Universidad de Valladolid

Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Matemáticas

La Descomposición en Valores Singulares: Implementación y Aplicaciones

Autora: Irene García Jiménez

Tutor: Ángel Durán Martín

Curso 2024-2025

Resumen

La descomposición en valores singulares (SVD) es una técnica de gran relevancia en álgebra lineal que descompone matrices rectangulares en factores ortogonales, permitiendo analizar sus propiedades fundamentales como el rango, la norma y los subespacios asociados.

En este trabajo de fin de grado se desarrollan los fundamentos teóricos de la SVD, incluyendo su interpretación geométrica y su relación con el teorema de Schur, así como extensiones como la SVD reducida y la descomposición en valores singulares generalizada (GSVD). También se profundiza en la implementación computacional de la SVD mediante métodos como Jacobi y el algoritmo de Demmel-Kahan, realizando un análisis comparativo de su eficiencia y estabilidad numérica.

Finalmente, se destacan aplicaciones prácticas en diversos campos, como el filtrado espectral, la detección de movimiento en vídeos, el reconocimiento facial y el análisis de redes neuronales convolucionales, con implementaciones en Python y MATLAB.

Palabras clave

SVD, Teorema de Schur, SVD reducida, GSVD, Pseudoinversa, Rango Bajo, Método de Jacobi, Algoritmo de Demmel-Kahan, Filtrado espectral, Procesamiento de imágenes, Reconocimiento facial, CNN, GoDec.

Abstract

The Singular Value Decomposition (SVD) is a highly relevant technique in linear algebra that decomposes rectangular matrices into orthogonal factors, enabling the analysis of fundamental properties such as rank, norm, and associated subspaces.

This bachelor's thesis delves into the theoretical foundations of the SVD, including its geometric interpretation and its relationship with Schur's theorem, as well as extensions like the reduced SVD and the generalized singular value decomposition (GSVD). It also explores the computational implementation of the SVD through methods such as Jacobi's and Demmel-Kahan's algorithm, performing a comparative analysis of their efficiency and numerical stability.

Finally, applications in various fields are highlighted, such as spectral filtering, motion detection, facial recognition, and the analysis of convolutional neural networks, with implementations in Python and MATLAB.

Keywords

SVD, Schur's Theorem, Reduced SVD, GSVD, Pseudoinverse, Low Rank, Jacobi Method, Demmel-Kahan Algorithm, Spectral Filtering, Image Processing, Facial Recognition, CNN, GoDec.

Índice general

Introducción

La descomposición en valores singulares, o SVD, es un método de factorización que permite generalizar a matrices rectangulares de cualquier dimensión, conceptos propios de matrices cuadradas relacionados con el espectro.

La SVD está basada en el llamado teorema de descomposición polar, que permite escribir toda matriz $A \in \mathbb{R}^{m\times n}$, $m \geq n$, como producto $A = UP$, con $U \in \mathbb{R}^{m\times n}$, cuyas columnas ortonormales representan geométricamente una rotación, y $P \in \mathbb{R}^{n\times n}$ simétrica y semidefinida positiva. A través del teorema espectral, es sencillo comprobar que la matriz P mencionada juega un papel de matriz de escalado en $\mathbb{R}^n$.

Mencionamos en esta introducción los principales trabajos que están en el origen de la SVD, a partir de [20]. Beltrami (1835-1899) es el precursor de la descomposición en valores singulares. En relación a la derivación teórica de su trabajo, comenzó definiendo una forma bilineal

$$f(x, y) = x^T Ay, \tag{1}$$

donde A es una matriz real de orden n. Si en (1) se realizan las sustituciones $x = U\xi$ y $y = V\eta$, entonces

$$f(x, y) = (U\xi)^T A(V\eta) = \xi^T S\eta, \qquad S = U^T AV.$$

Beltrami observó que, si U y V son ortogonales, hay $n^2 - n$ grados de libertad para anular los elementos fuera de la diagonal de S. Asumiendo que S es diagonal, es decir, que $S = \Sigma = \text{diag}(\sigma_1, ..., \sigma_n)$, y utilizando la ortogonalidad de V, se tiene

$$U^T A = \Sigma V^T. \tag{2}$$

De manera similar, se obtiene

$$AV = U\Sigma. \tag{3}$$

Sustituyendo el valor de U obtenido en (2) y (3) , se llega a

$$U^T(AA^T) = \Sigma^2 U^T, \tag{4}$$

$$(A^T A)V = V\Sigma^2. \tag{5}$$

Así, los σ_i son las raíces de las ecuaciones:

$$\det(AA^T - \sigma^2 I) = 0 \tag{6}$$

$$\det(A^T A - \sigma^2 I) = 0. \tag{7}$$

Beltrami concluye que las funciones (6) y (7) son idénticas, puesto que ambas tienen grado n, ambas son anuladas por los valores σ_i y además toman el mismo valor, $\det^2(A)$, al evaluarlas

en $\sigma = 0$. Este argumento requiere la suposición de que los valores singulares son distintos entre sí, y además no nulos, de manera que A es no singular. A continuación, Beltrami demuestra que los valores son reales y positivos, mediante la formula

$$0 < \|x^T A\|^2 = x^T(AA^T)x = \xi^T \Sigma^2 \xi,$$

lo que implica que σ_i^2 son positivos. Es importante señalar que Beltrami asume la existencia de un autovector de AA^T antes de demostrarlo, lo que causa confusión en su argumento.

En relación a la implementación, Beltrami proporciona un algoritmo para determinar la diagonalización:

- Calcular las raíces de (6).
- Hallar U a partir de (4). Beltrami señala que cada columna de U puede multiplicarse por 1 o -1 sin cambiar la validez de la descomposición, lo cual es válido solo si los σ_i son distintos. Además, asume implícitamente que U será ortogonal, lo que también requiere que los σ_i sean distintos.
- Determinar V a partir de (2). Este paso requiere que Σ sea no singular.

Se concluye pues, que Beltrami obtuvo la descomposición en valores singulares para una matriz cuadrada real no singular con valores singulares distintos. Su derivación es la que se encuentra en la mayoría de los libros de texto, pero carece de los elementos adicionales necesarios para manejar los casos que involucren matrices de rango bajo.

Por otra parte, Camille Jordan (1838-1921) es considerado como el codescubridor de la descomposición en valores singulares. Aunque publicó su trabajo un año después de Beltrami, ambos son independientes.

En relación a la derivación teórica de su trabajo, Jordan desarrolla la SVD de una matriz A buscando los máximos y mínimos de una forma cuadrática $P = x^T Ay$ bajo ciertas restricciones. Utiliza derivadas parciales y el método de deflación para simplificar el problema, permitiendo así encontrar los valores y vectores singulares de A. Comienza definiendo la forma

$$P = x^T Ay, \tag{8}$$

y busca el máximo y mínimo de (8) sujeto a $\|x\|_2 = \|y\|_2 = 1$. El máximo se determina mediante la ecuación

$$0 = dP = dx^T Ay + x^T Ady, \tag{9}$$

que debe satisfacerse para todo dx y dy que cumplan

$$dx^T x = 0 \quad \text{y} \quad dy^T y = 0. \tag{10}$$

ya que ello garantiza que dx y dy permanecen tangentes a la superficie de la esfera en los puntos x e y respectivamente. A partir de (9) y (10), Jordan deduce las expresiones

$$Ay = \sigma x \tag{11}$$

$$x^T A = \tau y^T. \tag{12}$$

De (11) se sigue que el máximo es $x^T(Ay) = \sigma x^T x = \sigma$. De manera similar, el máximo también es τ, por lo que se deduce la igualdad de ambos valores. Jordan observa que σ se determina por la anulación del determinante obtenido a partir de (11) y (12).

$$D = \begin{vmatrix} -\sigma I & A \\ A^T & -\sigma I \end{vmatrix}$$

También muestra que este determinante contiene solo potencias pares de σ. Sea σ_1 una raíz de la ecuación $D = 0$, y sean (11), (12) satisfechas por $x = u$ e $y = v$, donde $\|u\|_2 = \|v\|_2 = 1$. Se completan los vectores u y v con matrices U_* y V_* de manera que

$$\hat{U} = [u,\ U_*] \quad \text{y} \quad \hat{V} = [v,\ V_*]$$

sean ortogonales. Sean $x = \hat{U}\hat{x}$ y $y = \hat{V}\hat{y}$. Con estas sustituciones, se tiene

$$P = x^T A y = \hat{x}^T \hat{A} \hat{y}, \quad \text{con } \hat{A} = \hat{U} A \hat{V}.$$

P alcanza su máximo para $\hat{x} = \hat{y} = e_1$, donde $e_1 = (1, 0, \ldots, 0)^T$, lo que implica que σ_1 es la raíz mayor. Además, en el máximo se tiene

$$\hat{A}\hat{y} = \sigma_1 \hat{x} \quad \text{y} \quad \hat{x}^T \hat{A} = \sigma_1 \hat{y}^T,$$

luego $\hat{A}$ puede escribirse en la forma

$$\hat{A} = \begin{pmatrix} \sigma & 0 \\ 0 & A_1 \end{pmatrix},$$

para cierta matriz A_1. Así, tomando $\xi_1 = \hat{x}_1$ y $\eta_1 = \hat{y}_1$,

$$P = \sigma_1 \xi_1 \eta_1 + P_1,$$

donde P_1 es independiente de ξ_1 y η_1. Jordan aplica ahora la reducción de manera inductiva a P_1 para deducir la forma canónica

$$P = \xi^T \Sigma \eta.$$

Finalmente, observa que cuando las raíces de la ecuación característica $D = 0$ son simples, las columnas de U y V pueden calcularse directamente a partir de (8), (11) y (12).

En este trabajo, Jordan, al utilizar una solución parcial para reducir el problema a uno más manejable, método conocido como *deflación*, evita las complicaciones que surgen en el enfoque de Beltrami. La técnica de deflación estuvo aparentemente en desuso hasta que Schur la empleó en 1917 para establecer la forma triangular de matrices generales, convirtiéndose desde entonces en una herramienta teórica y algorítmica ampliamente utilizada. La matriz

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix},$$

de la cual se forma el determinante D, también goza de amplio uso, siendo su popularidad actual atribuida a Wielandt y Lanczos, quienes, probablemente, redescubrieron de manera independiente la descomposición en valores singulares. Otra consecuencia del enfoque de Jordan es la caracterización variacional del mayor valor singular como máximo de una función, aspecto crucial en teoremas de perturbación y localización de valores singulares.

Es importante destacar también el papel de James Joseph Sylvester (1814-1897), Erhard Schmidt (1876-1959) y Hermann Weyl (1885-1955) en el establecimiento y desarrollo de la teoría de la descomposición en valores singulares. Resumimos aquí sus principales contribuciones.

Sylvester, aunque no desarrolló directamente la SVD, contribuyó al marco teórico que la hizo posible, ya que fue pionero en el estudio de las propiedades de las matrices y los invariantes algebraicos. Introdujo el concepto de matriz invariante y exploró las propiedades de los valores característicos, un aspecto crucial para la comprensión de la SVD.

Por otro lado, Erhard Schmidt, matemático alemán, hizo una contribución significativa a la teoría de la SVD mediante su trabajo en ecuaciones integrales. En 1907, Schmidt desarrolló el proceso de ortogonalización que lleva su nombre, el proceso de Schmidt, el cual es esencial, por ejemplo, en la teoría de espacios de Hilbert. Su trabajo sobre los operadores compactos y las ecuaciones integrales proporcionó una formulación matemática que se aplica directamente a la SVD. Específicamente, Schmidt demostró que cualquier operador compacto en un espacio de Hilbert tiene una descomposición en valores singulares.

Por último, cabe destacar el trabajo de Hermann Weyl, otro matemático alemán, que amplió los trabajos de Schmidt en la teoría de operadores y espacios de Hilbert. En la década de 1920, Weyl realizó investigaciones fundamentales sobre los operadores compactos y la teoría espectral. Su trabajo en ecuaciones integrales y operadores autoadjuntos en espacios de Hilbert condujo a una comprensión más profunda de la descomposición en valores singulares. Weyl demostró que los operadores compactos pueden descomponerse en una serie de valores y vectores singulares, generalizando así los resultados de Schmidt.

En definitiva, Beltrami, Jordan y Sylvester sentaron las bases algebraicas, mientras que Schmidt y Weyl desarrollaron y ampliaron la teoría de la SVD desde la perspectiva de las ecuaciones integrales y los operadores compactos en espacios de Hilbert.

Este trabajo propone un estudio detallado de la SVD, a partir de la estructura siguiente:

- En el Capítulo 1 se incluyen los contenidos teóricos en los que se basa la descomposición en valores singulares. A partir del teorema fundamental, establecemos su relación con otras descomposiciones matriciales, describimos algunas extensiones de la SVD y mencionamos algunas consecuencias que la SVD ha tenido en el álgebra lineal, como en los problemas de mínimos cuadrados o en las aproximaciones de rango bajo de una matriz.
- El Capítulo 2 trata de la implementación de la SVD, haciendo referencia al Método de Jacobi, al Algoritmo de Demmel-Kahan, y desarrollando un estudio comparativo de ambos métodos.
- Por último, en el Capítulo 3 se presentarán las diferentes aplicaciones de la SVD en relación al tratamiento de imágenes, y, en concreto, se estudiarán los métodos de filtrado espectral, la detección de movimiento en vídeos y el reconocimiento facial. Se introducirán brevemente algunas de las aplicaciones de la SVD en redes neuronales.

Capítulo 1

La Descomposición en Valores Singulares

En este capítulo, se presenta la descomposición en valores singulares (SVD), abarcando desde sus fundamentos teóricos hasta algunas de sus aplicaciones teóricas y extensiones. Se inicia con el **teorema de la SVD**, donde se detallan la formulación matemática y propiedades fundamentales de la SVD, seguido de una discusión sobre la **interpretación Geométrica de la SVD**, que proporciona una perspectiva más intuitiva sobre su significado. Posteriormente, se examina la relación entre la SVD y el Teorema de Schur.

El capítulo prosigue con una revisión de algunas de las **extensiones de la SVD**, incluyendo la **SVD reducida** y la **descomposición en valores singulares generalizada (GSVD)**, explorando cómo estas variantes amplían la aplicabilidad de la SVD a un espectro aún más amplio de problemas matemáticos y computacionales.

Concluimos el capítulo presentando las aplicaciones teóricas de la SVD. Se aborda el uso de la SVD en contextos específicos como la **determinación de la norma euclídea** y el **número de condición** de una matriz, la solución de problemas lineales de **mínimos cuadrados** mediante la **pseudoinversa**, y la identificación de los **subespacios fundamentales de una matriz**, así como en la generación de **aproximaciones de rango bajo** de matrices.

1.1. Teorema de la SVD

Comenzamos mencionando un par de resultados básicos. Las demostraciones puede encontrarse en el libro de Ford ([8]).

Lema 1.1.1. Sea $A \in \mathbb{R}^{m\times n}$ una matriz simétrica. Entonces, sus autovalores son reales.

Lema 1.1.2. Sea $A \in \mathbb{R}^{m\times n}$. Los autovalores de una matriz A^TA de dimensión $n \times n$ son no negativos.

Definición 1.1.1. Sea $A \in \mathbb{R}^{m\times n}$, se definen los **valores singulares de A** como las raíces cuadradas de los autovalores de A^TA.

Siguiendo a Ford [8] se proporciona el enunciado y la demostración del teorema fundamental de la SVD, que factoriza una matriz $A \in \mathbb{R}^{m\times n}$ en el producto de dos matrices ortogonales y una matriz diagonal que contiene sus valores singulares.

Teorema 1.1.1. (de la SVD.) Sea $A \in \mathbb{R}^{m\times n}$, con $m \geq n$, $\sigma_1, \sigma_2, ..., \sigma_n$ los valores singulares de A, con r valores singulares positivos, es decir, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$ y $\sigma_{r+1} = \ ... \ = \sigma_n = 0$. Entonces, existen matrices ortogonales $U \in \mathbb{R}^{m\times m}$ y $V \in \mathbb{R}^{n\times n}$, así como una matriz $\bar{\Sigma} \in \mathbb{R}^{m\times n}$ de la forma

$$\bar{\Sigma} = \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix}, \tag{1.1}$$

con $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_r)$ tales que:

$$A = U\bar{\Sigma}V^T.$$

Demostración. El teorema espectral garantiza la existencia de una base ortonormal de autovectores $\{v_1, \ ..., \ v_n\}$ de A^TA para $\mathbb{R}^n$. Sea $V \in \mathbb{R}^{n\times n}$ la matriz cuyas columnas están formadas por los vectores de dicha base ortonormal:

$$V = [v_1, v_2...v_{n-1}, v_n].$$

A continuación, se obtiene la matriz U del enunciado. Al igual que V, U debe ser una matriz ortogonal, por lo que sus columnas formarán una base ortonormal de $\mathbb{R}^m$. Sea u_i definido como

$$u_i = \frac{Av_i}{\sigma_i}, \ 1 \leq i \leq r. \tag{1.2}$$

Como los vectores $v_1, v_2, ..., v_{n-1}, v_n$ son ortonormales, entonces:

$$\langle u_i, u_j \rangle = \frac{(Av_i)^T(Av_j)}{\sigma_i\sigma_j} = \frac{(\sigma_i v_i)^T(\sigma_j v_j)}{\sigma_i\sigma_j} = v_i^T v_j = \langle v_i, v_j \rangle = \begin{cases} 0 \ si \ i \neq j \\ 1 \ si \ i = j \end{cases}.$$

Por tanto, $u_1, \ ..., u_r$ construidos siguiendo (1.2) son ortonormales. No obstante, si $r < m$, aún se precisa añadir $m - r$ vectores adicionales $\{u_{r+1}, ..., u_{m-1}, u_m\}$ para formar una base de vectores ortonormales de $\mathbb{R}^m$. Partiendo del conjunto ortonormal

$$\{u_1, ..., u_{r-1}, u_r\},$$

se aplica el método de Gram-Schmidt al conjunto

$$\{u_1, \ ... \ , u_{r-1}, \ u_r, \ e_{r+1}\},$$

con e_{r+1} el vector $(r+1)-$ésimo de la base canónica de $\mathbb{R}^m$, obteniéndose un nuevo conjunto ortonormal

$$\{u_1, \ ... \ , u_{r-1}, \ u_r, \ u_{r+1}\}.$$

El proceso puede repetirse para los vectores $\{e_{r+2}, e_{r+3}...e_m\}$ para así obtener una base ortonormal $\{u_1, ..., u_m\}$ de $\mathbb{R}^m$. Colocando cada vector de esta base en forma de columna, se obtiene la matriz U de dimensión $m \times m$ definida como:

$$U = [u_1, \ u_2 \ ... \ u_{m-1}, \ u_m].$$

Para concluir la demostración, tenemos que probar que $A = U\bar{\Sigma}V^T$ o, equivalentemente, que $\bar{\Sigma} = U^TAV$. Se tienen en cuenta dos aspectos importantes:

1. Como $v_i = 0,\ i \in \{1, ..., r\}$ son los autovectores no nulos de A^TA, entonces

$$A^TAv_i = 0, \qquad r+1 \leq i \leq n. \tag{1.3}$$

Por tanto, multiplicando por v_i^T, se tiene

$$(Av_i)^T(Av_i) = 0, \quad r+1 \leq i \leq n \Longrightarrow \|Av_i\|_2^2 = 0, \quad r+1 \leq i \leq n,$$

lo que sólo ocurre si $Av_i = 0, \quad r+1 \leq i \leq n$.

2. Por otro lado, como U es ortogonal, entonces

$$U^TU = I_m, \tag{1.4}$$

con $I_m \in \mathbb{R}^{m\times m}$ la matriz identidad. En particular,

$$U^Tu_i = e_i,\ 1 \leq i \leq r. \tag{1.5}$$

Multiplicando por σ_i a ambos lados de (1.5), se obtiene la siguiente expresión:

$$\sigma_iU^Tu_i = \sigma_ie_i, \quad 1 \leq i \leq r. \tag{1.6}$$

Para concluir con la demostración, basta operar, teniendo en cuenta (1.3),(1.5) y (1.6), y se tiene que

$$\begin{aligned} U^TAV &= U^TA[v_1\ v_2\ \dots\ v_{n-1}\ v_n] \\ &= U^T[Av_1\ Av_2\ \dots\ Av_{n-1}\ Av_n] \\ &= U^T[Av_1\ Av_2\ \dots\ Av_r\ 0\ ...\ 0\ ] \\ &= U^T[\sigma_1u_1\ \sigma_2u_2\ ...\ \sigma_ru_r\ 0\ ...\ 0\ ] \\ &= [\sigma_1e_1\ \sigma_2e_2\ ...\ \sigma_re_r\ 0\ ...\ 0] \\ &= \bar{\Sigma}. \end{aligned}$$

Las columnas de U y V se llaman vectores singulares por la izquierda y por la derecha de A, respectivamente. □

Es importante reflejar que no existe pérdida de generalidad por asumir que $m \geq n$, pues en el caso de que ocurriera que $m < n$ basta con encontrar la descomposición en valores singulares para la matriz A^T en lugar de hallarla para la matriz A y, a continuación, trasponer de nuevo. Así, se obtendría la igualdad siguiente:

$$A^T = U\bar{\Sigma}V^T.$$

Finalmente, trasponiendo a ambos lados de la igualdad, se tendría que $A = V\bar{\Sigma}U^T$.

1.1.1. Interpretación Geométrica de la SVD

En esta sección, siguiendo a [3], damos una interpretación geométrica de la SVD, a partir del estudio de cómo una matriz $A \in \mathbb{R}^{m\times n}$, con $m \geq n$, transforma la esfera unitaria

$$S = \{x \in \mathbb{R}^n / \|x\|_2 \leq 1\}.$$

Sea $A = U\bar{\Sigma}V^T$ la SVD de la matriz A con valores singulares $\sigma_1 \geq \ ... \ \geq \sigma_r > 0, \ \ \sigma_i = 0, \quad i = r+1, \dots n$. Observemos que, como V es ortogonal, entonces V^T transforma S en S. Por tanto, si $x \in S$, entonces $y = V^T x = (y_1, \ ..., y_n)^T \in S$, de modo que:

$$\bar{\Sigma}y = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \sigma_1 y_1 \\ \sigma_2 y_2 \\ \vdots \\ \sigma_r y_r \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \bar{y_1} \\ \bar{y_2} \\ \vdots \\ \bar{y_r} \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

de manera que

$$\left(\frac{\bar{y_1}}{\sigma_1}\right)^2 + \left(\frac{\bar{y_2}}{\sigma_2}\right)^2 + \ ... \ + \left(\frac{\bar{y_r}}{\sigma_r}\right)^2 \leq 1,$$

que representa un elipsoide r-dimensional en $\mathbb{R}^m$ con semiejes de longitud σ_i, $i = 1, \ ..., \ r$. Finalmente, al ser U ortogonal, de nuevo representa geométricamente una rotación o reflexión en $\mathbb{R}^m$. En resumen, A transforma la esfera unitaria $S \subset \mathbb{R}^n$ en un elipsoide r-dimensional rotado en $\mathbb{R}^m$ con semiejes dados por los valores singulares no nulos.

Ejemplo 1.1.1. Vamos a ilustrar de forma gráfica la interpretación geométrica de la SVD. Para ello, se hará uso del código de MATLAB reflejado en el apéndice A.1.

Consideremos la matriz $A \in \mathbb{R}^{3\times 3}$ dada por:

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 2 & \frac{1}{2} & 1 \\ -1 & -1 & 2 \end{pmatrix}, \quad \text{con valores singulares} \quad \Sigma = \begin{pmatrix} 3{,}5369 & 0 & 0 \\ 0 & 2{,}4846 & 0 \\ 0 & 0 & 1{,}2517 \end{pmatrix}.$$

El código permite visualizar cómo una matriz A transforma una esfera unitaria en un elipsoide. Ello se representa gráficamente en 1.1 y 1.2.

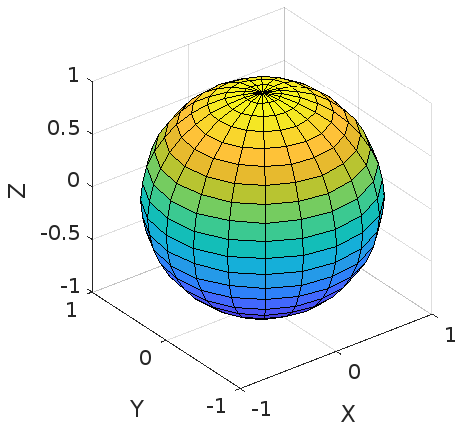


Figura 1.1: Esfera unitaria original.

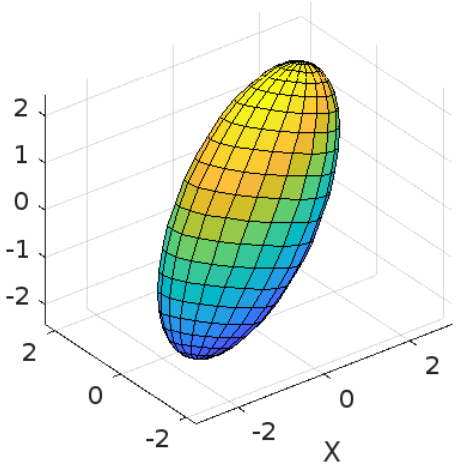


Figura 1.2: Esfera transformada.

Primero, genera las coordenadas de la esfera y muestra su representación original en un subgráfico. Luego, aplica A a cada punto de la esfera mediante multiplicación matricial, almacenando las coordenadas transformadas. Finalmente, en otro subgráfico, muestra el elipsoide resultante, ilustrando el efecto geométrico de la transformación.

Es interesante también ilustrar un caso en el que la matriz A considerada tenga algún valor singular nulo, veámoslo:

Ejemplo 1.1.2. De nuevo, para ilustrar este ejemplo se hará uso del código de MATLAB reflejado en el apéndice A.1. Sean ahora dos matrices $B, C \in \mathbb{R}^{3\times 3}$ de la que suponemos valores singulares dados por:

$$\Sigma_B = \begin{pmatrix} 3{,}5369 & 0 & 0 \\ 0 & 2{,}4846 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad y \quad \Sigma_C = \begin{pmatrix} 3{,}5369 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Es decir, se trata de la matriz Σ del ejemplo 1.1.1 a la que se le ha anulado el tercer valor singular y el segundo y tercer valor singular, respectivamente. El resultado de transformar la esfera con una matriz que tiene k valores singulares nulos es que ésta se aplana en k de sus dimensiones. En este ejemplo concreto, se proyecta sobre un espacio de dimensión $3 - k$. Ello se representa gráficamente en 1.3 y 1.4.

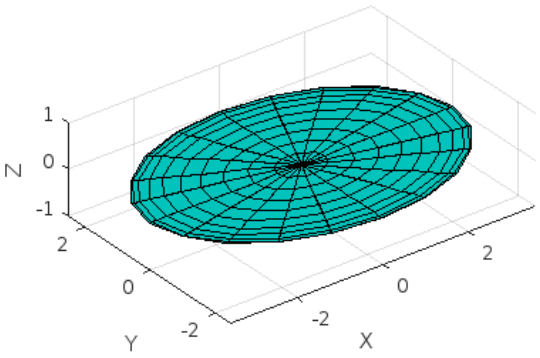


Figura 1.3: Esfera unitaria transformada por B.

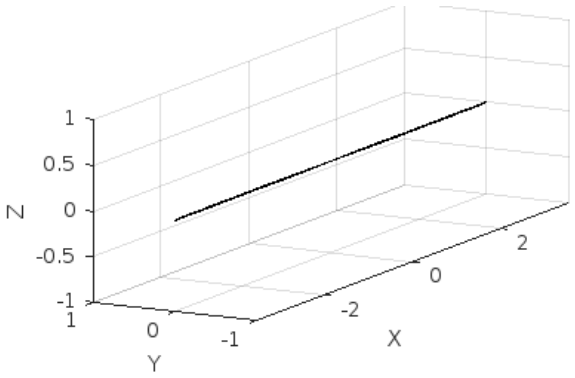


Figura 1.4: Esfera unitaria transformada por C.

1.1.2. La SVD y el Teorema de Schur

La descomposición en valores singulares está relacionada con la diagonalización de una matriz simétrica.

Teorema 1.1.2. Sea $A = U\tilde{\Sigma}V^T$ la SVD de una matriz $A \in \mathbb{R}^{m\times n}$, con $m \geq n$ y $\tilde{\Sigma}$ en la forma (1.1).

1. Los autovalores de la matriz simétrica A^TA son $\lambda_i = \sigma_i^2$, con $i = 1, \ ..., n$ y los vectores singulares por la derecha v_i son autovectores ortonormales.

2. Los autovalores de la matriz simétrica AA^T son σ_i^2, $i = 1, ..., r$ y $m - r$ ceros, y las columnas de U forman una base ortonormal de autovectores.

3. Supongamos que $m = n$, A es simétrica y $A = P\Lambda P^T$ es una diagonalización ortogonal de A, con $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$, $P = [P_1\ , ..., P_n]$ y $PP^T = I_n$. Entonces,

$$A = U\Sigma V^T,$$

con $U = P$, $\Sigma = \text{diag}(\sigma_1, ..., \sigma_n)$, $\sigma_i = |\lambda_i|$, con $i = 1...n$ y $V = [v_1,\ ..., v_n]$, con $v_i = \text{sign}(\lambda_i)P_i\ ,\ i = 1...n$.

Demostración.

1. De la SVD se tiene
$$A^TA = V\tilde{\Sigma}^TU^TU\tilde{\Sigma}V^T = V\tilde{\Sigma}^T\tilde{\Sigma}V^T, \tag{1.7}$$
donde, de (1.1), se tiene que
$$\tilde{\Sigma}^T\tilde{\Sigma} = \begin{pmatrix}\Sigma^2 & 0\\ 0 & 0\end{pmatrix} \in \mathbb{R}^{n\times n}, \tag{1.8}$$
de modo que (1.7) es una diagonalización de A^TA, con las columnas de V como autovectores y los elementos diagonales de (1.8) como autovalores (*cf.* definición 1.1.1).

2. De la demostración del teorema 1.1.1, podemos escribir
$$U = [U_1\ \ U_2],$$
donde $U_1 = [u_1, ..., u_r] \in \mathbb{R}^{m\times r}$, con $u_i,\ i = 1, ..., r$, dado por (1.2), y $U_2 = [u_{r+1}, ..., u_m] \in \mathbb{R}^{m\times(m-r)}$, completada por ortogonalización a partir de la base canónica. Entonces,
$$AA^T = U\tilde{\Sigma}V^TV\tilde{\Sigma}^TU^T = [U_1,\ U_2]\tilde{\Sigma}\tilde{\Sigma}^T[U_1,\ U_2]^T, \tag{1.9}$$
donde $\tilde{\Sigma}\tilde{\Sigma}^T = \begin{pmatrix}\Sigma^2 & 0\\ 0 & 0\end{pmatrix} \in \mathbb{R}^{m\times m}$, y , de (1.2) y el apartado 1, tenemos que,
$$AA^Tu_i = AA^T\frac{Av_i}{\sigma_i} = \sigma_iAv_i = \sigma_i^2u_i, \quad i = 1, ...r,$$
de manera que (1.9) es una diagonalización de AA^T.

3. Como $|\lambda_i| = \text{sign}(\lambda_i)\lambda_i, \quad i = 1, ..., n$, es claro que $U\Sigma V^T = P\Lambda P^T = A$. □

1.2. Extensiones de la SVD

En esta sección, se exploran variantes de la descomposición en valores singulares, incluyendo la SVD Reducida, la descomposición en valores singulares generalizada (GSVD) y la descomposición CS. Se subraya la eficiencia de la SVD Reducida, potenciada por optimizaciones en MATLAB, y se presenta la GSVD como una ampliación de la SVD para situaciones más complejas.

1.2.1. La SVD Reducida

Otra versión de la SVD que aparece en la literatura (véase, *e.g.*, [6]) factoriza la matriz $A \in \mathbb{R}^{m\times n}$, $m \geq n$, en la forma

$$A = \hat{U}\hat{\Sigma}\hat{V}^T,$$

con $\hat{U} \in \mathbb{R}^{m\times n}$ tal que $\hat{U}^T\hat{U} = I_n$, $\hat{V} \in \mathbb{R}^{n\times n}$ con $\hat{V}^T\hat{V} = I_n$ y $\hat{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n\times n}$.

Esta factorización de la matriz A se llama descomposición en valores singulares reducida. Observemos que las columnas de $\hat{U}$ son n vectores ortonormales en un espacio de dimensión m, por lo tanto si $m > n$ no constituyen una base; sin embargo, es posible agregarle $m - n$ columnas de manera que la matriz resultante sea ortogonal. De manera formal, se define como sigue:

Definición 1.2.1. La descomposición en valores singulares reducida de una matriz $A \in \mathbb{R}^{m\times n}$ se define como:

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

- $\hat{U} \in \mathbb{R}^{m\times n}$ es una matriz cuyas columnas son los vectores singulares izquierdos de A y forman una base ortonormal para el espacio columna de A (véase la definición 1.3.1).
- $\hat{\Sigma} \in \mathbb{R}^{n\times n}$ es una matriz diagonal que contiene los valores singulares de A, que son no negativos y están ordenados de manera decreciente.
- $\hat{V}^T \in \mathbb{R}^{n\times n}$ es la trasposición de una matriz cuyas columnas son los vectores singulares derechos de A y forman una base ortonormal para el espacio fila de A (véase la definición 1.3.1).

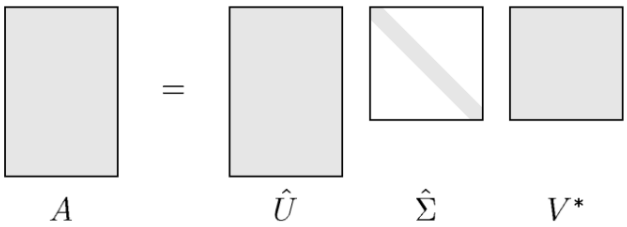


Figura 1.5: Representación esquemática de la SVD reducida, extraída de [8].

Comparación de Tiempos de Ejecución: SVD Completa y SVD Reducida

En MATLAB, se puede comprobar de forma sencilla que los tiempos de ejecución de la SVD reducida disminuyen notablemente frente a los tiempos de ejecución de la SVD general. Esto parece lógico, pues MATLAB tiene en cuenta la estructura de las matrices de entrada, y en concreto, la presencia o no de ceros. Esta última condición es aprovechada para optimizar los cálculos. Esto ocurre, por ejemplo, en el caso de matrices dispersas. Cuando a MATLAB se le proporciona la información de que la matriz con la que se va a operar tiene una estructura dispersa, los cálculos se efectúan de forma más eficiente.

Para ilustrarlo, se define en MATLAB la función `comparar TiemposSVD()` (véase el apéndice A.2), para comparar el tiempo de ejecución entre la SVD completa y la SVD reducida para

un conjunto de matrices aleatorias generadas. El proceso seguido por la función se describe a continuación:

1. **Solicitud del número de matrices:** La función comienza solicitando al usuario que ingrese el número de matrices (`n`) a procesar mediante la función `input`. Este número determinará cuántas matrices aleatorias se generarán y procesarán posteriormente.

2. **Solicitud de las dimensiones de las matrices:** Posteriormente, la función solicita al usuario que proporcione las dimensiones de las matrices a generar: `m` para el número de filas y `p` para el número de columnas. Estas entradas también son validadas para asegurar que sean enteros positivos.

3. **Inicialización de vectores de tiempos:** Se inicializan dos vectores, `tiempos_completa` y `tiempos_reducida`, con longitud `n` y todos sus elementos en cero. Estos vectores almacenarán los tiempos de ejecución para cada matriz procesada mediante SVD completa y SVD reducida, respectivamente.

4. **Procesamiento de las matrices:** La función entra en un bucle `for` que se ejecuta `n` veces. En cada iteración, se realiza lo siguiente:

   - Se genera una matriz aleatoria `A` de tamaño `m` x `p` utilizando `rand(m, p)`.
   - Se mide y calcula la SVD completa de `A`, almacenando el tiempo de ejecución en el vector `tiempos_completa`.
   - De manera similar, se mide y calcula la SVD reducida de `A`, almacenando el tiempo de ejecución en el vector `tiempos_reducida`.

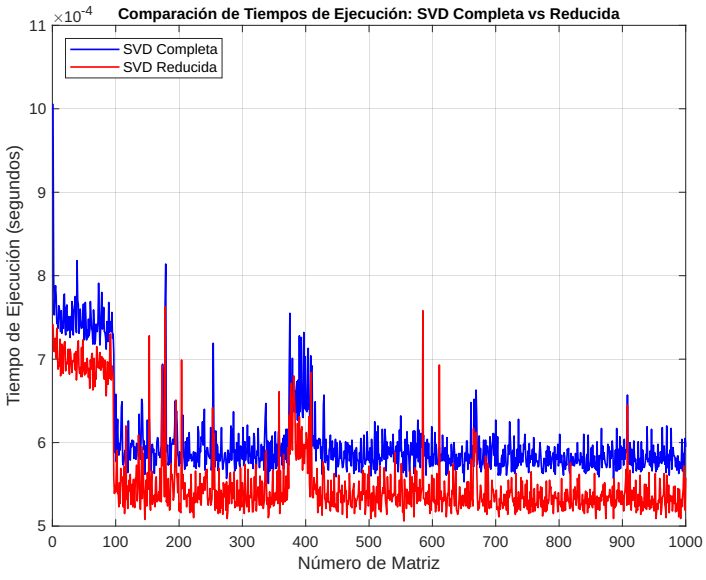


Figura 1.6: Eficiencia comparada en SVD reducida y completa en 1000 matrices de 100x50.

En la figura 1.6, se visualiza una bajada súbita en el tiempo de ejecución, aproximadamente tras la iteración con la matriz número 100. Esto se puede deber a los siguientes factores.

1. **Optimización Just-In-Time (JIT):** MATLAB optimiza el código mientras se ejecuta, haciéndolo más rápido después de las primeras iteraciones.

2. **Gestión de la Memoria:** Al principio, MATLAB ajusta cómo maneja la memoria, lo cual puede afectar el rendimiento inicial. Con el tiempo, este manejo se vuelve más eficiente.

3. **Efectos de Termorregulación:** Los procesadores pueden reducir su velocidad para evitar sobrecalentarse durante las primeras iteraciones. Una vez estabilizados, pueden funcionar más rápido.

Además de esto, en iteraciones aisladas del código, parece ocurrir un fenómeno contradictorio, pues se observan picos en el tiempo de CPU de la SVD reducida, mucho mayores en esas iteraciones concretas, que en el resto. Esto parece deberse a que, como las matrices se generaron de forma aleatoria, su estructura quizás fue más compleja y con menos entradas nulas que en otras matrices involucradas, lo que provoca un aumento considerable en el tiempo de CPU que conlleva la ejecución de la SVD, tanto reducida como completa. Es relevante recordar, además, que, cuanto menor sea el número de valores singulares nulos, más se aproxima el tiempo de ejecución de la SVD reducida al correspondiente a la SVD completa.

1.2.2. La Descomposición en Valores Singulares Generalizada (GSVD)

Siguiendo a [25], se estudia ahora la descomposición en valores singulares generalizada o GSVD, que extiende las capacidades de la SVD clásica.

Teorema 1.2.1. (**de la GSVD.**) Sean dos matrices $A \in \mathbb{R}^{m\times p}$ y $B \in \mathbb{R}^{n\times p}$ con $n \geq p$. Sea $q = \text{mín}\{m, p\}$. Entonces, existen dos matrices ortonormales $U_A \in \mathbb{R}^{m\times m}$ y $U_B \in \mathbb{R}^{n\times n}$, y una matriz invertible $X \in \mathbb{R}^{p\times p}$ tales que

$$U_A^T AX = \text{diag}(\alpha_1, \ldots, \alpha_q) \qquad U_B^T BX = \text{diag}(\beta_1, \ldots, \beta_p),$$

donde $\alpha_1 \geq \cdots \geq \alpha_q \geq 0$, y $0 \leq \beta_1 \leq \cdots \leq \beta_p$.

El teorema GSVD fue propuesto originalmente por Loan en [23], donde se requiere que $n \geq p$ (o $m \geq p$). Posteriormente, Paige y Saunders desarrollaron en 1981 una formulación más general, [15], para la GSVD en la cual las matrices A y B sólo requieren tener el mismo número de columnas. Paige y Saunders también estudiaron una GSVD de submatrices de una matriz de columnas ortonormales. Esto se conoce como descomposición CS, que se presenta a continuación.

Teorema 1.2.2. (**de la Descomposición CS**.) Sea $Q \in \mathbb{R}^{(m+n)\times p}$ una matriz de columnas ortonormales. Se divide dicha matriz como $Q^T = [Q_1^T, Q_2^T]$ donde $Q_1 \in \mathbb{R}^{m\times p}$ y $Q_2 \in \mathbb{R}^{n\times p}$. Entonces existen matrices ortonormales $U_1 \in \mathbb{R}^{m\times m}$, $U_2 \in \mathbb{R}^{n\times n}$, y $V_1 \in \mathbb{R}^{p\times p}$ tales que

$$U_1^T Q_1 V_1 = C \quad \text{y} \quad U_2^T Q_2 V_1 = S,$$

donde

$$C = \begin{pmatrix} I_r & 0 & 0 \\ 0 & C_1 & 0 \\ 0 & 0 & 0_C \end{pmatrix}, \quad S = \begin{pmatrix} 0_S & 0 & 0 \\ 0 & S_1 & 0 \\ 0 & 0 & I_{p-r-s} \end{pmatrix},$$

donde I_r es la matriz identidad de dimensión $r \times r$, 0_C es la matriz de ceros de dimensión $(m-r-s)\times(p-r-s)$, y 0_S es la matriz de ceros de dimensión $(n+r-p)\times r$. Además, $C_1 = \text{diag}(\alpha_1, \ldots, \alpha_s)$ y $S_1 = \text{diag}(\sqrt{1-\alpha_1^2}, \ldots, \sqrt{1-\alpha_s^2})$, con $1 > \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_s > 0$.

Las dimensiones de las matrices son las siguientes:

Matriz	Dimensión
$\mathbf{I_r}$	$r \times r$
$\mathbf{C_1}$	$s \times s$
$\mathbf{0_C}$	$(m-r-s) \times (p-r-s)$
$\mathbf{0_S}$	$(n+r-p) \times r$
$\mathbf{S_1}$	$s \times s$
$\mathbf{I_{p-r-s}}$	$(p-r-s) \times (p-r-s)$

Demostración. Véase [25]. □

Como observación, es importante señalar que $Q_1 = U_2 S V_1^T$ no necesariamente representa una SVD completa de Q_1, debido a que algunos de los elementos no nulos de S podrían no estar en la diagonal principal. Sin embargo, si $n \geq p$, entonces podemos mover las primeras $n-p$ filas de S para que sean las últimas $n-p$ filas mediante la premultiplicación por alguna matriz de permutación P. Es decir,

$$P^T U_2^T Q_1 V_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & S_1 & 0 \\ 0 & 0 & I_{p-r-s} \\ 0 & 0 & 0 \end{pmatrix},$$

donde las dimensiones de las columnas de la matriz, de la primera a la tercera, son r, s, y $p-s-r$, respectivamente, mientras que las dimensiones de las filas son, en orden, r, s, y $p-s-r$ y $n-p$. Esta es la razón por la cual se requiere la restricción $n \geq p$ en el teorema 1.2.1 (donde A y B corresponden a Q_1 y Q_2, respectivamente).

1.3. Algunas aplicaciones de la SVD

En esta sección, se estudian algunas de las aplicaciones teóricas de la SVD. Se inicia el análisis con una revisión de los subespacios fundamentales de una matriz. Se prosigue con los conceptos de norma euclídea y el número de condición, los cuales se expresan en términos de los valores singulares y son esenciales para la estabilidad y eficiencia de los cálculos matriciales.

A continuación, se analiza el rango de una matriz, determinado por el número de valores singulares no nulos, y su relación con problemas lineales de mínimos cuadrados, donde la SVD se utiliza para calcular la solución en el caso de que la matriz tenga rango máximo o la solución de norma mínima en el caso de rango deficiente, así como para determinar la pseudoinversa de una matriz. Finalmente discutiremos las aproximaciones de rango bajo de una matriz utilizando la SVD.

1.3.1. Subespacios fundamentales de una matriz

De acuerdo con [21], la relevancia de la SVD se extiende al análisis de los cuatro subespacios fundamentales asociados con cualquier matriz.

Definición 1.3.1. Dada $A \in \mathbb{R}^{m \times n}$, se definen:

1. $R(A) = \{y \mid y = Ax, x \in \mathbb{R}^n\} \subset \mathbb{R}^m$, el espacio columna.
2. $R(A)^\perp$, el complemento ortogonal de $R(A)$: si $z \in R(A)^\perp$ entonces $z^T y = 0$.
3. $R(A^T) = \{z \mid z = A^T y, y \in \mathbb{R}^m\} \subset \mathbb{R}^n$, el espacio fila.
4. $N(A) = \{x \mid Ax = 0\}$, el espacio nulo.

Teorema 1.3.1. Se cumplen las siguientes relaciones:

1. $R(A)^\perp = N(A^T)$. Así, $\mathbb{R}^m = R(A) \oplus N(A^T)$.
2. $R(A^T)^\perp = N(A)$. Así, $\mathbb{R}^n = R(A^T) \oplus N(A)$.

Demostración. Sea $y \in R(A)$ y $z \in R(A)^\perp$. Entonces por definición $0 = y^T z = (Ax)^T z = x^T(A^T z), \forall x$. Por lo tanto, se sigue que $A^T z = 0$ lo que significa que $z \in N(A^T)$ y por lo tanto $R(A)^\perp \subset N(A^T)$. De manera similar, se verifica el segundo enunciado. □

La SVD y los cuatro subespacios fundamentales

El rango de una matriz es el número de columnas o filas linealmente independientes. A continuación, presentamos un resultado previo que nos permitirá estudiar el teorema que establece la relación entre el rango de una matriz y sus valores singulares. Para consultar la demostración, véase *e.g.* [8].

Teorema 1.3.2. Si $A \in \mathbb{R}^{m\times n}$, $X \in \mathbb{R}^{m\times m}$ invertible, e $Y \in \mathbb{R}^{n\times n}$ invertible, entonces $\text{rango}(XAY) = \text{rango}(A)$.

Teorema 1.3.3. El rango de una matriz $A \in \mathbb{R}^{m\times n}$ es el número de valores singulares no nulos.

Demostración. Sea $A = U\tilde{\Sigma}V^T$ la descomposición en valores singulares de A. Las matrices ortogonales son invertibles, por lo que por el teorema 1.3.2,

$$\text{rango}(A) = \text{rango}(U\tilde{\Sigma}V^T) = \text{rango}(\tilde{\Sigma}).$$

La matriz $\tilde{\Sigma}$ tiene la forma

$$\tilde{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix},$$

donde σ_i son los valores singulares no nulos de A. Los vectores columna de Σ correspondientes a los valores singulares no nulos forman una base para el espacio columna de Σ, luego

$$\text{rango}(A) = \text{rango}(\tilde{\Sigma}) = r.$$

□

A partir de los componentes de la SVD, podemos determinar otras propiedades de la matriz original. Sean u_i, $1 \le i \le m$, y v_i, $1 \le i \le n$, los vectores columna de U y V, respectivamente. Entonces, de la SVD de A, se tiene que $AV = U\Sigma$ y, por tanto,

$$Av_i = \sigma_i u_i, \quad \sigma_i \neq 0, \quad 1 \le i \le r,$$

$$Av_i = 0, \quad r+1 \leq i \leq n.$$

Dado que U y V son matrices ortogonales, todos los u_i y v_i son linealmente independientes. Para $1 \leq i \leq r$, los $\sigma_i u_i$ están en el espacio columna de A, y además sabemos que $\sigma_i \neq 0,\ 1 \leq i \leq r$. Dado que, por el teorema 1.3.3 el rango de A es r, entonces $\{u_1, ..., u_r\}$ es una base para $R(A)$.

Para $r+1 \leq i \leq n$, $Av_i = 0$, por lo que $v_i \in N(A)$. Dado que, por el teorema 1.3.1, ocurre que $\dim R(A) + \dim N(A) = n$, la dimensión del espacio nulo de A es $n-r$, por lo que $\{v_{r+1}, ..., v_n\}$ es una base para el espacio nulo de A.

Por último, los dos subespacios fundamentales restantes son el espacio columna y el espacio nulo de A^T. Como antes, calcularemos una base de cada uno de ellos. Si tomamos la traspuesta de la SVD de A, el resultado es:

$$A^T = V\tilde{\Sigma}^T U^T. \tag{1.10}$$

Aplicando el mismo procedimiento que utilizamos para determinar una base ortonormal para $R(A)$, se sigue que $\{v_1, ..., v_r\}$, es una base para el espacio columna de A^T. Nótese que el espacio columna de A^T es el espacio fila de A. Sabemos que $\{v_{r+1}, ..., v_n\}$ es una base para el espacio nulo de A. Dado que todos los v_i son ortogonales, se sigue que los vectores en el espacio columna de A^T son ortogonales a los vectores en el espacio nulo de A.

Finalmente, a partir de (1.10) y siguiendo el procedimiento anterior, vemos que $\{u_{r+1}, ..., u_m\}$ es una base ortonormal para el espacio nulo de A^T. Hemos demostrado previamente que $\{u_1, ..., u_r\}$ es una base para el espacio columna de A. Así, el espacio columna de A es ortogonal al espacio nulo de A^T.

	Rango	**Espacio Nulo**
A	$u_i,\ 1 \leq i \leq r$	$v_i,\ r+1 \leq i \leq n$
A^T	$v_i,\ 1 \leq i \leq r$	$u_i,\ r+1 \leq i \leq m$

Tabla 1.1: Los cuatro subespacios fundamentales de A, extraído de [8].

Teorema 1.3.4. (de la dimensión del rango y del espacio fila.) La dimensión del espacio columna y la dimensión del espacio fila de una matriz son iguales y se denomina el rango de la matriz.

Demostración. Hemos comprobado que si r es el número de valores singulares no nulos, $\{u_1, ..., u_r\}$ es una base para el espacio columna de A, y $\{v_1, ..., v_r\}$ es una base para el espacio columna de A^T, que es el espacio fila de A. □

Gilbert Strang[1] [21], enfatiza la importancia de estos subespacios al describir el teorema fundamental del álgebra lineal. Este teorema se ilustra a menudo a través del diagrama de Strang, que visualiza la interacción entre estos subespacios y su ortogonalidad mutua. Se presenta a continuación:

[1]Gilbert Strang (1934-) es un matemático estadounidense, profesor en el MIT, y autor de influyentes libros de texto en álgebra lineal y análisis numérico.

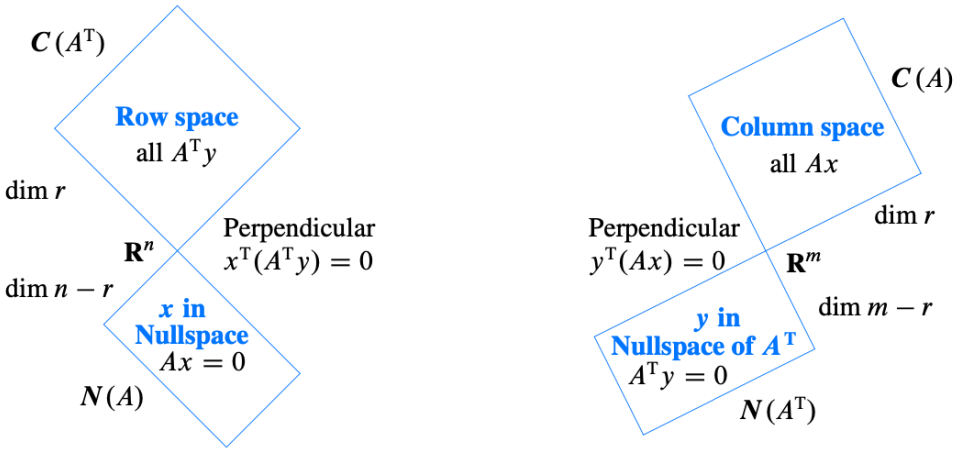


Figura 1.7: Diagrama de Strang que ilustra los subespacios fundamentales de una matriz.

La importancia del diagrama de Strang radica en los siguientes puntos:

- **Ilustración de la Ortogonalidad:** Muestra cómo el espacio columna de A es ortogonal al núcleo de A^T y cómo el espacio fila de A es ortogonal al núcleo de A.
- **Entendimiento del Rango:** El diagrama ayuda a visualizar que el rango de A (el número de dimensiones del espacio columna) es igual al rango de A^T (el número de dimensiones del espacio fila), el Teorema (1.3.4).
- **Aplicaciones en SVD:** La SVD de una matriz A se basa directamente en la relación entre estos cuatro subespacios. La SVD descompone A en el producto de tres matrices $U\Sigma V^T$, donde las columnas de U y V son bases ortonormales para los espacios columna y fila de A y A^T, respectivamente, y Σ contiene los valores singulares que conectan estos espacios.

1.3.2. Norma euclídea. Número de Condición de una Matriz

Siguiendo a [6, 8], estudiaremos aquí como la SVD está estrechamente relacionada con el cálculo de la norma euclídea de una matriz.

Lema 1.3.1. Sea A es una matriz de dimensión $m \times n$ arbitraria, y denotemos por $\{\sigma_1, ..., \sigma_n\}$ los valores singulares de A. Entonces,

$$\|A\|_2 = \max_{1 \leq j \leq n} \sigma_j$$

donde $\|A\|_2$ denota la norma 2 de la matriz A.

Demostración. Sea $B = A^T A \in \mathbb{R}^{n \times n}$. Como B es simétrica, por los lemas 1.1.1 y 1.1.2, sus autovalores son todos reales y no negativos, y, por tanto, es diagonalizable ortogonalmente. Sean $P \in \mathbb{R}^{n \times n}$ ortogonal y $D \in \mathbb{R}^{n \times n}$ diagonal tales que:

$$D = P^T B P = P^T A^T A P. \tag{1.11}$$

D contiene los autovalores λ_i, $i \in \{1, \ ..., n\}$ de $A^T A$, y P tiene por columnas una base de autovectores ortonormales. Sea $y = P^T x$, teniendo en cuenta la definición de norma matricial y operando con (1.11), se tiene que:

$$\|A\|_2^2 = \max_{x\neq 0}\frac{\|Ax\|_2^2}{\|x\|_2^2} = \max_{x\neq 0}\frac{x^TA^TAx}{\|x\|_2^2} = \max_{x\neq 0}\frac{x^TPDP^Tx}{\|x\|_2^2}$$
$$= \max_{x\neq 0}\frac{(P^Tx)^TD(P^Tx)}{\|P^Tx\|_2^2} = \max_{y\neq 0}\frac{y^TDy}{\|y\|_2^2} = \max_{y\neq 0}\frac{\sum_{i=1}^n \lambda_i y_i^2}{\sum_{i=1}^n y_i^2}.$$

Sea $\mu = \max \lambda_i \geq 0$. Entonces, se tiene la desigualdad:

$$\max_{y\neq 0}\frac{\sum_{i=1}^n \lambda_i y_i^2}{\sum_{i=1}^n y_i^2} \leq \mu\frac{\sum_{i=1}^n y_i^2}{\sum_{i=1}^n y_i^2} = \mu, \tag{1.12}$$

Por tanto, por (1.3.2), se deduce que $\|A\|_2 \leq \sqrt{\mu}$. Se supone ahora que $\mu = \lambda_k$ para cierto $k \in \{1, ..., n\}$. Se tiene que $Pe_k = x_k$ dónde x_k es la k-ésima columna de P, y e_k es el k-ésimo vector de la base canónica de $\mathbb{R}^n$. Como P es una matriz ortogonal, su matriz inversa es P^T, luego $e_k = P^Tx_k$. Escogiendo entonces $y = e_k$, la desigualdad (1.12) se convierte en una igualdad, luego $\|A\|_2 = \sqrt{\mu}$ y se concluye la demostración. □.

La SVD se puede calcular de forma precisa, por lo que nos proporciona un método alternativo para hallar tanto la norma euclídea, como la norma de Frobenius, que se define a continuación junto con la presentación de algunos resultados relevantes.

Definición 1.3.2. Sea $A \in \mathbb{R}^{m\times n}$, se define la norma de Frobenius de A como

$$\|A\|_F = \sqrt{\sum_{i=1}^m\sum_{j=1}^n |a_{ij}|^2}.$$

Lema 1.3.2. Sea $A \in \mathbb{R}^{m\times n}$, la norma de Frobenius se puede calcular como

$$\|A\|_F^2 = tr(A^TA) = tr(AA^T).$$

Demostración. Notemos que el elemento diagonal i-ésimo de A^TA viene dado por

$$(A^TA)_{ii} = \sum_{k=1}^m |a_{ki}|^2 = \sum_{k=1}^m a_{ki}^2,$$

donde la última igualdad es cierta por ser A real, y, por tanto,

$$tr(A^TA) = \sum_{i=1}^n (A^TA)_{ii} = \sum_{i=1}^n\sum_{k=1}^m a_{ki}^2 = \|A\|_F^2.$$

La demostración de $\|A\|_F^2 = tr(AA^T)$ es análoga. □

A continuación, vamos a probar la invariancia de la norma de Frobenius bajo la multiplicación por matrices ortogonales, resultado fundamental para establecer la relación entre el cálculo de la norma de Frobenius y los valores singulares matriciales.

Lema 1.3.3. Sea $A \in \mathbb{R}^{m\times n}$, y sean $U \in \mathbb{R}^{m\times m}$ y $V \in \mathbb{R}^{n\times n}$ matrices ortogonales, entonces $\|UAV\|_F^2 = \|A\|_F^2$.

Demostración. Siguiendo a [8], calculemos en primer lugar la norma de Frobenius del producto $UA \in \mathbb{R}^{m\times n}$,

$$\|UA\|_F^2 = tr((UA)^T(UA)) = tr((A^TU^T)(UA)) = tr(A^TA) = \|A\|_F^2.$$

Así, se ha demostrado que la norma de Frobenius es invariante bajo la multiplicación izquierda por una matriz ortogonal. Ahora, calculemos la norma de Frobenius del producto $AV \in \mathbb{R}^{m\times n}$,

$$\|AV\|_F^2 = tr((AV)(AV)^T) = tr((AV)(V^TA^T)) = tr(AA^T) = \|A\|_F^2,$$

por lo que la norma de Frobenius es invariante bajo la multiplicación derecha por una matriz ortogonal. Por último, calculemos la norma de Frobenius del producto completo,

$$\|UAV\|_F^2 = \|U(AV)\|_F^2 = \|AV\|_F^2 = \|A\|_F^2.$$

□

Teorema 1.3.5. Sea $A \in \mathbb{R}^{m\times n}$, y sean $\sigma_1, \sigma_2, ..., \sigma_r$ los valores singulares no nulos de A, entonces,

$$\|A\|_F = \left(\sum_{i=1}^{r} \sigma_i^2\right)^{1/2}.$$

Demostración. Por el teorema 1.1.1, existen matrices ortogonales U y V tales que $A = U\tilde{\Sigma}V^T$. Entonces, teniendo en cuenta el lema 1.3.3 y (1.1),

$$\|A\|_F = \|U\tilde{\Sigma}V^T\|_F = \|\tilde{\Sigma}\|_F.$$

□

Una vez estudiadas las propiedades de las normas matriciales en relación con la SVD, continuemos el análisis introduciendo el concepto de número de condición y recordando sus propiedades.

Definición 1.3.3. El número de condición de una matriz $A \in \mathbb{R}^{n\times n}$ invertible se define como $\kappa_p(A) = \|A\|_p \cdot \|A^{-1}\|_p$, donde $\|\cdot\|_p$ es una norma p matricial determinada.

En esta parte de la sección, nos centraremos fundamentalmente en la norma euclídea o norma 2, y en consecuencia, en el número de condición definido como,

$$\kappa_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2.$$

Teorema 1.3.6. Sea $A \in \mathbb{R}^{n\times n}$ con valores singulares $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n > 0$. La norma euclídea cumple las propiedades que siguen:

1. $\|A^T\|_2 = \sigma_1$.
2. $\|A^TA\|_2 = \sigma_1^2$.
3. $\|A^{-1}\|_2 = \frac{1}{\sigma_n}$.

Demostración.

1. Basta con demostrar que $\|A^T\|_2 = \|A\|_2$ puesto que, por el teorema 1.3.1, ya sabemos que $\|A\|_2 = \sigma_1$. En primer lugar, se opera con la norma euclídea y, empleando la definición de norma matricial,

$$\|Ax\|_2^2 = (Ax)^{\mathrm{T}}(Ax) = x^{\mathrm{T}}A^{\mathrm{T}}Ax = \langle x, A^{\mathrm{T}}Ax\rangle \leq \|x\|_2 \left\|A^{\mathrm{T}}Ax\right\|_2 \leq \|x\|_2^2 \left\|A^{\mathrm{T}}A\right\|_2. \quad (1.13)$$

Aplicando raíces cuadradas a ambos lados de (1.13),

$$\|Ax\|_2 \leq \sqrt{\|A^T A\|_2}\|x\|_2. \tag{1.14}$$

De nuevo, por la definición de norma matricial y teniendo en cuenta (1.14)

$$\|A\|_2 = \max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} \leq \max_{x\neq 0} \frac{\sqrt{\|A^{\mathrm{T}}A\|_2}\|x\|_2}{\|x\|_2} = \sqrt{\|A^{\mathrm{T}}A\|_2}. \tag{1.15}$$

Entonces,

$$\|A\|_2^2 \leq \|A^T A\|_2.$$

Ahora, es necesario discutir dos casos. Para el caso en que $\|A\|_2 \neq 0$:

$$\|A\|_2^2 \leq \|A^T A\|_2 \leq \|A\|_2\|A^T\|_2 \implies \|A\|_2 \leq \|A^T\|_2.$$

Por otro lado, para el caso en que $\|A\|_2 = 0$, ocurre que

$$\|A\|_2 = 0 \leq 0 = \|A^T\|_2.$$

Por último, reemplazando A por A^T, se llega a $\|A\|_2 = \|A^T\|_2$.

2. Como $A^T A$ es simétrica,

$$\|A^T A\|_2 = |\lambda_{\text{máx}}| = \text{máx}\{|\lambda| : \lambda \text{ autovalor de } \ A^T A\}.$$

El resultado se deduce de la definición 1.1.1.

3. Dada la descomposición en valores singulares de la matriz A:

$$A = U\tilde{\Sigma}V^T,$$

donde U y V son matrices unitarias, y $\tilde{\Sigma}$ es una matriz diagonal con los valores singulares $\sigma_1, \sigma_2, \ldots, \sigma_n$ en la diagonal, ordenados de mayor a menor. La matriz inversa de A se puede expresar como:

$$A^{-1} = V\tilde{\Sigma}^{-1}U^T$$

Donde $\tilde{\Sigma}^{-1}$ es la matriz diagonal con los inversos de los valores singulares en la diagonal, es decir, $\tilde{\Sigma}^{-1} = \text{diag}(1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_n)$. El resultado se deduce entonces del apartado 1. □

Corolario 1.3.1. Sea $A \in \mathbb{R}^{n\times n}$ una matriz no singular, entonces $\kappa_2(A) = \sigma_1/\sigma_n$, donde $\sigma_1 = \text{máx}_{1\leq j\leq n}\,\sigma_j$ y $\sigma_n = \text{mín}_{1\leq j\leq n}\,\sigma_j$.

1.3.3. Problema lineal de mínimos cuadrados. Pseudoinversa

En esta sección estudiaremos cómo utilizar la SVD para la resolución de sistemas lineales en el sentido de mínimos cuadrados.

Definición 1.3.4. (El problema de mínimos cuadrados.) Dada $A \in \mathbb{R}^{m\times n}$ y un vector $b \in \mathbb{R}^m$, el problema de mínimos cuadrados consiste en encontrar un vector real $x \in \mathbb{R}^n$ que minimice $\|Ax - b\|_2$. La solución de este problema no es necesariamente única.

Se considera el espacio columna $R(A)$ de la definición 1.3.1. Para el sistema $Ax = b$ existen dos posibilidades:

- **El vector** $b \in R(A)$**:** En este caso, por la definición de $R(A)$, existe solución $x \in \mathbb{R}^n$.
- **El vector** $b \notin R(A)$**:** En el caso en que el sistema $Ax = b$ no tiene solución, entonces b no puede ser combinación de las columnas de A, es decir, $b \notin R(A)$. Han de buscarse entonces valores de x de modo que, puesto que Ax no puede ser b, esté lo más cerca posible de b, en el sentido de que la norma euclídea $\|Ax - b\|_2$ sea la menor posible. Tales x juegan el papel de *soluciones* en un sentido generalizado. Su determinación comprende dos etapas:

  1. Hallar la mejor aproximación $\hat{b}$ a b por elementos de $R(A)$, esto es, $\hat{b}$ es la proyección ortogonal de b sobre $R(A)$.
  2. Resolver el sistema $Ax = \hat{b}$ en sentido convencional. La solución x_{LS} de este sistema se llama solución en el sentido de mínimos cuadrados, pues por la caracterización de la proyección ortogonal, hace mínima las distancias $\|Ax - b\|$ con $x \in \mathbb{R}^n$.

  Desarrollemos el proceso de forma más detallada. Denotemos por A_i la i-ésima columna de A. Puesto que $\hat{b}$ es la proyección ortogonal de b sobre $R(A)$, debe cumplir dos condiciones:

  - $\hat{b} \in R(A)$, luego existe $\hat{x} \in \mathbb{R}^n$ tal que $\hat{b} = A\hat{x}$.
  - $b - \hat{b} \in R(A)^{\perp}$, de manera que $< b - A\hat{x}, A_j >= 0, \quad j = 1 \dots n$ o, equivalentemente,

  $$\langle A\hat{x}, A_j \rangle = \langle A_j, b \rangle, \quad j = 1 \dots n,$$

  de modo que obtenemos las ecuaciones normales

  $$A^T A\hat{x} = A^T b. \tag{1.16}$$

Definición 1.3.5. Sea $A \in \mathbb{R}^{m \times n}$, entonces se dice que A tiene rango completo si $rango(A) = min\{m, n\}$.

Lema 1.3.4. Sea $A \in \mathbb{R}^{m \times n}$, donde $m \geq n$. Entonces A tiene rango completo si y sólo si $A^T A \in \mathbb{R}^{n \times n}$ es no singular.

Demostración: $\boxed{\Longrightarrow}$ Probemos el contrarrecíproco. Para probar esta implicación, se tiene por hipótesis que $A^T A$ es singular. Ahora bien, razonemos por reducción al absurdo, y supongamos que A tiene rango completo. Como $A^T A$ es singular, si se plantea el sistema $A^T Ax = 0$, éste posee una solución $\tilde{x}$ no trivial. Multiplicando a la izquierda por $\tilde{x}^T$, se obtiene la expresión:

$$\tilde{x}^T A^T A\tilde{x} = 0 \implies (A\tilde{x})^T (A\tilde{x}) = 0 \implies \langle A\tilde{x}, A\tilde{x} \rangle = \|A\tilde{x}\|_2^2 = 0$$

Y por definición de norma, ocurre que $A\tilde{x} = 0$, luego se llega a contradicción, ya que entonces A no tiene rango completo.

$\boxed{\Longleftarrow}$ De nuevo, se razona por reducción al absurdo. Se tiene que $A^T A$ es no singular por hipótesis. Ahora bien, supongamos que A no tiene rango completo. En este caso, existe un $x \in \mathbb{R}^n / x \neq 0$ tal que $Ax = 0$. Multiplicando por A^T a la izquierda de la expresión anterior, se llega a que:

$$A^T Ax = 0, \ x \neq 0.$$

Por ello, se llega a contradicción, dado que A^TA es singular y se tenía por hipótesis que A^TA era no singular. □

Volviendo al sistema (1.16), hay que observar que, si bien la proyección $\hat{b}$ es única, el vector $\hat{x}$ tal que $\hat{b} = A\hat{x}$ no lo es necesariamente. Si la matriz A tiene rango completo, por el lema 1.3.4, el sistema (1.16) tiene solución única $x_{LS} = \hat{x}$. En el caso de rango deficiente, es decir, rango(A) $<$ mín($\{m, n\}$), entonces hay infinitos $\hat{x}$ verificando (1.16), pues entonces A^TA es singular y sumar cualquier vector del núcleo a una solución del sistema nos da otra solución. Es decir, el conjunto de soluciones forma una variedad lineal afín paralela al núcleo. Se puede entonces demostrar que hay un único vector en esta variedad con norma euclídea mínima [6]. Tal vector x_{LS} recibe el nombre de solución de $Ax = b$ en el sentido de mínimos cuadrados.

Teorema 1.3.7. (Existencia y unicidad del problema de mínimos cuadrados.) Dada una matriz $A \in \mathbb{R}^{m\times n}$, donde $m \geq n$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$. Entonces x es la solución de mínimos cuadrados del sistema $Ax = b$ si y sólo si satisface las ecuaciones normales dadas por $A^TAx = A^Tb$. Además, dicha solución es única si y sólo si A tiene rango completo.

Demostración. Véase [8].

Para resolver problemas de mínimos cuadrados sobredeterminados, se recurre principalmente a las siguientes técnicas:

- A través de las ecuaciones normales (1.16).
- Mediante la factorización QR.
- Mediante la SVD.

Se expone, a continuación, cómo proceder en el caso que involucra a la SVD. Para conocer los procedimientos de las otras dos técnicas, véanse las referencias [6, 8].

Supongamos que $A \in \mathbb{R}^{m\times n}$, $m \geq n$, es de rango completo. Por el lema 1.3.4, A^TA es invertible, por tanto, el producto $(A^TA)^{-1}A^T$ está bien definido.

Definición 1.3.6. En las condiciones anteriores, la matriz $A^\dagger = (A^TA)^{-1}A^T$ se denomina *pseudoinversa* o la *inversa generalizada de Moore-Penrose* de A.

Si $m < n$ y el rango de A es m, entonces A^T tiene dimensiones $n \times m$, con $n > m$, por lo que ya se pueden aplicar los resultados presentados previamente, y la pseudoinversa se define para A^T. Se cumple que:

$$(A^T)^\dagger = (((A^T)^T)(A^T))^{-1}((A^T)^T) = (AA^T)^{-1}A. \tag{1.17}$$

Al tomar la traspuesta en (1.17), obtenemos:

$$A^\dagger = A^T((AA^T)^{-1})^T = A^T(AA^T)^{-1}.$$

Nótese que $AA^T \in \mathbb{R}^{m\times m}$ y es no singular dado que A tiene rango completo, por (1.3.4). En este caso, se define $A^\dagger = A^T(AA^T)^{-1}$.

Resolución a través de la SVD:

De las definiciones anteriores y la discusión de las ecuaciones normales, es claro que la solución de $Ax = b$ en el sentido de mínimos cuadrados cuando A tiene rango completo con $m \geq n$ es $x_{LS} = A^{\dagger}b$.

Sea $A = \hat{U}\hat{\Sigma}\hat{V}^T$ la SVD reducida de $A \in \mathbb{R}^{m\times n}$ (*cf.* sección 1.2.1). Entonces,

$$\begin{cases} A^TA = (\hat{U}\hat{\Sigma}\hat{V}^T)^T(\hat{U}\hat{\Sigma}\hat{V}^T) = (\hat{V}\hat{\Sigma}^T\hat{U}^T)(\hat{U}\hat{\Sigma}\hat{V}^T) = (\hat{V}\hat{\Sigma})(\hat{\Sigma}\hat{V}^T), \\ A^Tb = (\hat{V}\hat{\Sigma})\hat{U}^Tb. \end{cases}$$

Sustituyendo en las ecuaciones normales (1.16), se tiene:

$$(\hat{V}\hat{\Sigma})(\hat{\Sigma}\hat{V}^T)x = A^Tb = (\hat{V}\hat{\Sigma})\hat{U}^Tb. \tag{1.18}$$

Se recuerda que $\hat{V}$ es una matriz ortogonal, y que, como $\hat{U}$, $\hat{V}$ y $\hat{\Sigma}$ fueron obtenidas a través de un proceso de SVD reducida, entonces $\hat{\Sigma}$ es una matriz diagonal que solamente tiene entradas positivas, al ser A de rango completo y, por tanto, A^TA invertible. En consecuencia, la matriz producto $\hat{V}\hat{\Sigma}$ es invertible, por lo que, multiplicando (1.18) a ambos lados por $(\hat{V}\hat{\Sigma})^{-1}$, se obtiene:

$$(\hat{\Sigma}\hat{V}^T)x = \hat{U}^Tb.$$

Es decir, el problema de mínimos cuadrados original se ha reducido a resolver dos sistemas lineales sencillos.

- En primer lugar, se resuelve el sistema $\hat{\Sigma}y = \hat{U}^Tb$. Como $\hat{\Sigma}$ es una matriz diagonal, la resolución del mismo es inmediata.
- En segundo lugar, se resuelve el sistema dado por $\hat{V}^Tx = y$. Ahora bien, este proceso se agiliza teniendo en cuenta que $\hat{V}$ es una matriz ortogonal, por lo cuál, la solución se obtiene como $x = \hat{V}y$, ya que $\hat{V}^{-1} = \hat{V}^T$.

Veamos cómo podemos utilizar la SVD para calcular la solución de $Ax = b$ en el sentido de mínimos cuadrados, x_{LS}, en el caso de rango deficiente.

Proposición 1.3.1. Sea $A \in \mathbb{R}^{m\times n}$, $m \geq n$, con rango $r < n$. Escribimos la SVD reducida de A en la forma

$$A = [U_1,\ U_2]\begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix}[V_1,\ V_2]^T = U_1\Sigma_1V_1^T, \tag{1.19}$$

donde $\Sigma_1 \in \mathbb{R}^{r\times r}$ es no singular y U_1 y V_1 tienen r columnas. Entonces:

1. Todas las soluciones $\hat{x}$ de (1.16) son de la forma

   $$\hat{x} = V_1\Sigma_1^{-1}U_1^Tb + V_2z,$$

   con z un vector arbitrario.

2. La solución x_{LS} con norma euclídea mínima se obtiene cuando $z = 0$:

   $$x_{LS} = V_1\Sigma_1^{-1}U_1^Tb. \tag{1.20}$$

Demostración. Tomemos $\tilde{U} \in \mathbb{R}^{m\times(m-n)}$ tal que $[U_1,\ U_2,\ \tilde{U}] \in \mathbb{R}^{m\times m}$ es ortogonal. Entonces,

$$\|Ax-b\|_2^2 = \|[U_1,\ U_2,\ \tilde{U}]^T(Ax-b)\|_2^2 = \left\| \begin{bmatrix} U_1^T \\ U_2^T \\ \tilde{U}^T \end{bmatrix} (U_1\Sigma_1 V_1^T x - b) \right\|_2^2$$

$$= \left\| \begin{bmatrix} \Sigma_1 V_1^T x - U_1^T b \\ -U_2^T b \\ -\tilde{U}^T b \end{bmatrix} \right\|_2^2 = \|\Sigma_1 V_1^T x - U_1^T b\|_2^2 + \|U_2^T b\|_2^2 + \|\tilde{U}^T b\|_2^2. \tag{1.21}$$

1. De (1.21), es claro que $\|Ax-b\|_2$ es mínimo cuando $\Sigma_1 V_1^T x = U_1^T b$, es decir,

   $$\hat{x} = V_1\Sigma_1^{-1}U_1^T b + w,$$

   con $w \in \mathrm{Ker}(V_1^T)$. De la ortogonalidad entre las columnas de $V = [V_1, V_2]$, entonces w es de la forma $w = V_2 z$ con $z \in \mathbb{R}^{n-r}$ arbitrario.

2. Como las columnas de V_1 y V_2 son ortogonales entre sí,

   $$\|\hat{x}\|_2^2 = \|V_1\Sigma_1^{-1}U_1^T b\|_2^2 + \|V_2 z\|_2^2,$$

   que se hace mínimo cuando $z = 0$. □

Definición 1.3.7. En las condiciones de la proposición 1.3.1, se define la pseudoinversa de A como la matriz

$$A^\dagger = V_1\Sigma_1^{-1}U_1^T.$$

Entonces, la solución del problema de mínimos cuadrados (1.20) se escribe $x_{LS} = A^\dagger b$, y la SVD proporciona un método para su cálculo.

1.3.4. Aproximaciones de rango bajo de una matriz

Una matriz de rango bajo es aquella cuyo rango (el número máximo de columnas o filas linealmente independientes) es significativamente menor que sus dimensiones totales. Las aproximaciones de rango bajo buscan representar una matriz de grandes dimensiones mediante una versión simplificada que conserva las propiedades y la estructura esenciales de la matriz original. Se presenta la definición rigurosa a continuación:

Definición 1.3.8. Sea $A \in \mathbb{R}^{m\times n}$ con rango r. Se dice que A es una matriz de bajo rango si

$$r \ll \text{mín}\,(m,n)\,.$$

Definición 1.3.9. Sea $A \in \mathbb{R}^{m\times n}$ con rango $r > 0$ y sea $A = U\Sigma V^T$ su descomposición en valores singulares. Definimos para $k = 1,\dots,r-1$, $A_k = U\Sigma_k V^T$ donde Σ_k está dada por

$$\Sigma_k = \left(\begin{array}{cccc|c} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma_k & 0 \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right) \in \mathbb{R}^{m\times n}. \tag{1.22}$$

En lo que sigue se supone que los valores singulares están dispuestos en orden decreciente.

Teorema 1.3.8. La matriz A_k tiene un rango igual a k y cumple con la relación $\|A - A_k\|_2 = \text{mín}\{\|A - B\|_2 \,|\, \text{rango}(B) = k\} = \sigma_{k+1}$. En otras palabras, entre todas las matrices con rango k, A_k es la que se encuentra más cerca de A si se mide la distancia entre matrices con la norma euclídea.

Demostración. Por la descomposición en valores singulares de A y A_k se tiene que

$$A - A_k = U\Sigma V^T - U\Sigma_k V^T = U(\Sigma V^T - \Sigma_k V^T) = U(\Sigma - \Sigma_k)V^T.$$

Teniendo en cuenta la estructura de Σ, así como la forma de Σ_k dada en (1.22), se tiene que

$$\Sigma - \Sigma_k = \left(\begin{array}{cccc|c} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \sigma_{k+1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma_r & 0 \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right) \in \mathbb{R}^{m\times n},$$

donde σ_{k+1} es el mayor valor singular de $A - A_k$, por lo que, por el lema 1.3.1,

$$\|A - A_k\|_2 = \sigma_{k+1}.$$

Veamos ahora que, para cualquier otra matriz B de rango k, $\|A - B\|_2 \geq \sigma_{k+1}$. Dada una matriz $B \in \mathbb{R}^{m\times n}$ de rango k, el núcleo de B tiene dimensión $n - k$. Si $\{v_1, \ldots, v_n\}$ denotan las columnas de V, el subespacio vectorial generado por $\{v_1, \ldots, v_{k+1}\}$ tiene dimensión $k + 1$. Como tanto el núcleo de B como $span\{v_1, \ldots, v_{k+1}\}$ son subespacios de $\mathbb{R}^n$, y la suma de sus dimensiones es mayor que n, la intersección de ambos subespacios es no nula.

Sea ahora $x \in \ker(B) \cap span\{v_1, \ldots, v_{k+1}\}$, y supongamos, sin pérdida de generalidad, que $\|x\|_2 = 1$. Como $x \in span\{v_1, \ldots, v_{k+1}\}$, existen escalares $\alpha_1, \ldots, \alpha_{k+1}$ tales que $x = \alpha_1 v_1 + \cdots + \alpha_{k+1} v_{k+1}$. Por la ortonormalidad de $v_1, \ldots, v_{k+1}$, ocurre que $\|x\|_2 = |\alpha_1|^2 + \cdots + |\alpha_{k+1}|^2 = 1$ y, como $x \in \ker(B)$, $Bx = 0$. Además,

$$(A - B)x = Ax - Bx = Ax = \sum_{i=1}^{k+1} \alpha_i A v_i = \sum_{i=1}^{k+1} \sigma_i \alpha_i u_i,$$

donde $u_1, \ldots, u_{k+1}$ son las columnas de la matriz U, también ortonormales. Se tiene entonces

$$\|(A - B)x\|_2 = \left(\sum_{i=1}^{k+1} |\sigma_i \alpha_i|^2\right)^{1/2} \geq \sigma_{k+1} \left(\sum_{i=1}^{k+1} |\alpha_i|^2\right)^{1/2} = \sigma_{k+1}.$$

Por tanto,

$$\|A - B\|_2 \geq \|(A - B)x\|_2 \geq \sigma_{k+1}.$$

□

Lema 1.3.5. Sea $A \in \mathbb{R}^{m\times n}$ de rango n y sean $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ sus valores singulares. Si $B \in \mathbb{R}^{m\times n}$ satisface que $\|A - B\|_2 < \sigma_n$, entonces el rango de B también es n.

Demostración. Supongamos rango$(B) = k < n$. En ese caso,

$$\|A - B\|_2 \geq \|A - A_k\|_2 = \sigma_{k+1} \geq \sigma_n,$$

lo cual contradice el que $\|A - B\|_2 < \sigma_n$. □

Del lema 1.3.5 se deduce que las matrices que están suficientemente próximas respecto de la norma euclídea, tienen el mismo rango.

Aproximaciones de Rango Bajo a partir de la SVD

Si se desea aproximar de la mejor manera una matriz $A \in \mathbb{R}^{m\times n}$ por una matriz de rango bajo $k \ll \text{mín}\{m, n\}$, se puede utilizar la descomposición en valores singulares.

Esta descomposición permite expresar A como una suma de productos de vectores singulares y sus valores singulares correspondientes. De hecho, se puede escribir A como una suma de matrices de rango uno ponderadas por los valores singulares:

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

donde $r = \text{mín}\{m, n\}$, σ_i son los valores singulares no nulos, y $\mathbf{u}_i$ y $\mathbf{v}_i$ son los vectores singulares izquierdos y derechos, respectivamente.

Cada término $\sigma_i \mathbf{u}_i \mathbf{v}_i^T$ en la suma es una matriz de rango uno. Esto se debe a que $\mathbf{u}_i \mathbf{v}_i^T$ es un producto exterior de un vector columna $\mathbf{u}_i$ y un vector fila $\mathbf{v}_i^T$. El producto exterior de dos vectores da como resultado una matriz donde cada fila es un múltiplo del vector fila $\mathbf{v}_i^T$, lo cual implica que todas las filas (o columnas) son linealmente dependientes. Por lo tanto, la matriz $\mathbf{u}_i \mathbf{v}_i^T$ tiene rango uno.

Pues bien, como la SVD expresa una matriz A como una suma de matrices de rango uno ponderadas por los valores singulares correspondientes, una idea natural es mantener solo los primeros k términos del lado derecho de la ecuación. Se presenta a continuación, el proceso a seguir, expuesto en [19].

Proceso de Aproximación

Dada una matriz A y un rango objetivo k, la aproximación de rango k propuesta es:

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \tag{1.23}$$

donde se asume que los valores singulares están ordenados ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\text{mín}\{m,n\}} \geq 0$), y u_i y v_i denotan los i-ésimos vectores singulares izquierdos y derechos, respectivamente. Siendo la suma de k matrices de rango uno, A_k tiene rango de, a lo sumo, k.

Almacenar las matrices en el lado derecho de (1.23) requiere un espacio de $\mathcal{O}(k(m+n))$, en contraste con el espacio $\mathcal{O}(mn)$ requerido para almacenar la matriz original A. Esto representa una gran ventaja cuando k es relativamente pequeño y m y n son relativamente grandes.

Similar al resultado sobre la mejor aproximación de una matriz por matrices de un rango específico utilizando la norma euclídea para medir la distancia entre matrices, se puede formular un resultado equivalente empleando la norma de Frobenius para medir dicha distancia. Este resultado, junto con el Teorema 1.3.8, constituyen el llamado *Teorema de Eckart-Young*.

Teorema 1.3.9. Sea $A \in \mathbb{R}^{m\times n}$ de rango $r > 0$ y sea $A = U\Sigma V^T$ su descomposición en valores singulares. Definimos para $k = 1, \dots, r-1$, $A_k = U\Sigma_k V^T$, donde Σ_k está definida como en (1.22). Se verifica entonces que:

$$\|A - A_k\|_F = \text{mín}\{\|A - B\|_F \mid \text{rango}(B) = k\} = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}.$$

Demostración. Como vimos en la demostración del Teorema 1.3.8, teniendo en cuenta la descomposición en valores singulares de A y A_k, se tiene

$$A - A_k = U(\Sigma - \Sigma_k)V^T,$$

Por las propiedades de la norma de Frobenius, se tiene que

$$\|A - A_k\|_F = \|\Sigma - \Sigma_k\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}.$$

Veamos ahora que si B es otra matriz de rango k, entonces

$$\|A - B\|_F \geq \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}.$$

Utilizando la descomposición en valores singulares de B, se puede expresar B como

$$B = \sum_{i=1}^{k} x_i y_i^T,$$

puesto que cualquier matriz de rango k puede ser expresada como una suma de k matrices de rango uno y donde los vectores y_i se pueden elegir ortonormales. Los vectores x_i son ortogonales pero de norma euclídea igual a $\tilde{\sigma}_i$, siendo $\tilde{\sigma}_1, \dots, \tilde{\sigma}_k$ los valores singulares de B. Se verifica entonces que

$$\begin{aligned}
\|A - B\|_F^2 &= \text{tr}\left((A - \sum_{i=1}^{k} x_i y_i^T)(A - \sum_{i=1}^{k} x_i y_i^T)^T\right) \\
&= \text{tr}\left(AA^T + \sum_{i=1}^{k}(x_i - Ay_i)(x_i - Ay_i)^T - \sum_{i=1}^{k} Ay_i y_i^T A^T\right) \\
&= \|A\|_F^2 + \sum_{i=1}^{k} \|x_i - Ay_i\|_F^2 - \sum_{i=1}^{k} \|Ay_i\|_F^2.
\end{aligned}$$

Por ser el segundo sumando positivo o nulo, basta ver entonces que

$$\sum_{i=1}^{k} \|Ay_i\|_F^2 \leq \sum_{i=1}^{k} \sigma_i^2.$$

De esta forma se tendrá que

$$\|A-B\|_F^2 = \|A\|_F^2 + \sum_{i=1}^{k}\|x_i - Ay_i\|_F^2 - \sum_{i=1}^{k}\|Ay_i\|_F^2$$
$$\geq \sum_{i=1}^{r}\sigma_i^2 + \sum_{i=1}^{k}\|x_i - Ay_i\|_F^2 - \sum_{i=1}^{k}\sigma_i^2 \geq \sum_{i=k+1}^{r}\sigma_i^2 = \|A-A_k\|_F^2.$$

Teniendo en cuenta la SVD de la matriz A y particionando adecuadamente las matrices V y Σ de modo que $V = (V_1 \mid V_2)$ con V_1 formada por las k primeras columnas de V, se verifica que

$$\|Ay_i\|_F^2 = \|U\Sigma V^T y_i\|_F^2 = \|\Sigma V^T y_i\|_F^2 = \|\Sigma_1 V_1^T y_i\|_F^2 + \|\Sigma_2 V_2^T y_i\|_F^2.$$

Manipulando en esta expresión y teniendo en cuenta que $\sigma_k^2\|V^T y_i\|_F^2 = \sigma_k^2\|V_1^T y_i\|_F^2 + \sigma_k^2\|V_2^T y_i\|_F^2$, se obtiene que

$$\|Ay_i\|_F^2 = \sigma_k^2\|V^T y_i\|_F^2 + \left(\|\Sigma_1 V_1^T y_i\|_F^2 - \sigma_k^2\|V_1^T y_i\|_F^2\right) + \left(\|\Sigma_2 V_2^T y_i\|_F^2 - \sigma_k^2\|V_2^T y_i\|_F^2\right).$$

Como los elementos diagonales de Σ_2 son menores o iguales que σ_k, entonces

$$\left(\|\Sigma_2 V_2^T y_i\|_F^2 - \sigma_k^2\|V_2^T y_i\|_F^2\right) \leq 0.$$

Por otra parte, como V es ortogonal y los vectores y_i son ortonormales, $\|V^T y_i\|_F^2 = 1$. Por lo tanto,

$$\|Ay_i\|_F^2 \leq \sigma_k^2 + \left(\|\Sigma_1 V_1 y_i\|_F^2 - \sigma_k\|V_1^T y_i\|_F^2\right) \leq \sigma_k^2 + \sum_{j=1}^{k}(\sigma_j^2 - \sigma_k^2)|v_j^T y_i|^2.$$

En consecuencia,

$$\sum_{i=1}^{k}\|Ay_i\|_F^2 \leq k\sigma_k^2 + \sum_{i=1}^{k}\left(\sum_{j=1}^{k}(\sigma_j^2 - \sigma_k^2)|v_j^T y_i|^2\right)$$
$$= \sum_{j=1}^{k}\left(\sigma_k^2 + (\sigma_j^2 - \sigma_k^2)\sum_{i=1}^{k}|v_j^T y_i|^2\right) \leq \sum_{j=1}^{k}(\sigma_k^2 + (\sigma_j^2 - \sigma_k^2)\|v_j\|_F^2) = \sum_{j=1}^{k}\sigma_j^2. \quad \square$$

A la hora de realizar una aproximación de bajo rango de una matriz A, la elección de k debería guiarse por los valores singulares de A: si los primeros valores son grandes y el resto son pequeños, una solución obvia es tomar k igual al número de valores grandes. En condiciones ideales, se tomaría k lo más pequeño posible sujeto a obtener una aproximación útil.

En definitiva, la aproximación de rango bajo es una técnica de gran relevancia y que permite simplificar matrices preservando su información esencial. El porqué de su importancia se indica a continuación:

- **Reducción de Ruido:** Al eliminar la información menos relevante de la matriz, se puede reducir el ruido o las irregularidades, haciendo más visibles los patrones importantes y consistentes en los datos.
- **Reducción de Dimensionalidad:** Al simplificar los datos y reducir la cantidad de dimensiones (o variables), se puede hacer que el procesamiento de la información sea más rápido y se necesite menos espacio para almacenarla, sin perder información crucial.

- **Extracción de Características:** Al identificar y conservar las características más significativas de los datos, se puede mejorar el análisis y la creación de modelos, ya que se trabaja con la información más relevante y útil del conjunto de datos.

Para ilustrar gráficamente el concepto de aproximaciones de rango bajo de una matriz mediante la SVD, se ha propuesto un código MATLAB en el Apéndice A.3 que ejecuta varios pasos clave en este proceso. El propósito del código es ofrecer una representación visual clara de cómo se puede aproximar una matriz A por matrices de menor rango que conservan sus características esenciales.

El proceso comienza con la definición de una matriz A, para la cual se ha elegido una de tamaño 10×10 y generada de forma aleatoria.

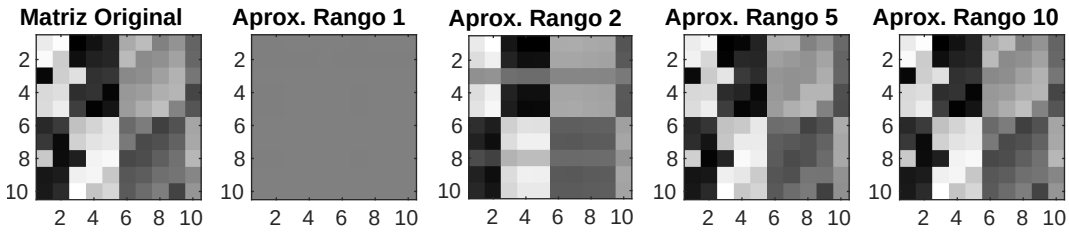


Figura 1.8: Ilustración de las aproximaciones de rango bajo.

El código procede luego a calcular aproximaciones de rango bajo de A, utilizando diferentes valores de k, que representan el número de componentes singulares a conservar en la aproximación. Para cada valor de k, se construye una aproximación A_k manteniendo solo los k valores singulares más grandes y sus vectores singulares asociados, lo que resulta en matrices de menor rango que se aproximan a A.

Cada aproximación de rango bajo, junto con la matriz original A, se visualiza en la figura 1.8 , utilizando un mapa de colores en escala de grises para facilitar la comparación visual. Este paso ilustra de manera efectiva cómo, a medida que aumenta k, la aproximación A_k se vuelve visualmente más cercana a A, destacando cómo se pueden mantener las características esenciales de A para valores de k relativamente bajos.

Capítulo 2

Implementación de la SVD

En este capítulo, desarrollaremos varios algoritmos para la implementación de la SVD reducida. Su origen se encuentra en la relación entre la SVD de una matriz A y la diagonalización de las matrices simétricas A^TA y AA^T, descrita en el capítulo 1, de manera que aquellos algoritmos para el cálculo numérico de autovalores y autovectores de una matriz simétrica pueden transformarse en algoritmos para la SVD.

Los métodos para resolver numéricamente el problema de autovalores de una matriz simétrica pueden, grosso modo, dividirse en dos grupos. Por un lado, están aquéllos basados en una reducción previa de la matriz, con los pasos siguientes.

1. Se reduce A a forma tridiagonal con una matriz Q_1 ortogonal, de manera que $A = Q_1TQ_1^T$.
2. Se busca una diagonalización
$$T = Q_2\Lambda Q_2^T,$$
con Λ diagonal y Q_2 una matriz ortogonal cuyas columnas son autovectores de los autovalores presentes en la diagonal de Λ.
3. Se combinan las dos descomposiciones para obtener
$$A = Q\Lambda Q^T, \quad Q = Q_1Q_2,$$
donde las columnas de Q son autovectores de A.

Por otro lado, aquellos algoritmos para la SVD reducida de una matriz G que pueden asociarse a esta familia de métodos tienen la estructura siguiente:

1. Se reduce G a forma bidiagonal B con matrices ortogonales $U_1,\ V_1$, de manera que $G = U_1BV_1^T$.
2. Se busca la SVD de $B = U_2\Sigma V_2^T$.
3. Se combinan ambas factorizaciones de manera que $G = U\Sigma V^T$ con $U = U_1U_2$ y $V = V_1V_2$, las matrices de los vectores singulares por la izquierda y por la derecha, respectivamente.

De este modo, esta familia de métodos para el cómputo de la SVD está basada en una reducción previa a forma bidiagonal. Es entonces necesario establecer una relación entre el problema de encontrar la SVD de una matriz bidiagonal B con el problema de autovalores de una matriz tridiagonal y simétrica T. Esto viene dado por el siguiente lema, véase [6].

Lema 2.0.1. Sea $B \in \mathbb{R}^{n\times n}$ bidiagonal, con diagonal $a_1, \ldots, a_n$ y superdiagonal $b_1, \ldots, b_{n-1}$. Existen tres formas de convertir el problema de calcular la SVD de B al problema de encontrar los autovalores y autovectores de una matriz tridiagonal simétrica:

1. Sea $A = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}$. 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 dimensión $2n \times 2n$. Entonces, $T_{ps} \equiv P^T AP$ es una matriz tridiagonal simétrica. Se puede demostrar que T_{ps} tiene ceros en su diagonal principal, y su superdiagonal y subdiagonal son $a_1, b_1, a_2, b_2, \ldots, b_{n-1}, a_n$.

   Si $T_{ps}x_i = \alpha_i x_i$ es un eigenpar de T_{ps}, con x_i un vector unitario, entonces $\alpha_i = \pm\sigma_i$, donde σ_i es un valor singular de B, y
   $$Px_i = \frac{1}{\sqrt{2}} \begin{bmatrix} v_i \\ \pm u_i \end{bmatrix},$$
   donde u_i y v_i son los vectores singulares izquierdo y derecho de B, respectivamente.

2. Sea $T_{BB^T} \equiv BB^T$. Entonces, T_{BB^T} es matriz tridiagonal simétrica con diagonal $a_1^2 + b_1^2, a_2^2 + b_2^2, \ldots, a_{n-1}^2 + b_{n-1}^2, a_n^2$, y superdiagonal y subdiagonal $a_2b_1, a_3b_2, \ldots, a_nb_{n-1}$. Los valores singulares de B son las raíces cuadradas de los valores propios de T_{BB^T}, y los vectores singulares izquierdos de B son los vectores propios de T_{BB^T}. T_{BB^T} no contiene información sobre los vectores singulares derechos de B.

3. Sea $T_{B^TB} \equiv B^TB$. Entonces, T_{B^TB} es matriz tridiagonal simétrica con diagonal $a_1^2, a_2^2 + b_1^2, a_3^2 + b_2^2, \ldots, a_n^2 + b_{n-1}^2$, y superdiagonal y subdiagonal $a_1b_1, a_2b_2, \ldots, a_nb_n$. Los valores singulares de B son las raíces cuadradas de los valores propios de T_{B^TB}, y los vectores singulares derechos de B son los vectores propios de T_{B^TB}. T_{B^TB} no contiene información sobre los vectores singulares izquierdos de B.

La segunda familia de métodos para resolver numéricamente el problema de autovalores de una matriz real simétrica A no utiliza una reducción previa a forma tridiagonal. Los modelos clásicos son la iteración basada en el cociente de Rayleigh y el método de Jacobi, [16].

En el caso del segundo, el método parte de la matriz original A (normalmente densa) y genera una sucesión de matrices utilizando rotaciones, que converge eventualmente a una matriz diagonal. Este procedimiento puede adaptarse al problema de calcular la SVD de una matriz densa G, aplicándolo a GG^T o G^TG, pero sin formar explícitamente ninguna de ellas. El algoritmo básico puede perder cierta estabilidad en general, aunque bajo ciertas condiciones sobre G, la precisión de la SVD resultante es relativamente alta, [6]. Dicha estabilidad viene determinada por el siguiente resultado general.

Teorema 2.0.1. Sea $A \in \mathbb{R}^{m\times n}$ y $B = A + E \in \mathbb{R}^{m\times n}$, con $m \geq n$. Sean $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$ y $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq ... \geq \tilde{\sigma}_n$, respectivamente, los valores singulares de A y B. Entonces, para cada $i \in \{1, 2 \ldots n\}$, se cumple que
$$|\sigma_i - \tilde{\sigma}_i| \leq \|E\|_2.$$

En este capítulo, describiremos un método de cada familia. Primero, se presenta **el método de Jacobi unilateral**, que se basa en rotaciones de Jacobi, que introduciremos a continuación, para calcular los valores propios de matrices simétricas. Luego se aborda **el algoritmo de Demmel y Kahan**, que utiliza reflexiones de Householder para transformar cualquier matriz $A \in \mathbb{R}^{m\times n}$ en una matriz bidiagonal. Esta matriz bidiagonal se reduce a una matriz diagonal,

que contiene los valores singulares, utilizando la técnica de *bulge-chasing*.

La sección final presenta un **estudio comparativo** entre ambos métodos. Aquí, se analiza de manera comparativa su eficiencia, complejidad computacional y aplicabilidad a diferentes tipos de matrices. Se ofrecen, así, perspectivas sobre la selección del método más adecuado en función de las necesidades específicas del problema.

2.1. Método de Jacobi

El método de Jacobi es un algoritmo desarrollado para aproximar los valores y vectores propios de matrices simétricas reales. El método transforma la matriz simétrica $A \in \mathbb{R}^{n\times n}$ directamente en una matriz diagonal utilizando transformaciones de similitud ortogonales. La estrategia utilizada en el algoritmo de Jacobi es desarrollar una iteración que haga que la suma de los cuadrados de los elementos fuera de la diagonal converja a 0, obteniendo así una matriz diagonal de valores propios.

El progreso de la iteración puede medirse a partir de la cantidad

$$f(A) = \sqrt{\sum_{\substack{i,\ j=1\\ j\neq i}}^{n} a_{ij}^2} = \sqrt{\|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2}\ ,$$

siendo $\|A\|_F$ la norma de Frobenius de $A \in \mathbb{R}^{n\times n}$. Observemos que $f(A)^2$ es la suma de los cuadrados de los elementos no diagonales de A, de manera que A es diagonal si y sólo si $f(A) = 0$.

El método de Jacobi genera una sucesión de matrices ortogonales $\{J_k\}_{k\geq 0}$ tales que si

$$\begin{cases} A_0 = A, \\ A_k = J_{k-1}^T A_{k-1} J_{k-1}, \quad k \geq 1, \end{cases} \tag{2.1}$$

entonces $\lim_{k\to\infty} \mathrm{f}(A_k) = 0$. Dado que $A = A_0$ es simétrica y $\left(J_{k-1}^T A_{k-1} J_{k-1}\right)^T = J_{k-1}^T A_{k-1} J_{k-1}$, ocurre que A_k es simétrica. Cada matriz ortogonal J_k es una rotación de Givens modificada ligeramente, que denotaremos con el nombre de rotación de Jacobi. Las rotaciones de Givens usuales son de la forma

$$J(i,j,c,s) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix},$$

donde los números c y s son elegidos de tal manera que el producto $J(i,j,c,s)A$, con $i \neq j$, sobre una matriz $A = (a_{kl})_{k,l=1}^n$, haga que el elemento a_{ji} se vuelva 0.

Para el método de Jacobi, elegimos c y s de modo que si

$$\bar{A} = J(i,j,c,s)^T A J(i,j,c,s) = (\bar{a}_{kl})_{k,l=1}^n, \tag{2.2}$$

entonces $\bar{a}_{ij} = \bar{a}_{ji} = 0$. Las fórmulas que se presentan a continuación son aplicables en cada paso k de (2.1) con $A = A_{k-1}$, $\bar{A} = A_k$, $J_{k-1} = J(i,j,c,s)$. El valor de los índices i, j con $i \neq j$, se especifica más adelante y asegura la convegencia del método. Al formar el producto (2.2), aparecen las fórmulas de la tabla 2.1.

Fórmulas de Iteración de Jacobi
$\bar{a}_{kl} = a_{kl},\ k,l \neq i,j$
$\bar{a}_{ik} = \bar{a}_{ki} = ca_{ik} - sa_{jk},\ k \neq i,j$
$\bar{a}_{jk} = \bar{a}_{kj} = sa_{ik} + ca_{jk},\ k \neq i,j$
$\bar{a}_{ij} = \bar{a}_{ji} = (c^2 - s^2)a_{ij} + cs(a_{ii} - a_{jj})$
$\bar{a}_{ii} = c^2 a_{ii} - 2csa_{ij} + s^2 a_{jj}$
$\bar{a}_{jj} = s^2 a_{ii} + 2csa_{ij} + c^2 a_{jj}$

Tabla 2.1: Fórmulas para la Iteración de Jacobi

Fijamos $i \neq j$ y buscamos que $\bar{a}_{ij} = \bar{a}_{ji} = 0$. Entonces,

$$\left(c^2 - s^2\right) a_{ij} + cs\left(a_{ii} - a_{jj}\right) = 0. \tag{2.3}$$

Si $a_{ij} = 0$, tomamos $c = 1,\ s = 0$. En otro caso, reescribimos (2.3) como

$$\frac{c^2 - s^2}{cs} = \frac{a_{jj} - a_{ii}}{a_{ij}}. \tag{2.4}$$

Dado que $J(i,j,c,s)$ es ortogonal, debemos tener $c^2 + s^2 = 1$, y que $c = \cos\theta$, $s = \sin\theta$ para algún θ. Sustituyendo estas relaciones en (2.4), obtenemos

$$\frac{\cos^2\theta - \sin^2\theta}{\cos\theta\sin\theta} = \frac{a_{jj} - a_{ii}}{a_{ij}}. \tag{2.5}$$

Dado que $\cos 2\theta = \cos^2\theta - \sin^2\theta$, y $\sin 2\theta = 2\sin\theta\cos\theta$, la ecuación (2.5) se puede escribir como

$$\cot 2\theta = \frac{1}{2}\left(\frac{a_{jj} - a_{ii}}{a_{ij}}\right). \tag{2.6}$$

Por otro lado,

$$\begin{aligned}
\tan^2\theta + 2\tan\theta\cot 2\theta - 1 &= \frac{\sin^2\theta}{\cos^2\theta} + 2\left(\frac{\sin\theta\cos 2\theta}{\cos\theta\sin 2\theta}\right) - 1 \\
&= \frac{\sin^2\theta}{\cos^2\theta} + 2\left(\frac{\sin\theta(\cos^2\theta - \sin^2\theta)}{2\sin\theta\cos^2\theta}\right) - 1 \\
&= \frac{\sin^2\theta}{\cos^2\theta} + \frac{\cos^2\theta - \sin^2\theta}{\cos^2\theta} - 1 = 0.
\end{aligned}$$

Nótese que $\tan\theta = \frac{s}{c}$. Ahora, usamos la identidad $\tan^2\theta + 2\tan\theta\cot 2\theta - 1 = 0$ con las ecuaciones (2.4) y (2.6) para obtener

$$\frac{s^2}{c^2} + 2\cdot\frac{s}{c}\cdot\frac{1}{2}\left(\frac{a_{jj} - a_{ii}}{a_{ij}}\right) - 1 = \frac{s^2}{c^2} + \frac{s}{c}\left(\frac{c^2 - s^2}{cs}\right) - 1 = 1 - 1 = 0. \tag{2.7}$$

Sea $t = \frac{s}{c}$; podemos escribir (2.7) como

$$t^2 + 2\tau t - 1 = 0, \tag{2.8}$$

donde $\tau = \frac{1}{2}\left(\frac{a_{jj}-a_{ii}}{a_{ij}}\right)$. Resolviendo la ecuación cuadrática (2.8), obtenemos dos raíces:

$$t = -\tau \pm \sqrt{\tau^2 + 1}.$$

Las soluciones se pueden reescribir como sigue:

$$t_1 = \left(-\tau + \sqrt{\tau^2+1}\right)\left(\frac{-\tau - \sqrt{\tau^2+1}}{-\tau - \sqrt{\tau^2+1}}\right) = \frac{-1}{-\tau - \sqrt{\tau^2+1}} = \frac{1}{\tau + \sqrt{\tau^2+1}},$$
$$t_2 = \left(-\tau - \sqrt{\tau^2+1}\right)\left(\frac{-\tau + \sqrt{\tau^2+1}}{-\tau + \sqrt{\tau^2+1}}\right) = \frac{-1}{-\tau + \sqrt{\tau^2+1}}.$$

Con el objetivo de evitar errores de cancelación en el denominador, cuando $\tau < 0$, elegiremos t_2, y cuando $\tau > 0$, elegiremos t_1, con lo que estaríamos añadiendo dos números positivos en el denominador.

Usando t, ahora debemos encontrar valores de c y s. Cuando encontremos un valor apropiado para c, simplemente despejaremos s mediante la expresión $s = ct$. Ahora, $1 + \tan^2\theta = 1 + t^2 = \sec^2\theta = \frac{1}{\cos^2\theta} = \frac{1}{c^2}$, por lo que

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

Cálculo de c y s en el Método de Jacobi

$$\tau = \frac{a_{jj} - a_{ii}}{2a_{ij}}$$

$$t = \begin{cases} \frac{1}{\tau + \sqrt{\tau^2+1}} & \text{if } \tau \geq 0 \\ \frac{1}{-\tau + \sqrt{\tau^2+1}} & \text{if } \tau < 0 \end{cases}$$

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

$$s = ct$$

Tabla 2.2: Resumen del cálculo de τ, t, c y s.

Nuestro siguiente objetivo será el estudio de la convergencia. Para ello, examinaremos cómo se comportan las entradas fuera de la diagonal a medida que la iteración progresa. Antes de continuar, es importante tener en cuenta el siguiente resultado elemental, [8].

Lema 2.1.1. Se verifican las siguientes propiedades:

1. Si $A, B \in \mathbb{R}^{n\times n}$, traza$(AB)$ = traza(BA).

2. Si $A \in \mathbb{R}^{n\times n}$ es simétrica, se cumple que

$$\text{traza}(A^2) = \sum_{i,\ j=1}^{n} a_{ij}^2.$$

3. Si $P \in \mathbb{R}^{n\times n}$ ortogonal, entonces $\|P^T AP\|_F = \|A\|_F$.

La convergencia del método de Jacobi se basa en dos resultados:

- La sucesión $\{f(A_k)\}_{k=0}^{\infty}$ es decreciente.
- Una elección conveniente de los índices i, j en cada paso k de la iteracion.

Para la primera propiedad, se tiene:

Lema 2.1.2. Si $\bar{A} = J(i, j, c, s)^T AJ(i, j, c, s)$, entonces

$$\text{f}(\bar{A})^2 = \text{f}(A)^2 - 2a_{ij}^2.$$

Demostración. A partir de los resultados ya obtenidos, es evidente que todas las entradas en la diagonal de $\bar{A}$ excepto a_{ii} y a_{jj} son las mismas que aquéllas de A. Las entradas en los índices (i, i) y (j, j) en la diagonal deben satisfacer la siguiente relación:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \bar{a}_{ii} & 0 \\ 0 & \bar{a}_{jj} \end{bmatrix},$$

y así

$$\left\| \begin{bmatrix} \bar{a}_{ii} & 0 \\ 0 & \bar{a}_{jj} \end{bmatrix} \right\|_F^2 = \left\| \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \right\|_F^2.$$

Por lo tanto, ocurre que

$$\bar{a}_{ii}^2 + \bar{a}_{jj}^2 = a_{ii}^2 + a_{jj}^2 + 2a_{ij}^2.$$

puesto que $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ y $\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$ son matrices ortogonales.

Ahora bien, ya que todas las otras entradas fuera de la diagonal de A son idénticas a las de $\bar{A}$, tenemos

$$\sum_{i=1}^{n} \bar{a}_{ii}^2 = \sum_{i=1}^{n} a_{ii}^2 + 2a_{ij}^2. \tag{2.9}$$

Dado que A y $\bar{A}$ son similares ortogonalmente, $\|A\|_F^2 = \|\bar{A}\|_F^2$ por el Lema 2.1.1(iii). Esto y (2.9) implican

$$\text{f}(\bar{A})^2 = \|\bar{A}\|_F^2 - \sum_{k=1}^{n} \bar{a}_{ii}^2 = \|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2 - 2a_{ij}^2 = \text{f}(A)^2 - 2a_{ij}^2.$$

□

La elección de los índices en cada paso que asegura la convergencia viene dada en el siguiente resultado.

Teorema 2.1.1. El método de Jacobi converge si, en cada iteración, se selecciona el elemento fuera de la diagonal con mayor magnitud para realizar la rotación.

Demostración. Consideremos $A_k = J_{k-1}^T A_{k-1} J_{k-1}$. Hay $n^2 - n$ entradas fuera de la diagonal, y si a_{ij} es la de mayor magnitud,

$$(\mathrm{f}(A_{k-1}))^2 \leq n(n-1)a_{ij}^2 \implies \frac{(\mathrm{f}(A_{k-1}))^2}{n(n-1)} \leq a_{ij}^2.$$

Por el Lema 2.1.2, $\mathrm{f}(A_k)^2 = \mathrm{f}(A_{k-1})^2 - 2a_{ij}^2$, y así,

$$\mathrm{f}(A_k)^2 \leq \mathrm{f}(A_{k-1})^2 - 2\left(\frac{(\mathrm{f}(A_{k-1}))^2}{n(n-1)}\right) = \left(1 - \frac{1}{N}\right)\mathrm{f}(A_{k-1})^2, \qquad \text{donde} \quad N = \frac{n(n-1)}{2}. \tag{2.10}$$

Así, después de k pasos de iteración de Jacobi y teniendo en cuenta (2.10),

$$\mathrm{f}(A_k) \leq \left(\sqrt{1 - \frac{1}{N}}\right)^k \mathrm{f}(A),$$

lo que muestra que la iteración de Jacobi converge. □

El término $\mathrm{f}(A_k)$ disminuye a un ritmo de $\sqrt{\left(1 - \frac{1}{N}\right)}$, lo que indica que la tasa de convergencia es lineal. Sin embargo, se puede demostrar que la tasa de convergencia promedio (tasa asintótica) es cuadrática, véase [10].

2.1.1. El Método de Jacobi Unilateral para el Cálculo de la SVD Reducida

Veamos ahora cómo utilizar el método de Jacobi para computar la SVD reducida de una matriz $A \in \mathbb{R}^{m\times n}$ de rango completo. Como se ha mencionado en la introducción, la idea principal consiste en utilizar el método de Jacobi sobre A^TA o AA^T pero sin calcular explícitamente una o la otra. Centrándonos en A^TA, el método que ahora describiremos se llama método de Jacobi unilateral.

Necesitamos evitar tener que calcular A^TA, por lo que adoptaremos un enfoque que indirectamente realice los cálculos en A^TA mientras trabajamos con una secuencia de problemas aparentemente diferentes. Usamos una secuencia de rotaciones de Jacobi que hacen que las columnas i, j-ésimas, con $i < j$, de $AJ(i, j, c, s)$ sean ortogonales. Haciendo el producto e imponiendo la ortogonalidad, se tiene

$$\left\langle \begin{bmatrix} ca_{1i} - sa_{1j} \\ ca_{2i} - sa_{2j} \\ \vdots \\ ca_{ii} - sa_{ij} \\ \vdots \\ ca_{ji} - sa_{jj} \\ \vdots \\ ca_{ni} - sa_{nj} \end{bmatrix}, \begin{bmatrix} sa_{1i} + ca_{1j} \\ sa_{2i} + ca_{2j} \\ \vdots \\ sa_{ii} + ca_{ij} \\ \vdots \\ sa_{ji} + ca_{jj} \\ \vdots \\ sa_{ni} + ca_{nj} \end{bmatrix} \right\rangle = 0. \tag{2.11}$$

Desarrollando el producto interno en (2.11), tenemos

$$\begin{aligned}(ca_{1i} - sa_{1j})(sa_{1i} + ca_{1j}) \; + \; \cdots + (ca_{ii} - sa_{ij})(sa_{ii} + ca_{ij}) \\ + \cdots + (ca_{ji} - sa_{jj})(sa_{ji} + ca_{jj}) \\ + \cdots + (ca_{ni} - sa_{nj})(sa_{ni} + ca_{nj}) = 0. \end{aligned} \tag{2.12}$$

Simplificando, (2.12) se transforma en

$$(c^2 - s^2)\sum_{k=1}^{n} a_{ki}a_{kj} + cs\left[\sum_{k=1}^{n} a_{ki}^2 - \sum_{k=1}^{n} a_{kj}^2\right] = 0,$$

y por lo tanto

$$\frac{c^2 - s^2}{cs} = \frac{\sum_{k=1}^{n} a_{kj}^2 - \sum_{k=1}^{n} a_{ki}^2}{\sum_{k=1}^{n} a_{ki}a_{kj}}. \tag{2.13}$$

Procedemos como hicimos en la ecuación (2.4), excepto que el lado derecho es diferente. El resultado es

$$t^2 + 2\tau t - 1 = 0, \quad \text{con } \tau = \frac{1}{2}\frac{\sum_{k=1}^{n} a_{kj}^2 - \sum_{k=1}^{n} a_{ki}^2}{\sum_{k=1}^{n} a_{ki}a_{kj}},$$

y, de nuevo, $s = ct$. La tabla 2.3 proporciona un resumen de los cálculos requeridos.

Cálculo de c y s en el Método de Jacobi Unilateral

$$\tau = \frac{1}{2}\frac{\sum_{k=1}^{n} a_{kj}^2 - \sum_{k=1}^{n} a_{ki}^2}{\sum_{k=1}^{n} a_{ki}a_{kj}}$$

$$t = \begin{cases} \frac{1}{\tau + \sqrt{\tau^2+1}} & \text{si } \tau \geq 0 \\ \frac{1}{-\tau + \sqrt{\tau^2+1}} & \text{si } \tau < 0 \end{cases}$$

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

$$s = ct$$

Tabla 2.3: Resumen del cálculo de τ, t, c y s en el método de Jacobi unilateral.

La relación con el método de Jacobi aplicado a A^TA es como sigue. Los elementos (i,i), (j,j), (i,j) y (j,i) de A^TA se muestran en la siguiente matriz:

$$A^TA = \begin{bmatrix} \ddots & & & \\ & \sum_{k=1}^{n} a_{ki}^2 & \cdots & \sum_{k=1}^{n} a_{ki}a_{kj} \\ & \vdots & \ddots & \vdots \\ & \sum_{k=1}^{n} a_{ki}a_{kj} & \cdots & \sum_{k=1}^{n} a_{kj}^2 \\ & & & \ddots \end{bmatrix}. \tag{2.14}$$

Ahora, la rotación de Jacobi J tal que J^TA^TAJ tiene ceros en las posiciones (i,j) y (j,i) se obtiene eligiendo c y s como la tabla 2.2, sustituyendo a_{ij}, a_{ii} y a_{jj} por los correspondientes

elementos de A^TA indicados en (2.14). Los valores de c y s obtenidos son los mismos que los de la tabla 2.3. Es decir, eligiendo c y s tales que las columnas i y j de $AJ(i,j,c,s)$ son ortogonales se consigue hacer cero en las posiciones (i,j) y (j,i) de

$$J(i,j,c,s)^T A^T A J(i,j,c,s).$$

El algoritmo entonces, es como sigue. Comienza con A y aplica una secuencia de rotaciones de Jacobi hasta que el resultado sea una matriz $\bar{U}$ con columnas *casi ortogonales*[1] :

$$AJ_1J_2J_3\cdots J_k = \bar{U} \tag{2.15}$$

Esto equivale, según lo dicho anteriormente, a construir la sucesión de transformaciones de Jacobi que lleven A^TA a la forma diagonal

$$J_k^T\cdots J_2^TJ_1^TA^TAJ_1J_2\cdots J_k \approx \Sigma^2, \quad \text{con} \quad \Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{bmatrix}, \tag{2.16}$$

donde los $\sigma_i, 1 \le i \le n$, son los valores singulares de A. De modo que, si $V = J_1\cdots J_k$, podemos escribir (2.15) como

$$AV = \bar{U}, \tag{2.17}$$

y, teniendo en cuenta la ortogonalidad de V:

$$A = \bar{U}V^T. \tag{2.18}$$

Trasponiendo (2.18), $A^T = V\bar{U}^T$. Usando este resultado en (2.16) obtenemos:

$$J_k^T\cdots J_2^TJ_1^TV\bar{U}^TAJ_1J_2\cdots J_k \approx \Sigma^2.$$

Usando de nuevo (2.17),

$$J_k^T\ldots J_2^TJ_1^TV\bar{U}^T\bar{U} \approx \Sigma^2.$$

Dado que $V = J_1J_2J_3\ldots J_k$, tenemos:

$$J_k^T\ldots J_2^TJ_1^TJ_1J_2J_3\ldots J_k\bar{U}^T\bar{U} \approx \Sigma^2,$$

y por tanto,

$$\bar{U}^T\bar{U} \approx \Sigma^2. \tag{2.19}$$

Asumiendo que las columnas de $\bar{U}$ son ortogonales[2], y escribiendo

$$\bar{U} = (\bar{u}_1 \mid \bar{u}_2 \mid \ldots \mid \bar{u}_n),$$

[1]Si $A \in \mathbb{R}^{m\times n}$ es una matriz con columnas $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$, diremos que las columnas de A son casi ortogonales si el producto escalar entre dos columnas diferentes es pequeño en magnitud, es decir:

$$< \mathbf{a}_i \ , \ \mathbf{a}_j > \approx 0 \quad \text{para} \quad i \neq j.$$

[2]En realidad, se trata de una simplificación. En la práctica, las columnas de $\bar{U}$ son *casi ortonormales*, por lo que las entradas no diagonales en Σ^2 serán aproximadamente cero.

entonces (2.19) se puede escribir

$$\begin{bmatrix} \|\bar{u}_1\|_2^2 & & & \\ & \|\bar{u}_2\|_2^2 & & \\ & & \ddots & \\ & & & \|\bar{u}_n\|_2^2 \end{bmatrix} \approx \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_n^2 \end{bmatrix}. \tag{2.20}$$

La ecuación (2.20) nos dice que la norma euclídea de cada columna de $\bar{U}$ toma, aproximadamente, los valores singulares de A, esto es,

$$\|\bar{u}_i\|_2 \approx \sigma_i, \quad i = 1, 2 \ \dots \ n.$$

Así, $U = \left(\frac{\bar{u}_1}{\sigma_1}, \frac{\bar{u}_2}{\sigma_2}, \ \cdots, \frac{\bar{u}_n}{\sigma_n}\right)$ es una matriz ortogonal, ya que $\frac{\bar{u}_i}{\sigma_i}$ es un vector unitario. Por otro lado, $\bar{U} = U\Sigma$, y usando esto en la ecuación (2.18), tenemos

$$A = U\Sigma V^T,$$

la descomposición en valores singulares de A.

Sabemos que el método de Jacobi aplicado a la matriz simétrica A^TA converge a una matriz diagonal que contiene sus valores propios. Hemos especificado que se debe continuar el algoritmo de Jacobi para la SVD hasta que $AJ_1J_2J_3 \dots J_k$ sea *casi ortogonal*. La duda que surge ahora es la de cómo medir el grado de ortogonalidad para asegurarnos de que el algoritmo proporcione resultados precisos. Esto puede implementarse fijando $\varepsilon > 0$ y exigiendo, para todo i, j en la iteración actual, que

$$\left| \left\langle \frac{\bar{u}_i}{\|\bar{u}_i\|_2}, \frac{\bar{u}_j}{\|\bar{u}_j\|_2} \right\rangle \right| \leq \varepsilon. \tag{2.21}$$

2.2. El Algoritmo de Demmel-Kahan

El método de Demmel-Kahan, conocido en la literatura como *Implicit Zero-Shift QR Downward Sweep algorithm*, es una técnica utilizada para calcular la descomposición en valores singulares de una matriz. Como se indicó en la introducción, este algoritmo requiere una reducción previa de la matriz $A \in \mathbb{R}^{m\times n}$ a otra bidiagonal superior, en este caso mediante el uso de reflectores de Householder.

2.2.1. Los Reflectores de Householder

Definición 2.2.1. Dado un vector $\mathbf{u} \in \mathbb{R}^n$ tal que $\mathbf{u} \neq 0$, se define el *reflector de Householder* o *transformación de Householder* asociada a u de la forma siguiente:

$$P(\mathbf{u}) = I - \frac{2}{\mathbf{u^T u}}\mathbf{u u^T}. \tag{2.22}$$

Por otro lado, teniendo en cuenta que $\|\mathbf{u}\|_2 = \sqrt{\mathbf{u^T u}}$, podemos sustituir dicha expresión en el denominador de (2.22) y obtenemos que:

$$P(\mathbf{u}) = I - 2\frac{\mathbf{u u^T}}{\|\mathbf{u}\|_2^2},$$

es decir:

$$P(\mathbf{u}) = I - 2\frac{\mathbf{u}\mathbf{u}^{\mathbf{T}}}{\|\mathbf{u}\|_2^2} = I - 2\frac{\mathbf{u}}{\|\mathbf{u}\|_2}\frac{\mathbf{u}^T}{\|\mathbf{u}\|_2} = P\left(\frac{\mathbf{u}}{\|\mathbf{u}\|_2}\right).$$

Por lo tanto, podemos considerar sin pérdida de generalidad que $\|\mathbf{u}\|_2 = 1$, y el valor de $P(\mathbf{u})$ depende únicamente de la dirección del vector $\mathbf{u}$ y no de su norma. En este caso, se escribe simplemente $P(\mathbf{u}) = I - 2\mathbf{u}\mathbf{u}^{\mathbf{T}}$.

Se puede entender el reflector de Householder $P(\mathbf{u})$ de un vector $\mathbf{v}$ como su reflexión respecto del plano dado por $\{\mathbf{u}\}^\perp = \{\mathbf{v} \in \mathbb{R}^n \mid \mathbf{u}^{\mathbf{T}}\mathbf{v} = 0\}$. Un resultado de gran relevancia es el siguiente teorema, que describe una de las propiedades más importantes de los reflectores de Householder, [8].

Lema 2.2.1. Si $\|\mathbf{a}\|_2 = \|\mathbf{b}\|_2 \neq 0$, entonces

$$P(\mathbf{a} - \mathbf{b})\mathbf{a} = \mathbf{b}.$$

Este resultado nos permite deducir que, dado un vector $\mathbf{x} \in \mathbb{R}^k$, no nulo, podemos encontrar una matriz Q ortogonal tal que $Q\mathbf{x}$ sea de la forma $\mathbf{y} = (\tilde{y}, 0, 0, \ldots, 0)^T$, donde $\tilde{y} = \pm\sqrt{x_1^2 + x_2^2 + \cdots + x_k^2}$. Basta con construir el reflector correspondiente a $\mathbf{x} - \mathbf{y} = (x_1 - \tilde{y}, x_2, \ldots, x_k)$.

2.2.2. Transformación de una Matriz a su Forma Bidiagonal Superior

Siguiendo a [9], exponemos aquí un esquema desarrollado por Lanczos [13]. Sea $A \in \mathbb{C}^{m\times n}$ y A^* su matriz adjunta. Supondremos, sin pérdida de generalidad, que $m \geq n$. La matriz

$$\tilde{A} = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}$$

tiene por valores propios los valores singulares de A, tanto con signo positivo como negativo. No obstante, al tratar de calcular de forma estándar los autovalores y autovectores de $\tilde{A}$, nos podemos encontrar con varios problemas, que discutiremos a continuación.

Con el objetivo de facilitar el cálculo de los valores singulares y la pseudoinversa de A, describiremos una descomposición matricial conveniente.

Teorema 2.2.1. Sea $A \in \mathbb{C}^{m\times n}$. Entonces, A puede escribirse como:

$$A = PJQ^*, \tag{2.23}$$

donde P y Q son matrices unitarias y J es una matriz bidiagonal de tamaño $m \times n$ de la forma:

$$J = \left(\begin{array}{cccccc} \alpha_1 & \beta_1 & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & \alpha_n \\ \hline 0 & \cdots & \cdots & 0 & 0 & 0 \\ 0 & \cdots & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & 0 \end{array}\right) \begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \Bigg\} (m-n) \times n$$

Demostración. La demostración es constructiva y utiliza transformaciones de Householder. Sean $A^{(1)} = A$ y sean $A^{(3/2)}, A^{(2)} \ldots, A^{(n)}, A^{(n+1/2)}$ definidas como:

$$A^{(k+1/2)} = P^{(k)} A^{(k)}, \quad k = 1, 2, \ldots, n,$$
$$A^{(k+1)} = A^{(k+1/2)} Q^{(k)}, \quad k = 1, 2, \ldots, n-1.$$

Aquí, $P^{(k)}$ y $Q^{(k)}$ son matrices unitarias hermíticas de la forma:

$$P^{(k)} = I - 2x^{(k)} x^{(k)*}, \quad x^{(k)*} x^{(k)} = 1,$$
$$Q^{(k)} = I - 2y^{(k)} y^{(k)*}, \quad y^{(k)*} y^{(k)} = 1.$$

Las transformaciones unitarias $P^{(k)}$ y $Q^{(k)}$ están determinadas de modo que

$$a_{i,k}^{(k+1/2)} = 0, \quad i = k+1, \ldots, m, \qquad\qquad a_{k,j}^{(k+1)} = 0, \quad j = k+2, \ldots, n.$$

Y $A^{(k+1)}$ tiene la forma:

$$A^{(k+1)} = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \alpha_3 & \beta_3 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \alpha_k & \beta_k & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & x & x & \cdots & x \\ \vdots & \vdots & \vdots & \cdots & 0 & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & x & x & \cdots & x \end{pmatrix}.$$

2.2.3. Cálculo de la SVD de una matriz bidiagonal. El Algoritmo Golub-Kahan.

Siguiendo a [9, 11], describimos ahora el proceso de cálculo de la SVD de una matriz bidiagonal. Para ello, se pueden seguir varios procedimientos.

Los valores singulares $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$ de J en (2.23) son los mismos que los de A. Se trata de las raíces cuadradas positivas de J^*J. Así, si la descomposición en valores singulares de J es $J = X\Sigma Y^*$ entonces

$$A = U\Sigma V^*,$$

de modo que $U = PX$, $V = QY$.

Es importante tener en cuenta que las últimas $m - n$ filas de ceros en J no contribuyen a los valores singulares, ni tienen más que un efecto trivial sobre la determinación de X e Y. Por lo tanto, es conveniente eliminar las últimas $m - n$ filas de J y escribir

$$J = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & 0 & \cdots & 0 \\ 0 & 0 & \alpha_3 & \beta_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \cdots & 0 & 0 & 0 & \alpha_n \end{pmatrix},$$

denotándola también como J. Esto se puede hacer porque las ecuaciones anteriores siguen siendo válidas después del siguiente proceso de eliminación de filas.

- Eliminar las últimas $m - n$ filas de ceros en J y Σ.
- Eliminar las últimas $m - n$ columnas de P y U.
- Eliminar las últimas $m - n$ filas y columnas de X.

De ahora en adelante, trabajaremos con estas matrices reducidas.

Se sabe que los valores singulares σ_i de J están relacionados con los valores propios de la matriz de dimensión $2n \times 2n$ siguiente

$$\tilde{J} = \begin{pmatrix} 0 & J \\ J^* & 0 \end{pmatrix}, \tag{2.24}$$

cuyos valores propios son $\pm\sigma_i$ para $i = 1, 2, \ldots, n$. El cálculo de los valores propios de $\tilde{J}$ se simplifica conceptualmente mediante una transformación a forma tridiagonal vía permutación. Consideremos la ecuación matricial

$$\tilde{J} \begin{pmatrix} x \\ \pm y \end{pmatrix} = \pm\sigma \begin{pmatrix} x \\ \pm y \end{pmatrix},$$

que, cuando se expande, se puede escribir como

$$Jy = \sigma x, \quad J^* x = \sigma y,$$

es decir,

$$\alpha_i y_i + \beta_i y_{i+1} = \sigma x_i, \quad \bar{\alpha}_1 x_1 = \sigma y_1, \quad \alpha_n y_n = \sigma x_n, \quad \bar{\beta}_{i-1} x_{i-1} + \bar{\alpha}_i x_i = \sigma y_i.$$

Consideramos ahora la sustitución

$$z_{2i} = x_i, \quad z_{2i-1} = \pm y_i.$$

De esta forma, obtenemos la ecuación

$$Tz = \pm\sigma z,$$

en la cual, T es una matriz tridiagonal de dimensión $2n \times 2n$ definida por

$$T = \begin{pmatrix} 0 & \bar{\alpha}_1 & 0 & 0 & \cdots & 0 \\ \alpha_1 & 0 & \beta_1 & 0 & \cdots & 0 \\ 0 & \bar{\beta}_1 & 0 & \bar{\alpha}_2 & \cdots & 0 \\ 0 & 0 & \alpha_2 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \bar{\alpha}_n \\ 0 & 0 & \cdots & \cdots & \alpha_n & 0 \end{pmatrix}.$$

Se puede demostrar ([9]) que existe una matriz diagonal unitaria D tal que

$$DTD^* = S = \begin{pmatrix} 0 & s_1 & 0 & 0 & 0 & \cdots & 0 \\ s_1 & 0 & t_1 & 0 & 0 & \cdots & 0 \\ 0 & t_1 & 0 & s_2 & 0 & \cdots & 0 \\ 0 & 0 & s_2 & 0 & t_2 & \cdots & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & s_n \\ 0 & 0 & 0 & \cdots & 0 & s_n & 0 \end{pmatrix},$$

produce una matriz tridiagonal S cuyos elementos, $s_i = |\alpha_i|$ y $t_i = |\beta_i|$, son todos reales y no negativos. Hay varios métodos para obtener los valores propios de una matriz simétrica tridiagonal (véase [8]). Uno puede simplificar el algoritmo aprovechando el hecho de que los elementos diagonales de T son cero.

Otro método para calcular los valores singulares de J es calcular los valores propios de

$$J^*J = \begin{pmatrix} |\alpha_1|^2 & \bar{\alpha_1}\beta_1 & 0 & 0 & 0 & \cdots & 0 \\ \alpha_1\bar{\beta_1} & |\alpha_2|^2 + |\beta_1|^2 & \bar{\alpha_2}\beta_2 & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2\bar{\beta_2} & |\alpha_3|^2 + |\beta_2|^2 & \bar{\alpha_3}\beta_3 & 0 & \cdots & 0 \\ 0 & 0 & \alpha_3\bar{\beta}_3 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \bar{\alpha}_{n-1}\beta_{n-1} \\ 0 & 0 & 0 & & & \alpha_{n-1}\bar{\beta}_{n-1} & |\alpha_n|^2 + |\beta_{n-1}|^2 \end{pmatrix}.$$

Nótese que, dado que J^*J es una matriz hermítica tridiagonal, existe una matriz diagonal unitaria, que denotaremos por Δ, tal que:

$$\Delta(J^*J)\Delta^* = K = \begin{pmatrix} s_1^2 & s_1t_1 & 0 & 0 & 0 & \cdots & 0 \\ s_1t_1 & s_2^2 + t_1^2 & s_2t_2 & 0 & 0 & \cdots & 0 \\ 0 & s_2t_2 & s_3^2 + t_2^2 & s_3t_3 & 0 & \cdots & 0 \\ 0 & 0 & s_3t_3 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & s_{n-1}t_{n-1} \\ 0 & 0 & 0 & & & s_{n-1}t_{n-1} & s_n^2 + t_{n-1}^2 \end{pmatrix},$$

donde $s_i = |\alpha_i|$ y $t_i = |\beta_i|$. Por lo tanto, K es una matriz tridiagonal real, simétrica y semidefinida positiva, y sus valores propios pueden ser calculados fácilmente.

El último método que estudiaremos es el llamado *método de deflación*, que implica el uso de la deflación para eliminar cada vector propio a medida que se obtiene y, por lo tanto, garantizar la ortogonalidad. Mediante una variante del algoritmo QR, la matriz J se diagonaliza iterativamente de modo que

$$J \to J^{(1)} \to \ldots \to \Sigma, \quad \text{donde} \quad J^{(i+1)} = S^{(i)T} J^{(i)} T^{(i)},$$

y $S^{(i)}, T^{(i)}$ son ortogonales. Las matrices $T^{(i)}$ se eligen de modo que la secuencia $M^{(i)} = J^{(i)T} J^{(i)}$ converge a una matriz diagonal, mientras que las matrices $S^{(i)}$ se eligen de modo que todas las $J^{(i)}$ sean de forma bidiagonal.

Para facilitar la notación, omitimos el sufijo y usamos la notación

$$J = J^{(i)}, \quad \bar{J} = J^{(i+1)}, \quad S = S^{(i)}, \quad T = T^{(i)}, \quad M \equiv J^T J, \quad \bar{M} \equiv \bar{J}^T \bar{J}.$$

La transición $J \to \bar{J}$ se logra mediante la aplicación de rotaciones de Givens a J alternativamente desde la derecha y la izquierda. Así,

$$\bar{J} = S_n^T S_{n-1}^T \ldots S_2^T J T_2 T_3 \ldots T_n, \tag{2.25}$$

donde

$$S_k = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & \cos\theta_k & -\sin\theta_k & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \sin\theta_k & \cos\theta_k & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ (k-1) \\ (k) \\ \\ \\ \\ \end{matrix},$$

y T_k se define de manera análoga a S_k con φ_k en lugar de θ_k. Definimos, además,

$$S = S_2 \ldots S_{n-1} S_n, \qquad T = T_2 T_3 \ldots T_n.$$

Sea el primer ángulo, φ_2, arbitrario mientras que todos los demás ángulos se eligen de modo que $\bar{J}$ tenga la misma forma que J. Así,

1. T_2 no anula nada, genera una entrada $\{J\}_{21}$,
2. S_2^T anula $\{J\}_{21}$, genera una entrada $\{J\}_{13}$,
3. T_3 anula $\{J\}_{13}$, genera una entrada $\{J\}_{32}$,
   $\vdots$
4. Finalmente, S_n^T anula $\{J\}_{n,n-1}$ y no genera ninguna entrada.

De hecho, este proceso se conoce frecuentemente con el nombre de *chasing* o *bulge chasing*. El algoritmo se ilustra gráficamente en la figura 2.1.

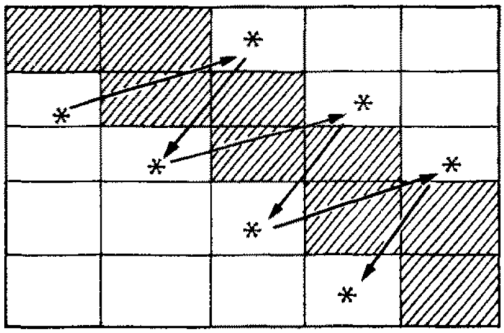


Figura 2.1: Algoritmo *chasing*. Extraído de [11]

Dado que $\bar{J} = S^T JT$, entonces $\bar{M} = \bar{J}^T \bar{J} = T^T MT$ y $\bar{M}$ es una matriz tridiagonal al igual que M. Veamos que el primer ángulo, φ_2, que aún está indeterminado, puede ser elegido de manera que la transición $M \to \bar{M}$ sea una transformación QR con un desplazamiento dado s. El algoritmo QR usual con desplazamientos se describe de la siguiente manera:

$$M - sI = T_s R_s, \tag{2.26}$$
$$R_s T_s + sI = \bar{M}_s, \tag{2.27}$$

donde $T_s^T T_s = I$ y R_s es una matriz triangular superior. Así, $\bar{M}_s = T_s^T M T_s$. No es necesario calcular (2.26) y (2.27) explícitamente, pero es posible realizar el desplazamiento implícitamente. Sea T por el momento una matriz arbitraria tal que

$$\{T_s\}_{k,1} = \{T\}_{k,1} \quad (k = 1, 2, \ldots, n),$$

es decir, los elementos de la primera columna de T_s son iguales a la primera columna de T, y

$$T^T T = I.$$

Teorema 2.2.2. (Francis.) [11] Si ocurre que:

1. $\bar{M} = T^T MT$,

2. $\bar{M}$ es una matriz tridiagonal,

3. Los elementos subdiagonales de M son diferentes de cero,

se sigue que $\bar{M} = D\bar{M}_s D$ donde D es una matriz diagonal cuyos elementos diagonales son ± 1.

Así, se elige T_2 en el algoritmo de *chasing* de manera que su primera columna sea proporcional a la de $M - sI$, lo mismo es cierto para la primera columna del producto $T = T_2 T_3 \ldots T_n$, que por lo tanto es idéntica a la de T_s. Por lo tanto, si la subdiagonal de M no contiene ningún elemento diferente de cero, las condiciones del teorema de Francis se cumplen y T es por lo tanto idéntica a T_s. Así, la transición (2.25) es equivalente a la transformación QR de $J^T J$ con un desplazamiento dado s. El parámetro de desplazamiento s está determinado por un valor propio del menor 2×2 más pequeño de M. Para esta elección de s, el método converge globalmente y casi siempre cúbicamente.

2.2.4. El Algoritmo QR Implícito con Desplazamiento Cero de Demmel-Kahan

En esta subsección, siguiendo a [5], presentamos una variación de este algoritmo estándar que calcula todos los valores singulares de una matriz bidiagonal, incluso los más pequeños, con una precisión relativa garantizada. Lo llamaremos método de Demmel-Kahan o algoritmo QR implícito con desplazamiento cero, ya que corresponde al algoritmo anterior cuando $s = 0$.

Sea B una matriz bidiagonal obtenida a partir del procedimiento de la subsección 2.2.2. A partir de $B_0 = B$, el algoritmo genera una sucesión de matrices B_i que converge a una matriz diagonal, cuyas entradas diagonales son aproximaciones a los valores singulares de B. El grado de la aproximación viene determinado por la propiedad de que el paso de B_i a B_{i+1} asegure la conservación de los valores singulares con precisión alta. Es importante tener en cuenta que este algoritmo se empleará si el número de condición de la matriz es grande.

Mientras que el algoritmo de Golub-Kahan solamente garantiza que los valores singulares son calculados con pequeños errores absolutos (véase el teorema 2.0.1), el método de Demmel-Kahan garantiza que los valores singulares son calculados con pequeños errores relativos ([5]),

$$\frac{|\tilde{\sigma}_i - \sigma_i|}{\sigma_i} \leq \mathcal{O}(\varepsilon) \quad i = 1 \dots n.$$

Esta excelente precisión se debe a que, tomando desplazamiento $s = 0$, se evitan posibles cancelaciones no deseadas.

2.3. Estudio Comparativo

En este estudio comparativo, se analizan los métodos de Jacobi y Demmel-Kahan Zero Shift para la descomposición en valores singulares, tanto a nivel teórico como experimental.

2.3.1. Una Comparativa Teórica de Ambos Métodos

La precisión del algoritmo de Jacobi puede medirse en ciertos casos.

Teorema 2.3.1. Sea $G = DX$ una matriz de dimensión $n \times n$, donde D es diagonal y no singular, y X es no singular. Sea $\tilde{G}$ la matriz tras aplicar el algoritmo de Jacobi unilateral en G, consistente en m rotaciones de Jacobi $J(j, k, c, s)$ consecutivas en aritmética de punto flotante. Sean $\sigma_1 \geq \cdots \geq \sigma_n$ los valores singulares de G, y $\tilde{\sigma}_1 \geq \cdots \geq \tilde{\sigma}_n$ los valores singulares de $\tilde{G}$. Entonces,

$$\frac{|\sigma_i - \tilde{\sigma}_i|}{\sigma_i} \leq \mathcal{O}(m\varepsilon) \cdot \kappa(X), \tag{2.28}$$

donde $\kappa(X) = \|X\| \cdot \|X^{-1}\|$ es el número de condición de X. Es decir, el error relativo en los valores singulares es pequeño si el número de condición de X es pequeño.

Demostración. Se puede consultar en [6].

Siguiendo a [5], se presentan a continuación dos teoremas que abordan la convergencia del método de Demmel-Kahan Zero Shift.

Teorema 2.3.2. (**Ley de inercia de Sylvester.**) Sea A una matriz simétrica y U una matriz no singular. Entonces, A y UAU^T tienen el mismo número de valores propios positivos, nulos y negativos.

Demostración. Véase [5].

Teorema 2.3.3. Sea B una matriz bidiagonal de tamaño $n \times n$ y B' la matriz obtenida tras ejecutar el algoritmo de Demmel-Kahan sobre B. Sean los valores singulares de B denotados por $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, y los valores singulares de B' por $\sigma'_1 \geq \sigma'_2 \geq \cdots \geq \sigma'_n$. Si se cumple que

$$\omega = 69n^2\varepsilon < 1, \tag{2.29}$$

entonces las diferencias relativas entre los valores singulares de B y B' están acotadas de la siguiente manera:

$$\frac{|\sigma_i - \sigma'_i|}{\sigma_i} \leq \frac{\omega}{1-\omega}, \quad \forall i.$$

Sea ahora B_k la matriz obtenida después de k repeticiones del algoritmo QR sin desplazamiento implícito, y $\sigma_{k1} \geq \sigma_{k2} \geq \cdots \geq \sigma_{kn}$ sus valores singulares. Si se cumple la condición 2.29, entonces:

$$\frac{|\sigma_i - \sigma_{ki}|}{\sigma_i} \leq \frac{1}{(1-\omega)^k} - 1 \approx 69kn^2\varepsilon,$$

donde la aproximación se cumple si $k\omega < 1$.

Demostración. Véase [5].

2.3.2. Una Comparativa Experimental de Ambos Métodos. Optimización

Para desarrollar un análisis comparativo entre los métodos de Jacobi y Demmel-Kahan sin desplazamiento, el trabajo de Alessandrini en [1] presenta un análisis exhaustivo y una implementación optimizada de diferentes métodos SVD. Inspirándonos en este enfoque, este apartado se centra en una comparativa experimental de dichos métodos, evaluando métricas como precisión, velocidad y consumo energético, con el objetivo de identificar el método más adecuado en cada caso.

En [1] se presenta un estudio comparativo de los cinco algoritmos más representativos para la descomposición en valores singulares: Golub-Reinsch, Demmel-Kahan, rotación de Jacobi, rotación unidireccional de Jacobi y el algoritmo de divide y vencerás (véase [6]). En este trabajo nos centramos principalmente en los algoritmos de Demmel-Kahan y Jacobi unidireccional debido a su relevancia para nuestro análisis, aunque incluimos los demás a modo ilustrativo para proporcionar un contexto completo sobre las opciones disponibles y sus características.

Los algoritmos se han probado con varias matrices $A \in \mathbb{R}^{m\times n}$. Para garantizar que funcionaran en los casos más generales, se han seleccionado varios conjuntos de matrices de tamaño creciente, cada conjunto con una relación diferente de filas/columnas. Teniendo en cuenta la memoria limitada del microcontrolador, se probaron tres conjuntos de matrices con una relación $m/n = 1, 4/3$ y 2, respectivamente. Aumentar aún más esta relación hubiera provocado que las matrices no pudieran ajustarse a la memoria limitada del microcontrolador sin reducir significativamente su rango, pues la ocupación de memoria debe tener en cuenta no solo las

matrices de entrada y salida, sino también las matrices y vectores intermedios necesarios por los diferentes algoritmos.

La matriz más grande A fue generada aleatoriamente, y todas las demás matrices utilizadas son la porción superior izquierda de la matriz A completa. Para los algoritmos que requieren una matriz simétrica, la matriz de entrada se convierte a una forma simétrica de la siguiente manera. Primero, se convierte a forma bidiagonal aplicando reflectores de Householder (ver Algoritmo 1 de [1]), luego, siendo B la porción triangular superior de la matriz bidiagonal, se aplica el algoritmo SVD a BB^T. De esta manera, los valores singulares de la matriz original son las raíces cuadradas de los eigenvalores de la última matriz tridiagonal.

Según [1], los algoritmos se implementaron inicialmente en MATLAB para probar su corrección, en un procesador Intel i7 de 6 núcleos y 12 hilos con 32 GB de RAM. La figura 2.2 muestra, a modo de comparación, los tiempos de los diferentes algoritmos implementados en MATLAB, en función del número de columnas n para un conjunto de matrices con una relación $\frac{m}{n} = \frac{4}{3}$.

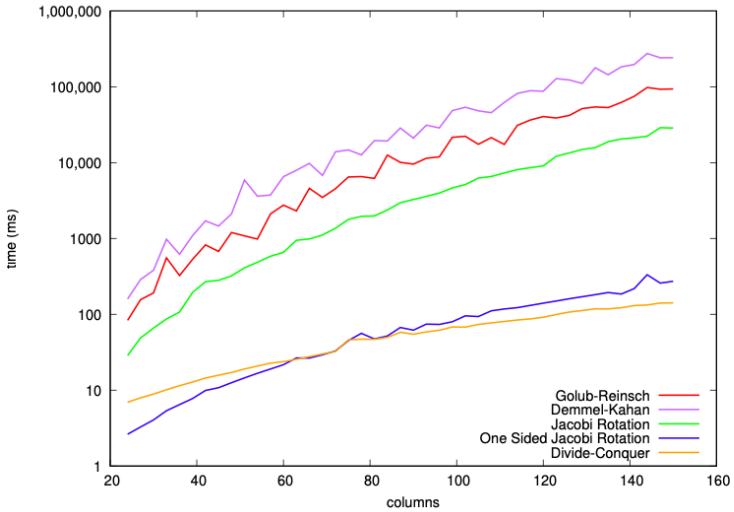


Figura 2.2: Tiempos de los algoritmos de SVD implementados en MATLAB para matrices $A \in \mathbb{R}^{m \times n}$ con $\frac{m}{n} = \frac{4}{3}$.

Por otro lado, la tabla 2.4 muestra la precisión de una muestra de las mismas pruebas, con los algoritmos implementados en MATLAB respecto a la función de SVD de MATLAB.

Size	Golub-Reinsch	Demmel-Kahan	Jacobi Rot.	One-Sided J.	Divide-Conquer
32×24	0	2.0×10^{-8}	0	0	0
48×36	0	2.7×10^{-8}	0	0	0
96×72	0	5.4×10^{-8}	0	0	0
128×96	0	7.9×10^{-8}	0	0	0
160×120	0	8.6×10^{-8}	0	0	0
200×150	0	1.9×10^{-7}	0	0	0

Tabla 2.4: Errores promedio de los algoritmos SVD implementados en MATLAB frente a la función integrada de MATLAB.

Los errores reportados se calculan como el promedio de los errores relativos entre los pares de valores singulares, calculados por la función integrada en MATLAB y las rutinas propuestas. Como se puede observar, todos los algoritmos, con excepción del Demmel-Kahan, garantizan la misma precisión que la función SVD integrada de MATLAB. Esto podría esperarse, ya que el algoritmo Demmel-Kahan fue diseñado específicamente para manejar bien los valores singulares muy pequeños, mientras que las matrices generadas aleatoriamente que se emplean en este experimento tienen valores singulares cercanos a la unidad.

Una vez verificada la corrección de los algoritmos en MATLAB, éstos fueron implementados y probados en un microcontrolador STM32F429ZI, un microcontrolador de STMicroelectronics basado en un núcleo ARM `Cortex-M4F` de 32 bits, con 2 MB de memoria flash y 256 KB de RAM, a 180 MHz. Este microcontrolador incluye una unidad de punto flotante (FPU) de 32 bits, necesaria para los cálculos con precisión simple en los algoritmos de SVD.

Los datos de solo lectura, como la matriz principal y los vectores de valores singulares, se almacenan en la memoria flash, más abundante que la RAM. Además, se optimizaron las rutinas matemáticas para reducir el uso de memoria, acelerando los cálculos dentro de los recursos limitados del sistema. Una optimización clave fue decidir cómo almacenar las matrices en memoria, ya sea por filas o columnas (véase la figura 2.3). Aunque los microcontroladores no tienen caché, almacenar las matrices por filas sigue siendo ventajoso debido a la eficiencia en el acceso secuencial a los datos. Como ejemplo del impacto de almacenar los datos por filas

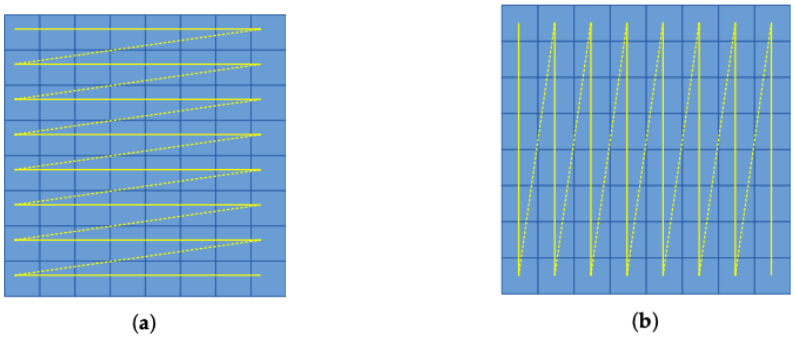
Matrix Size	Row-Major	Column-Major	Speed Increase	Transposition
48×24	19.6 ms	19.1 ms	2.6%	0.07 ms
72×36	73.1 ms	71.9 ms	1.7%	0.15 ms
96×48	192 ms	189.7 ms	1.2%	0.25 ms
120×60	370.6 ms	366.8 ms	1%	0.39 ms
144×72	634 ms	628.6 ms	0.9%	0.55 ms

Tabla 2.5: Impacto del orden de datos en memoria en los tiempos del método unidireccional de Jacobi.

o por columnas, se tomó el algoritmo de Jacobi unidireccional, que solo accede a la matriz por columnas. La tabla 2.5 muestra los tiempos del algoritmo para las matrices más grandes, tanto en tiempo total como en porcentaje de aumento de velocidad al almacenar los datos por filas en comparación con almacenarlos por columnas (véase la figura 2.3). Aunque el aumento de velocidad es pequeño, es comparativamente notable.

La última columna de la tabla 2.5 muestra el tiempo necesario para trasponer la matriz si está almacenada de forma diferente, lo cual es aproximadamente un orden de magnitud menor que la mejora de tiempo obtenida. Además, en muchos casos, la matriz de entrada para SVD se genera a partir de cálculos previos, por lo que se puede generar ya en el orden más adecuado.

Además de acceder a los datos en el orden adecuado, lo que mejora ligeramente la velocidad, y dado que los recursos de hardware disponibles son limitados, se debe optimizar el proceso reduciendo al máximo los cálculos innecesarios.



(a) Orden fila-mayor (b) Orden columna-mayor

Figura 2.3: Matriz almacenada en memoria en (a) orden fila-mayor y (b) orden columna-mayor. Ilustraciones extraídas de [1].

Un ejemplo es la multiplicación matricial, que es una de las operaciones más costosas en álgebra matricial. En general, si $C = AB$, con A y B cuadradas y de tamaño n, el elemento genérico de C se calcula como:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}. \tag{2.30}$$

Realizar este cálculo para todos los elementos de C de esta manera implica tres bucles anidados, lo cual es muy costoso computacionalmente, especialmente cuando se manejan matrices grandes. No obstante, el número de operaciones puede reducirse aprovechando propiedades específicas de las matrices. Un caso común es cuando se multiplica una matriz cuadrada por su traspuesta, como en $C = AA^T$. En este caso, la ecuación (2.30) se convierte en:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} a_{j,k}. \tag{2.31}$$

En el bucle interior, A siempre se procesa por filas. Esto permite aprovechar una lectura más eficiente si la matriz está almacenada por filas. Además, como AA^T es simétrica, basta con calcular la mitad de sus elementos, reduciendo significativamente las operaciones. Además, es posible reducir aún más el número de operaciones en este caso cuando A es una matriz bidiagonal con diagonal y primera superdiagonal no nulas, un caso común en ciertos algoritmos (*cf.* figura 2.4). Dado que $a_{i,j} \neq 0$ solo cuando $j = i$ o $j = i+1$, de la ecuación (2.31) se deduce que $c_{i,j} \neq 0$ solo cuando $j = i-1, i$ o $i+1$. De hecho, la matriz resultante es tridiagonal. Los términos no nulos en la suma son aquellos donde tanto $a_{i,k}$ como $a_{j,k}$ son distintos de cero. La fórmula final es:

$$c_{i,j} = \begin{cases} a_{i,i}a_{j,i} + a_{i,i+1}a_{j,i+1}, & j = i \\ a_{i,i+1}a_{j,i+1}, & j = i+1 \\ a_{i,i}a_{j,i}, & j = i-1 \\ 0, & \text{en otro caso.} \end{cases} \tag{2.32}$$

El elemento a_{i+1} puede no existir si $i+1$ supera el tamaño de la matriz. Además, dado que AA^T es simétrica, es suficiente calcular solo una mitad de la matriz, evitando cálculos redundantes, como se refleja en la figura 2.4.

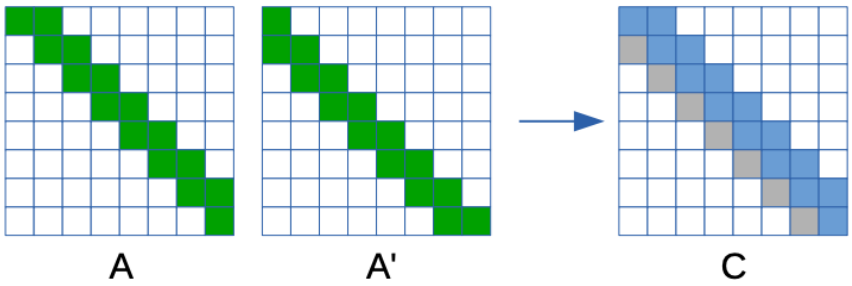


Figura 2.4: Producto matricial de AA^T, con A superior diagonal. En verde aparecen las celdas usadas en el cálculo, en azul las celdas a calcular y en gris las celdas duplicadas. Extraído de [1].

Otro ejemplo importante es la multiplicación por la izquierda de una matriz por una matriz de rotación G de Givens, en la forma GA (véase figura 2.5). Para simplificar el proceso, inicialmente establecemos $C = A$, lo que evita construir C desde cero. Según la definición de G, solo las filas p y q de C cambian debido a la multiplicación, mientras que el resto de las filas permanecen iguales. Además, para cada columna de A, únicamente se utilizan los elementos en las filas p y q.

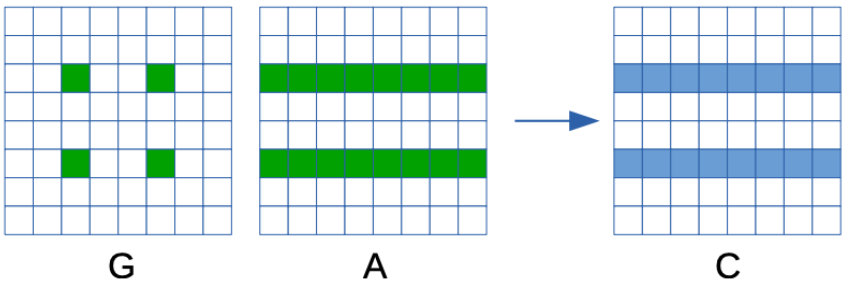


Figura 2.5: Producto matricial para la rotación de Givens. En verde, las celdas empleadas en el cálculo. En azul, las celdas a calcular. Extraído de [1].

Por lo tanto, los únicos valores que deben actualizarse en C son los de las filas p y q, cuyos valores se calculan como:

$$c_{p,j} = g_{p,p}a_{p,j} + g_{p,q}a_{q,j}, \quad c_{q,j} = g_{q,p}a_{p,j} + g_{q,q}a_{q,j}. \tag{2.33}$$

para cada columna j. Una fórmula similar se aplica para multiplicación por la derecha. La cantidad de operaciones necesarias para los productos matriciales, en los distintos casos especiales de matrices, puede evaluarse contando las multiplicaciones escalares realizadas según el tamaño de las matrices de entrada.

Matrix Product	Scalar Products	Code Size (Bytes)
$C = A \cdot B$, generic	n^3	148
$C = A \cdot A'$	$n^3/2 + n^2/2$	210
$C = A \cdot A'$, A upper-diagonal	$3n - 1$	304
$C = G \cdot A$, Givens rotation	$4n$	108

Tabla 2.6: Número de multiplicaciones escalares en productos matriciales para diferentes tipos de matriz $n \times n$. Extraído de [1].

Los resultados se resumen en la tabla 2.6, donde también se incluye el tamaño en bytes del código de las rutinas de software correspondientes.

Además, el efecto de estas optimizaciones en la velocidad de cálculo es medible. La tabla 2.7 presenta el incremento en la velocidad con respecto al tiempo original, sin aplicar dichas optimizaciones, para el conjunto de matrices más grande, utilizando dos algoritmos donde estas optimizaciones resultan especialmente efectivas.

Matrix Size	Golub-Reinsch	Demmel-Kahan
48×24	21%	16%
72×36	18%	16%
96×48	36%	23%
120×60	44%	16%
144×72	18%	16%

Tabla 2.7: Aumento de velocidad de los algoritmos de multiplicación matricial optimizados. Extraído de [1].

Por otro lado, además de tener en cuenta la optimización de los procesos matemáticos, es importante considerar el problema de la pérdida de precisión en operaciones de punto flotante de 32 bits, especialmente para valores pequeños que pueden afectar significativamente los resultados. Una solución típica es usar cálculos en 64 bits para mejorar la precisión, aunque esto aumenta el tiempo de ejecución. Alternativamente, técnicas como trabajar en el dominio logarítmico pueden mantener la precisión con menor coste computacional, aprovechando funciones matemáticas optimizadas.

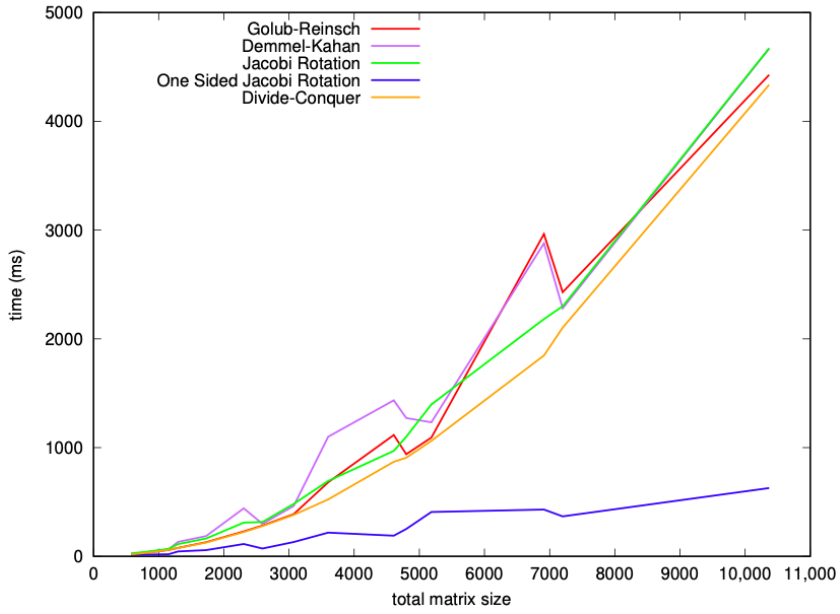


Figura 2.6: Tiempos de los algoritmos SVD en `Cortex-M4F` con respecto al tamaño total de la matriz $m \times n$. Extraído de [1].

La figura 2.6 muestra los tiempos de los algoritmos SVD completos implementados en el microprocesador para los tres conjuntos de matrices con diferentes relaciones fila/columna, con

respecto al tamaño total de la matriz $m \times n$. A través de los resultados experimentales se puede ver que el tiempo de los algoritmos depende en términos generales del tamaño total de la matriz, pero con irregularidades que sugieren que otros factores, como el rango de la matriz, afectan al tiempo total. Es importante resaltar que el algoritmo de rotación unidireccional de Jacobi da el tiempo de ejecución más bajo, en comparación con los demás.

La tabla 2.8 muestra la precisión de las pruebas específicas implementadas en el microcontrolador con respecto a la función `SVD` incorporada de MATLAB. Los errores presentados se calculan como el promedio de los errores relativos entre los valores singulares obtenidos por la función incorporada de MATLAB y los calculados por los algoritmos implementados.

Size	Golub-Reinsch	Demmel-Kahan	Jacobi Rot.	One-Sided J.	Divide-Conquer
24×24	3.8×10^{-7}	7.8×10^{-7}	1.0×10^{-6}	1.9×10^{-7}	2.2×10^{-6}
36×36	4.8×10^{-7}	2.3×10^{-6}	1.7×10^{-6}	3.5×10^{-7}	7.0×10^{-6}
48×48	4.7×10^{-7}	7.1×10^{-6}	3.0×10^{-6}	2.4×10^{-7}	6.6×10^{-6}
60×60	5.3×10^{-7}	4.8×10^{-6}	7.3×10^{-7}	3.0×10^{-7}	4.3×10^{-6}
72×72	4.6×10^{-7}	2.6×10^{-6}	1.7×10^{-6}	3.4×10^{-7}	4.7×10^{-6}
32×24	2.7×10^{-7}	5.7×10^{-7}	2.2×10^{-7}	1.7×10^{-7}	2.9×10^{-7}
48×36	3.6×10^{-7}	2.4×10^{-6}	4.0×10^{-7}	1.7×10^{-7}	2.6×10^{-7}
64×48	2.9×10^{-7}	2.7×10^{-6}	2.9×10^{-7}	1.7×10^{-7}	1.9×10^{-7}
80×60	4.1×10^{-7}	5.9×10^{-6}	5.4×10^{-7}	2.4×10^{-7}	4.0×10^{-7}
96×72	1.6×10^{-6}	6.5×10^{-6}	6.0×10^{-7}	2.7×10^{-7}	3.5×10^{-7}
48×24	3.0×10^{-7}	1.7×10^{-6}	1.6×10^{-7}	1.7×10^{-7}	9.5×10^{-8}
72×36	2.7×10^{-7}	6.5×10^{-7}	3.7×10^{-7}	1.5×10^{-7}	1.4×10^{-7}
96×48	1.0×10^{-6}	8.7×10^{-6}	4.1×10^{-7}	1.8×10^{-7}	1.5×10^{-7}
120×60	5.9×10^{-7}	3.9×10^{-6}	4.8×10^{-7}	2.0×10^{-7}	1.5×10^{-7}
144×72	4.8×10^{-7}	5.3×10^{-6}	5.8×10^{-7}	3.1×10^{-7}	1.3×10^{-7}

Tabla 2.8: Errores promedio de los algoritmos SVD en `Cortex-M4F` vs. la función SVD de MATLAB.

Se puede notar que la precisión de la implementación en el microcontrolador es considerablemente menor que la del código equivalente en MATLAB. Esto se debe a la menor precisión, de 32 bits, de la unidad de punto flotante del microcontrolador. En este caso, tanto el algoritmo de rotación de Jacobi unidireccional logra una precisión mejor que otros algoritmos.

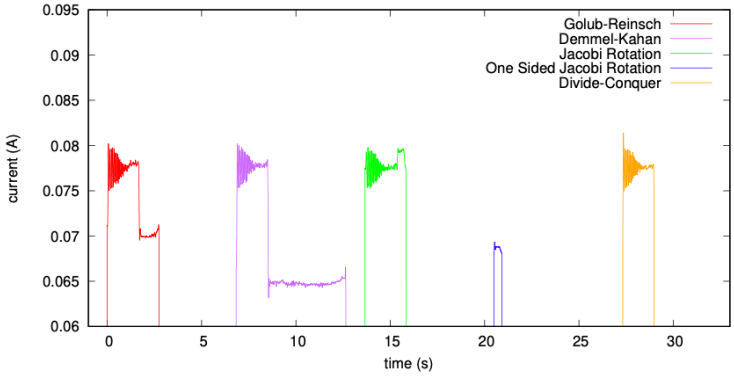


Figura 2.7: Consumo de corriente de los algoritmos SVD en el `Cortex-M4F` para la matriz de dimensión 96×72. Extraído de [1].

A modo de comparación, en la figura 2.7 se muestra el consumo de corriente de los cinco algoritmos para una de las matrices. El algoritmo de rotación de Jacobi unidireccional, además de ser más rápido, presenta claramente el menor consumo promedio de corriente. Es importante señalar que los otros algoritmos muestran el mismo patrón al inicio, lo que corresponde al paso de la bidiagonalización de Householder. Este paso, por lo tanto, tiene una relevancia significativa tanto en términos de tiempo como de energía, para aquellos algoritmos que lo requieren.

Además, se desarrollaron implementaciones propias de los algoritmos de Jacobi y Demmel-Kahan en MATLAB, lo que permitió contrastar los resultados obtenidos con las herramientas usadas en otros entornos, evaluando así su rendimiento y precisión. El código completo se puede consultar en las secciones A.4 y A.5 del Apéndice. Con el objetivo de realizar una comparativa sencilla de estos dos algoritmos implementados en MATLAB, se han estudiado los tiempos de compilado para matrices de distintas dimensiones. Los resultados se muestran en la figura 2.8.

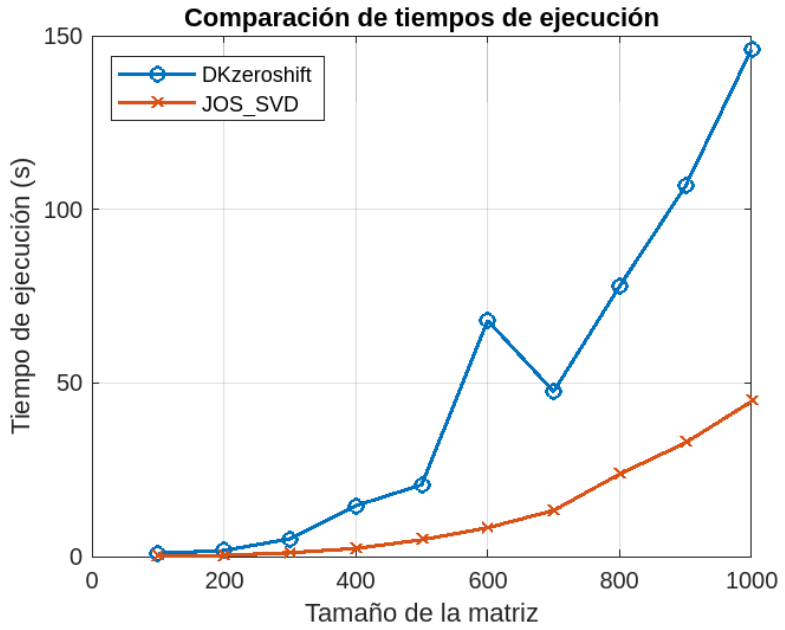


Figura 2.8: Comparativa de los tiempos de ejecución de los algoritmos de Jacobi unidireccional y Demmel-Kahan sin desplazamiento.

En general, ambos algoritmos muestran un crecimiento en su tiempo de ejecución conforme aumenta el tamaño de la matriz, pero con diferencias significativas en su escalabilidad. Para matrices pequeñas ($n < 200$), los tiempos son comparables, aunque Jacobi unidireccional es ligeramente más eficiente. Sin embargo, para tamaños mayores ($n > 400$), la diferencia se amplifica: Demmel-Kahan tiene un crecimiento no lineal en su tiempo de ejecución, mientras que Jacobi presenta un incremento más moderado. Para $n \approx 1000$, el tiempo de Demmel-Kahan puede ser más del doble que el de Jacobi, lo cuál parece indicar su menor eficiencia en altas dimensiones.

Esto se debe a que Demmel-Kahan incluye una etapa de transformación bidiagonal y rotaciones de Givens que incrementan su coste computacional. En contraste, Jacobi evita este paso inicial, lo que lo hace más eficiente y adecuado para manejar matrices grandes. Por otro lado, las matrices generadas aleatoriamente tendrán números de condición muy variables, lo cuál es un factor de gran relevancia a la hora de considerar un método u otro.

Capítulo 3

Una Introducción a las Aplicaciones de la SVD

En este capítulo, se aborda una introducción a las diversas aplicaciones de la SVD. Se inicia con los **Métodos de Filtrado Espectral**, examinándose cómo la SVD facilita la separación de componentes en el procesamiento de señales y la mejora de la calidad de imágenes.

La sección de **Reconocimiento Facial** muestra la aplicación de la SVD en la identificación de patrones y características faciales, fundamentales en los sistemas de seguridad y en el análisis de imágenes biométricas.

A continuación, se explora el **Algoritmo GoDec**, presentándose este método como una aproximación eficaz para la descomposición de matrices de rango bajo y su aplicación en la compresión de datos y la recuperación de imágenes.

En la parte dedicada a la **Detección de Movimiento en Vídeos**, se destaca el uso de la SVD para identificar y seguir objetos en movimiento dentro de secuencias de vídeo, técnica clave en la vigilancia y en aplicaciones multimedia.

Finalmente, se discute la incorporación de la **SVD en Redes Neuronales**, evidenciándose su contribución al entrenamiento de modelos más eficientes y a la reducción de dimensionalidad de datos de alta complejidad. A lo largo de este capítulo, se mostrará la importancia de la SVD como herramienta transversal para el análisis de datos, la ingeniería de software y el desarrollo de tecnologías avanzadas.

3.1. Métodos de Filtrado Espectral

Siguiendo a [12], se presenta el problema del desenfocado de imágenes, describiendo cómo el proceso de desenfocado puede representarse como un problema lineal y cómo este modelo se utiliza para intentar restaurar la imagen original.

Las imágenes digitales se representan como matrices de números donde cada valor (o intensidad) en la matriz corresponde a la *luminosidad* de un píxel. Los valores de intensidad en una imagen en escala de grises forman una matriz $X \in \mathbb{R}^{m \times n}$. Las imágenes en color se pueden

descomponer en componentes RGB, rojo, verde y azul, aunque por simplicidad nos centraremos en imágenes en escala de grises.

Una imagen borrosa puede expresarse mediante una transformación lineal de la matriz de la imagen original. En este modelo simple, el desenfoque actúa primero en las columnas y luego en las filas de la imagen:

$$A_c X A_r^T = B, \tag{3.1}$$

donde:

- $A_c \in \mathbb{R}^{m\times m}$ y $A_r \in \mathbb{R}^{n\times n}$ son matrices de desenfocado que actúan vertical y horizontalmente sobre la imagen.
- B es la imagen borrosa o desenfocada.

La multiplicación por A_c a la izquierda aplica un desenfocado a todas las columnas de X, y la multiplicación por A_r^T a la derecha aplica un desenfocado a todas las filas. En este modelo, el desenfocado se representa mediante operaciones matriciales, haciendo de la imagen borrosa una función de la imagen original X y de las matrices de desenfocado A_c y A_r.

Dado el modelo de desenfocado (3.1), la restauración de la imagen desenfocada podría interpretarse como una inversión de las matrices A_c y A_r, que se suponen no singulares, para obtener la imagen original X,

$$\bar{X} = A_c^{-1} B (A_r^T)^{-1}.$$

Sin embargo, esta aproximación resulta ineficaz porque amplifica el ruido, es decir, intensifica el error. En general, el modelo lineal debe tener en cuenta el ruido, por lo que la imagen borrosa se modela mejor como:

$$B = A_c X A_r^T + E, \tag{3.2}$$

donde E representa el ruido, es decir, pequeñas fluctuaciones y errores de cuantización en el proceso. Entonces, el intento de reconstrucción de la imagen previo se convierte en:

$$\bar{X} = A_c^{-1} B (A_r^T)^{-1} = X + A_c^{-1} E (A_r^T)^{-1}.$$

Debido a que A_c^{-1} y $(A_r^T)^{-1}$ aumentan el ruido en E, este modelo no logra eliminar el desenfocado y produce una imagen con ruido significativo.

En una formulación más general, el desenfocado se representa mediante una matriz A de grandes dimensiones, que actúa directamente sobre el vectorizado de la imagen original. Se define la función *vec*, que convierte la matriz de la imagen X en un vector columna $x \in \mathbb{R}^N$ (donde $N = m \cdot n$) apilando las columnas de X. Entonces, la relación de desenfocado se expresa como:

$$Ax = b, \tag{3.3}$$

donde:

- $b = vec(B)$ representa el vector de la imagen borrosa,
- $A \in \mathbb{R}^{N\times N}$ es la llamada *matriz de desenfocado*.

La reconstrucción de la imagen original mediante la inversión directa de A no es viable debido al efecto de *ruido invertido*. Al resolver x como solución de $Ax = b$, con A no singular, el término de ruido e en $b = b_{exacto} + e$ se amplifica, por lo que la solución se convierte en:

$$\bar{x} = A^{-1}b = A^{-1}b_{exacto} + A^{-1}e = x + A^{-1}e.$$

El término $A^{-1}e$ se denominará *ruido invertido*, y éste provocará que la solución obtenida esté dominada por el ruido.

Ahora bien, para entender el impacto del ruido en la solución, se emplea la SVD. Una representación útil de la SVD de la matriz A, que se supone no singular, será la siguiente:

$$A = \sum_{i=1}^{N} \sigma_i \mathbf{u}_i \mathbf{v}_i, \qquad A^{-1} = \sum_{i=1}^{N} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T \ . \tag{3.4}$$

Los vectores singulares $\mathbf{v}_i$ actúan como una base para representar las componentes espectrales. Las frecuencias altas están asociadas con vectores singulares que tienen más oscilaciones y corresponden a σ_i pequeños. Las frecuencias bajas, por otro lado, están representadas por vectores singulares con σ_i grandes y menos variaciones. Despejando x en $Ax = b$, y teniendo en cuenta la expresión (3.4), obtenemos:

$$x = \sum_{i=1}^{N} \frac{\mathbf{u}_i^T b}{\sigma_i} \mathbf{v}_i. \tag{3.5}$$

Aquí, cada término $\frac{\mathbf{u}_i^T b}{\sigma_i}\mathbf{v}_i$, representa la contribución de la i-ésima componente singular a la solución. El problema del desenfocado con ruido ocurre porque los valores singulares σ_i disminuyen rápidamente, lo que hace que $1/\sigma_i$ se vuelva muy grande y amplifique las partes del ruido e asociadas a esas componentes. Por tanto, la solución $\bar{x} = A^{-1}b$ resulta dominada por el ruido invertido:

$$\bar{x} = \sum_{i=1}^{N} \frac{\mathbf{u}_i^T b_{\text{exacto}}}{\sigma_i} \mathbf{v}_i + \sum_{i=1}^{N} \frac{\mathbf{u}_i^T e}{\sigma_i} \mathbf{v}_i. \tag{3.6}$$

El segundo término, que depende de e, es especialmente problemático porque las frecuencias altas[1], representadas por vectores singulares $\mathbf{v}_i$ asociados con valores σ_i pequeños, están dominadas por ruido.

Esto se observa mediante el análisis espectral, que revela que:

1. Las componentes espectrales de baja frecuencia ($\mathbf{v}_i$ con valores σ_i grandes) están asociadas con detalles generales de la imagen.

2. Las componentes espectrales de alta frecuencia ($\mathbf{v}_i$ con valores σ_i pequeños) tienden a ser dominadas por el ruido.

Para mitigar estos problemas, se introducen métodos de filtrado espectral que modifican o eliminan la contribución de los valores singulares pequeños. Dos enfoques comunes son:

[1]En el contexto del procesamiento de imágenes, las frecuencias altas representan cambios rápidos en la intensidad de los píxeles, como bordes, líneas delgadas o texturas pequeñas. Estas corresponden a transiciones bruscas en la imagen, donde los valores de intensidad varían significativamente entre píxeles vecinos.

1. **Truncamiento en SVD (TSVD):** En lugar de utilizar todos los valores singulares, se selecciona un umbral k y se considera solo los k valores singulares más grandes, (véase el capítulo 1 de esta memoria)

$$x_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^T b}{\sigma_i} \mathbf{v}_i.$$

   Esto permite incluir únicamente las componentes dominadas por datos útiles, mientras se descartan aquellas contaminadas por ruido. Sin embargo, la elección del parámetro k es crítica: valores bajos producen imágenes *sobre-suavizadas*, mientras que valores altos pueden reintroducir ruido.

2. **Filtrado general con factores de filtro ϕ_i:** Se introduce un filtro ϕ_i que depende de los valores singulares:

$$x_\phi = \sum_{i=1}^{N} \phi_i \frac{\mathbf{u}_i^T b}{\sigma_i} \mathbf{v}_i.$$

   Los valores de ϕ_i son elegidos de manera que reduzcan la influencia de los valores singulares pequeños, donde:

$$\phi_i \approx \begin{cases} 1 & \text{para } \sigma_i \text{ grandes,} \\ 0 & \text{para } \sigma_i \text{ pequeños.} \end{cases}$$

Condición de Picard Discreta

La condición de Picard establece que, para separar correctamente la señal del ruido, los coeficientes espectrales $\mathbf{u}_i^T b$ deben decaer más rápido que los valores singulares σ_i, lo cual se expresa como:

$$|\mathbf{u}_i^T b| \ll \sigma_i \quad \text{para valores grandes de i.}$$

Esto asegura que el ruido no domine en la solución, ya que las frecuencias altas, que contienen ruido, están asociadas a valores singulares pequeños.

La condición de Picard asegura que la contribución del ruido invertido en (3.6) sea mucho menor que la de la primera componente, asociada a (3.5). Esto se logra garantizando que los coeficientes $\mathbf{u}_i^T b_{\text{exacto}}$ sean mucho mayores que los coeficientes del ruido $\mathbf{u}_i^T e$. Visualmente, esta condición implica que los coeficientes espectrales deben decaer más rápido que los valores singulares σ_i, lo que asegura que el ruido tenga una menor influencia en la solución final.

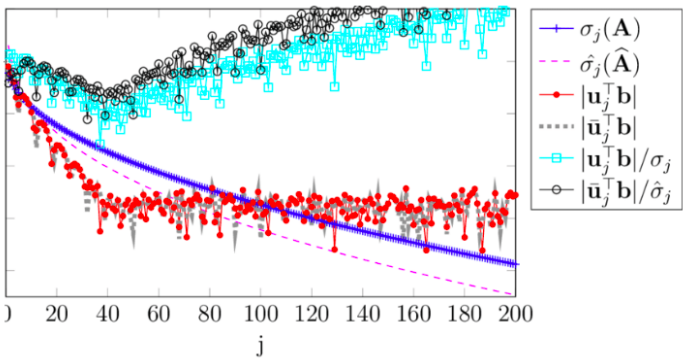


Figura 3.1: Gráfico de Picard Discreto correspondiente a la SVD y a la GSVD. Extraído de [4].

La condición de Picard también ayuda a determinar un índice k donde la transición de las componentes espectrales hacia el nivel del ruido ocurre. Si k es lo suficientemente grande, el ruido no afecta a la solución. Esta condición es esencial para el filtrado espectral, ya que permite separar las frecuencias bajas, dominadas por la señal útil, de las frecuencias altas, que están dominadas por el ruido. Elegir un valor adecuado de k es crucial para filtrar el ruido sin perder demasiada información relevante de la señal.

Finalmente, presentamos un caso ilustrativo del cumplimiento de la condición de Picard discreta en la figura 3.1. Los gráficos de Picard discretos correspondientes a la SVD y a la GSVD, Descomposición Generalizada en Valores Singulares, muestran que los valores singulares generalizados se descomponen más rápido que los valores singulares de A. Tanto los coeficientes de SVD como los de GSVD decaen más rápido que los valores singulares y los valores singulares generalizados, y luego se estabilizan en el nivel de ruido. Los coeficientes de la solución disminuyen al principio, pero eventualmente aumentan debido a la contaminación por ruido.

Además, también hemos implementado un programa MATLAB que analiza matrices de desenfocado generadas aleatoriamente, evaluando el impacto de diferentes niveles de ruido y tamaños de matriz en la SVD (véanse las figuras 3.2, 3.3 y 3.4). El programa aplica el método de regularización TSVD para mejorar la reconstrucción de soluciones aproximadas y genera gráficos que muestran cómo los valores singulares y los coeficientes espectrales varían con estos parámetros. Se resumen los resultados obtenidos:

- **Aumento del ruido**: Incrementar el ruido de 0.001 a 1 aumenta la dispersión de los valores singulares y los coeficientes, haciendo que se desvíen más de los valores originales, tanto con desenfoque aleatorio como gaussiano.
- **Número de puntos (n)**: Al pasar de $n = 50$ a $n = 100$, se observa que el incremento del ruido mantiene la misma tendencia de mayor dispersión.
- **Tipo de desenfocado**: La diferencia entre desenfocado aleatorio y gaussiano tiene un impacto menor en comparación con el nivel de ruido. Ambos tipos de desenfocado muestran patrones similares cuando el ruido aumenta.

En resumen, el nivel de ruido es el factor más significativo, causando una mayor dispersión de los valores singulares y los coeficientes, independientemente del tipo de desenfoque y del número de puntos (n).

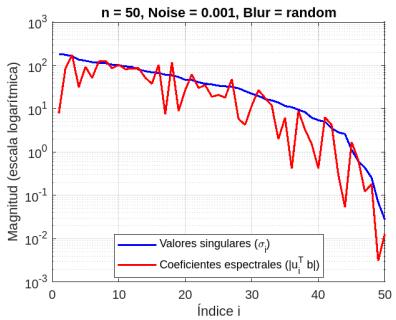


Figura 3.2: Filtrado 1: $n = 50$, Ruido = 0.001, Desenfoque = aleatorio.

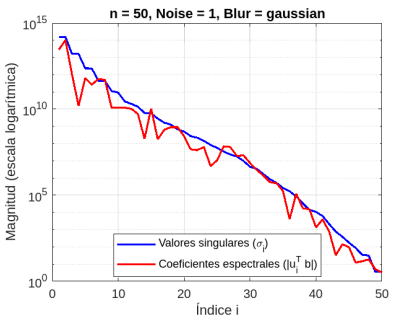


Figura 3.3: Filtrado 2: $n = 50$, Ruido = 1, Desenfoque = gaussiano.

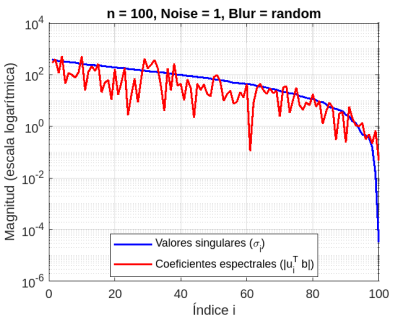


Figura 3.4: Filtrado 3: $n = 100$, Ruido = 1, Desenfoque = aleatorio.

3.2. Reconocimiento Facial

El reconocimiento facial es una aplicación relevante de la descomposición en valores singulares en el procesamiento de imágenes. Esta técnica se basa en la representación de imágenes faciales mediante un espacio de características derivado de los vectores singulares principales, denominados *rostros-base*. A continuación, se describe el procedimiento general de este método.

Consideremos un conjunto de N imágenes faciales, cada una de tamaño $m \times n$ píxeles. Estas imágenes se representan como vectores columna $f_i \in \mathbb{R}^{mn}$, reorganizando cada imagen en un vector de tamaño $mn \times 1$. La matriz $S \in \mathbb{R}^{mn \times N}$, que contiene todas las imágenes del conjunto, se construye como:

$$S = \begin{bmatrix} f_1 & f_2 & \cdots & f_N \end{bmatrix}.$$

El *rostro promedio*, $\bar{f}$, se calcula como:

$$\bar{f} = \frac{1}{N} \sum_{i=1}^{N} f_i,$$

y no es más que una columna resultado del promedio de todas las columnas que conforman S. A partir de S y $\bar{f}$, se define la matriz $A \in \mathbb{R}^{mn \times N}$, donde cada columna corresponde a una imagen centrada:

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_N \end{bmatrix}, \quad a_i = f_i - f \quad \forall i \in \{1,\ ...,\ N\}.$$

Pues bien, construimos la descomposición SVD de A:

$$A = U \Sigma V^T, \tag{3.7}$$

donde $U \in \mathbb{R}^{mn \times mn}$ y $V \in \mathbb{R}^{N \times N}$ son matrices ortogonales, y $\Sigma \in \mathbb{R}^{mn \times N}$ es una matriz diagonal con los valores singulares no nulos $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ en sus entradas principales.

El subespacio generado por las columnas de $U_r = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix}$, con $r = \text{rango}(A)$, forma una base ortonormal del espacio generado por las columnas de A, y se conoce como el *espacio de rostros*. Cada u_i es un *rostro-base*.

Ahora bien, dado un nuevo rostro f, de tamaño $m \times n$ y convertido en un vector columna de tamaño $mn \times 1$, podemos calcular sus coordenadas en el espacio generado por las columnas de U_r. Para esto, necesitamos encontrar un vector $w = (w_1, w_2, \ldots, w_r)^T$, donde cada w_i representa la contribución de la columna u_i en la representación de $f - \bar{f}$. Esto se puede escribir como:

$$f - \bar{f} = w_1 u_1 + w_2 u_2 + \cdots + w_r u_r.$$

Es decir, el vector $f - \bar{f}$ es una combinación lineal de las columnas de U_r. En forma matricial, esto se puede expresar como:

$$f - \bar{f} = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_r \end{bmatrix}.$$

Para hallar w, basta con multiplicar por la traspuesta de U_r, debido a la ortogonalidad de esta matriz:

$$w = U_r^T(f - \bar{f}) = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix}^T (f - \bar{f}).$$

De forma similar, las coordenadas de cada rostro de entrenamiento f_i en el espacio de rostros están dadas por:

$$x_i = U_r^T(f_i - \bar{f}) = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix}^T (f_i - \bar{f}), \quad \text{para } i = 1, \ldots, N.$$

El reconocimiento de la nueva imagen f se realiza midiendo la distancia euclídea entre w y cada uno de los vectores x_i asociados a las imágenes de entrenamiento.

$$d_i = \|w - x_i\|_2, \quad \text{para } i = 1, \ldots, N.$$

Denotamos por $j = \arg\min_i d_i$ el índice asociado a la mínima de estas distancias.

Finalmente, se compara d_j con un umbral predefinido ε_0 para decidir si la imagen corresponde a una de las imágenes conocidas o si es un rostro desconocido. La decisión se toma de la siguiente manera:

- Si $d_j < \varepsilon_0$, se clasifica f como la imagen f_j correspondiente.
- Si $d_j \geq \varepsilon_0$, f se clasifica como un rostro desconocido.

Según [7], con el objetivo de ilustrar este algoritmo, se realizó un experimento con una base de datos de 500 imágenes de rostros convertidas a escala de grises y redimensionadas a 156×111 píxeles. Los experimentos mostraron alta precisión en la identificación, destacando la validez del método frente a variaciones en expresión facial y posición.

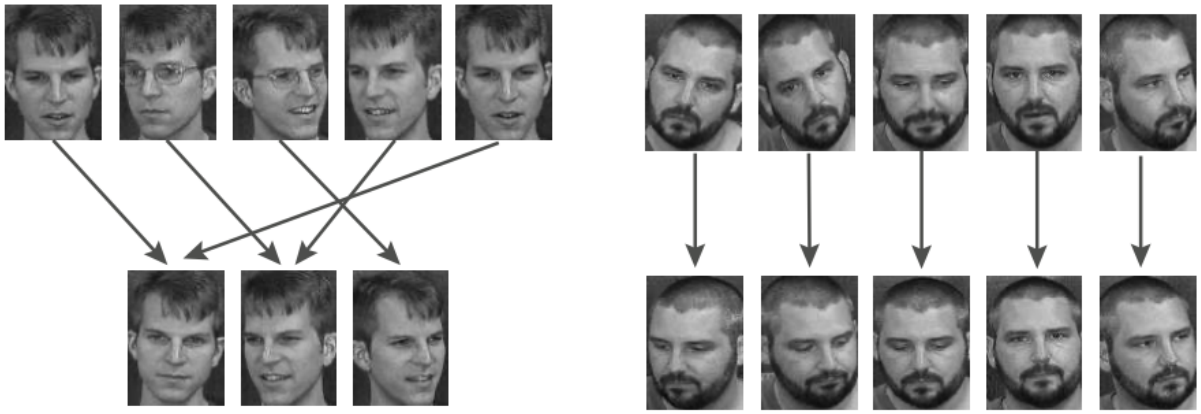


Figura 3.5: Resultados obtenidos al aplicar el algortimo de reconocimiento facial en el experimento realizado por [7].

La figura 3.5 presenta dos ejemplos específicos de los resultados obtenidos al aplicar el algoritmo. En la parte superior de cada ejemplo, se muestran cinco imágenes que no fueron consideradas entre las imágenes de entrenamiento. En la parte inferior, se encuentran las imágenes para testar la identificación. Las flechas indican la relación entre la imagen detectada por el algoritmo y la correspondiente en cada caso.

Otro experimento realizado fue el reflejado en [18], en el que se utilizó la base de datos ORL[2], compuesta por 400 imágenes de 40 individuos, con 10 imágenes por persona. Estas se dividieron en un conjunto de entrenamiento con 200 imágenes y otro de prueba con las 200 restantes. Para la reducción de dimensionalidad, se seleccionaron las primeras $k = 8$ autocaras, derivadas de los vectores singulares principales, las cuales definieron la base del subespacio de características. Este enfoque permitió representar las imágenes de manera compacta y eficiente.

Durante las pruebas, el sistema alcanzó una tasa de acierto del 86 % en la identificación de las imágenes de prueba. Posteriormente, con métodos mejorados como el 2D-PCA[3], esta precisión se incrementó al 94 %. Las autocaras obtenidas en el proceso reflejan las características más relevantes de los rostros, y las primeras cuatro se presentan en la figura 3.6. Este experimento es una implementación del método clásico desarrollado por Turk y Pentland (1991), documentado en *Face Recognition Using Eigenfaces*, adaptado para pruebas con el conjunto de datos ORL.

Figura 3.6: Algunas de las imágenes de entrenamiento de ORL, extraídas de [18].

[2]La base de datos ORL, creada por los laboratorios ATT de la Universidad de Cambridge, contiene 400 imágenes faciales de 40 individuos. Cada persona está representada por 10 imágenes con variaciones en expresión, posición y accesorios como gafas. Es ampliamente usada en investigaciones sobre reconocimiento facial.

[3]El 2D-PCA (*Two-Dimensional Principal Component Analysis*) es una extensión del PCA que opera directamente sobre matrices en lugar de vectores, conservando la estructura espacial de las imágenes. Esto permite una mejor precisión en tareas de reconocimiento facial, como se demostró en investigaciones que lograron tasas de acierto superiores al 90 %.

3.3. El Algoritmo GoDec

El algoritmo GoDec (*Go Decomposition*) es un método iterativo que permite descomponer una matriz A en dos componentes principales, $A \approx L+S$, con L una matriz de rango reducido y una matriz S dispersa. Formalmente, el algoritmo busca resolver el problema de optimización:

$$\min_{\text{rango}(L)\leq k,\, \text{card}(S)\leq c_0} \|A - L - S\|_F^2, \tag{3.8}$$

donde $\|\cdot\|_F$ es la norma de Frobenius, $\text{card}(S)$ representa el número de elementos no nulos en S, k es el rango máximo permitido para L, y c_0 es la cardinalidad máxima permitida para S. Esta matriz A también se suele descomponer en la suma siguiente:

$$A = L + S + G,$$

donde G es una matriz de ruído que modela el error de aproximación.

Como se expone en [7], el algoritmo GoDec se basa en los principios de la SVD y el teorema de Eckart-Young, el cual establece que la mejor aproximación de rango k de una matriz A está dada por:

$$L_k = \sum_{i=1}^{k} \sigma_i u_i v_i^\top = U_k \Sigma_k V_k^\top,$$

donde $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$ son los k valores singulares más grandes de A, y $U_k,\ \Sigma_k,\ V_k$ son las matrices truncadas correspondientes a los vectores y valores singulares. Esta propiedad asegura que:

$$\min_{\text{rango}(L)\leq k} \|A - L\|_F^2 = \|A - L_k\|_F^2. \tag{3.9}$$

Y además, la matriz L_k es única si y solo si $\sigma_k > \sigma_{k+1}$.

El algoritmo GoDec, en cada iteración t, calcula dos matrices, L_t y S_t, donde L_t es una matriz de rango reducido ($\text{rango}(L_t) \leq k$) y S_t es una matriz dispersa con un número máximo de entradas no nulas ($\text{card}(S_t) \leq c_0$). Estas matrices se actualizan iterativamente, y las sucesiones $\{L_t\}$ y $\{S_t\}$ convergen de forma lineal hacia un mínimo local del problema de optimización.

Para garantizar que S_t sea dispersa, el algoritmo utiliza una función de proyección P_{c_0}, que selecciona las c_0 entradas de mayor magnitud, en valor absoluto, de una matriz dada y coloca ceros en el resto de las entradas. Este enfoque asegura que S_t cumpla con la restricción previa mientras mantiene las características más significativas de los datos.

Pues bien, en cada iteración t, se calcula:

1. Una aproximación L_t a L, basada en una SVD truncada de $A - S_{t-1}$, es decir:

   $$[U, \Sigma, V^\top] = \text{svd}(A - S_{t-1}), \qquad L_t = U_k \Sigma_k V_k^\top,$$

   donde U_k, Σ_k, V_k corresponden a los k valores singulares principales.

2. Una actualización de S_t basada en el operador P_{c_0}, definido como:

   $$P_{c_0}(X) = \underset{Y:\text{card}(Y)\leq c_0}{\arg\max} \|X - Y\|_F^2.$$

Este operador preserva las c_0 entradas de mayor magnitud en valor absoluto en X, asignando cero al resto. Aplicando dicho operador a la matriz $A - L_t$, se obtiene S_t.

3. El error relativo de la iteración, calculado como $E_t = \frac{\|A-L_t-S_t\|_F^2}{\|A\|_F^2}$.

El algoritmo detiene las iteraciones cuando el cambio relativo en E_t entre dos iteraciones consecutivas cae por debajo de un umbral ϵ, garantizando así la convergencia hacia un mínimo local del problema de optimización.

A modo ilustrativo, se ha implementado un código en MATLAB que ejecuta el algoritmo *GoDec*, cuyo comportamiento se muestra en la figura 3.7. La función `godec_multiple_matrices` aplica el algoritmo GoDec a varias matrices de prueba, como matrices aleatorias, ortogonales y de correlación, y grafica el error relativo a lo largo de las iteraciones. Durante cada iteración, se calcula el error relativo entre la matriz original A y su aproximación $L + S$, mostrando cómo el algoritmo converge.

La función `godec` implementa el algoritmo, tomando como entrada la matriz A y parámetros como el rango reducido k, el factor de cardinalidad c_0, el número máximo de iteraciones `max_iter` y la tolerancia ϵ. La salida incluye el vector de errores, la matriz de bajo rango L y la matriz dispersa S. La función `sparse_projection` se encarga de la proyección dispersa.

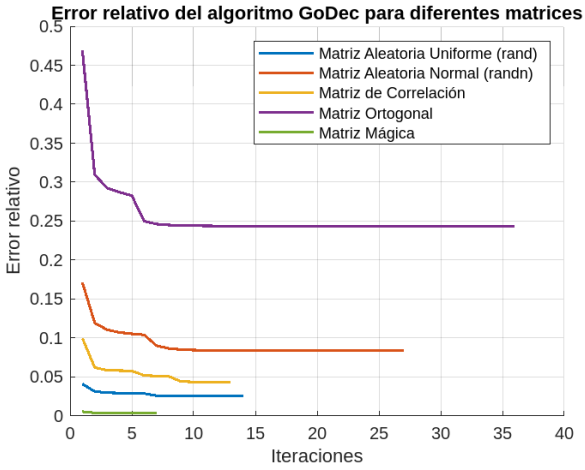


Figura 3.7: Evolución del error relativo durante las iteraciones del algoritmo GoDec para diferentes matrices de prueba. El código puede ser consultado en el Ápendice A.6.

En este ejemplo, el algoritmo GoDec muestra una rápida convergencia inicial para todas las matrices, con menor error final en las matrices aleatorias (uniforme y normal). La matriz ortogonal converge más lentamente y estabiliza en un error mayor ($\approx$0.25), mientras que la matriz mágica alcanza un error intermedio ($\approx$0.05).

3.3.1. Aplicación del algoritmo GoDec al procesamiento de videos

El algoritmo GoDec, diseñado para descomponer matrices como la suma de una componente de rango bajo L y una componente dispersa S, encuentra aplicaciones significativas en el procesamiento de videos. En este contexto, cada video se representa como una matriz $A \in \mathbb{R}^{mn\times K}$, donde K es el número de *frames*, y mn corresponde al número de píxeles por *frame*. La i-ésima

columna de A se forma vectorizando el i-ésimo *frame* del video, es decir, reorganizando los píxeles de la imagen bidimensional correspondiente al *frame* en un único vector columna.

El objetivo de la aplicación del algoritmo en este contexto es separar A en L, que modela el fondo estático o de baja variabilidad, y S, que contiene los cambios dinámicos en los *frames*, típicamente asociados con los objetos en movimiento.

En el experimento presentado en [7], se utilizó el algoritmo GoDec para procesar un video compuesto por $K = 60$ *frames* de resolución 540×960 píxeles, reorganizados en una matriz A de tamaño 518400×60. La matriz L, resultado de la descomposición, representa el fondo estimado, mientras que S captura los elementos dinámicos del video, como los objetos en movimiento.

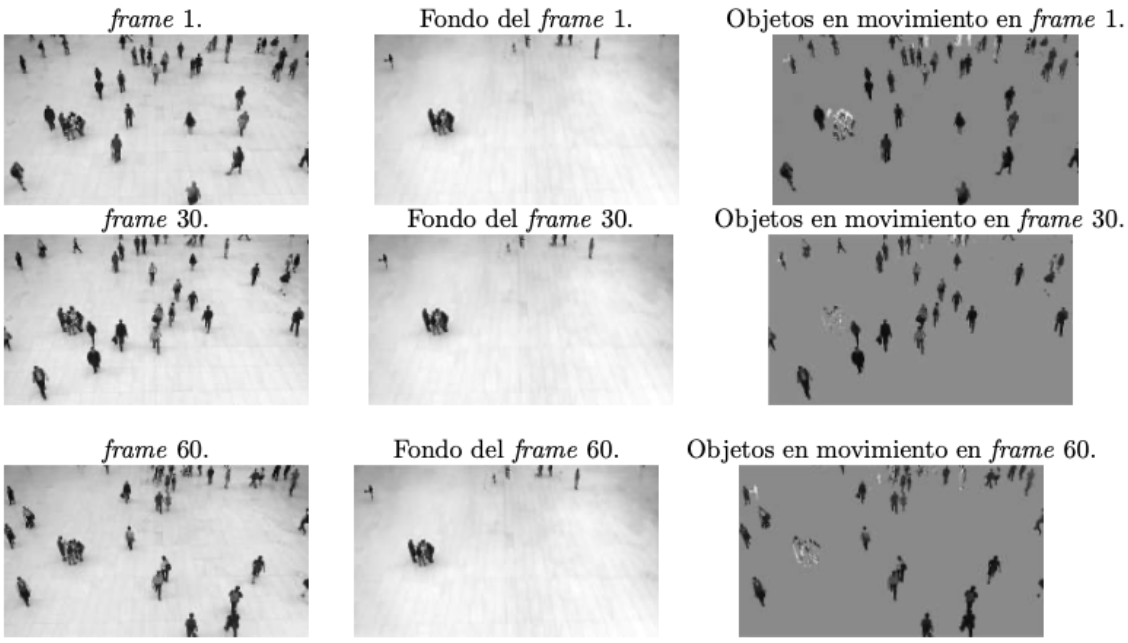


Figura 5: Procesamiento de un vídeo de 60 *frames* con el algoritmo *GoDec*.

Figura 3.8: Procesamiento de un video compuesto por 60 *frames* utilizando el algoritmo GoDec. Imagen extraída de [7].

El algoritmo se inicializó con un rango $k = 2$, correspondiente a las principales componentes del fondo, valor que se eligió bajo por permanecer el fondo relativamente invariante durante el vídeo, y con una tolerancia de convergencia $\epsilon = 10^{-9}$. La dispersión permitida en S se controló mediante el parámetro $c_0 = \lfloor 0{,}07 \cdot mn \cdot K \rfloor = 2177280$, asegurando que S capturase únicamente los cambios significativos. La iteración del algoritmo continuó hasta que la variación relativa entre iteraciones consecutivas, medida como $\|A - L - S\|_F / \|A\|_F$, cayera por debajo de ϵ.

Es importante conocer bien el vídeo específico que se va a procesar con el objetivo de introducir los parámetros más adecuados. Por ejemplo, en casos donde el fondo es altamente dinámico o los objetos en movimiento ocupan grandes áreas, S puede no separar completamente las regiones de interés. Ajustar el parámetro de dispersión c_0 y el rango k es crucial para optimizar los resultados en este tipo de aplicaciones.

3.4. La SVD en Redes Neuronales

La última aplicación mencionada en este capítulo describe el uso de la SVD en redes neuronales. Para su presentación, damos algunas definiciones previas ([24, 2, 14]).

Definición 3.4.1 (**Red Neuronal**)**.** Sea K un cuerpo. Una red neuronal $\mathcal{N}$ sobre el cuerpo K es una 3-tupla $\mathcal{N} := (\mathcal{G}, \mathcal{A}, \Phi)$ tal que:

i) $\mathcal{G} = (V, E)$ es un grafo orientado, donde V y E denotan los conjuntos de vértices y aristas, respectivamente. Los vértices vienen dados por

$$V = \{(i, j) : 0 \leq i \leq l, 0 \leq j \leq L_i\}.$$

Para cada vértice (i, j) del grafo, el índice i denota la profundidad del vértice y el índice j, con $1 \leq j \leq L_i$, corresponde a una enumeración del conjunto N_i de todos los vértices de profundidad i donde $L_i = card(N_i)$. Al conjunto N_i se le llama **capa de profundidad** i. Se denotará $L := \sum_{i=0}^{l} L_i$ la cantidad total de vértices de la red neuronal, también llamada talla de $\mathcal{N}$. Se define el Fan-in(v) de cada $v \in V$ como el conjunto de vértices de la capa anterior conectados con v por una arista de E, también denominado abanico de entrada.

ii) $\mathcal{A}$ es una clase de funciones $f : D(f) \subseteq K \to K$, donde $D(f)$ denota el dominio de f. La clase $\mathcal{A}$ se denominará clase de funciones de activación de $\mathcal{N}$.

iii) Cada arista de E tiene asignado un parámetro. El conjunto de todos los parámetros asociados a la red $\mathcal{N}$ se suele denotar

$$P_{\mathcal{N}} := \{A_v^{\mu} : v \in V, \mu \in \text{Fan-in}(v)\}.$$

iv) Los vértices de profundidad 0 se denominan vértices de entrada y están asociados a un número finito de variables $\{X_1, ..., X_n\}$ que tomarán valores en los dominios de la clase $\mathcal{A}$ precedente. Es decir, se tiene la siguiente asignación:

$$\Phi_0 : \mathcal{N}_0 \to \{X_1, ..., X_n\} \cup \{1\}$$

$$(0, 0) \to 1, \quad (0, j) \to X_j \ .$$

v) El tercer elemento de la 3-tupla Φ es una aplicación $\Phi : V \to \mathcal{A} \times P_{\mathcal{N}}^* \cup \{1, X_1, ..., X_n\}$, donde $P_{\mathcal{N}}^*$ son las palabras finitas sobre $P_{\mathcal{N}}$ como alfabeto y la función Φ viene designada por las reglas siguientes:

$$\Phi(v) := \begin{cases} (f_v, \{A_v^{\gamma} : \gamma \in \text{Fan-in}(v)\}) \in (\mathcal{A} \times P_{\mathcal{N}}^*) & \text{si d}(v) \geq 1 \\ \Phi_0(v) & \text{si } v = (0, j), 0 \leq j \leq n. \end{cases}$$

Una **neurona** puede ser definida como el par $(v, \Phi(v))$ con vértice $v \in V$.

Definición 3.4.2 (**Redes Neuronales Multicapa**)**.** Los MLP o Perceptrones multicapa consisten en la concatenación de varias capas dentro de las cuales hay n número de neuronas con la misma función de activación, que siguen la arquitectura del Perceptrón, dando lugar a la conocida como red neuronal profunda. Las entradas de una capa son las salidas de la capa anterior sin tener en cuenta la primera capa cuyas entradas son los datos introducidos a la red y la última capa cuya salida es la salida de la red.

Por último, se debe mencionar que, en la capa de salida de las redes neuronales, la función de activación suele ser distinta a la del resto de capas y esta se escoge dependiendo del problema a resolver. Por ejemplo, para resolver problemas de clasificación binaria, se suele utilizar la función de activación sigmoide, que proporciona valores dentro del intervalo [0,1]. Para problemas de clasificación multiclase se puede usar la función de activación *Softmax*[4] . Para resolver problemas de regresión, se suele utilizar la función de activación lineal, solo en la capa de salida.

A continuación, se presenta un ejemplo de Red Neuronal Multicapa sencilla:

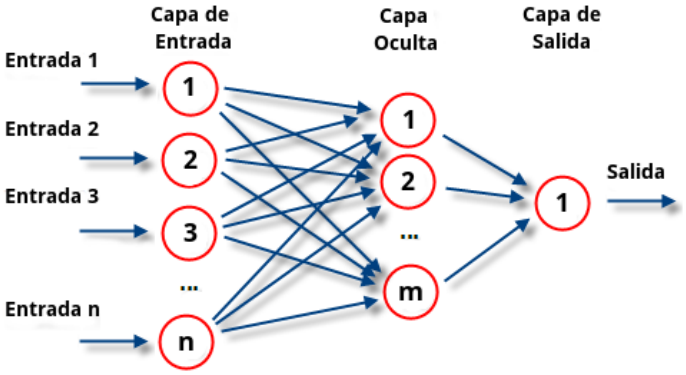


Figura 3.9: Red Neuronal Artificial Multicapa

Pues bien, la SVD tiene aplicaciones directas en redes neuronales al interpretar esta descomposición como una serie de transformaciones entre el espacio de entrada y el de salida [22]. Si se modela una matriz de pesos A como la relación entre entradas $\mathbf{x} \in \mathbb{R}^n$ y salidas $\mathbf{y} \in \mathbb{R}^m$, la transformación resultante puede expresarse como:

$$\mathbf{y} = U\Sigma V^\top \mathbf{x}.$$

En este modelo, la matriz $V^\top$ transforma las entradas al espacio ortogonal de características, Σ actúa como un cuello de botella que escala estas representaciones y U las proyecta al espacio de salida. Algunos usos de la SVD en este contexto son los siguientes:

- Como ya sabemos, una de las aplicaciones más importantes de la SVD es su capacidad para aproximar matrices de forma eficiente. La aproximación de rango k de una matriz A es (véase la sección 1.3.4):

  $$A_k = U_k \Sigma_k V_k^\top,$$

  donde U_k, Σ_k y V_k corresponden a las k primeras columnas de U y V, y a los k valores singulares principales de Σ. Este enfoque minimiza el error de reconstrucción según la

[4]La función Softmax es una generalización de la función sigmoide para múltiples clases. Dado un vector de entrada $\mathbf{z} = (z_1, z_2, \ldots, z_n)$, la función Softmax asigna a cada componente z_i un valor en el intervalo $[0, 1]$, de modo que la suma de todos los valores asignados sea 1. Está definida como:

$$Softmax\ (z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad i = 1, \ldots, n.$$

Esta función se utiliza frecuentemente en la capa de salida de las redes neuronales para problemas de clasificación multiclase, ya que transforma un conjunto de valores arbitrarios en una distribución de probabilidad.

norma de Frobenius:

$$\|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}.$$

En redes neuronales, esta propiedad se utiliza para comprimir matrices de pesos, reduciendo la cantidad de parámetros entrenables mientras se preserva la mayor parte de la información relevante [26].

- Además, la SVD es una herramienta poderosa para la regularización en problemas subdeterminados, es decir, cuando los datos disponibles no son suficientes para determinar de manera única los parámetros del modelo ([22, 26]). En estos casos, se puede construir una matriz regularizada eliminando los componentes asociados con valores singulares pequeños:

  $$A_{\text{mod}} = U_k \Sigma_k V_k^\top,$$

  donde solo se retienen los k valores singulares más significativos. Esto permite que el modelo sea más robusto frente a ruido en los datos.

- Otra aplicación importante de la SVD en redes neuronales es su uso como estrategia de inicialización de pesos. Dada una matriz de pesos A, la SVD permite descomponerla como:

  $$A = U \Sigma V^\top,$$

  y se pueden inicializar los pesos de la red como:

  $$W_{\text{entrada}} = V^\top, \quad W_{\text{salida}} = U\Sigma.$$

  Este enfoque garantiza que las conexiones iniciales preserven la información más relevante de los datos, mejorando la estabilidad y la velocidad de convergencia de los algoritmos de optimización.

- La pseudoinversa, cuyo cálculo puede realizarse a partir de la SVD, se utiliza en redes neuronales para resolver problemas de regresión ([22, 26]).

- Finalmente, la SVD también establece un puente entre redes neuronales lineales y no lineales [26].Para redes con una capa oculta lineal, la salida de la red se puede modelar como:

  $$\mathbf{y} = U_k \Sigma_k V_k^\top \mathbf{x},$$

  para cierto rango k, mientras que en redes no lineales, la SVD puede utilizarse para inicializar los pesos y mejorar la aproximación de mapeos entre el espacio de entrada y el de salida. Aunque la generalización de la SVD a redes profundas es más compleja, su aplicación iterativa capa por capa proporciona un enfoque práctico para optimizar redes de mayor complejidad.

Otro trabajo destacable y muy reciente en este ámbito es el presentado en [17]. El artículo propone una forma de entender cómo las redes neuronales convolucionales, o CNNs, procesan y clasifican imágenes. Utiliza la descomposición en valores singulares para analizar los pesos de las capas convolucionales, identificando qué características de las imágenes son más importantes

para la red. Además, introduce el uso de hipergrafos[5] para mostrar las relaciones entre clases y características, ayudando a interpretar cómo la red toma decisiones. Todo esto se acompaña de ejemplos prácticos y una herramienta llamada `DeepDataProfiler` para facilitar el análisis.

3.4.1. Análisis de Rendimiento de una Red Neuronal Convolucional (CNN) tras la Aplicación de la SVD en Python

El programa implementado en Python (véase el apéndice A.7) analiza el rendimiento de un modelo de clasificación utilizando la reducción de dimensionalidad mediante la descomposición en valores singulares (SVD). Se aplicó este método al conjunto de datos MNIST, el cual contiene imágenes de dígitos escritos a mano. A continuación, se describen las principales etapas del programa y los resultados obtenidos.

En cuanto a carga y preprocesamiento de datos, el programa utiliza el conjunto de datos MNIST, que incluye imágenes de 28×28 píxeles. Estas imágenes se normalizan dividiendo los valores de los píxeles por 255 y restando 0.5 para centrar los valores en torno a cero. Luego, las imágenes se aplanan en vectores de 784 dimensiones para facilitar su procesamiento.

```
# Cargar los datos
(training_images, training_labels), (test_images, test_labels) = mnist.
    load_data()

# Normalizar los datos
training_images = (training_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Aplanar las imagenes
training_images_flat = training_images.reshape(-1, 28*28)
test_images_flat = test_images.reshape(-1, 28*28)
```

En relación a la creación del modelo, se implementa una red neuronal sencilla con una capa densa de 128 neuronas, seguida de una capa de salida. La función de activación en la capa oculta es ReLU, mientras que la capa de salida utiliza `softmax` para clasificar las 10 categorías de dígitos.

```
def create_model(input_shape):
    model = Sequential([
        Dense(128, activation='relu', input_shape=input_shape),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Para reducir la dimensionalidad, se aplicó la descomposición en valores singulares (SVD) a las imágenes de entrada, variando el número de componentes principales (5, 25, 50, 100, 200).

[5]Es una generalización de un grafo donde las aristas, llamadas hiperaristas, pueden conectar más de dos nodos.

Posteriormente, las imágenes reconstruidas se utilizaron como entrada al modelo.

```
from sklearn.decomposition import TruncatedSVD

# Reduccion de dimensionalidad
svd = TruncatedSVD(n_components=num_components)
train_reduced = svd.fit_transform(training_images_flat)
train_reconstructed = svd.inverse_transform(train_reduced)
```

El modelo se evaluó en términos de precisión, comparando los datos originales y aquéllos procesados con SVD.

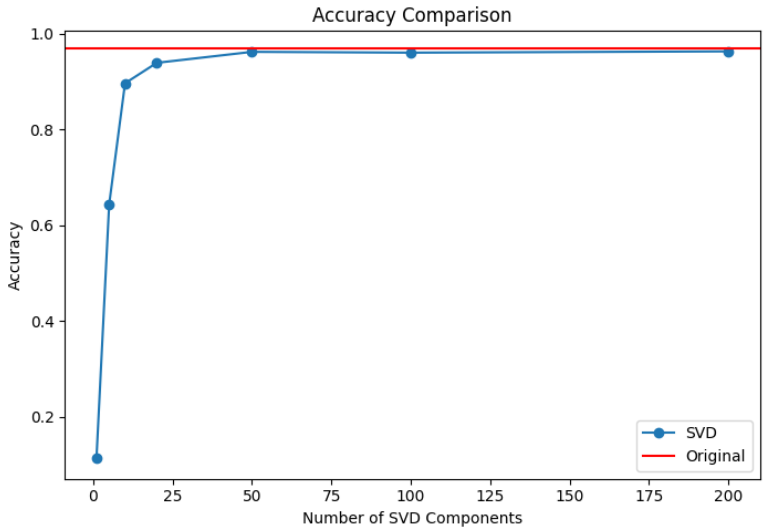


Figura 3.10: Comparación de precisión y tiempo computacional. La línea roja representa el rendimiento con los datos originales.

La figura 3.10 muestra que la precisión del modelo aumenta rápidamente conforme se incrementa el número de componentes en la descomposición SVD.

Con solo 50 componentes, el modelo ya alcanza una precisión prácticamente igual a la obtenida con los datos originales (línea roja). Esto indica que la mayor parte de la información relevante del conjunto de datos está contenida en las primeras 50 componentes principales, lo que demuestra la efectividad del método SVD para comprimir información sin pérdida significativa de calidad.

A partir de 50 componentes, el rendimiento del modelo se estabiliza, alcanzando valores cercanos al 100 % de precisión. Esto sugiere que usar más de 50 componentes no mejora significativamente el rendimiento, aunque podría aumentar el coste computacional.

En definitiva, los resultados muestran que la reducción de dimensionalidad es una técnica eficaz para mejorar la eficiencia de modelos de aprendizaje automático sin comprometer la precisión.

Bibliografía

[1] M. Alessandrini, G. Biagetti, P. Crippa, L. Falaschetti, L. Manoni, and C. Turchetti. Singular value decomposition in embedded systems based on arm cortex-m architecture. *Electronics*, 10(1):34, 2021.

[2] B. Bermeitinger, T. Hrycej, and S. Handschuh. *Singular Value Decomposition and Neural Networks*, page 153–164. Springer International Publishing, 2019.

[3] F. Buffo and A. Verdiell. La descomposición en valores singulares: Un enfoque geométrico y aplicaciones. *Revista de Educación Matemática.*, 19(1):33–45, 2004.

[4] J. Chung and A.K. Saibaba. Generalized hybrid iterative methods for large-scale bayesian inverse problems. *SIAM Journal on Scientific Computing*, 39(5):S24–S46, 2017.

[5] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput.*, 11(5):873–912, 1990.

[6] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, SIAM, Philadelphia, PA, USA, 1997.

[7] J. J. Fallas-Monge, J. Chavarría-Molina, and P. Soto-Quiros. Descomposición en valores singulares de una matriz: Un repaso por los fundamentos teóricos y sus aplicaciones en el procesamiento de imágenes. *Investigacion Operacional*, 42(2):148–173, 2021.

[8] W. Ford. *Numerical Linear Algebra with Applications using MATLAB*. Elsevier, London, 2015.

[9] G.H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[10] G.H. Golub and G.F. Van Loan. *Matrix Computation 4th ed.* John Hopkins University Press, Baltimore, MA, USA, 2013.

[11] G.H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14:403–420, 1970.

[12] P. C. Hansen, J. G. Nagy, and D. P. O'Leary. *Deblurring Images: Matrices, Spectra, and Filtering*. SIAM, Philadelphia, PA, 2006.

[13] C. Lanczos. *Linear Differential Operators*. Van Nostrand-Reinhold, London, UK, 1961.

[14] L. Martín Gil-Delgado. *Redes neuronales aplicadas a la generación de imágenes*. Grado en Ingeniería Informática, Universidad Politécnica de Madrid, 2023.

[15] C. C. Paige and M. A. Saunders. Towards a generalized singular value decomposition. *SIAM Journal on Numerical Analysis*, 18(3):398–405, 1981.

[16] B. Parlett. *The Symmetric Eigenvalue Problem*. Prentice Hall, Englewood Cliffs, New Jersey, 1980.

[17] B. Praggastis, D. Brown, C. Marrero, E. Purvine, M. Shapiro, and B. Wang. *The SVD of Convolutional Weights: A CNN Interpretability Framework*. arXiv preprint, 2022.

[18] Á. Rojas Matas and A. Cano Rojas. Trabajando con imágenes digitales en clase de matemáticas. *La Gaceta de la Real Sociedad Matemática Española*, 13(2):317–336, 2010.

[19] T. Roughgarden and G. Valiant. *CS168: The Modern Algorithmic Toolbox Lecture #9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations*. Stanford University, 2015.

[20] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35(4):551–566, 1993.

[21] G. Strang. The fundamental theorem of linear algebra. *The American Mathematical Monthly*, 100(9):848–855, 1993.

[22] J.E. Van Engelen and H.H. Hoos. A survey on semi-supervised learning. *Mach Learn*, 109:373–440, 2019.

[23] C. F. Van Loan. Generalizing the singular value decomposition. *SIAM Journal on Numerical Analysis*, 13(1):76–84, 1976.

[24] U. Zabaleta Gañán. *Redes Neuronales con función de Activación racional: función de crecimiento y Erzeugungsgrad*. Facultad de Ciencias, Grado en Matemáticas, Universidad de Cantabria, 2022.

[25] Z. Zang. The singular value decomposition, applications and beyond. *ArXiv*, abs/1510.08532, 2015.

[26] X. Zhu. *Semi-supervised learning literature survey*. Computer Sciences TR 1530, University of Wisconsin, Madison USA, 2019.

Apéndice A

Códigos empleados.

A.1. Interpretación Geométrica de la SVD

```
% Definimos la matriz A 3x3:
A = [a b c; d e f; g h i];

% Transformamos los puntos de la esfera por la matriz A:
points = A * [u(:)'; v(:)'; w(:)'];
transformed_u = reshape(points(1, :), size(u));
transformed_v = reshape(points(2, :), size(v));
transformed_w = reshape(points(3, :), size(w));
```

A.2. Comparativa de eficiencia entre SVD reducida y completa en matrices de dimensión dada

```
function compararTiemposSVD()
    % Solicitamos el numero de matrices y dimensiones
    n = validarEnteroPositivo('Ingrese el numero de matrices a tratar: ');
    m = validarEnteroPositivo('Ingrese el numero de filas de las matrices: ');
    p = validarEnteroPositivo('Ingrese el numero de columnas de las matrices: ');

    % Inicializamos los tiempos
    tiempos_completa = zeros(1, n);
    tiempos_reducida = zeros(1, n);

    % Calculamos la SVD para n matrices aleatorias
    for i = 1:n
        A = rand(m, p);
        tiempos_completa(i) = medirTiempo(@() svd(A));          % SVD completa
        tiempos_reducida(i) = medirTiempo(@() svd(A, 'econ')); % SVD reducida
    end
end

function t = medirTiempo(func)
    % Medir tiempo de ejecucion de una funcion
    tic; func(); t = toc;
```

```
end

function n = validarEnteroPositivo(mensaje)
    % Solicitar y validar un entero positivo
    n = input(mensaje);
    if n <= 0 || floor(n) ~= n
        error('El valor debe ser un entero positivo.');
    end
end
```

A.3. Aproximaciones de rango bajo de una matriz

```
% Definimos una matriz aleatoria y calculamos su SVD
A = rand(10);
[U, S, V] = svd(A);

% Configuracion inicial
k_values = [1, 2, 5, 10];
figure; colormap('gray');
sgtitle('Aproximaciones de Rango Bajo mediante la SVD');

% Graficamos la matriz original
subplot(1, numel(k_values) + 1, 1);
imagesc(A); axis square;
title('Matriz Original');

% Generamos y graficamos las aproximaciones de rango bajo
for i = 1:numel(k_values)
    k = k_values(i);
    Ak = U(:, 1:k) * S(1:k, 1:k) * V(:, 1:k)';
    subplot(1, numel(k_values) + 1, i + 1);
    imagesc(Ak); axis square;
    title(['Rango ', num2str(k)]);
end
```

A.4. Implementación del Método de Jacobi

```
function [U, S, V, elapsed_time, iter_count, error] = JOS_SVD(G)
%Establecemos los parametros necesarios
tol = 1e-10; iter_count = 0; error = 1;
n = size(G, 2);
V = eye(n); S = eye(n);

tic; % Inicio del temporizador
while error > tol
    error = 0; iter_count = iter_count + 1;
    for i = 1:n-1
        for j = i+1:n
            x = G(:, i); y = G(:, j);
            a = x' * x; b = y' * y; c = x' * y;
            error = max(error, abs(c) / sqrt(a * b)); % Actualizar error

            % Calculamos la rotacion de Jacobi
```

```
            e = (b - a) / (2 * c);
            t = sign(e) / (abs(e) + sqrt(1 + e^2));
            cs = 1 / sqrt(1 + t^2); sn = cs * t;

            % Actualizamos  G y V
            G(:, [i, j]) = [cs * x - sn * y, sn * x + cs * y];
            V(:, [i, j]) = [cs * V(:, i) - sn * V(:, j), sn * V(:, i) + cs * V(:, j)];
        end
    end
end
elapsed_time = toc;

% Construimos U y S
for i = 1:n
    S(i, i) = norm(G(:, i));
    G(:, i) = G(:, i) / S(i, i);
end
U = G;

% Ajustamos la salida si solo se solicita S
if nargout <= 1, U = S; end
end
```

A.5. Implementación del Algoritmo de Demmel-Kahan Zero Shift

```
function [B, U, V, numIterations, elapsedTime, error] = DKzeroshift(A)
% DKzeroshift: Diagonalizacion de una matriz bidiagonal usando rotaciones de Givens.

[numIterations, tol, maxIter] = deal(0, 1e-6, 50000);
tic; % Inicio del temporizador

% Bidiagonalizamos la matriz y calculamos los valores singulares exactos
[B, U, V] = bidiagonal(A);
[~, S_mat, ~] = svd(A); S_mat = diag(S_mat);
[m, n] = size(B);

if m < n, error('El numero de filas debe ser >= al de columnas.'); end
B = B(1:n, :); % Eliminar filas nulas de B
fin = n;

while (norm(S_mat - diag(B)) / norm(S_mat) > tol && numIterations < maxIter)
    numIterations = numIterations + 1;

    % Primera rotacion de Givens (derecha)
    [c, s] = givens(B(1, 1), B(1, 2));
    B(1:2, 1:2) = apply_givens(B(1:2, 1:2), c, s, 'right');
    V(:, 1:2) = apply_givens(V(:, 1:2), c, s, 'left');

    % Rotaciones de Givens en el resto de la matriz
    for i = 1:fin-2
        [c, s] = givens(B(i, i), B(i+1, i));
```

```
27          B(i:i+1, i:i+2) = apply_givens(B(i:i+1, i:i+2), c, s, 'left');
28          U(i:i+1, :) = apply_givens(U(i:i+1, :), c, s, 'left');
29
30          [c, s] = givens(B(i, i+1), B(i, i+2));
31          B(i:i+2, i+1:i+2) = apply_givens(B(i:i+2, i+1:i+2), c, s, 'right');
32          V(:, i+1:i+2) = apply_givens(V(:, i+1:i+2), c, s, 'left');
33      end
34
35      % Ultima rotacion de Givens
36      [c, s] = givens(B(fin-1, fin-1), B(fin, fin-1));
37      B(fin-1:fin, fin-1:fin) = apply_givens(B(fin-1:fin, fin-1:fin), c, s, '
    left');
38      U(fin-1:fin, :) = apply_givens(U(fin-1:fin, :), c, s, 'left');
39
40      % Reducimos el tamano de la matriz si B(fin-1, fin) < tolerancia
41      if fin > 2 && abs(B(fin-1, fin)) < tol, fin = fin - 1; end
42
43      % Aseguramos que hay elementos positivos en la diagonal
44      for i = 1:n
45          if B(i, i) < 0
46              B(i, i) = -B(i, i);
47              U(:, i) = -U(:, i);
48          end
49      end
50  end
51
52  % Ordenamos los elementos diagonales en orden descendente
53  [x, I] = sort(diag(B), 'descend');
54  U(1:n, :) = U(I, :); V = V(:, I);
55  B = [diag(x); zeros(m-n, n)];
56
57  elapsedTime = toc;
58  error = norm(S_mat - diag(B)) / norm(S_mat);
59  fprintf('Error: %e\n', error);
60  end
61
62  function [c, s] = givens(a, b)
63  % Genera rotaciones de Givens que anulan el elemento b.
64  if b == 0
65      c = 1; s = 0;
66  else
67      t = b / a;
68      c = 1 / sqrt(1 + t^2); s = c * t;
69  end
70  end
71
72  function [A, U, V] = bidiagonal(A)
73  % Bidiagonalizamos la matriz A usando reflectores de Householder.
74  [m, n] = size(A);
75  if m < n, error('El numero de filas debe ser >= al de columnas.'); end
76
77  U = eye(m); V = eye(n);
78  for k = 1:n-1
79      w = house(A(k:m, k));
80      beta = 2 / (w' * w);
81      A(k:m, k:n) = A(k:m, k:n) - beta * w * (w' * A(k:m, k:n));
82      U(k:m, :) = U(k:m, :) - beta * w * (w' * U(k:m, :));
```

```
    if k < n
        z = house(A(k, k+1:n)');
        gamma = 2 / (z' * z);
        A(k:m, k+1:n) = A(k:m, k+1:n) - gamma * (A(k:m, k+1:n) * z) * z';
        V(:, k+1:n) = V(:, k+1:n) - gamma * (V(:, k+1:n) * z) * z';
    end
end
end

function w = house(x)
% Generamos un reflector de Householder para x.
alpha = -sign(x(1)) * norm(x);
w = x; w(1) = w(1) - alpha; w = w / norm(w);
end

function M = apply_givens(M, c, s, direction)
% Aplicamos una rotacion de Givens a la matriz M.
switch direction
    case 'left', M = [c -s; s c] * M;
    case 'right', M = M * [c -s; s c]';
end
end
```

A.6. Implementación y Evaluación del Algoritmo GoDec para Diferentes Matrices

```
function godec_multiple_matrices()
    % Parametros del algoritmo
    max_iter = 40; k = 3; c0_factor = 0.1; epsilon = 1e-6;
    matrices = {rand(10, 10), 'Matriz Aleatoria Uniforme (rand)';
                randn(10, 10), 'Matriz Aleatoria Normal (randn)';
                corr(rand(10,10)), 'Matriz de Correlacion';
                orth(rand(10,10)), 'Matriz Ortogonal';
                magic(10), 'Matriz Magica'};

    % Inicializacion de la grafica
    figure; hold on;
    colors = lines(size(matrices, 1)); legend_entries = cell(size(matrices,
    1), 1);

    % Procesamos cada matriz
    for i = 1:size(matrices, 1)
        [errors, ~, ~] = godec(matrices{i, 1}, k, c0_factor, max_iter,
    epsilon);
        plot(1:length(errors), errors, 'Color', colors(i, :), 'LineWidth',
    1.5);
        legend_entries{i} = matrices{i, 2};
    end

    % Configuracion de la grafica
    xlabel('Iteraciones'); ylabel('Error relativo');
    title('Error relativo del algoritmo GoDec para diferentes matrices');
    legend(legend_entries, 'Location', 'northeast'); grid on; hold off;
end
```

```
function [errors, L, S] = godec(A, k, c0_factor, max_iter, epsilon)
    [m, n] = size(A); c0 = round(c0_factor * m * n);
    L = A; S = zeros(m, n); errors = zeros(max_iter, 1);

    for t = 1:max_iter
        [U, Sigma, V] = svd(A - S, 'econ'); % Actualiza L
        L = U(:, 1:k) * Sigma(1:k, 1:k) * V(:, 1:k)';
        residual = A - L; % Actualiza S
        S = sparse_projection(residual, c0);
        errors(t) = norm(A - L - S, 'fro')^2 / norm(A, 'fro')^2;

        % Verificamos la convergencia
        if t > 1 && abs(errors(t) - errors(t-1)) < epsilon
            errors = errors(1:t); break;
        end
    end
end

function S = sparse_projection(X, c0)
    [~, idx] = maxk(abs(X(:)), c0); % Selecciona las c0 entradas mas grandes
    S = zeros(size(X)); S(idx) = X(idx); % Asigna valores correspondientes
end
```

A.7. La SVD en el Entrenamiento de una Red Neuronal Convolucional (CNN) en Python

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
import time
import matplotlib.pyplot as plt

# Cargamos y normalizamos los datos
(training_images, training_labels), (test_images, test_labels) = mnist.
    load_data()
training_images, test_images = (training_images / 255) - 0.5, (test_images /
     255) - 0.5

# Aplanamos las imagenes
training_images_flat, test_images_flat = training_images.reshape(-1, 28*28),
     test_images.reshape(-1, 28*28)

# Creamos el modelo
def create_model(input_shape):
    model = Sequential([Dense(128, activation='relu', input_shape=
    input_shape), Dense(10, activation='softmax')])
    model.compile('adam', loss='categorical_crossentropy', metrics=['
    accuracy'])
    return model

# Entrenamiento con imagenes originales
```

```
model = create_model((28*28,))
start_time = time.time()
history_original = model.fit(training_images_flat, to_categorical(
    training_labels), batch_size=32, epochs=4, verbose=2, validation_data=(
    test_images_flat, to_categorical(test_labels)))
time_original, accuracy_original = time.time() - start_time,
    history_original.history['val_accuracy'][-1]

# Aplicamos la SVD y entrenamiento
def apply_svd(images, n_components):
    U, S, Vt = np.linalg.svd(images, full_matrices=False)
    S_truncated = np.zeros_like(S); S_truncated[:n_components] = S[:
    n_components]
    return np.dot(U, np.dot(np.diag(S_truncated), Vt))

# SVD con distintos componentes
components = [1, 5, 10, 20, 50, 100, 200]
accuracies_svd, times_svd = [], []

for n in components:
    training_images_svd, test_images_svd = apply_svd(training_images_flat, n
    ), apply_svd(test_images_flat, n)
    model = create_model((28*28,))
    start_time = time.time()
    history_svd = model.fit(training_images_svd, to_categorical(
    training_labels), batch_size=32, epochs=4, verbose=2, validation_data=(
    test_images_svd, to_categorical(test_labels)))
    accuracies_svd.append(history_svd.history['val_accuracy'][-1])
    times_svd.append(time.time() - start_time)
    print(f"Entrenado con {n} componentes SVD.")
```