Análisis de Rendimiento Multilenguaje: Diferencias Finitas y algoritmo Thomas

2025 Universidad distrital Francisco Jose de Caldas

Angie Lorena Pineda Morales
Juan David de los Rios Mahecha

Introduccion

Introduccion

- La resolución numérica de EDPs es clave en la simulación de fenómenos físicos.
- Se estudia la ecuación de Poisson unidimensional con condiciones de Dirichlet homogéneas:

$$-u''(x) = f(x), \quad x \in (0,1), \quad u(0) = u(1) = 0$$

- Se utiliza una función fuente conocida:

$$f(x) = -\pi^2 \cos(\pi x)$$

Solucion exacta

- La solución analítica es:

$$u(x) = \cos(\pi x) + 2x - 1$$

- Esta ecuación modelo es comúnmente utilizada para validar métodos numéricos debido a su estructura simple y solución conocida.

Método Numérico

- Se aplica el método de diferencias finitas centradas en una malla uniforme.
- Esto conduce a un sistema lineal tridiagonal.
- Se resuelve con el algoritmo de Thomas, eficiente para este tipo de sistemas.

Objetivo del Estudio

- Comparar el rendimiento computacional y la precisión numérica del enfoque en:
  - Python
  - C++
  - Fortran
- Métricas evaluadas:
  - Tiempo de ejecución
  - Error máximo respecto a la solución exacta
  - Comportamiento con distintos tamaños de malla

Diferencias finitas

Planteamiento del Problema

Ecuación de Poisson 1D

$$-u''(x) = f(x), \quad x \in (0,1), \quad u(0) = u(1) = 0$$

- $f : [0,1] \to \mathbb{R}$, con $f \in C^2([0,1])$
- Se busca una solución numérica en el intervalo cerrado con condiciones de Dirichlet homogéneas.

Discretización del Dominio

- Dividimos $[0, 1]$ en N subintervalos uniformes:

$$h = \frac{1}{N}, \quad x_i = ih, \quad i = 0, 1, \ldots, N$$

- Condiciones de frontera:

$$u_0 = u(0) = 0, \quad u_N = u(1) = 0$$

- Se buscan las aproximaciones $u_1, u_2, \ldots, u_{N-1}$

Aproximación de la Derivada Segunda

- En los nodos interiores x_i, $1 \leq i \leq N-1$:

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

- Sustituyendo en la ecuación diferencial:

$$-\left(\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}\right) = f(x_i)$$

- Multiplicando ambos lados por h^2:

$$-u_{i-1} + 2u_i - u_{i+1} = h^2 f(x_i)$$

Sistema Lineal Tridiagonal

- Esto conduce a un sistema lineal de la forma:

$$A\mathbf{u} = \mathbf{b}$$

  donde:
  - A es una matriz tridiagonal simétrica de tamaño $(N-1) \times (N-1)$
  - $\mathbf{u} = [u_1, \ldots, u_{N-1}]^T$
  - $\mathbf{b}_i = h^2 f(x_i)$
- El sistema se resuelve mediante el método de Thomas.

Algoritmo de Thomas

Planteamiento del Sistema Tridiagonal

Sistema Lineal Tridiagonal

$$A\mathbf{u} = \mathbf{d}$$

- $A \in \mathbb{R}^{n \times n}$ es una matriz tridiagonal con:
  - Diagonal inferior: $a_2, a_3, \ldots, a_n$
  - Diagonal principal: $b_1, b_2, \ldots, b_n$
  - Diagonal superior: $c_1, c_2, \ldots, c_{n-1}$

Planteamiento del Sistema Tridiagonal

- El sistema es de la forma:

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ 0 & a_3 & b_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n & b_n \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

Eliminación hacia Adelante

- Se define una diagonal modificada c'_i y un vector fuente modificado d'_i para convertir el sistema en triangular superior.
- Modificaciones:

$$c'_i = \frac{c_i}{b_i}, \quad d'_i = \frac{d_i - a_i d_{i-1}}{b_i}, \quad i = 2, 3, \ldots, n$$

- Estas modificaciones eliminan los elementos debajo de la diagonal principal.

Sustitución hacia Atrás

- Una vez que el sistema está en forma triangular superior, se procede con la sustitución hacia atrás.
- El primer valor se calcula como:

$$u_n = d'_n$$

- Luego, los valores anteriores se calculan de forma recursiva:

$$u_i = d'_i - c'_i u_{i+1}, \quad i = n-1, n-2, \ldots, 1$$

Complejidad Computacional

- El método de Thomas tiene una complejidad lineal en el número de ecuaciones:

$$O(n)$$

- Esto es considerablemente más eficiente que la eliminación gaussiana general, que tiene una complejidad de:

$$O(n^3)$$

- Por lo tanto, el método de Thomas es ideal para sistemas derivados de diferencias finitas en problemas unidimensionales, donde la matriz es tridiagonal.

Método de Gauss-Seidel

Planteamiento del Sistema Lineal

Sistema Lineal General

$$A\mathbf{x} = \mathbf{b}$$

- $A \in \mathbb{R}^{n \times n}$ es una matriz cuadrada con coeficientes a_{ij}.
- El sistema es de la forma:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Actualización Iterativa

- Se parte de una estimación inicial $\mathbf{x}^{(0)}$ y se actualiza cada componente de $\mathbf{x}$ de forma secuencial.
- La fórmula de actualización es:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

- Se repite el proceso hasta que se cumpla una tolerancia deseada en las diferencias entre iteraciones.

Esquema del Algoritmo

1. Elegir una estimación inicial $\mathbf{x}^{(0)}$.
2. Para cada $i = 1$ hasta n, actualizar x_i usando la fórmula anterior.
3. Repetir el procedimiento hasta que:

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \text{tolerancia}$$

4. La convergencia está garantizada si la matriz A es **diagonalmente dominante** o **definida positiva**.
5. Si no se cumple esta condición, el método puede no converger.

Complejidad Computacional

- Cada iteración del método tiene un costo computacional de:

$$O(n^2)$$

  debido a que cada incógnita se actualiza considerando hasta n elementos en el peor de los casos.

- En matrices dispersas con pocas entradas no nulas por fila, el costo por iteración puede reducirse significativamente a:

$$O(n)$$

  ya que solo se procesan los elementos no nulos.

- Si el sistema es **mal condicionado** o la matriz no es **diagonalmente dominante**, el método puede requerir muchas iteraciones o incluso divergir, lo que hace necesario considerar técnicas como *SOR* o *GMRES*.

Método de Jacobi

Planteamiento del Sistema Lineal

Sistema Lineal General

$$A\mathbf{x} = \mathbf{b}$$

- $A \in \mathbb{R}^{n \times n}$ es una matriz cuadrada con coeficientes a_{ij}.
- El sistema se puede escribir como:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Actualización Iterativa

- El método de Jacobi parte de una estimación inicial $\mathbf{x}^{(0)}$ y actualiza cada componente de forma **independiente**, usando solo los valores de la iteración anterior.
- La fórmula de actualización es:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

- Todas las variables se actualizan de manera simultánea al finalizar el cálculo de la iteración.

Esquema del Algoritmo

1. Elegir una estimación inicial $\mathbf{x}^{(0)}$.
2. Para cada $i = 1$ hasta n, actualizar x_i usando la fórmula anterior.
3. Repetir el procedimiento hasta que:

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \text{tolerancia}$$

4. La convergencia del método de Jacobi se garantiza si la matriz A es **diagonalmente dominante** o **definida positiva**.
5. Sin estas condiciones, el método puede fallar en converger o requerir un número elevado de iteraciones.
6. Generalmente, Jacobi converge más lentamente que el método de Gauss-Seidel.

Complejidad Computacional

- Cada iteración tiene un costo computacional de:

$$O(n^2)$$

  ya que cada incógnita depende de las n componentes anteriores.

- En sistemas dispersos, el costo puede reducirse a:

$$O(n)$$

  si la matriz tiene pocas entradas no nulas por fila.

- Como todas las actualizaciones se realizan de forma simultánea, el método es **altamente paralelizable**.

Resultados computacionales

Resultados computacionales

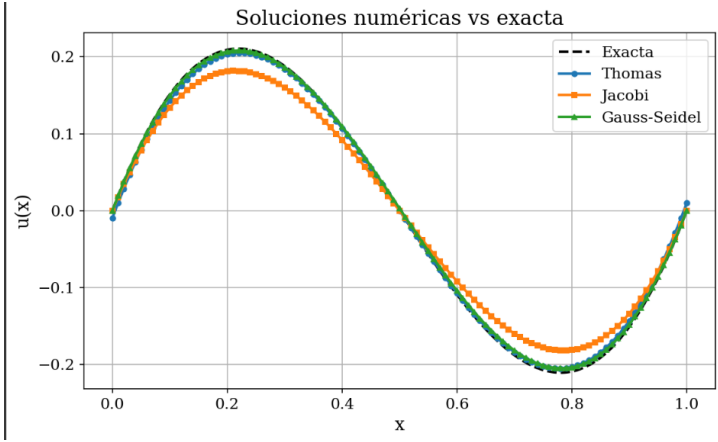
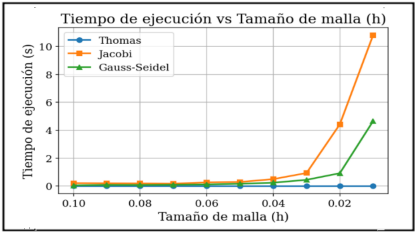
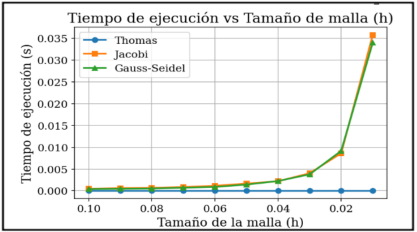


Figure 1: *Solución de la ecuación diferencial en una dimensión*

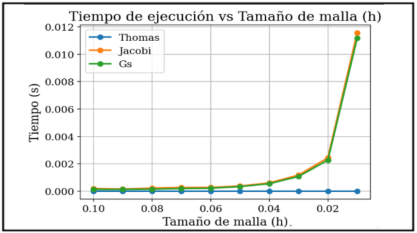
Tiempos de ejecución (s) vs Tamaño de la malla(h) para 1D



Python

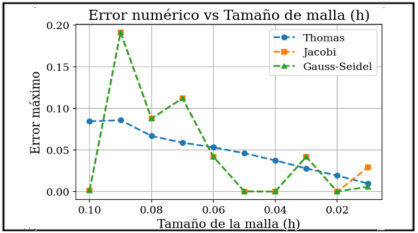


C

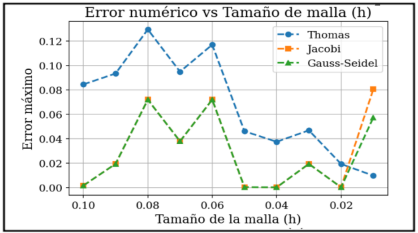


Fortran

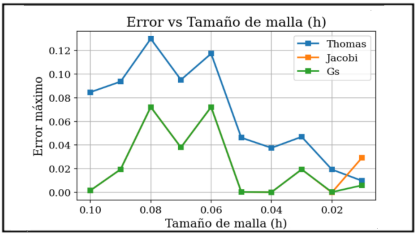
Error de ejecución (s) vs Tamaño de la malla(h) para 1D



Python



C



Fortran

Analisis de resultados computacionales para diferencias finitas en una dimension

Criterio	**Jacobi**	**G-S**	**Thomas**
Tipo	2D gen.	2D gen.	1D/2D tri.
Iter./Dir.	Iter.	Iter.	Dir.
Convergencia	Lenta	Mejor	Exacta
Complejidad	$O(Nk)$	$O(Nk')$	$O(N)$
Memoria	$O(N)$ x2	$O(N)$	$O(N)$
Tiempo	Alto	Medio	Bajo
Ventaja	Simple	Más rápido	Estable y exacto
Desventaja	Muy lento	Puede oscilar	Solo tridiagonal

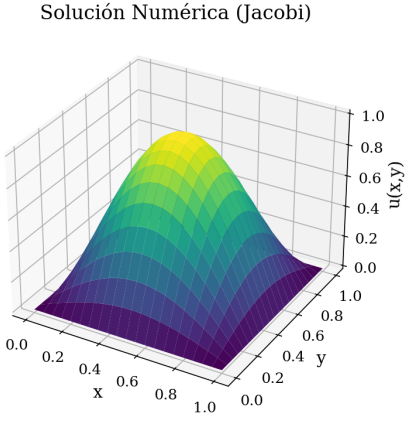
Table 1: Resumen comparativo de métodos

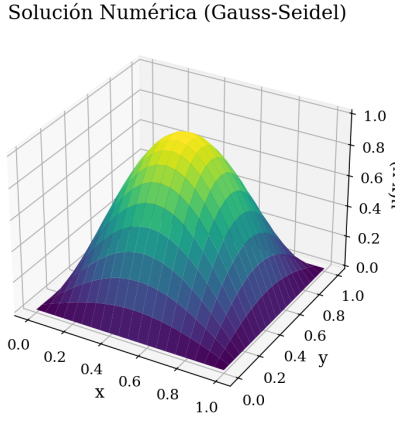
Los métodos iterativos de Jacobi y Gauss-Seidel permiten resolver la ecuación de Poisson en dos dimensiones con relativa facilidad, siendo el segundo más eficiente gracias a su convergencia más rápida. Sin embargo, ambos presentan tiempos de ejecución significativamente mayores que el método directo de Thomas, el cual es altamente eficiente pero limitado a sistemas tridiagonales, típicamente en 1D o en problemas 2D con ciertas condiciones estructurales.

En términos de complejidad, Thomas ofrece un rendimiento lineal $O(N)$, mientras que Jacobi y Gauss-Seidel dependen del número de iteraciones necesarias para converger, típicamente $O(Nk)$. Por tanto, la elección del método depende del equilibrio entre precisión, estructura del sistema y recursos computacionales disponibles.

Métodos de Diferencias Finitas en 2D

Resultados computacionales





Ecuación Diferencial Parcial en 2D

Consideremos una ecuación de Poisson en dos dimensiones:

$$-\nabla^2 u(x, y) = f(x, y)$$

donde ∇^2 es el operador laplaciano, $u(x, y)$ es la función desconocida, y $f(x, y)$ es la fuente.

La solución $u(x, y)$ puede ser aproximada mediante el método de **diferencias finitas**.

- Se discretiza la ecuación en una malla regular.
- Se resuelve iterativamente para encontrar los valores de u en cada punto de la malla.

Usamos una malla cuadrada con paso h en cada dirección. Definimos los puntos (x_i, y_j) donde la solución es desconocida.

Ecuación Diferencial Parcial en 2D

El operador laplaciano $\nabla^2 u$ se aproxima mediante diferencias finitas de segundo orden:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \quad \text{y} \quad \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

Así que la ecuación diferencial se convierte en:

$$-\frac{1}{h^2}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}\right) = f_{i,j}$$

Esto nos da un sistema de ecuaciones lineales para resolver $u_{i,j}$.

Para resolver el sistema de ecuaciones lineales, usamos métodos iterativos como **Jacobi** y **Gauss-Seidel**.

Ecuación Diferencial Parcial en 2D

- **Método de Jacobi:** La solución en cada punto de la malla se actualiza usando los valores de los puntos vecinos. La fórmula es:

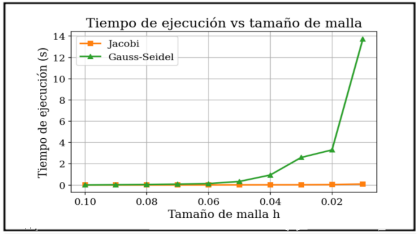
$$u_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)} - h^2 f_{i,j}\right)$$

- **Método de Gauss-Seidel:** A diferencia de Jacobi, se usan los valores más recientes de u en la misma iteración.

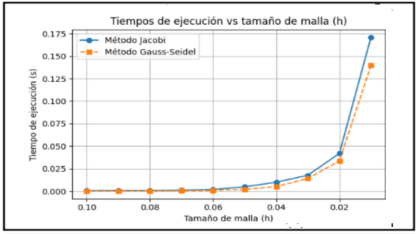
Estos métodos convergen iterativamente hasta que el error entre las iteraciones es lo suficientemente pequeño.

Estos métodos permiten resolver sistemas grandes con un costo computacional razonable.

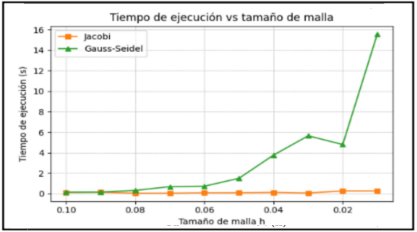
Tiempos de ejecución (s) vs Tamaño de la malla(h) para 2D



Python

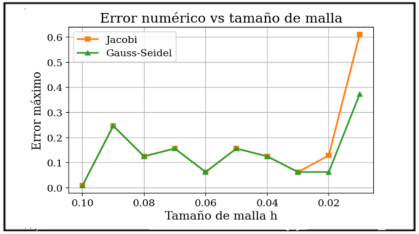


C

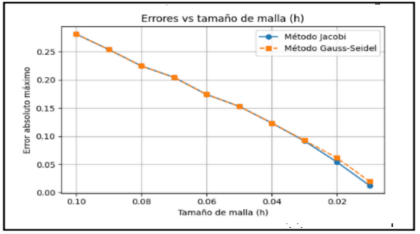


Fortran

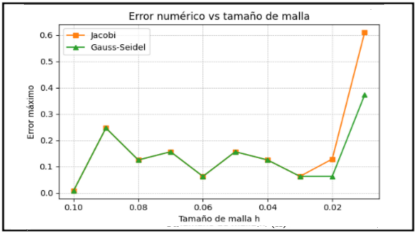
Error de ejecución (s) vs Tamaño de la malla(h) para 2D



Python



C



Fortran

Analisis de resultados computacionales para diferencias finitas en dos dimensiones

Criterio	Jacobi	Gauss-Seidel
Tipo	2D general	2D general
Iteraciones/Dirección	Iter.	Iter.
Convergencia	Lenta	Mejor
Complejidad	$\mathcal{O}(N \cdot k)$	$\mathcal{O}(N \cdot k')$
Memoria	$\mathcal{O}(N)$ x2	$\mathcal{O}(N)$
Tiempo	Alto	Medio
Ventaja	Simple	Más rápido
Desventaja	Muy lento	Puede oscilar

Table 2: Comparación entre los métodos Jacobi y Gauss-Seidel en 2D.

En la comparación de los métodos de Jacobi y Gauss-Seidel para resolver problemas en dos dimensiones, se observó que ambos métodos presentan características distintas que afectan su desempeño. El método de Jacobi, aunque es simple de implementar, es más lento debido a su convergencia más lenta y su mayor necesidad de memoria, ya que requiere almacenar dos matrices de tamaño N. Por otro lado, Gauss-Seidel, aunque también es un método iterativo, presenta una convergencia más rápida y requiere menos memoria, lo que lo hace más eficiente en comparación con Jacobi. Sin embargo, Gauss-Seidel puede experimentar oscilaciones durante el proceso de iteración, especialmente en casos donde la condición del sistema no es ideal. En términos de complejidad computacional, ambos métodos son de orden $O(N \cdot k)$, pero Gauss-Seidel generalmente realiza menos iteraciones, lo que mejora su tiempo de ejecución en la práctica.

Ecuación de Poisson 2D de una placa cuadrada hueca

Ecuación de Poisson 2D de una placa cuadrada hueca

Discretización de las PDE

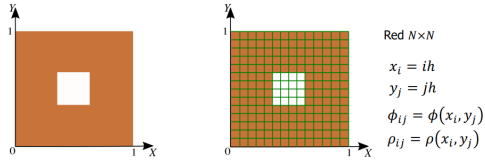


Figure 2: *Placa rectangular hueca y condiciones de borde de Dirichlet*

- Consideremos como ejemplo la ecuación de Poisson en 2D:

$$\nabla^2 \phi(\vec{r}) = -4\pi\rho(\vec{r})$$

- Con una placa rectangular hueca y condiciones de borde de Dirichlet en los bordes externos e internos.

Ecuación de Poisson 2D de una placa cuadrada hueca

- Definimos una red de puntos que cubre la región de interés en el plano (x, y) de ancho fijo h.
- Para las derivadas usamos elementos finitos:

$$\nabla^2\phi(\vec{r}) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = \frac{\phi_{i+1,j} + \phi_{i-1,j} - 2\phi_{i,j}}{h^2} + \frac{\phi_{i,j+1} + \phi_{i,j-1} - 2\phi_{i,j}}{h^2}$$

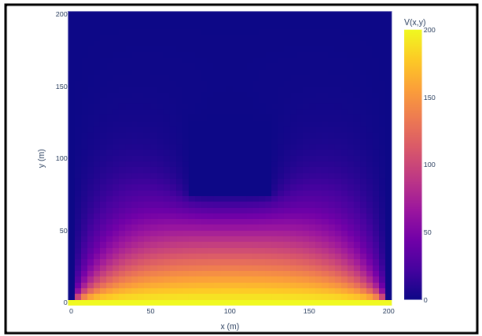
- Entonces la PDE se transforma en:

$$\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j} = -4\pi h^2 \rho_{i,j}$$

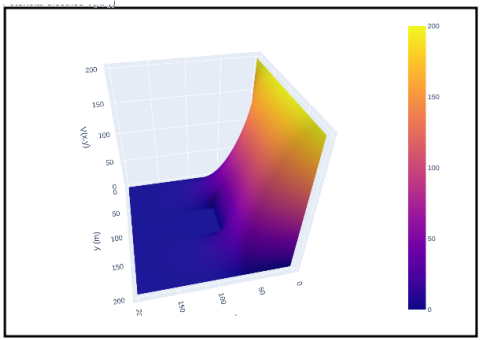
- Que se puede escribir en forma matricial:

$$\mathcal{M}\Phi = \mathcal{S}$$

Potencial eléctrico V (x,y) de la placa cuadrada hueca.



Mapa de calor de potencial eléctricico V(x,y)



Potencial eléctrico V(x,y)

Relación entre tamaño de malla, número de iteraciones, tiempo y error.

Malla (X×Y)	Iteraciones	Tiempo Real	Tiempo Usuario	Error Final
50 × 50	954	0m18.176s	0m0.034s	9.9111E-06
100 × 100	3364	0m22.235s	0m0.378s	9.9964E-06
150 × 150	6970	0m28.088s	0m1.819s	9.9976E-06
200 × 200	11348	0m28.749s	0m5.581s	9.9959E-06
250 × 250	16923	0m38.284s	0m15.448s	9.9969E-06
300 × 300	23391	0m56.848s	0m32.559s	9.9967E-06
350 × 350	30690	1m18.784s	0m54.356s	9.9999E-06
400 × 400	38307	1m53.430s	1m33.487s	9.9989E-06

La tabla presenta los resultados obtenidos al resolver numéricamente la ecuación de Poisson en mallas cuadradas de distintos tamaños. A medida que aumenta la resolución de la malla, se observa un incremento notable en el número de iteraciones requeridas y en los tiempos de ejecución, tanto en tiempo real como de CPU. Sin embargo, el error final se mantiene constante del orden de 10^{-5}, lo que evidencia la estabilidad del método numérico utilizado frente al tamaño de malla.

Tiempo (s) vs Tamaño de la malla

.