Cálculo Numérico Ingenuo de la SVD

Objetivos

General

Comprender, implementar y analizar el algoritmo numérico ingenuo para calcular la descomposición en valores singulares (SVD).

Específicos

- Exponer las conexiones teóricas entre la SVD y los problemas de autovalores simétricos,
- implementar el algoritmo utilizando los algoritmos QR de las notas de clase,
- analizar el comportamiento numérico del algoritmo clásico frente a métodos mal condicionados,
- discutir posibles estrategias prácticas.

En otro trabajo se puede aplicar la SVD computada numéricamente a un problema práctico.

Introducción

La descomposición en valores singulares (SVD) es una de las herramientas más poderosas y versátiles del álgebra lineal numérica. No solo existe para cualquier matriz y proporciona información estructural sobre ella —como su rango, condición y modos principales de acción—, sino que también constituye la base de aplicaciones fundamentales en análisis de datos, compresión, métodos de mínimos cuadrados, y problemas inversos.

Teoría

Definición y existencia de la SVD

Sea $A \in \mathbb{R}^{m \times n}$. La **descomposición en valores singulares (SVD)** de A es una factorización de la forma $A = U\Sigma V^T$, donde

- las matrices $U \in \mathbb{R}^{m \times m}$ y $V \in \mathbb{R}^{n \times n}$ son ortogonales,
- la matriz $\Sigma \in \mathbb{R}^{m \times n}$ es diagonal rectangular con entradas no negativas en la diagonal:

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & \\ & & & 0 \end{bmatrix}, \quad \text{con } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0,$$

y $r = \mathbf{rank}(A)$.

Las cantidades σ_i se llaman **valores singulares** de A, y son únicos. Los vectores columna de U y V se llaman **vectores singulares izquierdos** y **derechos**, respectivamente.

Importancia de la SVD

La importancia de la descomposición SVD está basada en que **caracteriza cualquier** matriz real $A \in \mathbb{R}^{m \times n}$, sin importar que sea cuadrada, rectangular, singular o no.

Intuición geométrica

La SVD describe la acción de la matriz A como una transformación que:

1. **Rota** el espacio fuente (a través de V^T),
2. **Escala** en direcciones ortogonales (a través de Σ),
3. **Rota** el espacio imagen (a través de U).

En otras palabras, toda matriz real lineal puede descomponerse como una secuencia de rotaciones/reflexiones y escalamientos. Esta intuición es presentada de manera gráfica por Visual Kernel en el Capítulo 1 Visualize Different Matrices de su colección SEE Matrix en Youtube.

Implementación

En este cuaderno compararemos dos métodos diferentes de descomposición SVD: el `svd` ya implementado de la librería `Linear Algebra` de Julia y una implementación propia del método ingenuo.

Set up

```
begin
    using LinearAlgebra
    using BenchmarkTools
    using Plots
    using DataFrames
end
```

```
struct SVDReconstruction
    U::Matrix{Float64}
    S::Vector{Float64}
    V::Matrix{Float64}
end

```

SVD de Julia

```
svd (generic function with 14 methods)
```

```
svd
```

Método ingenuo

De acuerdo con el libro el pseudocódigo es el siguiente:

1. Form $C = A^T A$
2. Use the symmetric QR algorithm to compute $V^T C V = diag(\sigma^2)$
3. Apply QR with column pivoting to AV to obtain $U^T(AV)N = R$

```
naiveSVD_classic_ (generic function with 1 method)
```

```
function naiveSVD_classic_(A::Matrix{Float64})
    #Paso 1
    C = transpose(A) * A

    #Paso 2
    T, V = RealSchur(C)
    λ = diag(T)
    σ = sqrt.(abs.(λ))

    #Reorden
    orden = sortperm(σ, rev=true)
    σ = σ[orden]
    V = V[:, orden]

    #Paso 3
    AV = A * V
    U, _, _ = QRPivotado_(AV)
    return SVDReconstruction(U, σ, V)
end
```

Paso 2: Obtención de autovalores y autovectores

Para obtener los autovalores y los autovectores (V) de C utilizamos la descomposición de Schur. Utilizaremos una adaptación del método `RealSchur` del cuaderno `SchurFinal.jl`.

La adaptación es para guardar también `Q` en `HessenbergQR` y `RealSchur`.

Ahora, comparemos el método propio de Julia `eigen` con la función `RealSchur` para autovalores y

autovectores.

Autovalores:

```
[4.02771, 0.428116, 0.15329, 0.147497, 0.00471828]
```

```
sort(eigen(C).values, rev=true)
```

```
5×5 Diagonal{Float64, Vector{Float64}}:
 4.02771    ⋅         ⋅         ⋅         ⋅
  ⋅       0.428116    ⋅         ⋅         ⋅
  ⋅         ⋅        0.15329    ⋅         ⋅
  ⋅         ⋅         ⋅       0.147497    ⋅
  ⋅         ⋅         ⋅         ⋅        0.00471828
```

```
Diagonal(RealSchur(C)[1])
```

Autovectores:

```
5×5 Matrix{Float64}:
 -0.0492655   0.437031    0.112491  -0.736121   -0.502045
  0.551959   -0.0201312   0.311537   0.434796   -0.639401
 -0.596954   -0.653446    0.202529  -0.0320571  -0.417863
  0.206946   -0.261612   -0.885792  -0.0940297  -0.308647
 -0.541973    0.559613   -0.254252   0.509126   -0.263145
```

```
eigen(Symmetric(C)).vectors
```

```
5×5 Matrix{Float64}:
 0.502045  -0.736121    0.112491   0.437031    0.0492655
 0.639401   0.434796    0.311537  -0.0201312  -0.551959
 0.417863  -0.0320571   0.202529  -0.653446    0.596954
 0.308647  -0.0940297  -0.885792  -0.261612   -0.206946
 0.263145   0.509126   -0.254252   0.559613    0.541973
```

```
RealSchur(C)[2]
```

Podemos ver que ambos métodos coinciden. Ahora veamos el código de `Real_Schur`, junto con las funciones auxiliares `HessenbergForm` y `HessenbergQR` que transforman la matriz en una matriz Hessenberg y luego aplican la descomposición QR.

```
RealSchur (generic function with 2 methods)
```

```
function RealSchur(A, iteraciones = 10000)
    H0 = A
    H1, Q = HessenbergForm(A)
    δ = 10
    for _ = 1:iteraciones
        H0 = H1
        H1,Q = HessenbergQR(H1,Q)
    end
    return H1, Q

end
```

```
Housev (generic function with 1 method)
```

```
begin
    function HessenbergForm(A)
        n = size(A)[1]
        H = copy(A)
        U = Matrix(1.0*I, n, n)
        for k = 1:n-2
            v, β = Housev(H[k+1:n, k])
            H[k+1:n, k:n] = (I - β*v*v')*H[k+1:n, k:n]
            H[1:n, k+1:n] = H[1:n, k+1:n]*(I - β*v*v')

            #U es necesaria para la verificar que la función devuelve los resultados correctos
            U[1:n, k+1:n] = U[1:n, k+1:n]*(I - β*v*v')
        end
        return H, U
    end

    function Housev(x)
        n = length(x)
        v = ones(size(x))
        v[2:n] = x[2:n]
        σ = norm(x[2:n])^2
        if σ == 0
            β = 0
        else
            μ = √(x[1]^2+σ)
            if x[1] ≤ 0
                v[1] = x[1] - μ
            else
                v[1] = -σ/(x[1]+μ)
            end
            β = 2*v[1]^2/(σ+v[1]^2)
            v = v/(v[1])
        end
        return v, β
    end
end
```

```
Givens (generic function with 1 method)
```

```
begin
    function HessenbergQR(H,Q)
        n = size(H)[1]
        H2 = copy(H)
        C, S = zeros(n-1), zeros(n-1)

        # Factorización QR de H
        # Q^T*H o Givens por la izquierda
        for k = 1:n-1
            C[k], S[k] = Givens(H2[k,k], H2[k+1, k])
            H2[k:k+1,k:n] = [C[k] -S[k]; S[k] C[k]]*H2[k:k+1,k:n]
            H2[k+1,k] = 0
            Q[:, k:k+1] = Q[:, k:k+1] * [C[k] S[k]; -S[k] C[k]]  # actualizar Q
        end

        # Matriz RQ
        # H*Q o Givens por la derecha
        for k = 1:n-1
            H2[1:k+1, k:k+1] = H2[1:k+1, k:k+1]*[C[k] S[k]; -S[k] C[k]]
        end

        return H2,Q
    end

    begin
        function Givens(a,b)
            if b==0
                c = 1
                s = 0
            else
                if abs(b)>abs(a)
                    τ=-a/b
                    s=-1/sqrt(1+τ^2)
                    c=s*τ
                else
                    τ=-b/a
                    c=1/sqrt(1+τ^2)
                    s=c*τ
                end
            end
            return c,s
        end
    end
end
```

Paso 3: Obtener U

Aplicamos el algoritmo QR con pivoteo de columna a AV para obtener $U^T(AV)N = R$.

```
QRPivotado_ (generic function with 1 method)
```

```julia
function QRPivotado_(A::Matrix{Float64})
    m, n = size(A)
    Q = zeros(m, n)
    R = zeros(n, n)
    P = collect(1:n)
    A_work = copy(A)
    norms = [norm(A[:, j]) for j in 1:n]

    for i in 1:n
        max_col = argmax(norms[i:end]) + i - 1
        A_work[:, [i, max_col]] = A_work[:, [max_col, i]]
        P[[i, max_col]] = P[[max_col, i]]
        norms[[i, max_col]] = norms[[max_col, i]]

        v = A_work[:, i]
        for j = 1:i-1
            R[j, i] = dot(Q[:, j], v)
            v -= R[j, i] * Q[:, j]
        end
        R[i, i] = norm(v)
        Q[:, i] = v / R[i, i]
    end

    return Q, R, P
end
```

Evaluación

En esta sección definiremos algunas funciones para poder evaluar los métodos y mostraremos algunos ejemplos.

Funciones de evaluación

reconstruct_from_svd

Reconstruye la matriz original A a partir de su descomposición SVD. Recibe un objeto F retornado por la función svd(A). Devuelve A ≈ U * Σ * Vᵀ

```
"""
Reconstruye la matriz original A a partir de su descomposición SVD.
Recibe un objeto F retornado por la función svd(A).
Devuelve A ≈ U * Σ * Vᵀ
"""
function reconstruct_from_svd(F; V_transposed=false)
    n = size(F.S, 1)
    Σ = zeros(n, n)
    for i in 1:length(F.S)
        Σ[i, i] = F.S[i]
    end
    if V_transposed
        return F.U * Σ * F.V
    else
        return F.U * Σ * F.V'
    end
end
```

validate_svd

Valida una descomposición SVD comparando la matriz original A con su reconstrucción desde la tupla o estructura SVD `F`.

Imprime la norma del error ‖A - UΣVᵀ‖₂.

```
"""
Valida una descomposición SVD comparando la matriz original A
con su reconstrucción desde la tupla o estructura SVD `F`.

Imprime la norma del error ‖A - UΣVᵀ‖₂.
"""
function validate_svd(A::Matrix{Float64}, F; V_transposed=false)
    A_hat = reconstruct_from_svd(F;V_transposed=V_transposed)
    error = norm(A - A_hat)
    display(A_hat)
    println("Error de reconstrucción ‖A - UΣVᵀ‖₂ = ", error)

    return error
end
```

Ejemplos

Veamos el error de reconstrucción para cada método. Podemos ver que para `svd` y para `naiveSVD_classic_` el error corresponde al error de máquina, lo cual es un buen signo.

```
begin
    A=rand(5,5)
    C=A'*A
    display(A)
end
```

```
5×5 Matrix{Float64}:
 0.791339   0.246093  0.278481  0.4405     0.0381061
 0.631365   0.874829  0.633694  0.234757   0.170661
 0.227865   0.221405  0.307687  0.0633271  0.00169795
 0.447363   0.787861  0.265672  0.226172   0.580245
 0.0197816  0.498335  0.360961  0.462313   0.283009
```

1.5785651367583568e-15

```
begin
    s1=svd(A)
    validate_svd(A, s1)
end
```

```
5×5 Matrix{Float64}:
 0.791339   0.246093  0.278481  0.4405     0.0381061
 0.631365   0.874829  0.633694  0.234757   0.170661
 0.227865   0.221405  0.307687  0.0633271  0.00169795
 0.447363   0.787861  0.265672  0.226172   0.580245
 0.0197816  0.498335  0.360961  0.462313   0.283009
Error de reconstrucción ‖A - UΣV^T‖₂ = 1.5785651367583568e-15
```

1.8277095360737476e-14

```
begin
    s3=naiveSVD_classic_(A)
    validate_svd(A, s3)
end
```

```
5×5 Matrix{Float64}:
 0.791339   0.246093  0.278481  0.4405     0.0381061
 0.631365   0.874829  0.633694  0.234757   0.170661
 0.227865   0.221405  0.307687  0.0633271  0.00169795
 0.447363   0.787861  0.265672  0.226172   0.580245
 0.0197816  0.498335  0.360961  0.462313   0.283009
Error de reconstrucción ‖A - UΣV^T‖₂ = 1.8277095360737476e-14
```

Análisis y resultados

Comparemos la diferencia de tiempo y de exactitud entre los diferentes métodos por medio de gráficas y tablas.

ns = 50:50:500

```
ns = 50:50:500
```

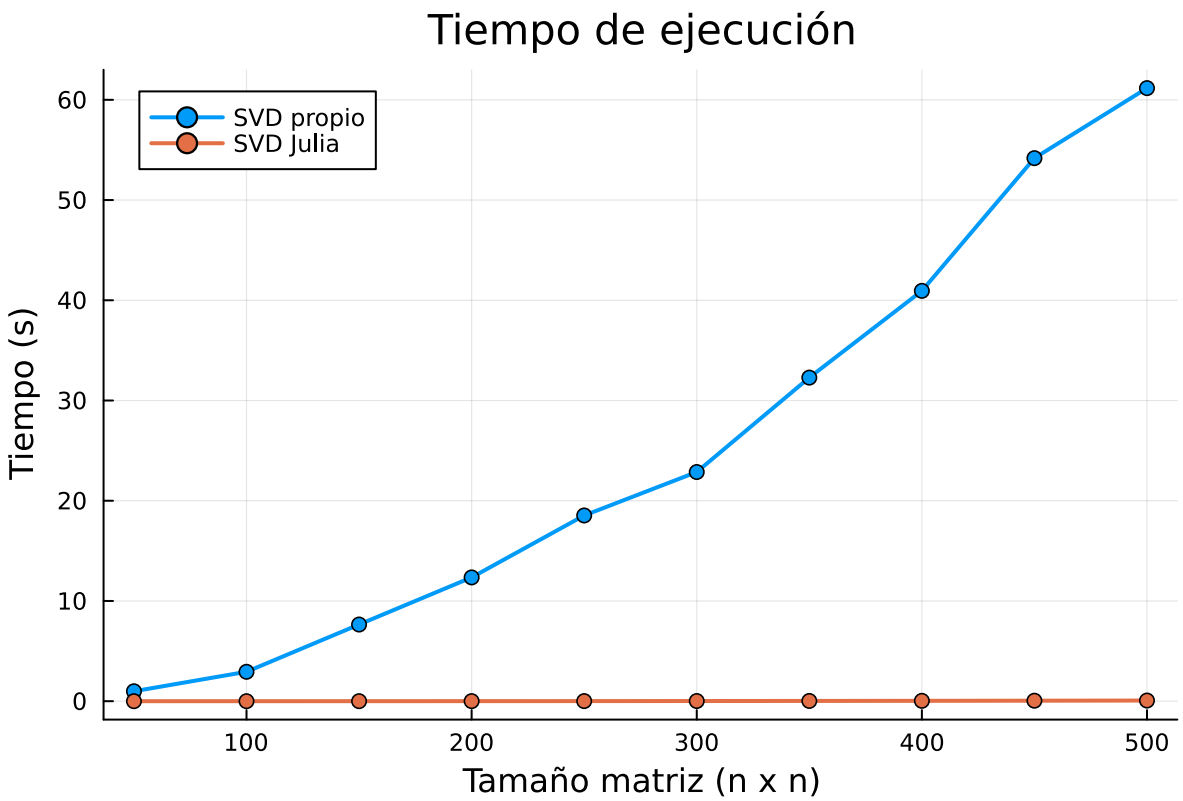
```
begin
    tiempos_naive = []
    tiempos_julia = []
    errores_naive = []
    errores_julia = []
    for n in ns
        local A = randn(n, n)

        # --- SVD Julia ---
        t_julia = @belapsed svd($A)
        err_julia = norm(A - reconstruct_from_svd(svd(A)))

        # --- SVD propia ---
        t_naive = @belapsed naiveSVD_classic_($A)
        err_naive = norm(A - reconstruct_from_svd(naiveSVD_classic_(A)))

        # Guardar
        push!(tiempos_naive, t_naive)
        push!(tiempos_julia, t_julia)
        push!(errores_naive, err_naive)
        push!(errores_julia, err_julia)
    end
end
```

Comparación de tiempo




```
plot(ns, [tiempos_naive, tiempos_julia],
    label=["SVD propio" "SVD Julia"],
    title="Tiempo de ejecución",
    xlabel="Tamaño matriz (n x n)",
    ylabel="Tiempo (s)",
    lw=2, marker=:circle)
```

```
display(
    DataFrame(
        n = collect(ns),
        Ingenuo = tiempos_naive,
        Julia = tiempos_julia
    )
)
```

10×3 DataFrame

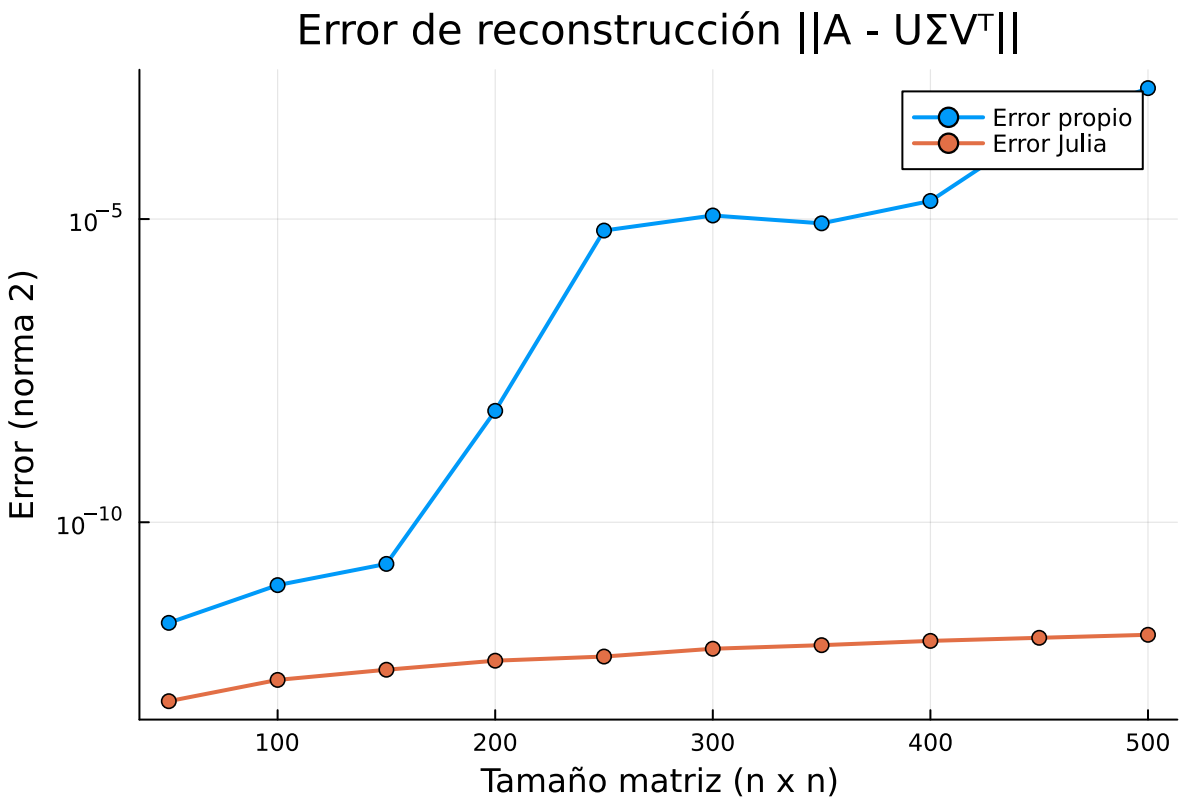
Row	n	Ingenuo	Julia
	Int64	Any	Any
1	50	0.990598	0.0002952
2	100	2.9338	0.0011648
3	150	7.65007	0.002833
4	200	12.3487	0.0060319
5	250	18.5297	0.0107555
6	300	22.8646	0.0169011
7	350	32.2853	0.0254003
8	400	40.9392	0.0367638
9	450	54.1789	0.050535
10	500	61.1659	0.0657683

Comparación de exactitud

```
display(
    DataFrame(
        n = collect(ns),
        Ingenuo = tiempos_naive,
        Julia = tiempos_julia
    )
)
```

10×3 DataFrame

Row	n	Ingenuo	Julia
	Int64	Any	Any
1	50	0.990598	0.0002952
2	100	2.9338	0.0011648
3	150	7.65007	0.002833
4	200	12.3487	0.0060319
5	250	18.5297	0.0107555
6	300	22.8646	0.0169011
7	350	32.2853	0.0254003
8	400	40.9392	0.0367638
9	450	54.1789	0.050535
10	500	61.1659	0.0657683




```
plot(ns, [errores_naive, errores_julia],
    label=["Error propio" "Error Julia"],
    title="Error de reconstrucción ||A - UΣVᵀ||",
    xlabel="Tamaño matriz (n x n)",
    ylabel="Error (norma 2)",
    lw=2, marker=:circle, yscale=:log10)
```

Resultados

Podemos observar que al comparar el `svd` de Julia con el método `naiveSVD_classic_` implementado en este cuaderno:

- El error de reconstrucción es similar en ambos casos, aunque ligeramente menor con `svd`.
- Sin embargo, el tiempo de ejecución es considerablemente mayor en `naiveSVD_classic_`. Esto se debe a que su implementación sigue un enfoque ingenuo y no optimizado, sin utilizar técnicas avanzadas como el paso de Golub-Kahan que emplea la función `svd` de Julia.

Contribución y reflexión

El desarrollo del código y análisis presentados en este cuaderno fue realizado por el estudiante, con las siguientes contribuciones específicas:

- **Adaptación del código base:** Se partió del código proporcionado por el profesor en Classroom, el cual fue modificado para que las funciones devolvieran no solo la matriz transformada, sino también la información ortogonal necesaria para su uso posterior en la implementación ingenua

de la SVD.
- **Diseño experimental:** Se diseñaron y ejecutaron pruebas comparativas de desempeño entre el svd de Julia y la implementación ingenua, para diferentes tamaños de matrices. Esto incluyó la medición de tiempos de ejecución y errores de reconstrucción, así como la organización visual de los resultados.
- **Análisis de resultados:** Se interpretaron los resultados numéricos y se redactaron conclusiones sobre la precisión y eficiencia de cada método, destacando fortalezas y limitaciones del enfoque ingenuo frente al método optimizado de Julia.

El principal aprendizaje que me dejó este notebook fue la importancia de la comunicación, dado que inicialmente no tenía claras las instrucciones y estaba haciendo un trabajo innecesario.

Declaración sobre el uso de inteligencia artificial y fuentes externas

Durante la realización de este trabajo se utilizó **asistencia de inteligencia artificial (IA)** para generar y mejorar partes del código y de fuentes externas, con el fin de estructurar mejor el contenido y optimizar la implementación.

Uso de IA:

En la elaboración de este cuaderno se utilizó inteligencia artificial (IA), específicamente ChatGPT (OpenAI), como herramienta de apoyo para:

- Generar código en Julia para implementar una versión ingenua de la descomposición en valores singulares (SVD), incluyendo el uso de algoritmos QR propios.
- Redactar explicaciones matemáticas, comentarios de código y secciones interpretativas del comportamiento numérico de los algoritmos.
- Reformular fragmentos de texto para mejorar la claridad y redacción técnica.

Los principales prompts utilizados fueron del estilo:

- "Cómo puedo mostrar los resultados de tiempos*naive y tiempos*julia en dos columnas contiguas"
- "¿Puedes explicar esta función paso a paso?"
- "Reescribe este párrafo de forma clara y académica."

Se supervisó cuidadosamente cada respuesta de la IA para verificar su validez, coherencia matemática y relevancia dentro del contexto del trabajo.

Fuentes externas consultadas:

- Código guía de la clase, cuaderno `SchurFinal.jl`
- Marangoz, S. (2023). *Image Imputation with SVD*. https://salihmarangoz.github.io/blog/Image-

Imputation-with-SVD

- Golub, G., & Van Loan, C. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.