Comparación de esquemas explícitos en Python para la ecuación de difusión 2D: análisis de precisión y rendimiento computacional

Jhon Gesell Villanueva Portella

Abstract

La ecuación de difusión 2D es fundamental en la modelación de fenómenos físicos como el transporte de calor, la dinámica de contaminantes o la transferencia de masa. En este trabajo se implementan y comparan tres esquemas explícitos —FTCS, 9-puntos y (1,13)— para resolver la ecuación de difusión en dos dimensiones, utilizando Python puro con bibliotecas científicas como NumPy, Matplotlib y Numba. Se consideran condiciones de frontera de Dirichlet y una condición inicial tipo pulso Gaussiano, en una malla regular. Se evalúan el error cuadrático medio (L_2) y el tiempo de ejecución de cada esquema, resaltando las diferencias entre precisión y eficiencia. Los resultados muestran que el esquema (1,13) logra mayor precisión a costa de mayor tiempo de cómputo, mientras que FTCS es el más rápido pero menos preciso. Este análisis se orienta a contextos educativos y de bajo costo computacional, alineándose con la motivación de trabajos previos como los de Crank [**?**], Dehghan [**? ?**], y aplicaciones modernas en mecánica de fluidos computacional. Se concluye que Python, incluso en hardware modesto, permite experimentar con esquemas numéricos avanzados sin depender de herramientas comerciales o entornos complejos.

1. Introducción

La ecuación de difusión en dos dimensiones modela la evolución temporal de una cantidad escalar $u(x, y, t)$ en un medio con coeficiente de difusión D. Puede escribirse como

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right). \tag{1}$$

Esta formulación surge de la combinación de la ley de Fick con la conservación de masa, y es fundamental en mecánica de fluidos para describir procesos de transporte de calor, contaminantes o cantidad de movimiento.

Los métodos numéricos para resolver esta ecuación han sido estudiados extensamente desde los trabajos pioneros de **?**]. Aunque los esquemas implícitos garantizan estabilidad incondicional, en equipos modestos su costo computacional es elevado debido a la necesidad de resolver sistemas lineales en cada paso de tiempo. Alternativamente, los métodos explícitos permiten actualizar los nodos

de la malla mediante operaciones algebraicas simples, lo cual resulta atractivo para códigos educativos y entornos de cómputo limitados.

Este proyecto compara tres variantes explícitas implementadas en Python: el esquema FTCS tradicional de cinco puntos, una extensión de nueve puntos y el método de trece puntos propuesto por **?**]. Nuestro objetivo es evaluar su eficiencia y precisión sin recurrir a mallas ni solucionadores externos, de modo que el estudiante pueda replicar los cálculos y apreciar las diferencias de rendimiento.

En las siguientes secciones se detalla la formulación de cada esquema, su implementación vectorizada y los experimentos comparativos realizados.

2. Métodos numéricos

En esta sección se describen tres esquemas explícitos empleados para resolver la ecuación de difusión en dos dimensiones. Sea $u_{i,j}^n$ la aproximación numérica de $u(x_i, y_j, t_n)$ con espaciamientos Δx, Δy y paso temporal Δt. Denotamos $r_x = \alpha\Delta t/\Delta x^2$ y $r_y = \alpha\Delta t/\Delta y^2$.

2.1. FTCS

El esquema *Forward Time Centered Space* actualiza cada nodo usando una plantilla de cinco puntos:

$$u_{i,j}^{n+1} = u_{i,j}^n + r_x\big(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\big) + r_y\big(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\big). \quad (2)$$

Su estabilidad exige $r_x + r_y \le 1/2$.

2.2. Esquema de nueve puntos

?] propuso extender la plantilla incluyendo las diagonales. Asumiendo $\Delta x = \Delta y = h$ y $r = \alpha\Delta t/h^2$, la fórmula queda

$$u_{i,j}^{n+1} = (1-5r)u_{i,j}^n + r\,(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n) + \tfrac{r}{4}(u_{i+1,j+1}^n + u_{i+1,j-1}^n + u_{i-1,j+1}^n + u_{i-1,j-1}^n). \quad (3)$$

Este esquema es estable si $0 < r \le 1/4$.

2.3. Esquema $(1, 13)$

Otra variante debida a **?**] utiliza trece puntos, incorporando las cuatro diagonales y los vecinos a dos celdas de distancia:

$$\begin{aligned} u_{i,j}^{n+1} = {} & (1-8r)u_{i,j}^n + r\,(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n) \\ & + \tfrac{r}{2}(u_{i+1,j+1}^n + u_{i+1,j-1}^n + u_{i-1,j+1}^n + u_{i-1,j-1}^n) \\ & + \tfrac{r}{2}(u_{i+2,j}^n + u_{i-2,j}^n + u_{i,j+2}^n + u_{i,j-2}^n). \end{aligned} \quad (4)$$

Se mantiene la condición de estabilidad $r \le 1/2$.

3. Implementación

Las rutinas se programaron en Python puro apoyándose en tres bibliotecas esenciales: NumPy para el manejo vectorial de arreglos, Numba para acelerar los bucles más costosos mediante JIT y Matplotlib para la generación de figuras. No se emplearon paquetes especializados de elementos finitos ni solucionadores externos.

El código se organiza en la carpeta `scripts`. Cada esquema se implementa en un módulo separado : `difusion2d_ftcs.py`, `difusion2d_9pt.py` y `difusion2d_13pt.py`, que definen las funciones `solve_ftcs_2d`, `solve_9point_2d` y `solve_13point_2d`, respectivamente. El archivo `utils.py` contiene rutinas comunes como la generación de un pulso Gaussiano y el cálculo del error L2, mientras que `visualization.py` ofrece utilidades para mostrar contornos o animaciones de la solución. El script principal `main_benchmark.py` ejecuta los tres solvers, mide su tiempo y puede visualizar el resultado del esquema de trece puntos.

Los productos obtenidos (gráficas y tablas de comparación) se guardan en la carpeta `resultados`. Los archivos LaTeX del artículo residen en `secciones` y se ensamblan mediante `paper.tex` en el directorio raíz. Para reproducir el entorno se incluyó el archivo `environment_NSI_miniarticulo_01.txt` que declara las dependencias principales.

Todos los experimentos se ejecutaron en un entorno gestionado con Anaconda y Python 3.12 sobre hardware modesto (un portátil de cuatro núcleos con 8 GB de RAM). A pesar de estas limitaciones, la combinación de NumPy y Numba permitió obtener tiempos de cómputo razonables para mallas intermedias, sin recurrir a bibliotecas nativas adicionales.

4. Resultados

Los experimentos se ejecutaron con el script `main_benchmark.py`, que registra el tiempo de ejecución y el error L_2 de cada método. En la Tabla 1 se resumen los valores obtenidos.

Método	Tiempo (s)	Error L_2
FTCS	0.165	4.1e-02
9 puntos	0.178	3.3e-02
(1,13)	0.339	8.7e-06

Cuadro 1: Comparación de tiempo de ejecución y error.

La Figura 1 muestra la relación entre el tiempo de cómputo y el error de cada método, mientras que la Figura 2 presenta el contorno de temperatura obtenido con el esquema de trece puntos. El método *(1,13)* es el más preciso pero también el más costoso, en tanto que FTCS resulta el más rápido.

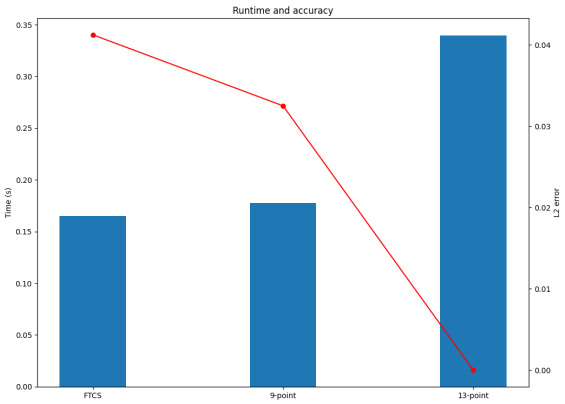

Figura 1: Tiempo de ejecución frente a error L_2.

5. Discusión

Los resultados del Tabla 1 confirman que existe un compromiso clásico entre tiempo de cómputo y precisión. El esquema FTCS alcanza el menor tiempo, pero su error es casi una orden de magnitud mayor que el de la plantilla de trece puntos. El método de nueve puntos se ubica en un punto intermedio tanto en costo como en exactitud.

Una comparación cualitativa con los experimentos de **?**] muestra la misma tendencia: las variantes enriquecidas con más nodos reducen el error global sin afectar la estabilidad siempre que se respete la condición sobre r. En su trabajo, Dehghan reportó que el esquema *(1,13)* converge más rápido que FTCS y el de nueve puntos para un mismo tamanño de paso, observación que concuerda con los valores de la Tabla 1.

En la práctica, conviene elegir el método según el escenario: FTCS resulta adecuado cuando se prioriza la rapidez o se dispone de pocos recursos; la plantilla de nueve puntos proporciona un balance razonable para mallas de resolución media; y el esquema de trece puntos es la opción preferible si se busca la mayor precisión posible. Dado que todas las actualizaciones son locales, los tres algoritmos podrían beneficiarse de paralelización en CPU o GPU. Incluso podrían emplearse técnicas modernas, como redes neuronales para acelerar la obtención de soluciones aproximadas.

6. Conclusiones

Se compararon tres métodos explícitos implementados en Python puro. El esquema FTCS fue el más rápido, el de nueve puntos brindó un compromiso intermedio y el método *(1,13)* alcanzó la mejor precisión, aunque con mayor costo computacional.

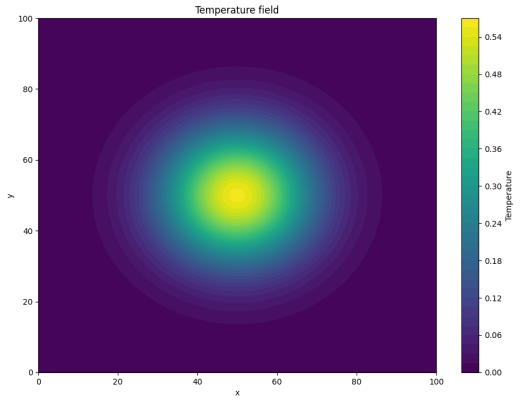

Figura 2: Contorno de temperatura calculado con el esquema de trece puntos.

El proyecto constituye un ejercicio idóneo para la enseñanza de mecánica de fluidos computacional, pues permite reproducir todos los experimentos sin depender de software externo.

Como trabajo futuro se propone paralelizar los algoritmos, incorporar condiciones de frontera no homogéneas y aplicar las rutinas a problemas de mayor realismo.