FÍSICA COMPUTACIONAL

TAREA 9

Presentado por:
Cinthia Alejandra Olvera Bautista

Índice general

2

Introducción

1.1 Diagramas Presión temperatura volumen

Los diagramas **Presión-Temperatura-Volumen** (PTV) son herramientas fundamentales en el análisis termodinámico y de mecánica de fluidos, particularmente en el estudio de sistemas energéticos, turbomáquinas y procesos de flujo compresible. Estos diagramas permiten visualizar las relaciones entre tres variables críticas en el comportamiento de los fluidos bajo diversas condiciones operativas.

En termodinámica clásica, los diagramas tradicionales como los de *Presión-Volumen* (PV) o *Temperatura-Entropía* (TS) han sido ampliamente utilizados. Sin embargo, los diagramas PTV emergen como una alternativa poderosa cuando el análisis requiere incorporar efectos dinámicos y cinemáticos, especialmente en aplicaciones donde la velocidad del fluido es un parámetro determinante (e.g., toberas, difusores, turbinas).

1.1.1 Conceptos Básicos

Un diagrama PTV típico representa:

- Eje X: Temperatura (T) [K] o velocidad (v) [m/s]
- Eje Y: Presión (P) [Pa]
- Curvas características: Líneas de flujo isoentrópico, fronteras de fase, o regiones de transición entre regímenes subsónico y supersónico

Matemáticamente, estas relaciones pueden expresarse mediante la ecuación de estado y conservación de energía:

$$P = \rho RT \quad \text{(Ecuación de estado para gas ideal)}$$

$$h + \frac{v^2}{2} = \text{constante} \quad \text{(Ecuación de Bernoulli para flujo estacionario)}$$

donde ρ es la densidad, R la constante del gas, y h la entalpía.

1.1.2 Aplicaciones

Entre las aplicaciones más relevantes se encuentran:

1. **Diseño de sistemas de propulsión**: Se utilizan diagramas PTV para determinar las condiciones óptimas de presión y temperatura que maximicen el empuje de motores a reacción o sistemas similares.

2. **Optimización de ciclos termodinámicos**: Los diagramas permiten analizar los puntos críticos en ciclos Rankine, Brayton o combinados, mejorando la eficiencia global del sistema.

3. **Análisis de choques y expansiones supersónicas**: En sistemas de flujo compresible, se estudian las transiciones entre regímenes subsónicos y supersónicos, esenciales para el diseño de toberas o difusores.

4. **Validación numérica en CFD (Computational Fluid Dynamics)**: Los diagramas son una herramienta clave para verificar simulaciones y comparar resultados obtenidos mediante métodos numéricos.

En el contexto de la física computacional, los diagramas PTV pueden construirse implementando algoritmos basados en las ecuaciones fundamentales de la termodinámica y el flujo de fluidos. A continuación, se presenta un pseudocódigo que ilustra cómo calcular y graficar un diagrama PTV:

```
# Pseudocódigo para construir un diagrama PTV

# Definir constantes
R = 287  # Constante del gas [J/(kg*K)]

# Funciones auxiliares
def ecuacion_estado(rho, T):
    return rho * R * T  # P = \rho R T

def bernoulli(h, v):
    return h + (v**2) / 2  # h + v^2 / 2 = constante

# Inicializar variables
T_values = rango_de_temperaturas()  # Vector de temperaturas
rho_values = rango_de_densidades() # Vector de densidades

# Calcular presiones para diferentes condiciones
P_values = []
```

```
for T in T_values:
    for rho in rho_values:
        P = ecuacion_estado(rho, T)
        P_values.append((T, rho, P))

# Graficar resultados
plot_diagrama_PTV(P_values)  # Función que grafica P frente a T y \rho
```

Este esquema puede adaptarse a programas más complejos que integren simulaciones de flujo y transiciones de fase. Los códigos finales suelen desarrollarse en lenguajes como Python, MATLAB o C++ y se combinan con herramientas de visualización como Matplotlib o Paraview.

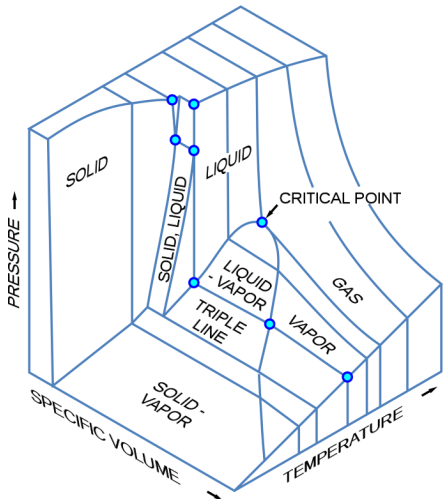

Figura 1.1: Ejemplo esquemático de un diagrama PTV mostrando regiones de flujo compresible (British Columbia Campus).

Este documento explora los principios teóricos, metodologías de construcción y casos de estudio prácticos donde los diagramas PTV proporcionan insights clave para ingenieros y científicos.

1.2 Ecuaciones de Calor

La ecuación de calor, derivada de la ley de Fourier y la conservación de energía, es una ecuación diferencial parabólica fundamental en física matemática. Su forma general para una dimensión arbitraria d es:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u + Q(\mathbf{x}, t)$$

donde:

- $u(\mathbf{x}, t)$: Distribución de temperatura (campo escalar)

- α: Difusividad térmica $[m^2/s]$
- Q: Fuente o sumidero de calor
- ∇^2: Operador Laplaciano (espacial)

En este documento se analizan soluciones numéricas de la ecuación de calor en una, dos y tres dimensiones, utilizando el método de **Crank-Nicolson** como enfoque principal por su estabilidad incondicional y su precisión de segundo orden tanto en el tiempo como en el espacio.

1.2.1 Soluciones Numéricas

La ecuación de calor puede resolverse en diferentes dimensiones dependiendo del dominio de interés:

1D: Solución en una Dimensión

Para un dominio unidimensional $u(x,t)$ sin fuentes ($Q = 0$), la ecuación discretizada mediante Crank-Nicolson es:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\alpha}{2}\left(\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}\right)$$

Reorganizando, se obtiene un sistema tridiagonal que puede resolverse eficientemente:

$$-\lambda u_{i-1}^{n+1} + (1+2\lambda)u_i^{n+1} - \lambda u_{i+1}^{n+1} = \lambda u_{i-1}^n + (1-2\lambda)u_i^n + \lambda u_{i+1}^n$$

con $\lambda = \alpha\Delta t/(2\Delta x^2)$.

2D y 3D: Soluciones en Dimensiones Superiores

En dominios bidimensionales o tridimensionales, se utiliza el enfoque *Alternating Direction Implicit* (ADI), que descompone el operador Laplaciano en pasos alternados para resolver ecuaciones en direcciones individuales:

$$(1 - \frac{\Delta t}{2}\alpha\delta_x^2)u^{n+1/2} = (1 + \frac{\Delta t}{2}\alpha\delta_y^2)u^n$$

$$(1 - \frac{\Delta t}{2}\alpha\delta_y^2)u^{n+1} = (1 + \frac{\Delta t}{2}\alpha\delta_x^2)u^{n+1/2}$$

Esto permite manejar sistemas tridimensionales complejos con eficiencia computacional.

1.3 Método de Crank-Nicolson

1.3.1 Derivación Matemática

El método de Crank-Nicolson combina el esquema explícito e implícito para lograr una precisión de segundo orden. Considerando la ecuación de calor en 1D:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Discretizamos temporalmente usando el promedio en t_n y t_{n+1}:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\alpha}{2} \left(\delta_x^2 u^{n+1} + \delta_x^2 u^n \right)$$

donde δ_x^2 es el operador de diferencias finitas centradas en el espacio:

$$\delta_x^2 u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

Este esquema conduce a un sistema lineal que puede resolverse de manera eficiente mediante el método de Thomas para matrices tridiagonales.

1.3.2 Análisis de Estabilidad

El método de Crank-Nicolson es estable incondicionalmente, como demuestra el análisis de von Neumann. El factor de amplificación G satisface:

$$|G| = \left| \frac{1 - 2\lambda \sin^2(k\Delta x/2)}{1 + 2\lambda \sin^2(k\Delta x/2)} \right| \leq 1 \quad \forall \lambda > 0$$

Esto garantiza que las soluciones permanecen acotadas independientemente de los valores de Δt y Δx.

1.3.3 Pseudocódigo

```
1. Definir los parámetros iniciales:
   - u^0: Vector inicial de temperatura.
   - alpha: Difusividad térmica.
   - L: Longitud del dominio.
   - T: Tiempo total de simulación.
   - delta_x, delta_t: Espaciado espacial y temporal.

2. Calcular lamda = alpha delta_t / (2 delta_x^2).
```

```
3. Construir las matrices:
   - Matriz A: Tridiagonal con (1+2lambda) en la diagonal principal
     y -lambda en las diagonales superior e inferior.
   - Matriz B: Tridiagonal con (1-2lambda) en la diagonal principal
     y lambda en las diagonales superior e inferior.

4. Iterar para cada paso de tiempo n:
   a. Calcular el vector b^n = B * u^n.
   b. Resolver el sistema A * u^{n+1} = b^n
      (usando el método de Thomas).
   c. Visualizar o guardar u^{n+1}.

5. Finalizar cuando se alcanza el tiempo total T.
```

1.4 Visualización de Resultados

1.4.1 Representaciones Gráficas

Los resultados de la ecuación de calor pueden visualizarse de diversas maneras:

- **1D**: La evolución temporal de la distribución de temperatura puede representarse como una superficie (x, t, u).
- **2D**: Se pueden generar mapas de color para $u(x, y)$ en tiempos seleccionados.
- **3D**: En sistemas tridimensionales, es útil graficar isosuperficies que representen valores críticos de temperatura.

Este enfoque permite explorar soluciones en una amplia variedad de dominios, ofreciendo herramientas versátiles para problemas de transferencia de calor.

Metodología

En esta sección se muestran

- Diagramas PV, VT y TP variando el valor de la constante en cada diagrama.
- Ecuación de calor para una varilla cuya fuente de calor se ubique en el centro y en una de las orillas.
- Ecuación de calor para una lámina cuya fuente de calor se ubique en el centro y la temperatura de las orillas permanezca constante.
- *Opcional:* Ecuación de calor para una esfera o un cubo.

2.1 Diagramas PTV

El programa en python se puede encontrar en el anexo PTV_diagrams.py

2.1.1 Diagrama PV

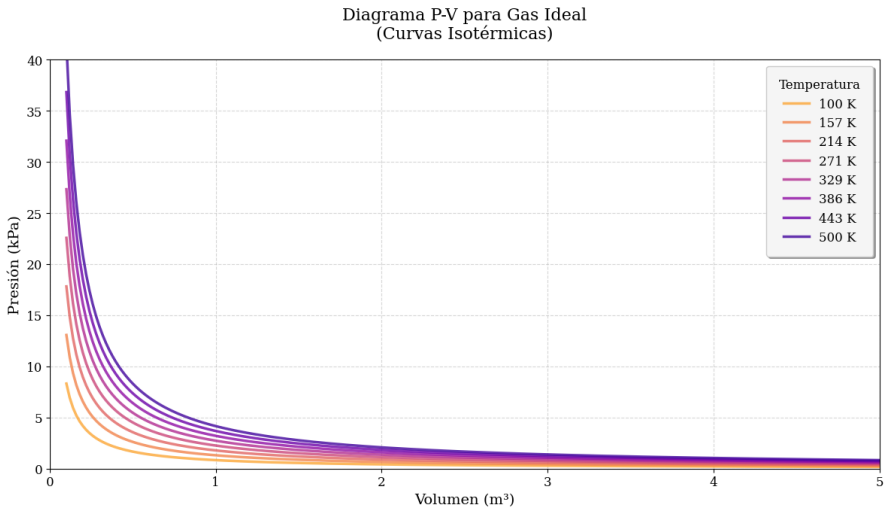

Figura 2.1: Diaframa PV a diferentes valores en la temperatura

2.1.2 Diagrama PT

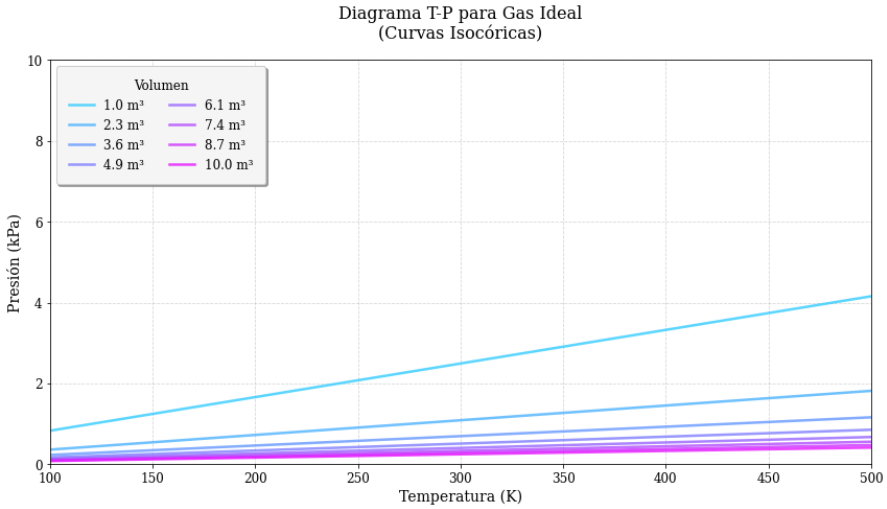

Figura 2.2: Diagrama PT a diferentes valores en el volumen.

2.1.3 Diagrama VT

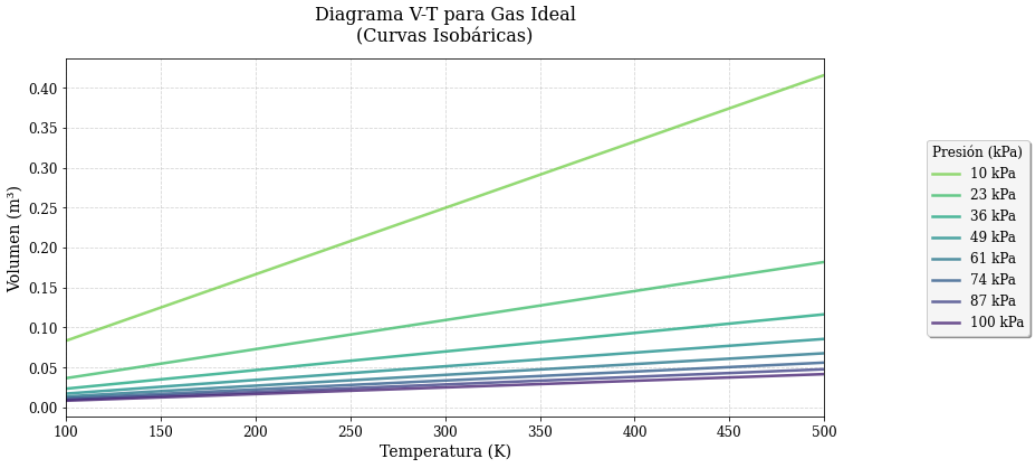

Figura 2.3: Digrama a diferentes valores de la presión.

2.2 Ecuación de calor para una varilla

2.2.1 Fuente de calor en el centro

El programa en python se puede encontrar en el anexo Varilla_centro.py

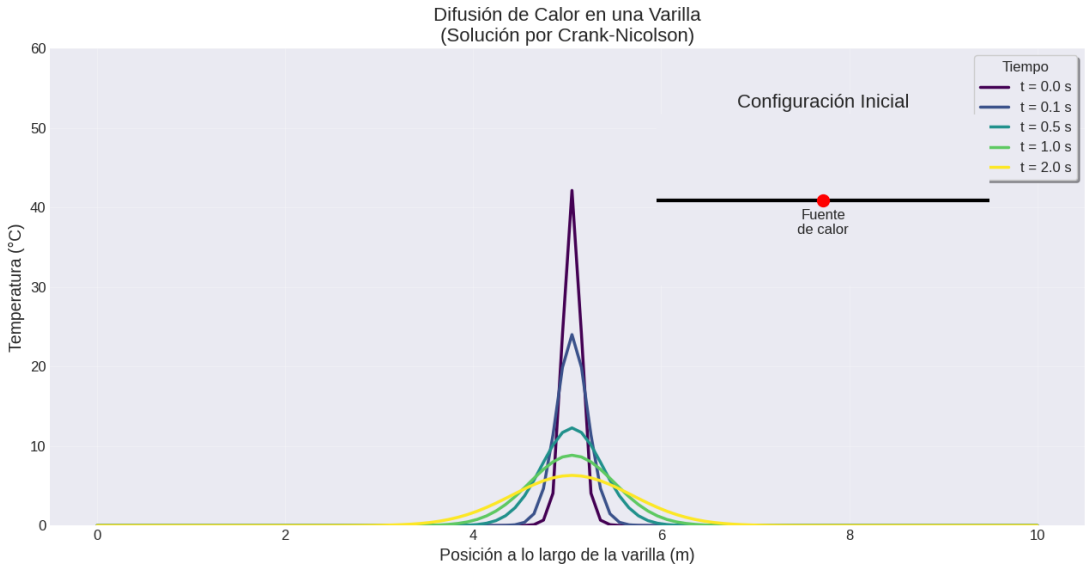

Figura 2.4: Diagrama de la temperatura en la varilla.

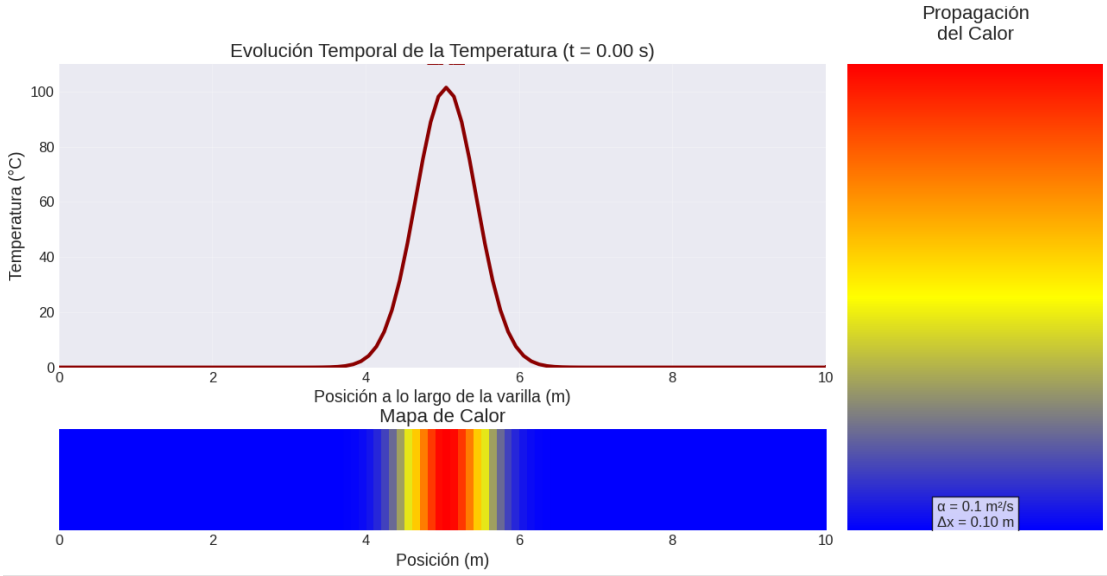

Figura 2.5: Animación creada en python sobre la temperatura en la varilla.

2.2.2 Fuente de calor en la orilla

El programa en python se puede encontrar en el anexo Varilla_orilla.py

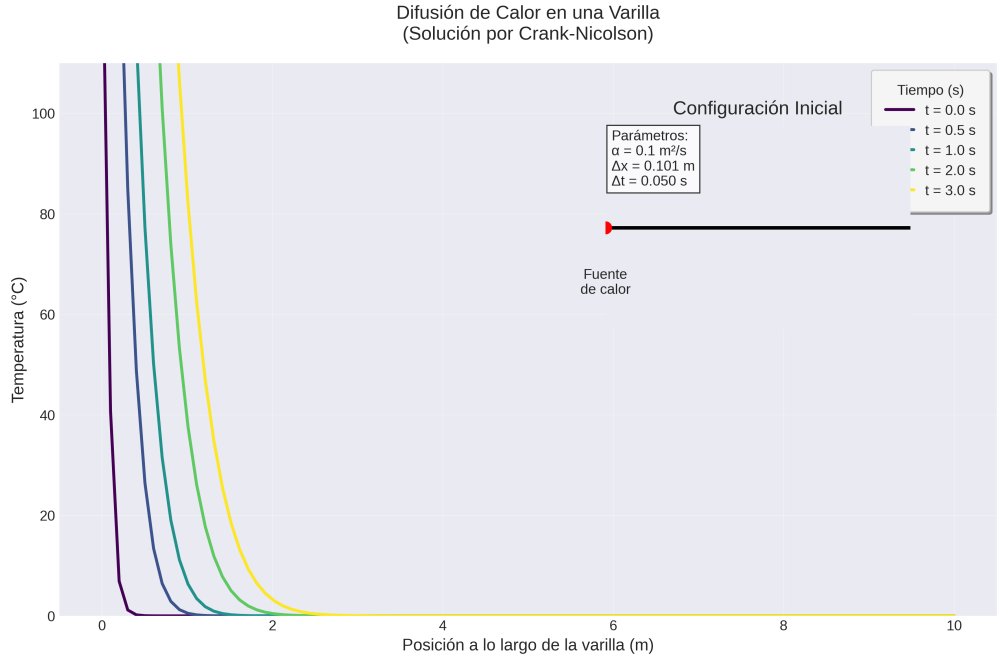

Figura 2.6: Diagrama de la temperatura en la varilla.

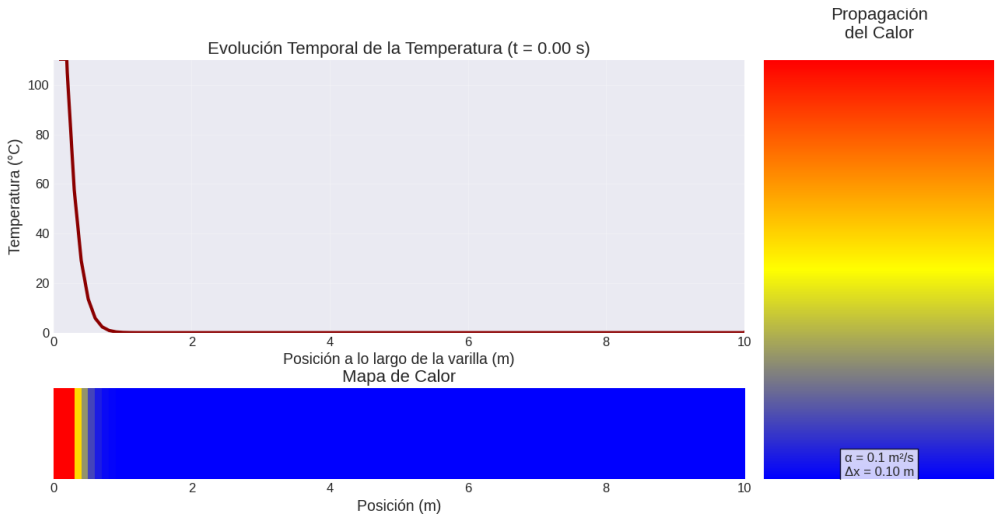

Figura 2.7: Animación creada en python sobre la temperatura en la varilla.

2.3 Ecuación de calor para una lámina

2.3.1 Fuente de calor en el centro

El programa en python se puede encontrar en el anexo Lamina_centro.py

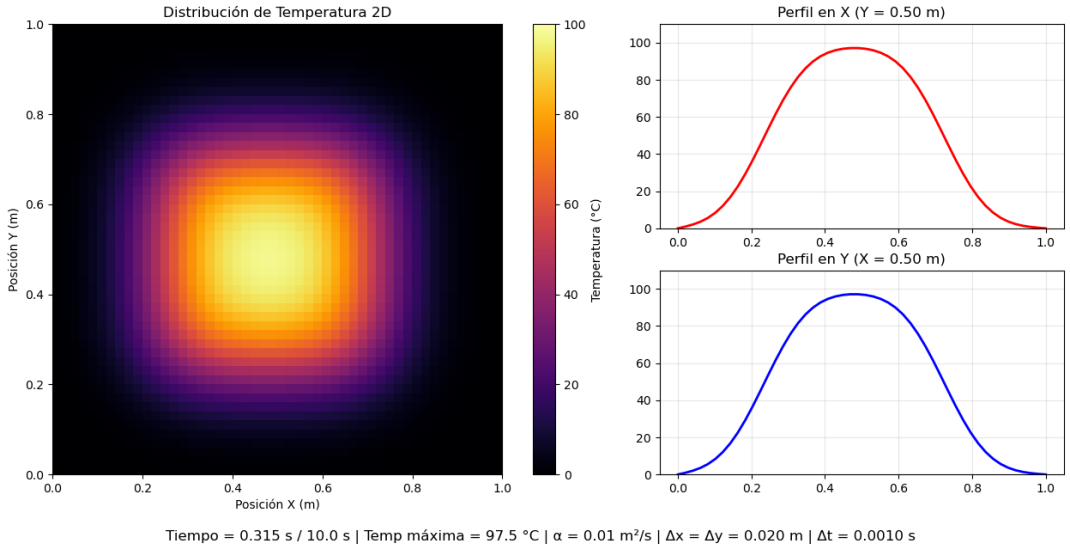

Figura 2.8: Diagrama de la temperatura en la lámina, graficando los ejes x y y.

2.3.2 Fuente de calor en la orilla

El programa en python se puede encontrar en el anexo Lamina_orilla.py

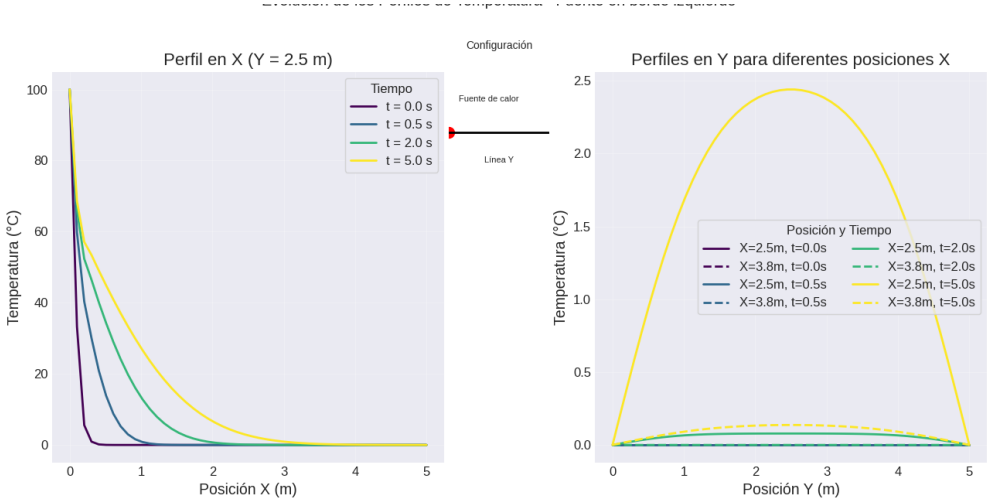

Figura 2.9: Diagrama de la temperatura en la lámina graficando los ejes x y y.

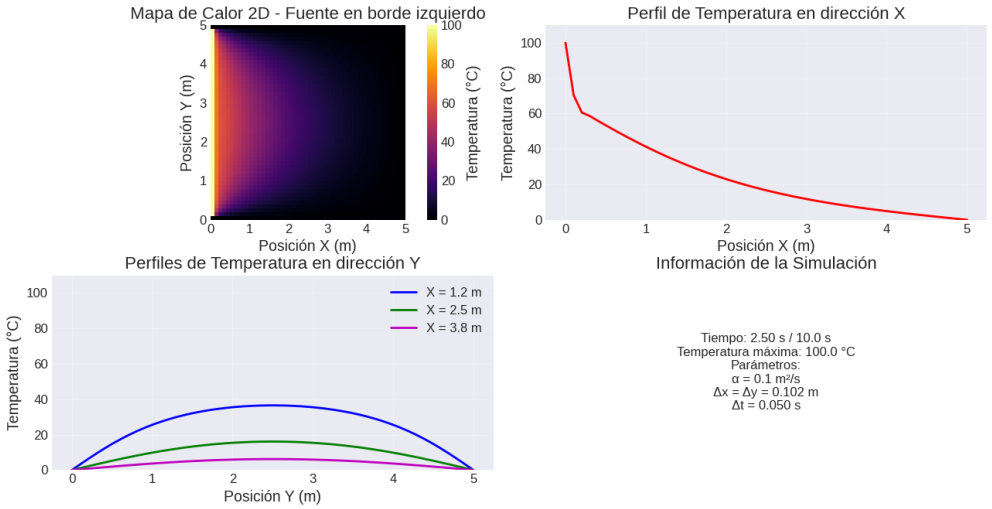

Figura 2.10: Animación creada en python sobre la temperatura en la lámina.

2.4 Cubo

El programa en python se puede encontrar en el anexo Cubo.py

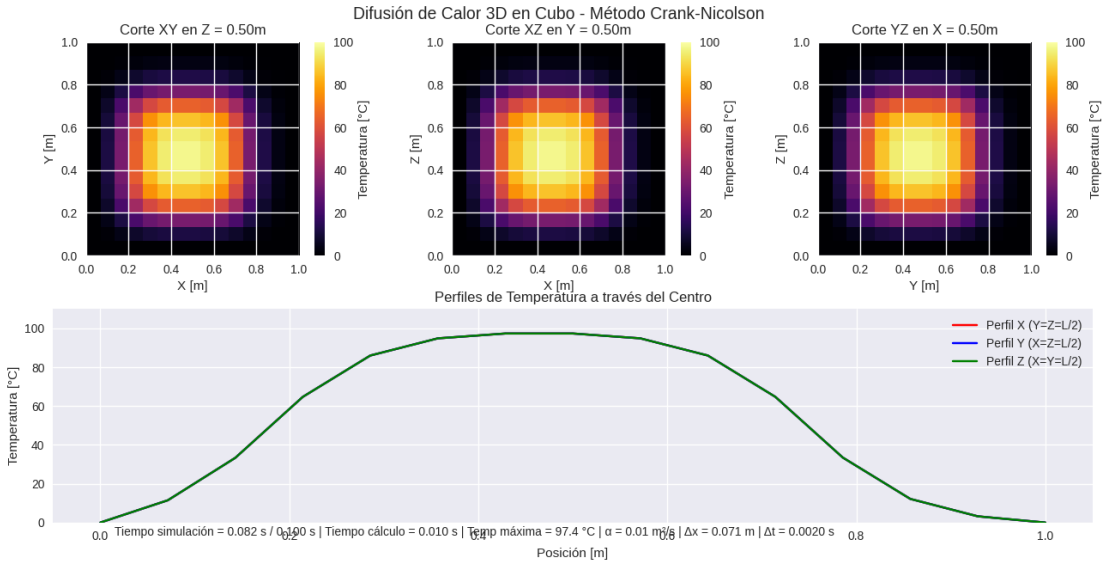

Figura 2.11: Animación creada en python sobre la temperatura en el cubo con cortes transversales en cada eje.

Conclusión

A través de este trabajo logré comprender distintos métodos aplicados a una situación que se aproxima más a un contexto real, específicamente en el ámbito de la simulación numérica. En particular, estudié la ecuación de calor en una, dos y tres dimensiones, y analicé cómo puede resolverse computacionalmente. Este proceso me permitió identificar tanto las limitaciones de algunos métodos como las ventajas de otros, destacando especialmente la eficacia del método de Crank-Nicolson para este tipo de problemas.

Una vez entendido por qué este método representa una de las mejores opciones, abordé su implementación computacional. Aunque representó un desafío, fue especialmente valioso poder realizar una simulación que no solo funcionara desde el punto de vista numérico, sino que también reflejara correctamente el comportamiento físico esperado. Esta experiencia me permitió fortalecer mi intuición física y mi capacidad para interpretar los resultados de una simulación desde una perspectiva tanto matemática como computacional.

Referencias

- ChatGPT. (2025). OpenAI. https://chat.openai.com/
- Método Crank-Nicolson: Crank, J., & Nicolson, P. (1947). .A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type."Proc. Camb. Phil. Soc., 43(1), 50-67. DOI: 10.1017/S0305004100023197
- Deepseek.
- British Columbia Campus. (s.f.). Phase diagrams. En Thermodynamics and Chemistry. BCcampus. https://pressbooks.bccampus.ca/thermo1/chapter/phase-diagrams/

Anexos

Los códigos se encuentran en el link https://github.com/cinthia-bao/Fisica-Computacional-.git en la carpeta **TAREA 9**.

4.1 PTV_diagrams.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 24 11:28:29 2025

@author: ale
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.gridspec import GridSpec

# Configuracin global de estilo
rcParams.update({
    'font.size': 12,
    'font.family': 'serif',
    'axes.labelsize': 14,
    'axes.titlesize': 16,
    'xtick.labelsize': 12,
    'ytick.labelsize': 12,
    'legend.fontsize': 12,
    'axes.grid': True,
    'grid.alpha': 0.3
```

```
})

# Parmetros fsicos
n = 1.0 # Moles de gas
R = 8.314 # Constante de los gases ideales (J/molK)

# Rangos de valores
V = np.linspace(0.1, 10, 500) # Volumen (m)
T = np.linspace(100, 500, 500) # Temperatura (K)
P_kpa = np.linspace(1, 100, 10) # Presin (kPa)

# ==========================================
# 1. Diagrama P-V (Isotermas)
# ==========================================
plt.figure(figsize=(12, 7), dpi=100)
temperaturas = np.linspace(100, 500, 8) # 8 isotermas
colormap = plt.cm.plasma_r(np.linspace(0.2, 0.9, len(temperaturas)))

for temp, color in zip(temperaturas, colormap):
    P_ideal = (n * R * temp) / V # Ley de los gases ideales (en Pa)
    plt.plot(V, P_ideal/1e3, color=color, lw=2.5,
             label=f'{temp:.0f} K', alpha=0.8)

plt.title("Diagrama P-V para Gas Ideal\n(Curvas Isotrmicas)", pad=20)
plt.xlabel("Volumen (m)")
plt.ylabel("Presin (kPa)")
plt.xlim(0, 10)
plt.ylim(0, 450)
plt.grid(True, linestyle='--', alpha=0.5)

# Leyenda mejorada
legend = plt.legend(title="Temperatura", frameon=True,
                    shadow=True, borderpad=1)
legend.get_frame().set_facecolor('#f5f5f5')

# Anotacin fsica
plt.annotate(r'$PV = nRT$', xy=(6, 400), fontsize=18,
             bbox=dict(boxstyle="round", fc="#f5f5f5", ec="0.5", alpha=0.9)
                 )
```

```
plt.tight_layout()
plt.show()

# ================================================
# 2. Diagrama V-T (Isobaras)
# ================================================
fig = plt.figure(figsize=(14, 6))
gs = GridSpec(1, 2, width_ratios=[3, 1])

# Grfica principal
ax0 = plt.subplot(gs[0])
presiones = np.linspace(10, 100, 8) # 8 isobaras
colormap = plt.cm.viridis_r(np.linspace(0.2, 0.9, len(presiones)))

for pressure, color in zip(presiones, colormap):
    V_ideal = (n * R * T) / (pressure * 1e3) # Convertir kPa a Pa
    ax0.plot(T, V_ideal, color=color, lw=2.5,
           label=f'{pressure:.0f} kPa', alpha=0.8)

ax0.set_title("Diagrama V-T para Gas Ideal\n(Curvas Isobricas)", pad=15)
ax0.set_xlabel("Temperatura (K)")
ax0.set_ylabel("Volumen (m)")
ax0.set_xlim(100, 500)
ax0.grid(True, linestyle='--', alpha=0.5)

# Leyenda como subplot
ax1 = plt.subplot(gs[1])
ax1.axis('off')
legend = ax1.legend(*ax0.get_legend_handles_labels(),
                title="Presin (kPa)", loc='center',
                frameon=True, shadow=True)
legend.get_frame().set_facecolor('#f5f5f5')

# Anotacin cientfica
ax0.annotate(r'$\frac{V}{T} = \frac{nR}{P} = \text{const.}$',
           xy=(350, 0.8), fontsize=16,
           bbox=dict(boxstyle="round", fc="#f5f5f5", ec="0.5", alpha=0.9)
               )

plt.tight_layout()
```

```
plt.show()

# ============================================
# 3. Diagrama T-P (Isocoras)
# ============================================
plt.figure(figsize=(12, 7))
volumenes = np.linspace(1, 10, 8) # 8 isocoras
colormap = plt.cm.cool(np.linspace(0.2, 0.9, len(volumenes)))

for volume, color in zip(volumenes, colormap):
    P_ideal = (n * R * T) / volume # en Pa
    plt.plot(T, P_ideal/1e3, color=color, lw=2.5,
             label=f'{volume:.1f} m', alpha=0.8)

plt.title("Diagrama T-P para Gas Ideal\n(Curvas Isocricas)", pad=20)
plt.xlabel("Temperatura (K)")
plt.ylabel("Presin (kPa)")
plt.xlim(100, 500)
plt.ylim(0, 450)

# Leyenda con estilo
legend = plt.legend(title="Volumen", ncol=2, frameon=True,
                    shadow=True, borderpad=1, loc='upper left')
legend.get_frame().set_facecolor('#f5f5f5')

# Ecuacin destacada
plt.annotate(r'$\frac{P}{T} = \frac{nR}{V} = \text{const.}$',
             xy=(120, 400), fontsize=16,
             bbox=dict(boxstyle="round", fc="#f5f5f5", ec="0.5", alpha=0.9)
                 )

plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

4.2 Varilla_centro.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 24 12:11:24 2025

@author: ale
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.gridspec import GridSpec

# Configuracin de estilo profesional
plt.style.use('seaborn-darkgrid')
plt.rcParams.update({
    'font.size': 12,
    'figure.facecolor': 'white',
    'axes.grid': True,
    'grid.alpha': 0.3,
    'axes.titlesize': 16,
    'axes.labelsize': 14
})

# ==========================================
# 1. ANIMACIN MEJORADA DE LA VARILLA
# ==========================================

# Parmetros fsicos
L = 10.0 # Longitud de la varilla (m)
Nx = 100 # Aumentamos la resolucin espacial
Tmax = 5.0 # Tiempo mximo (s) - Ms tiempo para ver evolucin
alpha = 0.1 # Reducimos difusividad para mejor visualizacin

# Discretizacin
dx = L / (Nx - 1)
dt = 0.05 # Paso de tiempo
```

```
Nt = int(Tmax / dt)

# Coeficiente de Crank-Nicolson
r = alpha * dt / (2 * dx**2)

# Inicializacin de la malla de temperatura
u = np.zeros(Nx)
u[Nx // 2] = 1000 # Fuente de calor en el centro

# Configuracin de matrices tridiagonales
main_diag = np.ones(Nx) * (1 + 2*r)
off_diag = np.ones(Nx-1) * (-r)
A = np.diag(main_diag) + np.diag(off_diag, k=1) + np.diag(off_diag, k=-1)
B_main = np.ones(Nx) * (1 - 2*r)
B = np.diag(B_main) + np.diag(off_diag*(-1), k=1) + np.diag(off_diag*(-1),
    k=-1)

# Condiciones de frontera fijas
A[0, 0] = A[-1, -1] = 1
A[0, 1] = A[-1, -2] = 0
B[0, 0] = B[-1, -1] = 1
B[0, 1] = B[-1, -2] = 0

# Crear figura con dos subplots
fig = plt.figure(figsize=(14, 8))
gs = GridSpec(2, 2, width_ratios=[3, 1], height_ratios=[3, 1])
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[:, 1])

# Configuracin del plot principal
x = np.linspace(0, L, Nx)
line, = ax1.plot(x, u, lw=3, color='darkred')
ax1.set_ylim(0, 110)
ax1.set_xlim(0, L)
ax1.set_xlabel("Posicin a lo largo de la varilla (m)")
ax1.set_ylabel("Temperatura (C)")
ax1.set_title("Evolucin Temporal de la Temperatura")

# Configuracin del plot inferior (gradiente)
```

```
cmap = LinearSegmentedColormap.from_list('heatmap', ['blue', 'yellow', '
    red'])
gradient = np.vstack((u, u))
img = ax2.imshow(gradient, cmap=cmap, aspect='auto', extent=[0, L, 0, 1],
    vmin=0, vmax=100)
ax2.set_xlabel("Posicin (m)")
ax2.set_yticks([])
ax2.set_title("Mapa de Calor")

# Configuracin del plot lateral (leyenda de color)
cbar = fig.colorbar(img, cax=ax3)
cbar.set_label('Temperatura (C)')
ax3.set_title("Propagacin\ndel Calor", pad=20)
ax3.axis('off')

# Texto informativo
info_text = ax3.text(0.5, 0.1,
                     f" = {alpha} m/s\nx = {dx:.2f} m\nt = {dt:.2f} s",
                     ha='center', va='center',
                     bbox=dict(facecolor='white', alpha=0.8))

# Funcin de animacin mejorada
def animate(t):
    global u
    b = B @ u
    u = np.linalg.solve(A, b)

    # Actualizar lnea principal
    line.set_ydata(u)

    # Actualizar mapa de calor
    gradient = np.vstack((u, u))
    img.set_array(gradient)

    # Actualizar ttulo con tiempo
    ax1.set_title(f"Evolucin Temporal de la Temperatura (t = {t*dt:.2f} s)"
        )

    # Actualizar informacin
    info_text.set_text(f" = {alpha} m/s\nx = {dx:.2f} m\nt = {dt:.2f} s\nt
```

```
        = {t*dt:.2f} s")

    return line, img, info_text

# Crear animacin
ani = FuncAnimation(fig, animate, frames=Nt, interval=50, blit=True)

plt.tight_layout()
plt.show()

# ============================================
# 2. IMAGEN ESTTICA DEL PROCESO (para exportar)
# ============================================

# Simular algunos pasos para visualizacin
u_static = np.zeros(Nx)
u_static[Nx // 2] = 100
snapshots = []
times = [0, 0.1, 0.5, 1, 2, 5] # Tiempos especficos para capturas

for t in range(int(5/dt)):
    b = B @ u_static
    u_static = np.linalg.solve(A, b)
    if t*dt in times:
        snapshots.append((t*dt, u_static.copy()))

# Crear figura esttica
plt.figure(figsize=(12, 8))
colors = plt.cm.viridis(np.linspace(0, 1, len(snapshots)))

for (time, temp), color in zip(snapshots, colors):
    plt.plot(x, temp, lw=2.5, color=color, label=f't = {time:.1f} s')

plt.title("Difusin de Calor en una Varilla\n(Solucin por Crank-Nicolson)")
plt.xlabel("Posicin a lo largo de la varilla (m)")
plt.ylabel("Temperatura (C)")
plt.ylim(0, 60)
plt.grid(True, alpha=0.3)
plt.legend(title="Tiempo", frameon=True, shadow=True)
```

```
# Aadir diagrama esquemtico
inset_ax = plt.axes([0.6, 0.5, 0.3, 0.3])
inset_ax.set_xlim(0, L)
inset_ax.set_ylim(0, 1)
inset_ax.set_xticks([])
inset_ax.set_yticks([])

# Dibujar varilla
inset_ax.plot([0, L], [0.5, 0.5], 'k-', lw=3)
inset_ax.plot(L/2, 0.5, 'ro', markersize=10) # Fuente de calor
inset_ax.text(L/2, 0.3, 'Fuente\nde calor', ha='center')
inset_ax.set_title("Configuracin Inicial")

plt.tight_layout()
plt.savefig('difusion_calor_varilla.png', dpi=300)
plt.show()
```

4.3 Varilla_orilla.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 24 12:11:24 2025

@author: ale
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.gridspec import GridSpec

# Configuracin de estilo profesional
plt.style.use('seaborn-darkgrid')
plt.rcParams.update({
    'font.size': 12,
    'figure.facecolor': 'white',
    'axes.grid': True,
    'grid.alpha': 0.3,
    'axes.titlesize': 16,
    'axes.labelsize': 14
})

# Parmetros fsicos
L = 10.0 # Longitud de la varilla (m)
Nx = 100 # Aumentamos la resolucin espacial
Tmax = 5.0 # Tiempo mximo (s)
alpha = 0.1 # Difusividad trmica (m/s)

# Discretizacin
dx = L / (Nx - 1)
dt = 0.05 # Paso de tiempo
Nt = int(Tmax / dt)

# Coeficiente de Crank-Nicolson
r = alpha * dt / (2 * dx**2)
```

```
# Inicializacin de la malla de temperatura
u = np.zeros(Nx)
u[0] = 100 # Fuente de calor en el extremo izquierdo

# Configuracin de matrices tridiagonales
main_diag = np.ones(Nx) * (1 + 2 * r)
off_diag = np.ones(Nx - 1) * (-r)
A = np.diag(main_diag) + np.diag(off_diag, k=1) + np.diag(off_diag, k=-1)
B_main = np.ones(Nx) * (1 - 2 * r)
B = np.diag(B_main) + np.diag(off_diag * -1, k=1) + np.diag(off_diag * -1,
    k=-1)

# Condiciones de frontera
A[0, 0] = A[-1, -1] = 1
A[0, 1] = A[-1, -2] = -1
B[0, 0] = B[-1, -1] = 1
B[0, 1] = B[-1, -2] = -1

# ============================================
# 1. ANIMACIN MEJORADA
# ============================================

# Crear figura con dos subplots
fig = plt.figure(figsize=(14, 8))
gs = GridSpec(2, 2, width_ratios=[3, 1], height_ratios=[3, 1])
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[:, 1])

# Configuracin del plot principal
x = np.linspace(0, L, Nx)
line, = ax1.plot(x, u, lw=3, color='darkred')
ax1.set_ylim(0, 110)
ax1.set_xlim(0, L)
ax1.set_xlabel("Posicin a lo largo de la varilla (m)")
ax1.set_ylabel("Temperatura (C)")
ax1.set_title("Evolucin Temporal de la Temperatura")

# Configuracin del plot inferior (gradiente)
cmap = LinearSegmentedColormap.from_list('heatmap', ['blue', 'yellow', '
```

```
    red'])
gradient = np.vstack((u, u))
img = ax2.imshow(gradient, cmap=cmap, aspect='auto', extent=[0, L, 0, 1],
    vmin=0, vmax=100)
ax2.set_xlabel("Posicin (m)")
ax2.set_yticks([])
ax2.set_title("Mapa de Calor")

# Configuracin del plot lateral (leyenda de color)
cbar = fig.colorbar(img, cax=ax3)
cbar.set_label('Temperatura (C)')
ax3.set_title("Propagacin\ndel Calor", pad=20)
ax3.axis('off')

# Texto informativo
info_text = ax3.text(0.5, 0.1,
                     f" = {alpha} m/s\nx = {dx:.2f} m\nt = {dt:.2f} s",
                     ha='center', va='center',
                     bbox=dict(facecolor='white', alpha=0.8))

# Funcin de animacin
def animate(t):
    global u
    b = B @ u
    u = np.linalg.solve(A, b)

    # Actualizar lnea principal
    line.set_ydata(u)

    # Actualizar mapa de calor
    gradient = np.vstack((u, u))
    img.set_array(gradient)

    # Actualizar ttulo con tiempo
    ax1.set_title(f"Evolucin Temporal de la Temperatura (t = {t*dt:.2f} s)"
        )

    # Actualizar informacin
    info_text.set_text(f" = {alpha} m/s\nx = {dx:.2f} m\nt = {dt:.2f} s\nt
        = {t*dt:.2f} s")
```

```
    return line, img, info_text

# Crear animacin
ani = FuncAnimation(fig, animate, frames=Nt, interval=50, blit=True)

plt.tight_layout()
plt.show()

# ============================================
# 2. IMAGEN ESTTICA FINAL
# ============================================

# Simular para tiempos especficos
u_static = np.zeros(Nx)
u_static[0] = 100 # Misma condicin inicial
snapshot_times = [0, 0.5, 1, 2, 3, 5] # Tiempos para capturas
snapshots = []

for t in range(int(Tmax/dt)):
    b = B @ u_static
    u_static = np.linalg.solve(A, b)
    if t*dt in snapshot_times:
        snapshots.append((t*dt, u_static.copy()))

# Crear figura esttica
plt.figure(figsize=(12, 8))
colors = plt.cm.viridis(np.linspace(0, 1, len(snapshots)))

for (time, temp), color in zip(snapshots, colors):
    plt.plot(x, temp, lw=2.5, color=color, label=f't = {time:.1f} s')

plt.title("Difusin de Calor en una Varilla\n(Solucin por Crank-Nicolson)",
     pad=20)
plt.xlabel("Posicin a lo largo de la varilla (m)")
plt.ylabel("Temperatura (C)")
plt.ylim(0, 110)
plt.grid(True, alpha=0.3)

# Leyenda mejorada
```

```
legend = plt.legend(title="Tiempo (s)", frameon=True, shadow=True,
                borderpad=1, loc='upper right')
legend.get_frame().set_facecolor('#f5f5f5')

# Aadir diagrama esquemtico
inset_ax = plt.axes([0.6, 0.5, 0.3, 0.3])
inset_ax.set_xlim(0, L)
inset_ax.set_ylim(0, 1)
inset_ax.set_xticks([])
inset_ax.set_yticks([])

# Dibujar varilla y fuente de calor
inset_ax.plot([0, L], [0.5, 0.5], 'k-', lw=3)
inset_ax.plot(0, 0.5, 'ro', markersize=10) # Fuente de calor en extremo
    izquierdo
inset_ax.text(0, 0.3, 'Fuente\nde calor', ha='center', va='top')
inset_ax.set_title("Configuracin Inicial", pad=10)

# Aadir informacin tcnica
plt.text(0.02, 0.98,
        f"Parmetros:\n = {alpha} m/s\nx = {dx:.3f} m\nt = {dt:.3f} s",
        transform=plt.gca().transAxes,
        ha='left', va='top',
        bbox=dict(facecolor='white', alpha=0.8))

plt.tight_layout()
plt.savefig('difusion_calor_varilla_estatico.png', dpi=300, bbox_inches='
    tight')
plt.show()
```

4.4 Lamina_centro.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Solucin 2D de la ecuacin de calor con Crank-Nicolson
Con grficos de perfil de temperatura como en el caso 1D
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.gridspec import GridSpec
from scipy.sparse import diags, eye
from scipy.sparse.linalg import spsolve

# Configuracin de estilo
plt.style.use('seaborn-darkgrid')
plt.rcParams.update({
    'font.size': 12,
    'figure.facecolor': 'white',
    'axes.grid': True,
    'grid.alpha': 0.3,
    'axes.titlesize': 16,
    'axes.labelsize': 14
})

# ============================================
# 1. PARMETROS Y DISCRETIZACIN
# ============================================

# Parmetros fsicos
L = 5.0 # Longitud de cada lado (m)
N = 50 # Nmero de nodos por dimensin
Tmax = 10.0 # Tiempo mximo (s)
alpha = 0.1 # Difusividad trmica (m/s)
source_temp = 1000.0 # Temperatura de la fuente (C)

# Discretizacin
dx = dy = L / (N - 1)
```

```
dt = 0.05
Nt = int(Tmax / dt)

# Coeficientes Crank-Nicolson
r = alpha * dt / (2 * dx**2)

# ==========================================
# 2. IMPLEMENTACIN NUMRICA
# ==========================================

# Inicializacin
u = np.zeros((N, N))
u[N//2, N//2] = source_temp # Fuente en el centro

# Matrices para ADI
def build_matrices(size, r):
    main_diag = (1 + 2*r) * np.ones(size)
    off_diag = -r * np.ones(size-1)
    A = diags([off_diag, main_diag, off_diag], [-1, 0, 1])
    B = diags([r*np.ones(size-1), (1-2*r)*np.ones(size), r*np.ones(size-1)
        ], [-1, 0, 1])
    return A, B

A_x, B_x = build_matrices(N, r)
A_y, B_y = build_matrices(N, r)

def update_solution(u):
    # Primera mitad (implcito en x)
    u_half = np.zeros_like(u)
    for j in range(1, N-1):
        b = B_x @ u[:,j] + r * (u[:,j+1] + u[:,j-1] - 2*u[:,j])
        u_half[:,j] = spsolve(A_x, b)

    # Segunda mitad (implcito en y)
    u_new = np.zeros_like(u)
    for i in range(1, N-1):
        b = B_y @ u_half[i,:] + r * (u_half[i+1,:] + u_half[i-1,:] - 2*
            u_half[i,:])
        u_new[i,:] = spsolve(A_y, b)
```

```
    # Condiciones de frontera
    u_new[0,:] = u_new[-1,:] = u_new[:,0] = u_new[:,-1] = 0
    return u_new

# ============================================
# 3. ANIMACIN MEJORADA CON PERFILES
# ============================================

fig = plt.figure(figsize=(14, 8))
gs = GridSpec(2, 2, width_ratios=[1, 1], height_ratios=[1, 1])
ax1 = fig.add_subplot(gs[0, 0]) # Mapa de calor
ax2 = fig.add_subplot(gs[0, 1]) # Perfil en x
ax3 = fig.add_subplot(gs[1, 0]) # Perfil en y
ax4 = fig.add_subplot(gs[1, 1]) # Informacin

# Configuracin de los ejes
x = y = np.linspace(0, L, N)
X, Y = np.meshgrid(x, y)

# Mapa de calor
img = ax1.imshow(u.T, cmap='inferno', extent=[0, L, 0, L], origin='lower',
     vmin=0, vmax=source_temp/5)
plt.colorbar(img, ax=ax1, label='Temperatura (C)')
ax1.set_title("Mapa de Calor 2D")
ax1.set_xlabel("Posicin X (m)")
ax1.set_ylabel("Posicin Y (m)")

# Perfil en X (y = L/2)
line_x, = ax2.plot(x, u[:, N//2], 'r-', lw=2, label='Perfil en X')
ax2.set_title(f"Perfil de Temperatura en X (Y = {L/2:.1f} m)")
ax2.set_xlabel("Posicin X (m)")
ax2.set_ylabel("Temperatura (C)")
ax2.set_ylim(0, 80)
ax2.grid(True)

# Perfil en Y (x = L/2)
line_y, = ax3.plot(y, u[N//2, :], 'b-', lw=2, label='Perfil en Y')
ax3.set_title(f"Perfil de Temperatura en Y (X = {L/2:.1f} m)")
ax3.set_xlabel("Posicin Y (m)")
ax3.set_ylabel("Temperatura (C)")
```

```
ax3.set_ylim(0,80)
ax3.grid(True)

# Informacin
ax4.axis('off')
info_text = ax4.text(0.5, 0.5, "", transform=ax4.transAxes, ha='center',
    va='center')
ax4.set_title("Informacin de la Simulacin")

def animate(t):
    global u
    u = update_solution(u)

    # Actualizar mapa de calor
    img.set_array(u.T)
    img.set_clim(vmin=0, vmax=np.max(u))

    # Actualizar perfiles
    line_x.set_ydata(u[:, N//2])
    line_y.set_ydata(u[N//2, :])

    # Actualizar informacin
    info = f"Tiempo: {t*dt:.2f} s / {Tmax:.1f} s\n"
    info += f"Temperatura mxima: {np.max(u):.1f} C\n"
    info += f"Parmetros:\n = {alpha} m/s\nx = y = {dx:.3f} m\nt = {dt:.3f}
        s"
    info_text.set_text(info)

    return img, line_x, line_y, info_text

ani = FuncAnimation(fig, animate, frames=Nt, interval=50, blit=True)
plt.tight_layout()
plt.show()

# ============================================
# 4. GRFICOS ESTTICOS DE PERFILES
# ============================================

# Simular para tiempos especficos
snapshot_times = [0, 0.5, 2, 5, 10]
```

```
snapshots = []
u_static = np.zeros((N, N))
u_static[N//2, N//2] = source_temp

for t in range(int(Tmax/dt)):
    u_static = update_solution(u_static)
    if t*dt in snapshot_times:
        snapshots.append((t*dt, u_static.copy()))

# Crear figura esttica
fig_static, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
fig_static.suptitle("Evolucin de los Perfiles de Temperatura", y=1.02)

# Perfil en X
colors = plt.cm.viridis(np.linspace(0, 1, len(snapshots)))
for (time, temp), color in zip(snapshots, colors):
    ax1.plot(x, temp[:, N//2], color=color, lw=2, label=f't = {time:.1f} s'
        )

ax1.set_title(f"Perfil en X (Y = {L/2:.1f} m)")
ax1.set_xlabel("Posicin X (m)")
ax1.set_ylabel("Temperatura (C)")
ax1.legend(title="Tiempo", frameon=True)
ax1.grid(True)

# Perfil en Y
for (time, temp), color in zip(snapshots, colors):
    ax2.plot(y, temp[N//2, :], color=color, lw=2, label=f't = {time:.1f} s'
        )

ax2.set_title(f"Perfil en Y (X = {L/2:.1f} m)")
ax2.set_xlabel("Posicin Y (m)")
ax2.set_ylabel("Temperatura (C)")
ax2.legend(title="Tiempo", frameon=True)
ax2.grid(True)

# Diagrama esquemtico
ax_sketch = fig_static.add_axes([0.45, 0.6, 0.1, 0.3])
ax_sketch.set_xlim(0, 1)
ax_sketch.set_ylim(0, 1)
```

```
ax_sketch.axis('off')
ax_sketch.plot([0.5], [0.5], 'ro', markersize=10)
ax_sketch.plot([0, 1], [0.5, 0.5], 'k--', alpha=0.5)
ax_sketch.plot([0.5, 0.5], [0, 1], 'k--', alpha=0.5)
ax_sketch.text(0.5, 0.7, "Lnea X", ha='center')
ax_sketch.text(0.7, 0.5, "Lnea Y", va='center')
ax_sketch.set_title("Cortes", fontsize=10)

plt.tight_layout()
plt.savefig('perfiles_temperatura_2d.png', dpi=300, bbox_inches='tight')
plt.show()
```

4.5 Lamina_orilla.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Solucin 2D de la ecuacin de calor con Crank-Nicolson
Fuente de calor en el borde izquierdo (x=0) - Versin corregida
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.gridspec import GridSpec
from scipy.sparse import diags, eye, lil_matrix
from scipy.sparse.linalg import spsolve

# Configuracin de estilo
plt.style.use('seaborn-darkgrid')
plt.rcParams.update({
    'font.size': 12,
    'figure.facecolor': 'white',
    'axes.grid': False, # Cambiado a False para evitar warning
    'grid.alpha': 0.3,
    'axes.titlesize': 16,
    'axes.labelsize': 14
})

# ==========================================
# 1. PARMETROS Y DISCRETIZACIN
# ==========================================

# Parmetros fsicos
L = 5.0 # Longitud de cada lado (m)
N = 50 # Nmero de nodos por dimensin
Tmax = 10.0 # Tiempo mximo (s)
alpha = 0.1 # Difusividad trmica (m/s)
source_temp = 100.0 # Temperatura de la fuente (C)

# Discretizacin
dx = dy = L / (N - 1)
```

```
dt = 0.05
Nt = int(Tmax / dt)

# Coeficientes Crank-Nicolson
r = alpha * dt / (2 * dx**2)

# ==========================================
# 2. IMPLEMENTACIN NUMRICA CORREGIDA
# ==========================================

# Inicializacin - Fuente en el borde izquierdo (x=0)
u = np.zeros((N, N))
u[0, :] = source_temp # Toda la columna x=0 a temperatura fuente

# Matrices para ADI - Versin corregida
def build_matrices(size, r):
    # Usamos lil_matrix para facilitar el slicing
    A = lil_matrix((size, size))
    B = lil_matrix((size, size))

    for i in range(size):
        if i > 0:
            A[i, i-1] = -r
            B[i, i-1] = r
        A[i, i] = 1 + 2*r
        B[i, i] = 1 - 2*r
        if i < size-1:
            A[i, i+1] = -r
            B[i, i+1] = r

    # Convertir a formato CSR para mejor eficiencia en solve
    return A.tocsr(), B.tocsr()

A_x, B_x = build_matrices(N, r)
A_y, B_y = build_matrices(N, r)

def update_solution(u):
    # Primera mitad (implcito en x)
    u_half = u.copy()
```

```
    for j in range(1, N-1):
        b = B_x @ u[:,j] + r * (u[:,j+1] + u[:,j-1] - 2*u[:,j])
        # Resolver solo para puntos internos (excluyendo bordes)
        u_half[1:-1,j] = spsolve(A_x[1:-1,1:-1], b[1:-1])

    # Segunda mitad (implcito en y)
    u_new = u_half.copy()

    for i in range(1, N-1):
        b = B_y @ u_half[i,:] + r * (u_half[i+1,:] + u_half[i-1,:] - 2*
            u_half[i,:])
        u_new[i,1:-1] = spsolve(A_y[1:-1,1:-1], b[1:-1])

    # Mantener condiciones de frontera:
    u_new[0, :] = source_temp # Fuente de calor permanente
    u_new[-1,:] = 0 # Borde derecho
    u_new[:, 0] = 0 # Borde inferior
    u_new[:,-1] = 0 # Borde superior

    return u_new

# ===========================================
# 3. ANIMACIN MEJORADA CON PERFILES
# ===========================================

fig = plt.figure(figsize=(14, 8))
gs = GridSpec(2, 2, width_ratios=[1, 1], height_ratios=[1, 1])
ax1 = fig.add_subplot(gs[0, 0]) # Mapa de calor
ax2 = fig.add_subplot(gs[0, 1]) # Perfil en x
ax3 = fig.add_subplot(gs[1, 0]) # Perfil en y
ax4 = fig.add_subplot(gs[1, 1]) # Informacin

# Configuracin de los ejes
x = y = np.linspace(0, L, N)

# Mapa de calor
img = ax1.imshow(u.T, cmap='inferno', extent=[0, L, 0, L], origin='lower',
     vmin=0, vmax=source_temp)
cbar = plt.colorbar(img, ax=ax1)
cbar.set_label('Temperatura (C)')
```

```
ax1.set_title("Mapa de Calor 2D - Fuente en borde izquierdo")
ax1.set_xlabel("Posicin X (m)")
ax1.set_ylabel("Posicin Y (m)")

# Perfil en X (y = L/2)
line_x, = ax2.plot(x, u[:, N//2], 'r-', lw=2, label=f'Perfil en Y = {L
    /2:.1f} m')
ax2.set_title("Perfil de Temperatura en direccin X")
ax2.set_xlabel("Posicin X (m)")
ax2.set_ylabel("Temperatura (C)")
ax2.set_ylim(0, source_temp*1.1)
ax2.grid(True)

# Perfil en Y (x = L/4, L/2, 3L/4)
line_y1, = ax3.plot(y, u[N//4, :], 'b-', lw=2, label=f'X = {L/4:.1f} m')
line_y2, = ax3.plot(y, u[N//2, :], 'g-', lw=2, label=f'X = {L/2:.1f} m')
line_y3, = ax3.plot(y, u[3*N//4, :], 'm-', lw=2, label=f'X = {3*L/4:.1f} m'
    )
ax3.set_title("Perfiles de Temperatura en direccin Y")
ax3.set_xlabel("Posicin Y (m)")
ax3.set_ylabel("Temperatura (C)")
ax3.set_ylim(0, source_temp*1.1)
ax3.legend()
ax3.grid(True)

# Informacin
ax4.axis('off')
info_text = ax4.text(0.5, 0.5, "", transform=ax4.transAxes, ha='center',
    va='center')
ax4.set_title("Informacin de la Simulacin")

def animate(t):
    global u
    u = update_solution(u)

    # Actualizar mapa de calor
    img.set_array(u.T)
    img.set_clim(vmin=0, vmax=np.max(u))

    # Actualizar perfiles
```

```
    line_x.set_ydata(u[:, N//2])
    line_y1.set_ydata(u[N//4, :])
    line_y2.set_ydata(u[N//2, :])
    line_y3.set_ydata(u[3*N//4, :])

    # Actualizar informacin
    info = f"Tiempo: {t*dt:.2f} s / {Tmax:.1f} s\n"
    info += f"Temperatura mxima: {np.max(u):.1f} C\n"
    info += f"Parmetros:\n = {alpha} m/s\nx = y = {dx:.3f} m\nt = {dt:.3f}
        s"
    info_text.set_text(info)

    return img, line_x, line_y1, line_y2, line_y3, info_text

ani = FuncAnimation(fig, animate, frames=Nt, interval=50, blit=True)
plt.tight_layout()
plt.show()

# ============================================
# 4. GRFICOS ESTTICOS DE PERFILES (CORREGIDO)
# ============================================

# Simular para tiempos especficos
snapshot_times = [0, 0.5, 2, 5, 10]
snapshots = []
u_static = np.zeros((N, N))
u_static[0, :] = source_temp # Condicin inicial con fuente en el borde

for t in range(int(Tmax/dt)):
    u_static = update_solution(u_static)
    if t*dt in snapshot_times:
        snapshots.append((t*dt, u_static.copy()))

# Crear figura esttica
fig_static, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
fig_static.suptitle("Evolucin de los Perfiles de Temperatura - Fuente en
    borde izquierdo", y=1.02)

# Perfil en X (y = L/2)
colors = plt.cm.viridis(np.linspace(0, 1, len(snapshots)))
```

```
for (time, temp), color in zip(snapshots, colors):
    ax1.plot(x, temp[:, N//2], color=color, lw=2, label=f't = {time:.1f} s'
        )

ax1.set_title(f"Perfil en X (Y = {L/2:.1f} m)")
ax1.set_xlabel("Posicin X (m)")
ax1.set_ylabel("Temperatura (C)")
ax1.legend(title="Tiempo", frameon=True)
ax1.grid(True)

# Perfiles en Y para diferentes X
for (time, temp), color in zip(snapshots, colors):
    ax2.plot(y, temp[N//2, :], color=color, lw=2, linestyle='-', label=f'X
        ={L/2:.1f}m, t={time:.1f}s')
    ax2.plot(y, temp[3*N//4, :], color=color, lw=2, linestyle='--', label=f
        'X={3*L/4:.1f}m, t={time:.1f}s')

ax2.set_title("Perfiles en Y para diferentes posiciones X")
ax2.set_xlabel("Posicin Y (m)")
ax2.set_ylabel("Temperatura (C)")
ax2.legend(title="Posicin y Tiempo", frameon=True, ncol=2)
ax2.grid(True)

# Diagrama esquemtico
ax_sketch = fig_static.add_axes([0.45, 0.6, 0.1, 0.3])
ax_sketch.set_xlim(0, 1)
ax_sketch.set_ylim(0, 1)
ax_sketch.axis('off')
ax_sketch.plot([0], [0.5], 'ro', markersize=10) # Fuente en el borde
ax_sketch.plot([0, 1], [0.5, 0.5], 'k-', lw=2) # Lnea horizontal
ax_sketch.text(0.1, 0.7, "Fuente de calor", ha='left', fontsize=8)
ax_sketch.text(0.5, 0.3, "Lnea Y", ha='center', fontsize=8)
ax_sketch.set_title("Configuracin", fontsize=10)

plt.tight_layout()
plt.savefig('perfiles_temperatura_borde.png', dpi=300, bbox_inches='tight')

plt.show()
```

4.6 Cubo.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Simulacin 3D de la ecuacin de calor en un cubo con Crank-Nicolson
Visualizacin optimizada con cortes 2D y perfiles de temperatura
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.sparse import diags, eye, kron
from scipy.sparse.linalg import spsolve
from matplotlib import cm
import time

# ==========================================
# 1. CONFIGURACIN Y PARMETROS
# ==========================================

# Configuracin para mejor rendimiento
plt.style.use('seaborn')
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['axes.grid'] = True

# Parmetros fsicos optimizados
L = 1.0 # Longitud del cubo [m]
N = 15 # Puntos por dimensin (reducido para rendimiento)
alpha = 0.01 # Difusividad trmica [m/s]
Tmax = 0.1 # Tiempo total de simulacin [s]
source_temp = 100 # Temperatura inicial en la fuente [C]

# Discretizacin espacial y temporal
dx = L / (N - 1)
dt = 0.002 # Paso de tiempo [s]
Nt = int(Tmax / dt)
print(f"Simulacin 3D con {N} puntos y {Nt} pasos temporales")

# Coeficiente para Crank-Nicolson
```

```
r = alpha * dt / (2 * dx**2)

# ============================================
# 2. INICIALIZACIN Y MATRICES
# ============================================

# Campo de temperatura 3D
u = np.zeros((N, N, N))

# Fuente de calor en el centro (cubo pequeo)
u[N//4:3*N//4, N//4:3*N//4, N//4:3*N//4] = source_temp

# Construccin de matrices para ADI en 3D
def build_system_matrices(N, r):
    """Construye matrices del sistema para Crank-Nicolson 3D"""
    # Matriz 1D
    main_diag = (1 + 2*r) * np.ones(N)
    off_diag = -r * np.ones(N-1)
    A = diags([off_diag, main_diag, off_diag], [-1, 0, 1], format='csr')
    B = diags([r*np.ones(N-1), (1-2*r)*np.ones(N), r*np.ones(N-1)], [-1, 0,
        1], format='csr')

    # Matrices identidad
    I = eye(N, format='csr')

    # Productos Kronecker para 3D
    Ax = kron(kron(A, I), I)
    Ay = kron(kron(I, A), I)
    Az = kron(kron(I, I), A)

    Bx = kron(kron(B, I), I)
    By = kron(kron(I, B), I)
    Bz = kron(kron(I, I), B)

    return Ax, Ay, Az, Bx, By, Bz

# Matrices del sistema (precomputadas para eficiencia)
Ax, Ay, Az, Bx, By, Bz = build_system_matrices(N, r)

# ============================================
```

```
# 3. FUNCIN DE ACTUALIZACIN
# ============================================

def update_temperature(u):
    """Actualiza el campo de temperatura usando ADI en 3D"""
    u_flat = u.reshape(-1)

    # Primera etapa: implcito en X
    b = Bx @ u_flat
    u_flat = spsolve(Ax, b)

    # Segunda etapa: implcito en Y
    b = By @ u_flat
    u_flat = spsolve(Ay, b)

    # Tercera etapa: implcito en Z
    b = Bz @ u_flat
    u_flat = spsolve(Az, b)

    # Reformar y aplicar condiciones de frontera
    u_new = u_flat.reshape((N, N, N))
    u_new[0,:,:] = u_new[-1,:,:] = u_new[:,0,:] = u_new[:,-1,:] = u_new
        [:,:,0] = u_new[:,:,-1] = 0

    return u_new

# ============================================
# 4. VISUALIZACIN INTERACTIVA
# ============================================

# Configuracin de la figura
fig = plt.figure(figsize=(15, 8))
fig.suptitle('Difusin de Calor 3D en Cubo - Mtodo Crank-Nicolson',
    fontsize=14)

# Crear subplots para cortes 2D
ax1 = fig.add_subplot(231) # Corte XY
ax2 = fig.add_subplot(232) # Corte XZ
ax3 = fig.add_subplot(233) # Corte YZ
ax4 = fig.add_subplot(212) # Perfiles
```

```
# Configuracin inicial de las visualizaciones
slice_idx = N // 2 # ndice para cortes centrales
x = y = z = np.linspace(0, L, N)

# Cortes 2D iniciales
im1 = ax1.imshow(u[:,:,slice_idx].T, cmap='inferno', origin='lower',
              extent=[0, L, 0, L], vmin=0, vmax=source_temp)
ax1.set_title(f'Corte XY en Z = {z[slice_idx]:.2f}m')
ax1.set_xlabel('X [m]')
ax1.set_ylabel('Y [m]')
plt.colorbar(im1, ax=ax1, label='Temperatura [C]')

im2 = ax2.imshow(u[:,slice_idx,:].T, cmap='inferno', origin='lower',
              extent=[0, L, 0, L], vmin=0, vmax=source_temp)
ax2.set_title(f'Corte XZ en Y = {y[slice_idx]:.2f}m')
ax2.set_xlabel('X [m]')
ax2.set_ylabel('Z [m]')
plt.colorbar(im2, ax=ax2, label='Temperatura [C]')

im3 = ax3.imshow(u[slice_idx,:,:].T, cmap='inferno', origin='lower',
              extent=[0, L, 0, L], vmin=0, vmax=source_temp)
ax3.set_title(f'Corte YZ en X = {x[slice_idx]:.2f}m')
ax3.set_xlabel('Y [m]')
ax3.set_ylabel('Z [m]')
plt.colorbar(im3, ax=ax3, label='Temperatura [C]')

# Perfiles de temperatura
line_x, = ax4.plot(x, u[:, N//2, N//2], 'r-', label='Perfil X (Y=Z=L/2)')
line_y, = ax4.plot(y, u[N//2, :, N//2], 'b-', label='Perfil Y (X=Z=L/2)')
line_z, = ax4.plot(z, u[N//2, N//2, :], 'g-', label='Perfil Z (X=Y=L/2)')
ax4.set_title('Perfiles de Temperatura a travs del Centro')
ax4.set_xlabel('Posicin [m]')
ax4.set_ylabel('Temperatura [C]')
ax4.legend()
ax4.grid(True)
ax4.set_ylim(0, source_temp*1.1)

# Texto informativo
info_text = fig.text(0.1, 0.05, '', fontsize=10)
```

```
# ============================================
# 5. ANIMACIN
# ============================================

def init():
    """Inicializacin de la animacin"""
    im1.set_array(u[:,:,slice_idx].T)
    im2.set_array(u[:,slice_idx,:].T)
    im3.set_array(u[slice_idx,:,:].T)
    line_x.set_ydata(u[:, N//2, N//2])
    line_y.set_ydata(u[N//2, :, N//2])
    line_z.set_ydata(u[N//2, N//2, :])
    info_text.set_text(f'Tiempo = 0.000 s / {Tmax:.3f} s | Temp mxima = {np.
        max(u):.1f} C')
    return im1, im2, im3, line_x, line_y, line_z, info_text

def animate(t):
    """Actualizacin del frame de animacin"""
    global u
    start_time = time.time()

    # Actualizar solucin
    u = update_temperature(u)

    # Actualizar visualizaciones
    im1.set_array(u[:,:,slice_idx].T)
    im2.set_array(u[:,slice_idx,:].T)
    im3.set_array(u[slice_idx,:,:].T)
    line_x.set_ydata(u[:, N//2, N//2])
    line_y.set_ydata(u[N//2, :, N//2])
    line_z.set_ydata(u[N//2, N//2, :])

    # Calcular tiempo de simulacin real
    sim_time = t * dt
    comp_time = time.time() - start_time

    # Actualizar informacin
    info_text.set_text(
        f'Tiempo simulacin = {sim_time:.3f} s / {Tmax:.3f} s | '
```

```
        f'Tiempo clculo = {comp_time:.3f} s | '
        f'Temp mxima = {np.max(u):.1f} C | '
        f' = {alpha} m/s | x = {dx:.3f} m | t = {dt:.4f} s'
    )

    return im1, im2, im3, line_x, line_y, line_z, info_text

# Configurar y mostrar animacin
ani = FuncAnimation(
    fig, animate, frames=min(Nt, 100), # Limitar a 100 frames para demo
    init_func=init, interval=100, blit=False
)

plt.tight_layout()
plt.show()
```