Métodos Computacionales - Trabajo Práctico 1

Resoluciones Numéricas de Ecuaciones Diferenciales

Mariño Martina, Martinez Kiara
Universidad Torcuato Di Tella

23 de septiembre de 2025

Índice

1. Ecuación del Calor

1.1. Formulación matemática - Derivación de los métodos explícito e implícito

En esta sección vamos a ver cómo se llega a las fórmulas de los métodos explícito e implícito para resolver la ecuación del calor usando diferencias finitas. La idea es empezar de la ecuación original, discretizarla, y después aproximar las derivadas.

La ecuación que queremos resolver

La ecuación del calor en una dimensión es:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad x \in (0,1), \quad t > 0$$

Donde:

- $u(x,t)$ representa la temperatura en el punto x y tiempo t.
- $\alpha > 0$ es la constante de difusión térmica.

Además, tenemos condiciones de frontera de Dirichlet:

$$u(0,t) = 0, \quad u(1,t) = 0$$

y una condición inicial que nos da la distribución de temperatura al inicio:

$$u(x,0) = f(x).$$

En palabras, se trata de una barra de longitud 1 donde los extremos se mantienen a temperatura cero. Sabemos cómo estaba la temperatura al comienzo y queremos ver cómo cambia con el tiempo.

Discretización: malla de espacio y tiempo

Para resolver el problema de forma numérica dividimos el espacio y el tiempo en puntos separados por pasos fijos:

- Dividimos el intervalo espacial $[0,1]$ en N puntos con separación Δx. Así, las posiciones son:
$$x_j = j\Delta x, \quad j = 0, 1, \ldots, N$$
donde $\Delta x = \frac{1}{N}$.
- El tiempo se divide en pasos de tamaño Δt. Los instantes de tiempo quedan como:
$$t_n = n\Delta t, \quad n = 0, 1, 2, \ldots, M$$
con $M\Delta t = T$ siendo el tiempo final de simulación.

Notación: llamamos u_j^n a la aproximación numérica de $u(x_j, t_n)$. Los nodos de frontera son $j = 0$ y $j = N$ (donde ya conocemos u gracias a las condiciones de borde). Los nodos internos son $j = 1, \ldots, N-1$, que son los que vamos a actualizar.

Aproximación de las derivadas con diferencias finitas

Vamos a reemplazar las derivadas por aproximaciones usando diferencias finitas:

- Para la derivada temporal usamos una **diferencia hacia adelante**:

$$\frac{\partial u}{\partial t}(x_j, t_n) \approx \frac{u_j^{n+1} - u_j^n}{\Delta t}.$$

- Para la segunda derivada espacial usamos **diferencias centradas**:

$$\frac{\partial^2 u}{\partial x^2}(x_j, t_n) \approx \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}.$$

Esto introduce un error, pero mientras Δx y Δt sean pequeños, la aproximación es bastante buena.

Sustitución en la ecuación original — Derivación explícito

Si reemplazamos las aproximaciones en la ecuación del calor, nos queda:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \alpha \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}.$$

Definimos un parámetro muy útil llamado r:

$$r = \frac{\alpha \Delta t}{(\Delta x)^2}.$$

Este parámetro combina la constante física α con el paso de tiempo Δt y el paso espacial Δx.

Reemplazando esta definición en la ecuación anterior:

$$u_j^{n+1} - u_j^n = r\big(u_{j+1}^n - 2u_j^n + u_{j-1}^n\big).$$

El siguiente objetivo es aislar u_j^{n+1} para que quede explícito. Sumamos u_j^n en ambos lados:

$$u_j^{n+1} = u_j^n + r\big(u_{j+1}^n - 2u_j^n + u_{j-1}^n\big).$$

Expandimos el paréntesis:

$$u_j^{n+1} = u_j^n + r\, u_{j+1}^n - 2r\, u_j^n + r\, u_{j-1}^n.$$

Agrupando términos similares, en especial los que dependen de u_j^n:

$$u_j^{n+1} = (1 - 2r)u_j^n + r\, u_{j-1}^n + r\, u_{j+1}^n.$$

Esta última ecuación muestra que el nuevo valor de temperatura en la posición j y tiempo $n+1$ se calcula como una combinación lineal de:

- El valor previo en la misma posición, u_j^n, ponderado por $(1 - 2r)$.
- El vecino de la izquierda, u_{j-1}^n, ponderado por r.
- El vecino de la derecha, u_{j+1}^n, también ponderado por r.

Así, nos queda el esquema **explícito** completo:

$$\boxed{u_j^{n+1} = (1 - 2r)u_j^n + r\, u_{j-1}^n + r\, u_{j+1}^n},$$

Condición de estabilidad: Este método solo funciona bien si se cumple:

$$r = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}.$$

Si no se cumple, la solución explota y empieza a oscilar de manera irreal.

Interpretación: La temperatura en j tiende a "suavizarse" dependiendo de cómo están los vecinos $j-1$ y $j+1$. Es como un promedio que se va corrigiendo paso a paso.

Derivación Método implícito

En este método, la segunda derivada espacial se evalúa usando valores en el tiempo $n+1$, o sea, el nuevo tiempo. Esto da la ecuación:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \alpha \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{(\Delta x)^2}.$$

Reordenando un poco:

$$u_j^{n+1} - r\big(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\big) = u_j^n.$$

Esto es una ecuación donde los u^{n+1} están mezclados, así que no se puede actualizar punto por punto como en el explícito. Hay que resolver un sistema lineal en cada paso de tiempo.

Ventajas y desventajas:

- Ventaja: es **incondicionalmente estable**, no importa el valor de r.
- Desventaja: es más costoso computacionalmente porque hay que resolver un sistema lineal en cada paso.

1.1.1. 7. Forma matricial: sistema tridiagonal

Para organizar mejor el método implícito, agrupamos todas las incógnitas internas en un vector:

$$U^{n+1} = \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \end{bmatrix}.$$

Así, la ecuación se puede escribir como:

$$A\,U^{n+1} = U^n,$$

donde A es una matriz tridiagonal de tamaño $(N-1) \times (N-1)$ con esta forma:

$$A = \begin{bmatrix} 1+2r & -r & 0 & \cdots & 0 \\ -r & 1+2r & -r & \cdots & 0 \\ 0 & -r & 1+2r & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & -r \\ 0 & 0 & 0 & -r & 1+2r \end{bmatrix}.$$

En cada paso temporal, el vector U^n se conoce y tenemos que resolver $A\,U^{n+1} = U^n$ para encontrar el siguiente estado. Esto se puede hacer con funciones como `numpy.linalg.solve` o usando el algoritmo de Thomas, que es más rápido para tridiagonales.

Resumen:

- El método explícito es más simple pero exige $r \leq 0{,}5$ para no explotar.
- El método implícito es más robusto, no tiene restricción en r, pero es más pesado porque hay que resolver un sistema en cada paso.

1.2. Implementación en Python

El siguiente fragmento muestra parte del código implementado:

```python
import numpy as np

def metodo_explicito(alpha, f, dx, dt, T):
    N = int(1/dx) + 1
    M = int(T/dt) + 1
    u = np.zeros((M, N))
    x = np.linspace(0, 1, N)
    u[0, :] = f(x)

    r = alpha * dt / dx**2
    for n in range(0, M-1):
        for j in range(1, N-1):
            u[n+1, j] = u[n, j] + r*(u[n, j+1] - 2*u[n, j] + u[n, j-1])
    return u, x
```

1.3. Resultados

En la Figura ?? se observa la evolución temporal de la temperatura usando el método explícito.

[Aquí se incluirá la figura una vez generada]

1.4. Discusión

Comparando los métodos explícito e implícito se observa que...

2. Ecuación de Transporte

2.1. Formulación

La ecuación de transporte está dada por:

$$\frac{\partial u}{\partial t} = a\frac{\partial u}{\partial x}, \quad u(x,0) = \sin(\pi x) \tag{1}$$

2.2. Resultados y análisis

3. Conclusiones

En este trabajo se implementaron métodos numéricos para resolver ecuaciones diferenciales parciales. Se concluye que...