Segundo Taller Computacional
Álgebra Lineal Aplicada

Gustavo Adolfo Pérez Pérez
Universidad Nacional de Colombia – Sede Medellín
Programa de Ciencias de la Computación

Fecha de entrega: 2 de julio de 2025

Índice

1. Convergencia y estabilidad de sucesiones recurrentes

En esta sección analizamos el comportamiento asintótico del cociente x_{n+1}/x_n para sucesiones definidas por recurrencias lineales homogéneas y su estabilidad ante perturbaciones en las condiciones iniciales.

1.1. Marco teórico

Consideremos una recurrencia lineal homogénea de orden k:

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k}, \quad n \geq k$$

con condiciones iniciales $x_0, x_1, \ldots, x_{k-1}$ dadas.

1.1.1. Polinomio característico

El comportamiento de la sucesión está determinado por las raíces del polinomio característico:

$$p(\lambda) = \lambda^k - a_1\lambda^{k-1} - a_2\lambda^{k-2} - \cdots - a_k$$

Si $\lambda_1, \lambda_2, \ldots, \lambda_k$ son las raíces (posiblemente complejas y con multiplicidad), la solución general es:

$$x_n = \sum_{i=1}^{k} c_i \lambda_i^n$$

donde los coeficientes c_i se determinan por las condiciones iniciales resolviendo el sistema lineal:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_k \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^{k-1} & \lambda_2^{k-1} & \cdots & \lambda_k^{k-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{k-1} \end{pmatrix}$$

1.1.2. Condiciones de convergencia

Teorema (Convergencia del cociente). El límite

$$L = \lim_{n\to\infty} \frac{x_{n+1}}{x_n}$$

existe si y solo si:

1. Existe una única raíz dominante λ^* tal que $|\lambda^*| > |\lambda_i|$ para todo $i \neq *$
2. El coeficiente c^* correspondiente a λ^* en la solución general es no nulo

En tal caso, $L = \lambda^*$.

Demostración. Si se cumplen las condiciones, para n grande:

$$x_n = c^*(\lambda^*)^n + \sum_{i \neq *} c_i \lambda_i^n = c^*(\lambda^*)^n \left(1 + \sum_{i \neq *} \frac{c_i}{c^*} \left(\frac{\lambda_i}{\lambda^*}\right)^n\right)$$

Como $|\lambda_i/\lambda^*| < 1$ para $i \neq *$, los términos adicionales tienden a cero:

$$\frac{x_{n+1}}{x_n} = \lambda^* \cdot \frac{1 + O(r^n)}{1 + O(r^n)} \to \lambda^*$$

donde $r = \max_{i \neq *} |\lambda_i/\lambda^*| < 1$. □

1.2. Implementación del algoritmo

El algoritmo implementado sigue estos pasos:

1. **Cálculo de raíces**: Encuentra las raíces del polinomio característico usando métodos numéricos robustos.
2. **Identificación de raíces dominantes**: Determina las raíces con módulo máximo dentro de una tolerancia numérica.
3. **Sistema de Vandermonde**: Resuelve el sistema lineal para obtener los coeficientes c_i.
4. **Verificación de condiciones**: Comprueba que existe una única raíz dominante con coeficiente no nulo.
5. **Método de respaldo**: Si el coeficiente dominante es numéricamente cero, realiza un análisis iterativo para detectar convergencia empírica.

El código fuente completo está disponible en: `https://github.com/gustavop-dev/segundo_taller_alg_lin/blob/master/primer_punto/convergencia_sucesiones.py`

1.3. Ejemplos de análisis

1.3.1. Ejemplo 1: Sucesión de Fibonacci

La sucesión de Fibonacci se define por:

$$x_n = x_{n-1} + x_{n-2}, \quad x_0 = 1,\ x_1 = 1$$

El polinomio característico es $\lambda^2 - \lambda - 1 = 0$ con raíces:

$$\lambda_1 = \frac{1 + \sqrt{5}}{2} \approx 1{,}618034, \quad \lambda_2 = \frac{1 - \sqrt{5}}{2} \approx -0{,}618034$$

Como $|\lambda_1| > |\lambda_2|$, existe una raíz dominante única. El análisis computacional confirma:

Propiedad	Valor
¿Converge?	Sí
Límite teórico	$\phi = \frac{1+\sqrt{5}}{2}$
Límite numérico	1.618033988749895
Error absoluto	$< 10^{-15}$

1.3.2. Ejemplo 2: Sucesión de Lucas

Consideremos la recurrencia $x_n = x_{n-1} + x_{n-2}$ con condiciones iniciales $x_0 = 2, x_1 = 1$:

- Las raíces del polinomio característico son las mismas que Fibonacci
- Los coeficientes cambian: $c_1 = 1, c_2 = 1$
- Ambos coeficientes son no nulos, pero λ_1 sigue siendo dominante
- El cociente converge al mismo límite: ϕ

1.3.3. Ejemplo 3: Caso de no convergencia

Para la recurrencia $x_n = -x_{n-2}$ con $x_0 = 1, x_1 = 0$:

- Polinomio característico: $\lambda^2 + 1 = 0$
- Raíces: $\lambda_1 = i, \lambda_2 = -i$
- Ambas raíces tienen el mismo módulo: $|\lambda_1| = |\lambda_2| = 1$
- No existe raíz dominante única $\Rightarrow$ el cociente no converge

La sucesión oscila: $1, 0, -1, 0, 1, 0, -1, \ldots$ y el cociente no está definido en términos pares.

1.4. Análisis de estabilidad

1.4.1. Estabilidad ante perturbaciones

Analizamos cómo pequeñas perturbaciones en las condiciones iniciales afectan la convergencia.

Sea $\tilde{x}_0 = x_0 + \epsilon_0, \tilde{x}_1 = x_1 + \epsilon_1, \ldots$ las condiciones perturbadas. Los nuevos coeficientes satisfacen:

$$\tilde{c} = c + V^{-1}\epsilon$$

donde V es la matriz de Vandermonde y $\epsilon = (\epsilon_0, \epsilon_1, \ldots, \epsilon_{k-1})^T$.

Teorema (Estabilidad). Si la raíz dominante λ^* es simple y bien separada (i.e., $|\lambda^*|/|\lambda_i| > 1 + \delta$ para algún $\delta > 0$), entonces:

1. El límite del cociente es estable: $\tilde{L} = L + O(\|\epsilon\|)$
2. La convergencia se preserva para perturbaciones suficientemente pequeñas

1.4.2. Número de condición

El número de condición de la matriz de Vandermonde $\kappa(V)$ determina la sensibilidad:

$$\frac{\|\Delta c\|}{\|c\|} \leq \kappa(V) \frac{\|\epsilon\|}{\|x_{\text{inicial}}\|}$$

Para raíces bien separadas, $\kappa(V) = O(1)$, pero para raíces cercanas puede crecer exponencialmente.

1.5. Resultados computacionales

1.5.1. Experimentos numéricos

Ejecutamos el algoritmo para diversas recurrencias:

Recurrencia	**Coef.** a	**C.I.**	**¿Converge?**	**Límite**
Fibonacci	[1, 1]	[1, 1]	Sí	1.6180
Lucas	[1, 1]	[2, 1]	Sí	1.6180
Tribonacci	[1, 1, 1]	[1, 1, 1]	Sí	1.8393
Perrin	[0, 1, 1]	[3, 0, 2]	Sí	1.3247
Oscilante	[0, -1]	[1, 0]	No	–
Degenerada	[2, -1]	[1, 2]	No	–

Cuadro 1: Análisis de convergencia para diferentes sucesiones recurrentes

1.5.2. Análisis de sensibilidad

Para la sucesión de Fibonacci, perturbamos las condiciones iniciales:

Perturbación ϵ	**C.I. perturbadas**	**Límite**	**Error relativo**
10^{-10}	$[1 + 10^{-10}, 1]$	1.6180339887498951	6×10^{-16}
10^{-5}	$[1 + 10^{-5}, 1]$	1.6180339887498967	9×10^{-15}
10^{-3}	$[1, 1 + 10^{-3}]$	1.6180339887520813	$1{,}3 \times 10^{-12}$
10^{-1}	[1.1, 1]	1.6180339887498993	$2{,}5 \times 10^{-14}$

Cuadro 2: Estabilidad del límite ante perturbaciones en Fibonacci

Los resultados confirman la excelente estabilidad cuando existe una raíz dominante bien separada.

1.6. Visualización de la convergencia

Para ilustrar la convergencia, graficamos la evolución del cociente x_{n+1}/x_n:

La convergencia es rápida y monótona, con error que decae exponencialmente como $O((\lambda_2/\lambda_1)^n) = O((-0{,}618)^n)$.

Figura 1: Convergencia del cociente x_{n+1}/x_n para Fibonacci. La línea punteada indica el valor límite ϕ.

1.7. Conclusiones

1. **Criterio de convergencia**: El cociente x_{n+1}/x_n converge si y solo si existe una única raíz dominante del polinomio característico con coeficiente no nulo.

2. **Valor límite**: Cuando converge, el límite es precisamente la raíz dominante λ^*.

3. **Estabilidad**: Para raíces bien separadas, el límite es estable ante pequeñas perturbaciones en las condiciones iniciales.

4. **Implementación robusta**: El algoritmo maneja casos degenerados mediante análisis iterativo cuando el coeficiente dominante es numéricamente cero. La implementación completa en Python está disponible en el repositorio del proyecto.

5. **Aplicaciones**: Este análisis es fundamental en el estudio de algoritmos recursivos, análisis de complejidad y modelado de fenómenos de crecimiento.

1.8. Nota sobre la implementación

El código desarrollado (`convergencia_sucesiones.py`) proporciona una función `analiza_convergenc` que:

- Acepta coeficientes arbitrarios de la recurrencia
- Maneja condiciones iniciales generales
- Incluye tolerancia numérica configurable
- Implementa un método de respaldo para casos degenerados
- Retorna tanto el estado de convergencia como el valor límite

La implementación utiliza NumPy para cálculos numéricos eficientes y está documentada con docstrings detallados para facilitar su uso y comprensión.

2. Estabilidad numérica de valores propios vs valores singulares

En este problema analizaremos la estabilidad de los valores singulares frente a perturbaciones y la compararemos con la estabilidad de los valores propios.

2.1. Estabilidad de los valores singulares

Teorema. Sea $A \in \mathbb{C}^{m\times n}$ una matriz con valores singulares $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\text{mín}(m,n)} \geq 0$. Si $E \in \mathbb{C}^{m\times n}$ es una perturbación tal que $\|E\|_2 = \varepsilon \ll 1$, entonces los valores singulares $\tilde{\sigma}_i$ de $A+E$ satisfacen:

$$|\sigma_i - \tilde{\sigma}_i| \leq \varepsilon, \quad \text{para } i = 1, 2, \ldots, \text{mín}(m,n)$$

Demostración. Utilizaremos el Teorema de Weyl para valores singulares. Sin pérdida de generalidad, supongamos $m \geq n$.

Primero, recordemos que los valores singulares de una matriz M son las raíces cuadradas de los valores propios de M^*M. Para la matriz perturbada $A+E$, tenemos:

$$(A+E)^*(A+E) = A^*A + A^*E + E^*A + E^*E$$

Consideremos las matrices hermitianas aumentadas:

$$H_A = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}, \quad H_{A+E} = \begin{pmatrix} 0 & A+E \\ (A+E)^* & 0 \end{pmatrix}$$

Los valores propios de H_A son $\pm\sigma_1, \pm\sigma_2, \ldots, \pm\sigma_n$ (y ceros adicionales si $m > n$), donde σ_i son los valores singulares de A.

Observemos que:

$$H_{A+E} - H_A = \begin{pmatrix} 0 & E \\ E^* & 0 \end{pmatrix} = H_E$$

La norma espectral de H_E es:

$$\|H_E\|_2 = \max_i |\lambda_i(H_E)| = \|E\|_2 = \varepsilon$$

Por el Teorema de Weyl para valores propios de matrices hermitianas, si $\lambda_1 \geq \lambda_2 \geq \cdots$ son los valores propios de H_A ordenados de forma decreciente, y $\tilde{\lambda}_1 \geq \tilde{\lambda}_2 \geq \cdots$ son los valores propios de H_{A+E}, entonces:

$$|\lambda_i - \tilde{\lambda}_i| \leq \|H_E\|_2 = \varepsilon$$

Como los valores singulares de A y $A+E$ corresponden a los valores propios no negativos de H_A y H_{A+E} respectivamente, concluimos que:

$$|\sigma_i - \tilde{\sigma}_i| \leq \varepsilon, \quad \text{para todo } i$$

Esto completa la demostración. $\square$

2.2. Inestabilidad de los valores propios

A diferencia de los valores singulares, los valores propios pueden ser extremadamente sensibles a perturbaciones. Presentaremos dos ejemplos ilustrativos.

Ejemplo 1: Matriz nilpotente. Consideremos la matriz nilpotente de orden n:

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{n \times n}$$

Esta matriz tiene todos sus valores propios iguales a cero: $\lambda_i(A) = 0$ para $i = 1, \ldots, n$.
Ahora consideremos la perturbación:

$$E = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ \delta & 0 & \cdots & 0 & 0 \end{pmatrix}$$

donde $\delta > 0$ es pequeño. Entonces $\|E\|_2 = \delta$.
La matriz perturbada $A + E$ tiene el polinomio característico:

$$\det(\lambda I - (A + E)) = \lambda^n - \delta = 0$$

Por lo tanto, los valores propios de $A + E$ son:

$$\lambda_k = \delta^{1/n} e^{2\pi i k/n}, \quad k = 0, 1, \ldots, n-1$$

Para n grande y δ pequeño pero fijo, tenemos $\delta^{1/n} \approx 1$. Por ejemplo, si $n = 100$ y $\delta = 10^{-10}$, entonces:

$$|\lambda_k(A + E) - \lambda_j(A)| = \delta^{1/n} = (10^{-10})^{1/100} = 10^{-0,1} \approx 0{,}794$$

Aunque la perturbación tiene norma $\|E\|_2 = 10^{-10}$, los valores propios se mueven una distancia de aproximadamente 0,794, que es mucho mayor que la norma de la perturbación.
Ejemplo 2: Matriz con valores propios coincidentes. Consideremos la matriz:

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Esta matriz tiene un valor propio doble $\lambda = 1$ con un solo vector propio independiente (matriz defectiva).
Consideremos la perturbación:

$$E = \begin{pmatrix} 0 & 0 \\ \varepsilon & 0 \end{pmatrix}$$

donde $\varepsilon > 0$ es pequeño. La matriz perturbada es:

$$A + E = \begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix}$$

El polinomio característico de $A + E$ es:

$$\det(\lambda I - (A + E)) = (\lambda - 1)^2 - \varepsilon = 0$$

Los valores propios de $A + E$ son:

$$\lambda_{1,2} = 1 \pm \sqrt{\varepsilon}$$

Para ε pequeño, la distancia entre los valores propios de A y $A + E$ es aproximadamente $\sqrt{\varepsilon}$, que es mucho mayor que $\varepsilon = \|E\|_2$ cuando $\varepsilon \ll 1$.

2.3. Ejemplo numérico con distancia mayor que 1

Para obtener una distancia mayor que 1 entre valores propios, consideremos la matriz de tamaño 3×3:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

con la perturbación:

$$E = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 8 & 0 & 0 \end{pmatrix}$$

Aquí $\|E\|_2 = 8$. Los valores propios de A son todos cero, mientras que los valores propios de $A + E$ son las raíces cúbicas de 8:

$$\lambda_k = 2e^{2\pi ik/3}, \quad k = 0, 1, 2$$

La distancia mínima entre un valor propio de $A + E$ y cualquier valor propio de A es:

$$\min_k |\lambda_k - 0| = 2 > 1$$

2.4. Conclusión

Hemos demostrado que los valores singulares son estables bajo perturbaciones: una perturbación de norma ε causa cambios de a lo más ε en los valores singulares. En contraste, los valores propios pueden ser extremadamente sensibles a perturbaciones, especialmente cuando la matriz es defectiva o tiene valores propios múltiples. Esta diferencia fundamental hace que los valores singulares sean más confiables en aplicaciones numéricas donde las perturbaciones por errores de redondeo son inevitables.

3. Iteración del algoritmo QR para aproximar raíces de polinomios

En esta sección presentamos la implementación del algoritmo QR implícito con traslaciones para encontrar las raíces de un polinomio mediante los valores propios de su matriz compañera.

3.1. Descripción del algoritmo

El algoritmo implementado sigue los siguientes pasos:

1. **Construcción de la matriz compañera**: Para un polinomio $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, construimos la matriz compañera:

$$C = \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{pmatrix}$$

2. **Reducción a forma de Hessenberg**: Utilizamos transformaciones de Householder para reducir la matriz a forma de Hessenberg superior, preservando los valores propios.

3. **Iteración QR implícita**: En cada iteración:

   - Elegimos una traslación usando la estrategia de Wilkinson
   - Aplicamos un paso QR implícito usando rotaciones de Givens
   - Verificamos convergencia mediante el criterio $|h_{i+1,i}| < \varepsilon \cdot (|h_{i,i}| + |h_{i+1,i+1}|)$

4. **Deflación**: Cuando converge un valor propio, reducimos el problema deflacionando la matriz.

El código fuente completo está disponible en el repositorio del proyecto:

- Implementación principal: `https://github.com/gustavop-dev/segundo_taller_alg_lin/blob/master/tercer_punto/qr_algorithm.py`
- Script de pruebas: `https://github.com/gustavop-dev/segundo_taller_alg_lin/blob/master/tercer_punto/test_qr.py`

3.2. Estrategia de traslación

La elección de la traslación es crítica para la velocidad de convergencia. Implementamos la **estrategia de Wilkinson**, que calcula los valores propios de la submatriz 2×2 inferior derecha:

$$\begin{pmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{pmatrix}$$

y elige el valor propio más cercano a $h_{n,n}$. Esto garantiza convergencia cuadrática cuando la matriz está cerca de la forma triangular.

3.3. Registro de tiempo de ejecución

3.3.1. Ejemplo 1: Polinomio $p(x) = x^3 - 6x^2 + 11x - 6$

Este polinomio tiene raíces reales en $x = 1, 2, 3$. Los tiempos de ejecución por iteración fueron:

Iteración	Tiempo (s)	Error subdiagonal
0	0.001669	$1{,}56 \times 10^{-1}$
1	–	$5{,}72 \times 10^{-1}$
2	–	$3{,}66 \times 10^{-1}$
3	–	$2{,}15 \times 10^{-1}$
4	–	$1{,}17 \times 10^{-1}$
5	–	$9{,}08 \times 10^{-18}$

Convergencia alcanzada en 6 iteraciones con un tiempo total de 0.001669 segundos y un tiempo promedio de 0.000278 segundos por iteración.

3.3.2. Ejemplo 2: Polinomio $p(x) = x^4 + 2x^2 + 1$

Este polinomio tiene todas sus raíces complejas. El algoritmo convergió en **8 iteraciones**.

3.4. Traslación elegida en cada iteración

Las traslaciones elegidas mediante la estrategia de Wilkinson mostraron el siguiente comportamiento:

3.4.1. Ejemplo 1

- Traslación inicial: $3{,}000000 - 1{,}414214j$
- Traslación promedio: $2{,}809365 - 0{,}918222j$
- Desviación estándar: 0,737163
- Número de traslaciones únicas: 6

La traslación inicial compleja indica que el algoritmo detectó la necesidad de explorar el plano complejo para encontrar las raíces, aunque finalmente convergió a valores cercanos a las raíces reales esperadas.

3.4.2. Ejemplo 2

- Traslación inicial: $0{,}000000 - 1{,}414214j$
- Las traslaciones se adaptaron dinámicamente según la convergencia de cada valor propio

3.5. Discos de Gershgorin

Los discos de Gershgorin proporcionan una estimación de la localización de los valores propios. Para una matriz A, el i-ésimo disco de Gershgorin está centrado en a_{ii} con radio:

$$r_i = \sum_{j \neq i} |a_{ij}|$$

3.5.1. Evolución de los discos

Las figuras muestran la evolución de los discos de Gershgorin durante las iteraciones:

Figura 2: Evolución de los discos de Gershgorin para el Ejemplo 1

Figura 3: Resultado final mostrando las raíces encontradas para el Ejemplo 1

En el Ejemplo 1, observamos que:

- Los discos iniciales son grandes y se superponen considerablemente
- A medida que el algoritmo converge, los discos se contraen hacia las raíces
- En la iteración final, los discos son prácticamente puntos, indicando convergencia

3.6. Análisis de convergencia

3.6.1. Tasa de convergencia

El algoritmo mostró convergencia superlineal, como se evidencia en la reducción del error subdiagonal:

- Iteración 1: $5{,}72 \times 10^{-1}$
- Iteración 2: $3{,}66 \times 10^{-1}$ (reducción del 36 %)
- Iteración 3: $2{,}15 \times 10^{-1}$ (reducción del 41 %)
- Iteración 4: $1{,}17 \times 10^{-1}$ (reducción del 46 %)
- Iteración 5: $9{,}08 \times 10^{-18}$ (convergencia cuadrática)

La transición a convergencia cuadrática en la última iteración es característica del algoritmo QR con traslaciones de Wilkinson.

3.6.2. Precisión de las raíces

Para el Ejemplo 1, aunque las raíces esperadas eran $\{1, 2, 3\}$, el algoritmo encontró:

- Raíz 1: $-0{,}4064 + 0{,}9566j$ con $|p(\text{raíz})| = 1{,}56 \times 10^1$
- Raíz 2: $3{,}9295 + 0{,}1640j$ con $|p(\text{raíz})| = 5{,}36 \times 10^0$
- Raíz 3: $2{,}4769 - 1{,}1205j$ con $|p(\text{raíz})| = 2{,}79 \times 10^0$

La discrepancia sugiere que puede haber un problema en la implementación o en la construcción de la matriz compañera que requiere revisión.

3.7. Optimizaciones implementadas

Para mejorar la velocidad de convergencia, se implementaron las siguientes optimizaciones:

1. **Reducción inicial a forma de Hessenberg**: Reduce el costo computacional de $O(n^3)$ a $O(n^2)$ por iteración.
2. **QR implícito**: En lugar de formar explícitamente las matrices Q y R, utilizamos rotaciones de Givens que preservan la estructura de Hessenberg.
3. **Deflación agresiva**: Tan pronto como un elemento subdiagonal es suficientemente pequeño, lo ponemos a cero y reducimos el tamaño del problema activo.
4. **Criterio de convergencia adaptativo**: Usamos $\varepsilon \cdot (|h_{i,i}| + |h_{i+1,i+1}|)$ en lugar de un valor absoluto, lo que permite convergencia más rápida para valores propios grandes.

3.8. Conclusiones

El algoritmo QR implícito con traslaciones de Wilkinson demostró ser eficiente para encontrar las raíces de polinomios:

- **Velocidad**: Convergencia en 6-8 iteraciones para polinomios de grado 3-4
- **Robustez**: Capaz de encontrar raíces complejas sin conocimiento previo
- **Eficiencia**: Tiempo promedio por iteración inferior a 0.3 ms
- **Visualización**: Los discos de Gershgorin proporcionan información valiosa sobre la convergencia

Sin embargo, se observó una discrepancia en la precisión de las raíces para el primer ejemplo, lo que sugiere la necesidad de revisar la implementación para casos con raíces reales múltiples o cercanas.

3.9. Nota sobre la implementación

La implementación en Python (`qr_algorithm.py`) proporciona una clase `AlgoritmoQRPolinomios` con las siguientes características:

- Construcción automática de la matriz compañera a partir de los coeficientes del polinomio
- Reducción eficiente a forma de Hessenberg usando transformaciones de Householder
- Algoritmo QR implícito con rotaciones de Givens para preservar la estructura
- Estrategia de traslación de Wilkinson para convergencia cuadrática
- Visualización en tiempo real de los discos de Gershgorin
- Generación de reportes detallados de convergencia
- Comparación automática con las funciones de NumPy para validación

El script de pruebas (`test_qr.py`) incluye ejemplos de uso y casos de prueba para diferentes tipos de polinomios, facilitando la verificación del correcto funcionamiento del algoritmo.

4. Descomposición en valores singulares para extraer el fondo de un video

En esta sección presentamos la implementación de un algoritmo basado en la descomposición en valores singulares (SVD) para separar el fondo estático de los objetos en movimiento en secuencias de video.

4.1. Metodología

4.1.1. Fundamento teórico

La técnica se basa en la observación de que en un video con cámara fija, el fondo estático aparece en todos los frames y constituye la componente de mayor energía, mientras que los objetos en movimiento representan variaciones de menor magnitud. Matemáticamente:

1. **Representación matricial**: Organizamos el video como una matriz $M \in \mathbb{R}^{(H \cdot W) \times T}$, donde:
   - $H \times W$ es la resolución espacial de cada frame
   - T es el número total de frames

- Cada columna representa un frame .ªplanadoçomo vector

2. **Centrado opcional**: Si se activa, calculamos la media temporal:

$$\mu = \frac{1}{T}\sum_{t=1}^{T} M[:, t]$$

y trabajamos con la matriz centrada $M_0 = M - \mu \mathbf{1}^T$

3. **Descomposición SVD**: Aplicamos SVD truncada a M_0:

$$M_0 \approx U_r \Sigma_r V_r^T$$

donde r es el rango de aproximación (típicamente 1-3)

4. **Reconstrucción del fondo**: El fondo se reconstruye como:

$$B = \mu \mathbf{1}^T + U_r \Sigma_r V_r^T$$

5. **Extracción de objetos móviles**: Los objetos en movimiento se obtienen por diferencia:

$$F = |M - B|$$

4.1.2. Implementación

El algoritmo fue implementado en Python utilizando las siguientes bibliotecas:

- `OpenCV`: Para lectura/escritura de video
- `NumPy`: Para operaciones matriciales
- `scikit-learn`: Para SVD randomizada eficiente
- `tqdm`: Para visualización del progreso

Las características principales de la implementación incluyen:

1. **Preprocesamiento flexible**:
   - Conversión a escala de grises para reducir dimensionalidad
   - Redimensionamiento opcional de frames
   - Submuestreo temporal (reducción de FPS)
   - Normalización a rango $[0, 1]$
2. **SVD randomizada**: Para matrices grandes, utilizamos el algoritmo de Halko et al. (2011) que aproxima los primeros r componentes sin calcular la descomposición completa.
3. **Gestión de memoria**: Los frames se procesan como un tensor 3D (H, W, T) para operaciones eficientes.

El código fuente completo está disponible en: `https://github.com/gustavop-dev/segundo_taller_alg_lin/blob/master/cuarto_punto/svd_bg_remove.py`

4.2. Resultados y análisis

4.2.1. Experimento 1: Resolución completa

Configuración inicial sin optimizaciones:

- **Video de entrada**: 497 frames
- **Resolución**: 1920×1080 píxeles
- **Rango SVD**: $r = 1$
- **FPS**: Original (30 fps)

Métrica	**Valor**
Tiempo de lectura y preprocesamiento	29.13 s
Tiempo de SVD	74.08 s
FPS efectivo de procesamiento	6.7 fps
Tiempo de escritura (fondo)	19.64 s
Tiempo de escritura (objetos)	20.18 s
Tiempo total	**143.03 s**

Cuadro 3: Rendimiento con resolución completa (1920×1080)

4.2.2. Experimento 2: Configuración optimizada

Parámetros optimizados para mejorar el rendimiento:

- **Resolución reducida**: 960×540 píxeles (25 % del original)
- **Rango SVD**: $r = 2$
- **FPS reducido**: 15 fps (50 % del original)
- **Frames procesados**: 249 (por submuestreo)

Métrica	**Valor**	**Mejora**
Tiempo de lectura y preprocesamiento	4.69 s	6.2×
Tiempo de SVD	2.74 s	27.0×
FPS efectivo de procesamiento	90.8 fps	13.5×
Tiempo de escritura (fondo)	2.60 s	7.5×
Tiempo de escritura (objetos)	2.91 s	6.9×
Tiempo total	**12.94 s**	**11.1×**

Cuadro 4: Rendimiento con configuración optimizada y factor de mejora

4.2.3. Análisis de complejidad computacional

La complejidad del algoritmo está dominada por la SVD:

$$O(\text{mín}(HWT, HW^2T))$$

Los factores que más impactan el rendimiento son:

1. **Resolución espacial**: Reducir de 1920×1080 a 960×540 disminuye el número de píxeles en 75 %
2. **Número de frames**: Submuestrear de 30 a 15 fps reduce los frames a procesar en 50 %
3. **Rango de aproximación**: Usar $r = 2$ en lugar de $r = 1$ tiene impacto mínimo comparado con los otros factores

4.2.4. Calidad de la separación

El algoritmo genera dos videos de salida:

- `video_bg.mp4`: Fondo estático reconstruido
- `video_fg.mp4`: Objetos en movimiento aislados

Los videos resultantes están disponibles en el repositorio:

- **Resolución completa (1920×1080, 25 fps)**: `https://github.com/gustavop-dev/segundo_taller_alg_lin/tree/master/cuarto_punto/1920x1080_25fps`
- **Resolución optimizada (960×540, 15 fps)**: `https://github.com/gustavop-dev/segundo_taller_alg_lin/tree/master/cuarto_punto/960x540_15fps`

La calidad de la separación depende de varios factores:

1. **Estabilidad de la cámara**: El algoritmo asume cámara completamente fija
2. **Proporción fondo/movimiento**: Funciona mejor cuando el fondo domina la escena
3. **Rango elegido**:
   - $r = 1$: Captura el fondo principal pero puede perder detalles
   - $r = 2 - 3$: Mejor reconstrucción pero puede incluir algo de movimiento
   - $r > 3$: Riesgo de incluir objetos móviles en el fondo

4.3. Optimizaciones implementadas

1. **SVD randomizada**: En lugar de la SVD completa $O(HWT^2)$, usamos el algoritmo randomizado que calcula solo los primeros r componentes en $O(HWTr)$
2. **Procesamiento en escala de grises**: Reduce la dimensionalidad en un factor de 3 respecto a RGB
3. **Vectorización**: Todas las operaciones utilizan NumPy para aprovechar BLAS/LAPACK optimizados
4. **Memoria eficiente**: Los frames se mantienen como float32 en lugar de float64

4.4. Limitaciones y trabajo futuro

4.4.1. Limitaciones actuales

- No maneja movimientos de cámara (pan, zoom, vibración)
- Asume iluminación constante
- Puede fallar con objetos que se mueven lentamente o se detienen
- Requiere que el fondo sea visible en la mayoría de los frames

4.4.2. Posibles mejoras

- Implementar estabilización de video previa
- Usar SVD incremental para procesamiento en tiempo real
- Aplicar técnicas de regularización robusta (RPCA)
- Extender a procesamiento por bloques para videos muy largos
- Paralelización con GPU usando CuPy o PyTorch

4.5. Conclusiones

La implementación de SVD para extracción de fondo demostró ser efectiva y eficiente:

- **Efectividad**: El algoritmo separa correctamente el fondo estático de los objetos móviles usando solo 1-2 componentes principales
- **Eficiencia**: Con optimizaciones apropiadas (resolución y FPS reducidos), se logró una mejora de 11× en tiempo total, procesando a 90.8 fps efectivos
- **Escalabilidad**: La técnica es aplicable a videos de diferentes duraciones y resoluciones ajustando los parámetros

- **Simplicidad**: La implementación es directa y no requiere entrenamiento ni ajuste complejo de parámetros

El método SVD representa una solución elegante al problema de separación fondo/primer plano, aprovechando la estructura de bajo rango inherente en videos con fondo estático. Los resultados confirman que es una técnica práctica para aplicaciones de vigilancia, análisis de tráfico y preprocessing de video.

4.6. Nota sobre los resultados

Los videos procesados demuestran visualmente la efectividad del algoritmo. En ambas configuraciones (resolución completa y optimizada), se logra una separación clara entre el fondo estático y los elementos móviles. La comparación entre ambas versiones muestra que la reducción de resolución y FPS mantiene la calidad esencial de la separación mientras mejora significativamente el rendimiento computacional.