DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo práctico 1

Eliminación gaussiana, matrices tridiagonales y difusión

Métodos numéricos

2do cuatrimestre 2023

Integrante	LU	Correo electrónico
Tomás Bossi	50/17	tomasbossi97@gmail.com
Ignacio Enrique Niesz	722/10	ignacio.niesz@gmail.com
Brian Ivan Rios	917/19	ivan.rios2010@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

https://exactas.uba.ar

Índice

1. Introducción

Cualquier sistema de ecuaciones lineales puede ser representado en forma matricial como $Ax = b$, donde A es la matriz de los coeficientes que acompañan a cada una de las incognitas, x es el vector de incognitas y b es el vector de términos independientes. Resolver un sistema de ecuaciones lineales significa, entonces, dados A y b encontrar x tal que $Ax = b$. Son particularmente interesantes los sistemas con igual cantidad de ecuaciones que incognitas, representables por una matriz de coeficientes A cuadrada, que pueden según A y b no tener solución, tener solución única, o tener infinitas soluciones.

Existen algoritmos que permiten hallar la solución única de un dado sistema de ecuaciones, si es que tal solución existe. Los más conocidos y sencillos de estos algoritmos son los de eliminación gaussiana (EG), que en sus versiones más básicas consisten en realizar operaciones elementales entre filas que convierten al sistema en sistemas equivalentes (con el mismo vector solución x) progresivamente simplificados. Una vez que se llega a un sistema equivalente suficientemente sencillo, que en general consiste en un sistema en el que la matriz de coeficientes es triangular superior, los elementos de x son despejados uno por uno. Todos los algoritmos de EG se basan en la operación elemental de reemplazo de una fila por la resta entre sí misma y un múltiplo de otra. Algunos algoritmos usan además la operación elemental de intercambio entre filas para resolver sistemas de otra manera irresolubles y disminuir el error numérico en las soluciones halladas, a lo que se le denomina "pivoteo parcial"[1][2].

Los algoritmos de EG de propósito general, diseñados para sistemas dados por cualquier matriz de coeficientes cuadrada, son de complejidad $O(n^3)$. Para matrices ralas (con la mayoría de sus elementos nulos) pertenecientes a una dada familia puede ser posible diseñar algoritmos de una complejidad menor. Un caso trivial es el de los sistemas diagonales de solución única (con matriz diagonal, sin ceros en la diagonal), que ya están prácticamente resueltos. Estos sistemas requieren de a lo sumo n divisiones para hallar x, y son por lo tanto de complejidad $O(n)$. Para sistemas tridiagonales (con matriz de coeficientes tridiagonal), se sabe que puede diseñarse un algoritmo que también es de complejidad $O(n)$.

El desarrollo de este trabajo puede ser dividido en cuatro partes. En la primera (sección 2) se diseñaron algoritmos de EG de propósito general sin y con pivoteo parcial (2.1 y 2.2) y se analizaron casos particulares de uso interesantes (2.3). En la segunda parte (sección 3) se desarrolló un algoritmo de complejidad $O(n)$ específico a sistemas tridiagonales (3.1), y se lo mejoró separándolo en una primera etapa de precómputo y una segunda etapa de resolución de el o los sistemas lineales dados (3.2). En la tercera parte (sección 4) se realizaron análisis de los tiempos de cómputo de los algoritmos diseñados, estudiando entre otras cosas su dependencia con el tamaño de las matrices input y verificando que sus complejidades algorítmicas sean las esperadas. Finalmente, la cuarta parte del desarrollo (sección 5) explora aplicaciones de los sistemas tridiagonales, en particular a los problemas de la búsqueda de funciones a partir de sus derivadas segundas (5.1) y al de la difusión en el tiempo y el espacio en una dimensión (5.2).

2. Sistemas de ecuaciones

2.1. Eliminación gaussiana sin pivoteo

El algoritmo de EG sin pivoteo (**alg. (1)**) es el más sencillo y elemental de los algoritmos de su tipo, y consta de dos etapas. La primera consiste en eliminar los elementos debajo de la diagonal de la matriz aumentada de coeficientes, yendo desde la primera fila hasta la última y usando como pivote al elemento en la diagonal correspondiente a cada paso de la triangulación (**alg. (2)**). Si el algoritmo encuentra un 0 en la posición de la diagonal que corresponde al paso actual de triangulación, no puede continuar ni hallar la solución del sistema. Si el algoritmo termina de triangular a la matriz y no quedan elementos nulos en su diagonal puede realizarse la segunda etapa, que consiste en despejar el valor de todos los elementos del vector solución x, yendo desde la última fila de la matriz hacia la primera, a lo que se le conoce como *backwards substitution* (**alg. (3)**).

Algorithm 1 EG sin pivoteo

Input: A matriz de coeficientes cuadrada
Input: b vector de términos independientes del sistema Ax=b
Output: x vector columna solución única del sistema Ax=b. Si no existe solución única o no puede encontrarse sin realizar pivoteo, devuelve una excepción

1: **function** *solucion_sistema_lineal_eliminacion_gaussiana_sin_pivoteo*(A, b)
2: $A' \leftarrow concatenar(A, b)$ ▷ A' es la matriz aumentada
3: $A' \leftarrow escalonar\_matriz\_sin\_pivoteo(A')$ ▷ **alg. (2)**
4: $x \leftarrow solucion\_unica\_sistema\_lineal(A', 0)$ ▷ **alg. (3)**
5: **return** x

Algorithm 2 Triangulación sin pivoteo

Input: A' matriz de coeficientes aumentada de dimensiones nx$n+1$
Output: A' matriz de coeficientes aumentada triangulada sin pivoteo. Si no puede realizarse la triangulación sin pivoteo (se encuentra un 0 en la diagonal en algún paso) devuelve una excepción

1: **function** *escalonar_matriz_sin_pivoteo*(A')
2: **for** $i = 1 \dots n$ **do**
3: **if** $a'_{ii} == 0$ && $a'_{i+1,\dots,n;i} == 0$ **then**
4: **then return Exception:** "No puede encontrarse solución única"
5: **for** $j = i + 1 \dots n$ **do**
6: $m_{ij} \leftarrow a'_{ji}/a'_{ii}$
7: $A'_j \leftarrow A'_j - A'_i * m_{ij}$ ▷ **alg. (11)** y **alg. (12)**
8: **return** A'

Algorithm 3 Solución única de sistema pretriangulado

Input: A' matriz de coeficientes aumentada triangulada de dimensiones nx$n+1$
Input: *atol* tolerancia absoluta para la comparación con el 0
Output: x vector columna solución única del sistema Ax=b. Si no existe solución única o no puede encontrarse sin realizar pivoteo, devuelve una excepción Si en algún paso se debe dividir por un número cercano a 0 (cercano de acuerdo a *atol*), imprime una advertencia

1: **function** *solucion_unica_sistema_lineal*(A')
2: $x \leftarrow [\,]$
3: **for** $i = n \dots 1$ **do**
4: **if** $a'_{ii} == 0$ **then return Exception:** "No puede encontrarse solución única"
5: **if** $|a'_{ii}| < atol$ **then print:** "Advertencia: división por valor cercano a 0"
6: $b_i \leftarrow a'_{i(n+1)}$ ▷ término independiente
7: $suma\_terminos \leftarrow x \cdot \left[\, a'_{i(i+1)}\ a'_{i(i+2)}\ \cdots\ a'_{in} \,\right]$ ▷ **alg. (10)**
8: $x_i \leftarrow (b_i - suma\_terminos)/a_{ii}$
9: $x \leftarrow concatenar([x_i], x)$ ▷ agrega x_i al principio de x
10: **return** x

2.2. Eliminación gaussiana con pivoteo

El algoritmo sin pivoteo puede ser levemente modificado para incorporar pivoteo parcial, que implica la posibilidad de realizar intercambios entre filas de la matriz aumentada de coeficientes (**alg. (4)**, **alg. (5)**). Como se ejemplificará en la sección 2.3, esto permite resolver sistemas que son irresolubles sin pivoteo. Para intercambiar filas el algoritmo busca el mejor pivote posible, aquel elemento de la columna a ser eliminada (sobre la diagonal de la matriz o debajo de ella) de valor máximo en módulo. Esto evita divisiones por números pequeños, reduciendo el error numérico en las soluciones halladas.

Algorithm 4 EG con pivoteo parcial

Input: A matriz de coeficientes cuadrada
Input: b vector de términos independientes del sistema Ax=b
Input: *atol* tolerancia absoluta para la comparación con el 0
Output: x vector columna solución única del sistema Ax=b. Si no existe solución única, devuelve una excepción

1: **function** *solucion_sistema_lineal_eliminacion_gaussiana_pivoteo_parcial*(A, b, *atol*)
2: $A' \leftarrow concatenar(A, b)$ ▷ A' es la matriz aumentada
3: $A' \leftarrow escalonar\_matriz\_pivoteo\_parcial(A', atol)$ ▷ **alg. (5)**
4: $x \leftarrow solucion\_unica\_sistema\_lineal(A', atol)$ ▷ **alg. (3)**
5: **return** x

Algorithm 5 Triangulación con pivoteo parcial

Input: A' matriz de coeficientes aumentada de dimensiones nx$n+1$

Input: *atol* tolerancia absoluta para la comparación con el 0

Output: A' matriz de coeficientes aumentada triangulada con pivoteo parcial. Si se determina que no existe solución única para el sistema $Ax = b$ (se encuentra que 0 es el mejor pivote en algún paso) devuelve una excepción

function $escalonar\_matriz\_pivoteo\_parcial(A', atol)$
  for $i = 0 \dots n$ **do**
    $p \leftarrow a'_{ii}$
    $idp \leftarrow i$
    for $j = i + 1 \dots n$ **do**
      if $|a'_{ji}| > |p|$ **then**
        $p \leftarrow a'_{ji}$
        $idp \leftarrow j$
    $fila\_pivote \leftarrow A'_{idp}$
    $A'_{idp} \leftarrow A'_i$
    $A'_i \leftarrow fila\_pivote$
    if $p == 0$ **then return Exception:** "No puede encontrarse solución única"
    if $|p| < atol$ **then print:** "Advertencia: división por valor cercano a 0"
    for $j = i + 1 \dots n$ **do**
      $m_{ij} \leftarrow a'_{ji}/p$
      $A'_j \leftarrow A'_j - A'_i * m_{ij}$ ▷ **alg. (11)** y **alg. (12)**
  return A'

2.3. Observaciones sobre sistemas particulares

Sean los siguientes sistemas de ecuaciones:

$$(a)\begin{bmatrix}0 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}1\\ 1\end{bmatrix} \quad (b)\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}1\\ 1\end{bmatrix} \quad (c)\begin{bmatrix}2\epsilon & 2\\ \epsilon & 2-\epsilon\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}2+\beta\\ 2\end{bmatrix}$$

Donde $\epsilon = 10^{-15}$ y $\beta = 2{,}220446049250313 \times 10^{-15}$. (a) Es un sistema no resoluble realizando EG sin pivoteo (**alg. (1)**) pero sí con pivoteo parcial (**alg. (4)**), (b) es un sistema no resoluble por EG con o sin pivoteo parcial por tener infinitas soluciones, y (c) es un sistema cuya solución exacta es aproximadamente $x_{real} = \begin{bmatrix}1 & 1\end{bmatrix}^T$, pero que al aplicarle EG con pivoteo parcial con $atol = 10\epsilon$ (**alg. (4)**) ocasiona que dispare la advertencia por división por un número pequeño y produce una solución incorrecta a causa de error numérico, aproximadamente $x_{exp} = \begin{bmatrix}1{,}11 & 1\end{bmatrix}^T$. La matriz del sistema (c) tiene número de condición de aproximadamente $\kappa = 4 \times 10^{15}$.

3. Sistemas de ecuaciones tridiagonales

La representación matricial de un sistema de ecuaciones tridiagonal es de la forma:

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}. \tag{1}$$

La solución del sistema queda determinada por 4 vectores: a, b, c (las diagonales) y d (el vector de términos independientes), todos de largo n, dónde se asume que $a_1 = c_n = 0$. Esto permite diseñar un algoritmo más eficiente en tiempo y espacio que el de EG convencional.

3.1. Eliminación gaussiana para sistemas tridiagonales

La estructura de los sistemas tridiagonales permite el desarrollo de un algoritmo que reduce la complejidad de resolución del sistema de $O(n^3)$ a $O(n)$, descripto en **alg. (6)**.

Algorithm 6 Solución única de sistema tridiagonal

Input: a, b, c, d vectores de largo n con $a_1 = c_n = 0$
Input: $atol$ tolerancia absoluta para la comparación con el 0
Output: x vector columna solución única del sistema tridiagonal. Si no existe solución única, devuelve una excepción

1: **function** solucion_sistema_tridiagonal_eliminacion_gaussiana_cuidadosa(a, b, c, d, $atol$)
2: $x \leftarrow d$ ▷ copia profunda
3: **for** $i = 1 \dots n-1$ **do**
4: **if** $b_i == 0$ **then return Exception:** "No puede encontrarse solución única"
5: **if** $|b_i| < atol$ **then print:** "Advertencia: división por valor cercano a 0"
6: $\alpha \leftarrow a_{i+1}/b_i$
7: $b_{i+1} \leftarrow b_{i+1} - \alpha c_i$
8: $d_{i+1} \leftarrow d_{i+1} - \alpha d_i$
9: **if** $b_n == 0$ **then return Exception:** "No puede encontrarse solución única"
10: **for** $i = n \dots 2$ **do**
11: **if** $|b_i| < atol$ **then print:** "Advertencia: división por valor cercano a 0"
12: $\alpha \leftarrow c_{i-1}/b_i$
13: $d_{i-1} \leftarrow d_{i-1} - \alpha d_i$
14: $x_i \leftarrow d_i/b_i$
15: $x_1 \leftarrow d_1/b_1$
16: **return** x

En un primer paso, el algoritmo recorre a b desde adelante hacia atrás para anular por completo al vector a. Esto genera cambios sobre b y d, pero no sobre c. En un segundo paso, recorre al b modificado desde atrás hacia adelante para anular c, modificando nuevamente a d. Esto resulta en un único vector b' que es diagonal única de la representación matricial del sistema, y un d' que es el vector de términos independientes modificado por todas las operaciones realizadas. A partir de b' y d', hallar x (si existe) es trivial.

3.2. Eliminación gaussiana en dos etapas para sistemas tridiagonales (precómputo)

Si se desea resolver una serie de sistemas de ecuaciones para una misma matriz de coeficientes, variando nada más que el vector de términos independientes, recomputar la enteritud de la EG para cada sistema es un desperdicio de recursos. Resulta más eficiente computar el resultado de realizar EG sobre la matriz de coeficientes una única vez, conservando de alguna manera toda la información necesaria sobre las operaciones realizadas en el proceso. Luego, esas mismas operaciones pueden ser aplicadas en el orden adecuado sobre cualquier vector de términos independientes sin necesidad de volver a operar sobre la matriz de coeficientes. En el caso de sistemas tridiagonales esto es particularmente sencillo, y el precómputo consiste en simplemente aplicar una versión modificada de **alg. (6)** que no recibe d y devuelve b' y el vector de las operaciones realizadas (los factores multiplicativos calculados) en orden. (**alg. (9)**). Luego otro algoritmo puede ser utilizado para, a partir de b', el vector de operaciones, y un vector d, operar sobre d y resolver el sistema trivialmente (**alg. (7)**).

Algorithm 7 Solución única de sistema tridiagonal con precómputo

Input: b' vector diagonal del sistema diagonal
Input: d vector de términos independientes del sistema tridiagonal original
Input: *operaciones* vector de factores multiplicativos necesarios para operar sobre d
Input: *atol* tolerancia absoluta para la comparación con el 0
Output: x vector columna solución única del sistema tridiagonal original. Si no existe solución única, devuelve una excepción

```
1: function solucion_sistema_tridiagonal_precomputado(b', d, operaciones, atol)
2:     x ← d                                                    ▷ copia profunda
3:     d' ← preoperar_d_sistema_tridiagonal(d, operaciones)          ▷ alg. (8)
4:     for i = 1 ... n do
5:         if b'_i == 0 then return Exception: "No puede encontrarse solución única"
6:         if |b'_i| < atol then print: "Advertencia: división por valor cercano a 0"
7:         x_i ← d'_i / b'_i
8:     return x
```

Algorithm 8 Preoperaciones sobre el vector de términos independientes

Input: d vector de longitud n de los términos independientes del sistema tridiagonal
Input: *operaciones* vector de factores multiplicativos necesarios para operar sobre d, de longitud $2 * (n - 1)$
Output: d vector de términos independientes modificado de acuerdo a *operaciones* para corresponder con su sistema diagonal asociado

function preoperar_d_sistema_tridiagonal(d, *operaciones*)
    for $i = 1 \dots n - 1$ **do**
        $d_{i+1} \leftarrow d_{i+1} - operaciones_i * d_i$
    for $i = n \dots 2$ **do**
        $d_{i-1} \leftarrow d_{i-1} - operaciones_{2n-i} * d_i$
    return d

Algorithm 9 EG para sistemas tridiagonales (precómputo)

Input: a, b, c vectores de largo n con $a_1 = c_n = 0$
Input: *atol* tolerancia absoluta para la comparación con el 0
Output: b' vector diagonal del sistema diagonal
Output: *operaciones* vector de factores multiplitcativos utilizados en la EG, en orden

function precomputo_sistema_tridiagonal_EG_cuidadosa(a, b, c, *atol*)
    operaciones $\leftarrow [\,]$
    for $i = 1 \dots n - 1$ **do**
        if $b_i == 0$ **then return Exception:** "No puede encontrarse solución única"
        if $|b_i| < atol$ **then print:** "Advertencia: división por valor cercano a 0"
        $\alpha \leftarrow a_{i+1}/b_i$
        operaciones $\leftarrow concatenar(operaciones, [\alpha])$ ▷ agrega α al final de *operaciones*
        $b_{i+1} \leftarrow b_{i+1} - \alpha c_i$
    if $b_n == 0$ **then return Exception:** "No puede encontrarse solución única"
    for $i = n \dots 2$ **do**
        if $|b_i| < atol$ **then print:** "Advertencia: división por valor cercano a 0"
        $\alpha \leftarrow c_{i-1}/b_i$
        operaciones $\leftarrow concatenar(operaciones, [\alpha])$
    return $b, operaciones$

4. Tiempos de ejecución

Para medir tiempos de cómputo en función del tamaño de la matriz input, se eligió como métrica al tiempo de ejecución mínimo a partir de n_rep repeticiones. Más especificamente, para cada algoritmo y bajo cada condición, se utilizó al comando `%timeit` de IPython[3] para realizar n_rep*k mediciones individuales totales, separadas en grupos de k ejecuciones cada uno. En todos los casos se midió el *CPU User time*[4]. El tiempo informado MT (por Mejor Tiempo) para un dado algoritmo para un dado tamaño del input es el mínimo de los tiempos promedio de cada uno de esos n_rep grupos de k ejecuciones:

$$MT = \min_{i=1,\ldots,n\_rep} \{\frac{1}{k} \sum_{j=1}^{k} med_{ij}\}$$

Donde med_{ij} es la medición individual j (de k mediciones por grupo) del grupo de mediciones i (de n_rep grupos).

Las matrices de prueba fueron en todos los casos precomputadas, de forma tal que el tiempo necesario para su generación no forma parte de las mediciones. Para la comparación entre tanto los algoritmos de EG común con y sin pivoteo como entre EG con pivoteo y EG para tridiagonales, se procuró para cada tamaño de matriz analizado siempre utilizar al mismo grupo de $n\_rep * k$ matrices precomputadas. En todos los casos se usó al vector columna $(1, 1, \ldots, 1)^T$ de la dimensión correspondiente como vector de términos independientes.

Para el caso particular de la medición de tiempos de cómputo para comparar al algoritmo de EG para tridiagonales con y sin precómputo, se utilizó como métrica al mínimo de los tiempos totales de ejecución de cada uno de los n_rep grupos, que equivale simplemente a calcular $k * MT$. Se utilizaron únicamente 4 matrices tridiagonales generadas al azar, una para cada tamaño de matriz analizado (32x32, 64x64, 96x96 y 128x128).

Las mediciones se resumen en las figuras 1, 2 y 3. En todos los casos, se obtuvieron los resultados esperados de acuerdo a nuestros conocimientos previos sobre los algoritmos analizados.

Se encontró que los algoritmos de EG común con y sin pivoteo parcial son muy similares entre sí en cuanto a sus tiempos de cómputo, por lo menos para los tamaños de matriz analizados. Los resultados comprueban además que ambos algoritmos son aproximadamente $O(n^3)$ (figura 1).

Por otro lado, se observó que al operar sobre sistemas tridiagonales el algoritmo diseñado específicamente para este tipo de matrices es muy significativamente más rápido que el algoritmo de EG común con pivoteo parcial (figura 2a). Por ejemplo, para el caso puntual de matrices tridiagonales de tamaño 128x128, el algoritmo de EG para tridiagonales es más de 200 veces más rápido. Esta diferencia en velocidad se debe mayormente a la diferencia cualitativa en la complejidad de estos algoritmos, siendo el algorítmo de EG para tridiagonales de complejidad $O(n)$ (figura 2b).

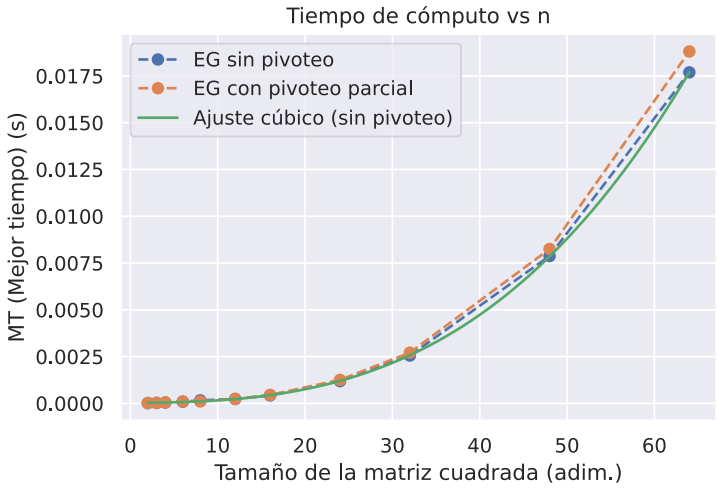
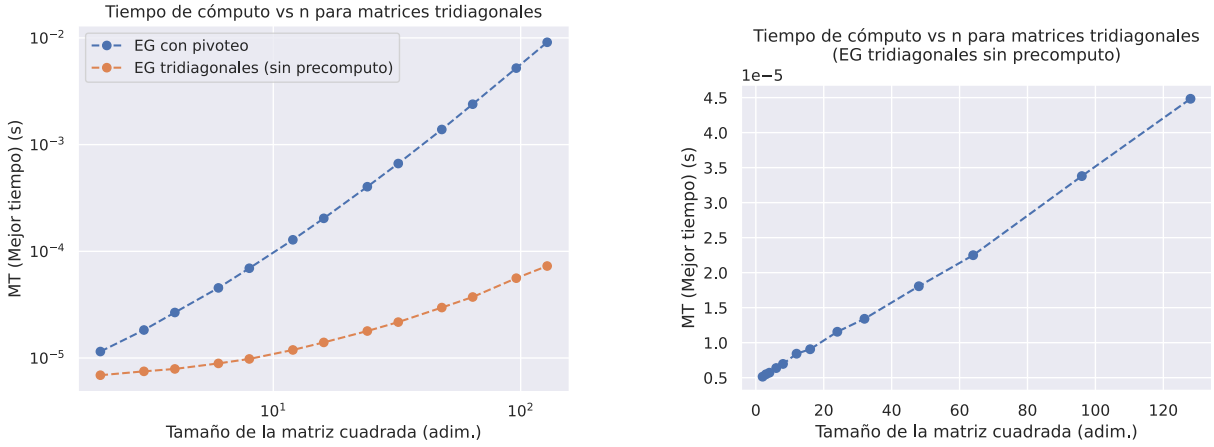


Figura 1: Tiempos de cómputo vs tamaño de la matriz input para los algoritmos de EG convencional con y sin pivoteo (**alg. (4)**, **alg. (1)**). Se utilizó $k = 10$ y $n\_rep = 100$, con 1000 matrices triangulares inferiores pregeneradas para cada tamaño. Se muestra también el mejor ajuste por cuadrados mínimos a los datos (sin pivoteo) con un polinomio grado 3.



(a) (b)

Figura 2: Tiempos de cómputo vs tamaño de la matriz input para los algoritmos de EG convencional con pivoteo y de EG para matrices tridiagonales sin precómputo (**alg. (4)**, **alg. (6)**). **(a)** Comparación entre ambos algoritmos en escala logarítmica. Se utilizó $k = 50$ y $n\_rep = 100$, con 5000 matrices tridiagonales pregeneradas para cada tamaño. **(b)** Resultados para el algoritmo de EG para matrices tridiagonales sin precómputo con $k = 1000$, $n\_rep = 200$ y 200000 triplas de diagonales de matrices tridiagonales pregeneradas.

En cuanto a la diferencia entre el algoritmo de EG para tridiagonales con y sin precómputo, de los resultados puede concluirse que con precómputo el algoritmo es más rápido, aunque no deja de ser de complejidad $O(n)$. El efecto del precómputo es relativamente mayor

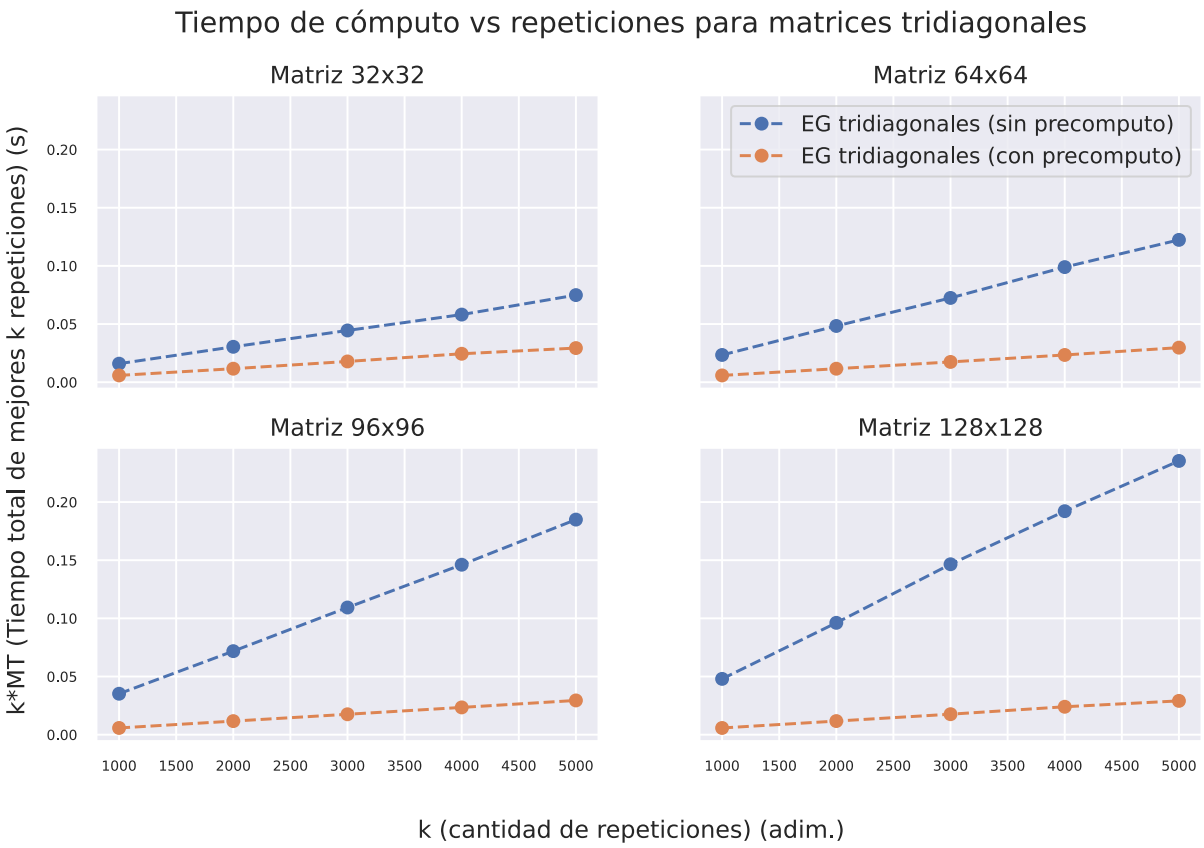


Figura 3: Tiempos de cómputo vs cantidad de repeticiones de la ejecución del programa (k) para los algoritmos de EG para matrices tridiagonales con y sin precómputo (**alg. (7)**, **alg. (6)**). Se utilizo $n\_rep = 100$ con k variable. Para cada uno de los cuatro tamaños de matriz analizados se pregeneró una única tripla de diagonales al azar.

cuantas más repeticiones son realizadas y cuanto más grande (y costosa de preprocesar) es la matriz input, como era esperable (figura 3).

Para lograr resultados estables y reproducibles fue necesario realizar las mediciones de tiempos de ejecución localmente, no en los servidores de Google Colab.

5. Aplicaciones

5.1. Laplaciano discreto

Utilizando L, la matriz tridiagonal del operador laplaciano unidimensional, y mediante el uso del algoritmo de eliminación gaussiana para sistemas tridiagonales con precómputo (**alg. (9)** y **alg. (7)**), se encontraron vectores solución u para los sistemas $Lu = d$ dados por los siguientes vectores de términos independientes, todos de dimensión $n = 101$:

(a) $d_i^a = \begin{cases} 0 \\ 4/n & i = \lfloor n/2 \rfloor + 1 \end{cases}$

(b) $d_i^b = 4/n^2$

(c) $d_i^c = 12(-1 + 2i/(n-1))/n^2$

Como L es la aproximación discreta al operador Laplaciano, haber resuelto estos sistemas implica haber encontrado, de manera aproximada, la función (dada de manera discreta por u) correspondiente a su respectiva derivada segunda (dada de manera discreta por d). La figura 4 muestra el resultado de graficar los vectores solución u^a, u^b y u^c, provenientes de resolver el sistema para sus respectivos d.

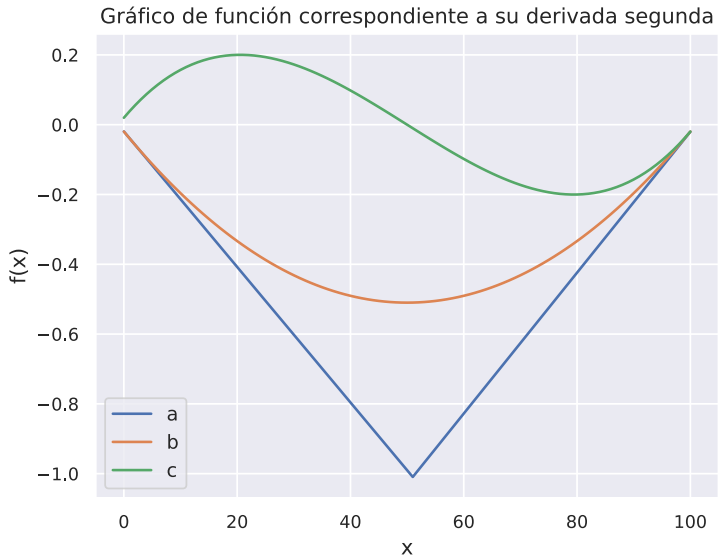


Figura 4: Gráficos de las funciones halladas para cada una de las derivadas segundas dadas por d_a, d_b y d_c. Concretamente, se muestra el resultado de graficar u_a, u_b y u_c vs x, donde para x se eligió de manera arbitraria el rango de 0 a 100 inclusive.

5.2. Simulación de proceso de difusión

Partiendo de la ecuación de difusión discreta en una dimensión,

$$u_i^{(k)} - u_i^{(k-1)} = \alpha(u_{i-1}^{(k)} - 2u_i^{(k)} + u_{i+1}^{(k)}) \tag{2}$$

Distribuyendo α y reordenando los terminos para despejar $u_i^{(k-1)}$ se obtiene:

$$u_i^{(k-1)} = -\alpha u_{i-1}^{(k)} + (2\alpha + 1)u_i^{(k)} - \alpha u_{i+1}^{(k)} \tag{3}$$

De **eq. (3)** se deduce que la matriz D tal que $Du^{(k)} = u^{(k-1)}$ es una matriz tridiagonal de la forma:

$$D = \begin{bmatrix} 2\alpha+1 & -\alpha & & & 0 \\ -\alpha & 2\alpha+1 & -\alpha & & \\ & -\alpha & 2\alpha+1 & \ddots & \\ & & \ddots & \ddots & -\alpha \\ 0 & & & -\alpha & 2\alpha+1 \end{bmatrix}$$

De tamaño nxn, siendo n la cantidad de filas de u.

Tanto en esta forma matricial como en **eq. (3)** queda evidenciado que α es un factor asociado a la tasa de difusión: a mayor α, mayor la proporción de particulas que difunde desde una posición a sus posiciones vecinas desde el tiempo $k-1$ al tiempo siguiente k.

Se computó la evolución espacial y temporal resultante de aplicar iterativamente a D sobre un vector $u^{(0)}$ de dimensión $n = 101$ para distintos valores de α, para un total de 1000 iteraciones, usando EG para matrices tridiagonales con precómputo (**alg. (9)** y **alg. (7)**).

Los elementos de $u^{(0)}$ fueron inicializados de forma tal que $u_i^{(0)} = 1$ si $i = 40 \dots 60$ o $u_i^{(0)} = 0$ si no, lo que representa a un segmento lineal con concentración máxima alrededor de su punto medio y concentración nula hacia sus extremos. Los resultados se resumen en la figura 5, en la que se puede observar que, como era de esperarse, a mayor α mayor es la tasa de difusión.

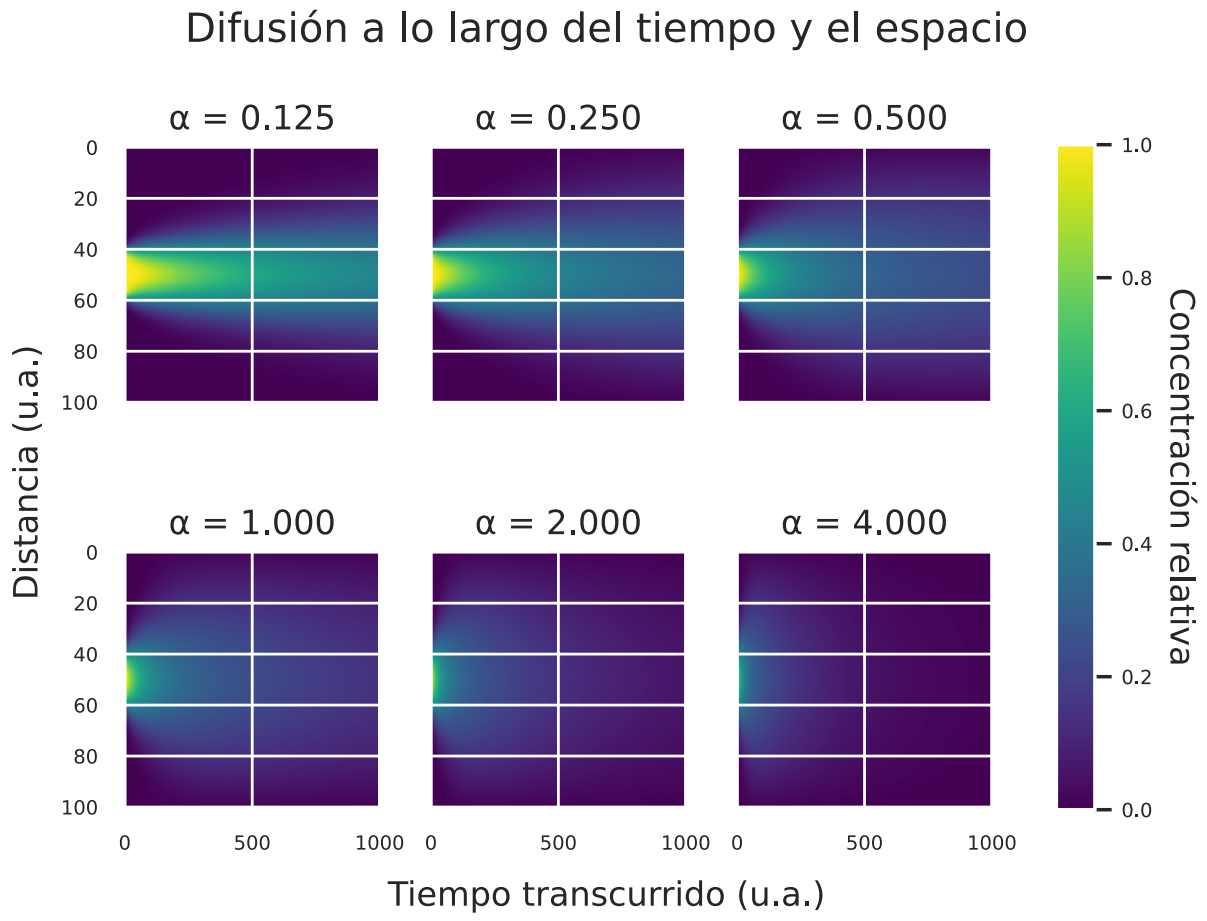


Figura 5: Simulación del proceso de difusión a lo largo del tiempo (eje x) y el espacio (eje y), por la aplicación iterativa de la forma matricial de la ecuación de difusión discreta en una dimensión.

6. Conclusiones

Se diseñaron e implementaron algoritmos de EG para resolución de sistemas de ecuaciones lineales en Python, sin el uso de bibliotecas de álgebra lineal como Numpy o afines. Entre ellos, se diseñó un algoritmo de EG con pivoteo parcial (intercambio entre filas) que disminuye el error numérico, permite encontrar la solución de más sistemas, y no incrementa considerablemente la complejidad algoritmica respecto al algoritmo sin pivoteo, siendo ambos $O(n^3)$.

Aprovechando la estructura rala particular de las matrices tridiagonales, fue posible diseñar un algoritmo de EG de complejidad $O(n)$ que convierte a un sistema tridiagonal en uno equivalente diagonal a partir del cual es trivial encontrar la solución. Resolviendo sistemas tridiagonales dados por la matriz de la aproximación discreta al operador Laplaciano y varios vectores de términos independientes, se logró efectivamente encontrar aproximaciones discretas a funciones a partir de sus derivadas segundas.

Fue posible mejorar el algoritmo de EG para matrices tridiagonales de forma tal de permitir el precómputo de su sistema diagonal equivalente, permitiendo una disminución en tiempo de cómputo considerable para casos en los que es necesario reutilizar a una misma matriz de coeficientes para operar sobre una serie de vectores de términos independientes. Este último fue por ejemplo el caso de la simulación del proceso físico de difusión. Partiendo de la matriz tridiagonal asociada a la aproximación discreta a la ecuación de difusión, se precomputó su sistema diagonal equivalente y se lo resolvió iterativamente a partir de una condición inicial, efectivamente simulando la evolución temporal de la difusión a lo largo de un segmento lineal del espacio.

Para la cuantificación de los tiempos de cómputo de los distintos algoritmos se eligió una métrica robusta, el tiempo mínimo de los promedios de múltiples grupos de ejecuciones, mucho menos sensible al ruido propio del entorno en el que se ejecuta el código que otras como son el tiempo promedio. La medición de los tiempos nos permitió comprobar nuestros saberes previos acerca de cuales algoritmos esperabamos sean más rápidos, y con qué complejidades algorítmicas.

Referencias

[1] Richard L. Burden, Douglas J. Faires & Annette M. Burden, *Numerical Analysis*, 10th ed. 2016. **link**.

[2] Métodos Numéricos (UBA, FCEyN, DC), *Sistemas de ecuaciones lineales y Eliminación Gaussiana*. Segundo cuatrimestre 2023. **link**.

[3] Documentación oficial de `%timeit`. **link**.

[4] David A. Patterson & John L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. 2005. **link**.

Apéndice

Todo el código necesario para computar los resultados mostrados y discutidos en este informe puede encontrarse en y ser ejecutado desde la siguiente Jupyter notebook: **link**.

A continuación se comparten los pseudocódigos de algunos de los algoritmos auxiliares que fueron parte del diseño de los algoritmos principales ya presentados.

Algorithm 10 Producto escalar

Input: $fila$, $columna$ vectores de dimensiones $1\texttt{x}n$ y $n\texttt{x}1$ respectivamente
Output: res escalar tal que $fila \cdot columna = res$

```
1: function producto_escalar_vector_fila_vector_columna(fila, columna)
2:     res ← 0
3:     for i = 0 . . . |fila| do
4:         res ← res + fila[i] ∗ columna[i]
5:     return res
```

Algorithm 11 Suma de matrices

Input: A, B matrices de la misma dimensión
Output: res matriz tal que $A + B = res$

```
1: function suma_matricial(A, B)
2:     res ← A                                   ▷ copia profunda
3:     for i = 0 . . . |A| do
4:         for j = 0 . . . |A[0]| do
5:             res[i][j] ← A[i][j] + B[i][j]
6:     return res
```

Algorithm 12 Producto por escalar

Input: A matriz
Input: k escalar
Output: res matriz tal que $kA = res$

```
1: function producto_por_escalar(A, k)
2:     res ← A                                   ▷ copia profunda
3:     for i = 0 . . . |A| do
4:         for j = 0 . . . |A[0]| do
5:             res[i][j] ← k ∗ A[i][j]
6:     return res
```