlab en el que se resaltó la gran ventaja que posee la SVD en este contexto:

Se consideró una imagen cuyo fichero fue llamado `niño.jpg`. Mediante las instrucciones

```
>> A=imread('niño.jpg');
>> A=rgb2gray(A);
>> A=double(A);
>> k=rank(A);
```

se construyó la matriz A asociada a la imagen en escala de grises, cuyas dimensiones fueron de 442×442, de rango $r = 442$. La imagen puede verse en la figura 6.1.

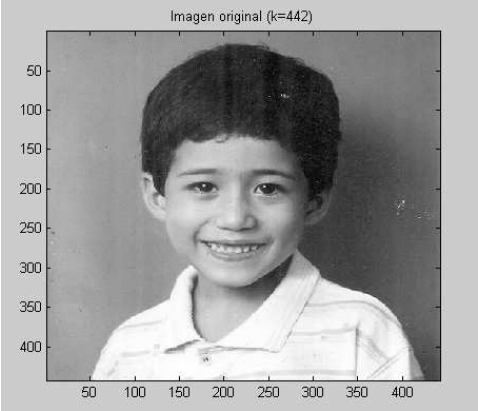


Figura 5.7: Imagen original, correspondiente a la matriz A con $k = r = 442$

La norma $\|A\|_2$ calculada mediante la instrucción

```
>> norm(A,2)
```

es el valor singular más grande (véase Proposición 2.6):

$$\|A\|_2 = \sigma_1 \approx 6{,}5908\times 10^4. \tag{5.1}$$

Para obtener la SVD de A se hizo uso del algoritmo SVD Jacobi (Algoritmo 4.12) cuya instrución fue:

```
>> [U,S,V] = svd_jacobi(A).
```

De la Proposición 2.17 se recuerda el siguiente hecho para una matriz A de $m \times n$, con rango $r = \text{mín}(m, n)$:

$$\min_{\text{rang}(\tilde{A})=k} \|A - \tilde{A}\|_2 = \|A - A_k\|_2 = \sigma_{k+1}, \tag{5.2}$$

donde $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^t$ es la matriz de rango $k < r$ que mejor aproxima a A en norma espectral.

Por consiguiente, lo último fue útil para obtener las aproximaciones necesarias que permitieron un buen esquema ilustrativo de la aplicabilidad de la SVD en la compresión de imágenes. Para ello, se tomaron diversos valores para k, los cuales se presentan más adelante.

Para construir las matrices $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^t$ de manera sencilla, se usó la siguiente asignación en Matlab:

```
>> Ak=U(:,1:k)*S(1:k,1:k)*V(:,1:k)',
```

la cual también se define como una SVD truncada de A. Luego de esto se utilizó

```
>> image(Ak)
```

para poder obtener la imagen aproximada correspondiente.

Para cada caso se calculó el factor de compresión (fc) que se consigue y también el error relativo (e) que se comete, por medio de las siguientes fórmulas:

$$\begin{aligned} fc &= \frac{k \cdot m + k \cdot n + k}{m \cdot n} = \frac{k(m+n+1)}{m \cdot n} = \frac{k(442+442+1)}{442 \cdot 442} = \frac{885k}{195.364} \\ e &= \frac{\|A - A_k\|_2}{\|A\|_2} = \frac{\sigma_{k+1}}{\sigma_1} \qquad \text{por (5.1) y (5.2).} \end{aligned}$$

El valor fc representa el cociente entre el espacio requerido para almacenar la imagen aproximada y el necesario para almacenar la matriz original. Si se multiplica fc por 100, se obtiene el valor porcentual de compresión de la imagen original sobre la matriz de aproximación A_k.

Ahora, se expone los resultados en el cuadro 6.1 para algunos valores de k y posteriormente, se presentan las imágenes correspondientes:

Cuadro 5.10: Resultados numéricos para $k = 10, 80, 180$

k	**Error relativo** $e = \frac{\sigma_{k+1}}{6{,}5908 \times 10^4}$	**Factor de compresión** $fc = \frac{885k}{195.364}$
10	0,0242	0,0453
80	0,0032	0,3624
180	0,0013	0,8154

- **Para** $k = 10$

En este primer caso, el factor de compresión fue de $fc = 0{,}0453$, es decir, la imagen aproximada ocupó 4,53 % de la imagen original. El error relativo σ_{11}/σ_1 tuvo el valor de 0,0242, y merece la pena destacar que para 10 valores singulares de los 442 ya puede apreciarse la imagen del perfil de una persona. Véase la figura 6.2.

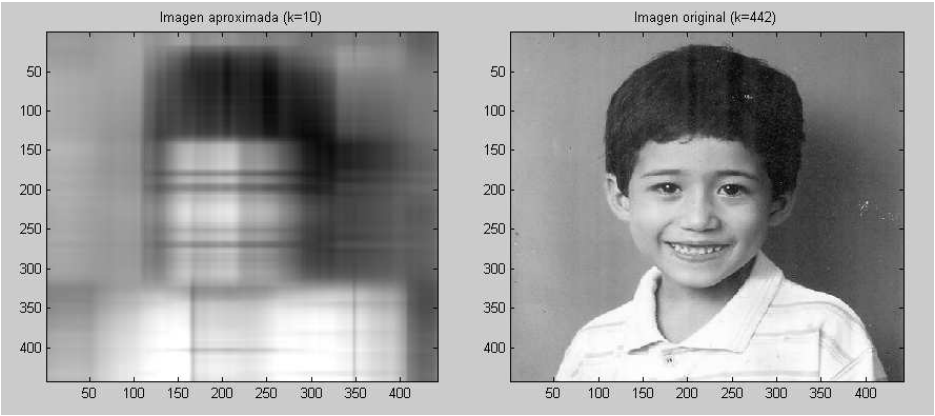


Figura 5.8: Comparación entre imagen aprox.$(k = 10)$ **e imagen original**

- **Para** $k = 80$

Ahora, el factor de compresión dió 0,3624, el cual indica que 36,24 % de la imagen original es representada por la imagen aproximada asociada a la matriz A_{80}. Por otro lado, el error relativo estuvo dado por $e = \sigma_{81}/\sigma_1 = 0{,}0032$. Esta última cifra es más pequeña que el dado para el caso $k = 10$, lo cual muestra una mayor aproximación a la matriz original. En la figura 6.3 puede apreciarse la similitud entre las imágenes.

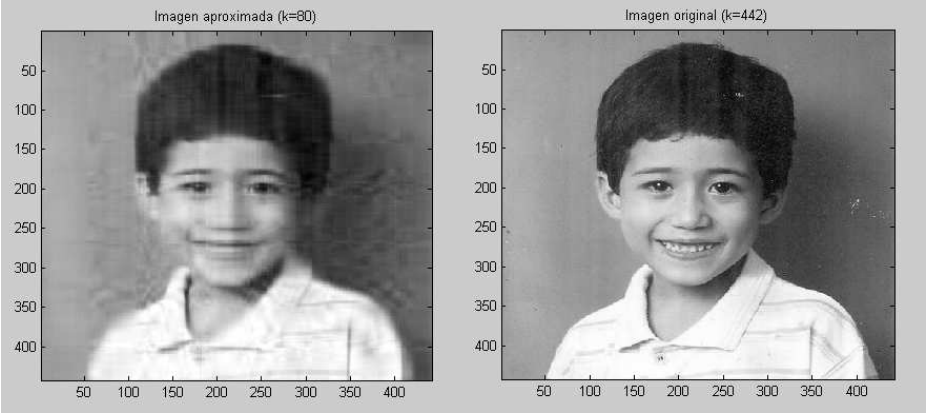


Figura 5.9: Comparación entre imagen aprox.$(k = 80)$ **e imagen original**

- **Para** $k = 180$

En este último caso (véase Figura 6.4) ya es posible distinguir perfectamente al niño fotografiado. El error relativo fue aún más pequeño que para $k = 80$ ($e = 0{,}0013$) y el factor de compresión, $fc = 0{,}8154$, permitió concluir que 81,54 % de la información contenida en la imagen original fue comprimida en la matriz A_{180}.

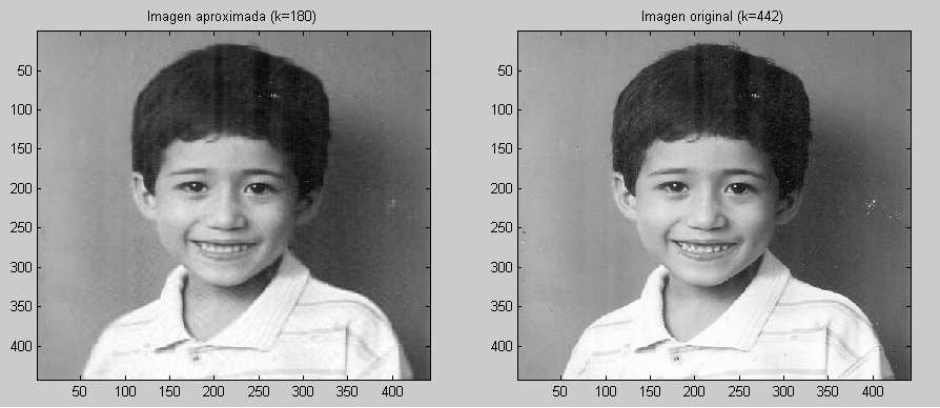


Figura 5.10: Comparación entre imagen aprox.$(k = 180)$ **e imagen original**

Por lo tanto, este experimento comprueba, efectivamente, la aplicabilidad que posee la SVD sobre la compresión de imágenes. Se pudo observar que un 40,72 % del total de los valores singulares hizo posible obtener una imagen bien parecida a la imagen original correspondiente.

CONCLUSIONES

En esta investigación se ha presentado detalladamente la composición teórica de siete métodos para el cálculo de la SVD de matrices a entradas reales (SVD basado en: Golub-Kahan-Reinsch, Chan, Demmel-Kahan, Divide y Vencerás, Bisección e Iteración Inversa, DQDS y Jacobi), para muchos de los cuales se mostraron sus respectivos algoritmos en lenguaje pseudoformal. El capítulo 5 fue dedicado a mostrar los resultados numéricos correspondientes a dichos algoritmos implementados en MATLAB 7.8.0, cuyos parámetros de estudio fueron: *tiempo* (*seg.*), *iteraciones*, $\|\Sigma - \Sigma_{Matlab}\|_2$, $\|A - U\Sigma V^t\|_2$ y $\|\Sigma - \Sigma_{Exacto}\|_2$. Por lo tanto, después de haber ejecutado los diversos algoritmos y analizado los respectivos resultados que sustentan esta investigación, a continuación se exponen las siguientes conclusiones para las diversas matrices de prueba consideradas:

- Los valores singulares calculados por los siete algoritmos presentaron alta similitud en relación a los obtenidos por svd.m de Matlab.
- El algoritmo SVD Demmel-Kahan calculó los valores singulares bastante similares a los de svd.m, y fue más eficiente en tiempo de ejecución que sus antecesores SVD Golub-Kahan-Reinsch y SVD Chan, para cuatro matrices densas y tres del tipo sparse.

- SVD Bisección con B^tB fue semejante a SVD Bisección con $T_{ps} = P^t \begin{pmatrix} 0 & B^t \\ B & 0 \end{pmatrix} P$ en cuanto a los valores singulares, para matrices de órdenes entre 100×100 y 1033×320. En relación al tiempo, el primero dominó sobre el segundo porque T_{ps} es de $(m+n) \times (m+n)$ y B^tB es de $n \times n$.

- Entre SVD: Demmel-Kahan, Divide y Vencerás, Bisección con B^tB y DQDS; DQDS fue el más eficaz en tiempo de ejecución para la mayoría de las matrices de prueba rectangulares mientras que Jacobi presentó gran rapidez para algunas matrices cuadradas que oscilaron entre los órdenes 100×100 y 225×225.

- En cuanto a la alta precisión de la SVD completa ($\|A - U\Sigma V^t\|_2$), SVD Divide y Vencerás superó a SVD Bisección con B^tB para algunas matrices rectangulares, mientras que para las matrices cuadradas superó también a SVD Jacobi.

- SVD: Demmel-Kahan, Divide y Vencerás, Bisección con B^tB y DQDS presentaron una excelente precisión en los valores singulares de $AR1$, $AR2$ y $AR3$, esto quedó comprobado cuando se analizó la norma $\|\Sigma - \Sigma_{Exacto}\|_2$, donde Σ_{Exacto} representó los valores singulares exactos calculados previamente a las ejecuciones. SVD Divide y Vencerás resultó ser el más rápido de los cinco algoritmos para las tres matrices mencionadas.

- Finalmente se comprobó la gran utilidad de la SVD en la compresión digital de imágenes. Se mostró, por medio de una imagen digital en particular, cómo dicha descomposición logró ahorrar aproximadamente un 19 % en espacio computacional con tan sólo considerar un estimado del 40 % de los valores singulares totales correspondientes a la matriz original.

BIBLIOGRAFÍA

[1] Aishima, K., Matsuo T., Murota K., Sugihara M., *A survey on convergence theorems of the dqds algorithm for computing singular values*, Journal of Math-for-industry, Vol.2, pp.1-11, November 2009.

[2] Aishima, K., Matsuo T., Murota K., Sugihara M., *On convergence of the dqds Algorithm for Singular Value Computation*, Mathematical Engineering Technical Reports, November 2006.

[3] Björck, Å., *Numerical Methods for Least Squares Problems*, SIAM. Philadelphia, 1996.

[4] Blomgren, P., *Numerical Matrix Analysis*, San Diego State University, Departament of Mathematics and Statistics. Spring 2008.

[5] Boley, D., *On bilinear functions*, Technical Report, University of Minnesota, Department of Computer Science, 37-90, 1990.

[6] Castellet M., Llerena I., *Álgebra lineal y geometría*, Universidad Autónoma de Barcelona, Editorial Reverté, S.A., 2000.

[7] Datta B.N., *Numerical Linear Algebra and Applications*, SIAM. Philadelphia, 2010.

[8] Demmel J.W., *Applied Numerical Linear Algebra*. SIAM. Philadelphia, 1997.

[9] Demmel J., Gragg W., *On computing accurate singular values and eigenvalues of acyclic matrices*, Linear Algebra Appl., 185:203-208, 1993.

[10] Demmel J., Veselić K., *Jacobi's method is more accurate than QR*. SIAM J. Mat. Anal. Appl. 13 (4), 1204-1246, 1992.

[11] García C., Valle S., *Detección de fallas en un proceso continuo utilizando descomposición de valores singulares*, Revista mexicana de Ingeniería Química, Vol. 5, Supl. 1, 177-181. 2006.

[12] Grégoire A., Sidi M.K., *Numerical Linear Algebra*, Springer. 2008.

[13] Golub G.H., Van Loan Ch.F., *Matrix computations*, SIAM. Philadelphia, 1996.

[14] Jordan, C., *Mémoire sur les formes bilinéaires*, Journal de Mathématiques Pures et Appliquées 19, 35-54, 1874.

[15] Jordan, C., *Sur la réduction des formes bilinéaires*, Comptes Rendues de l' Académie des Sciences 78, 614-617, 1874.

[16] Kahan W., *Accurate eigenvalues of a symmetric tridiagonal matrix*, Computer Science Dept. Technical Report CS41, Stanford University, Stanford, CA, July 1966.

[17] Luna Y., Morata A., Martín M., Valero F., *Influencia de los patrones de teleconexión del Atlántico Norte en la precipitación primaveral del Mediterráneo occidental*, Universidad Complutense de Madrid, Dpto. Astrofísica y CC. de la Atmósfera, Física de la tierra, Vol. 16. 2004.

[18] Paige L., Dean J., Slobko T., *Elementos de álgebra lineal*. Editorial Reverté, S. A., 1982.

[19] Parlett B., *The new qd algorithms*. Acta Numérica, 459-491, 1995.

[20] Parlett B., *The Symmetric Eigenvalue Problem*. Prentice Hall, Englewood Cliffs, N J, 1980.

[21] Regino M.C., *Álgebra Multilineal*, Editorial Universidad de Antioquia, 2006.

[22] Schmidt E., *Zur Theorie der linearen und nichtlinearen Integral gleichungen. I Teil. Entwicklung willürlichen Funktionen nach System vorgeschriebener*, Mathematische Annalen 63, 433-476, 1907.

[23] Stewart G.W., *On the early history of the singular value decomposition*, IMA Preprint Series 952, April 1992.

[24] Sundarapandian V., *Numerical Linear Algebra*, Prentice Hall, New Delhi, 2008.

[25] Sylvester J.J., *A new proof that a general quadric may be reduced to its canonical form (that is, a linear function of squares) by means of a real orthogonal substitution*, The Messenger of Mathematics 19, 1-5, 1889.

[26] Sylvester J.J., *Sur la réduction biorthogonale d' une forme linéo-linéaire à sa forme cannonique*, Comptes Rendues de l' Académie des Sciences 108, 651-653, 1889.

[27] Trefethen LL.N., Bau D., *Numerical Linear Algebra*. SIAM. Philadelphia, 1997.

[28] Weyl H., *Inequalities between the two kinds of eigenvalues of a linear transformation*, Proceedings of the National Academy of Sciences 35, 408-411, 1949.

[29] Wilkinson, J.H., *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, UK, 1965.

[30] Zaballa I., *Análisis Matricial Aplicado*, Departamento de Matemática Aplicada y Estadística e Investigación Operativa, Universidad del País Vasco. 2010.