

Diferencias finitas para ecuaciones diferenciales parciales

Contents

- [11.1. Introducción](#)
- [11.2. Ecuaciones elípticas \(EDP estacionaria\)](#)
- [11.3. Ecuaciones parabólicas \(EDP transiente\)](#)
- [11.4. EDP transientes en 2D](#)
- [11.5. Referencias](#)

MEC301 - Métodos Numéricos

Profesor: Francisco Ramírez Cuevas

Fecha: 14 de noviembre 2022

11.1. Introducción

Como mencionamos en la [unidad 9](#), una **ecuación en derivadas parciales (EDP)** corresponde a una **ecuación diferencial para una función desconocida con derivadas respecto a dos o más variables dependientes**.

Por ejemplo,

$$\frac{\partial^2 u}{\partial x^2} + 2xy \frac{\partial^2 u}{\partial y^2} = 0 \quad (11.1)$$

$$\frac{\partial^3 u}{\partial x^2 \partial y} + x \frac{\partial^2 u}{\partial y^2} + 8u = 0 \quad (11.2)$$

$$\left(\frac{\partial^2 u}{\partial x^2} \right)^3 + 6 \frac{\partial^3 u}{\partial x \partial y^2} = x \quad (11.3)$$

$$\left(\frac{\partial^2 u}{\partial x^2} \right)^3 + xu \frac{\partial u}{\partial y} = x \quad (11.4)$$

El **orden** de una EDP se define **respecto a la derivada parcial de mayor orden**. Así, la ecuación (11.1) y (11.4) son de segundo orden, mientras que (11.2) y (11.3) son de tercer orden.

Se dice que una EDP es **lineal si, dada dos soluciones u_1 y u_2 , la función $u_1 + u_2$**

también es solución. Por ejemplo, (11.1) y (11.2) son lineales, mientras que (11.3) y (11.4) no lo son.

En este curso nos enfocaremos en EDP lineales de segundo orden de la forma:

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D = 0 \quad (11.5)$$

donde A, B, C son funciones de x e y , mientras que D es una función de $x, y, \partial u / \partial x$ y $\partial u / \partial y$.

A partir de esta forma general podemos clasificar una EDP lineal de segundo orden como elíptica, parabólica o hiperbólica.

$B^2 - 4AC$	Categoría	Ejemplo
< 0	Elíptica	Ecuación de Laplace (estado estacionario con dos dimensiones espaciales) $\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$
= 0	Parabólica	Ecuación de conducción del calor (variable de tiempo y una dimensión espacial) $\frac{\partial T}{\partial t} = k' \frac{\partial^2 T}{\partial x^2}$
> 0	Hiperbólica	Ecuación de onda (variable de tiempo y una dimensión espacial) $\frac{\partial^2 y}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 y}{\partial t^2}$

Esta clasificación es relevante dado que cada categoría se relaciona con problemas de ingeniería específicos, que demandan soluciones especiales.

Comúnmente, las ecuaciones **elípticas** se utilizan para caracterizar sistemas en **estado estacionario**, tales como la deflección de una placa sometida a una carga, o la distribución de temperaturas en una cavidad 2D.

Las ecuaciones **parabólicas**, en cambio, se usan generalmente cuando la **función depende del tiempo**. Por ejemplo, la deflección de una barra, inicialmente sin carga, sometida a una carga en la punta.

Las ecuaciones **hiperbólicas** son comúnmente **utilizadas en problemas oscilatorios**, como vibraciones o propagación de ondas electromagnéticas.

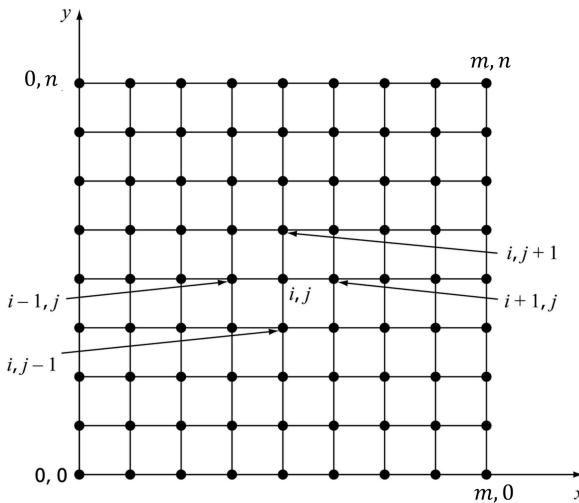
En esta unidad revisaremos EDP de tipo elípticas y parabólica, dada su relevancia en problemas de ingeniería mecánica.

11.2. Ecuaciones elípticas (EDP estacionaria)

La ecuación de Laplace es una EDP elíptica. Se utiliza comúnmente en problemas de conducción de calor. En el caso 2D, la ecuación de Laplace describe la distribución de temperaturas en el espacio.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \quad (11.6)$$

El método más sencillo para resolver EDPs de este tipo es diferencias finitas.



Primero debemos discretizar el dominio en una serie de nodos.

El segundo paso es aplicar un esquema de diferencias finitas en cada nodo. En este caso, lo más conveniente es aplicar diferencia central:

$$\frac{\partial^2 T_{ij}}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 T_{ij}}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2}$$

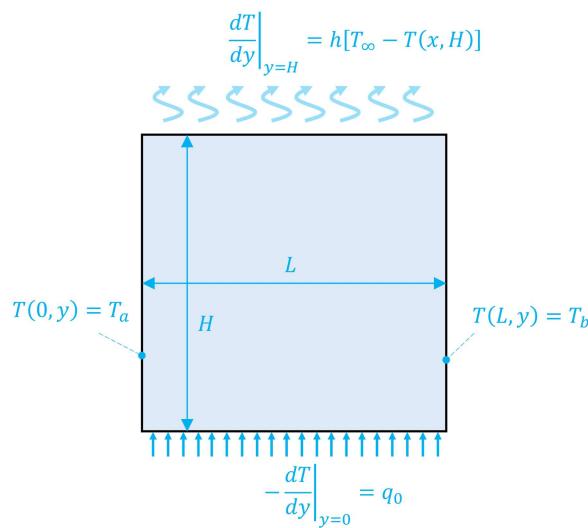
Reemplazando en la ecuación de Laplace, obtenemos una ecuación de la forma:

$$\Delta y^2 T_{i+1,j} - (2\Delta y^2 + 2\Delta x^2) T_{i,j} + \Delta y^2 T_{i-1,j} + \Delta x^2 T_{i,j+1} + \Delta x^2 T_{i,j-1} = 0$$

por conveniencia, **dejamos los valores de Δx y Δy en el numerador** para evitar problemas de crecimiento del error. Esto debido a que $\Delta x < 1$ y $\Delta y < 1$.

El siguiente paso es discretizar las condiciones de borde.

Considerando una placa de ancho L y alto H con las condiciones de borde de la figura



En el caso del lado izquierdo y derecho, tenemos condiciones de Dirichlet.

$$\begin{aligned} T(0, y) = T_a &\Rightarrow T_{0,j} = T_a \\ T(L, y) = T_b &\Rightarrow T_{m,j} = T_b \end{aligned}$$

En el caso del borde sobre y bajo la placa tenemos condiciones de Neumann. Acá, utilizamos diferencia hacia adelante o hacia atrás según corresponda:

$$-\frac{\partial T}{\partial y} \Big|_{y=0} = q_0 \quad \Rightarrow \quad T_{i,1} - T_{i,0} = -\Delta y q_0$$

$$\frac{\partial T}{\partial y} \Big|_{y=H} = h [T_\infty - T(x, H)] \Rightarrow (1 + h\Delta y)T_{i,n} - T_{i,n-1} = \Delta y h T_\infty$$

Finalmente, obtenemos un sistema de ecuaciones de la forma:

$$\Delta y^2 T_{i+1,j} - (2\Delta y^2 + 2\Delta x^2)T_{i,j} + \Delta y^2 T_{i-1,j} + \Delta x^2 T_{i,j+1} + \Delta x^2 T_{i,j-1} = 0$$

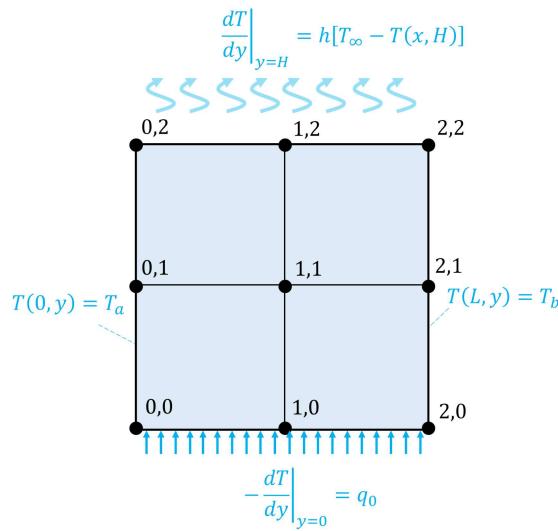
$$T_{i,j} = T_a$$

$$T_{i,j} = T_b$$

$$T_{i,j+1} - T_{i,j} = -\Delta y q_0$$

$$(1 + h\Delta y)T_{i,j} - T_{i,j-1} = \Delta y h T_\infty$$

Consideremos, por ejemplo, una malla de 3x3 nodos



El sistema de ecuaciones, en su forma matricial, es:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \Delta y^2 & 0 & \Delta x^2 & -2\Delta x^2 - 2\Delta y^2 & \Delta x^2 & 0 & \Delta y^2 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 + h\Delta y & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} T_{00} \\ T_{01} \\ T_{02} \\ T_{10} \\ T_{11} \\ T_{12} \\ T_{20} \\ T_{21} \\ T_{22} \end{pmatrix} = \begin{pmatrix} T \\ T \\ T \\ \Delta y \\ 0 \\ \Delta y h \\ T \\ T \\ T \end{pmatrix}$$

Cuando extendemos el número de nodos, el sistema toma la forma de una matriz diagonal dominante

Por ejemplo, si consideramos $\Delta x = \Delta y = 1$ y condiciones de borde de Dirichlet, podemos comprobar que la matriz A toma la forma:

$$\begin{bmatrix} -4 & 1 & & 1 \\ 1 & -4 & 1 & & 1 \\ & 1 & -4 & 1 & & 1 \\ & 1 & & 1 & -4 & 1 & & 1 \\ & & \ddots & & \ddots & & \ddots & & \ddots \\ & & & \ddots & & \ddots & & & \ddots \\ & & & & \ddots & & \ddots & & & \ddots \\ & & & & & 1 & 1 & -4 & 1 & & 1 \\ & & & & & 1 & 1 & -4 & 1 & & 1 \\ & & & & & 1 & & 1 & -4 & 1 & \\ & & & & & 1 & & & 1 & -4 & \end{bmatrix}$$

¿Cómo podemos implementar un código computacional que genere el sistema $Ax = b$ con este esquema, de forma genérica y para cualquier tipo de condición de borde?

11.2.1. Solución EDP lineal estacionaria

Antes de implementar nuestro código, debemos recordar que la solución a este tipo de sistemas está dado por el método de Gauss-Seidel.

Recordando la [unidad 2](#) del curso, el método de Gauss-Seidel se resume en los siguientes pasos:

1. Asumimos un valor inicial para $x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}$ (con excepción de $x_1^{(0)}$).
2. Calculamos un nuevo valor para $x_1^{(1)}$ mediante:

$$x_1^{(1)} = \frac{1}{a_{1,1}} \left[y_1 - \sum_{j \neq 1}^n a_{1,j} x_j^{(0)} \right]$$

3. Utilizando el nuevo valor $x_1^{(1)}$ y el resto de $x^{(0)}$ (con excepción de $x_2^{(0)}$), determinamos $x_2^{(1)}$.

$$x_2^{(1)} = \frac{1}{a_{2,2}} \left[y_2 - \sum_{j \neq 1,2}^n a_{2,j} x_j^{(0)} - a_{2,1} x_1^{(1)} \right]$$

4. Repetimos el paso 3 hasta completar todos los elementos del vector x .
5. Continuaremos iterando hasta que $\|x^{(i)} - x^{(i-1)}\| < \varepsilon$, donde ε es la tolerancia definida respecto al error absoluto.

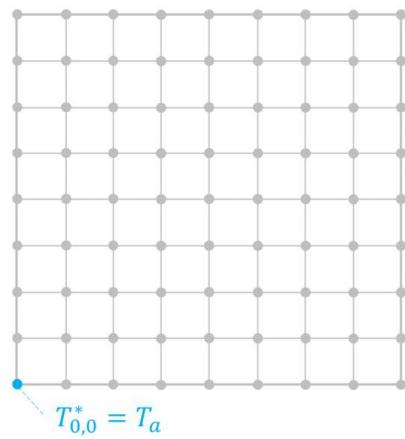
Siguiendo este esquema, una forma alternativa es implementar el método de Gauss-Seidel directamente, es decir, sin desarrollar la forma explícita del sistema $Ax = b$.

En otras palabras, implementamos el método iterativo a través de las ecuaciones:

$$\begin{aligned}
 T_{i,j}^* &= \frac{1}{2\Delta y^2 + 2\Delta x^2} (\Delta y^2 T_{i+1,j} + \Delta y^2 T_{i-1,j} + \Delta x^2 T_{i,j+1} + \Delta x^2 T_{i,j-1}) && \text{if } i \neq 0, n \\
 T_{i,j}^* &= T_a && \text{if } j = 0 \\
 T_{i,j}^* &= T_b && \text{if } j = n \\
 T_{i,j}^* &= T_{i,j+1} + \Delta y q_0 && \text{if } i = n \\
 T_{i,j}^* &= \frac{1}{(1 + h\Delta y)} (T_{i,j-1} + \Delta y h T_\infty) && \text{if } i = 0
 \end{aligned}$$

donde $T_{i,j}^*$ corresponde al valor obtenido después de cada iteración.

La siguiente animación ilustra el proceso iterativo:



11.2.2. Solución EDP estacionaria en python

Analicemos esto en un código.

Considerando los siguientes parámetros:

$$\begin{aligned}
 T_a &= 500 \text{ K} \\
 T_b &= 500 \text{ K} \\
 T_\infty &= 300 \text{ K} \\
 h &= 100 \text{ m}^{-1} \\
 q_0 &= 1000 \text{ K/m} \\
 L &= 1 \text{ m} \\
 H &= 1.5 \text{ m}
 \end{aligned}$$

```

# definimos las constantes del problema
Ta = 500 # Temperatura al Lado izquierdo (K)
Tb = 500 # Temperatura al Lado derecho (K)
Too = 300 # Temperatura del aire (K)
h = 100 # Coeficiente convectivo (m^-1)
q0 = 1000 # flujo de calor (K/m)
L, H = 1, 1.5 # Largo y ancho de la cavidad (m)

```

```
import numpy as np
from numpy.linalg import norm # norma de Frobenius
from itertools import product # Librería para iteración
```

Implementamos el método en una **función**, definida en base al número de nodos N_x y N_y y la tolerancia relativa $\frac{\|x-x_{\text{old}}\|}{\|x\|} < \varepsilon_{\text{rel}}$

Como valor de entrada en la iteración, consideramos $T_{i,j}^0 = T_\infty$

```
def T_plate(Nx,Ny,rel_tol, k_iter = 10000):
    # Definimos las características de la malla
    nx, ny = Nx - 1, Ny - 1           # índice último nodo
    dx, dy = L/(Nx - 1), H/(Ny - 1) # espaciamiento entre nodos

    # Iteramos
    T = np.ones((Nx,Ny)) # valores primera iteración
    converged = False      # booleano para chequear convergencia
    for k in range(k_iter):
        Told = T.copy()          # guardamos la iteración previa

        for i, j in product(range(Nx),range(Ny)): # Loop sobre i y
            j

            # condiciones de borde
            if i == 0: T[i,j] = Ta
            elif i == nx: T[i,j] = Tb
            elif j == 0: T[i,j] = T[i,j+1] + dy*q0
            elif j == ny: T[i,j] = 1/(1 + dy*h)*(T[i,j-1] +
            dy*h*T0)

            # nodos centrales
            else: T[i,j] = 1/(2*dx**2 + 2*dy**2)*
            (dy**2*T[i+1,j] + dy**2*T[i-1,j] + dx**2*T[i,j+1] + dx**2*T[i,j-
            1])

            E_rel = norm(T - Told)/norm(T) # comprobamos el error
            relativo
            if E_rel < rel_tol:          # condición de convergencia
                print('Converged!\n\tN. iter = %i\n\tE_error = %.3f%%' %
                (k, E_rel*100))
                converged = True
                break

        if not converged: print('Method did not converge!\n\tE_error =
        %.3f%%' % (E_rel*100))
    return T
```

```
# Definimos las características de la malla
Nx, Ny = 51, 76           # total de nodos
rel_tol = 0.00001          # tolerancia error relativo
T = T_plate(Nx,Ny,rel_tol) # Determinamos T(x,y)
```

```
Converged!
N. iter = 2073
E_error = 0.001%
```

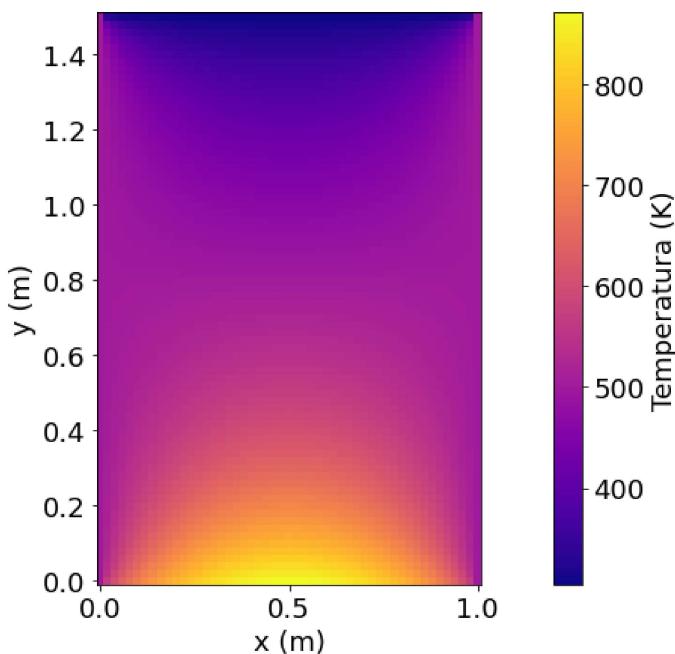
```
%capture showplot
import matplotlib.pyplot as plt
from matplotlib import cm

x = np.linspace(0,L,Nx) # coordenadas x
y = np.linspace(0,H,Ny) # coordenadas y
xx, yy = np.meshgrid(x,y) # malla x-y

plt.figure(figsize = (7, 7))
plt.rcParams.update({'font.size': 18})

plt.pcolor(xx, yy, T.T, cmap=cm.get_cmap(cm.plasma))
plt.colorbar(label="Temperatura (K)", orientation="vertical")
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.axis('scaled')
plt.show()
```

showplot()



Notar que para graficar usamos la traspuesta de la solución mediante el operador `.T`

```
plt.pcolor(xx, yy, T.T)
```

Esto es debido a que el índice i , asociado al eje x , corresponde a las filas del arreglo `T`, mientras que el índice j recorre las columnas de `T`.

Notar también que, en este problema, requerimos de un error relativo pequeño (0.001% en este caso) para asegurar convergencia. Si aumentamos la tolerancia, la solución se hace sensible al valor inicial en la iteración.

11.2.3. Variables secundarias

En el problema anterior, la temperatura es la variable principal del problema. Sin embargo, es común que también que necesitemos información de **variables secundarias** basadas en la derivada o integral de la variable principal.

Por ejemplo, el **flujo de calor** está definido por: $\vec{q} = -k\nabla T = -k \left(\frac{\partial T}{\partial x} \hat{x} + \frac{\partial T}{\partial y} \hat{y} \right)$

En `python` usamos `gradient` para determinar la derivada en problemas de dos o más dimensiones

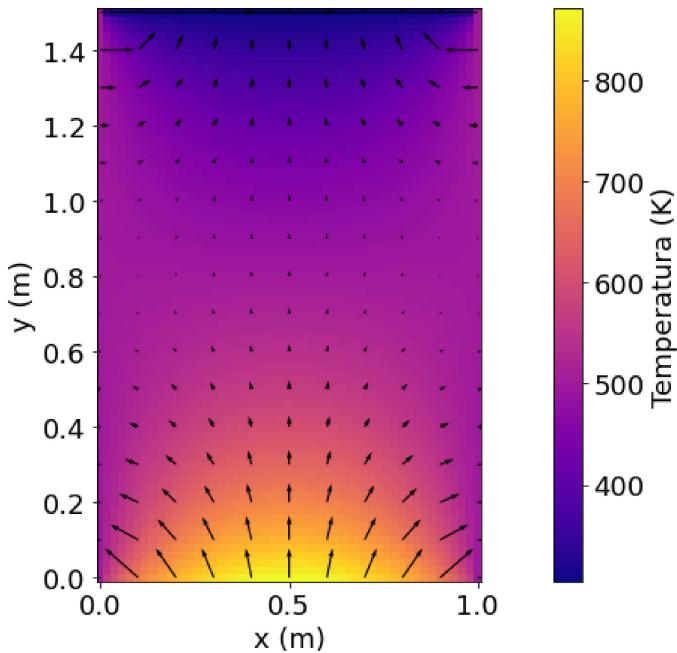
```
%capture showplot1

skip = (slice(None, None, 5), slice(None, None, 5))
qy, qx = np.gradient(T.T,y,x) # gradiente

plt.figure(figsize = (7, 7))
plt.rcParams.update({'font.size': 18})

plt.pcolor(xx, yy, T.T, cmap=cm.get_cmap(cm.plasma))
plt.colorbar(label="Temperatura (K)", orientation="vertical")
plt.quiver(xx[skip],yy[skip],- qx[skip],- qy[skip]) # campo
vectorial de q
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.axis('scaled')
plt.show()
```

```
showplot1()
```



Notar el uso de la función `gradient` y `quiver` en este caso .

En el caso de `gradient`, debemos alternar los ejes x e y en la forma:

```
qy, qx = np.gradient(T.T,y,x) # gradiente
```

Para poder determinar el valor de q debemos aplicar el negativo de `qx` y `qy`. Esto lo vemos en la función `quiver`:

```
plt.quiver(xx,yy,- qx,- qy) # campo vectorial de q
```

Por último, usamos `slice` para reducir el número de índices a graficar por `quiver`. En este contexto, la instrucción `skip = (slice(None, None, 5), slice(None, None, 5))`, es equivalente a:

```
plt.quiver(xx[::5, ::5], yy[::5, ::5], - qx[::5, ::5], - qy[::5, ::5]) # campo vectorial de q
```

Para EDP lineales estacionarias, como la ecuación de Laplace, el método de Gauss-Seidel es práctico y sencillo.

Para ecuaciones no lineales, el método puede ser aplicado mediante esquemas de linearización (similar a [Newton Raphson generalizado](#)). Esto dada la dificultad de determinar una expresión explícita para $T_{i,j}$ con el sistema no-lineal.

Alternativamente, podemos utilizar EDP de tipo transiente y evaluar la evolución de la solución hasta alcanzar un comportamiento estacionario. Esto lo revisaremos a continuación.

11.3. Ecuaciones parabólicas (EDP transiente)

Una de las ecuaciones parabólicas más comunes es la ley de conducción de calor 1D transiente:

$$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2}$$

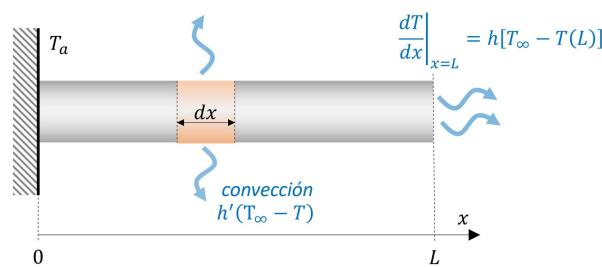
Esta ecuación requiere de dos condiciones de borde para x y una condición inicial para t .

También podemos usar el método de diferencias finitas para resolver este problema. Sin embargo, en este caso tenemos un problema de condiciones de borde mezclada con un problema de valor inicial.

Como revisamos en la [unidad 9](#), existen diversos métodos para resolver problemas de valor inicial. En esta unidad nos enfocaremos en el **método de Euler explícito**.

11.3.1. EDP lineal 1D transiente

Nuevamente, revisemos una barra que se enfriá por convección. Ahora, la barra se encuentra inicialmente a temperatura ambiente T_∞



$$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2} + h'(T_\infty - T)$$

$$T(0, x) = T_\infty$$

$$T(t, 0) = T_a; \quad \left. \frac{\partial T}{\partial x} \right|_{x=L} = h^* [T_\infty - T(t, L)]$$

Usando diferencias finitas con el método de Euler explícito:

$$\frac{T_i^{l+1} - T_i^l}{\Delta t} = k \frac{T_{i+1}^l - 2T_i^l + T_{i-1}^l}{\Delta x^2} + h'(T_\infty - T_i^l) \quad \text{if } i \neq 0, m$$

$$T_i^{l+1} = T_a \quad \text{if } i = 0$$

$$\frac{T_i^{l+1} - T_{i-1}^{l+1}}{\Delta x} = h'(T_\infty - T_i^{l+1}) \quad \text{if } i = m$$

El sistema a resolver esta dado por:

$$T_i^{l+1} = (1 - \Delta t h') T_i^l + \frac{k \Delta t}{\Delta x^2} (T_{i+1}^l - 2T_i^l + T_{i-1}^l) + \Delta t h' T_\infty \quad \text{if } i \neq 0, m$$

$$T_i^{l+1} = T_a \quad \text{if } i = 0$$

$$T_i^{l+1} = \frac{1}{1 + \Delta x h'} (T_{i-1}^{l+1} + \Delta x h' T_\infty) \quad \text{if } i = m$$

Notar que las condiciones de borde están definidas respecto al valor futuro. Esto es simplemente por convención en el método explícito

Resolvemos la ecuación para los siguientes parámetros:

$$\begin{aligned} T_a &= 400 \text{ K} \\ T_\infty &= 300 \text{ K} \\ h' &= 0.002 \text{ s}^{-1} \\ k &= 0.0001 \text{ m}^2/\text{s} \\ L &= 1 \text{ m} \end{aligned}$$

```
# definimos las constantes del problema
Ta = 400 # Temperatura al lado izquierdo (K)
Too = 300 # Temperatura del aire (K)
h = 0.002 # Coeficiente convectivo (m^-1)
k = 0.0001 # flujo de calor (m^2/s)
L = 1 # Largo de La barra (m)
```

Para asegurar convergencia y estabilidad en nuestra solución, se debe cumplir la condición:

$$\Delta t \leq \frac{1}{2} \frac{\Delta x^2}{k} \quad (11.7)$$

```

Nx = 20                      # número total de nodos
n = Nx - 1                    # índice del nodo extremo
dx = L/(Nx - 1)               # espaciamiento entre nodos
dt = 0.5*1/2*dx**2/k          # paso de tiempo
t = np.arange(0,100*dt,dt)    # Intervalo de tiempo

T0 = Too*np.ones(Nx)          # condición inicial
T_time = [T0]                 # arreglo para almacenar soluciones

T = T0.copy()
for l in range(len(t)) :
    Tl = T.copy()
    for i in range(Nx):
        if i == 0: T[i] = Ta
        elif i == n: T[i] = 1/(1+dx*h)*(T[i-1] + dx*h*Too)
        else:
            T[i] = (1 - dt*h)*Tl[i] + k*dt/dx**2*(Tl[i+1] -
            2*Tl[i] + Tl[i-1]) + dt*h*Too
    T_time.append(T.copy())

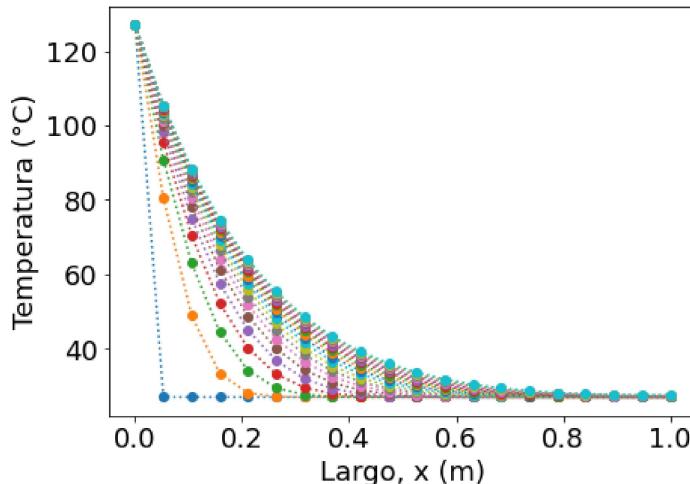
```

Ahora graficamos nuestro resultado

```

x = np.linspace(0,L,Nx)          # arreglo de puntos en x
plt.figure(figsize = (7, 5))
plt.rcParams.update({'font.size': 18})
for it in range(1,len(t),5):      # iteraremos sobre nuestras
soluciones
    plt.plot(x,T_time[it] - 273,'o:')
plt.xlabel('Largo, x (m)')
plt.ylabel('Temperatura (°C)')
plt.show()

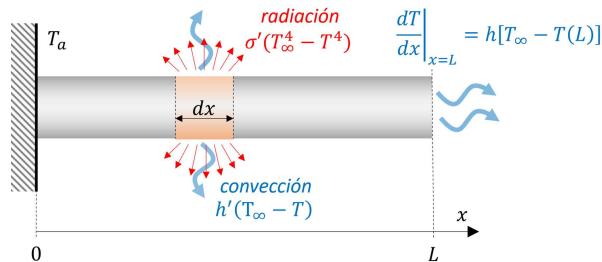
```



11.3.2. EDP no-lineal 1D transiente

Una ventaja del método de Euler explícito es que nos permite fácilmente extender el resultado a EDP del tipo no lineal. Esto debido a que los valores de T_i^{l+1} siempre quedan expresados en función de valores conocidos.

Analicemos esto con el problema de la barra ahora con enfriamiento por convección y radiación



$$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2} + h'(T_\infty - T) + \sigma'(T_\infty^4 - T^4)$$

$$T(0, x) = T_\infty$$

$$T(t, 0) = T_a; \quad \left. \frac{\partial T}{\partial x} \right|_{x=L} = h^* [T_\infty - T(t, L)]$$

Al aplicar diferencias finitas, notamos que el problema, nuevamente, queda definido de forma explícita en función de valores conocidos:

$$T_i^{l+1} = (1 - \Delta t h') T_i^l - \Delta t \sigma' (T_i^4)^l + \frac{k \Delta t}{\Delta x^2} (T_{i+1}^l - 2T_i^l + T_{i-1}^l) + \Delta t h' T_\infty + \Delta t \sigma'$$

$$T_i^{l+1} =$$

$$T_i^{l+1} = \frac{1}{1 + \Delta x h'} (T_{i-1}^{l+1} + \Delta x h' T_i^l)$$

Analicemos la solución de este problema para los siguientes parámetros:

$$\begin{aligned} T_a &= 400 \text{ K} \\ T_\infty &= 300 \text{ K} \\ h' &= 0.002 \text{ s}^{-1} \\ \sigma' &= 1 \times 10^{-10} \text{ s}^{-1} \text{ K}^{-3} \\ k &= 0.0001 \text{ m}^2/\text{s} \\ L &= 1 \text{ m} \end{aligned}$$

```
# definimos las constantes del problema
Ta = 400 # Temperatura al lado izquierdo (K)
Too = 300 # Temperatura del aire (K)
h = 0.002 # Coeficiente convectivo (m^-1)
S = 1E-10 # Coeficiente convectivo (m^-1)
k = 0.0001 # flujo de calor (m^2/s)
L = 1 # Largo de la barra (m)
```

```

Nx = 20                      # número total de nodos
n = Nx - 1                    # índice del nodo extremo
dx = L/(Nx - 1)               # espaciamiento entre nodos
dt = 0.5*1/2*dx**2/k          # paso de tiempo
t = np.arange(0,100*dt,dt)    # Intervalo de tiempo

T0 = Too*np.ones(Nx)          # condición inicial
T_time = [T0]                 # arreglo para almacenar soluciones

T = T0.copy()
for l in range(len(t)) :
    Tl = T.copy()
    for i in range(Nx):
        if i == 0: T[i] = Ta
        elif i == n: T[i] = 1/(1+dx*h)*(T[i-1] + dx*h*Too)
        else:
            T[i] = (1 - dt*h)*Tl[i] - dt*S*Tl[i]**4 + k*dt/dx**2*(Tl[i+1] - 2*Tl[i] + Tl[i-1]) + dt*h*Too + dt*S*Too**4
    T_time.append(T.copy())

```

Graficamos nuestro resultado

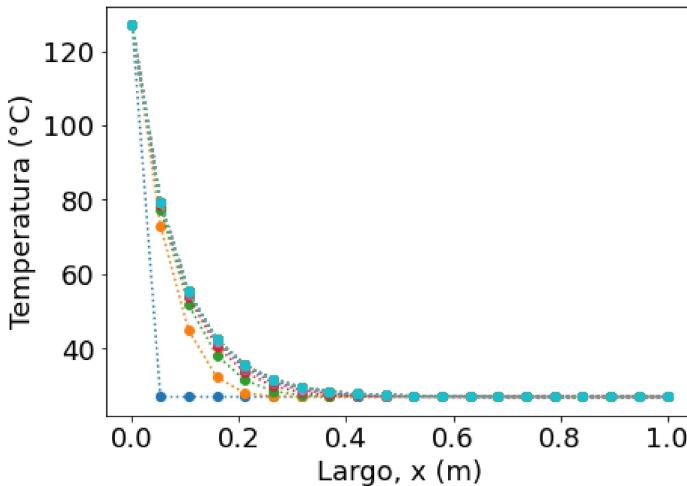
```

x = np.linspace(0,L,Nx)          # arreglo de puntos en x

plt.figure(figsize = (7, 5))
plt.rcParams.update({'font.size': 18})

for it in range(1,len(t),5):      # iteraremos sobre nuestras
    soluciones
        plt.plot(x,T_time[it] - 273,'o:')
plt.xlabel('Largo, x (m)')
plt.ylabel('Temperatura (°C)')
plt.show()

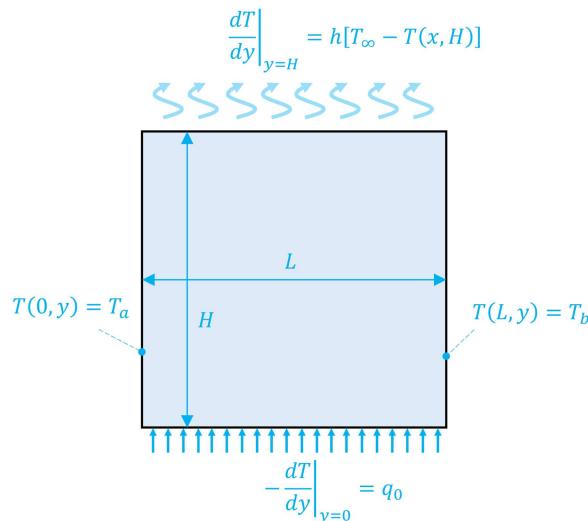
```



11.4. EDP transientes en 2D

El método explícito puede ser fácilmente extendido a problemas con más dimensiones, EDP no lineales, o incluso, con más de una variable dependiente (sistemas de EDPs).

Revisemos una EDP 2D, considerando el problema de la placa analizado anteriormente.



Ahora como un problema transiente:

$$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2} + k \frac{\partial^2 T}{\partial y^2}$$

con condición inicial $T(0, x, y) = T_\infty$

Usando diferencias finitas, con Euler explícito, derivamos un sistema de la forma:

$$\begin{aligned}
 T_{i,j}^{l+1} &= T_{i,j}^l + \frac{k\Delta t}{\Delta x^2} (T_{i+1,j}^l - 2T_{i,j}^l + T_{i-1,j}^l) + \frac{k\Delta t}{\Delta y^2} (T_{i,j+1}^l - 2T_{i,j}^l + T_{i,j-1}^l) && \text{if } i \\
 T_{i,j}^{l+1} &= T_a && \text{if } i \\
 T_{i,j}^{l+1} &= T_b && \text{if } i \\
 T_{i,j+1}^{l+1} - T_{i,j}^{l+1} &= -\Delta y q_0 && \text{if } . \\
 (1 + h\Delta y)T_{i,j}^{l+1} - T_{i,j-1}^{l+1} &= \Delta y h T_\infty && \text{if } .
 \end{aligned}$$

Nuevamente, usamos valores futuros para las condiciones de borde

En este caso, la condición de estabilidad y convergencia está dada por:

$$\Delta t \leq \frac{1}{8} \frac{\Delta x^2 + \Delta y^2}{k} \quad (11.8)$$

Resolvamos este problema en un código, considerando los parámetros

$$\begin{aligned}
 T_a &= 500 \text{ K}; & h &= 100 \text{ m}^{-1} \\
 T_b &= 500 \text{ K}; & k &= 0.0001 \text{ m}^2/\text{s} \\
 T_\infty &= 300 \text{ K}; & q_0 &= 1000 \text{ K/m} \\
 L &= 1 \text{ m}; & H &= 1.5 \text{ m}
 \end{aligned}$$

```

# definimos las constantes del problema
Ta = 500 # Temperatura al Lado izquierdo (K)
Tb = 500 # Temperatura al Lado derecho (K)
Too = 300 # Temperatura del aire (K)
h = 100 # Coeficiente convectivo (m^-1)
k = 0.0001 # conductividad térmica (m^2/s)
q0 = 1000 # flujo de calor (K/m)
L, H = 1, 1.5 # Largo y ancho de la cavidad (m)

```

En este caso, definiremos una función `T_plate_time` que determinará la distribución de temperaturas para un tiempo `tend`

```
def T_plate_time(Nx,Ny, tend):
    # Definimos las características de la malla
    nx, ny = Nx - 1, Ny - 1           # índice último nodo
    dx, dy = L/(Nx - 1), H/(Ny - 1)   # espacio entre nodos
    dt = 0.5*1/8*(dx**2 + dy**2)/k   # paso de tiempo
    t = np.arange(0,tend,dt)          # Intervalo de tiempo

    # Iteramos
    T0 = Too*np.ones((Nx,Ny))         # condición inicial
    T = T0.copy()
    for l in range(len(t)):
        Tl = T.copy()                 # guardamos la iteración previa

        for i, j in product(range(Nx),range(Ny)): # Loop sobre i y
            j

            # condiciones de borde
            if i == 0: T[i,j] = Ta
            elif i == nx: T[i,j] = Tb
            elif j == 0: T[i,j] = T[i,j+1] + dy*q0
            elif j == ny: T[i,j] = 1/(1 + dy*h)*(T[i,j-1] +
            dy*h*Too)

            # nodos centrales
            else: T[i,j] = Tl[i,j] + k*dt/dx**2*(Tl[i+1,j] -
            2*Tl[i,j] + Tl[i-1,j]) \
                    + k*dt/dy**2*(Tl[i,j+1] -
            2*Tl[i,j] + Tl[i,j-1])

    return T
```

```
%%capture showplot2

# Definimos las características de la malla
Nx, Ny = 51, 76                      # total de nodos
T = T_plate_time(Nx,Ny, tend = 200) # Determinamos T(x,y)

x = np.linspace(0,L,Nx) # coordenadas x
y = np.linspace(0,H,Ny) # coordenadas y
xx, yy = np.meshgrid(x,y) # malla x-y

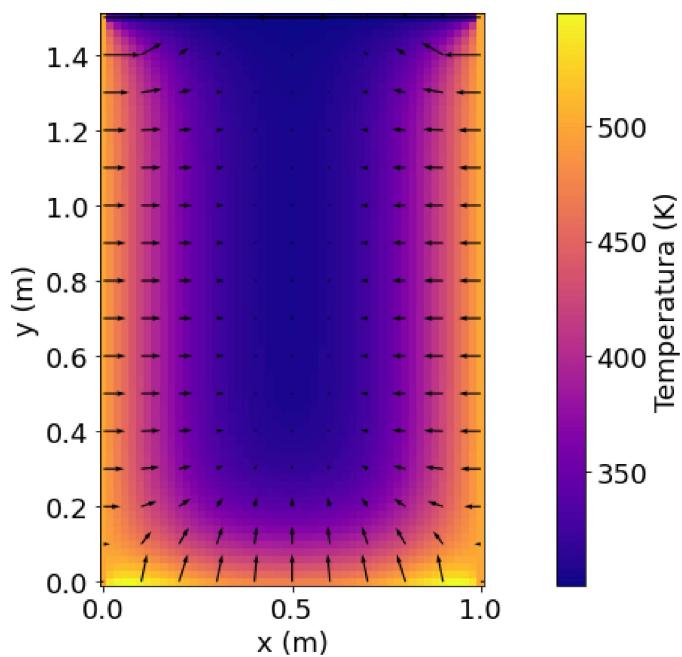
skip = (slice(None, None, 5), slice(None, None, 5))
qy, qx = np.gradient(T.T,y,x) # gradiente

plt.figure(figsize = (7, 7))
plt.rcParams.update({'font.size': 18})

plt.pcolor(xx, yy, T.T, cmap=cm.get_cmap(cm.plasma))
plt.colorbar(label="Temperatura (K)", orientation="vertical")

# campo vectorial de q
plt.quiver(xx[skip],yy[skip],- qx[skip],- qy[skip])
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.axis('scaled')
plt.show()
```

```
showplot2()
```



11.5. Referencias

- Chapra S., Canale R. **Capítulo 28: Diferencias finitas: ecuaciones elípticas** en *Métodos Numéricos para Ingenieros*, 6ta Ed., McGraw Hill, 2011
- Chapra S., Canale R. **Capítulo 29: Diferencias finitas: ecuaciones parabólicas** en *Métodos Numéricos para Ingenieros*, 6ta Ed., McGraw Hill, 2011
- Williams H. P. **Chapter 20: Partial differential equations** in "Numerical Recipes" 3rd Ed, Cambridge University Press, 2007

By Francisco V. Ramirez-Cuevas

© Copyright 2022.