DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 1 - Eliminación Gaussiana Tridiagonal

Algoritmos de Eliminación Gaussiana para matrices tridiagonales y difusión.

Métodos Numéricos
Primer Cuatrimestre de 2024

Integrante	LU	Correo electrónico
Emilio Giménez	765/17	emigimenezj@zohomail.com
Nicolás Carratalá	736/17	nacarratala@gmail.com
Federico Alamino	316/17	federicoalamino2@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Contenido

1. Introducción

En este informe científico presentaremos un estudio exhaustivo de los algoritmos de eliminación gaussiana aplicados a matrices de diferentes tipos, con un enfoque especial en matrices tridiagonales. Utilizaremos una implementación óptima de este algoritmo para resolver un problema de difusión de manera eficiente y, a continuación, experimentaremos con diversas configuraciones para demostrar los resultados que respaldan la mejora en el rendimiento de la eliminación gaussiana para matrices tridiagonales. Estos resultados experimentales nos permitirán determinar los beneficios específicos de esta optimización.

A continuación, presentamos los conceptos fundamentales que serán necesarios para la comprensión de los resultados y la metodología de nuestro estudio sobre dichos algoritmos. Estos conceptos incluyen la definición y propiedades de las matrices tridiagonales, la formulación del problema de difusión y las técnicas de optimización de algoritmos empleadas.

1.1. Sistemas de ecuaciones lineales

Un sistema de ecuaciones lineales se puede representar de manera compacta como una ecuación matricial de la forma:

$$Ax = b \tag{1}$$

Donde $A \in \mathbb{R}^{n \times n}$ una matriz cuadrada y $x, b \in \mathbb{R}^n$ son ambos vectores columna en donde x representa las incógnitas del sistema y b los términos independientes.

El objetivo es encontrar el vector x que resuelve la ecuación, es decir, que satisface que al ser multiplicado a izquierda por la matriz A se obtiene el vector b. Para lograr esto, se utiliza el algoritmo de eliminación gaussiana con una serie de reglas que veremos a continuación.

1.2. Resolución de sistemas

Resolver un sistema de ecuaciones lineales es un problema ampliamente conocido. La eliminación gaussiana es un algoritmo comúnmente utilizado para dicho propósito. La idea es transformar el sistema original en uno equivalente que sea más fácil de resolver. Este proceso implica la manipulación de las ecuaciones originales mediante operaciones elementales en las filas de la matriz, incluyendo la multiplicación por un escalar y la adición o sustracción de otra fila que también puede ser multiplicada por un escalar. El objetivo es reducir la matriz a una forma triangular superior, donde los elementos por debajo de la diagonal principal son cero.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots \\ 0 & & & a_{nn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2}$$

Además, durante el proceso de triangulación de la eliminación gaussiana, las operaciones deben aplicarse a ambos lados de la igualdad, por lo que también hay que operar sobre los elementos del vector de términos independientes b. Este paso es crucial, dado que de otra forma se estaría cambiando el sistema de ecuaciones original comprometiendo la validez de los resultados obtenidos.

Una vez se obtiene el sistema triangulado, la técnica de sustitución hacia atrás permite determinar los valores de las incógnitas x_i del sistema original. El procedimiento consiste en empezar a resolver desde la última ecuación ($a_{nn}x_n = b_n$) y a medida que se van despejando las variables $x_n, x_{n-1}, \ldots$ se utilizan sus valores nuevos en la ecuación inmediatamente superior hasta determinar los valores de $x_i \forall i$. De esta forma, se obtiene el vector solución completo $x = (x_1, x_2, \ldots, x_n)^t$.

1.3. Matrices tridiagonales

Una matriz tridiagonal es un tipo especial de matriz en la que la mayoría de elementos son cero, excepto los que se encuentran en la diagonal principal y sus diagonales adyacentes (superior e inferior).

$$\begin{bmatrix} b_1 & c_1 & & 0 \\ a_1 & b_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ 0 & & a_{n-1} & b_n \end{bmatrix} \tag{3}$$

La estructura de una matriz tridiagonal es muy diferente a la de una matriz común, ya que sólo se almacenan una pequeña fracción de los elementos en la memoria, lo que hace que los algoritmos diseñados específicamente para este tipo de matrices sean mucho más eficientes en términos de memoria y tiempo de cómputo como se verá más adelante en este trabajo.

El sistema de ecuaciones asociado a una matriz tridiagonal es de la siguiente forma:

$$\begin{bmatrix} b_1 & c_1 & & 0 \\ a_1 & b_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ 0 & & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \tag{4}$$

Donde cada ecuación del sistema se define como

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \tag{5}$$

1.4. Laplaciano discreto ∇^2

El operador Laplaciano discreto se define como la suma de las segundas derivadas parciales de una función en una posición específica. La versión discreta del operador se representa típicamente mediante una matriz tridiagonal donde su diagonal principal es negativa con valores positivos en sus diagonales adyacentes y es ampliamente utilizada en procesamiento de imágenes y grafos para cosas como detección de bordes, eliminación de ruido, segmentación de imágenes, entre otras cosas.

En el caso unidimensional, el operador es equivalente a la derivada segunda $\frac{d^2}{dx^2}$ y la operación discreta de $\frac{d^2}{dx^2}u = d$ está dada por la siguiente ecuación:

$$u_{i-1} - 2u_i + u_{i+1} = d_i \tag{6}$$

Nótese la similitud con la ecuación (5) y que la derivada de un valor constante es 0 ya que los coeficientes $(1, -2, 1)$ suman 0. En forma matricial, se obtiene:

$$\begin{bmatrix} -2 & 1 & & 0 \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \tag{7}$$

De esta forma, usando la eliminación gaussiana se puede encontrar el vector u para cada vector d. Es interesante remarcar que cuando $n \to \infty$, la solución del sistema es equivalente a encontrar una función u cuya segunda derivada es la función d (para funciones continuas). Esto último sera un objeto de estudio en este trabajo más adelante.

1.5. Difusión

La difusión es un fenómeno estocástico en el que una entidad se propaga desde una región con mayor concentración hacia otra con menor concentración. Este proceso se utiliza ampliamente en el modelado de situaciones estocásticas en áreas como física, estadística, probabilidad, finanzas, redes neuronales, entre otras.

A modo de ejemplo, se puede considerar la evolución temporal de una variable $x^{(k)}$ que obedece la fórmula $x^{(k)} = x^{(k-1)} + \epsilon$, donde ϵ es una variable aleatoria que toma valores -1 o 1. El gráfico muestra varias simulaciones de este proceso.

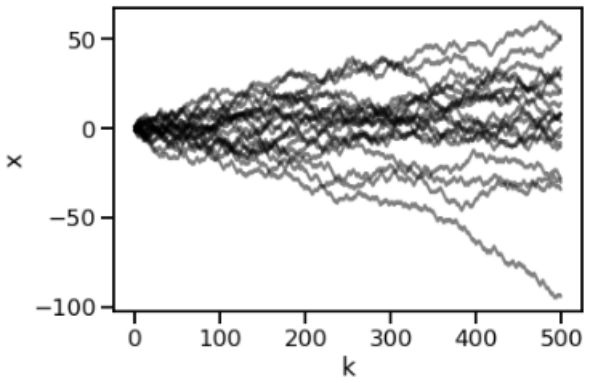


Figura 1: un gráfico que muestra la expansión de un efecto de difusión

2. Desarrollo

En esta sección presentamos las implementaciones y aplicaciones desarrolladas en el marco del trabajo. Se incluyen las implementaciones de eliminación gaussiana con y sin pivoteo, así como una versión adaptada para matrices tridiagonales. También se presenta el cálculo de derivadas segundas mediante el operador Laplaciano y el desarrollo de una simulación de difusión.

2.1. Eliminación gaussiana sin pivoteo

La eliminación gaussiana es un método fundamental para la resolución de sistemas de ecuaciones lineales. En su versión sin pivoteo, el algoritmo procede a la triangulación de la matriz de coeficientes sin intercambiar filas, lo cual simplifica en gran medida la implementación. Sin embargo, es importante tener en cuenta que, para que este algoritmo funcione correctamente, se requiere que la diagonal principal de la matriz nunca contenga un valor nulo.

En nuestro enfoque inicial, decidimos suponer que esta condición se cumple siempre, lo que nos permitió simplificar la implementación y enfocarnos en los pasos esenciales de la triangulación. Estos pasos incluyen el cálculo del inverso y la aplicación de la operación adecuada a toda la fila y el término independiente correspondiente para obtener los ceros debajo del pivote. De esta manera, pudimos avanzar en la implementación del algoritmo sin preocuparnos por este caso particular, aunque es importante tener en cuenta que la eliminación gaussiana sin pivoteo puede fallar si la matriz de coeficientes tiene una diagonal principal con valores nulos.

Algorithm 1 : primer acercamiento a la eliminación gaussiana sin pivoteo

```
1: procedure EGSinPivoteo(A, b)
2:     n ← len(A)
3:     for i in 0 ... n − 1 do
4:         for j in i + 1 ... n − 1 do
5:             inverso ← A[j][i] / A[i][i]
6:             for k in i + 1 ... n − 1 do
7:                 A[j][k] ← A[j][k] − inverso · A[i][k]
8:             A[j][i] ← 0
9:             b[j] ← b[j] − inverso · b[i]
```

Ahora, para garantizar robustez en nuestro algoritmo, es necesario considerar el caso en el que la diagonal tenga valores nulos. Por lo tanto, añadimos una verificación adicional para evitar que el algoritmo se rompa para ese caso particular como mostramos a continuación.

Algorithm 2 : eliminación gaussiana sin pivoteo manejando valores nulos en la diagonal

```
1: procedure EGSinPivoteo(A, b)
2:     n ← len(A)
3:     for i in 0 ... n − 1 do
4:         for j in i + 1 ... n − 1 do
5:             if A[i][i] = 0 then                ▷ Paso de verificación de nulidad.
6:                 return None
7:             inverso ← A[j][i] / A[i][i]
8:             for k in i + 1 ... n − 1 do
9:                 A[j][k] ← A[j][k] − inverso · A[i][k]
10:            A[j][i] ← 0
11:            b[j] ← b[j] − inverso · b[i]
```

Con esto, ya somos capaces de llegar a un sistema triangulado que sea equivalente al original de manera algorítmica, lo que nos permite plantear una resolución del sistema triangulado para poder encontrar una solución del sistema original. Cabe destacar que el algoritmo presentado no tiene un retorno explícito para el caso en el que se logra la triangulación porque la matriz A y el vector solución b que se reciben por parámetro se operan por referencia, lo que implica que se modifican los valores originales que le llegan a nuestra función en caso de que todo funcione correctamente.

Para poder ver en acción la verificación de valor nulo en las diagonales, solo basta usar una matriz cualquiera que tenga un cero en su primera coordenada. Es decir, buscaríamos una matriz A cualquiera tal que $a_{11} = 0$.

2.2. Eliminación gaussiana con pivoteo

A diferencia de la eliminación gaussiana sin pivoteo, este algoritmo utiliza el intercambio de filas para asegurarse de que en la diagonal principal siempre se encuentra el valor absoluto más grande de la columna correspondiente, lo que garantiza una mejor estabilidad numérica. En este caso, si la diagonal posee un valor nulo, el primer paso a realizar consiste en encontrar el máximo valor absoluto de la columna correspondiente a la posición actual del pivote, que se utiliza como nuevo pivote y se sitúa en la diagonal principal de la matriz al realizar el intercamio de filas correspondiente.

A continuación, se hacen las operaciones necesarias para que los valores debajo del pivote sean iguales a cero, lo que implica que la matriz resultante sea triangular superior. La forma en que se aplican estas operaciones es similar a la eliminación gaussiana sin pivoteo, con la salvedad de que se utiliza la fila correspondiente al nuevo pivote como pivote en los cálculos.

En el siguiente pseudocódigo, podemos ver cómo implementar la eliminación gaussiana con pivoteo. Se indica oportunamente con comentarios en dónde se utiliza el intercambio de filas para evitar el problema del pivote nulo.

Algorithm 3 : eliminación gaussiana evitando ceros en la diagonal con intercambio de filas

```
1:  procedure EGConPivoteo(A, b)
2:      n ← len(A)
3:      for j in 0 ... n − 1 do
4:          if A[i][i] = 0 then                          ▷ Verificación de valor nulo en la diagonal
5:              max ← i
6:              for j in i + 1 ... n − 1 do              ▷ Búsqueda del máximo en la columna del pivote
7:                  if abs(A[j][i]) > abs(A[max][i]) then
8:                      max ← j
9:              if max ≠ i then          ▷ Intercambio de filas si se encuentra algún número distinto de cero
10:                 intercambiar A[max] ↔ A[i]
11:                 intercambiar b[max] ↔ b[i]
12:         for j in i + 1 ... n − 1 do                  ▷ A partir de este punto es lo mismo que sin pivoteo
13:             inverso ← A[j][i] / A[i][i]
14:             checkNumericError(inverso, ϵ)   ▷ Verificación de error numérico por aproximación al cero.
15:             for k in i + 1 ... n − 1 do
16:                 A[j][k] ← A[j][k] − inverso · A[i][k]
17:             A[j][i] ← 0
18:             b[j] ← b[j] − inverso · b[i]
```

Al igual que en la eliminación gaussiana sin pivoteo, es posible que la eliminación gaussiana con pivoteo falle si se encuentra una columna debajo del pivote con todos sus valores nulos. Para evitar este problema, se pueden implementar otro tipo de estrategias de pivoteo, como el pivoteo completo que también ofrece intercambios de columnas. Dichas estrategias quedan por fuera del alcance de este trabajo. Un ejemplo que puede exponer este problema podría ser la siguiente matriz.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{Fila 3} \leftrightarrow \text{Fila 1}} \begin{bmatrix} 1 & 1 & 1 \\ 0 & \boxed{0} & 1 \\ 0 & \boxed{0} & 1 \end{bmatrix} \tag{8}$$

Como puede verse, no se puede realizar un segundo paso de eliminación ya que no solo el pivote a utilizar es nulo, sino que también el valor de su columna en la única fila que tiene por debajo también. Por lo que no existe máximo que tomar para poder hacer un intercambio de filas y salvar la situación tal y como se realizó en el primer paso de eliminación.

Por otra parte, en caso de que las operaciones realizadas en la eliminación lleven a resultados numéricos demasiado pequeños se podría tener un problema de representación. Esto se debe a que los lenguajes tienen un límite mínimo de representación numérica de punto flotante ya que no es posible tener infinitos valores reales porque no se posee infinita memoria. En nuestro algoritmo de eliminación gaussiana, cada pivote se usa para calcular un factor a través de la multiplicación de la coordenada actual sobre el pivote. Usar el pivote como denominador es un buen detalle a tener en cuenta para generar una matriz que produzca error numérico, ya que para obtener resultados numéricos intermedios muy pequeños, necesitamos dividir por algo muy grande, por lo tanto necesitamos un pivote grande.

$$\begin{bmatrix} 100000000 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{9}$$

Nótese, que en este ejemplo particular, el algoritmo deberá obtener el factor, el cual se obtiene al realizar 1 / 100000000. Teóricamente hablando, el pivote debería valer $1E-8$. Sin embargo, sabemos por lo desarrollado anteriormente que no todo numero flotante tiene representación dentro de la arquitectura de las computadores. Apoyándonos en las herramientas provistas por la cátedra, se puede observar que el resultado programático es $9{,}999999392252902907785028219223022460937 5E-9$, lo cual significa que se produce un error numérico valuado en $-6{,}077470970922149717807769775E-17$.

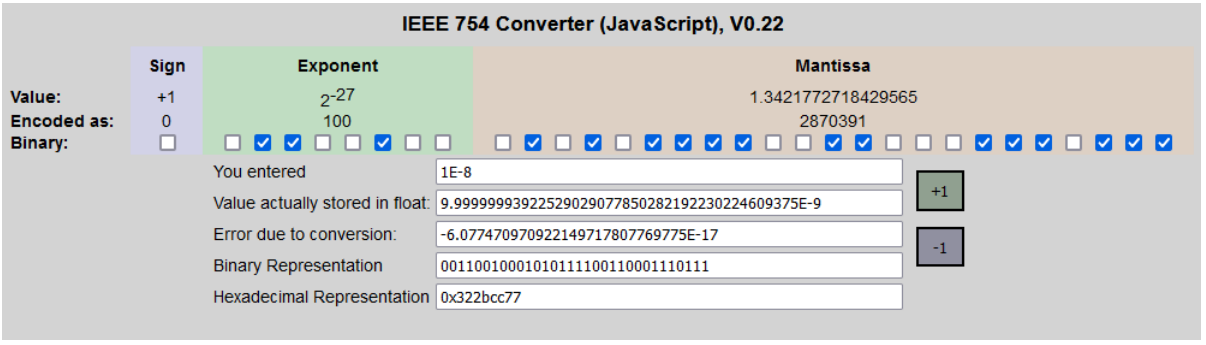


Figura 2: Cálculo del error numérico según el estándar IEEE 754

Cabe destacar que el pivote calculado es utilizado posteriormente para modificar alguna fila, por lo tanto este error numérico es arrastrado y por consecuencia, agravado.

2.3. Eliminación gaussiana para un sistema tridiagonal

En la eliminación gaussiana para un sistema tridiagonal se aprovecha la estructura de la matriz tridiagonal para realizar un algoritmo más eficiente. De echo, esta es la ventaja principal de trabajar con una matriz tridiagonal, que la mayoría de los elementos son nulos, lo que permite reducir considerablemente el tiempo de ejecución al realizar las operaciones necesarias. En este caso, la eliminación gaussiana se puede simplificar aún más y definiendo únicamente tres vectores: el vector A con los elementos de la diagonal inferior, el vector B con los elementos de la diagonal principal, y el vector C con los elementos de la diagonal superior.

El siguiente pseudocódigo muestra cómo se puede implementar la eliminación gaussiana para una matriz tridiagonal de manera óptima usando el vector D como vector solución:

Algorithm 4 : eliminación gaussiana óptima para una matriz tridiagonal

```
1: procedure EGTRIDIAGONAL(A, B, C, D)
2:     n ← len(B)
3:     for i in 0 ... n − 1 do
4:         if B[i] = 0 then
5:             return None
6:         inverso ← A[i] / B[i]
7:         A[i] ← 0
8:         B[i + 1] ← B[i + 1] − inverso · C[i]
9:         D[i + 1] ← D[i + 1] − inverso · D[i]
```

La eliminación gaussiana se resuelve en un único ciclo en vez de tres como la eliminación gaussiana estándar dado que la cantidad de elementos a iterar en la columna del pivote y la cantidad de elementos a operar en cada fila usando el inverso son cantidades fijas.

Es importante notar que después de concluir esta triangulación, el vector A queda nulo, el vector B es el que tiene las operaciones de fila realizadas y el vector C no sufre modificaciones. Esto va a ser importante más adelante.

2.4. Resolución de sistema de ecuaciones

Hasta este punto solo se estuvieron explorando algoritmos que permiten triangular matrices por medio de eliminación gaussiana para obtener un sistema de ecuaciones equivalente al original pero que es más fácil de resolver, pero todavía no mostramos cómo resolverlo.

Para eso está la técnica de *Backward Substitution*, que es una forma eficiente de resolver sistemas de ecuaciones lineales, especialmente cuando la matriz es triangular superior. En esta técnica, se resuelve el sistema comenzando por la última ecuación y se va retrocediendo hacia la primera.

Algorithm 5 : sustitución hacia atrás para cualquier sistema expresado como matriz triangular superior

```
1: procedure BACKWARDSUBSTITUTION(A, b)
2:     n ← len(A)
3:     x ← [0 ... 0] ∈ ℝ^n
4:     for i in n − 1 ... 0 do
5:         suma ← 0
6:         for j in i + 1 ... n do
7:             suma ← suma + A[i][j] · x[j]
8:         x[i] ← (b[i] − suma) / A[i][i]
9:     return x
```

En este caso particular, se genera un nuevo vector solución x dentro del algoritmo que es la respuesta a la ecuación matricial inicial tal y como se describió en la ecuación (1).

Sin embargo, cuando se trabaja en particular con matrices tridiagonales, es posible aplicar una optimización al algoritmo de *Backward Substitution* para reducir aún más su complejidad. Al igual que con las optimizaciones realizadas con los algoritmos de eliminación gaussiana, esta optimización se basa en aprovechar la estructura de la matriz tridiagonal para simplificar las operaciones de sustitución hacia atrás, reduciendo así el número de operaciones necesarias para resolver el sistema.

Algorithm 6 : sustitución hacia atrás para sistemas tridiagonales

```
1:  procedure BACKWARDSUBSTITUTIONTRIDIAGONAL(B, C, D)
2:      n ← len(B)
3:      x ← [0 ... 0] ∈ ℝ^n
4:
5:      x[n − 1] ← D[n−1] / B[n−1]
6:
7:      for i in n − 2 ... 0 do
8:          x[i] ← (D[i] − C[i] · x[i+1]) / B[i]
9:
10:     return x
```

Este nuevo algoritmo toma un vector B como la diagonal principal de la matriz triangulada, un vector C como la diagonal adyacente superior y un vector D como el vector de términos independientes. Nuevamente, se genera un vector solución x dentro del algoritmo que es la respuesta a la ecuación matricial tridiagonal inicial.

Esencialmente, este nuevo algoritmo es un caso particular del algoritmo anterior adaptado al caso concreto de las matrices tridiagonales para que sea más eficiente, pero en teoría ambos algoritmos resuelven un sistema tridiagonal correctamente.

2.5. Optimización de triangulación de vectores solución

La factorización LU es un método para la resolución de sistemas de ecuaciones lineales que se basa en la descomposición de una matriz cuadrada A en dos matrices triangulares: una matriz triangular inferior L y una matriz triangular superior U, tales que $A = LU$. La matriz L tiene 1 en la diagonal principal, mientras que la matriz U tiene elementos no nulos en su diagonal principal.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix} \tag{10}$$

Esta técnica es muy útil ya que una de sus ventajas es que una vez que tenemos la matriz L, podemos utilizarla para triangular un vector solución D sin necesidad de volver a triangular toda la matriz nuevamente. En la matriz L están almacenados los *inversos* que se utilizaron en las operaciones de triangulación de la matriz original, y gracias a eso podemos utilizarla para triangular otros vectores. La manera de hacerlo sería simplemente realizando la multiplicación matricial $L * D$. El resultado de esta operación devuelve un nuevo vector D', que está triangulado gracias a los *inversos* que ya estaban presentes en la matriz L.

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1^{(n)} \\ d_2^{(n)} \\ \vdots \\ d_n^{(n)} \end{bmatrix} \tag{11}$$

Atención: usamos la notación $d^{(n)}$ para indicar el valor de d después de n pasos de triangulación.

Este proceso de triangular un vector solución utilizando la matriz L es la parte fundamental de la optimización, ya que se pueden ahorrar muchos pasos y hace que el proceso de resolución del sistema sea mucho más eficiente cuando se tienen muchos vectores solución para una misma matriz. En lugar de tener que triangular toda la matriz nuevamente para cada nuevo vector solución, simplemente se utiliza la matriz L que ya tenemos disponible.

Sin embargo, esa no es la única optimización. En los procesos de derivada segunda y difusión que se trabajarán a continuación, en todo momento se usan matrices triangulares. Esto es importante, ya que afecta a la estructura de la matriz L. Como se mencionó anteriormente al final de la sección 2.3, en una triangulación para matrices tridiagonales, el vector A queda nulo y esto sucede porque los únicos inversos envueltos en las operaciones de triangulación son los que vuelven nulo al vector A. O dicho de otra forma, cuando se triangula una matriz tridiagonal, solo se obtiene un único vector de inversos. Por lo tanto, la matriz L de la factorización LU tendrá 1 en su diagonal y el único vector de inversos que fueron necesarios para triangular en la diagonal adyacente inferior.

$$L = \begin{bmatrix} 1 & & & & 0 \\ \boxed{a_1} & 1 & & & \\ & \boxed{a_2} & 1 & & \\ & & \ddots & \ddots & \\ 0 & & & \boxed{a_{n-1}} & 1 \end{bmatrix} \tag{12}$$

Teniendo en cuenta esta observación, usando como notación que A es el vector de inversos que está dentro de la matriz L de la factorización LU y b el nuevo vector de términos independientes que se quiere triangular, se puede resolver la multiplicación matricial de $Lb = x$ con la siguiente fórmula:

$$x_i = \begin{cases} b_i & \text{si } i = 1 \\ x_{i-1} \cdot A_{i-1} + b_i & \text{si } i \neq 0 \end{cases} \tag{13}$$

Finalmente, nuestra implementación con todas las optimizaciones mencionadas anteriormente queda definida por el siguiente pseudocódigo.

Algorithm 7 : optimización del proceso de triangulación de nuevos vectores solución

```
1: procedure PRODUCTOMATRICIALOPTIMIZADO(A, b)
2:     n ← len(b)
3:     x ← [0 . . . 0] ∈ ℝ^n
4:     x[0] ← b[0]
5:     for i in 1 . . . n − 1 do
6:         x[i] ← x[i − 1] · A[i − 1] + b[i]
7:     return x
```

Con esto conseguimos una forma altamente optimizada de resolver el sistema original para varios vectores de términos independientes diferentes sin necesidad de volver a triangular la matriz original asociada al sistema. Más adelante en este trabajo, en la sección de experimentación, pondremos a prueba los beneficios de las optimizaciones planteadas en esta sección.

2.6. Derivada segunda

El objetivo de esta sección es poner a prueba varios de los conceptos ya trabajados hasta ahora. Para eso, se plantea encontrar u para el problema $\frac{d^2}{dx^2}u = d$ utilizando la matriz del operador laplaciano expuesta en la ecuación (7) para los siguientes vectores solución:

$$\text{a)}\quad d_i = \begin{cases} 0 \\ 4/n \quad i = \lfloor \frac{n}{2} \rfloor + 1 \end{cases}$$

$$\text{b)}\quad d_i = \frac{4}{n^2}$$

$$\text{c)}\quad d_i = \frac{-1 + \frac{2i}{n-1}}{n^2} \cdot 12$$

El algoritmo desarrollado para esta resolución hace uso de técnicas y algoritmos previamente explicados en este trabajo. En el pseudocódigo del algoritmo, se incluirán referencias a los algoritmos utilizados para mejorar su comprensión. Como notación, se utilizan los vectores A, B y C para hacer referencia a las diagonales del operador laplaciano y d_a, d_b o d_c para el vector solución del punto a), b) o c) respectivamente.

Algorithm 8 : resolución de derivada segunda de forma optimizada

procedure RESOLVERDERIVADASSEGUNDAS
  $n \leftarrow 101$
  A $\leftarrow [1 \dots 1] \in \mathbb{R}^n$
  B $\leftarrow [(-2) \dots (-2)] \in \mathbb{R}^n$
  C $\leftarrow [1 \dots 1] \in \mathbb{R}^n$
  A $\leftarrow$ EXTRAERVECTORDEINVERSOS(A, B, C)
  $d_a^{(n)} \leftarrow$ TRIANGULAR(A, d_a) ▷ Usando el **Algoritmo 7**
  $x_a \leftarrow$ BACKWARDSUSTITUTION(B, C, $d_a^{(n)}$) ▷ Usando el **Algoritmo 6**
  $d_b^{(n)} \leftarrow$ TRIANGULAR(A, d_b) ▷ Usando el **Algoritmo 7**
  $x_b \leftarrow$ BACKWARDSUSTITUTION(B, C, $d_b^{(n)}$) ▷ Usando el **Algoritmo 6**
  $d_c^{(n)} \leftarrow$ TRIANGULAR(A, d_c) ▷ Usando el **Algoritmo 7**
  $x_c \leftarrow$ BACKWARDSUSTITUTION(B, C, $d_c^{(n)}$) ▷ Usando el **Algoritmo 6**
  return x_a, x_b, x_c

Al obtener los valores de x_a, x_b, y x_c, se pudo realizar el siguiente gráfico que muestra de manera continua la forma de las derivadas segundas de cada una de las funciones que representan los vectores soluciones proporcionados en el problema.

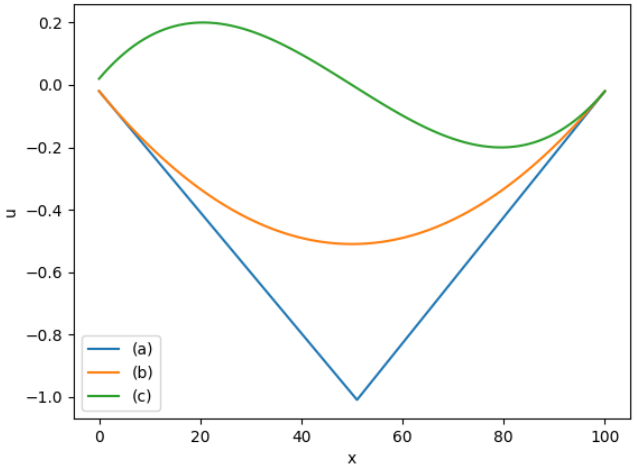


Figura 3: gráfico superpuesto de derivadas segundas de los vectores solución

2.7. Difusión

En esta sección se aborda el proceso de difusión mencionado en la sección 1.5 que está en la introducción de este trabajo con la idea de continuar aplicando conceptos ya explorados. Como se puede ver en la Figura 1, para varias simulaciones de este proceso se puede notar cómo la variable x se difunde a lo largo del tiempo. En cada simulación, se utilizó una variable aleatoria diferente para generar la secuencia $x^{(0)}, x^{(1)}, \ldots, x^{(T)}$, donde T es el número de pasos de tiempo. Se puede ver que, a medida que aumenta T, la distribución de $x^{(T)}$ se va aproximando a una distribución gaussiana. Esto último es importante porque se verá reflejado más adelante con un gráfico.

El objetivo a completar esta vez es resolver un sistema de forma iterativa para una condición inicial dada por el siguiente vector u:

$$(u_{(0)})_i = \begin{cases} 0 & \\ 1 & \lfloor \frac{n}{2} \rfloor - r < i < \lfloor \frac{n}{2} \rfloor + r \end{cases} \tag{14}$$

Tomando como parámetros fijos los valores de $n = 101$, $r = 10$ y $m = 1000$.

Para este caso particular donde se debe resolver un problema de difusión unidimensional iterativamente, se pueden utilizar varias herramientas ya exploradas en este trabajo como la triangulación y sustitución para sistemas tridiagonales, como también la aplicación de triangulación a nuevos vectores solución sin recalcular la triangulación del sistema original en cada paso de simulación.

Para hacer un uso correcto de las optimizaciones mencionadas, la implementación desarrollada por nuestra parte se ajusta al pseudocódigo presentado a continuación.

Algorithm 9 : proceso de difusión con resolución de sistemas tridiagonales optimizada

```
1:  procedure ResolverDifusion(n, r, m)
2:      A ← [(−1) . . . (−1)] ∈ ℝ^n
3:      B ← [3 . . . 3] ∈ ℝ^n
4:      C ← [(−1) . . . (−1)] ∈ ℝ^n
5:      A ← ExtraerVectorDeInversos(A, B, C)
6:      vectoresSolucion[0] ← u_(0)
7:      for i in 1 . . . m − 1 do
8:          u^(n)_(i−1) ← Triangular(A, u_(i−1))          ▷ Usando el Algoritmo 7
9:          u_i ← BackwardSustitution(B, C, u^(n)_(i−1))  ▷ Usando el Algoritmo 6
10:         vectoresSolucion[i] ← u_i
11:     return vectoresSolucion
```

Es importante destacar que, al simular el proceso de difusión a través del tiempo, se puede observar que la distribución de los valores del vector u se aproximan a una distribución gaussiana luego de cada paso de simulación. Esto se debe a que, con el tiempo, los valores del vector se dispersan y se distribuyen de manera uniforme a lo largo del espacio como se muestra en el siguiente gráfico.

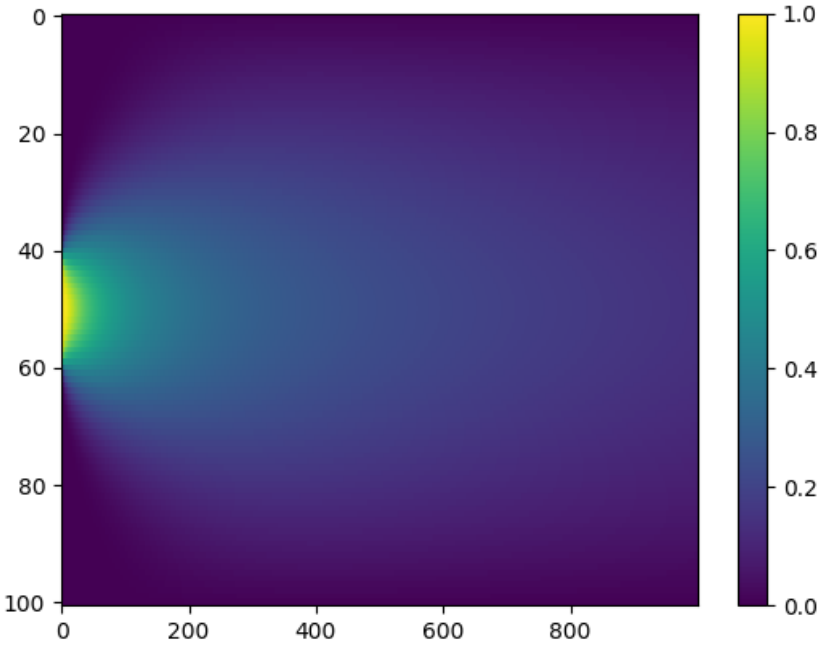


Figura 4: gráfico del proceso de simulación de difusión

En definitiva, el objetivo de esta sección es proporcionar herramientas y algoritmos que permitan ejecutar una simulación eficiente y precisa de los procesos de difusión. La resolución de sistemas tridiagonales optimizada es una técnica altamente efectiva para lograr este objetivo, ya que permite la simulación de procesos complejos con una gran cantidad de datos en un tiempo razonable. Al implementar esta técnica en la simulación de la difusión, se pueden obtener resultados precisos y confiables que permiten una mejor comprensión de los fenómenos estudiados tal y como se muestra en la Figura 4.

3. Experimentación

En esta sección, presentamos los resultados de la experimentación realizada para analizar la velocidad de ejecución de los algoritmos implementados en secciones anteriores. El objetivo de esta experimentación es proporcionar una comparación cuantitativa de los diferentes enfoques utilizados y demostrar la eficacia de los algoritmos enfocados en matrices tridiagonales por sobre los algoritmos enfocados en matrices estándar.

Para llevar a cabo esta experimentación, se generaron diferentes muestras de datos temporales de la velocidad de ejecución de los algoritmos. Estos datos se obtuvieron mediante la ejecución de cada algoritmo un número determinado de veces (aproximadamente 2000 repeticiones) con matrices de una determinada dimensión y se midió el tiempo de ejecución en cada una de ellas. Decidimos quedarnos con los tiempos mínimos calculados asumiendo que es la ejecución con menos interferencia de rendimientos de procesos externos ajenos a nuestro análisis. Todo el resto de detalles experimentales se encuentran en la correspondiente sección de cada experimento. Luego, se promediaron los tiempos de ejecución para obtener un resultado más estable, uniforme y confiable.

En síntesis, el propósito principal de estos experimentos es ofrecer resultados que proporcionen una visión más detallada del rendimiento de los algoritmos en términos de tiempo de ejecución.

3.1. Coste computacional de eliminación gaussiana sin pivoteo

Esta experimentación se enfoca en medir los tiempos de ejecución de la eliminación gaussiana con pivoteo para matrices de diferente dimensión, y el objetivo principal es analizar cómo varía el tiempo en función de la dimensión de la matriz. Cabe destacar que para simplificar el experimento, los elementos de la matriz y del vector solución están definidos únicamente para enteros del 1 al 50. También es importante mencionar que se midió únicamente el tiempo de procesamiento de la triangulación y la sustitución para resolver el sistema, dejando fuera del análisis otros posibles factores que puedan influir en el tiempo total de ejecución como lo pueden ser la generación de la matriz y el vector resultante o la escritura de los datos en archivos de texto.

Como es de esperar, el comportamiento esperado en este experimento es que a medida que se vaya aumentado la dimensión de la matriz y su respectivo vector solución se necesite mayor tiempo de ejecución para llegar al vector resultado debido a que, simplemente, a mayor dimensión, mayor es la cantidad de operaciones necesarias para triangular y resolver el sistema.

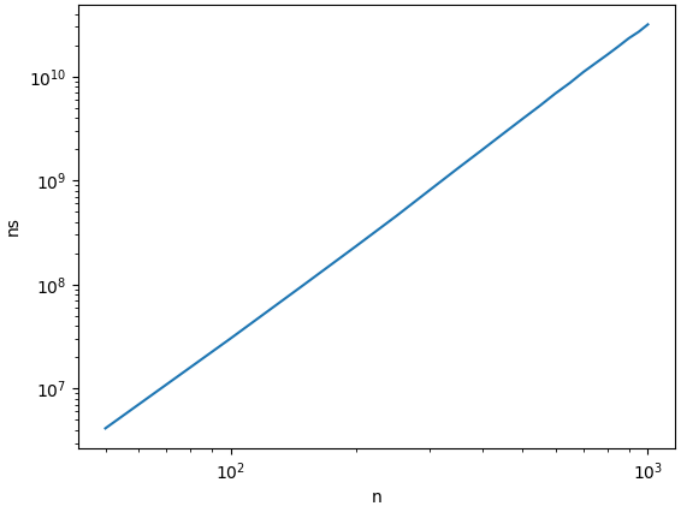


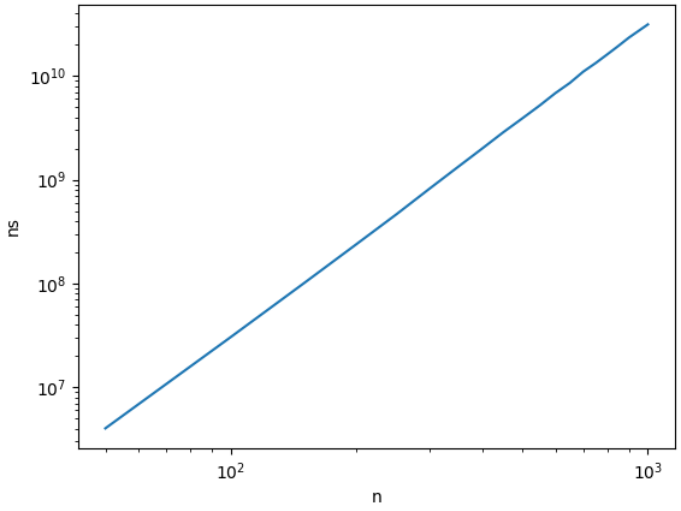
Figura 5: Tiempo de ejecución en nanosegundos y en escala logarítmica para matrices de dimensión n.

Como se puede ver en el gráfico, a medida que la dimensión de la matriz crece, también lo hace el tiempo de ejecución que demandan los algoritmos de resolución. De hecho, se puede observar que este crecimiento es exponencial. Esto era esperable, ya que los algoritmos de eliminación gaussiana tienen una complejidad teórica del orden de $O(n^3)$ lo que se complementa satisfactoriamente con el crecimiento exponencial que presenta el gráfico. Por lo tanto, se puede concluir que los resultados obtenidos son coherentes con lo que se esperaba y se alinean con los conocimientos previos sobre la complejidad temporal de estos algoritmos.

3.2. Coste computacional de eliminación gaussiana con pivoteo

Siguiendo con el mismo hilo de pensamiento que se usó en la sección anterior, esta nueva experimentación busca relacionar los tiempos de ejecución para sistemas de diferentes dimensiones pero esta vez utilizando el algoritmo de eliminación gaussiana con pivoteo.

En este caso particular, las únicas operaciones extra que pueden marcar la diferencia es el intercambio de filas que se realiza al encontrar una diagonal con un valor nulo. Por lo que se espera que los datos obtenidos de esta experimentación sean equivalentes al experimento anterior o, en su defecto, que se vean unos tiempos de ejecución algo superiores que reflejen las operaciones extra realizadas por los intercambios de filas.

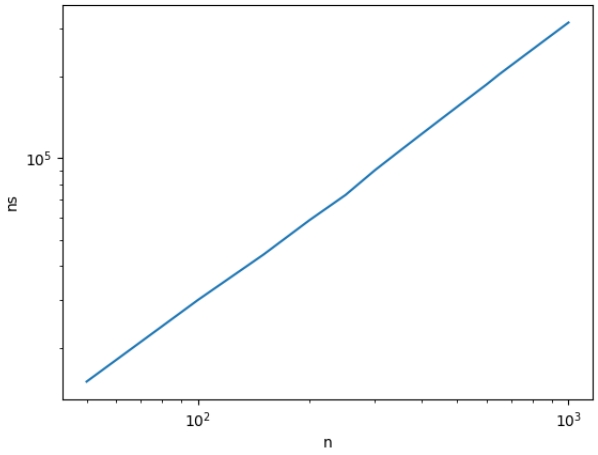


Como se puede ver en el gráfico, los resultados arrojados son similares a la experimentación anterior para el mismo algoritmo sin pivoteo. La comparación de los resultados de ambas experimentaciones permite concluir que el uso del pivoteo no tiene un impacto significativo en el tiempo de ejecución del algoritmo de eliminación gaussiana.

3.3. Eficiencia de la estructura tridiagonal

En esta experimentación se busca medir el tiempo de ejecución del algoritmo de eliminación gaussiana sin pivoteo pero esta vez para matrices tridiagonales de diferente dimensión con el objetivo de analizar estos nuevos tiempos de ejecución.

En función de lo que ya se estudió en este trabajo, pensamos que los tiempos de ejecución que se medirán esta vez deberían ser más pequeños. La razón principal es que tanto la eliminación gaussiana como la resolución del sistema en el caso de las matrices tridiagonales tiene iteraciones de tamaño fijo como ya se había mencionado anteriormente. Este aspecto particular simplifica mucho los ciclos de los algoritmos haciendo que su coste computacional se vea reducido drásticamente. Considerando que solo se necesita 1 ciclo para llegar a los resultados deseados, esperamos obtener una muestra de datos que exponga un crecimiento lineal en función del tamaño de la dimensión de la matriz.



Como deja en evidencia este nuevo gráfico, los tiempos de ejecución son considerablemente menores que los obtenidos en experimentaciones previas. Además, también se puede observar en esta ocación que el crecimiento es lineal, ya que al tener estructuras de matrices en forma tridiagonal y sacar provecho de ellas se puede conseguir que la complejidad teórica de los algoritmos se reduzca hasta el orden de $O(n)$.

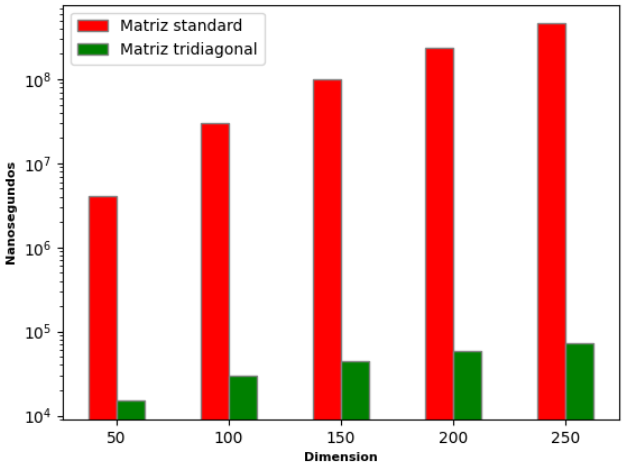
Cuadro 1: Experimento 1

Dimensión	Tiempo (ms)
5	14000
10	51000
20	301000
30	946000
40	2173000
50	4152000
100	31215000
200	239567000
300	825820000
400	1993258000
500	3851622000
1000	31365448000

Cuadro 2: Experimento 3

Dimensión	Tiempo (ms)
5	6000
10	5000
20	8000
30	11000
40	14000
50	17000
100	35000
200	69000
300	110000
400	139000
500	176000
1000	363000

Con el objetivo de dar un apoyo visual, se presenta el siguiente gráfico de barras, donde se puede apreciar mas fácilmente la notable diferencia en tiempos de ejecución, nuevamente medido en nanosegundos y escala logarítmica.



Haciendo la comparación para ejecuciones de igual dimensión del sistema, se puede ver que nuestro algoritmo optimizado para matrices tridiagonales obtiene una ganancia en tiempos de ejecución de hasta un 50 % con respecto al algoritmo utilizado para matrices estándar. Por lo tanto, se corrobora con esta experimentación que todos los detalles explorados para optimizar tanto la eliminación gaussiana como la resolución de sistemas aprovechando la estructura de la matriz funcionaron correctamente.

3.4. Eficiencia de la factorización LU

En esta última experimentación, se busca ver qué tanta eficiencia se está consiguiendo al guardar la matriz L de la factorización LU del sistema para poder calcular la triangulación de nuevos vectores solución sin volver a recalcular la triangulación del sistema original.

Como hipótesis experimental, esperamos obtener una muestra de datos que justifique la creación de dicha matriz L para simplificar los procesos de triangulación de un mismo sistema. Es decir, esperamos ver una mejora de tiempo en el algoritmo cuando no requiere volver a triangular el sistema original ya que todas esas operaciones es costo computacional que se está ahorrando al poder usar la matriz L solo sobre los nuevos vectores solución.

Una diferencia con los experimentos anteriores es que ahora hay 2 variables para modificar durante la experimentación. Una es la dimensión del sistema y otra es la cantidad de vectores solución que se quieren resolver. Nuestra idea es armar una muestra de datos para analizar los tiempos en función de la dimensión de la matriz, otra muestra de datos para analizar los tiempos en función de la cantidad de vectores solución, y por último otra muestra de datos que analice ambas variables creciendo a la par.

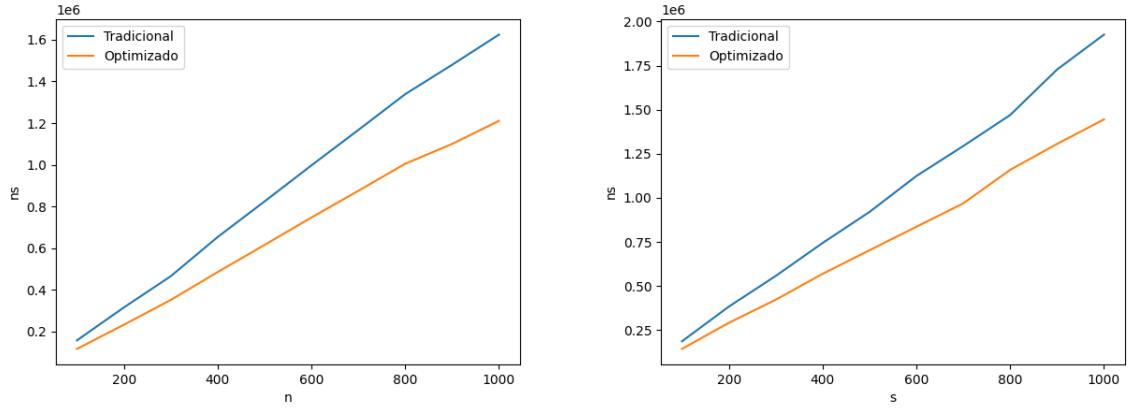


Figura 6: Variación de tiempos de cómputo en función de la dimensión (n) y las repeticiones (s)

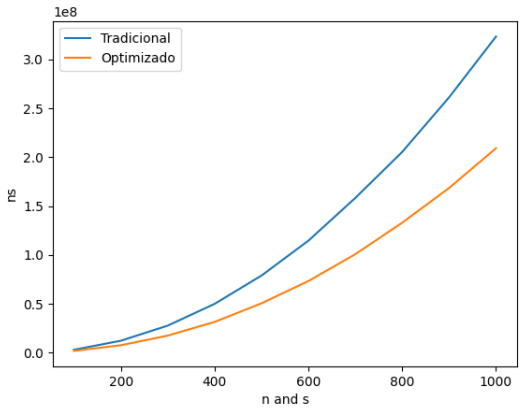
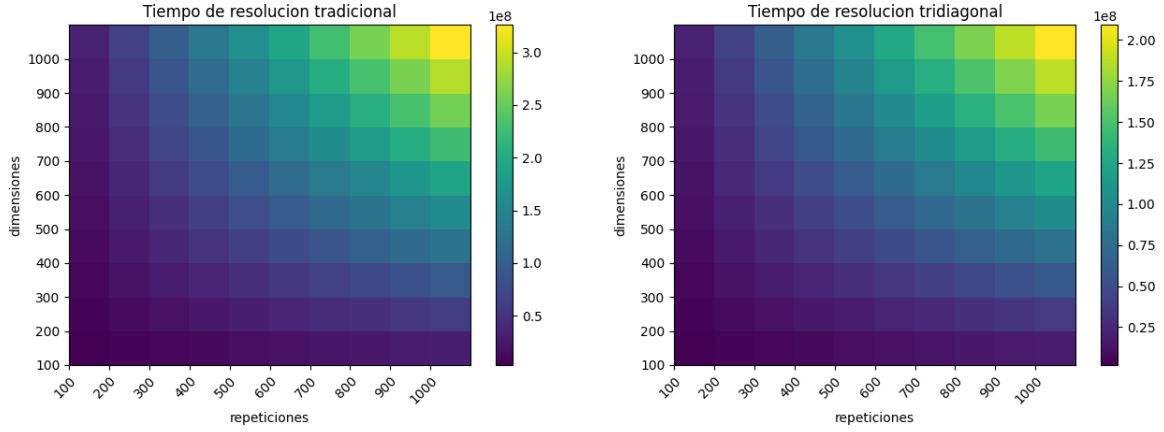


Figura 7: Variación del tiempo en función del crecimiento de dimensiones y repeticiones a la par

Los gráficos de la figura 6 muestran la relación del tiempo en función de la dimensión de la matriz (n) y de la cantidad de vectores solución (repeticiones) (s) respectivamente. Mientras que el gráfico de la figura 7 es el que muestra el crecimiento conjunto de ambas variables. Los resultados obtenidos indican que los algoritmos optimizados que se aprovechan de la matriz L de la factorización LU presentaron un rendimiento significativamente mejor que los algoritmos estándar en todos los casos analizados. Esto se refleja en una reducción notable de los tiempos de ejecución para la búsqueda de nuevas soluciones. Además, se pudo observar que el impacto de la dimensión de la matriz y la cantidad de vectores solución en los tiempos de ejecución es significativo, pero la diferencia entre los dos tipos de algoritmos se mantiene constante.

Como se ha explicado anteriormente, el objetivo de este experimento es el análisis en conjunto de ambas variables (dimensión de la matriz y cantidad de repeticiones). Con eso en mente, se exponen los siguientes heatmaps.



Nótese como el diferente valor de las escalas logarítmicas corrobora los resultados anteriores, donde la resolución tradicional significaba un considerable aumento en los tiempos de ejecución respecto a la tridiagonal.

Habiendo marcado este comportamiento, dejamos atrás las comparativas y nos enfocamos en la relación de las variables en sí. Esta a la vista que los tiempos de ejecución no se ven fuertemente afectados si tan solo aumentamos una de las dos variables. Sin embargo, la situación es otra si aumentamos ambas de manera progresiva. En este caso, evidenciado que el algoritmo sufre una notable caída en la performance. Esto valida los resultados obtenidos en la figura 7, donde se pudo notar un marcado crecimiento en el tiempo de ejecución comparado con los gráficos de la figura 6.

En conclusión, esta experimentación corrobora que las optimizaciones realizadas con la matriz L de la factorización LU fueron determinantes para reducir el costo computacional de los algoritmos.

4. Conclusiones

En este trabajo, hemos investigado en detalle los procedimientos para aplicar la eliminación gaussiana a una matriz y posteriormente resolver sistemas de ecuaciones lineales. Además, hemos explorado cómo se puede aprovechar la estructura de una matriz tridiagonal para mejorar la eficiencia computacional en comparación con el caso estándar. Los resultados experimentales han demostrado una ventaja significativa en términos de costo computacional, con mejoras de hasta el 50 % en comparación con el caso estándar. Esto demuestra que el análisis estructural de matrices puede llevar a desarrollar herramientas más eficientes para trabajar en el campo de matrices.

Además, hemos visto aplicaciones prácticas de estas metodologías en el fenómeno de difusión, que es de gran importancia en muchas áreas como la física, la química, la biología, la ingeniería, la economía y la informática, entre otras. Y pudimos ver de cerca el desarrollo de herramientas óptimas para calcular una simulación precisa de dicho fenómeno en un entorno de experimentación controlado.

En resumen, este trabajo nos permitió obtener una comprensión detallada de los procedimientos comunes que se suelen llevar a cabo con matrices, así como también cómo optimizar dichos procedimientos utilizando estructuras matriciales específicas para maximizar su eficiencia en ambientes que demanden un elevado rendimiento en términos de coste computacional.

4.1. Dificultades

- Implementar el algoritmo de *Backward Sustitution* en Python.
- Pensar en las ventajas de tener matrices con estructura tridiagonal.
- Identificar la ecuación (13) para triangular nuevos vectores solución sin triangular nuevamente el sistema original.
- Entender el operador Laplaciano.
- Entender el fenómeno estocástico de la difusión.
- Definir una forma efectiva de transmitir todo el conocimiento adquirido a través de este informe.
- LaTeX