

8 Conductividad variable.

Objetivo.

Considere la ecuación de Poisson con κ variable y condiciones de frontera de tipo Dirichlet:

$$-\frac{d}{dx} \left(\kappa \frac{du}{dx} \right) = f \quad \text{con} \quad \kappa = \kappa(x)$$

Resolver el problema para los siguientes casos: * Caso 1:

$L = 1$, $N = 50$, $A = 2.0$, $B = 1.0$, $\kappa = |\sin(4\pi x)| + \delta\kappa$ con $\delta\kappa = 0.1$. * Caso 2:

$L = 1$, $N = 50$, $A = 2.0$, $B = 1.0$, $\kappa = \text{random}(x) + \delta\kappa$ con $\delta\kappa = 0.1$.

[MACTI-Analisis_Numerico_01](#) by [Luis M. de la Cruz](#) is licensed under

[Attribution-ShareAlike 4.0 International](#) 

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101019, PE101922

```
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis

def buildMatrix(N, k, f):
    """
    Parameters:
    N: int Tamaño de la matriz.
    k: float Conductividad.
    f: función para calcular las conductividades
    """
    # Matriz de ceros
    A = np.zeros((N,N))

    # Primer renglón
    A[0,0] = (f(k[0], k[1]) + f(k[1], k[2]))
    A[0,1] = -f(k[1], k[2])

    # Renglones interiores
    for i in range(1,N-1):
        """ BEGIN SOLUTION
        A[i,i] = (f(k[i+2], k[i+1]) + f(k[i+1], k[i]))
        A[i,i+1] = -f(k[i+1], k[i+2])
        A[i,i-1] = -f(k[i+1], k[i])
        """ END SOLUTION

    # Último renglón
    A[N-1,N-2] = -f(k[N-1], k[N])
    A[N-1,N-1] = (f(k[N-1], k[N]) + f(k[N], k[N+1]))
```

```
return A
```

```
# Parámetros físicos
L = 1.0
bA = 2.0 # Valor de u en A (Dirichlet)
bB = 1.0 # Valor de u en B (Dirichlet)
S = 0.0

# Parámetros numéricos
N = 50 # Número de incógnitas
h = L / (N+1)
r = 1 / h**2

# Coordenadas de los nodos
x = np.linspace(0, L, N+2)

# Conductividad variable
#k = np.abs(np.sin(4 * np.pi * x)) + 0.1
k = np.random.rand(N+2) + 0.1
```

```
# Promedio Aritmético y Media Armónica
### BEGIN SOLUTION
def pAritmetico(a, b):
    return 0.5 * (a + b)

def mArmonica(a, b):
    return 2 * a * b / (a + b)
### EDN SOLUTION
```

```
A = buildMatrix(N, k, pAritmetico) # Construcción de la matriz
b = np.zeros(N) # Lado derecho del sistema

b[1:] = S / r # Fuente o sumidero
b[0] += pAritmetico(k[1], k[0]) * bA # Condición de frontera en A
b[-1] += pAritmetico(k[N+1], k[N]) * bB # Condición de frontera en B

u1 = np.zeros(N+2) # Arreglo para almacenar la solución
u1[0] = bA # Frontera izquierda Dirichlet
u1[-1] = bB # Frontera derecha Dirichlet
u1[1:N+1] = np.linalg.solve(A,b) # Sol. del sist. lineal

A = buildMatrix(N, k, mArmonica) # Construcción de la matriz
b = np.zeros(N) # Lado derecho del sistema
b[1:] = S / r # Fuente o sumidero
b[0] += mArmonica(k[1], k[0]) * bA # Condición de frontera en A
b[-1] += mArmonica(k[N+1], k[N]) * bB # Condición de frontera en B

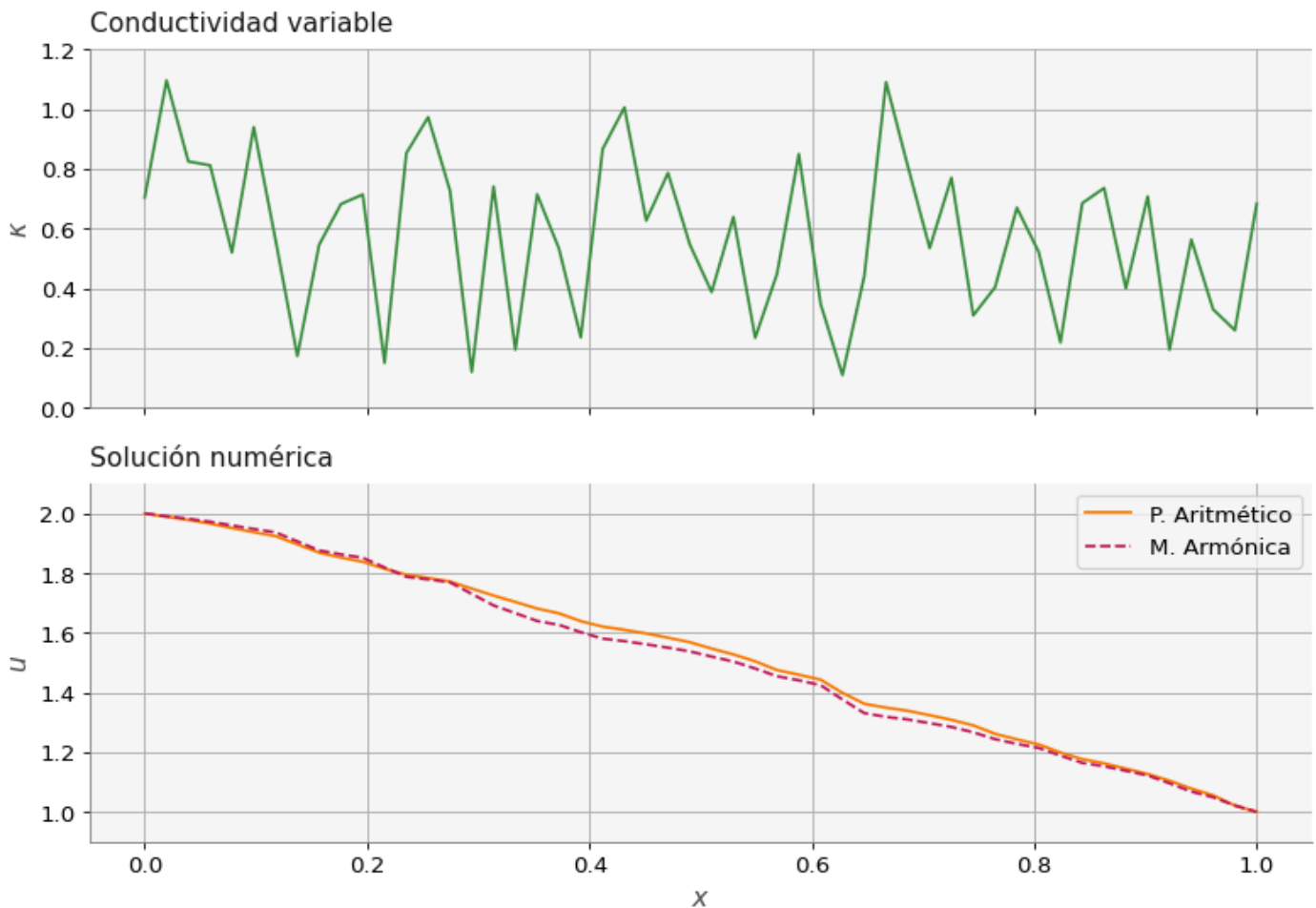
u2 = np.zeros(N+2) # Arreglo para almacenar la solución
```

```

u2[0] = bA          # Frontera izquierda Dirichlet
u2[-1] = bB         # Frontera derecha Dirichlet
u2[1:N+1] = np.linalg.solve(A,b) # Sol. del sist. lineal

fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,7))
ax1.plot(x,k,'C2-')
ax1.set_ylabel('$\kappa$')
ax1.set_title('Conductividad variable')
ax1.set_ylim(0.0,1.2)
ax1.grid()
ax2.plot(x,u1,'C1-', label='P. Aritmético')
ax2.plot(x,u2,'C3--', label='M. Armónica')
ax2.set_xlabel('$x$')
ax2.set_ylabel('$u$')
ax2.legend()
ax2.set_title('Solución numérica')
ax2.set_ylim(0.9,2.1)
ax2.grid()
plt.tight_layout()

```





1 Repaso de cálculo: derivadas

Objetivo general - Realizar ejercicios de derivadas en una variable.

[MACTI-Analysis_Numerico_01](#) by [Luis M. de la Cruz](#) is licensed under [Attribution-ShareAlike 4.0 International](#)

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

```
# Importamos todas las bibliotecas a usar
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sympy as sym
import macti.visual
from macti.evaluation import *
```

```
quizz = Quizz('q1', 'notebooks', 'local')
```

1.1 Ejercicios.

Calcula las derivadas de las funciones descritas siguiendo las reglas del apartado [Reglas de derivación](#). Deberás escribir tu respuesta matemáticamente usando notación de Python en la variable **respuesta**.

Por ejemplo la para escribir $4x^{m-1} + \cos^2(x)$ deberás escribir:

```
respuesta = 4 * x**(m-1) + sym.cos(x)**2
```

1.1.1 1. Potencias:

1. a. $f(x) = x^5, f'(x) = ?$

```
# Definimos el símbolo x
x = sym.symbols('x')

# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = 5*x**4

file_answer = FileAnswer()
file_answer.write('1a', str(respuesta))
```

```
#### END SOLUTION
```

```
display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$5x^4$$

```
quizz.eval_expression('1a', respuesta)
```

1a | Tu respuesta:
es correcta.

$$5x^4$$

1. b. $f(x) = x^m, f'(x) = ?$

```
# Definimos el símbolo m
m = sym.symbols('m')

# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = m * x**(m-1)

file_answer.write('1b', str(respuesta))
#### END SOLUTION

display(respuesta)
```

$$mx^{m-1}$$

```
quizz.eval_expression('1b', respuesta)
```

1b | Tu respuesta:
es correcta.

$$mx^{m-1}$$

2. Constantes

2. a. $f(x) = \pi^{435}, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = 0

file_answer.write('2a', str(respuesta))
#### END SOLUTION
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

0

```
quizz.eval_expression('2a', respuesta)
```

2a | Tu respuesta:
es correcta.

0

2. b. $f(x) = e^\pi$, $f'(x) = i$?

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = 0

file_answer.write('2b', str(respuesta))
#### END SOLUTION

display(respuesta)
```

0

```
quizz.eval_expression('2b', respuesta)
```

2b | Tu respuesta:
es correcta.

0

3. Multiplicación por una constante

3. a. $f(x) = 10x^4, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = 40 * x ** 3

file_answer.write('3a', str(respuesta))
#### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$40x^3$

```
quizz.eval_expression('3a', respuesta)
```

3a | Tu respuesta:
es correcta.

$40x^3$

3. b. $f(x) = Ax^n, f'(x) = ?$

```
# Definimos los símbolos A y n
A, n = sy.symbols('A n')

# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = A * n * x ** (n-1)

file_answer.write('3b', str(respuesta))
#### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$Anx^{n-1}$$

```
quizz.eval_expression('3b', respuesta)
```

 3b | Tu respuesta:
 es correcta.

$$Anx^{n-1}$$

4. Suma y Diferencia

4. a. $f(x) = x^2 + x + 1, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = 2*x + 1

file_answer.write('4a', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
 Respuestas y retroalimentación almacenadas.

$$2x + 1$$

```
quizz.eval_expression('4a', respuesta)
```

 4a | Tu respuesta:
 es correcta.

$$2x + 1$$

4. b. $f(x) = \sin(x) - \cos(x), f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = sy.cos(x) + sy.sin(x)
```



```
file_answer.write('4b', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$\sin(x) + \cos(x)$$

```
quizz.eval_expression('4b', respuesta)
```

4b | Tu respuesta:
es correcta.

$$\sin(x) + \cos(x)$$

4. c. $f(x) = Ax^m - Bx^n + C$, $f'(x) = ?$

```
# Definimos los símbolos B y C
B, C = sy.symbols('B C')

# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = A * m * x ** (m-1) - B * n * x ** (n-1)

file_answer.write('4c', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$Amx^{m-1} - Bnx^{n-1}$$

```
quizz.eval_expression('4c', respuesta)
```

4c | Tu respuesta:
es correcta.

$$A m x^{m-1} - B n x^{n-1}$$

5. Producto de funciones

NOTA: Reduce la solución a su mínima expresión

5. a. $f(x) = (x^4)(x^{-2})$, $f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = 2 * x

file_answer.write('5a', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

2x

```
quizz.eval_expression('5a', respuesta)
```

5a | Tu respuesta:
es correcta.

2x

5. b. $f(x) = \sin(x) \cos(x)$, $f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = -sy.sin(x)**2 + sy.cos(x)**2

file_answer.write('5b', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$-\sin^2(x) + \cos^2(x)$$

```
quizz.eval_expression('5b', respuesta)
```

 5b | Tu respuesta:
 es correcta.

$$-\sin^2(x) + \cos^2(x)$$

6. Cociente de funciones

Nota: Reduce la expresión del numerador

Formato: (f(x))/(g(x))

6. a. $f(x) = \frac{\sin(x)}{x}, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = sy.cos(x) / x - sy.sin(x) / x**2

file_answer.write('6a', str(respuesta))
#### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
 Respuestas y retroalimentación almacenadas.

$$\frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$$

```
quizz.eval_expression('6a', respuesta)
```

 6a | Tu respuesta:
 es correcta.

$$\frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$$

6. b. $f(x) = \frac{1}{x^2 + x + 1}, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = (-2*x-1) / (x**2 + x + 1) ** 2

file_answer.write('6b', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$\frac{-2x - 1}{(x^2 + x + 1)^2}$$

```
quizz.eval_expression('6b', respuesta)
```

6b | Tu respuesta:
es correcta.

$$\frac{-2x - 1}{(x^2 + x + 1)^2}$$

7. Regla de la Cadena

7. a. $f(x) = (5x^2 + 2x)^2, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = (20*x+4)*(5*x**2+2*x)

file_answer.write('7a', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$(20x + 4)(5x^2 + 2x)$$

```
quizz.eval_expression('7a', respuesta)
```

 7a | Tu respuesta:
 es correcta.

$$(20x + 4)(5x^2 + 2x)$$

7. b. $f(x) = \cos(x^2 + 3)$, $f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = -2*x*sy.sin(x**2+3)

file_answer.write('7b', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
 Respuestas y retroalimentación almacenadas.

$$-2x \sin(x^2 + 3)$$

```
quizz.eval_expression('7b', respuesta)
```

 7b | Tu respuesta:
 es correcta.

$$-2x \sin(x^2 + 3)$$

8. Derivadas de alto orden

Calcular la primera, segunda, tercera y cuarta derivada de $f(x) = 3x^4 + 2x^2 - 20$.

8. a. $f(x) = 3x^4 + 2x^2 - 20$, $f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...
```

```

#### BEGIN SOLUTION
respuesta = 12 * x**3 + 4*x

file_answer.write('8a', str(respuesta))
#### END SOLUTION

display(respuesta)

```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$12x^3 + 4x$$

```
quizz.eval_expression('8a', respuesta)
```

8a | Tu respuesta:
es correcta.

$$12x^3 + 4x$$

8. b. $f(x) = 3x^4 + 2x^2 - 20$, $f''(x) = ?$

```

# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = 36 * x**2 + 4

file_answer.write('8b', str(respuesta))
#### END SOLUTION

display(respuesta)

```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$36x^2 + 4$$

```
quizz.eval_expression('8b', respuesta)
```

8b | Tu respuesta:
es correcta.

$$36x^2 + 4$$

8. c. $f(x) = 3x^4 + 2x^2 - 20$, $f'''(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = 72 * x

file_answer.write('8c', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$72x$$

```
quizz.eval_expression('8c', respuesta)
```

8c | Tu respuesta:
es correcta.

$$72x$$

8. d. $f(x) = 3x^4 + 2x^2 - 20$, $f''''(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = 72

file_answer.write('8d', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$72$$

```
quizz.eval_expression('8d', respuesta)
```

8d | Tu respuesta:
es correcta.

72

Realiza las gráficas de las cuatro derivadas y observa su comportamiento.

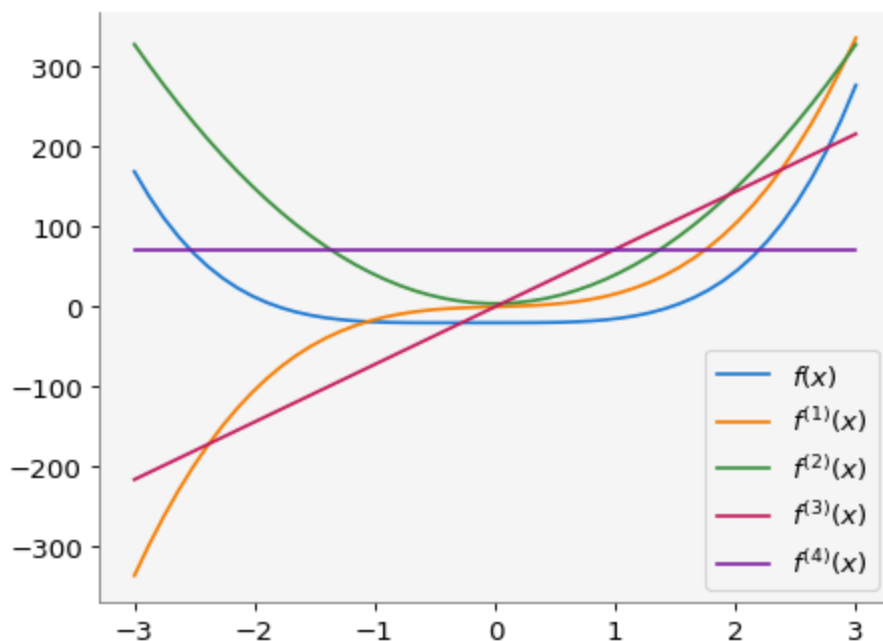
```
# Definimos la función y sus cuatro derivadas
f = lambda x: 3*x**4 + 2*x**3 -20

### BEGIN SOLUTION
f1 = lambda x: 12*x**3 + 4*x
f2 = lambda x: 36*x**2 + 4
f3 = lambda x: 72*x
f4 = lambda x: 72*np.ones(len(x))
### END SOLUTION
# f1 = lambda x: ...
# f2 = lambda x: ...
# f3 = lambda x: ...
# f4 = lambda x: ...

xc = np.linspace(-3, 3, 50) # Codominio de la función

# Graficamos la función y sus derivadas
plt.title('$f(x)=3x^4 + 2x^3 -20$ y sus derivadas')
plt.plot(xc, f(xc), label='$f(x)$')
plt.plot(xc, f1(xc), label='$f^{(1)}(x)$')
plt.plot(xc, f2(xc), label='$f^{(2)}(x)$')
plt.plot(xc, f3(xc), label='$f^{(3)}(x)$')
plt.plot(xc, f4(xc), label='$f^{(4)}(x)$')
plt.legend()
plt.show()
```


$f(x) = 3x^4 + 2x^3 - 20$ y sus derivadas



Encuentra la primera y segunda derivada de la siguientes funciones: - a) $f(x) = x^5 - 2x^3 + x$ - b) $f(x) = 4 \cos x^2$

8. e. $f(x) = x^5 - 2x^3 + x$, $f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

### BEGIN SOLUTION
respuesta = 5*x**4-6*x**2+1

file_answer.write('8e', str(respuesta))
### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$5x^4 - 6x^2 + 1$$

```
quizz.eval_expression('8e', respuesta)
```

8e | Tu respuesta:
es correcta.

$$5x^4 - 6x^2 + 1$$

8. f. $f(x) = x^5 - 2x^3 + x, f''(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = 20*x**3-12*x

file_answer.write('8f', str(respuesta))
#### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$20x^3 - 12x$$

```
quizz.eval_expression('8f', respuesta)
```

8f | Tu respuesta:
es correcta.

$$20x^3 - 12x$$

8. g. $f(x) = 4 \cos x^2, f'(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = -8 * x * sy.sin(x**2)

file_answer.write('8g', str(respuesta))
#### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
Respuestas y retroalimentación almacenadas.

$$-8x \sin(x^2)$$

```
quizz.eval_expression('8g', respuesta)
```

 8g | Tu respuesta:
 es correcta.

$$-8x \sin(x^2)$$

8. h. $f(x) = 4 \cos x^2$, $f''(x) = ?$

```
# Escribe tu respuesta como sigue
# respuesta = ...

#### BEGIN SOLUTION
respuesta = -8*sy.sin(x**2) - 16*x**2*sy.cos(x**2)

file_answer.write('8h', str(respuesta))
#### END SOLUTION

display(respuesta)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe
 Respuestas y retroalimentación almacenadas.

$$-16x^2 \cos(x^2) - 8 \sin(x^2)$$

```
quizz.eval_expression('8h', respuesta)
```

 8h | Tu respuesta:
 es correcta.

$$-16x^2 \cos(x^2) - 8 \sin(x^2)$$

Realiza las gráficas de las dos funciones y de su primera y segunda derivadas.

```
f = lambda x: x**5 - 2*x**3 + x

#### BEGIN SOLUTION
f1 = lambda x: 5*x**4 - 6*x**2 + 1
f2 = lambda x: 20*x**3 - 12*x
#### END SOLUTION
# f1 = lambda x: ...
# f2 = lambda x: ...

# Definimos la segunda función y sus derivadas
g = lambda x: 4*np.cos(x**2)

#### BEGIN SOLUTION
```

```

g1 = lambda x: -8*x*np.sin(x**2)
g2 = lambda x: -8*np.sin(x**2) - 16*x**2*np.cos(x**2)
### END SOLUTION
# g1 = lambda x: ...
# g2 = lambda x: ...

xc = np.linspace(-3, 3, 50) # Codominio de las funciones

# Graficamos las funciones y sus derivadas
plt.figure(figsize=(16,6))
ax1 = plt.subplot(1,2,1)
ax2 = plt.subplot(1,2,2)

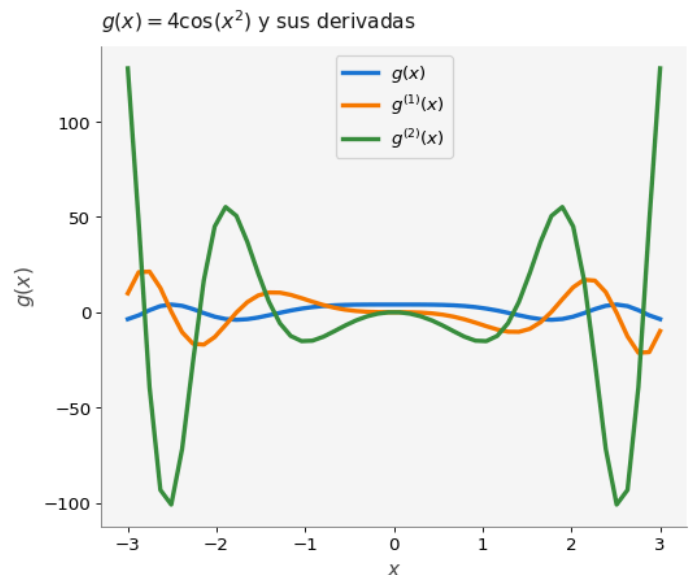
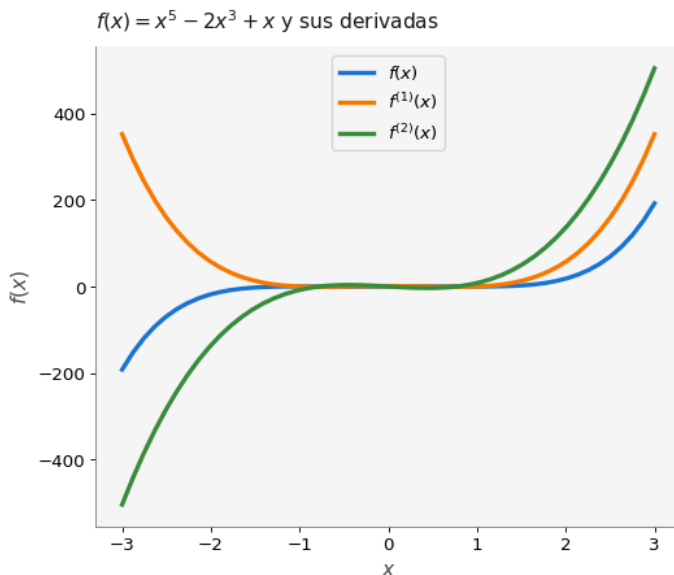
ax1.plot(xc, f(xc), label='$f(x)$',lw=3)
ax1.plot(xc, f1(xc), label='$f^{(1)}(x)$',lw=3)
ax1.plot(xc, f2(xc), label='$f^{(2)}(x)$',lw=3)
ax1.legend(loc='upper center')
ax1.set_title('$f(x)=x^5 - 2x^3 + x$ y sus derivadas')
ax1.set_xlabel

ax2.plot(xc, g(xc), label='$g(x)$',lw=3)
ax2.plot(xc, g1(xc), label='$g^{(1)}(x)$',lw=3)
ax2.plot(xc, g2(xc), label='$g^{(2)}(x)$',lw=3)
ax2.legend(loc='upper center')
ax2.set_title('$g(x)=4\cos(x^2)$ y sus derivadas')

ax1.set_xlabel("$x$")
ax1.set_ylabel("$f(x)$")
ax2.set_xlabel("$x$")
ax2.set_ylabel("$g(x)$")

plt.show()

```



9. Aplicación de la regla de L'Hopital

Utilizando la regla de L'Hopital encuentra el límite de $f(x) = \frac{\sin(x)}{x}$ cuando x tiende a cero.

Solución.

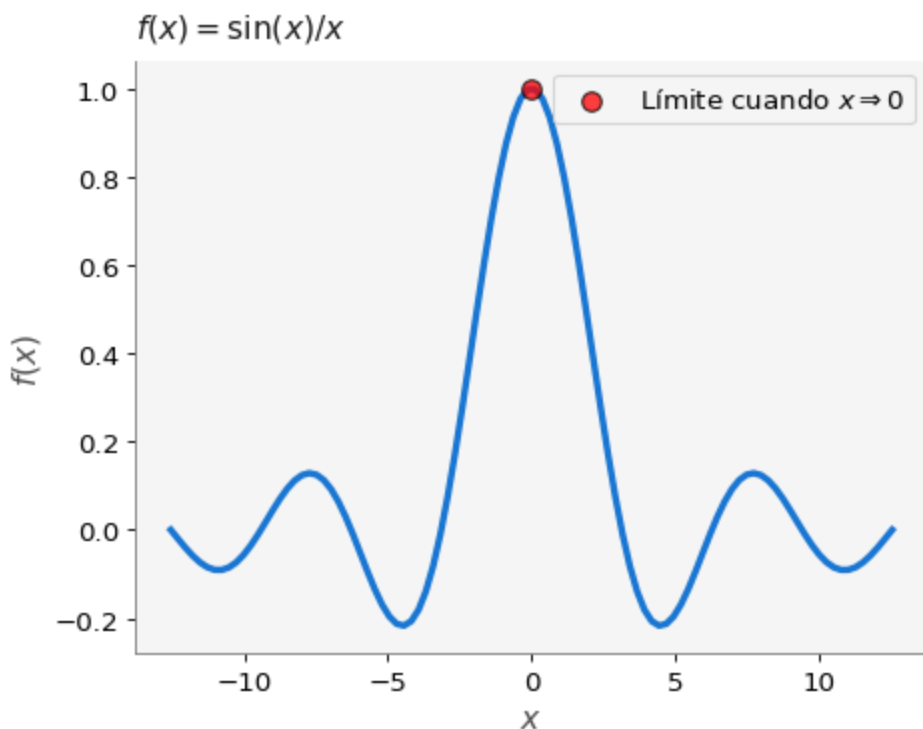
Al cumplirse las condiciones de la regla podemos asegurar que:

$$\lim_{x \rightarrow 0} \frac{\sin(x)}{x} = \lim_{x \rightarrow 0} \frac{\sin'(x)}{x'} = \lim_{x \rightarrow 0} \frac{\cos(x)}{1} = 1$$

```
f = lambda x: np.sin(x) / x

x = np.linspace(-4*np.pi, 4*np.pi, num=100) # Codominio de la función

# Graficamos la función y el punto (0, f(0))
plt.title('$f(x)=\sin(x) / x$')
plt.ylabel("$f(x)$")
plt.xlabel("$x$")
plt.plot(x, f(x), lw=3)
plt.scatter(0, 1, label='Límite cuando $x \rightarrow 0$', fc='red', ec='black',
plt.legend()
plt.show()
```



10. Ejemplo del teorema de Rolle. Considere la función $f(x) = x^2 + 5$, la cual es continua en todo \mathbb{R} . Tomemos el intervalo $[-5, 5]$ y hagamos la gráfica de esta función. Observe en la gráfica que sigue, que se cumplen las condiciones del Teorema de Rolle y por lo tanto es posible encontrar un punto c , punto rojo, donde la derivada es cero (línea roja).

```

# Dominio e imagen de la gráfica
xc = np.linspace(-10,10,200)
f = lambda i: i**2 + 5

# Configuración de la grafica
plt.xticks(range(-10,11,5))
plt.yticks(range(-10,110,10))
plt.xlabel("$x$",)
plt.ylabel("$f(x)$")
plt.title("$f(x)=x^{2}+5$")

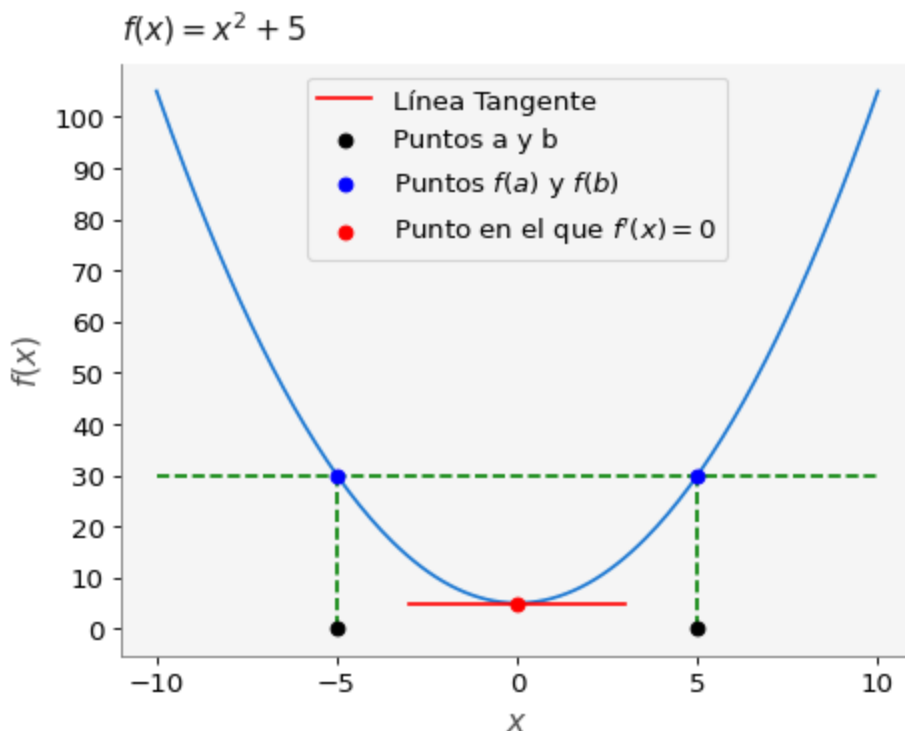
# Función
plt.plot(xc,f(xc))

# Dibujamos algunas líneas en la gráfica
plt.plot(np.linspace(-10,10,2),[f(5)]*2,ls="dashed",color="green")
plt.plot((5,5),(0,f(5)),ls="dashed",color="green")
plt.plot((-5,-5),(0,f(5)),ls="dashed",color="green")
plt.plot((-3,3),(5,5),color="red",label="Línea Tangente")

# Dibujamos algunos puntos en la gráfica
plt.scatter((-5,5),(0,0),color="black",label="Puntos a y b",zorder=5)
plt.scatter((-5,5),(f(-5),f(5)),color="blue",label="Puntos $f(a)$ y $f(b)$",zorder=5)
plt.scatter(0,f(0),color="red",label="Punto en el que $f'(x)=0$",zorder=5)

plt.legend(loc="upper center")
plt.show()

```



Reglas de derivación

En general no es complicado calcular la derivada de cualquier función y existen reglas para hacerlo más fácil.

Regla de potencias

Para cualquier número real n si $f(x) = x^n$, entonces

$$f'(x) = nx^{n-1}$$

Regla de la función constante

Si $f(x) = c$ es una función constante, entonces

$$f'(x) = 0$$

Regla de la multiplicación por constante

Si c es cualquier constante y $f(x)$ es diferenciable, entonces $g(x) = cf(x)$ también es diferenciable y su derivada es:

$$g'(x) = cf'(x)$$

Regla de suma y diferencia

Si $f(x)$ y $g(x)$ son diferenciables, entonces $f(x) + g(x)$ y $f(x) - g(x)$ también son diferenciables y sus derivadas son:

$$[f(x) + g(x)]' = f'(x) + g'(x)$$

$$[f(x) - g(x)]' = f'(x) - g'(x)$$

Regla del producto

Si $f(x)$ y $g(x)$ son funciones diferenciables, entonces $f(x)g(x)$ es diferenciable y su derivada es:

$$[f(x)g(x)]' = f(x)g'(x) + g(x)f'(x)$$

Regla del cociente

Si f y g son funciones diferenciables y $g(x) \neq 0$, entonces $f(x)/g(x)$ es diferenciable y su derivada es:

$$\left[\frac{f(x)}{g(x)} \right]' = \frac{f(x)g'(x) - f'(x)g(x)}{g(x)^2}$$

Regla de la cadena

Si la función $f(u)$ es diferenciable, donde $u = g(x)$, y la función $g(x)$ es diferenciable, entonces la composición $y = (f \circ g)(x) = f(g(x))$ es diferenciable:

$$f(g(x))' = f'(g(x)) \cdot g'(x)$$

Regla de L'Hôpital

Esta regla es utilizada en caso de indeterminaciones donde $f(x)$ y $g(x)$ son dos funciones continuas definidas en el intervalo $[a, b]$, derivables en (a, b) y sea c perteneciente a (a, b) tal que $f(c) = g(c) = 0$ y $g'(x) \neq 0$ si $x \neq c$. Si existe el límite L de f'/g' en c , entonces existe el límite de $f(x)/g(x)$ (en c) y es igual a L . Por lo tanto:

$$\lim_{x \rightarrow c} \frac{f(x)}{g(x)} = \lim_{x \rightarrow c} \frac{f'(x)}{g'(x)} = L$$

Derivadas de funciones trigonométricas

$$\sin'(x) = \cos(x)$$

$$\cos'(x) = -\sin(x)$$

$$\tan'(x) = \sec^2(x)$$

$$\sec'(x) = \sec(x) \tan(x)$$

$$\cot'(x) = -\csc^2(x)$$

$$\csc'(x) = -\csc(x) \cot(x)$$

Derivada la función exponencial

$$[e^x]' = e^x$$

Teorema de Rolle : Sea $a < b$ y suponga que $f : [a, b] \rightarrow \mathbb{R}$ es derivable en (a, b) y continua en $[a, b]$ y $f(a) = f(b)$. Entonces $\exists x_0 \in (a, b)$ tal que $f'(x_0) = 0$

Lo anterior quiere decir que, dadas las condiciones del teorema, es posible encontrar un punto de la función $f(x)$ dentro del intervalo (a, b) donde la derivada es cero; en otras palabras, en ese punto de la función la línea tangente es horizontal

Derivadas de orden superior

Es posible obtener la derivada de la derivada, es decir, si tenemos una función $f(x)$ cuya derivada es $f'(x)$, entonces podemos calcular la derivada a esta última función, para obtener $f''(x)$, a esta última función, si es que existe, se le conoce como la segunda derivada de $f(x)$. También se puede denotar a la segunda derivada con $f^{(2)}(x)$.

En general, si $f(x)$ es derivable k veces, entonces es posible obtener la k -ésima derivada de dicha función, que se escribe como:


$$\frac{d^k f(x)}{dx^k} = f^{(k)}(x)$$



3 Diferencias finitas: cálculo del error

Objetivo general - Implementar varias fórmulas de aproximación de la primera derivada y compararlas entre ellas mediante el cálculo del error.

Objetivos particulares - Revisar las fórmulas de aproximación de la primera derivada: Forward, Backward, Central. - Implementar funciones para calcular las fórmulas. - Calcular el error que introducen estas fórmulas. - Mostrar de manera gráfica el error. - Implementar funciones de varios órdenes para compararlas con las fórmulas anteriores.

[MACTI-Analysis_Numerico_01](#) by [Luis M. de la Cruz](#) is licensed under [Attribution-ShareAlike 4.0 International](#) 

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import macti.visual as mvis
from macti.evaluation import *
```

```
quizz = Quizz("q3", "notebooks", "local")
```

Introducción

La siguiente herramienta tiene como propósito mostrar diferentes funciones y sus derivadas exactas así como el cálculo numérico de las derivadas usando varias aproximaciones. Puedes elegir la función y el tipo de aproximación. Después, puedes mover el punto donde se realiza la aproximación (punto azul) y el tamaño de la h .

```
%run "./zinteractivo3.ipynb"
```

```
<function FD.numericalDer(f, x0, h, aprox='All')>
```

Diferencias finitas hacia adelante (Forward).

$\$_{h}$

La siguiente función de Python implementa la aproximación de **diferencias finitas hacia adelante**.

```
def forwardFD(u,x,h):
    """
    Esquema de diferencias finitas hacia adelante.

    Parameters
    -----
    u : función.
```

Función a evaluar.

`x : array`

Lugar(es) donde se evalúa la función

`h : array`

Tamaño(s) de la diferencia entre $u(x+h)$ y $u(x)$.

Returns

Cálculo de la derivada numérica hacia adelante.

"""

`return (u(x+h)-u(x))/h`

3.1 Ejemplo 1.

La derivada de $\sin(x)$ es $\frac{d \sin(x)}{dx} = \cos(x)$. Si evaluamos la derivada en $x = 1$ obtenemos:
 $\cos(1.0) = 0.5403023058681398$.

Vamos a aproximar este valor usando diferencias finitas hacia adelante con la función `forwardFD()`. Dado que esta aproximación será mejor cuando $h \rightarrow 0$, usaremos el siguiente conjunto de valores h para hacer varias aproximaciones:

$$H = \{h | h = \frac{1}{2^i} \text{ para } i = 1, \dots, 5\}$$

$$= \{1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125\}$$

```
# Definimos un arreglo con diferentes tamaños de h:
N = 6
h = np.array([1 / 2**i for i in range(0,N)])

# Definimos un arreglo con valores de 1.0 (donde evaluaremos el cos(x)):
x = np.ones(N)

print('h = {}'.format(h))
print('x = {}'.format(x))
```

```
h = [1.      0.5     0.25    0.125   0.0625  0.03125]
x = [1.  1.  1.  1.  1.  1.]
```

Ahora usamos la función `forwardFD()` para aproximar la derivada de la función $\sin(x = 1.0)$:

```
forwardFD(np.sin, x, h)
```

```
array([0.06782644, 0.312048 , 0.43005454, 0.48637287, 0.51366321,
       0.52706746])
```

El **error absoluto** entre la derivada exacta y la aproximación se puede calcular usando la fórmula:

$$Error = ||\cos(x) - D_+ \sin(x)||$$

donde D_+ representa la aplicación de la fórmula hacia adelante. Recuerda que la derivada de $\sin(x)$ es $\cos(x)$.

```
# Calculamos el error entre la derivada exacta y la derivada numérica:
ef = np.fabs(np.cos(x) - forwardFD(np.sin, x, h) )
print(ef)
```

```
[0.47247586 0.2282543 0.11024777 0.05392943 0.0266391 0.01323485]
```

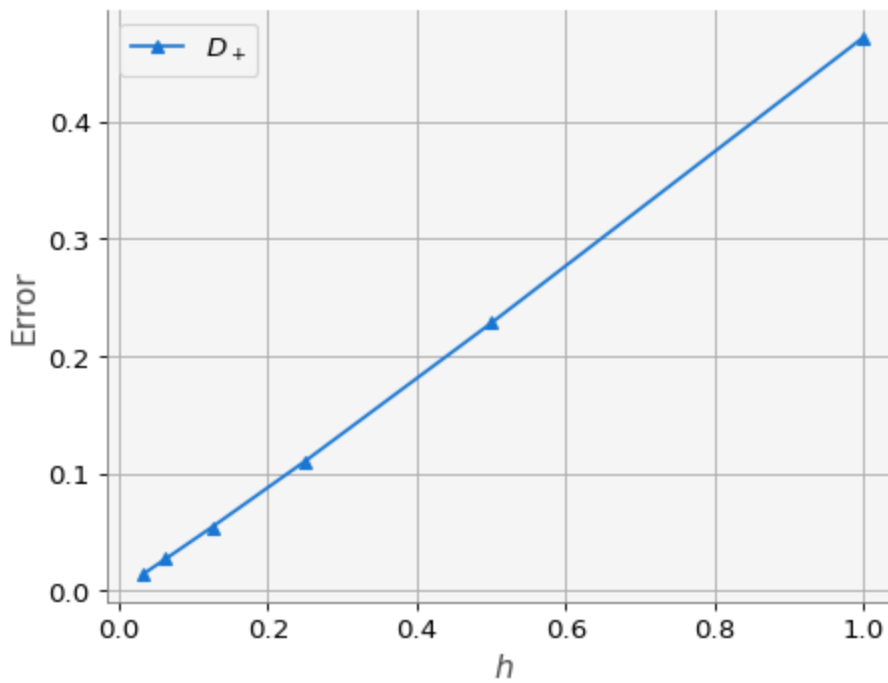
```
# Colocamos la información de h y del error en un Dataframe y mostramos el result
Error = pd.DataFrame(np.array([h, ef]).T,
                      columns=['$h$', '$D_+$'])

Error
```

	\$h\$	\$D_+\$
0	1.00000	0.472476
1	0.50000	0.228254
2	0.25000	0.110248
3	0.12500	0.053929
4	0.06250	0.026639
5	0.03125	0.013235

```
# Hacemos el gráfico del error vs h
plt.plot(h, ef, '^-', label='$D_+$')
plt.xlabel('$h$')
plt.ylabel('Error')
plt.title('Aproximación de la derivada')
plt.legend()
plt.grid()
plt.show()
```

Aproximación de la derivada



Diferencias finitas hacia atrás (Backward).

$\$_{\{h\}} \$$

La siguiente función de Python implementa la aproximación de **diferencias finitas hacia atrás**.

```
def backwardFD(u,x,h):
    """
    Esquema de diferencias finitas hacia atrás.

    Parameters
    -----
    u : función.
    Función a evaluar.

    x : array
    Lugar(es) donde se evalúa la función

    h : array
    Tamaño(s) de la diferencia entre u(x+h) y u(x).

    Returns
    -----
    Cálculo de la derivada numérica hacia atrás.
    """
    return (u(x)-u(x-h))/h
```

3.2 Ejercicio 1.

Tomando como base el ejemplo de diferencias finitas hacia adelante, calcula el error entre la derivada exacta y la aproximación con diferencias finitas hacia atrás usando la fórmula:

$$Error = ||\cos(x) - D_- \sin(x)||$$

donde D_- representa la aplicación de la fórmula hacia atrás.

```
# Calculamos el error entre la derivada exacta y la derivada numérica:
### BEGIN SOLUTION
eb = np.fabs( np.cos(x) - backwardFD(np.sin,x,h) )
file_answer = FileAnswer()
file_answer.write('1', eb, 'La implementación del error no es correcta, checa tan
### END SOLUTION

print(eb)
```

Creando el directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/`
Respuestas y retroalimentación almacenadas.

[0.30116868 0.18378859 0.09902659 0.05111755 0.02593572 0.01305898]

```
quizz.eval_numeric('1', eb)
```

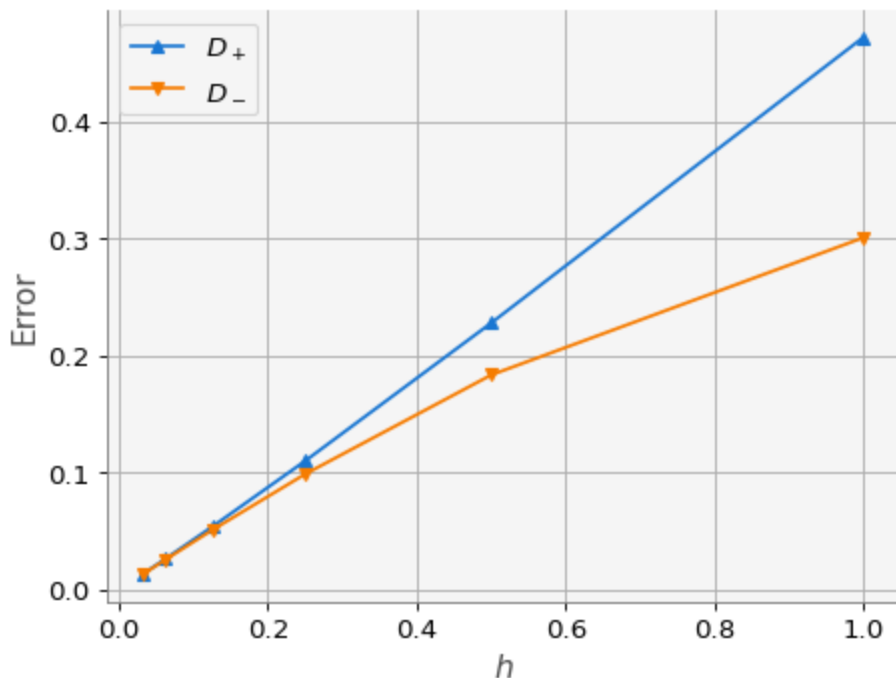
1 | Tu resultado es correcto.

```
# Agregamos la columna del error de diferencias finitas hacia atrás
Error['$D_-'] = eb
Error
```

	\$h\$	\$D_+\$	\$D_-\$
0	1.00000	0.472476	0.301169
1	0.50000	0.228254	0.183789
2	0.25000	0.110248	0.099027
3	0.12500	0.053929	0.051118
4	0.06250	0.026639	0.025936
5	0.03125	0.013235	0.013059

```
# Hacemos el gráfico del error vs h
plt.plot(h, ef, '^-', label='$D_+$')
plt.plot(h, eb, 'v-', label='$D_-$')
plt.xlabel('$h$')
plt.ylabel('Error')
plt.title('Aproximación de la derivada')
plt.legend()
plt.grid()
plt.show()
```

Aproximación de la derivada



Diferencias finitas centradas.

Δ_h

La siguiente función de Python implementa la aproximación de **diferencias finitas centradas**.

```
def centeredFD(u,x,h):
    """
    Esquema de diferencias finitas centradas.

    Parameters
    -----
    u : función.
        Función a evaluar.

    x : array
        Lugar(es) donde se evalúa la función

    h : array
```

Tamaño(s) de la diferencia entre $u(x+h)$ y $u(x)$.

Returns

Cálculo de la derivada numérica centrada.

++++

return (u(x+h)-u(x-h))/(2*h)

3.3 Ejercicio 2.

Tomando como base el ejercicio 1, calcula el error entre la derivada exacta y la aproximación con diferencias finitas centradas usando la fórmula:

$$Error = ||\cos(x) - D_0 \sin(x)||$$

donde D_0 representa la aplicación de la fórmula de diferencias centradas.

```
# Metemos la información de h y del error en un Dataframe y mostramos el resultado
### BEGIN SOLUTION
# Calculamos el error entre la derivada exacta y la derivada numérica:
ec = np.fabs( np.cos(x) - centeredFD(np.sin,x,h) )

file_answer.write('2', ec, 'La implementación del error no es correcta, checka tu código')
### END SOLUTION

print(ec)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe. Respuestas y retroalimentación almacenadas.

```
[8.56535925e-02 2.22328579e-02 5.61058720e-03 1.40593842e-03
 3.51690617e-04 8.79355346e-05]
```

```
quizz.eval_numeric('2', ec)
```

2 | Tu resultado es correcto.

3.4 Ejercicio 3.

Tomando como base los ejemplos de diferencias finitas hacia adelante y hacia atrás, agrega una columna con los resultados del error de la aproximación de diferencias centradas en el DataFrame `Error`.

```
# Agregamos la columna del error de diferencias finitas centradas
# Error['...'] = ...

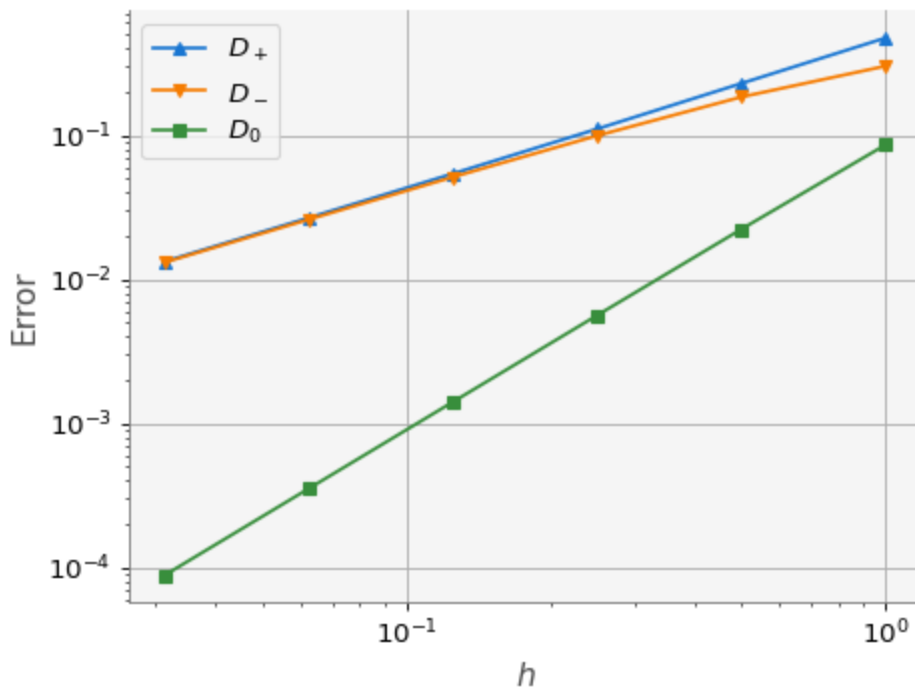
#### BEGIN SOLUTION
Error['$D_0$'] = ec
Error
#### END SOLUTION
Error
```

	<code>\$h\$</code>	<code>\$D_+\$</code>	<code>\$D_-\$</code>	<code>\$D_0\$</code>
0	1.00000	0.472476	0.301169	0.085654
1	0.50000	0.228254	0.183789	0.022233
2	0.25000	0.110248	0.099027	0.005611
3	0.12500	0.053929	0.051118	0.001406
4	0.06250	0.026639	0.025936	0.000352
5	0.03125	0.013235	0.013059	0.000088

Observe que en este caso los errores son varios órdenes de magnitud más pequeños. Para hacer una gráfica más representativa usaremos escala `loglog`:

```
# Hacemos el gráfico del error vs h
plt.plot(h, ef, '^-', label='$D_+$')
plt.plot(h, eb, 'v-', label='$D_-$')
plt.plot(h, ec, 's-', label='$D_0$')
plt.xlabel('$h$')
plt.ylabel('Error')
plt.title('Aproximación de la derivada')
plt.legend()
plt.grid()
plt.loglog() # Definimos la escala log-log
plt.show()
```


Aproximación de la derivada



Como se puede apreciar, la gráfica anterior muestra que la aproximación con diferencias finitas centradas es mejor, pues es de orden cuadrático.

3.5 Ejercicio 4. Aproximación con cuatro puntos

Implementar a siguiente fórmula de aproximación para el cálculo de la primera derivada y usarla para calcular la derivada del $\sin(x)$ en $x = 1.0$ y compararla con las anteriores calculando el error y graficando.

$$D_3u = \frac{1}{6h} [2u_{i+1} + 3u_i - 6u_{i-1} + u_{i-2}]$$

Hint: Recuerde que $u_i = u(x)$, $u_{i+1} = u(x + h)$, $u_{i-1} = u(x - h)$ y $u_{i-2} = u(x - 2h)$.

```
# Implementación de D3
def D3(u,x,h):
    ### BEGIN SOLUTION
    return (2*u(x+h)+3*u(x)-6*u(x-h)+u(x-2*h)) / (6*h)
    ### END SOLUTION
```

```
### BEGIN SOLUTION
# Calculamos el error entre la derivada exacta y la derivada numérica:
e3 = np.fabs( np.cos(x) - D3(np.sin,x,h) )

file_answer.write('3', e3, 'La implementación del error no es correcta, chequea tan
```

```
#### END SOLUTION
```

```
print(e3)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe
Respuestas y retroalimentación almacenadas.

```
[4.32871647e-02 7.31425947e-03 1.01447520e-03 1.32213104e-04
 1.68339444e-05 2.12244935e-06]
```

```
quizz.eval_numeric('3', e3)
```

3 | Tu resultado es correcto.

3.6 Ejercicio 5.

Tomando como base los ejemplos de diferencias finitas anteriores, agrega una columna con los resultados del error de la aproximación de diferencias con cuatro puntos en el DataFrame **Error**.

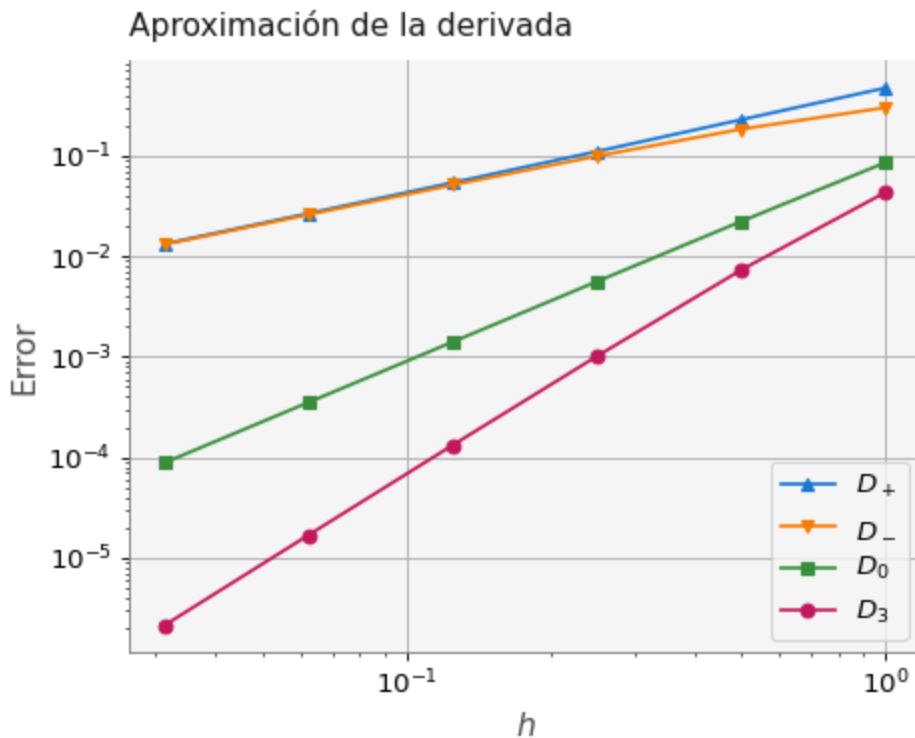
```
# Agregamos la columna del error de diferencias finitas centradas
#### BEGIN SOLUTION
Error['$D_3$'] = e3

#### END SOLUTION
Error
```

	\$h\$	\$D_+\$	\$D_-\$	\$D_0\$	\$D_3\$
0	1.00000	0.472476	0.301169	0.085654	0.043287
1	0.50000	0.228254	0.183789	0.022233	0.007314
2	0.25000	0.110248	0.099027	0.005611	0.001014
3	0.12500	0.053929	0.051118	0.001406	0.000132
4	0.06250	0.026639	0.025936	0.000352	0.000017
5	0.03125	0.013235	0.013059	0.000088	0.000002

```
# Hacemos el gráfico del error vs h
plt.plot(h, ef, '^-', label='$D_+$')
plt.plot(h, eb, 'v-', label='$D_-$')
plt.plot(h, ec, 's-', label='$D_0$')
plt.plot(h, e3, 'o-', label='$D_3$')
```

```
plt.xlabel('$h$')
plt.ylabel('Error')
plt.title('Aproximación de la derivada')
plt.legend()
plt.loglog() # Definimos la escala log-log
plt.grid()
plt.show()
```



3.7 Ejercicio 6. Aproximación con tres puntos (left).

Implementar a siguiente fórmula de aproximación para el cálculo de la primera derivada y usarla para calcular la derivada del $\sin(x)$ en $x = 1.0$ y compararla con las anteriores.

$$D_{2l}f' = \frac{3f_i - 4f_{i-1} + f_{i-2}}{2h}$$

```
# Implementación
def D2l(u,x,h):
    ### BEGIN SOLUTION
    return (3*u(x) - 4*u(x-h) + u(x-2*h)) / (2*h)
    ### END SOLUTION
```

```
### BEGIN SOLUTION
# Calculamos el error entre la derivada exacta y la derivada numérica:
```

```
e2l = np.fabs( np.cos(x) - D2l(np.sin,x,h) )

file_answer.write('4', e2l, 'La implementación del error no es correcta, checa ta
#### END SOLUTION

print(e2l)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe Respuestas y retroalimentación almacenadas.

```
[3.01168679e-01 6.64084941e-02 1.42646000e-02 3.20851614e-03
 7.53883067e-04 1.82238417e-04]
```

```
quizz.eval_numeric('4', e2l)
```

4 | Tu resultado es correcto.

3.8 Ejercicio 7.

Tomando como base los ejemplos de diferencias finitas anteriores, agrega una columna con los resultados del error de la aproximación de diferencias con tres puntos-left en el DataFrame `Error`.

```
# Colocamos la información de h y del error en un Dataframe y mostramos el result
#### BEGIN SOLUTION
Error['$D_{2l}$']=e2l

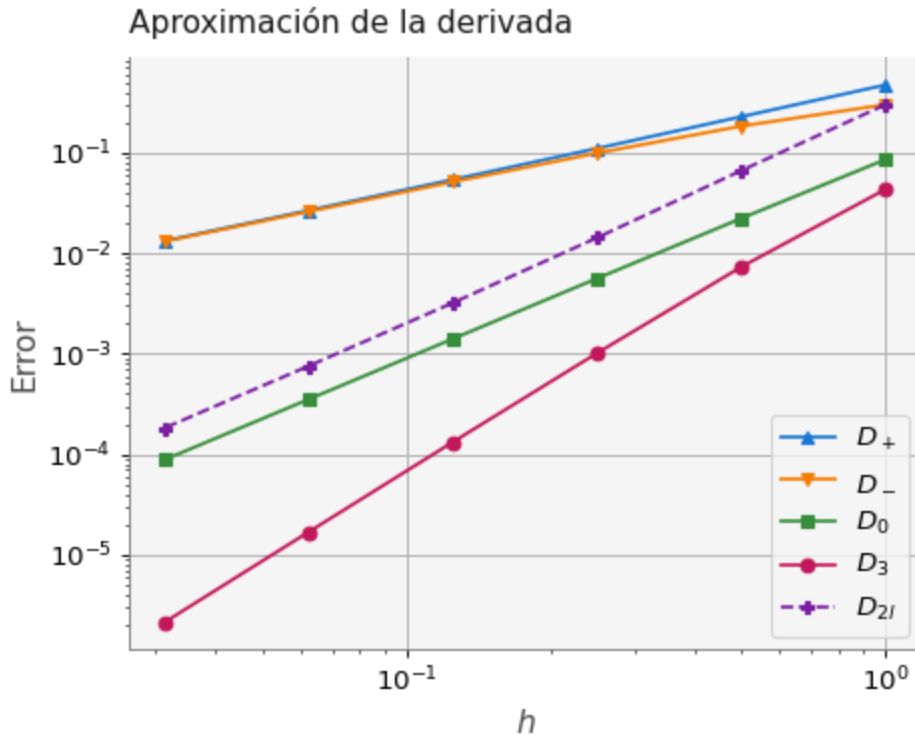
#### END SOLUTION
Error
```

	<code>\$h\$</code>	<code>\$D_+\$</code>	<code>\$D_-\$</code>	<code>\$D_0\$</code>	<code>\$D_3\$</code>	<code>\$D_{2l}\$</code>
0	1.00000	0.472476	0.301169	0.085654	0.043287	0.301169
1	0.50000	0.228254	0.183789	0.022233	0.007314	0.066408
2	0.25000	0.110248	0.099027	0.005611	0.001014	0.014265
3	0.12500	0.053929	0.051118	0.001406	0.000132	0.003209
4	0.06250	0.026639	0.025936	0.000352	0.000017	0.000754
5	0.03125	0.013235	0.013059	0.000088	0.000002	0.000182

```
# Hacemos el gráfico del error vs h
plt.plot(h, ef, '^-', label='$D_+$')
plt.plot(h, eb, 'v-', label='$D_-$')
```

```
plt.plot(h, ec, 's-', label='$D_0$')
plt.plot(h, e3, 'o-', label='$D_3$')
plt.plot(h, e2l, 'P--', label='$D_{2l}$')

plt.xlabel('$h$')
plt.ylabel('Error')
plt.title('Aproximación de la derivada')
plt.legend()
plt.loglog() # Definimos la escala log-log
plt.grid()
plt.show()
```



3.9 Ejercicio 6. Aproximación con tres puntos (right).

Obtener los coeficientes A , B y C para una aproximación del siguiente tipo:

$$D_{2r}f' = Af_i + Bf_{i+1} + Cf_{i+2}$$

y luego implementar la fórmula y graficarla junto con los resultados anteriores.

```
# Implementación
def D2r(u,x,h):
    ### BEGIN SOLUTION
```

```
return (-3*u(x) + 4*u(x+h) - u(x+2*h)) / (2*h)
### END SOLUTION
```

```
### BEGIN SOLUTION
# Calculamos el error entre la derivada exacta y la derivada numérica:
e2r = np.fabs( np.cos(x) - D2r(np.sin,x,h) )

file_answer.write('5', e2r, 'La implementación del error no es correcta, checa ta
file_answer.to_file('q3')
### END SOLUTION

print(e2r)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe Respuestas y retroalimentación almacenadas.

[0.05447393 0.01596726 0.00775877 0.0023889 0.00065123 0.0001694]

```
quizz.eval_numeric('5', e2r)
```

5 | Tu resultado es correcto.

3.10 Ejercicio 7.

Tomando como base los ejemplos de diferencias finitas anteriores, agrega una columna con los resultados del error de la aproximación de diferencias con **tres puntos-right** en el DataFrame **Error**.

```
# Colocamos la información de h y del error en un Dataframe y mostramos el result
### BEGIN SOLUTION
Error['$D_{2r}$'] = e2r

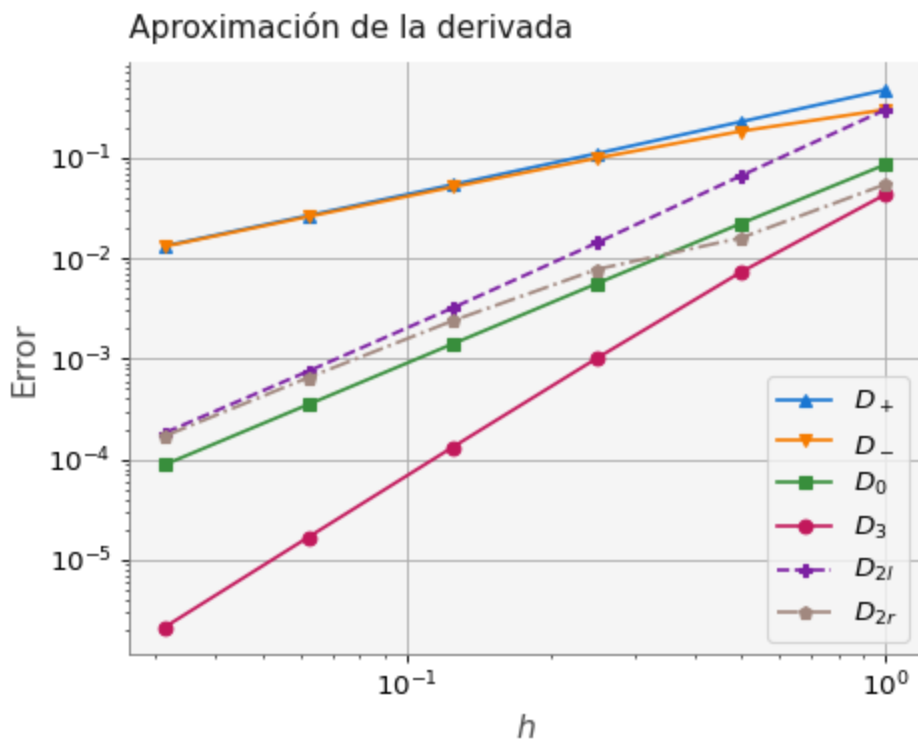
### END SOLUTION
Error
```

	\$h\$	\$D_{+}\$	\$D_{-}\$	\$D_0\$	\$D_3\$	\$D_{2l}\$	\$D_{2r}\$
0	1.00000	0.472476	0.301169	0.085654	0.043287	0.301169	0.054474
1	0.50000	0.228254	0.183789	0.022233	0.007314	0.066408	0.015967
2	0.25000	0.110248	0.099027	0.005611	0.001014	0.014265	0.007759
3	0.12500	0.053929	0.051118	0.001406	0.000132	0.003209	0.002389
4	0.06250	0.026639	0.025936	0.000352	0.000017	0.000754	0.000651

	h	D_+	D_-	D_0	D_3	D_{2l}	D_{2r}
5	0.03125	0.013235	0.013059	0.000088	0.000002	0.000182	0.000169

```
# Hacemos el gráfico del error vs h
plt.plot(h, ef, '^-', label='D+')
plt.plot(h, eb, 'v-', label='D-')
plt.plot(h, ec, 's-', label='D0')
plt.plot(h, e3, 'o-', label='D3')
plt.plot(h, e2l, 'P--', label='D2l')
plt.plot(h, e2r, 'p-.', label='D2r')

plt.xlabel('h')
plt.ylabel('Error')
plt.title('Aproximación de la derivada')
plt.legend()
plt.loglog() # Definimos la escala log-log
plt.grid()
plt.show()
```





7 Conducción de calor 1D: condiciones de tipo Neumann

Objetivo.

Resolver numéricamente el siguiente problema usando diferencias finitas.

$$-\frac{d^2u(x)}{dx^2} = -e^x \quad x \in [0, 1]$$

$$\frac{du}{dn}(0) = 0$$

$$u(1) = 3$$

cuya solución analítica es: $u(x) = e^x - x - e + 4$

Observa que en este caso se proporciona una condición de tipo **Neumann** en la frontera izquierda del dominio ($x = 0$).

[MACTI-Analisis_Numerico_01](#) by [Luis M. de la Cruz](#) is licensed under [Attribution-ShareAlike 4.0 International](#)



Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101019, PE101922

```
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis
from macti.evaluation import *
```

```
quizz = Quizz('q02', 'notebooks', 'local')
```

7.1 Definición de funciones. [↗](#)

```
def buildMatrix(N, d):
    """
    Construye la matriz del sistema.
    """
    # Matriz de ceros
    A = np.zeros((N,N))

    # Primer renglón
    A[0,0] = d
    A[0,1] = -1

    # Renglones interiores
    for i in range(1,N-1):
        A[i,i] = d
        A[i,i+1] = -1
```



```

        A[i,i-1] = -1

    # Último renglón
    A[N-1,N-2] = -1
    A[N-1,N-1] = d

    return A

def solExact(x):
    return np.exp(x) - x - np.e + 4

```

7.2 Definición de parámetros del problema físico y numérico.

```

# Parámetros físicos
L = 1.0
f_A = 0.0 # Flujo en A (Neumman)
b_B = 3.0 # Valor de u en B (Dirichlet)
k = 1.0

# Parámetros numéricos
N = 4 # Número de incógnitas
h = L / (N+1)

# Coordenadas de los nodos
x = np.linspace(0, L, N+2)

# Solución exacta en los nodos
sol_e = solExact(x)

# Construcción de la matriz
A = buildMatrix(N+1, 2)

# Lado derecho del sistema
b = np.zeros(N+1)

# Fuente o sumidero
b[1:] = -np.exp(x[1:-1])*h**2

# Condición de frontera en B
b[-1] += b_B

```

```

print(A)
print(b)

```

```

[[ 2. -1.  0.  0.  0.]
 [-1.  2. -1.  0.  0.]
 [ 0. -1.  2. -1.  0.]
 [ 0.  0. -1.  2. -1.]

```

```
[ 0.  0.  0. -1.  2.]]
[ 0.          -0.04885611 -0.05967299 -0.07288475  2.91097836]
```

7.3 Ejercicio 1.

Definir lo siguiente:

- La función `Neumann_I(A, b, bcond)` que implemente la aproximación I de la presentación “Problemas de Calibración” (página 12).
- Llamada a la función `Neumann_I(A, b, bcond)` con los parámetros adecuados.

```
#### BEGIN SOLUTION
def Neumann_I(A, b, bcond):
    A[0][0] = 1
    A[0][1] = -1
    A[0][2] = 0
    b[0] = bcond

# Corrección de la matriz y el RHS, orden lineal
Neumann_I(A, b, h * f_A)
#### END SOLUTION

# Arreglo para almacenar la solución
u1 = np.zeros(N+2)

# Condición de frontera del lado derecho: Dirichlet
u1[-1] = b_B

# Solución del sistema lineal
u1[:N+1] = np.linalg.solve(A,b)

file_answer = FileAnswer()
file_answer.write('1', u1, 'La solución no es correcta, checa tu implementación c

print('Solución numérica: {}'.format(u1))
```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
Respuestas y retroalimentación almacenadas.

Solución numérica: [2.39076545 2.39076545 2.43962156 2.54815066 2.72956451 3.]

```
quizz.eval_numeric('1', u1)
```

1 | Tu resultado es correcto.

7.4 Ejercicio 2.

- Calcular el error de la solución numérica `u1` con respecto a la solución exacta `sol_e` usando la definición **Grid norm 2** descrita en la “Problemas de Calibración” (página 20).

```
#### BEGIN SOLUTION
# Cálculo del error
e1 = np.sqrt(h) * np.linalg.norm(sol_e - u1, 2)

file_answer.write('2', e1, 'La implementación del error Grid norm 2 parece no ser
#### END SOLUTION

print('Error: {}'.format(e1))
```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
Respuestas y retroalimentación almacenadas.

Error: 0.07266429782458109

```
quizz.eval_numeric('2', e1)
```

2 | Tu resultado es correcto.

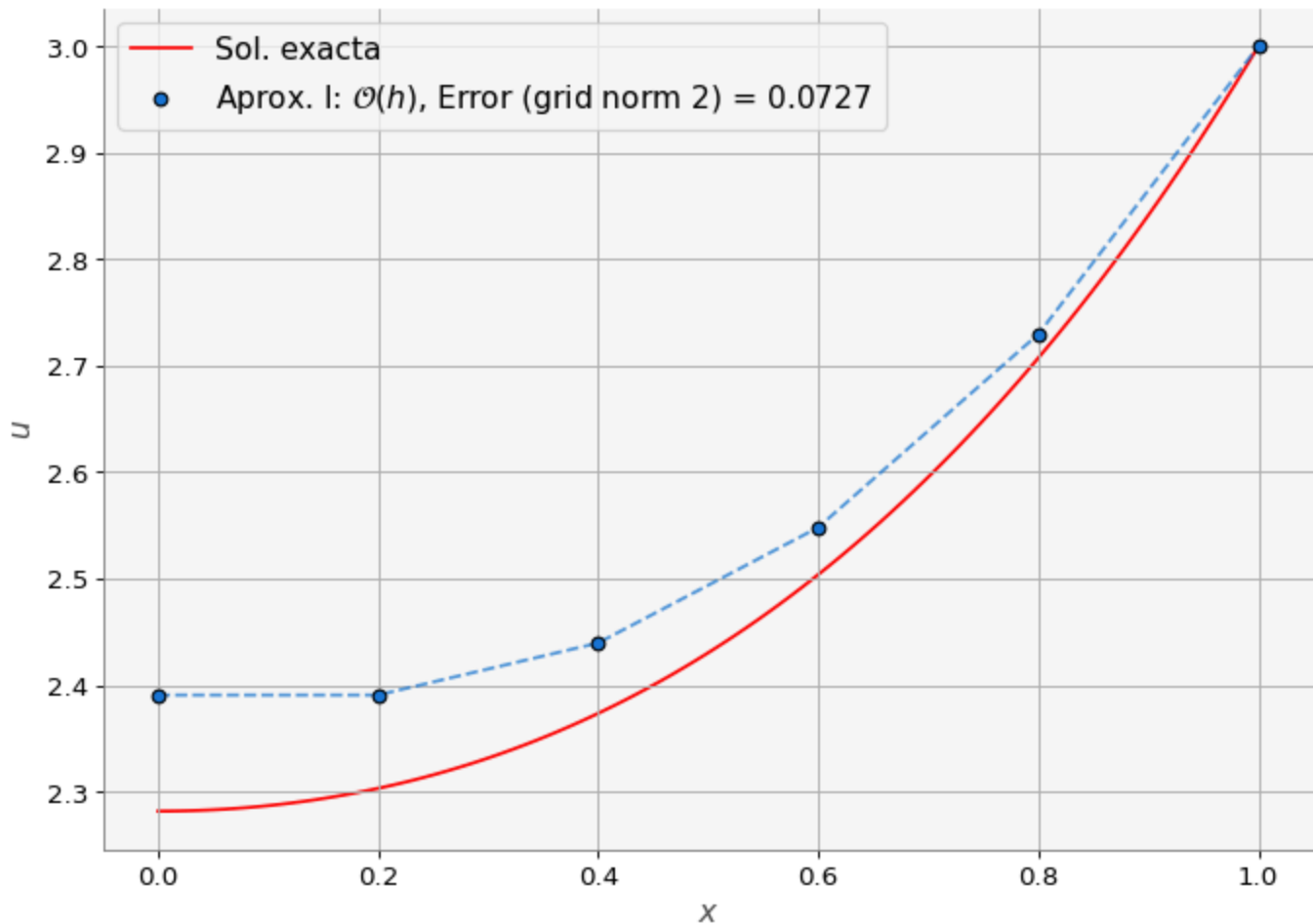
```
# Graficación de la solución y el error
error_label_1 = 'Error (grid norm {}) = {:.4f}'.format(2, e1)

plt.figure(figsize=(10,7))

# Graficación de la solución exacta
xsol = np.linspace(0,1,100)
plt.plot(xsol, solExact(xsol), 'r-', label='Sol. exacta', zorder=0)

# Graficación de la solución numérica
plt.scatter(x, u1, marker='o', edgecolor='k', zorder=5,
            label='Aprox. I:  $\mathcal{U}_0(h)$ , ' + error_label_1)
plt.plot(x, u1, '--', lw=1.5, alpha=0.75)

# Decoración de la gráfica
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend(loc='upper left', fontsize=14)
plt.grid()
plt.show()
```



7.5 Ejercicio 3.

Definir lo siguiente:

- La función `Neumann_II(A, b, bcond)` que implemente la aproximación II de la presentación “Problemas de Calibración” (página 15).
- Llamada a la función `Neumann_II(A, b, bcond)` con los parámetros adecuados.

```

#### BEGIN SOLUTION
def Neumann_II(A, b, bcond):
    A[0][0] = 3
    A[0][1] = -4
    A[0][2] = 1
    b[0] = bcond

    # Corrección de la matriz y el RHS, orden lineal
    Neumann_II(A, b, 2 * h * f_A)
#### END SOLUTION

# Arreglo para almacenar la solución

```

```

u2 = np.zeros(N+2)

# Condición de frontera del lado derecho: Dirichlet
u2[-1] = b_B

# Solución del sistema lineal
u2[:N+1] = np.linalg.solve(A,b)

file_answer.write('3', u2, 'La solución no es correcta, checa tu implementación c
file_answer.to_file('q02')

print('Solución numérica: {}'.format(u2))

```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
 Respuestas y retroalimentación almacenadas.

Solución numérica: [2.26862518 2.29305323 2.3663374 2.49929455 2.70513646 3.]

```
quizz.eval_numeric('3', u2)
```

 3 | Tu resultado es correcto.

7.6 Ejercicio 4.

- Calcular el error de la solución numérica `u2` con respecto a la solución exacta `sol_e` usando la definición **Grid norm 2** descrita en la “Problemas de Calibración” (página 20).

```

#### BEGIN SOLUTION
# Cálculo del error
e2 = np.sqrt(h) * np.linalg.norm(sol_e - u2, 2)

file_answer.write('4', e2, 'La implementación del error Grid norm 2 parece no ser
#### END SOLUTION
file_answer.to_file('q02')

print(e2)

```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
 Respuestas y retroalimentación almacenadas.
 0.008364723421555469

```
quizz.eval_numeric('4', e2)
```

 4 | Tu resultado es correcto.

```

# Graficación de la solución y el error
error_label_2 = 'Error (grid norm {}) = {:.4f}'.format(2, e2)

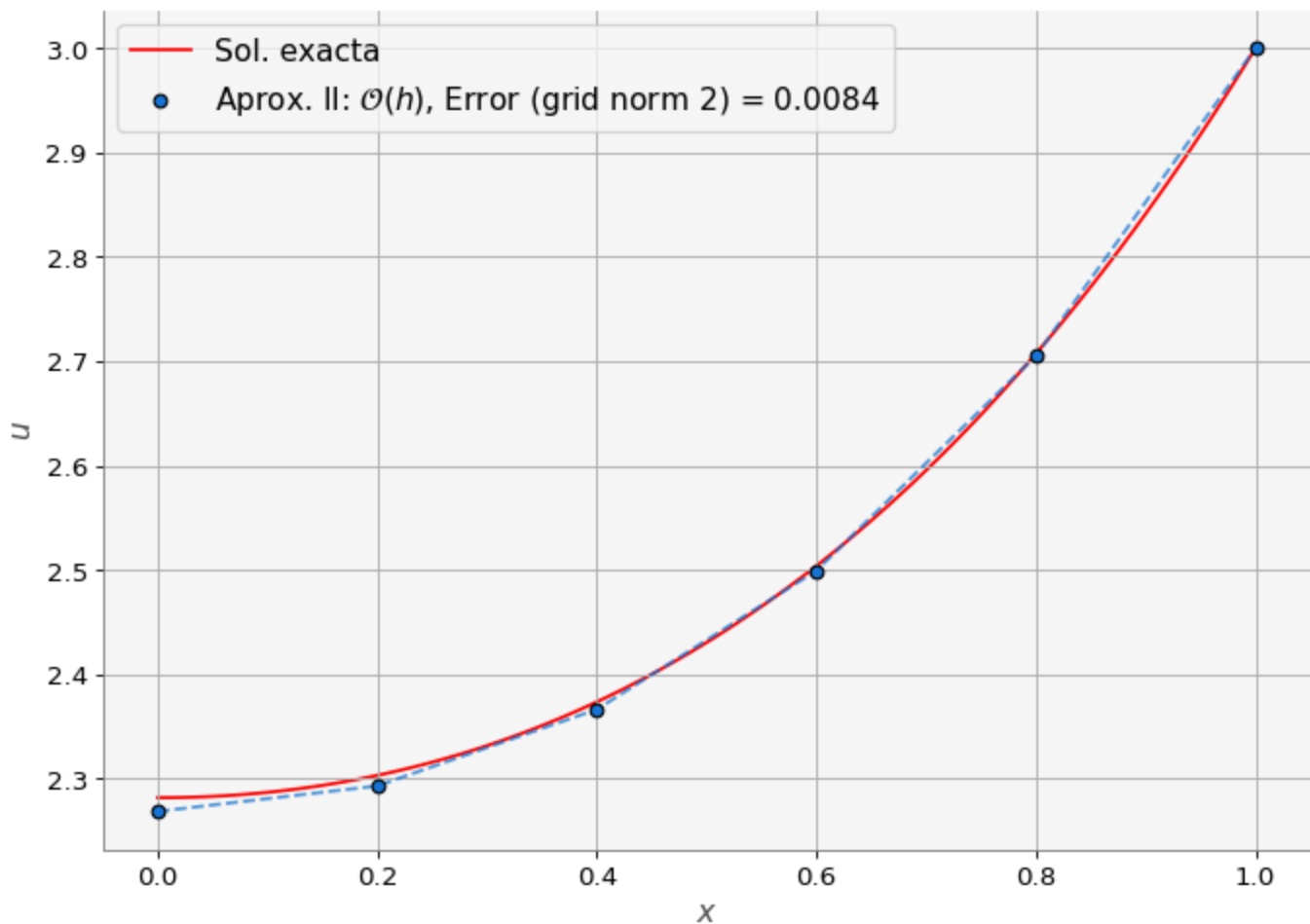
plt.figure(figsize=(10,7))

# Graficación de la solución exacta
plt.plot(xsol, solExact(xsol), 'r-', label='Sol. exacta', zorder=0)

# Graficación de la solución numérica
plt.scatter(x, u2, marker='o', edgecolor='k', zorder=5,
            label='Aprox. II:  $\mathcal{O}(h)$ , ' + error_label_2)
plt.plot(x, u2, '--', lw=1.5, alpha=0.75)

# Decoración de la gráfica
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend(loc='upper left', fontsize=14)
plt.grid()
plt.show()

```



7.7 Ejercicio 5.

Definir lo siguiente:

- La función `Neumann_III(A, b, bcond)` que implemente la aproximación II de la presentación “Problemas de Calibración” (página 18).
- Llamada a la función `Neumann_III(A, b, bcond)` con los parámetros adecuados.

```
### BEGIN SOLUTION
def Neumann_III(A, b, bcond):
    A[0][0] = 2
    A[0][1] = -2
    A[0][2] = 0
    b[0] = bcond

# Corrección de la matriz y el RHS, orden lineal
Neumann_III(A, b, -np.exp(x[0]) * h**2 + 2 * h * f_A)
### END SOLUTION

# Arreglo para almacenar la solución
u3 = np.zeros(N+2)

# Condición de frontera del lado derecho: Dirichlet
u3[-1] = b_B

# Solución del sistema lineal
u3[:N+1] = np.linalg.solve(A,b)

file_answer.write('5', u3, 'La solución no es correcta, chequea tu implementación c

print('Solución numérica: {}'.format(u3))
```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
Respuestas y retroalimentación almacenadas.

Solución numérica: [2.29076545 2.31076545 2.37962156 2.50815066 2.70956451 3.]

```
quizz.eval_numeric('5', u3)
```

5 | Tu resultado es correcto.

7.8 Ejercicio 6.

- Calcular el error de la solución numérica u_3 con respecto a la solución exacta sol_e usando la definición **Grid norm 2** descrita en la “Problemas de Calibración” (página 20).

```
#### BEGIN SOLUTION
# Cálculo del error
e3 = np.sqrt(h) * np.linalg.norm(sol_e - u3, 2)

file_answer.write('6', e3, 'La implementación del error Grid norm 2 parece no ser
#### END SOLUTION

print('Error: {}'.format(e3))
```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
 Respuestas y retroalimentación almacenadas.
 Error: 0.006342955932511579

```
quizz.eval_numeric('6', e3)
```

 6 | Tu resultado es correcto.

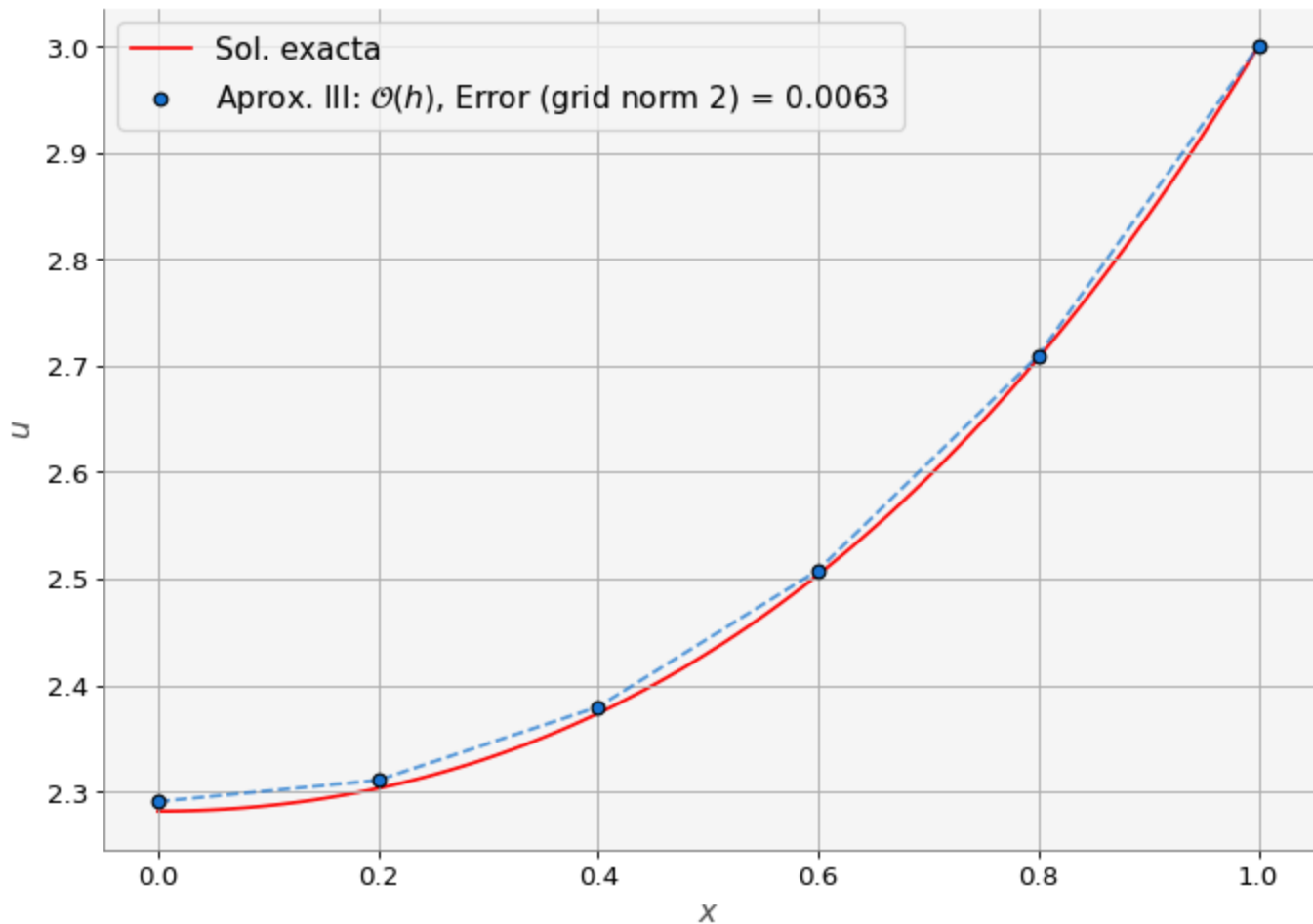
```
# Graficación de la solución y el error
error_label_3 = 'Error (grid norm {}) = {:.4f}'.format(2, e3)

plt.figure(figsize=(10,7))

# Graficación de la solución exacta
plt.plot(xsol, solExact(xsol), 'r-', label='Sol. exacta', zorder=0)

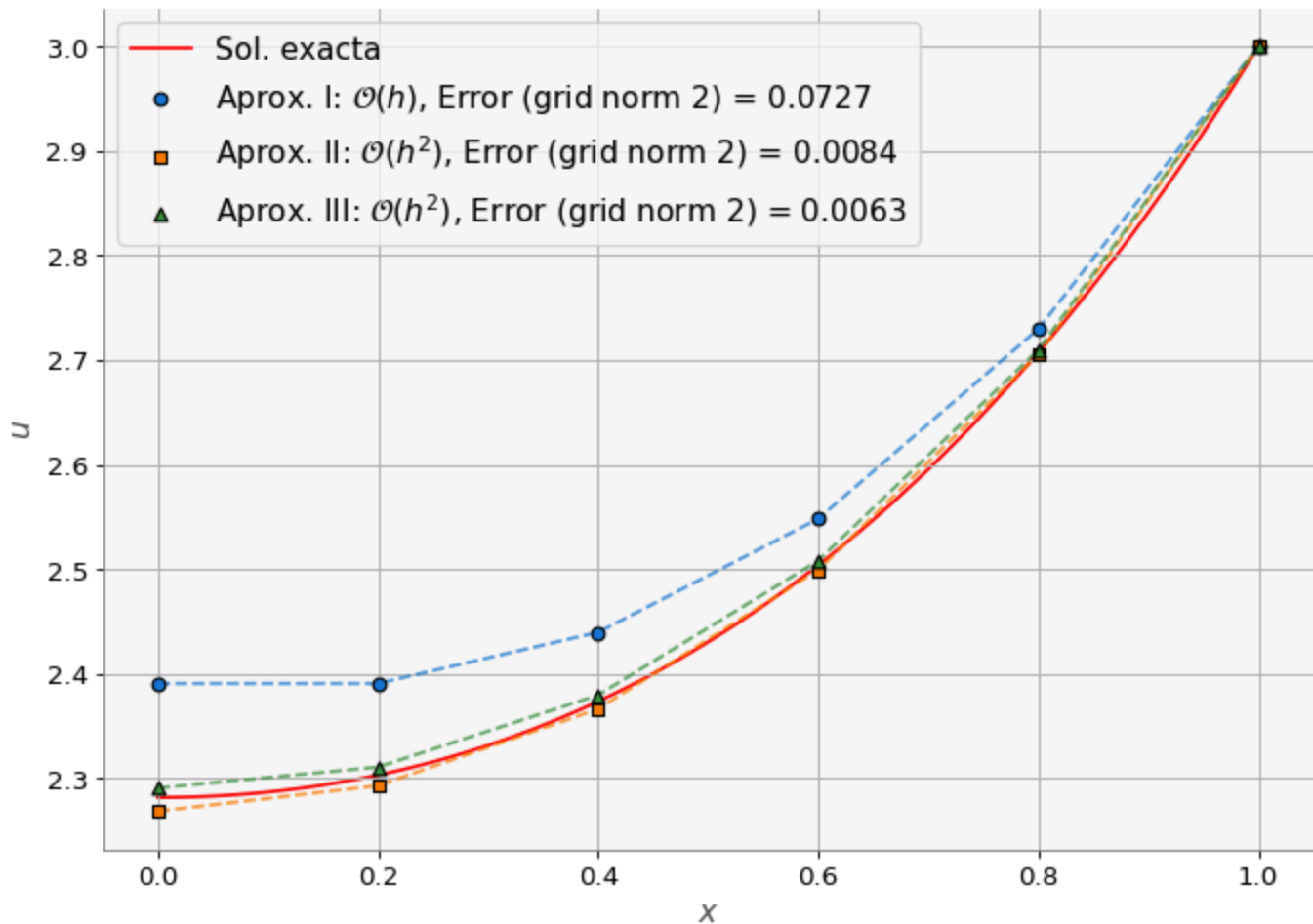
# Graficación de la solución numérica
plt.scatter(x, u3, marker='o', edgecolor='k', zorder=5,
            label='Aprox. III:  $\mathcal{O}(h)$ , ' + error_label_3)
plt.plot(x, u3, '--', lw=1.5, alpha=0.75)

# Decoración de la gráfica
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend(loc='upper left', fontsize=14)
plt.grid()
plt.show()
```

7.9 Graficación de las tres aproximaciones.

```
plt.figure(figsize=(10,7))
xsol = np.linspace(0,1,100)
plt.plot(xsol, solExact(xsol), 'r-', label='Sol. exacta', zorder=0)
plt.scatter(x, u1, marker='o', edgecolor='k', zorder=5,
            label='Aprox. I:  $\mathcal{O}(h)$ , ' + error_label_1)
plt.plot(x, u1, '--', lw=1.5, alpha=0.75)
plt.scatter(x, u2, marker='s', edgecolor='k', zorder=5,
            label='Aprox. II:  $\mathcal{O}(h^2)$ , ' + error_label_2)
plt.plot(x, u2, '--', lw=1.5, alpha=0.75)
plt.scatter(x, u3, marker='^', edgecolor='k', zorder=5,
            label='Aprox. III:  $\mathcal{O}(h^2)$ , ' + error_label_3)
plt.plot(x, u3, '--', lw=1.5, alpha=0.75)
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend(loc='upper left', fontsize=14)
plt.grid()
plt.show()
```



7.10 Estudio de refinamiento de malla.

La función `meshRefining(...)` realiza un estudio de refinamiento de malla para determinar la mejor aproximación de las condiciones de frontera de tipo Neumann.

```
def meshRefining(fcond, nodos, norma):
    """
    Función que permite realizar un estudio de
    refinamiento de malla.

    Parameters:
    -----
    fcondNeumman: function
    función que establece la aproximación para la condición
    de frontera de tipo Neumman.

    nodes: list
    Lista de número de nodos que se usarán para el estudio
    de refinamiento de malla.

    norma:
    Define el tipo de grid norm que se usará para calcular el
```

error con respecto a la solución exacta.

Returns:

```

-----
e_lista: list
Lista con los errores calculados para los diferentes números
de nodos.
"""
e_lista = []
for N in nodos:
    h = L / (N+1)
    r = k / h**2

    # Coordenadas de los nodos
    x = np.linspace(0, L, N+2)

    sole = solExact(x)

    A = buildMatrix(N+1, 2)      # Construcción de la matriz
    b = np.zeros(N+1)           # Lado derecho del sistema
    b[1:] = -np.exp(x[1:-1]) / r # Fuente o sumidero
    b[-1] += b_B                 # Condición de frontera en B

    if fcond.__name__ == 'Neumann_I':
        bcond = h * f_A
    elif fcond.__name__ == 'Neumann_II':
        bcond = 2 * h * f_A
    elif fcond.__name__ == 'Neumann_III':
        bcond = -1/r + 2 * h * f_A

    fcond(A, b, bcond) # Corrección de la matriz y el RHS

    u = np.zeros(N+2) # Arreglo para almacenar la solución
    u[-1] = b_B       # Frontera derecha Dirichlet
    u[1:N+1] = np.linalg.solve(A,b)      # Sol. del sist. lineal
    e_lista.append(np.linalg.norm(sole - u, norma)) # Cálculo del error

return e_lista

```

7.11 Ejercicio 7.

Usando la función `meshRefining(...)` realiza un estudio de refinamiento de malla para determinar la mejor aproximación de las condiciones de frontera de tipo Neumann para el problema planteado en esta notebook.

- Definir un arreglo con el número de nodos como sigue: $nodos = \{2^i | \forall i \in [2, 3, \dots, 8]\}$
- Calcular las h 's correspondientes a estos nodos y almacenarlas en el arreglo: `h_lista`.

```

#### BEGIN SOLUTION
nodos = [2**i for i in range(2,8)]
h_lista = [L/(n+1) for n in nodos]

file_answer.write('7', h_lista, 'Checa la lista de nodos y el cálculo de las h's.
#### END SOLUTION

print('Nodos: {}'.format(nodos))
print('h: {}'.format(h_lista))

```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
Respuestas y retroalimentación almacenadas.

Nodos: [4, 8, 16, 32, 64, 128]

h: [0.2, 0.1111111111111111, 0.058823529411764705, 0.030303030303030304,
0.015384615384615385, 0.007751937984496124]

```
quizz.eval_numeric('7', h_lista)
```

7 | Tu resultado es correcto.

7.12 Ejercicio 8.

Usando los arreglos `nodos`, `h_lista`, las funciones `Neumann_I()`, `Neumann_II()`, `Neumann_III()` y `meshRefining()` realiza un estudio de refinamiento de malla para determinar la mejor aproximación de las condiciones de frontera de tipo Neumann para el problema planteado en esta notebook.

```

#### BEGIN SOLUTION
norma = np.inf
e1m = meshRefining(Neumann_I, nodos, norma)
e2m = meshRefining(Neumann_II, nodos, norma)
e3m = meshRefining(Neumann_III, nodos, norma)

file_answer.write('8', e1m, 'La implementación de la llamada a meshRefining() par
file_answer.write('9', e2m, 'La implementación de la llamada a meshRefining() par
file_answer.write('10', e3m, 'La implementación de la llamada a meshRefining() pa
#### END SOLUTION

print('Errores para Neumann_I: {}'.format(e1m))
print('Errores para Neumann_II: {}'.format(e2m))
print('Errores para Neumann_III: {}'.format(e3m))

```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe
Respuestas y retroalimentación almacenadas.

Errores para Neumann_I: [0.10904728226552596, 0.05835083948479003, 0.030195480658391283,

```
0.01535951849054662, 0.007745922239629444, 0.0038895813335773077]
Errores para Neumann_II: [0.013092993550491094, 0.003733553223093544,
0.000998285860006387, 0.00025816090671559877, 6.564366654870923e-05, 1.6550689755145953e-
05]
Errores para Neumann_III: [0.009047282265525869, 0.002795283929234227,
0.0007837159525103665, 0.00020800333903103763, 5.361454732044635e-05,
1.3612341328350652e-05]
```

```
quizz.eval_numeric('8', e1m)
```

8 | Tu resultado es correcto.

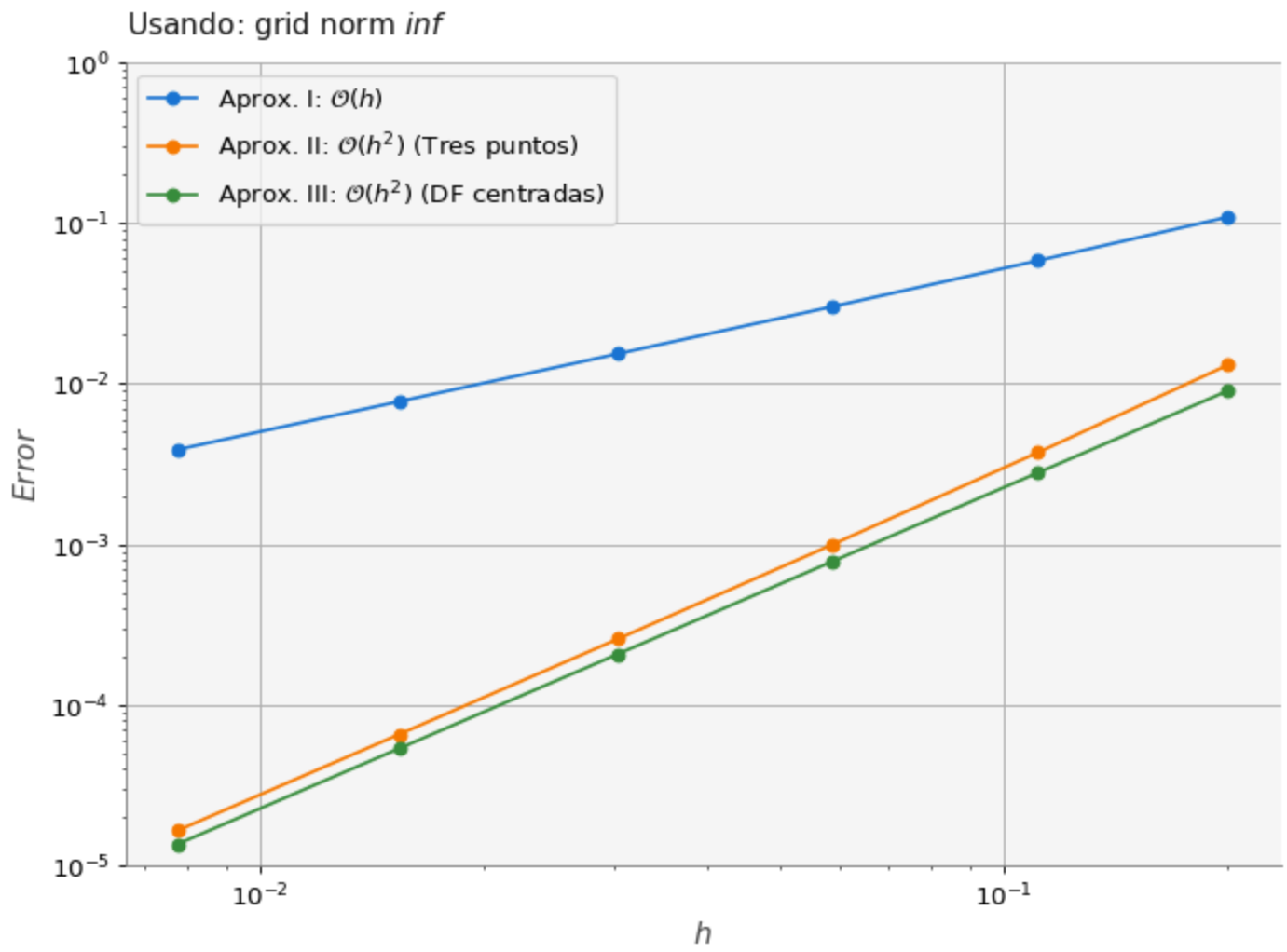
```
quizz.eval_numeric('9', e2m)
```

9 | Tu resultado es correcto.

```
quizz.eval_numeric('10', e3m)
```

10 | Tu resultado es correcto.

```
#
# El siguiente código genera las gráficas usando los resultados anteriores:
#
plt.figure(figsize=(10,7))
plt.plot(h_lista, e1m, 'o-', label='Aprox. I: $\mathcal{O}(h)$')
plt.plot(h_lista, e2m, 'o-', label='Aprox. II: $\mathcal{O}(h^2)$ (Tres puntos)')
plt.plot(h_lista, e3m, 'o-', label='Aprox. III: $\mathcal{O}(h^2)$ (DF centradas)')
plt.yscale('log')
plt.xscale('log')
plt.legend(fontsize=12)
plt.ylim(1e-5,1)
plt.ylabel('$Error$')
plt.xlabel('$h$')
plt.title('Usando: grid norm ${}$'.format(norma))
plt.grid()
plt.show()
```



4 Derivadas numéricas: aplicación

Objetivo general - Entender la utilidad de una derivada y su aproximación numérica en una aplicación simple.

```
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis
from macti.evaluation import *
```

```
quizz = Quizz("q4", "notebooks", "local")
```

Masa y densidad

Un experimentado maestro albañil, necesita cortar una varilla de metal en varias secciones para construir una escalera. Realiza las marcas de la varilla y se ven como en la siguiente figura:



Como se observa, el tamaño de cada sección de la varilla es de 0.5 m. Por razones de la estructura, se necesita conocer el peso de cada sección de la varilla para evitar que la escalera se derrumbe. El maestro albañil realizó los cortes y pesó cada sección, obteniendo los siguientes resultados:

Sección	1	2	3	4	5	6	7	8
Masa [Kg]	0.595	0.806	0.369	1.078	1.704	1.475	2.263	3.282

4.1 Ejercicio 1.

Definir los arreglos de **numpy** para las secciones de la varilla como sigue:

- longitud** : para almacenar las marcas hechas en la varillas, comenzando en 0 y terminando en 4.0.
- masas_sec** : para almacenar el valor de la masa de cada sección.

```
# longitud = np...
# masas_sec = np...

### BEGIN SOLUTION
# Marcas sobre la varilla de cada sección
longitud = np.linspace(0,4.0,9)
#np.array([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0])

# Peso de cada sección [kg]
```

```

masas_sec = np.array([0.595, 0.806, 0.369, 1.078, 1.704, 1.475, 2.263, 3.282])

file_answer = FileAnswer()
file_answer.write("1a", longitud, 'Checa el arreglo secciones')
file_answer.write("1b", masas_sec, 'Checa el arreglo masas_sec')
#### END SOLUTION

print('Longitud = {}'.format(longitud))
print('Masas = {}'.format(longitud))

```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe
 Respuestas y retroalimentación almacenadas.

Longitud = [0. 0.5 1. 1.5 2. 2.5 3. 3.5 4.]

Masas = [0. 0.5 1. 1.5 2. 2.5 3. 3.5 4.]

```
quizz.eval_numeric('1a', longitud)
```

 1a | Tu resultado es correcto.

```
quizz.eval_numeric('1b', masas_sec)
```

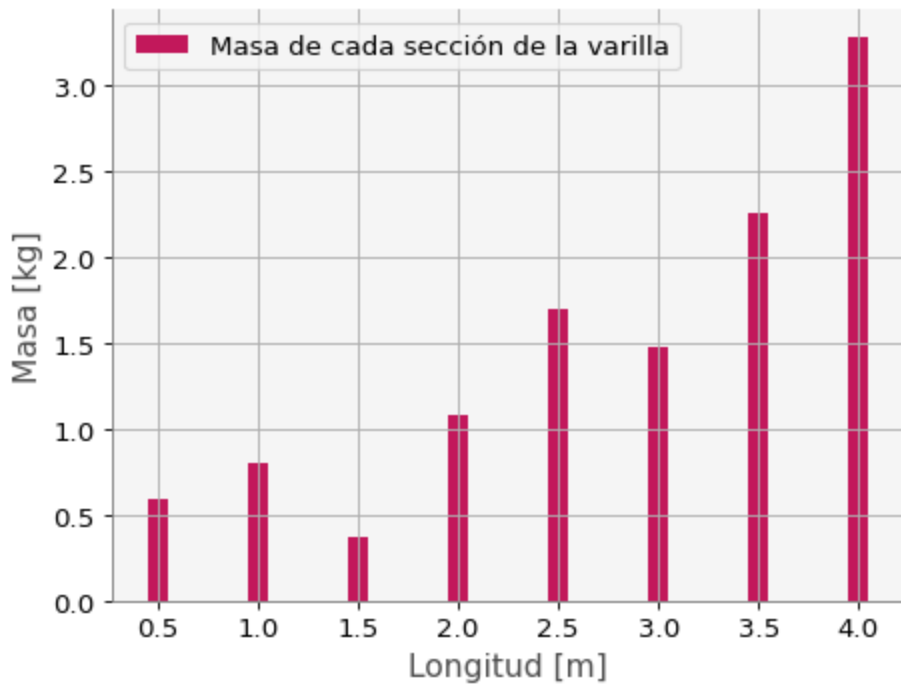
 1b | Tu resultado es correcto.

```

# Gráfica de la masa para cada sección en forma de barras verticales.
plt.bar(longitud[1:], masas_sec,
        width=0.1, color='C3',
        label='Masa de cada sección de la varilla')

plt.xlabel('Longitud [m]')
plt.ylabel('Masa [kg]')
plt.grid()
plt.legend()
plt.show()

```

4.2 Ejercicio 2.

Escribe un código que genere el arreglo de numpy `masa` con ceros, del mismo tamaño que el arreglo `longitud`. En la primera posición del arreglo `masa` deje el valor de cero; en la segunda posición ponga el valor de la masa de la primera sección; en la tercera posición el valor de la primera sección más el valor de la masa de la segunda sección; y así sucesivamente hasta obtener el peso total de la varilla en la última posición. Diseña un algoritmo para realizar este proceso y escríbalo en la siguiente celda.

```
# masa = ... # arreglo para almacenar la masa de las secciones
# for ...
#     ...

#### BEGIN SOLUTION
masa = np.zeros(len(longitud))
for i, ms in enumerate(masas_sec):
    masa[i+1] = masa[i] + ms

file_answer.write("2", masa, 'Checa la construcción del arreglo masa')
#### END SOLUTION

print('Masa = {}'.format(masa))
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe Respuestas y retroalimentación almacenadas.

Masa = [0. 0.595 1.401 1.77 2.848 4.552 6.027 8.29 11.572]

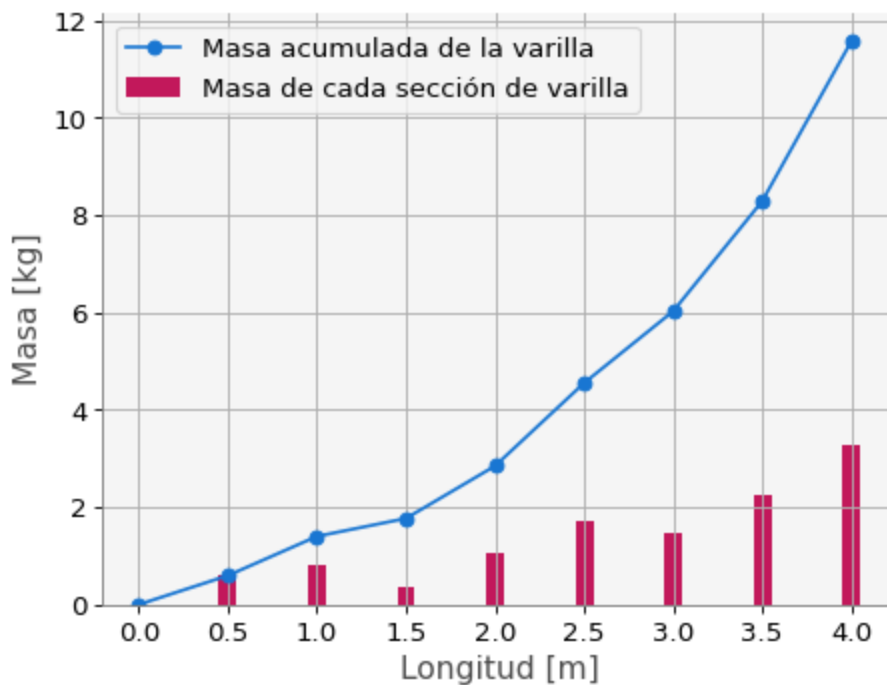
```
quizz.eval_numeric('2', masa)
```

2 | Tu resultado es correcto.

```
# Gráfica de la masa como función de la posición
plt.plot(longitud, masa,
         'o-', label='Masa acumulada de la varilla')

# Gráfica de la masa para cada sección en forma de barras verticales.
plt.bar(longitud[1:], masas_sec,
        width=0.1, color='C3',
        label='Masa de cada sección de varilla')

plt.xlabel('Longitud [m]')
plt.ylabel('Masa [kg]')
plt.legend()
plt.grid()
plt.show()
```



Si todo se hizo correctamente, se verá que la masa no crece linealmente. Se sospecha que la densidad de la varilla no cambia homogéneamente en toda su longitud. ¿Cómo podemos determinar la densidad de la varilla en cada uno de sus puntos?

Suponemos que todo está en una dimensión, de tal manera que podemos definir una densidad “lineal” de la siguiente manera:

3 | Tu resultado es correcto.

```

:::
:::

```

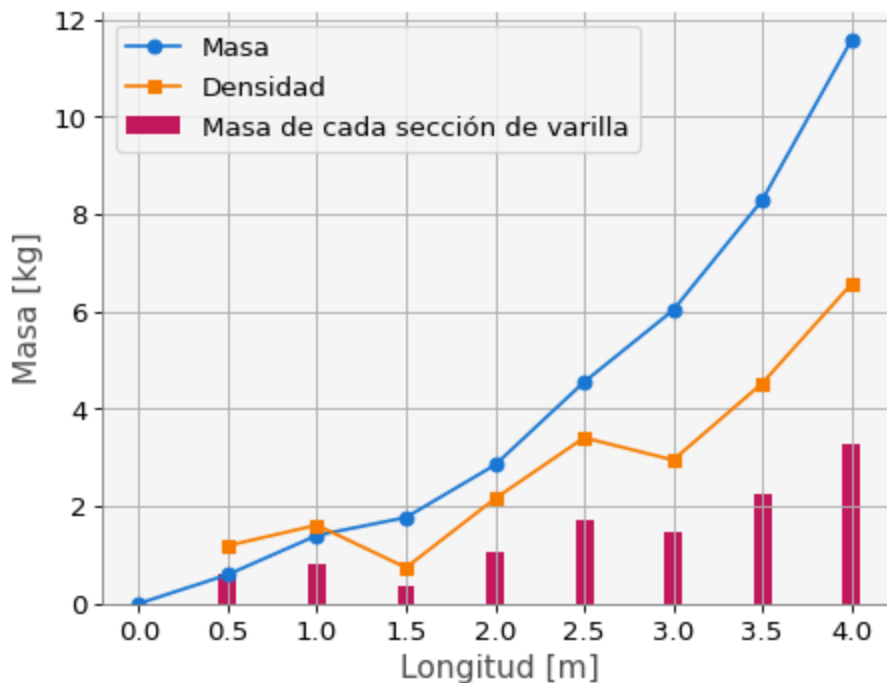
```

::: {#cell-19 .cell nbgrader={"grade":false,"grade_id":"cell-
fde016e0198c4d2e","locked":false,"schema_version":3,"solution":true,"task":false}'
execution_count=18}
``` {.python .cell-code}
Gráfica de la masa y de la densidad para cada sección
plt.plot(longitud, masa, 'o-', label='Masa')
plt.plot(longitud[1:], densidad, 's-', label='Densidad')

Gráfica de la masa para cada sección en forma de barras verticales.
plt.bar(longitud[1:], masas_sec,
 width=0.1, color='C3',
 label='Masa de cada sección de varilla')

plt.xlabel('Longitud [m]')
plt.ylabel('Masa [kg]')
plt.legend()
plt.grid()
plt.show()

```



```

:::

```

Después de una búsqueda sobre las especificaciones de la varilla, se encuentra que la densidad está dada por siguiente fórmula:

$$\rho = (1000x^2 + 5000 \sin^2(2x))A \quad (2)$$

donde  $x$  es la posición en la varilla y  $A$  es el área transversal de la misma. Al medir el diámetro de la varilla se encuentra el valor de  $d = 0.02$  m, por lo tanto el radio es  $r = 0.01$  m.

## 4.3 Ejercicio 4.

Implemente la fórmula de la densidad (2) en la función `calc_densidad(x, A)` y evalúa dicha fórmula con los datos del radio antes definido y puntos que están en el intervalo  $[0, 4.5]$  separados por una distancia de 0.1. Almacena el resultado en la variable `p`. Posteriormente compara gráficamente el resultado con la aproximación realizada en el ejercicio anterior.

```
r = 0.01
A = np.pi * r ** 2

def calc_densidad(x, A):
...
#
x = ...
p = ...

BEGIN SOLUTION
calc_densidad = lambda x, A: (1000 * x**2 + 5000 * np.sin(2*x)**2) * A

#def calc_densidad(x, A):
return (1000 * x**2 + 5000 * np.sin(2*x)**2) * A

Puntos donde se evaluó la fórmula de la densidad
x = np.arange(0.0, 4.5, .1)

Cálculo de la densidad en cada posición del arreglo x
p = [calc_densidad(l,A) for l in x]

file_answer.write("4", p, 'Verifica que la fórmula (2) esté bien implementada y c
END SOLUTION
file_answer.to_file('q4')

print(p)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe Respuestas y retroalimentación almacenadas.

```
[0.0, 0.065140142984144, 0.25077236406386394, 0.5290773824209537, 0.8585968970424002,
1.1907789408649767, 1.4776431688135958, 1.6793558992965574, 1.7705189766656613,
1.7441795815382646, 1.6129279280991666, 1.406909546114531, 1.169065964621893,
0.9483551886937212, 0.7920223069253689, 0.7381405307711528, 0.8096002716097072,
1.0104952481017955, 1.3254761780264852, 1.7221740924437288, 2.156310684160917,
2.578688878846925, 2.9429599713234333, 3.212941066996997, 3.368327565325648,
```

3.4078988097868033, 3.349710802702375, 3.2282455594876254, 3.0889671566078496,  
 2.9811439535688815, 2.9500702022640968, 3.0299150805307193, 3.238328130281404,  
 3.5736527829144573, 4.0151878950714055, 4.526456003996436, 5.060962316838884,  
 5.569535216116573, 6.00808937693044, 6.344585870417729, 6.564090406147258,  
 6.671131128203245, 6.688983720799259, 6.65599668952679, 6.619536975529502]

```
quizz.eval_numeric('4',p)
```

4 | Tu resultado es correcto.

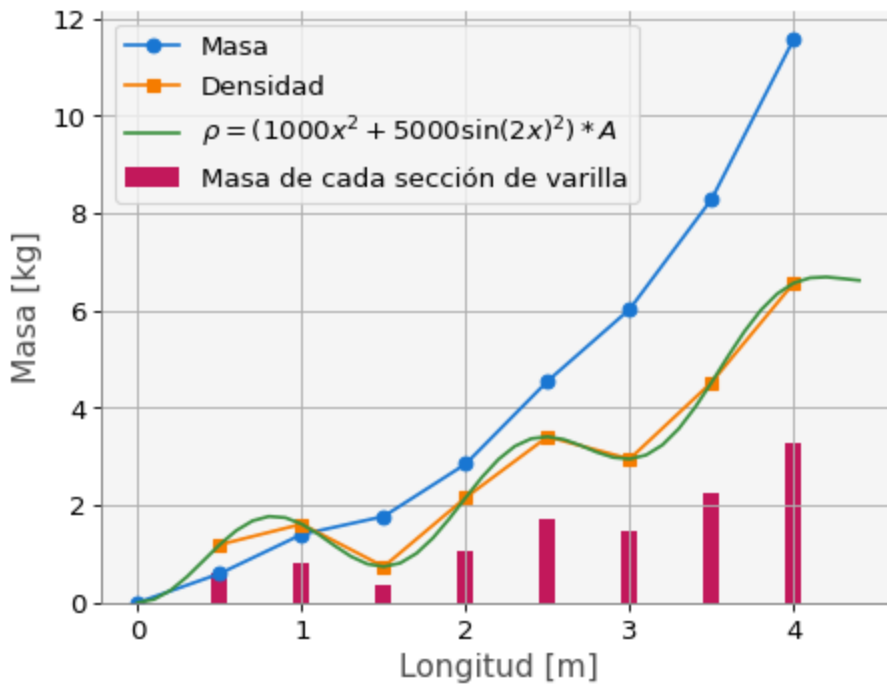
```
Gráfica de la masa como función de las secciones
plt.plot(longitud, masa, 'o-', label='Masa')

Gráfica de la densidad como función de las secciones
plt.plot(longitud[1:], densidad, 's-', label='Densidad')

Gráfica de la densidad exacta
plt.plot(x, rho, label = '$\rho =(1000 x^2 + 5000 \sin(2x)^2) * A $')

Gráfica de la masa para cada sección en forma de barras verticales.
plt.bar(longitud[1:], masas_sec,
 width=0.1, color='C3',
 label='Masa de cada sección de varilla')

plt.xlabel('Longitud [m]')
plt.ylabel('Masa [kg]')
plt.legend()
plt.grid()
plt.show()
```



Para evaluar la aproximación, se puede usar el error absoluto y el error relativo los cuales se definen como sigue.

$$Error_{absoluto} = ||v_e - v_a||$$

$$Error_{relativo} = \frac{||v_e - v_a||}{||v_e||}$$

donde  $v_e$  es el valor exacto y  $v_a$  es el valor aproximado.

## 4.4 Ejercicio 5. Error absoluto y error relativo.

Implemente las fórmulas del error absoluto y relativo en las funciones `lambda error_absoluto(ve, va)` y `error_relativo(ve, va)` respectivamente.

- 5a. Calcular el valor de la densidad exacta con la fórmula (2) para cada sección. Almacene el resultado en la variable `densidad_e`.
- 5b. Comparar la aproximación (1) con el resultado del inciso 5a usando el error absoluto. Almacene el error en la variable `error_a`.
- 5c. Comparar la aproximación (1) con el resultado del inciso 5a usando el error relativo. Almacene el error en la variable `error_r`.

```
error_absoluto = lambda ...
error_relativo = lambda ...
densidad_e = ...
error_a = ...
```

```

error_r = ...

BEGIN SOLUTION
error_absoluto = lambda ve, va: np.fabs(ve - va)
error_relativo = lambda ve, va: np.fabs(ve - va) / np.fabs(ve)

Calculamos la densidad en cada sección con la fórmula (2)
densidad_e = calc_densidad(longitud[1:], A)

Calculamos los errores con respecto de la aproximación

error_a = [error_absoluto(e, a) for e, a in zip(densidad_e, densidad)]
error_r = [error_relativo(e, a) for e, a in zip(densidad_e, densidad)]

#error_a = []
#error_r = []
#for e,a in zip(densidad_e, densidad):
error_a.append(error_absoluto(e,a))
error_r.append(error_relativo(e,a))

file_answer.write("5a", densidad_e, '')
file_answer.write("5b", error_a, '')
file_answer.write("5c", error_r, '')
END SOLUTION

print('Densidad exacta secciones = {}'.format(densidad_e))
print('Error absoluto = {}'.format(error_a))
print('Error relativo = {}'.format(error_r))

```

Densidad exacta secciones = [1.19077894 1.61292793 0.73814053 2.15631068 3.40789881  
2.9500702  
4.526456 6.56409041]

Error absoluto = [0.0007789408649767626, 0.0009279280991665306, 0.00014053077115283585,  
0.00031068416091750706, 0.00010119021319621169, 7.020226409748531e-05,  
0.00045600399643586087, 9.040614725819296e-05]

Error relativo = [0.0006541439710135815, 0.0005753066104200283, 0.0001903848458314842,  
0.0001440813530256208, 2.969284560492631e-05, 2.3796811358457512e-05,  
0.0001007419482335081, 1.3772837006255698e-05]

```
quizz.eval_numeric('5a', densidad_e)
```

-----  
5a | Tu resultado es correcto.  
-----

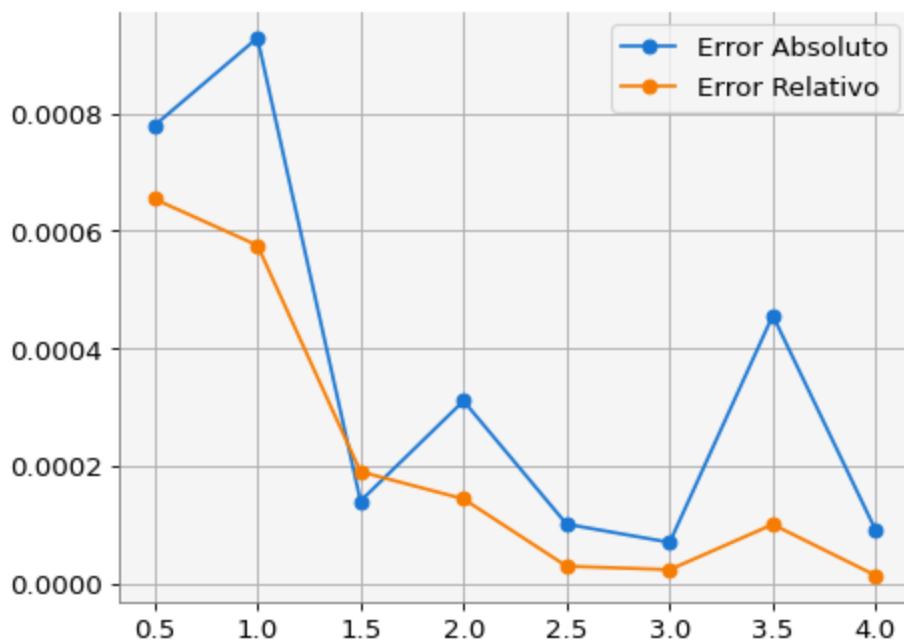
```
quizz.eval_numeric('5b', error_a)
```

5b | Tu resultado es correcto.

```
quizz.eval_numeric('5c', error_r)
```

5c | Tu resultado es correcto.

```
Gráficas del error absoluto y del error relativo
plt.plot(longitud[1:], error_a, 'o-', label='Error Absoluto')
plt.plot(longitud[1:], error_r, 'o-', label='Error Relativo')
plt.legend()
plt.grid()
plt.show()
```



Si tenemos la fórmula de la densidad, ecuación (2), podemos encontrar la fórmula para la masa haciendo la integral de la densidad.

$$m(x) = \int \rho dx = \int (1000x^2 + 5000 \sin^2(2x)) A dx = A \int f(x) dx = i? \quad (3)$$

## 4.5 Ejercicio 5. Fórmula exacta para la masa.

- 6a. Calcula la integral definida en (3) donde  $f(x) = 1000x^2 + 5000 \sin^2(2x)$ . Escribe su respuesta en la variable `masa_e` usando expresiones de Python y funciones de Sympy.



- 6b. Posteriormente calcula la masa para cada sección usando el resultado de la integración. Para ello escribe una función *lambda* con la fórmula de la integral. Almacena el resultado en la variable *m*.

Compare el resultado gráficamente con los datos de la masa calculados al inicio.

**NOTA.** Puede usar Sympy para calcular la integral.

```
importamos funciones de sympy para escribir la respuesta.
from sympy import Symbol, sin, cos

definimos el símbolo x
x = Symbol('x')

definimos la función original para la densidad
f = 1000 * x**2 + 5000 * sin(2*x)**2

masa_e = ...

BEGIN SOLUTION
#from sympy import integrate
#masa_e = integrate(f, x)

masa_e = 1000*x**3/3 + 2500*x - 1250*sin(2*x)*cos(2*x)

file_answer.write("6a", str(masa_e), "Revisa el cálculo de la integral.")
END SOLUTION

print("Densidad exacta:")
display(f)
print("Masa exacta:")
display(masa_e)
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe Respuestas y retroalimentación almacenadas.

Densidad exacta:

Masa exacta:

$$1000x^2 + 5000 \sin^2(2x)$$

$$\frac{1000x^3}{3} + 2500x - 1250 \sin(2x) \cos(2x)$$

```
quizz.eval_expression('6a', masa_e)
```

-----  
6a | Tu respuesta:

es correcta.  
-----

$$\frac{1000x^3}{3} + 2500x - 1250 \sin(2x) \cos(2x)$$

```
Calcula la masa usando la fórmula exacta obtenida anteriormente.
calc_masa = lambda x: ...
x = ...
m = ...

BEGIN SOLUTION
calc_masa = lambda x: (1000 * x**3 / 3 + 2500*x - 1250 * np.sin(2*x) * np.cos(2*x)
x = np.arange(0.0, 4.5, .1)
m = [calc_masa(l) for l in x]

file_answer.write("6b", m, "Checa la implementación de la fórmula exacta para la
file_answer.to_file('q4')
END SOLUTION

print('Masa exacta = {}'.format(m))
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/DerivadasNumericas/` ya existe Respuestas y retroalimentación almacenadas.

```
Masa exacta = [0.0, 0.002182423384241263, 0.017064851646832385, 0.05544143582414131,
0.1245955116842955, 0.2272489188359526, 0.3612314797824474, 0.519922820911168,
0.6933967815987044, 0.8700877343973358, 1.0387157409844714, 1.1901666040809675,
1.319029996479012, 1.4245528304264585, 1.510857351304525, 1.5863895234025789,
1.6626847955472912, 1.7526461050549453, 1.8686059853195873, 2.0204787276384506,
2.2142943302921436, 2.4513456920843106, 2.7280836897293286, 3.036776704061495,
3.3668304703789333, 3.7065598775737922, 4.045132987924423, 4.374380359569433,
4.6901840194284095, 4.993226798285815, 5.288983724504746, 5.586956835113887,
5.8992742107720435, 6.238874416184963, 6.617562982912055, 7.044247773193793,
7.523631822050594, 8.055570029143981, 8.6351912646101, 9.25376661103931,
9.900186665284211, 10.56281466662354, 11.231422883091394, 11.898906543147781,
12.56250472056554]
```

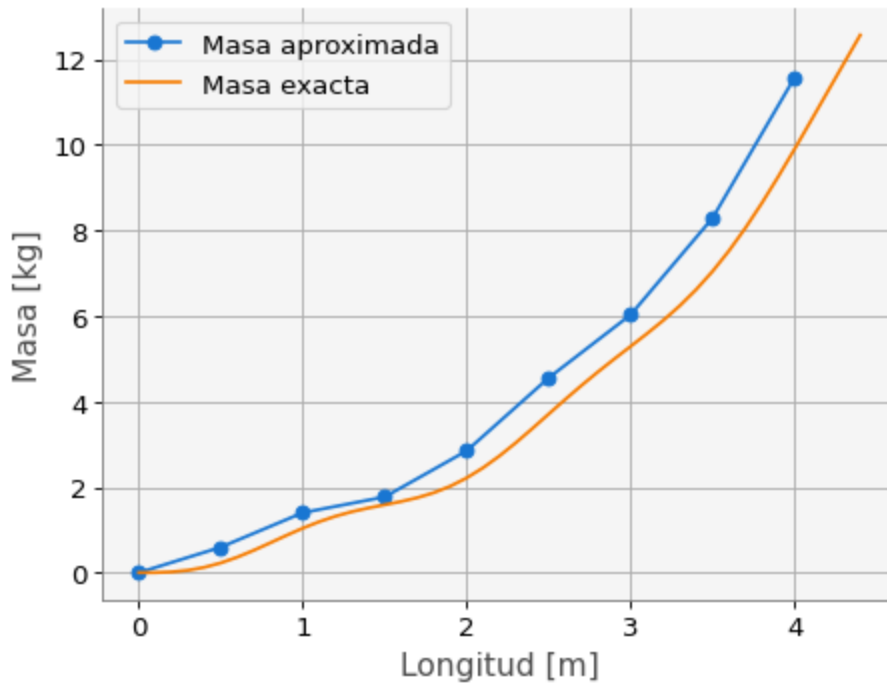
```
quizz.eval_numeric('6b', m)
```

-----  
6b | Tu resultado es correcto.  
-----

```
Gráfica de la masa exacta y de la aproximada
plt.plot(longitud, masa, 'o-', label='Masa aproximada')
plt.plot(x, m, '-', label = 'Masa exacta')

plt.xlabel('Longitud [m]')
plt.ylabel('Masa [kg]')
plt.legend()
```

```
plt.grid()
plt.show()
```





## 10 Conducción de calor No estacionaria.

### Objetivo.

Resolver la ecuación de calor no estacionaria en 2D usando un método explícito.

[HeCompA - 02\\_cond\\_calor](#) by [Luis M. de la Cruz](#) is licensed under

[Attribution-ShareAlike 4.0 International](#)

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

## 11 Conducción de calor

**Jean-Baptiste Joseph Fourier** fue un matemático y físico francés que ejerció una fuerte influencia en la ciencia a través de su trabajo *Théorie analytique de la chaleur*. En este trabajo mostró que es posible analizar la conducción de calor en cuerpos sólidos en términos de series matemáticas infinitas, las cuales ahora llevan su nombre: *Series de Fourier*. Fourier comenzó su trabajo en 1807, en Grenoble, y lo completó en París en 1822. Su trabajo le permitió expresar la conducción de calor en objetos bidimensionales (hojas muy delgadas de algún material) en términos de una ecuación diferencial:

$$\frac{\partial u}{\partial t} = \kappa \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

donde  $u$  representa la temperatura en un instante de tiempo  $t$  y en un punto  $(x, y)$  del plano Cartesiano y  $\kappa$  es la conductividad del material.

La solución a la ecuación anterior se puede aproximar usando el método de diferencias y una fórmula explícita de dicha solución es la siguiente:

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{h_t \kappa}{h^2} (u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n)$$

donde: -

$u_{i,j} = u(x_i, y_j)$ ,  $u_{i+1,j} = u(x_{i+1}, y_j)$ ,  $u_{i-1,j} = u(x_{i-1}, y_j)$ ,  $u_{i,j+1} = u(x_i, y_{j+1})$ ,  $u_{i,j-1} = u(x_i, y_{j-1})$

. - El superíndice indica el instante de tiempo, entonces el instante actual es  $n = t$  y el instante siguiente es

$n + 1 = t + h_t$ , con  $h_t$  el paso de tiempo. - En este ejemplo  $h_x = h_y$ .

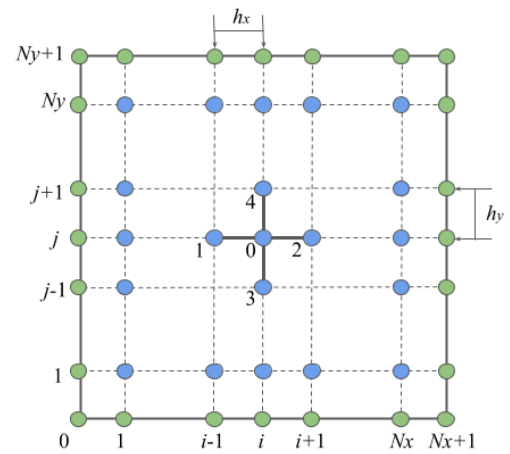
Usando esta aproximación, vamos a realizar una ejemplo de conducción de calor, pero para ello necesitamos conocer las herramientas de [numpy](#) y de [matplotlib](#).

### 11.1 Ejercicio 1.

Calculemos la transferencia de calor por conducción en una placa cuadrada unitaria usando el método de diferencias finitas. El problema se describe de la siguiente manera:

$$\frac{\partial u}{\partial t} = \kappa \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

$u(x, y, t = 0) = 0$	Condición inicial
$u(0, y, t) = 20$	Condiciones
$u(1, y, t) = 5$	de
$u(x, 0, t) = 50$	frontera
$u(x, 1, t) = 8$	



## 1. Definir los parámetros físicos y numéricos del problema:

```
import numpy as np
```

```
Parámetros físicos
k = 1.0 # Conductividad
Lx = 1.0 # Longitud del dominio en dirección x
Ly = 1.0 # Longitud del dominio en dirección y

Parámetros numéricos
Nx = 9 # Número de incógnitas en dirección x
Ny = 9 # Número de incógnitas en dirección y
h = Lx / (Nx+1) # Espaciamento entre los puntos de la rejilla
ht = 0.0001 # Paso de tiempo
N = (Nx + 2)* (Ny + 2) # Número total de puntos en la rejilla
```

## 2. Definir la rejilla donde se hará el cálculo (malla):

```
x = np.linspace(0,Lx,Nx+2) # Arreglo con las coordenadas en x
y = np.linspace(0,Ly,Ny+2) # Arreglo con las coordenadas en y
print(x)
print(y)
```

```
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
```

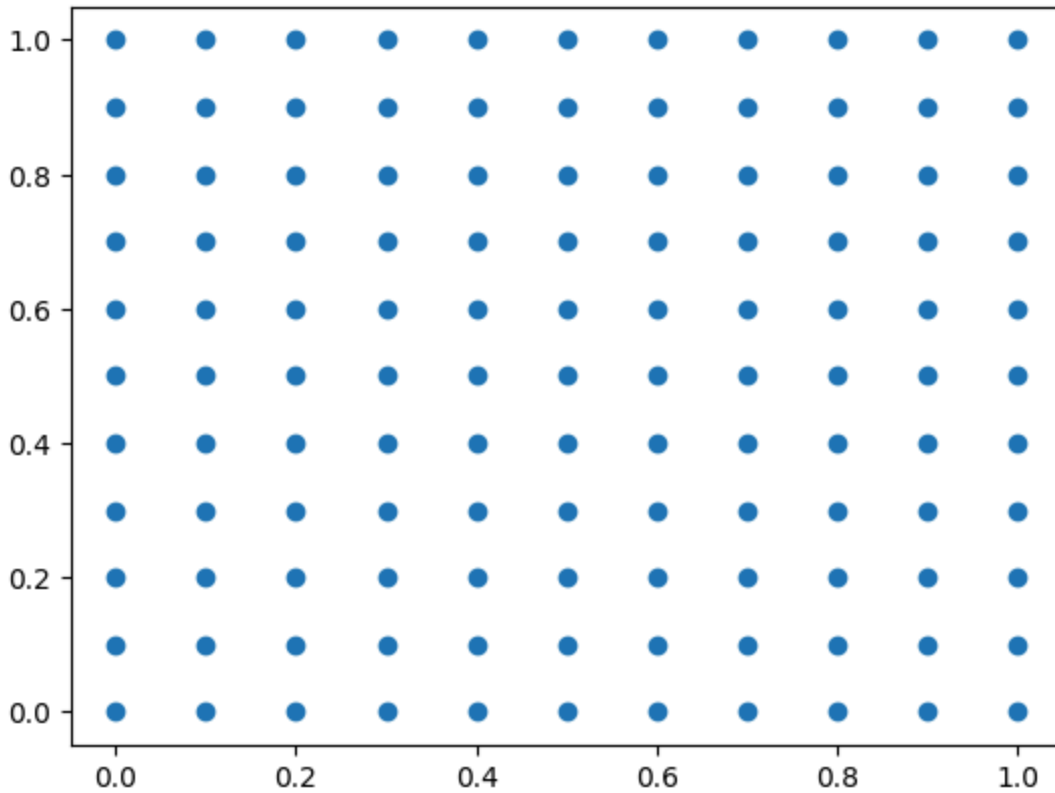
```
xg, yg = np.meshgrid(x,y) # Creamos la rejilla para usarla en Matplotlib
print(xg)
print(yg)
```

```
[[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
 [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
 [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
 [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
 [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
 [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
 [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
```

```
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
 [0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2]
 [0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]
 [0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6]
 [0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7]
 [0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8]
 [0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

```
import matplotlib.pyplot as plt
```

```
plt.scatter(xg, yg) # Graficamos la rejilla
```



**3. Definir las condiciones iniciales y de frontera:**

$u(x, y, t = 0) = 0$	Condición inicial
$u(0, y, t) = 20$	Condiciones de frontera
$u(1, y, t) = 5$	
$u(x, 0, t) = 50$	
$u(x, 1, t) = 8$	

```

u = np.zeros((Nx+2, Ny+2))
#u = np.zeros(N).reshape(Nx+2, Ny+2) # Arreglo para almacenar la aproximación
print(u)
u[0,:] = 20 # Pared izquierda
u[Nx+1,:] = 5 # Pared derecha
u[:,0] = 50 # Pared inferior
u[:,Ny+1] = 8 # Pared superior
print(u)

```

```

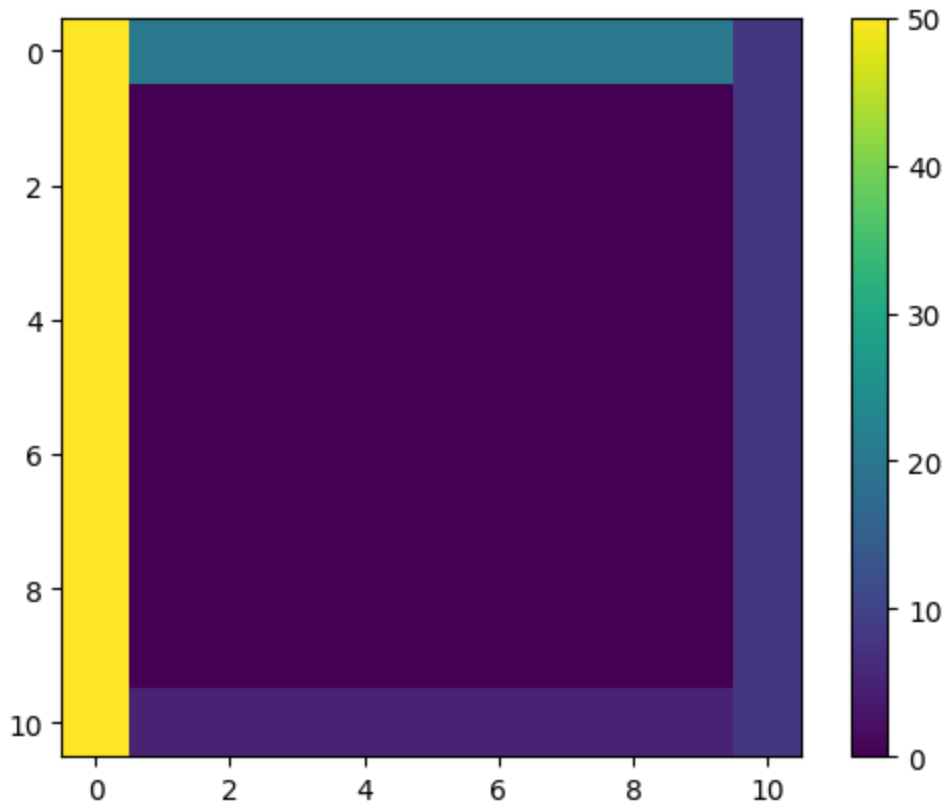
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[50. 20. 20. 20. 20. 20. 20. 20. 20. 20. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 0. 0. 0. 0. 0. 0. 0. 0. 0. 8.]
 [50. 5. 5. 5. 5. 5. 5. 5. 5. 5. 8.]]

```

```

f = plt.imshow(u)
plt.colorbar(f)

```



#### 4. Implementar el algoritmo de solución:

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{h_t \kappa}{h^2} (u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n)$$

```

ht = 0.001
r = k * ht / h**2
u_new = u.copy()
tolerancia = 9.0e-1 #1.0e-3
error = 1.0
error_lista = []
while(error > tolerancia):
 for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 u_new[i,j] = u[i,j] + r * (u[i+1,j] + u[i-1,j] + u[i,j+1] + u[i,j-1])
 error = np.linalg.norm(u_new - u)
 error_lista.append(error)
print(error)
 u[:] = u_new[:]

print(error_lista)

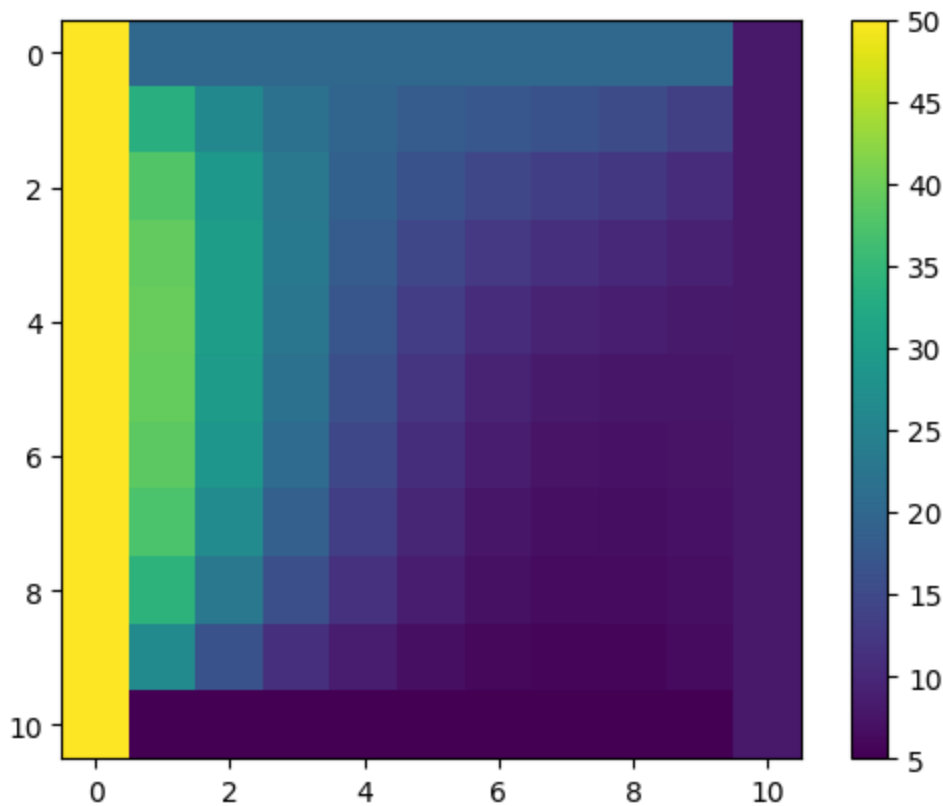
f = plt.imshow(u)
plt.colorbar(f)

```

[17.2629661414254, 13.602554907075358, 11.121464292079526, 9.370340343338656,  
8.089431314598079, 7.121431307338295, 6.368158288285477, 5.766710029025608,



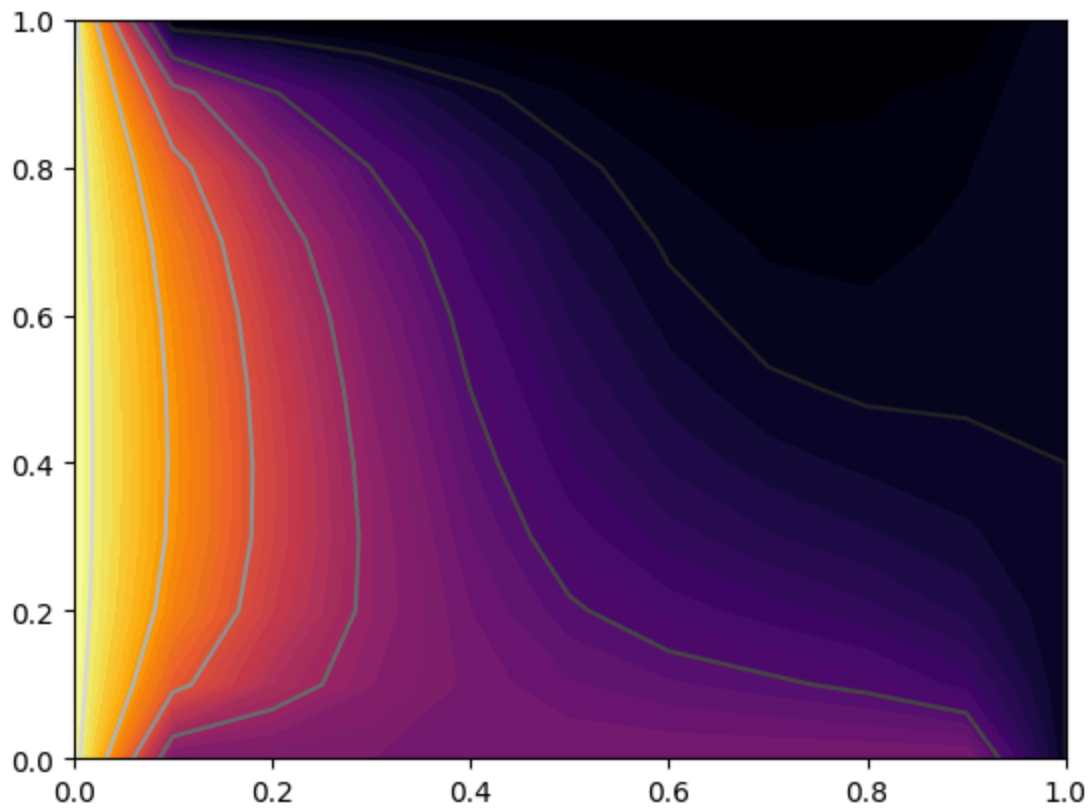
5.275727580531657, 4.867291496014421, 4.522054027452314, 4.226261372943586,  
 3.9698959855402296, 3.745493517737013, 3.547373613271126, 3.3711295389631717,  
 3.21328287724892, 3.0710454523767496, 2.9421521475712167, 2.824741357872151,  
 2.7172679513259683, 2.6184387521297463, 2.5271638648303663, 2.4425193146763595,  
 2.363717902820954, 2.290086125094306, 2.2210456430727876, 2.1560982312045556,  
 2.094813422134956, 2.0368182790286324, 1.9817888683696612, 1.9294431092766995,  
 1.8795347490871392, 1.8318482687809046, 1.7861945617665091, 1.7424072597326865,  
 1.7003396024699928, 1.6598617667041742, 1.620858583384764, 1.5832275844672883,  
 1.5468773296753073, 1.5117259715044917, 1.4777000231834065, 1.4447332996949114,  
 1.4127660064863383, 1.3817439543093082, 1.3516178818527333, 1.322342870562428,  
 1.293877838356635, 1.2661851009139355, 1.239229990882043, 1.2129805267782563,  
 1.1874071245620865, 1.1624823458905373, 1.1381806779428072, 1.114478340447452,  
 1.0913531161802335, 1.0687842017418048, 1.0467520758851223, 1.025238383054624,  
 1.0042258301337272, 0.9836980946818261, 0.9636397431848914, 0.9440361580507419,  
 0.9248734722567947, 0.9061385107087614, 0.8878187374976269]



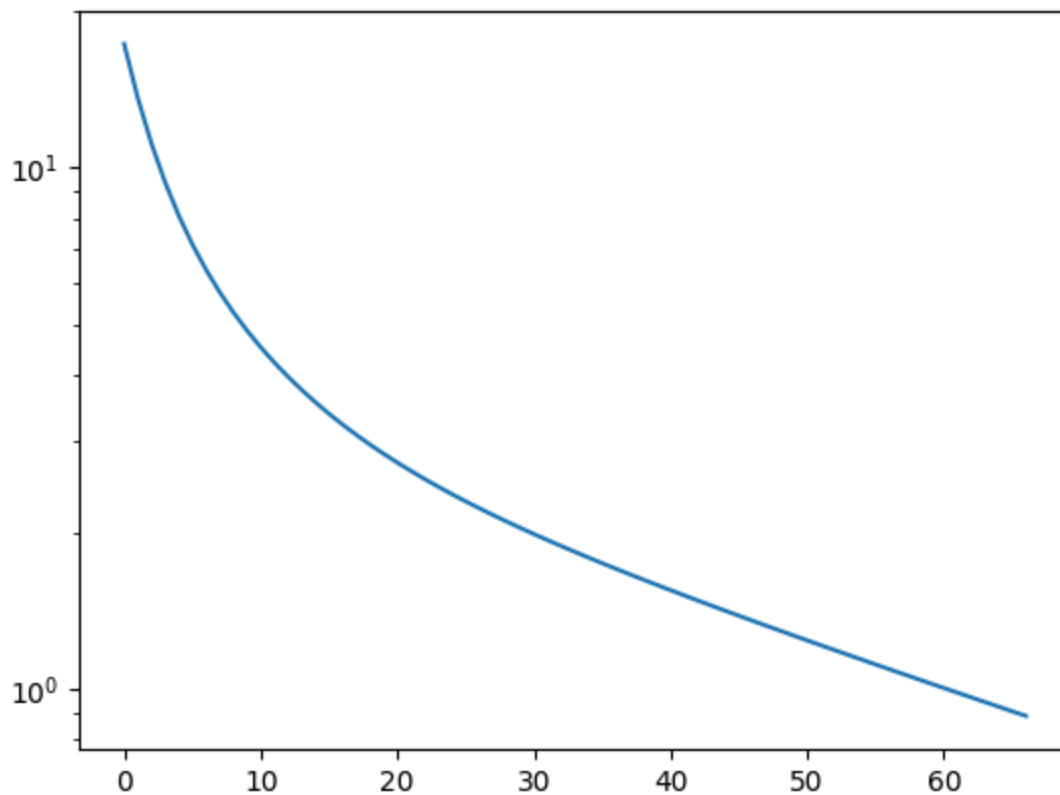
## 11.2 Ejercicio 2.

Realiza los siguiente gráficos de la solución anterior: 1. Contornos llenos ( `contourf` ) y líneas de contorno negras sobrepuestas ( `contour` ). 2. Almacena el error en cada iteración y gráficalo en semi-log. 3. Realiza las dos gráficos anteriores en un solo renglón.

```
plt.contour(xg, yg, u, cmap='gray', levels=5)
plt.contourf(xg, yg, u, levels=50, cmap='inferno')
```



```
plt.plot(error_lista)
plt.yscale('log')
```



```

fig = plt.figure()

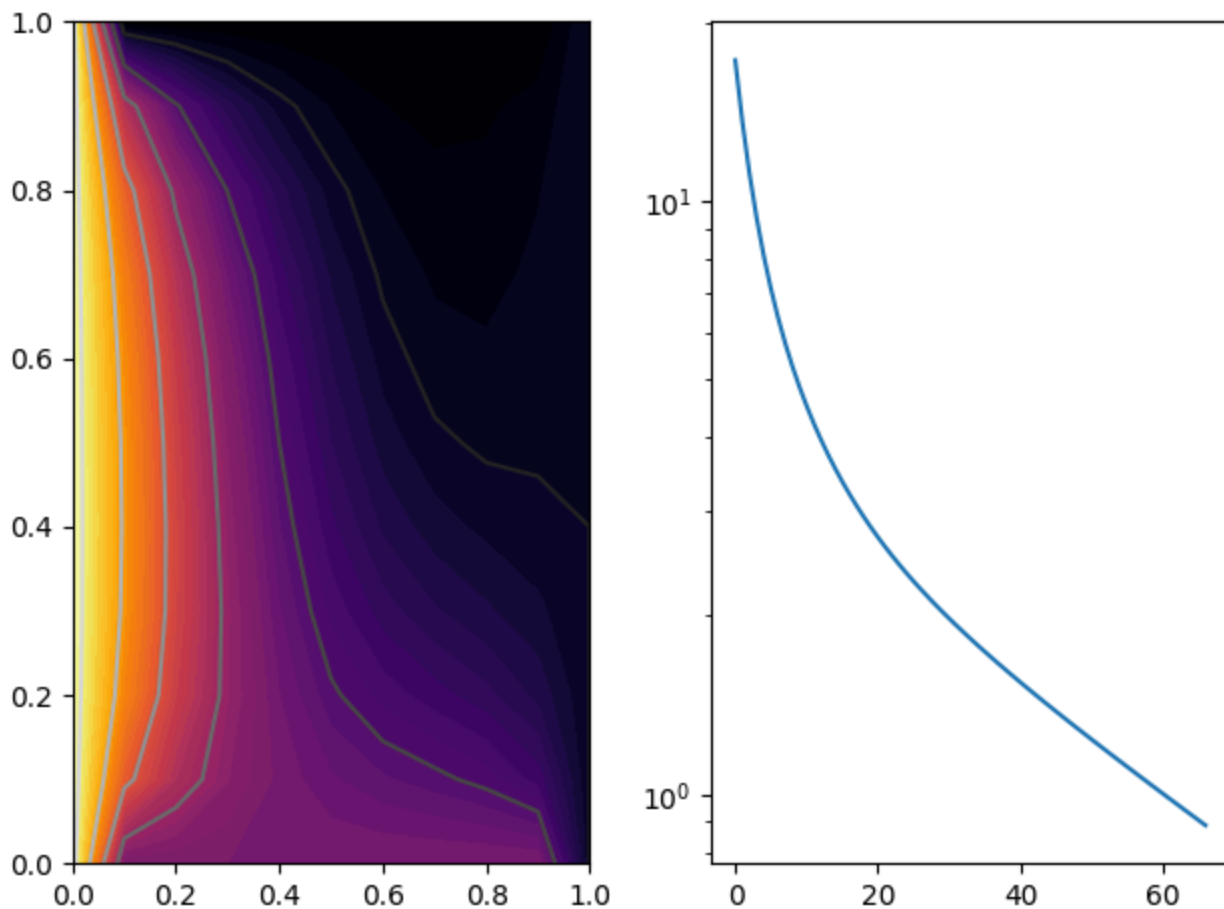
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

ax1.contour(xg, yg, u, cmap='gray', levels=5)
ax1.contourf(xg, yg, u, levels=50, cmap='inferno')

ax2.plot(error_lista)
ax2.set_yscale('log')

plt.tight_layout()

```



## 11.3 Flujo de calor

Fourier también estableció una ley para el flujo de calor que se escribe como:

$$\vec{q} = -\kappa \nabla u = -\kappa \left( \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \right)$$

## 11.4 Ejercicio 3.

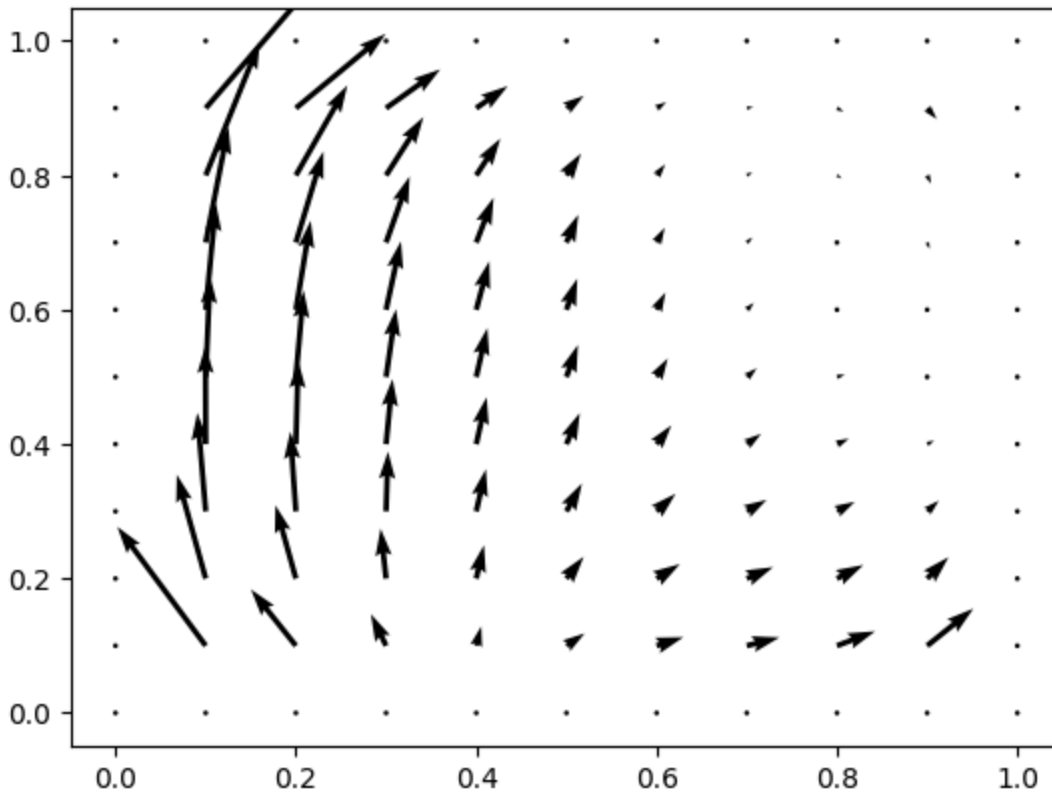
Usando la información calculada de la temperatura (almacenada en el arreglo `u`), vamos a calcular el flujo de calor usando la siguiente fórmula en diferencias:

$$\vec{q}_{i,j} = (qx_{i,j}, qy_{i,j}) = -\frac{\kappa}{2h}(u_{i+1,j} - u_{i-1,j}, u_{i,j+1} - u_{i,j-1})$$

```
qx = np.zeros((Nx+2, Ny+2))
qy = qx.copy()

s = k / 2*h
for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 qx[i,j] = -s * (u[i+1,j] - u[i-1,j])
 qy[i,j] = -s * (u[i,j+1] - u[i,j-1])

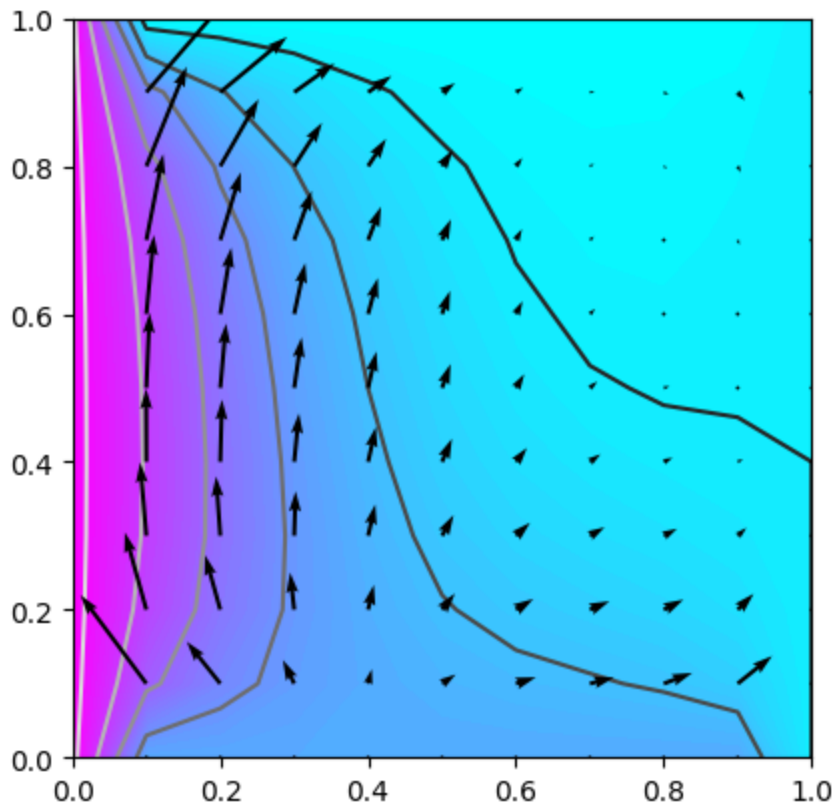
plt.quiver(xg, yg, qx, qy, scale=10, zorder=10)
```



## 11.5 Ejercicio 4.

Grafica el campo vectorial del flujo de calor, junto con los contornos de la temperatura ( `contourf` y `contour` ). Haz que tu gráfica se vea con razón de aspecto correcta de 1 por 1.

```
plt.contour(xg, yg, u, cmap='gray', levels=5)
plt.contourf(xg, yg, u, levels=50, cmap='cool', zorder=1)
plt.quiver(xg, yg, qx, qy, scale=10, zorder=10)
ax = plt.gca()
ax.set_aspect('equal')
```



## 12 Seguimiento de partículas

Si soltamos una partícula en un flujo, dicha partícula seguirá la dirección del flujo y delinearé una trayectoria como se muestra en la siguiente figura. Para calcular los puntos de la trayectoria debemos resolver una ecuación como la siguiente:

$$\frac{\partial \vec{x}}{\partial t} = \vec{v} \quad \text{con} \quad \vec{x}(t=0) = \vec{x}_o$$

donde  $\vec{x} = (x, y)$  representa la posición de la partícula y  $\vec{v} = (v_x, v_y)$  su velocidad. El método más sencillo para encontrar las posiciones de la partícula es conocido como de *Euler hacia adelante* y se escribe como:

$$\vec{x}_i^{n+1} = \vec{x}_i^n + h_t * \vec{v}_i^n$$

donde  $\vec{x}_i^n$  representa la posición de la partícula  $i$  en el instante  $n$ ,  $h_t$  es el paso de tiempo y  $\vec{v}_i^n$  es la velocidad en la partícula  $i$  en el instante  $n$ .

## 12.1 Ejercicio 5.

Calcular y graficar las trayectorias de varias partículas usando el campo vectorial generado por el flujo de calor del ejemplo 2.

Escribimos la fórmula de *Euler hacia adelante* en componentes como sigue:

$$\begin{aligned}x_i^{n+1} &= x_i^n + h_t * vx_i^n \\ y_i^{n+1} &= y_i^n + h_t * vy_i^n\end{aligned}$$

### 1. Definimos un punto inicial de forma aleatoria en el cuadrado unitario:

```
xo = 0.2 #np.random.rand(1)
yo = 0.5 #np.random.rand(1)
print(xo)
print(yo)
```

0.2

0.5

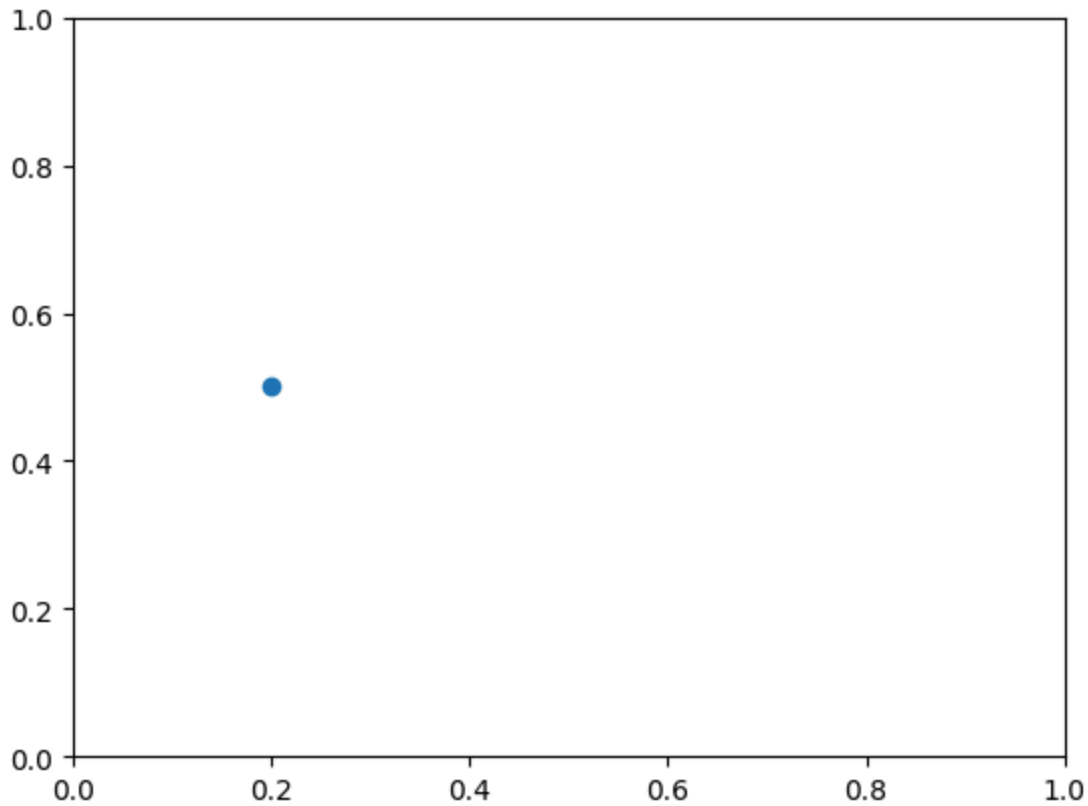
### 2. Definimos arreglos para almacenar las coordenadas de la trayectoria:

```
Pasos = 10
xp = np.zeros(Pasos)
yp = np.zeros(Pasos)
xp[0] = xo
yp[0] = yo
print(xp)
print(yp)
```

[0.2 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0.5 0. 0. 0. 0. 0. 0. 0. 0. 0.]

```
plt.plot(xp[0], yp[0], 'o-')
plt.xlim(0,1)
plt.ylim(0,1)
```



### 3. Implementamos el método de Euler hacia adelante:

```
Interpolación de la velocidad
def interpolaVel(qx, qy, xpi, ypi, h):
 # localizamos la partícula dentro de la rejilla:
 li = int(xpi/h)
 lj = int(ypi/h)
 return (qx[li,lj], qy[li,lj])
```

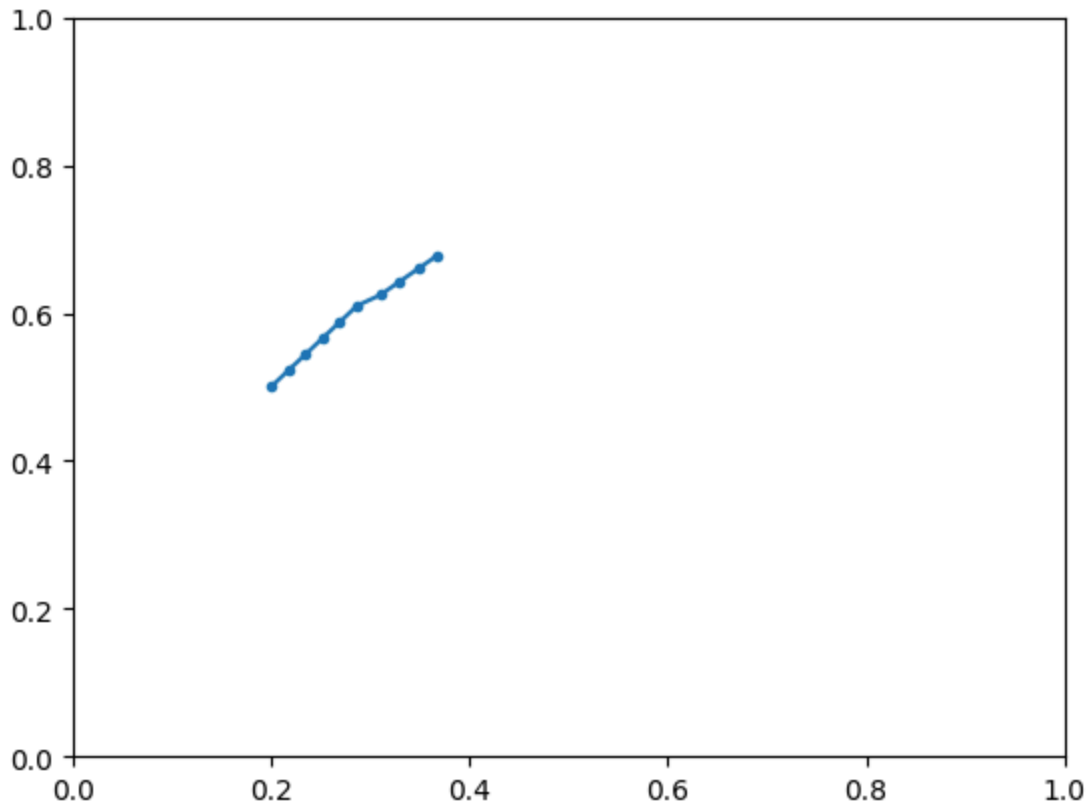
```
ht = 0.1
for n in range(1,Pasos):
 vx, vy = interpolaVel(qx, qy, xp[n-1], yp[n-1], h)
 xp[n] = xp[n-1] + ht * vx
 yp[n] = yp[n-1] + ht * vy
```

```
print(xp)
print(yp)
```

```
[0.2 0.21738397 0.23476794 0.25215191 0.26953588 0.28691984
 0.31035226 0.32940321 0.34845415 0.36750509]
[0.5 0.52197449 0.54394898 0.56592346 0.58789795 0.60987244
 0.62441928 0.64213907 0.65985885 0.67757864]
```

```
plt.plot(xp, yp, '-.')
plt.xlim(0,1)
```

```
plt.ylim(0,1)
```

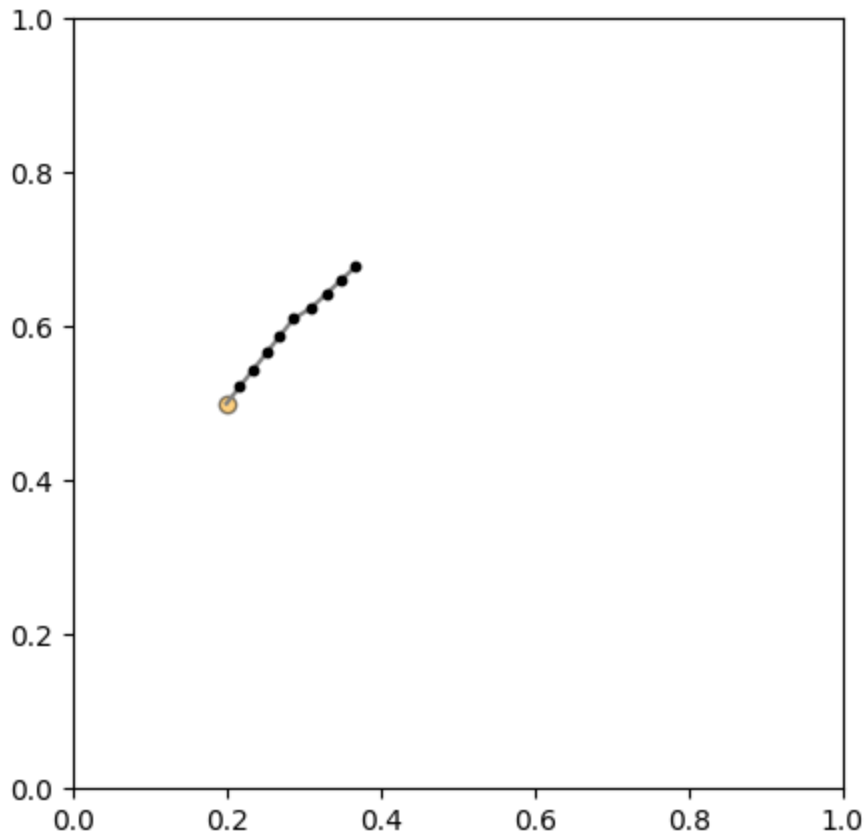


## 12.2 Ejercicio 6.

Dibuja la trayectoria de la siguiente manera. - El primer punto color naranja transparente y contorno negro. - Las posiciones siguientes de color negro sobre puestas sobre la trayectoria. - La trayectoria de color gris. - Verifica que la trayectoria no se salga del cuadrado unitario.

```
plt.figure(figsize=(5,5))
plt.scatter(xp[0], yp[0], c='orange', edgecolor='k', alpha=0.5)
plt.plot(xp, yp, c='gray')
plt.scatter(xp[1:], yp[1:], c='k', s=10, zorder=5)
plt.xlim(0,1)
plt.ylim(0,1)
ax = plt.gca()
ax.set_aspect('equal')
plt.savefig('trayectoria1.pdf')
```





### 12.3 Ejercicio 7.

---

Dibuja varias trayectorias que inicien en sitios diferentes.

### 12.4 Ejercicio 8.

---

Implementa una interpolación bilineal para calcular la velocidad.



## 6 Conducción de calor estacionaria en 1D.

### Objetivo.

Resolver el siguiente problema usando diferencias finitas:

$$-\left(k \frac{d^2 T(x)}{dx^2} + \omega^2 T(x)\right) = 0, \quad x \in [0,1] \tag{1}$$


$$T(0) = T_A$$

$$T(1) = T_B$$

donde  $T_A = T_B = k = 1$  y cuya solución analítica es:

$$T(x) = \frac{1 - \cos(\omega)}{\sin(\omega)} \sin(\omega x) + \cos(\omega x)$$

donde  $\omega = \text{constante}$ .

[MACTI-Analisis\\_Numerico\\_01](#) by [Luis M. de la Cruz](#) is licensed under [Attribution-ShareAlike 4.0 International](#) 

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101019, PE101922

```
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis
from macti.evaluation import *
```

```
quizz = Quizz('q01', 'notebooks', 'local')
```

### 6.1 Ejercicio 1.

Definir los parámetros del problema:

- Parámetros físicos:  $L, T_A, T_B, k, S, w = 2.5 * \pi$
- Parámetros numéricos:  $N = 39, h$
- Coordenadas de la malla:  $x$

```
Parámetros físicos: L, TA, TB, k, S, w = 2.5 * pi
L = ...
...

Parámetros numéricos: N, h
N = ...
...
```

```
Coordenadas de la malla
x = ...
```

```
BEGIN SOLUTION
Parámetros físicos: L, TA, TB, k, S, w = 2.5 * pi
L = 1.0
TA = 1.0
TB = 1.0
k = 1.0
S = 0.0
w = 2.5 * np.pi

Parámetros numéricos: N, h
N = 39
h = L / (N+1)

Coordenadas de la malla
x = np.linspace(0, L, N+2)

file_answer = FileAnswer()
file_answer.write('1', h, 'Checa el cómo debe calcularse h.')
file_answer.write('2', x, 'Debes usar x = np.linspace(...) y poner los argumentos')
END SOLUTION

Parámetros físicos: L, TA, TB, k, S, w = 2.5 * pi

Parámetros numéricos: N, h

Coordenadas de la malla
```

Creando el directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/`  
 Respuestas y retroalimentación almacenadas.

```
quizz.eval_numeric('1', h)
```

-----  
 1 | Tu resultado es correcto.  
 -----

```
quizz.eval_numeric('2', x)
```

-----  
 2 | Tu resultado es correcto.  
 -----

## 6.2 Ejercicio 2.

Definir lo siguiente:

- Arreglo para almacenar la solución: `T`
- Valores de la temperatura conocidos en las fronteras: `T[0]` y `T[-1]`
- Lado derecho del sistema: `b`
- Aplicación de las condiciones de frontera tipo Dirichlet: `b[0]` y `b[-1]`
- Matriz del sistema lineal. Para ello usa la función `buildMatrix(...)` con los parámetros correctos.
- Solución del sistema lineal usando la función `np.linalg.solve(...)`.

```
Arreglo para almacenar la solución
T = ...

Valores de la temperatura conocidos en las fronteras:
T[0] = ...
T[-1] = ...

Lado derecho del sistema
b = ...

Aplicación de las condiciones de frontera tipo Dirichlet
b[0] = ...
b[-1] = ...

Matriz del sistema lineal
w = ...
A = ...

Solución del sistema lineal
T[1:-1] = ...
```

```
def buildMatrix(N, d):
 """
 Parameters:
 N: int Tamaño de la matriz.
 d: float Contenido de la diagonal.
 """
 # Matriz de ceros
 A = np.zeros((N,N))

 # Primer renglón
 A[0,0] = d
 A[0,1] = -1

 # Renglones interiores
 for i in range(1,N-1):
 A[i,i] = d
```

```

 A[i,i+1] = -1
 A[i,i-1] = -1

Último renglón
A[N-1,N-2] = -1
A[N-1,N-1] = d

return A

```

```

BEGIN SOLUTION
Arreglo para almacenar la solución
T = np.zeros(N+2)

Los valores en los extremos son conocidos debido a las cond. Dirichlet
T[0] = TA
T[-1] = TB

Lado derecho del sistema
b = np.zeros(N)

Aplicacion de las condiciones de frontera Dirichlet
b[0] += TA
b[-1] += TB

file_answer.write('3', b, 'Checa el tamaño correcto de b y los valores en los extremos')

Construcción de la matriz
w = 2.5 * np.pi
A = buildMatrix(N, 2-(w * h)**2) # Matriz del sistema

file_answer.write('4', A.diagonal(), 'Debes usar A = buildMatrix(...) y poner los valores en la diagonal')

Solución del sistema lineal
T[1:-1] = np.linalg.solve(A,b)

file_answer.write('5', T, 'Checa el tamaño correcto de T y los valores en los extremos')

END SOLUTION
Arreglo para almacenar la solución

Los valores en los extremos son conocidos debido a las cond. Dirichlet

Lado derecho del sistema

Aplicacion de las condiciones de frontera Dirichlet

Construcción de la matriz

```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe  
 Respuestas y retroalimentación almacenadas.

```
quizz.eval_numeric('3', b)
```

-----  
3 | Tu resultado es correcto.  
-----

```
quizz.eval_numeric('4', A.diagonal())
```

-----  
4 | Tu resultado es correcto.  
-----

```
quizz.eval_numeric('5', T,)
```

-----  
5 | Tu resultado es correcto.  
-----

## 6.3 Ejercicio 3.

- Agregar una función para calcular la solución exacta.

```
def solExact(x, w):
 ...
```

- Calcular el error entre solución exacta y numérica.

```
BEGIN SOLUTION
Agrega la función: def solExact(x, w):
def solExact(x,w):
 return ((1.0 - 1.0 * np.cos(w))/np.sin(w)) * np.sin(w * x) + np.cos(w * x)

Cálculo del error:
Error = np.linalg.norm(solExact(x,w) - T, 2)

file_answer.write('6', Error, 'Checa la implementación de la solución exacta y el

END SOLUTION
Agrega la función: def solExact(x, w):

Cálculo del error:

print(Error)
```

El directorio `:/home/jovyan/macti/notebooks/.ans/Diferencias_finitas_01/` ya existe  
Respuestas y retroalimentación almacenadas.

0.04614723768419929

quizz.eval\_numeric('6', Error)

-----

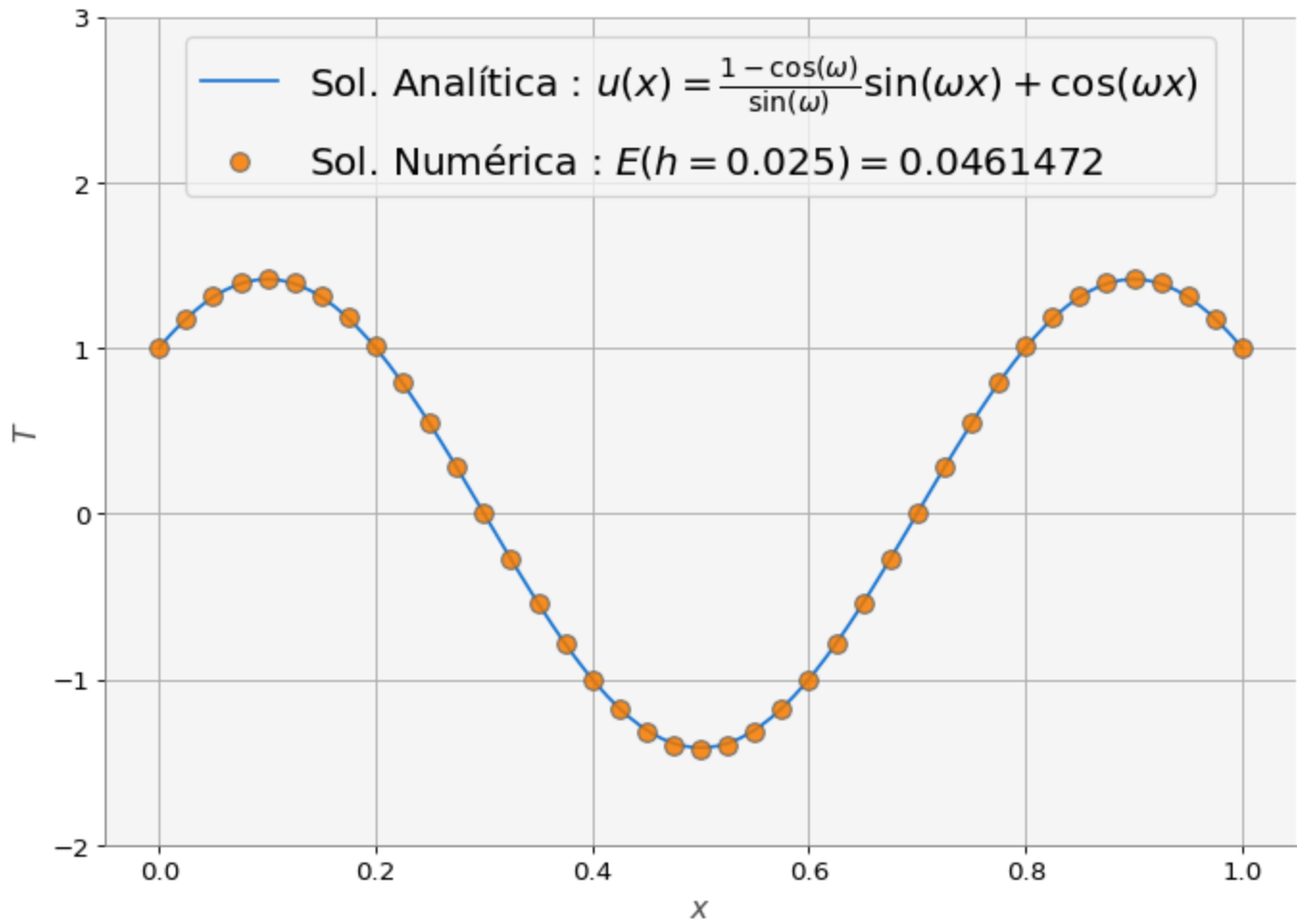
6 | Tu resultado es correcto.

-----

```
#
El código siguiente realiza las gráficas usando lo calculado en las celdas ante
#
titulo = 'Ecuación : $\frac{\partial^2 u(x)}{\partial x^2} = \omega^2 u(x); \backslash, \backslash, \backslash, u(a) =$
numerica = 'Sol. Numérica : $E(h = \%g) = \%g$ ' % (h, Error)
exacta = 'Sol. Analítica : $u(x) = \frac{1 - \cos(\omega)}{\sin(\omega)} \sin(\omega x)$ '

plt.figure(figsize=(10,7))
xsol = np.linspace(0,1,100)
plt.plot(xsol, solExact(xsol,w),'-', label=exacta)
plt.scatter(x, T, fc = 'C1', ec='dimgray', s=75, alpha=0.85, zorder= 10, label=numerica)
plt.title(titulo)
plt.xlabel('x')
plt.ylabel('T')
plt.ylim(-2.0,3.0)
plt.legend(loc='upper center', fontsize=18)
plt.grid()
plt.show()
```

Ecuación :  $\partial^2 u(x)/\partial x^2 = \omega^2 u(x)$ ;  $u(a) = u(b) = 1$







## 13 Contaminante en un flujo de agua subterráneas.

**Objetivo.** Resolver numéricamente el transporte de un contaminante en un acuífero.

[MACTI NOTES](#) by Luis Miguel de la Cruz Salas is licensed under [CC BY-NC-SA 4.0](#)

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

```
import sys
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis

from macti.evaluation import *

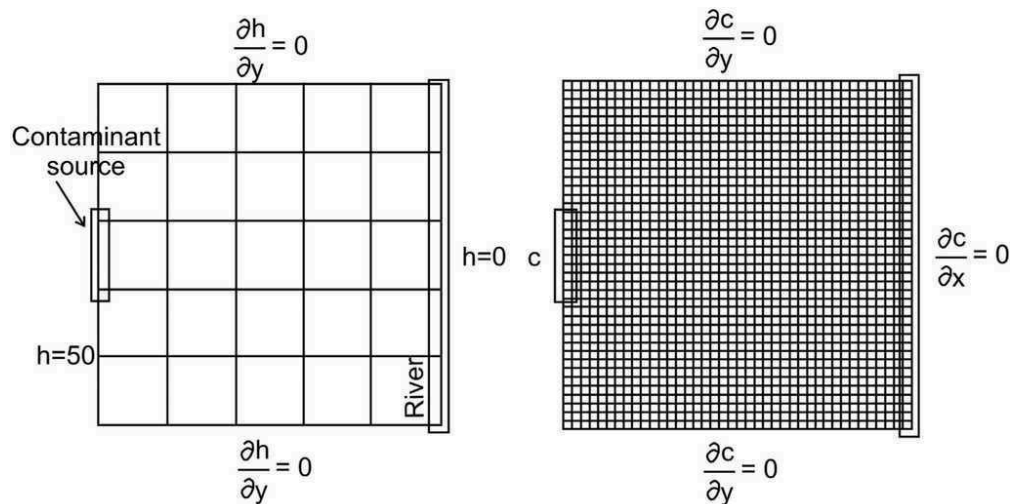
print('Python', sys.version)
print(np.__name__, np.__version__)
print(plt.matplotlib.__name__, plt.matplotlib.__version__)
```

Python 3.11.6 | packaged by conda-forge | (main, Oct 3 2023, 10:40:35) [GCC 12.3.0]  
numpy 1.26.2  
matplotlib 3.8.1

```
quizz = Quizz('1', 'notebooks', 'local')
```

## Modelo conceptual.

Consideremos un acuífero de  $Lx = 804.7$  [m] y  $Ly = 804.7$  [m] y una fuente de contaminante localizada en la pared izquierda y acotado por un río en la pared derecha, véase figura. Se considera que el contaminante que se modela es conservativo, es decir, que su concentración no varía al interactuar con el medio y que, por tanto, al atravesar el acuífero mantiene sus propiedades durante todo su recorrido. Se cuenta con un modelo de flujo y transporte de una sola capa, en dos dimensiones. El flujo del agua en el acuífero está en estado de equilibrio. Las condiciones de frontera son las que se muestran en la figura, para la carga hidráulica  $h$  y para la concentración  $c$ .



El valor de la carga hidráulica en la pared izquierda es de  $h = 50$  [m] y en la pared derecha es de  $h = 0$  [m], en las otras paredes se considera no flujo. La fuente de contaminante que está activa durante el periodo de simulación tiene un valor constante de  $c = 50$  ppm. En los otros lugares de la frontera se considera no flujo del contaminante. Inicialmente  $h = 25$  [m] y  $c = 0$  ppm en el interior del dominio.

La porosidad tiene un valor de  $\phi = 0.25$ , la dispersividad en dirección  $x$  tiene un valor de  $Dx = 33$  [m] y en dirección  $y$  tiene el valor de  $Dy = 3.3$  [m].  $K = 21.22$  [m/s] y  $S_s = 1.0$

**Fuente:** Leyva-Suárez, Esther, Herrera, Graciela S., & de la Cruz, Luis M.. (2015). A parallel computing strategy for Monte Carlo simulation using groundwater models. *Geofísica internacional*, 54(3), 245-254.

<https://doi.org/10.1016/j.gi.2015.04.020>

```
Parámetros físicos
K = 1.0 # 21.22 # Conductividad
Dx = 1.0 # 33.0
Dy = 1.0 # 3.3
phi = 1.0 # 0.25
Lx = 1.0 # 804.7 # Longitud del dominio en dirección x
Ly = 1.0 # 804.7 # Longitud del dominio en dirección y

print('Parámetros físicos' + '\n' + 40*'-')
print('Conductividad K = {}'.format(K))
print('Conductividad (Dx, Dy) = ({} , {})'.format(Dx, Dy))
print('Porosidad phi = {}'.format(phi))
print('Longitud en x = {} | Longitud en y = {}'.format(Lx, Ly))
```

#### Parámetros físicos

```

Conductividad K = 1.0
Conductividad (Dx, Dy) = (1.0, 1.0)
Porosidad phi = 1.0
Longitud en x = 1.0 | Longitud en y = 1.0
```

## Modelo matemático. Para este modelo usaremos las ecuaciones de flujo y transporte acopladas por la ley de Darcy para describir la evolución de la pluma del contaminante. Estas ecuaciones se van a resolver para las cargas

hidráulicas y las concentraciones del contaminante y se escriben como sigue:

$$S_s \frac{\partial h}{\partial t} = K \left( \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right) \quad (1)$$

$$V = -K \nabla h \quad (2)$$

$$\phi \frac{\partial c}{\partial t} + V \cdot \nabla c = D_x \frac{\partial^2 c}{\partial x^2} + D_y \frac{\partial^2 c}{\partial y^2} \quad (3)$$

donde  $S_s$  es el almacenamiento específico,  $K$  es la conductividad hidráulica,  $h$  la carga hidráulica,  $c$  la concentración del soluto,  $D = (D_x, D_y)$  es la dispersión hidrodinámica,  $V = (V_x, V_y)$  la velocidad de poro y la  $\phi$  porosidad efectiva.

La ecuación de flujo (1) describe el flujo del agua a través del acuífero, la ecuación de transporte (3) describe los cambios en la concentración del contaminante a través del tiempo para un soluto conservativo. La ley de Darcy (2) acopla las ecuaciones (1) y (3) y con ella se calcula la velocidad de poro del agua subterránea utilizando las cargas y la conductividad hidráulica.

Las condiciones de frontera, de acuerdo con la figura de la sección anterior, son las siguientes:

$$\begin{aligned} h(t, x, y) &= 50 [m] && \text{para } x = 0, \quad \forall t \\ h(t, x, y) &= 0 [m] && \text{para } x = Lx, \quad \forall t \\ \frac{\partial h(t, x, y)}{\partial y} &= 0 && \text{para } y = 0, Ly, \quad \forall t \\ c(t, x, y) &= 50 [ppm] && \text{para } x = 0, \quad y \in [\frac{3}{8}Ly, \frac{5}{8}Ly], \quad \forall t \\ \frac{\partial c(t, x, y)}{\partial y} &= 0 && \text{para la frontera restante, } \forall t \end{aligned}$$

## Modelo numérico.

El dominio se discretiza en una malla de  $40 \times 40$ . Se va a simular durante 48 pasos de tiempo, con un paso de 15.2083 días.

La forma discreta del modelo matemático, ecuaciones (1), (2) y (3), usando diferencias finitas de segundo orden es la siguiente:

$$\begin{aligned} h_{i,j}^{n+1} &= h_{i,j}^n + \frac{\delta_t K_{i,j}}{\delta^2} (h_{i+1,j}^n + h_{i-1,j}^n + h_{i,j+1}^n + h_{i,j-1}^n - 4h_{i,j}^n) \\ (Vx_{i,j}, Vy_{i,j}) &= -\frac{K_{i,j}}{2\delta} (h_{i+1,j} - h_{i-1,j}, h_{i,j+1} - h_{i,j-1}) \\ c_{i,j}^{n+1} &= c_{i,j}^n + \frac{\delta_t D_{x,i,j}}{\delta^2 \phi} (c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n) + \frac{\delta_t D_{y,i,j}}{\delta^2 \phi} (c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n) - \\ &\quad \frac{\delta_t Vx_{i,j}}{2\delta \phi} (c_{i+1,j} - c_{i-1,j}) - \frac{\delta_t Vy_{i,j}}{2\delta \phi} (c_{i,j+1} - c_{i,j-1}) \end{aligned}$$

donde  $\delta$  representa el tamaño de las celdas en ambas direcciones y  $\delta_t$  el paso de tiempo.

```
Parámetros numéricos
Nx = 28 # Número de incógnitas en dirección x
Ny = 28 # Número de incógnitas en dirección y
δ = Lx / (Nx+1) # Espaciamiento entre los puntos de la rejilla
δt = 0.001 # Paso de tiempo
N = (Nx + 2)* (Ny + 2) # Número total de puntos en la rejilla

print('\nParámetros numéricos' + '\n' + 40*'-')
print('Nodos en x = {} | Nodos en y = {}'.format(Nx+2,Ny+2))
print('δ = {} | δt = {}'.format(δ, δt))
```

#### Parámetros numéricos

```

Nodos en x = 30 | Nodos en y = 30
δ = 0.034482758620689655 | δt = 0.001
```

## Modelo computacional.

### Algoritmo. Los pasos a seguir son los siguientes.

#### 1. Definir los parámetros físicos y numéricos del problema: (ya se definieron antes)

```
print('Parámetros físicos' + '\n' + 40*'-')
print('Conductividad K = {}'.format(K))
print('Conductividad (Dx, Dy) = ({}, {})'.format(Dx, Dy))
print('Porosidad φ = {}'.format(φ))
print('Longitud en x = {} | Longitud en y = {}'.format(Lx,Ly))
print('\nParámetros numéricos' + '\n' + 40*'-')
print('Nodos en x = {} | Nodos en y = {}'.format(Nx+2,Ny+2))
print('δ = {} | δt = {}'.format(δ, δt))
```

#### Parámetros físicos

```

Conductividad K = 1.0
Conductividad (Dx, Dy) = (1.0, 1.0)
Porosidad φ = 1.0
Longitud en x = 1.0 | Longitud en y = 1.0
```

#### Parámetros numéricos

```

Nodos en x = 30 | Nodos en y = 30
δ = 0.034482758620689655 | δt = 0.001
```

#### 2. Definir la rejilla donde se hará el cálculo (malla):

### Ejercicio 1. Calcule los arreglos x y y correctos para generar la malla.

```
BEGIN SOLUTION
x = np.linspace(0,Lx,Nx+2) # Arreglo con las coordenadas en x
```

```

y = np.linspace(0,Ly,Ny+2) # Arreglo con las coordenadas en y

file_answer = FileAnswer()
file_answer.write('1', x, 'Revisa la definición de las coordenadas en dirección x')
file_answer.write('2', y, 'Revisa la definición de las coordenadas en dirección y')
END SOLUTION

xg, yg = np.meshgrid(x,y,indexing='ij', sparse=False) # Creamos la rejilla para u

```

Creando el directorio :/home/jovyan/macti\_notes/notebooks/.ans/RAUGM2023/  
Respuestas y retroalimentación almacenadas.

```

quizz.eval_numeric('1', x)
quizz.eval_numeric('2', y)

```

-----  
1 | Tu resultado es correcto.  
-----  
-----

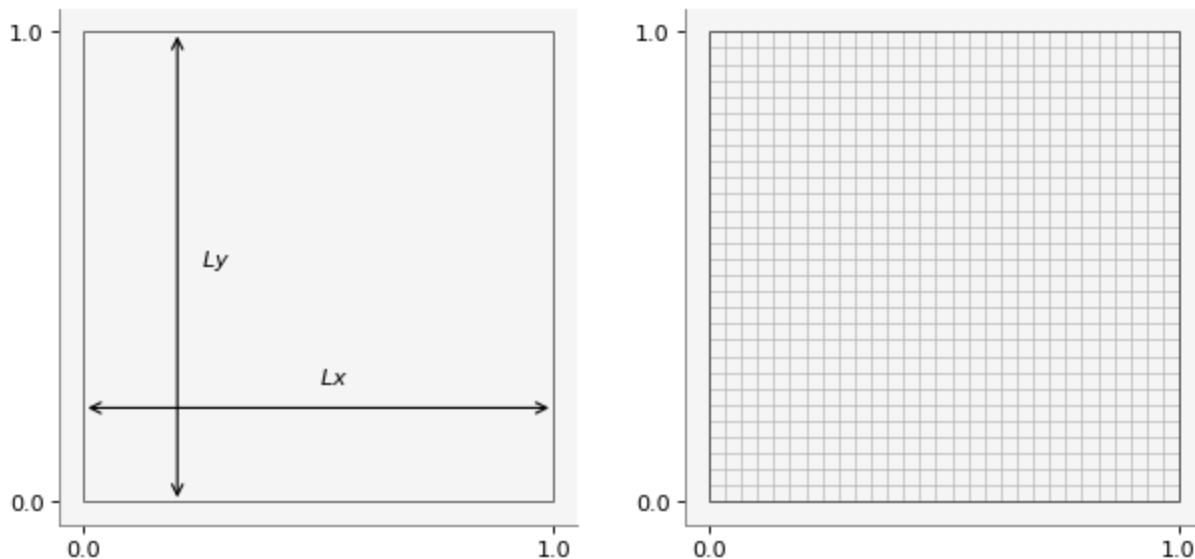
2 | Tu resultado es correcto.  
-----  
-----

```

vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize=(10,10)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg)
vis.plot_frame(2, xg, yg)

```



### 3. Definir las condiciones iniciales y de frontera:

Para el caso de la concentración, necesitamos definir la fuente de contaminante en la pared izquierda:

### Ejercicio 2. Calcule el lugar donde se debe aplicar la condición de frontera distinta de cero para la concentración  $c$ . Recuerde que el intervalo es  $y \in [\frac{3}{8}Ly, \frac{5}{8}Ly]$ .

```
h = np.zeros((Nx+2, Ny+2))
h[0,:] = 50 # Pared izquierda
h[Nx+1,:] = 0 # Pared derecha

c = np.zeros((Nx+2, Ny+2))
c[0,:] = 0 # Pared izquierda
c[Nx+1,:] = 0 # Pared derecha
c[:,0] = 0 # Pared inferior
c[:,Ny+1] = 0 # Pared superior

BEGIN SOLUTION
N1 = int((Ly * 3.0 / 8.0) / delta)
N2 = int((Ly * 5.0 / 8.0) / delta)

file_answer.write('3', N1, 'Revisa el cálculo de límite inferior del intervalo')
file_answer.write('4', N2, 'Revisa el cálculo de límite superior del intervalo')
END SOLUTION
c[0, N1:N2] = 50 # Pared izquierda

print('N1 = {}'.format(N1))
print('N2 = {}'.format(N2))
```

N1 = 10

N2 = 18

```
quizz.eval_numeric('3', N1)
quizz.eval_numeric('4', N2)
```

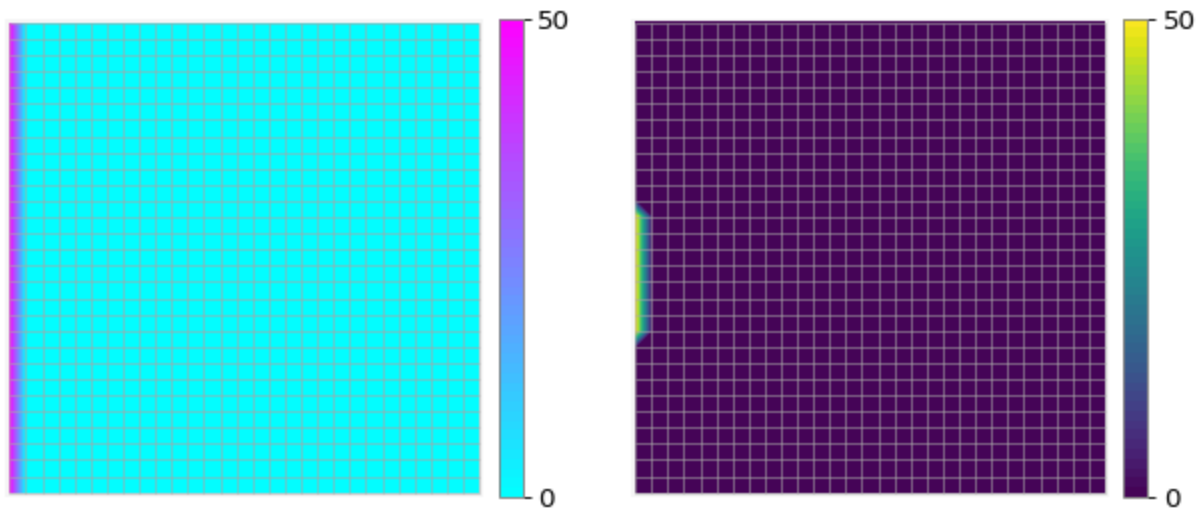
-----  
3 | Tu resultado es correcto.  
-----  
-----

4 | Tu resultado es correcto.  
-----

```
vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize=(10,10)))

cax1 = vis.set_canvas(1,Lx,Ly)
c_h = vis.contourf(1, xg, yg, h, levels=50, cmap='cool')
vis.fig.colorbar(c_h, cax=cax1, ticks = [h.min(), h.max()], shrink=0.5, orientation='vertical')
vis.plot_mesh2D(1, xg, yg)

cax2 = vis.set_canvas(2,Lx,Ly)
c_c = vis.contourf(2, xg, yg, c, levels=50, cmap='viridis')
vis.fig.colorbar(c_c, cax=cax2, ticks = [c.min(), c.max()], shrink=0.5, orientation='vertical')
vis.plot_mesh2D(2, xg, yg)
```



#### 4. Implementar el algoritmo de solución:

$$h_{i,j}^{n+1} = h_{i,j}^n + \frac{\delta_t K_{i,j}}{\delta^2} (h_{i+1,j}^n + h_{i-1,j}^n + h_{i,j+1}^n + h_{i,j-1}^n - 4h_{i,j}^n)$$

$$(Vx_{i,j}, Vy_{i,j}) = - \frac{K_{i,j}}{2\delta} (h_{i+1,j} - h_{i-1,j}, h_{i,j+1} - h_{i,j-1})$$

$$c_{i,j}^{n+1} = c_{i,j}^n + \frac{\delta_t D x_{i,j}}{\delta^2} (c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n) + \frac{\delta_t D y_{i,j}}{\delta^2} (c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n) - \frac{\delta_t V x_{i,j}}{2\delta} (c_{i+1,j} - c_{i-1,j}) - \frac{\delta_t V y_{i,j}}{2\delta} (c_{i,j+1} - c_{i,j-1})$$

```

delta_t = 0.0001
r = delta_t / delta**2
s = 1.0 / 2*delta
t = delta_t / 2*delta

h_new = h.copy()
c_new = c.copy()

tolerancia = 1.0e-3
error = 1.0
error_lista = []

Vx = np.zeros((Nx+2, Ny+2))
Vy = Vx.copy()

```

#### 4.1 Hacemos una prueba resolviendo primero la ecuación de flujo

### Ejercicio 3. Implemente la fórmula en diferencias para la carga hidráulica  $h$ .

```

iteraciones_max = 1000
iteraciones = 0
while(error > tolerancia and iteraciones < iteraciones_max):
 iteraciones += 1

```

```

for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 if j == 1: # No flujo en la pared inferior
 h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] - 3 * h[i,j])
 if j == Ny: # No flujo en la pared superior
 h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j-1] - 3 * h[i,j])
 else:
 h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] + h[i,j-1] - 4 * h[i,j])

Condición de frontera de no flujo
h_new[:,0] = h_new[:,1]
h_new[:,Ny+1] = h_new[:,Ny]

Actualización de la carga hidráulica
h[:] = h_new[:]

print(iteraciones, sep=' ', end= ' ')

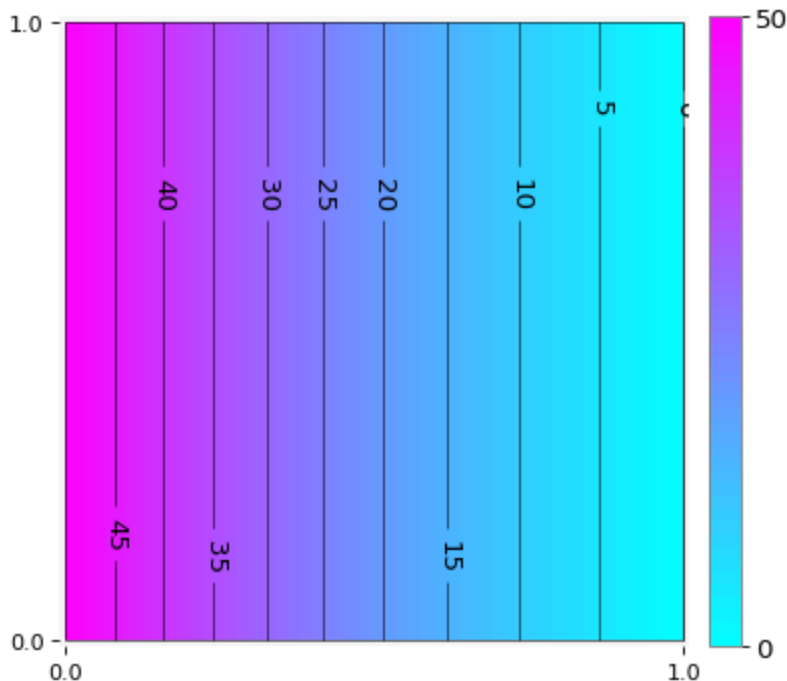
```

```

vis = mvis.Plotter(1,1,[dict(aspect='equal')])

cax1 = vis.set_canvas(1,Lx,Ly)
c_h = vis.contourf(1, xg, yg, h, levels=50, cmap='cool')
cl_h = vis.contour(1, xg, yg, h, levels=10, colors='k', linewidths=0.5)
plt.clabel(cl_h, inline=True, fontsize=12.0)
vis.fig.colorbar(c_h, cax=cax1, ticks = [h.min(), h.max()], shrink=0.5, orientation='vertical')
vis.plot_frame(1, xg, yg)

```



## 4.2 Calculamos ahora la velocidad con la $h$ antes calculada

### Ejercicio 4. Calcule la velocidad con la fórmula correspondiente.



```

BEGIN SOLUTION
for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 Vx[i,j] = -K * s * (h[i+1,j] - h[i-1,j])
 Vy[i,j] = -K * s * (h[i,j+1] - h[i,j-1])

file_answer.write('5', Vx, '')
file_answer.write('6', Vy, '')
END SOLUTION

```

```

quizz.eval_numeric('5', Vx)
quizz.eval_numeric('6', Vy)

```

-----

5 | Tu resultado es correcto.

-----

-----

6 | Tu resultado es correcto.

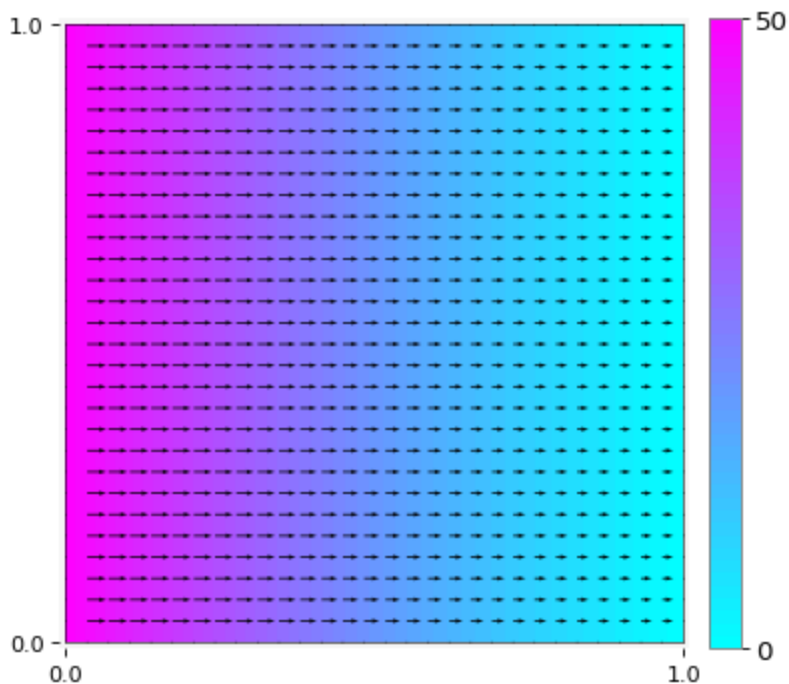
-----

```

vis = mvis.Plotter(1,1,[dict(aspect='equal')])

cax1 = vis.set_canvas(1,Lx,Ly)
c_h = vis.contourf(1, xg, yg, h, levels=50, cmap='cool')
vis.fig.colorbar(c_h, cax=cax1, ticks = [h.min(), h.max()], shrink=0.5, orientati
vis.plot_frame(1, xg, yg)
vis.quiver(1, xg, yg, Vx, Vy, scale=2.5)
vis.show()

```



### 4.3 Implementamos el algoritmo completo

$$h_{i,j}^{n+1} = h_{i,j}^n + \frac{\delta_t K_{i,j}}{\delta^2} (h_{i+1,j}^n + h_{i-1,j}^n + h_{i,j+1}^n + h_{i,j-1}^n - 4h_{i,j}^n)$$

$$(Vx_{i,j}, Vy_{i,j}) = -\frac{K_{i,j}}{2\delta} (h_{i+1,j} - h_{i-1,j}, h_{i,j+1} - h_{i,j-1})$$

$$c_{i,j}^{n+1} = c_{i,j}^n + \frac{\delta_t D x_{i,j}}{\delta^2 \phi} (c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n) + \frac{\delta_t D y_{i,j}}{\delta^2 \phi} (c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n) -$$

$$\frac{\delta_t V x_{i,j}}{2\delta \phi} (c_{i+1,j} - c_{i-1,j}) - \frac{\delta_t V y_{i,j}}{2\delta \phi} (c_{i,j+1} - c_{i,j-1})$$

### Ejercicio 5. Complete el cálculo de la concentración  $c$  en el código que sigue.

```
h = np.ones((Nx+2, Ny+2)) * 25
h[0,:] = 50 # Pared izquierda
h[Nx+1,:] = 0 # Pared derecha

c = np.zeros((Nx+2, Ny+2))
c[0,:] = 0 # Pared izquierda
c[Nx+1,:] = 0 # Pared derecha
c[:,0] = 0 # Pared inferior
c[:,Ny+1] = 0 # Pared superior

c[0, N1:N2] = 50 # Pared izquierda

iteraciones_max = 1000
iteraciones = 0
while(error > tolerancia and iteraciones < iteraciones_max):
 iteraciones += 1
 for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 if j == 1: # No flujo
 h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] - 3*h[i,j])
 if j == Ny: # No flujo
 h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j-1] - 3*h[i,j])
 else:
 h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] + h[i,j-1] - 4*h[i,j])

 h_new[:,0] = h_new[:,1]
 h_new[:,Ny+1] = h_new[:,Ny]

 for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 Vx[i,j] = -K * s * (h[i+1,j] - h[i-1,j])
 Vy[i,j] = -K * s * (h[i,j+1] - h[i,j-1])

 for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 if j == 1: # No flujo
```

```

c_new[i,j] = c[i,j] + Dx * r * (c[i+1,j] - 2*c[i,j] + c[i-1,j]) /
+ Dy * r * (c[i,j+1] - c[i,j]) / ϕ \
- t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
- t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

if j == Ny: # No flujo
 c_new[i,j] = c[i,j] + Dx * r * (c[i+1,j] - 2*c[i,j] + c[i-1,j]) /
+ Dy * r * (c[i,j+1] - c[i,j]) / ϕ \
- t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
- t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

if i == Nx: # No flujo
 c_new[i,j] = c[i,j] + Dx * r * (c[i-1,j] - c[i,j]) / ϕ \
+ Dy * r * (c[i,j+1] - 2*c[i,j] + c[i,j-1]) /
- t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
+ t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

else:
 c_new[i,j] = c[i,j] + Dx * r * (c[i+1,j] - 2*c[i,j] + c[i-1,j]) /
+ Dy * r * (c[i,j+1] - 2*c[i,j] + c[i,j-1]) / ϕ \
- t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
- t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

c_new[:,0] = c_new[:,1]
c_new[:,Ny+1] = c_new[:,Ny]
c_new[Nx+1:] = c_new[Nx,:]

error = np.linalg.norm(h_new - h)
error_lista.append(error)
h[:] = h_new[:]
c[:] = c_new[:]

print(iteraciones, sep=' ', end= ' ')

```

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293
294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337
338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447

```

448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469  
 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491  
 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513  
 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535  
 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557  
 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579  
 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601  
 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623  
 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645  
 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667  
 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689  
 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711  
 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733  
 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755  
 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777  
 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799  
 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821  
 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843  
 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865  
 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887  
 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909  
 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931  
 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953  
 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975  
 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997  
 998 999 1000

```

ax1 = dict(aspect='equal', title='Malla')
ax2 = dict(aspect='equal', title='Carga hidráulica')
ax3 = dict(aspect='equal', title='Velocidad')
ax4 = dict(aspect='equal', title='Concentración')

vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
 dict(figsize=(8,8)))

vis.plot_mesh2D(1, xg, yg)
vis.plot_frame(1, xg, yg)

cax3 = vis.set_canvas(2,Lx,Ly)
c_h = vis.contourf(2, xg, yg, h, levels=50, cmap='cool')
vis.fig.colorbar(c_h, cax=cax3, ticks = [h.min(), h.max()], shrink=0.5, orientati
vis.contour(2, xg, yg, h, levels=10, linewidths=0.5)
vis.plot_frame(2, xg, yg)

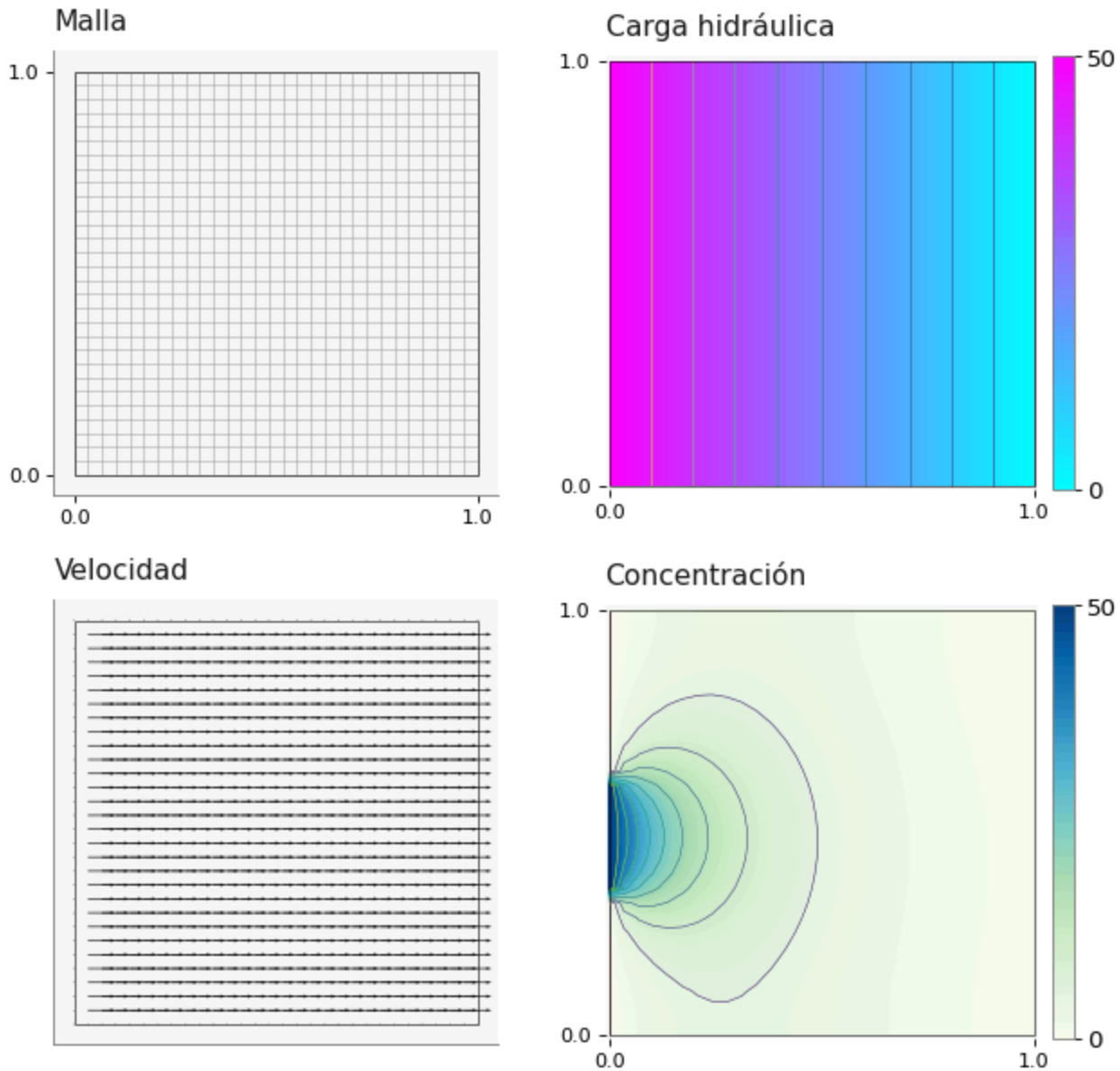
vis.plot_frame(3, xg, yg)
vis.quiver(3, xg, yg, Vx, Vy, scale=1)

cax4 = vis.set_canvas(4,Lx,Ly)
c_c = vis.contourf(4, xg, yg, c, levels=50, cmap='GnBu')
vis.fig.colorbar(c_c, cax=cax4, ticks = [c.min(), c.max()], shrink=0.5, orientati

```

```
vis.contour(4, xg, yg, c, levels=10, linewidths=0.5)
vis.plot_frame(4, xg, yg)

vis.show()
```



**¿Podría hacer el mismo cálculo con una permeabilidad hidráulica y dispersividad variable en el dominio de estudio?**

```
K = np.ones((Nx+2, Ny+2))

nn1 = int(Ny*0.25)
nn2 = int(Ny*.75)
print(nn1, nn2, nn2-nn1)
K[:, nn1:nn2] = np.random.rand(Nx+2, nn2-nn1) * 0.5

Dx = np.random.rand(Nx+2, Ny+2) * 0.5
Dy = np.random.rand(Nx+2, Ny+2) * 2.5
```

7 21 14

```

ax1 = dict(aspect='equal', title='Permeabilidad')
ax2 = dict(aspect='equal', title='Dispersividad x')
ax3 = dict(aspect='equal', title='Dispersividad y')

vis = mvis.Plotter(1,3,[ax1, ax2, ax3],
 dict(figsize=(10,8)))

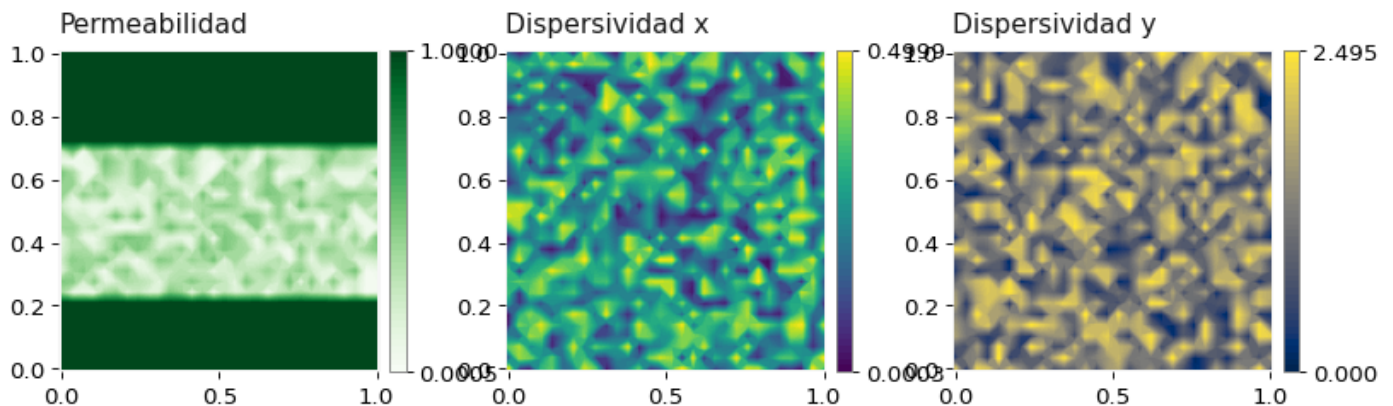
cax1 = vis.set_canvas(1,Lx,Ly)
p_v = vis.contourf(1, xg, yg, K, levels=50, cmap='Greens')
vis.fig.colorbar(p_v, cax=cax1, ticks = [K.min(), K.max()], shrink=0.5, orientati

cax2 = vis.set_canvas(2,Lx,Ly)
d_x = vis.contourf(2, xg, yg, Dx, levels=50, cmap='viridis')
vis.fig.colorbar(d_x, cax=cax2, ticks = [Dx.min(), Dx.max()], shrink=0.5, orienta

cax3 = vis.set_canvas(3,Lx,Ly)
d_y = vis.contourf(3, xg, yg, Dy, levels=50, cmap='cividis')
vis.fig.colorbar(d_y, cax=cax3, ticks = [Dy.min(), Dy.max()], shrink=0.5, orienta

vis.show()

```



### 13.0.1 Ejercicio 6.

Copie y modifique el código que calcula toda la solución, de tal manera que tome en cuenta la permeabilidad hidráulica y la dispersividad variables.

```

BEGIN SOLUTION
h = np.ones((Nx+2, Ny+2)) * 25
h[0,:] = 50 # Pared izquierda
h[Nx+1,:] = 0 # Pared derecha

c = np.zeros((Nx+2, Ny+2))
c[0,:] = 0 # Pared izquierda
c[Nx+1,:] = 0 # Pared derecha
c[:,0] = 0 # Pared inferior

```

```

c[:,Ny+1] = 0 # Pared superior

c[0, N1:N2] = 50 # Pared izquierda

iteraciones_max = 1000
iteraciones = 0
while(error > tolerancia and iteraciones < iteraciones_max):
 iteraciones += 1
 for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 if j == 1: # No flujo
 h_new[i,j] = h[i,j] + K[i,j] * r * (h[i+1,j] + h[i-1,j] + h[i,j+1]
 if j == Ny: # No flujo
 h_new[i,j] = h[i,j] + K[i,j] * r * (h[i+1,j] + h[i-1,j] + h[i,j-1]
 else:
 h_new[i,j] = h[i,j] + K[i,j] * r * (h[i+1,j] + h[i-1,j] + h[i,j+1]

h_new[:,0] = h_new[:,1]
h_new[:,Ny+1] = h_new[:,Ny]

for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 Vx[i,j] = -K[i,j] * s * (h[i+1,j] - h[i-1,j])
 Vy[i,j] = -K[i,j] * s * (h[i,j+1] - h[i,j-1])

for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 if j == 1: # No flujo
 c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i+1,j] - 2*c[i,j] + c[i-1,j]
 + Dy[i,j] * r * (c[i,j+1] - c[i,j]) / ϕ \
 - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
 - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

 if j == Ny: # No flujo
 c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i+1,j] - 2*c[i,j] + c[i-1,j]
 + Dy[i,j] * r * (c[i,j-1] - c[i,j]) / ϕ \
 - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
 - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

 if i == Nx: # No flujo
 c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i-1,j] - c[i,j]) / ϕ \
 + Dy[i,j] * r * (c[i,j+1] - 2*c[i,j] + c[i,j-1]
 - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
 + t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

 else:
 c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i+1,j] - 2*c[i,j] + c[i-1,j]
 + Dy[i,j] * r * (c[i,j+1] - 2*c[i,j] + c[i,j-1]) /
 - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
 - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ

c_new[:,0] = c_new[:,1]
c_new[:,Ny+1] = c_new[:,Ny]
c_new[Nx+1:] = c_new[Nx,:]

```

```

error = np.linalg.norm(h_new - h)
error_lista.append(error)
h[:] = h_new[:]
c[:] = c_new[:]

print(iteraciones, sep=' ', end= ' ')
END SOLUTION

```

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293
294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337
338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469
470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535
536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579
580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601
602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623
624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645
646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667
668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711
712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733
734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755
756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777
778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799
800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821
822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843
844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865
866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887
888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909
910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931

```



932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953  
 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975  
 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997  
 998 999 1000

```
ax1 = dict(aspect='equal', title='Malla')
ax2 = dict(aspect='equal', title='Carga hidráulica')
ax3 = dict(aspect='equal', title='Velocidad')
ax4 = dict(aspect='equal', title='Concentración')

vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
 dict(figsize=(8,8)))

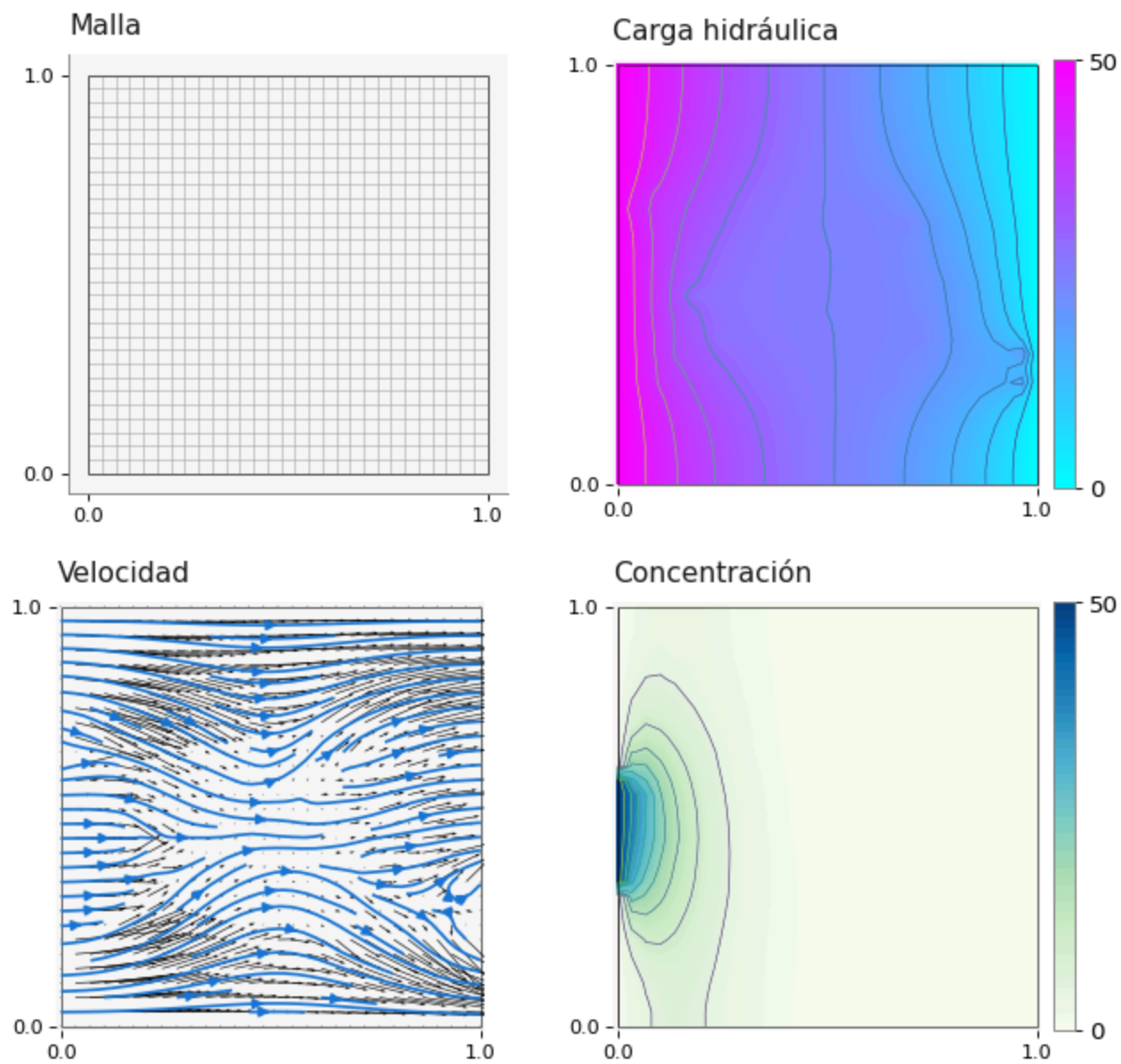
vis.plot_mesh2D(1, xg, yg)
vis.plot_frame(1, xg, yg)

cax3 = vis.set_canvas(2,Lx,Ly)
c_h = vis.contourf(2, xg, yg, h, levels=50, cmap='cool')
vis.fig.colorbar(c_h, cax=cax3, ticks = [h.min(), h.max()], shrink=0.5, orientati
vis.contour(2, xg, yg, h, levels=10, linewidths=0.5)
vis.plot_frame(2, xg, yg)

vis.set_canvas(3,Lx,Ly)
vis.plot_frame(3, xg, yg)
vis.streamplot(3, xg, yg, Vx, Vy)
vis.quiver(3, xg, yg, Vx, Vy, scale=0.5)

cax4 = vis.set_canvas(4,Lx,Ly)
c_c = vis.contourf(4, xg, yg, c, levels=50, cmap='GnBu')
vis.fig.colorbar(c_c, cax=cax4, ticks = [c.min(), c.max()], shrink=0.5, orientati
vis.contour(4, xg, yg, c, levels=10, linewidths=0.5)
vis.plot_frame(4, xg, yg)

plt.savefig('contaminante.pdf')
vis.show()
```





## 9 Conducción de Calor estacionaria en 2D.

**Objetivo General** - Resolver numérica y computacionalmente la ecuación de conducción de calor estacionaria en dos dimensiones usando un método implícito.

**Objetivos particulares** - Definir los parámetros físicos y numéricos. - Definir la malla del dominio. - Definir la temperatura inicial junto con sus condiciones de frontera y graficarla sobre la malla. - Definir el sistema lineal y resolverlo. - Graficar la solución.

[HeCompA - 02\\_cond\\_calor](#) by [Luis M. de la Cruz](#) is licensed under

[Attribution-ShareAlike 4.0 International](#)

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

### 9.1 Introducción.

**Jean-Baptiste Joseph Fourier** fue un matemático y físico francés que ejerció una fuerte influencia en la ciencia a través de su trabajo *Théorie analytique de la chaleur*. En este trabajo mostró que es posible analizar la conducción de calor en cuerpos sólidos en términos de series matemáticas infinitas, las cuales ahora llevan su nombre: *Series de Fourier*. Fourier comenzó su trabajo en 1807, en Grenoble, y lo completó en París en 1822. Su trabajo le permitió expresar la conducción de calor en objetos bidimensionales (hojas muy delgadas de algún material) en términos de una ecuación diferencial:

$$\frac{\partial T}{\partial t} = \kappa \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + S$$

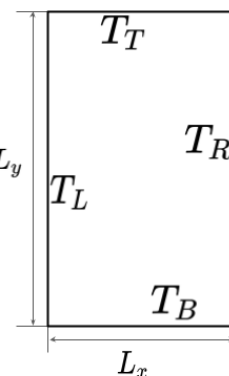
donde  $u$  representa la temperatura en un instante de tiempo  $t$  y en un punto  $(x, y)$  del plano Cartesiano,  $\kappa$  es la conductividad del material y  $S$  una fuente de calor.

### 9.2 Conducción estacionaria en 2D.

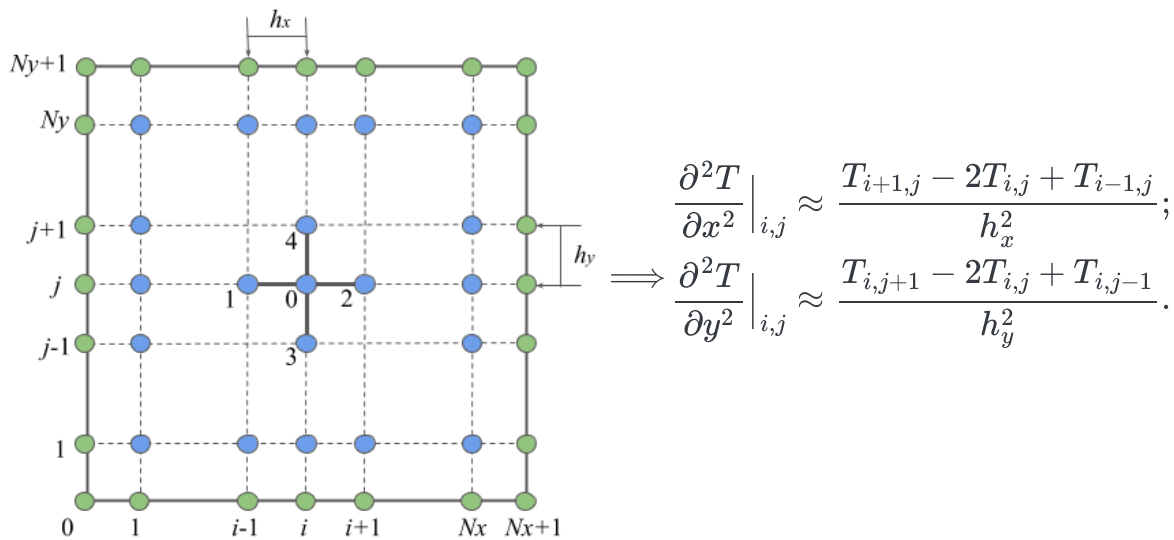
Cuando el problema es estacionario, es decir no hay cambios en el tiempo, y el dominio de estudio es una placa en dos dimensiones, como la que se muestra en la figura, podemos escribir el problema como sigue:

$$-\kappa \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) = S \quad (1)$$

Podemos aplicar condiciones de frontera son de tipo Dirichlet o Neumann en las paredes de la placa. En la figura se distingue  $T_L$ ,  $T_R$ ,  $T_T$  y  $T_B$  que corresponden a las temperaturas dadas en las paredes izquierda (LEFT), derecha (RIGHT), arriba (TOP) y abajo (BOTTOM), respectivamente.



A la ecuación (1) le podemos aplicar el método de diferencias finitas:



de tal manera que obtendríamos un sistema de ecuaciones lineales como el siguiente:

$$\begin{bmatrix}
 -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\
 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\
 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\
 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\
 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & \dots \\
 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & \dots \\
 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & \dots \\
 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & \dots \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & \dots \\
 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & \dots \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \dots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
 \end{bmatrix}
 \begin{bmatrix}
 u_{1,1} \\
 u_{2,1} \\
 u_{3,1} \\
 u_{4,1} \\
 u_{1,2} \\
 u_{2,2} \\
 u_{3,2} \\
 u_{4,2} \\
 u_{1,3} \\
 u_{2,3} \\
 u_{3,3} \\
 \vdots
 \end{bmatrix}
 =
 \begin{bmatrix}
 f_{1,1} \\
 f_{2,1} \\
 f_{3,1} \\
 f_{4,1} \\
 f_{1,2} \\
 f_{2,2} \\
 f_{3,2} \\
 f_{4,2} \\
 f_{1,3} \\
 f_{2,3} \\
 f_{3,3} \\
 \vdots
 \end{bmatrix}$$

En general un sistema de ecuaciones lineales puede contener  $n$  ecuaciones con  $n$  incógnitas y se ve como sigue:

$$A \cdot \mathbf{x} = \mathbf{b} \implies
 \begin{bmatrix}
 a_{00} & a_{01} & a_{02} & \dots & a_{0n} \\
 a_{10} & a_{11} & a_{12} & \dots & a_{1n} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n1} & a_{n2} & \dots & a_{nn}
 \end{bmatrix}
 \begin{bmatrix}
 x_0 \\
 x_1 \\
 \vdots \\
 x_n
 \end{bmatrix}
 =
 \begin{bmatrix}
 b_0 \\
 b_1 \\
 \vdots \\
 b_n
 \end{bmatrix}$$

El sistema se puede resolver usando diferentes tipos de métodos.

```

import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis

```

## 9.3 Parámetros físicos y numéricos

```

Tamaño del dominio
Lx = 1.0
Ly = 1.0
k = 1.0
Número de nodos en cada eje
Nx = 4

```

```

Ny = 4

Número total de nodos en cada eje incluyendo las fronteras
NxT = Nx + 2
NyT = Ny + 2

Número total de nodos
NT = NxT * NyT

Número total de incógnitas
N = Nx * Ny

Tamaño de la malla en cada dirección
hx = Lx / (Nx+1)
hy = Ly / (Ny+1)

Coordenadas de la malla
xn = np.linspace(0, Lx, NxT)
yn = np.linspace(0, Ly, NyT)

Generación de una rejilla
xg, yg = np.meshgrid(xn, yn, indexing='ij')

```

```

print('Total de nodos en x = {}, en y = {}'.format(NxT, NyT))
print('Total de incógnitas = {}'.format(N))
print('Coordenadas en x : {}'.format(xn))
print('Coordenadas en y : {}'.format(yn))
print('hx = {}, hy = {}'.format(hx, hy))

```

```

Total de nodos en x = 6, en y = 6
Total de incógnitas = 16
Coordenadas en x : [0. 0.2 0.4 0.6 0.8 1.]
Coordenadas en y : [0. 0.2 0.4 0.6 0.8 1.]
hx = 0.2, hy = 0.2

```

### 9.3.1 Graficación de la malla del dominio

```

from mpl_toolkits.axes_grid1 import make_axes_locatable
def set_axes(ax):
 """
 Configura la razón de aspecto, quita las marcas de los ejes y el marco.

 Parameters

 ax: axis
 Ejes que se van a configurar.
 """
 ax.set_aspect('equal')
 ax.set_xticks([])

```

```

ax.set_yticks([])
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)

def plot_mesh(ax, xg, yg):
 """
 Dibuja la malla del dominio.

 Paramters

 ax: axis
 Son los ejes donde se dibujará la malla.

 xn: np.array
 Coordenadas en x de la malla.

 yn: np.array
 Coordenadas en y de la malla.
 """
 set_axes(ax)

 xn = xg[:,0]
 yn = yg[0,:]

 for xi in xn:
 ax.vlines(xi, ymin=yn[0], ymax=yn[-1], lw=0.5, color='darkgray')

 for yi in yn:
 ax.hlines(yi, xmin=xn[0], xmax=xn[-1], lw=0.5, color='darkgray')

 ax.scatter(xg,yg, marker='.', color='darkgray')

def plot_frame(ax, xn, yn, lw = 0.5, color = 'k'):
 """
 Dibuja el recuadro de la malla.

 Paramters

 ax: axis
 Son los ejes donde se dibujará la malla.

 xn: np.array
 Coordenadas en x de la malla.

 yn: np.array
 Coordenadas en y de la malla.
 """
 set_axes(ax)

 # Dibujamos dos líneas verticales
 ax.vlines(xn[0], ymin=yn[0], ymax=yn[-1], lw = lw, color=color)

```

```

ax.vlines(xn[-1], ymin=yn[0], ymax=yn[-1], lw = lw, color=color)

Dibujamos dos líneas horizontales
ax.hlines(yn[0], xmin=xn[0], xmax=xn[-1], lw = lw, color=color)
ax.hlines(yn[-1], xmin=xn[0], xmax=xn[-1], lw = lw, color=color)

def set_canvas(ax, Lx, Ly):
 """
 Configura un lienzo para hacer las gráficas más estéticas.

 Parameters

 ax: axis
 Son los ejes que se van a configurar.

 Lx: float
 Tamaño del dominio en dirección x.

 Ly: float
 Tamaño del dominio en dirección y.

 Returns

 cax: axis
 Eje donde se dibuja el mapa de color.
 """
 set_axes(ax)

 lmax = max(Lx, Ly)
 offx = lmax * 0.01
 offy = lmax * 0.01
 ax.set_xlim(-offx, Lx+offx)
 ax.set_ylim(-offy, Ly+offy)
 ax.grid(False)

 ax.set_aspect('equal')
 divider = make_axes_locatable(ax)
 cax = divider.append_axes("right", "5%", pad="3%")
 cax.set_xticks([])
 cax.set_yticks([])
 cax.spines['bottom'].set_visible(False)
 cax.spines['left'].set_visible(False)

 return cax

```

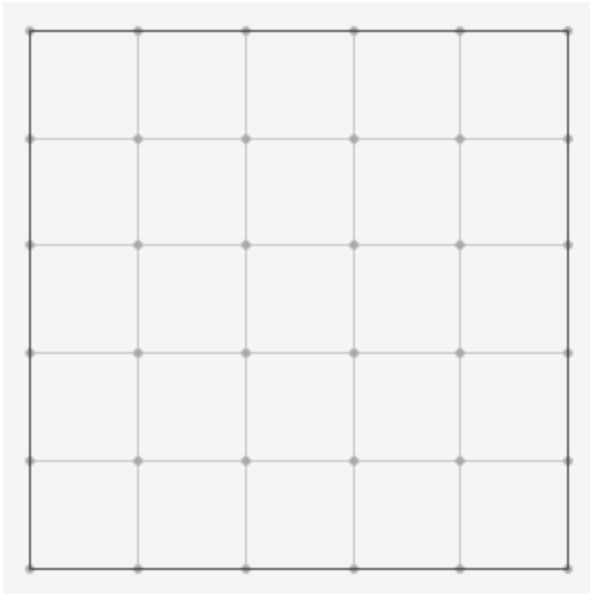
```

fig = plt.figure()
ax = plt.gca()

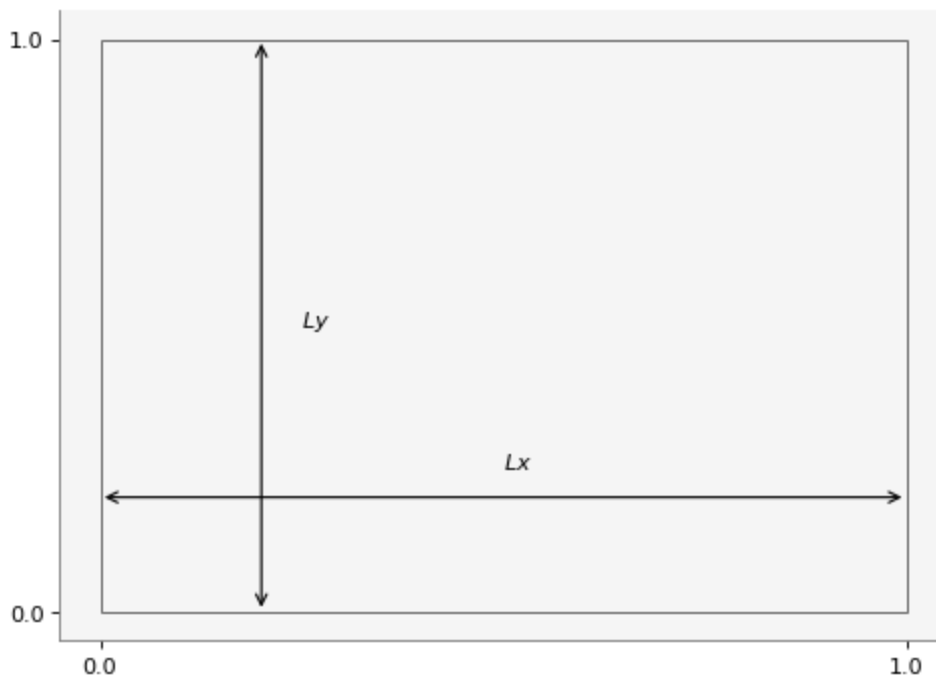
Ejecutamos la función plot_mesh(...)
plot_mesh(ax, xg, yg)

```

```
Dibujamos el recuadro con la función plot_fame(...)
plot_frame(ax, xn, yn)
```



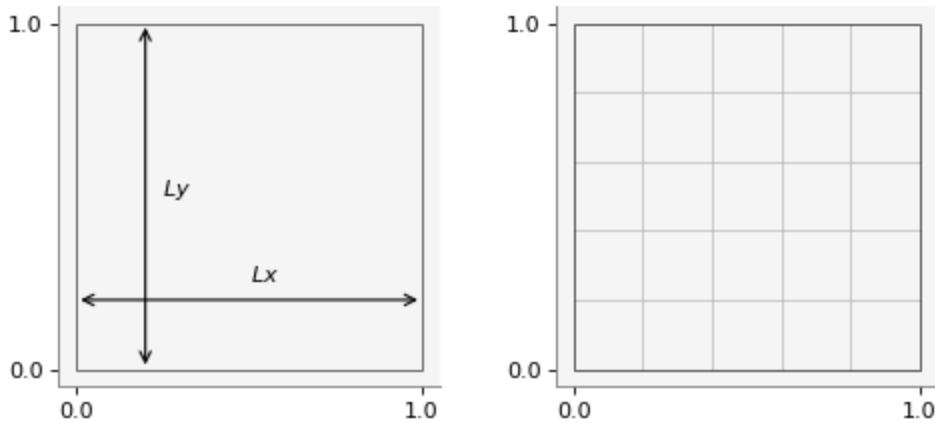
```
vis = mvis.Plotter(1,1)
vis.draw_domain(1, xg, yg)
```



```
vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')])
vis.draw_domain(1, xg, yg)
```



```
vis.plot_mesh2D(2, xg, yg)
vis.plot_frame(2, xg, yg)
```



## 9.4 Campo de temperaturas y sus condiciones de frontera

```
Definición de un campo escalar en cada punto de la malla
T = np.zeros((NxT, NyT))

Condiciones de frontera
TB = 1.0
TT = -1.0

T[0, :] = 0.0 # LEFT
T[-1, :] = 0.0 # RIGHT
T[:, 0] = TB # BOTTOM
T[:, -1] = TT # TOP

print('Campo escalar T ({}):\n {}'.format(T.shape, T))
```

```
Campo escalar T ((6, 6)):
[[1. 0. 0. 0. 0. -1.]
 [1. 0. 0. 0. 0. -1.]
 [1. 0. 0. 0. 0. -1.]
 [1. 0. 0. 0. 0. -1.]
 [1. 0. 0. 0. 0. -1.]
 [1. 0. 0. 0. 0. -1.]]
```

### 9.4.1 Graficación del campo escalar sobre la malla

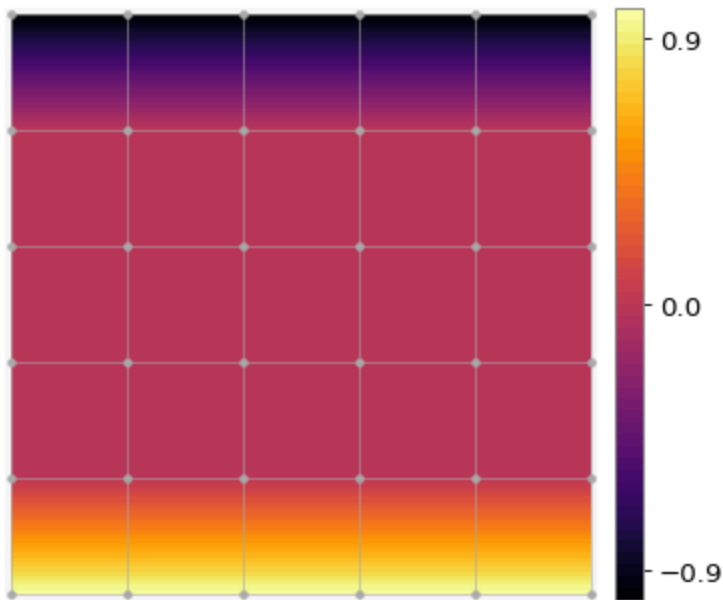
```
fig = plt.figure()
ax = plt.gca()
cax = set_canvas(ax, Lx, Ly)

c = ax.contourf(xg, yg, T, levels=50, cmap='inferno')
```

```

plot_mesh(ax, xg, yg)
fig.colorbar(c, cax=cax, ticks=[-0.9, 0.0, 0.9])
plt.show()

```

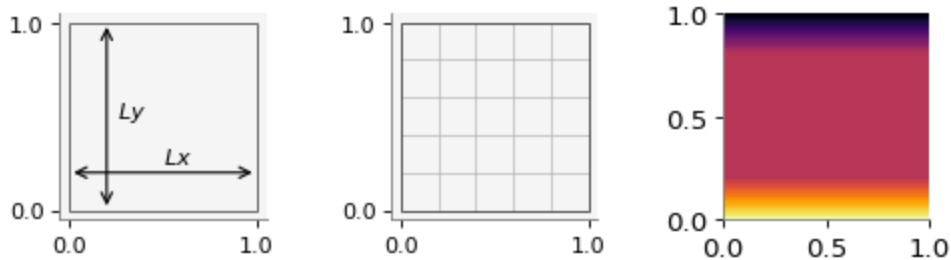


```

vis = mvis.Plotter(1,3,[dict(aspect='equal'), dict(aspect='equal'), dict(aspect='

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg)
vis.plot_frame(2, xg, yg)
vis.contourf(3, xg, yg, T, levels=50, cmap='inferno')
vis.show()

```

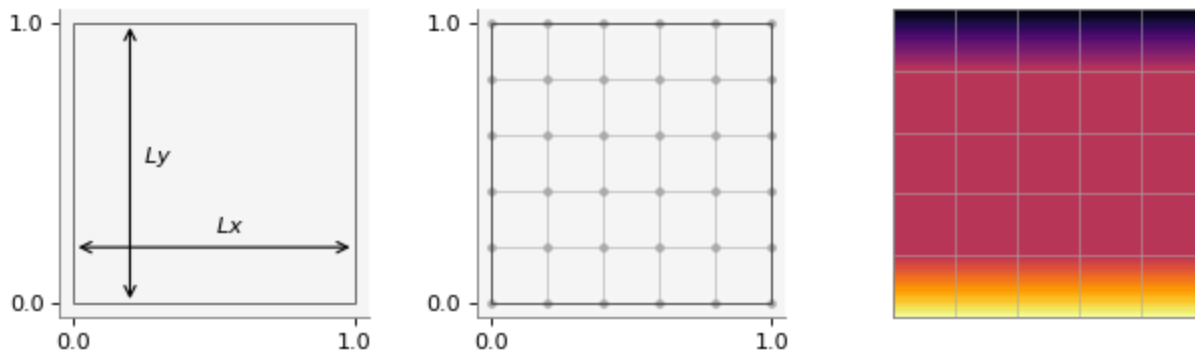


```

vis = mvis.Plotter(1,3,[dict(aspect='equal'), dict(aspect='equal'), dict(aspect='
dict(figsize=(8,16)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg, nodeson=True)
vis.plot_frame(2, xg, yg)
vis.contourf(3, xg, yg, T, levels=50, cmap='inferno')
vis.plot_mesh2D(3,xg, yg)
vis.show()

```



## 9.5 Flujo de calor

Fourier también estableció una ley para el flujo de calor que se escribe como:

$$\vec{q} = -\kappa \nabla u = -\kappa \left( \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \right)$$

```
def heat_flux(T, hx, hy):
 NxT, NyT = T.shape
 qx = np.zeros(T.shape)
 qy = qx.copy()

 for i in range(1, NxT-1):
 for j in range(1, NyT-1):
 qx[i,j] = -k * (T[i+1,j] - T[i-1,j]) / 2 * hx
 qy[i,j] = -k * (T[i,j+1] - T[i,j-1]) / 2 * hy
 return qx, qy
```

```
qx, qy = heat_flux(T, hx, hy)
```

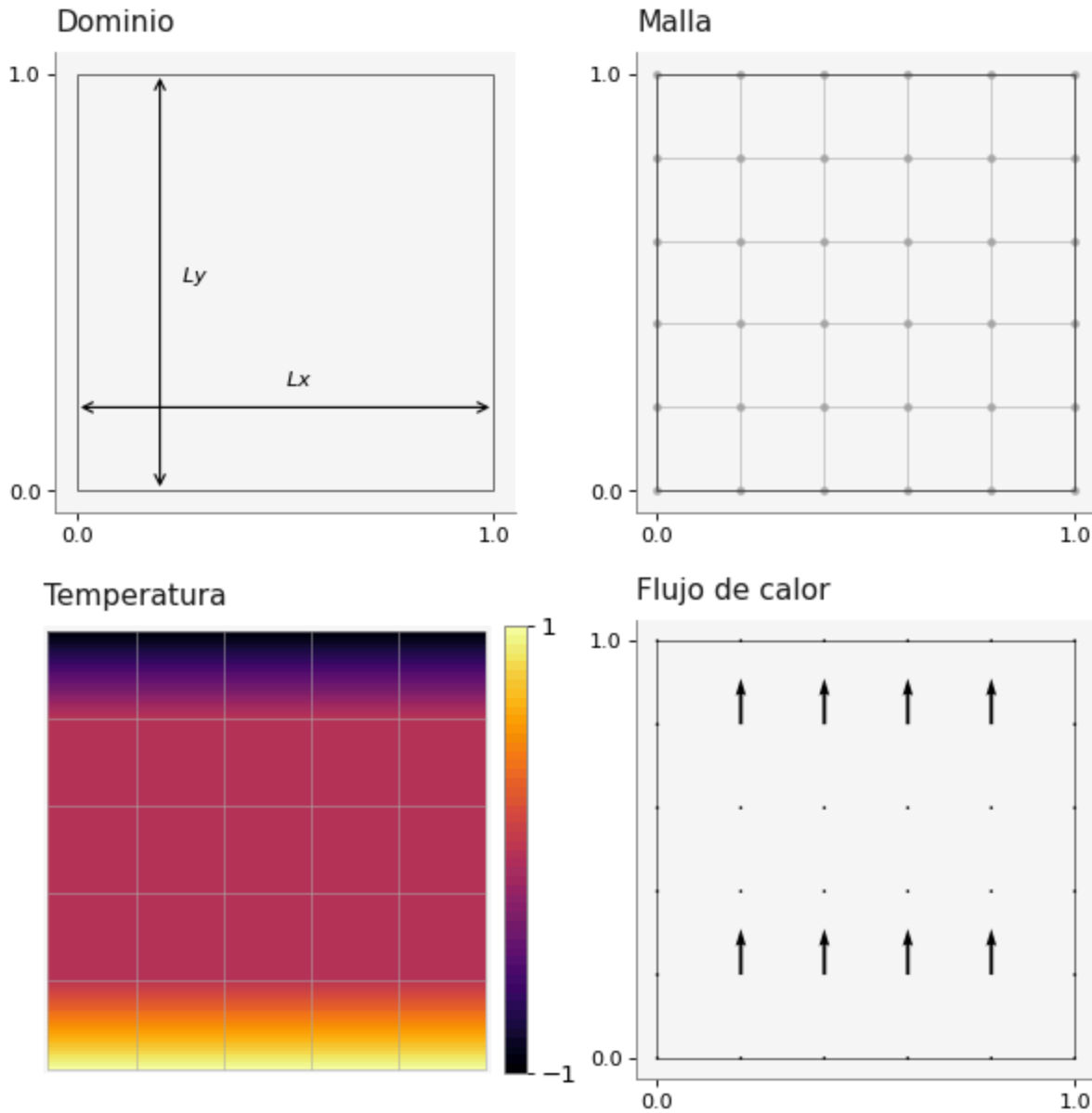
```
ax1 = dict(aspect='equal', title='Dominio')
ax2 = dict(aspect='equal', title='Malla')
ax3 = dict(aspect='equal', title='Temperatura')
ax4 = dict(aspect='equal', title='Flujo de calor')

vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
 dict(figsize=(8,8)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg, nodeson=True)
vis.plot_frame(2, xg, yg)

cax3 = vis.set_canvas(3,Lx,Ly)
c = vis.contourf(3, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c, cax=cax3, ticks = [T.min(), T.max()], shrink=0.5, orientatio
vis.plot_mesh2D(3, xg, yg)
```

```
vis.plot_frame(4, xg, yg)
vis.quiver(4, xg, yg, qx, qy, scale=1)
vis.show()
```



## 9.6 Sistema lineal

```
import FDM
La matriz del sistema. Usamos la función predefinida buildMatrix2D()
A = FDM.buildMatrix2D(Nx,Ny,-4)
A
```

```
array([[-4., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0.],
 [1., -4., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0.]])
```

```

 0., 0., 0.],
[0., 1., -4., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
 0., 0., 0.],
[0., 0., 1., -4., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
 0., 0., 0.],
[1., 0., 0., 0., -4., 1., 0., 0., 1., 0., 0., 0., 0.,
 0., 0., 0.],
[0., 1., 0., 0., 1., -4., 1., 0., 0., 1., 0., 0., 0.,
 0., 0., 0.],
[0., 0., 1., 0., 0., 1., -4., 1., 0., 0., 1., 0., 0.,
 0., 0., 0.],
[0., 0., 0., 1., 0., 0., 1., -4., 0., 0., 0., 1., 0.,
 0., 0., 0.],
[0., 0., 0., 0., 1., 0., 0., 0., -4., 1., 0., 0., 1.,
 0., 0., 0.],
[0., 0., 0., 0., 0., 1., 0., 0., 1., -4., 1., 0., 0.,
 1., 0., 0.],
[0., 0., 0., 0., 0., 0., 1., 0., 0., 1., -4., 1., 0.,
 0., 1., 0.],
[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., -4., 0.,
 0., 0., 1.],
[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., -4.,
 1., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1.,
 -4., 1., 0.],
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
 1., -4., 1.],
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
 0., 1., -4.]]))

```

```

RHS
b = np.zeros((Nx,Ny))
b[:, 0] -= TB # BOTTOM
b[:, -1] -= TT # TOP
b

```

```

array([[-1., 0., 0., 1.],
 [-1., 0., 0., 1.],
 [-1., 0., 0., 1.],
 [-1., 0., 0., 1.]])

```

## 9.7 Solución del sistema

Revisamos el formato del vector b

```
b.shape
```

(4, 4)

El vector debe ser de una sola dimensión:

```
b.flatten()
```

```
array([-1., 0., 0., 1., -1., 0., 0., 1., -1., 0., 0., 1., -1.,
 0., 0., 1.])
```

```
Calculamos la solución.
T_temp = np.linalg.solve(A, b.flatten())
T_temp
```

```
array([0.40909091, 0.11363636, -0.11363636, -0.40909091, 0.52272727,
 0.15909091, -0.15909091, -0.52272727, 0.52272727, 0.15909091,
 -0.15909091, -0.52272727, 0.40909091, 0.11363636, -0.11363636,
 -0.40909091])
```

```
T_temp.shape
```

(16,)

Colocamos la solución en el campo escalar T de manera adecuada

```
T[1:-1,1:-1] = T_temp.reshape(Nx,Ny)
T
```

```
array([[1., 0., 0., 0., 0.,
 -1.,],
 [1., 0.40909091, 0.11363636, -0.11363636, -0.40909091,
 -1.,],
 [1., 0.52272727, 0.15909091, -0.15909091, -0.52272727,
 -1.,],
 [1., 0.52272727, 0.15909091, -0.15909091, -0.52272727,
 -1.,],
 [1., 0.40909091, 0.11363636, -0.11363636, -0.40909091,
 -1.,],
 [1., 0., 0., 0., 0.,
 -1.,]])
```

```
qx, qy = heat_flux(T, hx, hy)
```

### 9.7.1 Gráfica de la solución

```

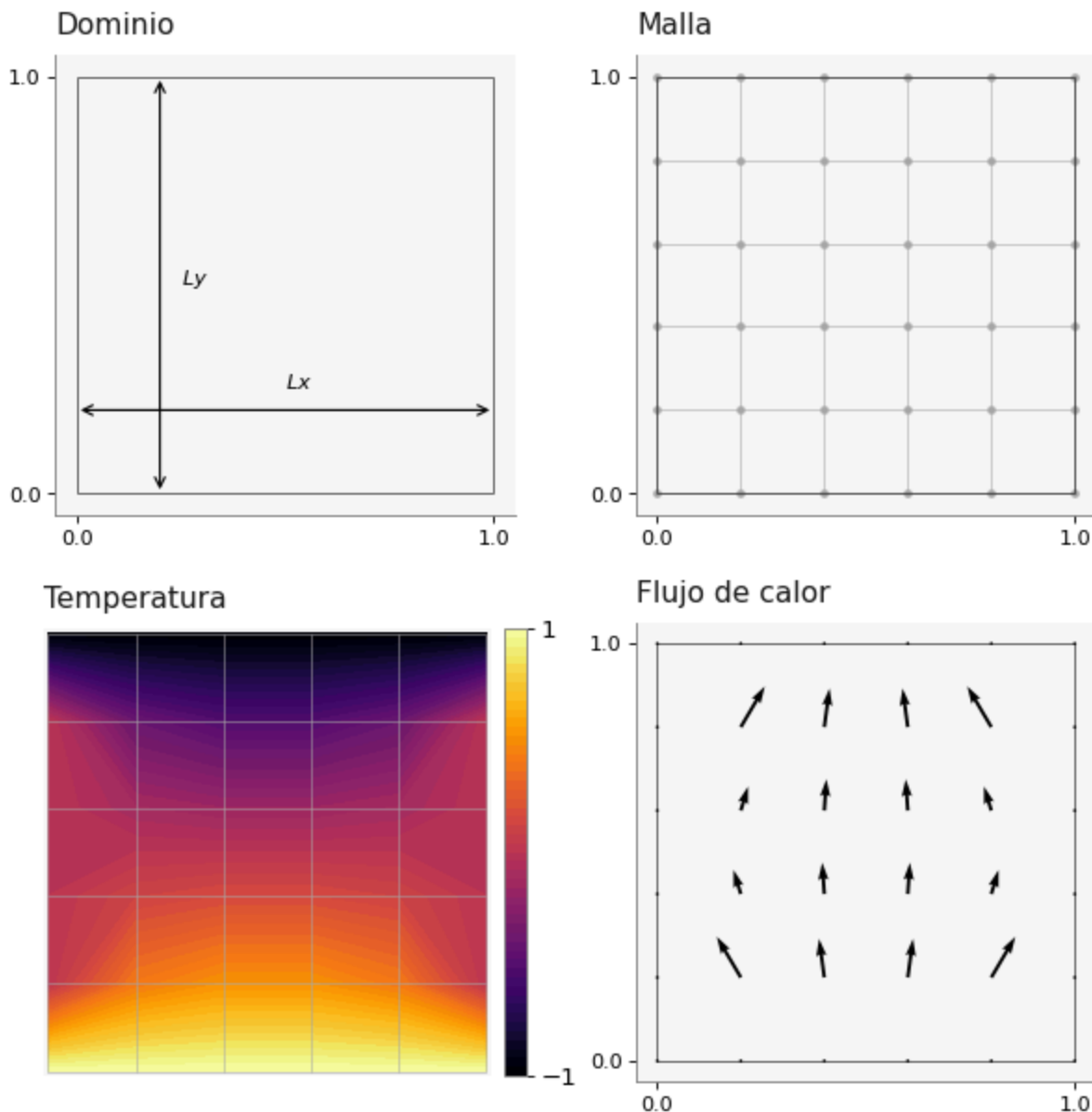
vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
 dict(figsize=(8,8)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg, nodeson=True)
vis.plot_frame(2, xg, yg)

cax3 = vis.set_canvas(3,Lx,Ly)
c = vis.contourf(3, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c, cax=cax3, ticks = [T.min(), T.max()], shrink=0.5, orientation='vertical')
vis.plot_mesh2D(3, xg, yg)

vis.plot_frame(4, xg, yg)
vis.quiver(4, xg, yg, qx, qy, scale=1)
vis.show()

```



## 9.7.2 Interactivo

```
def heat_cond(Lx, Ly, Nx, Ny):
 # Número total de nodos en cada eje incluyendo las fronteras
 NxT = Nx + 2
 NyT = Ny + 2

 # Número total de nodos
 NT = NxT * NyT

 # Número total de incógnitas
 N = Nx * Ny

 # Tamaño de la malla en cada dirección
 hx = Lx / (Nx+1)
 hy = Ly / (Ny+1)

 # Coordenadas de la malla
 xn = np.linspace(0, Lx, NxT)
 yn = np.linspace(0, Ly, NyT)

 # Generación de una rejilla
 xg, yg = np.meshgrid(xn, yn, indexing='ij')

 # Definición de un campo escalar en cada punto de la malla
 T = np.zeros((NxT, NyT))

 # Condiciones de frontera
 TB = 1.0
 TT = -1.0

 T[0, :] = 0.0 # LEFT
 T[-1, :] = 0.0 # RIGHT
 T[:, 0] = TB # BOTTOM
 T[:, -1] = TT # TOP

 # La matriz del sistema. Usamos la función predefinida buildMatrix2D()
 A = FDM.buildMatrix2D(Nx, Ny, -4)

 # RHS
 b = np.zeros((Nx, Ny))
 b[:, 0] -= TB # BOTTOM
 b[:, -1] -= TT # TOP

 # Calculamos la solución.
 T[1:-1, 1:-1] = np.linalg.solve(A, b.flatten()).reshape(Nx, Ny)

 # Calculamos el flujo de calor
 qx, qy = heat_flux(T, hx, hy)
```



```
ax1 = dict(aspect='equal', title='Dominio')
ax2 = dict(aspect='equal', title='Malla')
ax3 = dict(aspect='equal', title='Temperatura')
ax4 = dict(aspect='equal', title='Flujo de calor')

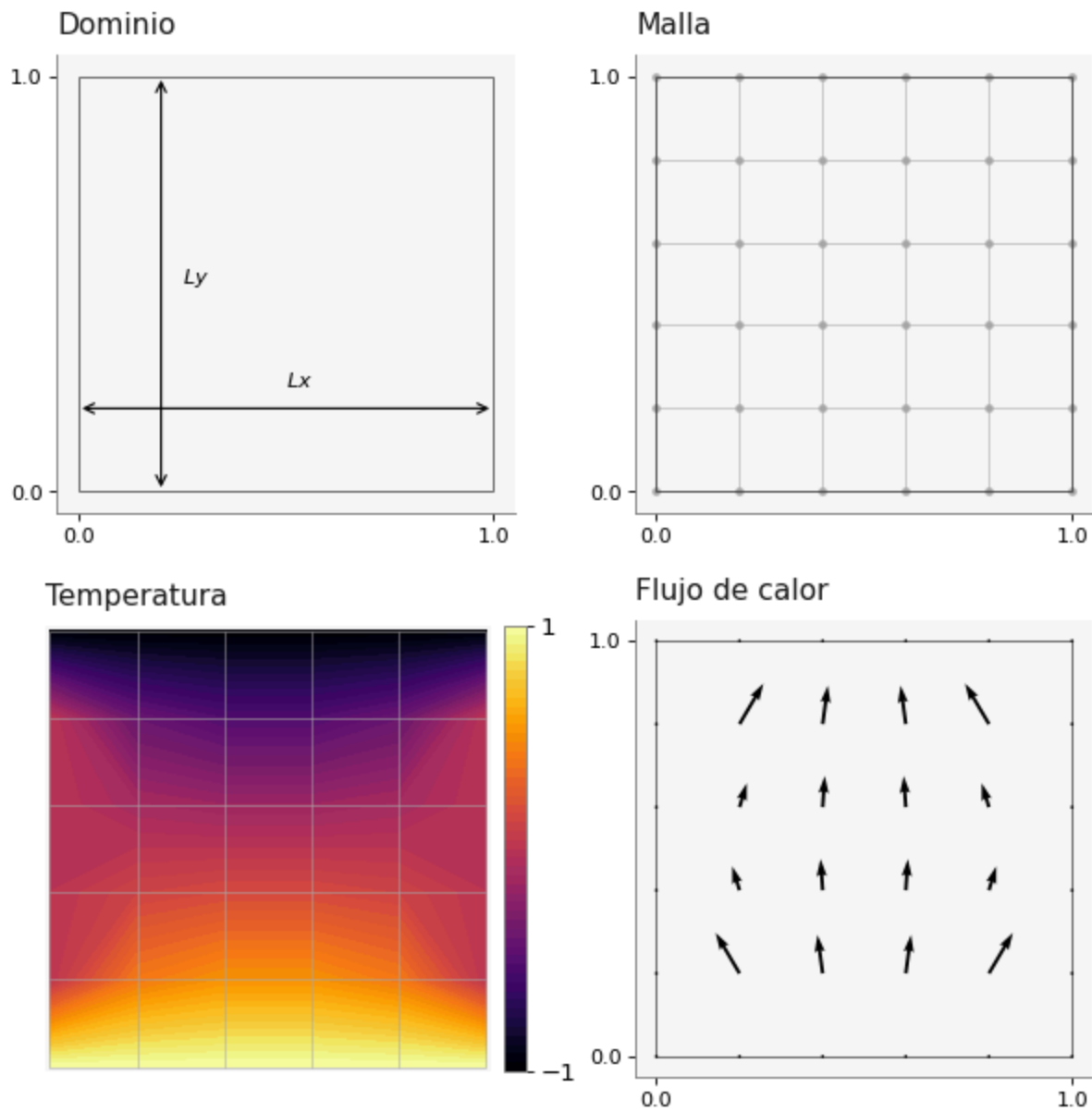
vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
 dict(figsize=(8,8)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg, nodeson=True)
vis.plot_frame(2, xg, yg)

cax3 = vis.set_canvas(3,Lx,Ly)
c = vis.contourf(3, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c, cax=cax3, ticks = [T.min(), T.max()], shrink=0.5, orienta
vis.plot_mesh2D(3, xg, yg)

vis.plot_frame(4, xg, yg)
vis.quiver(4, xg, yg, qx, qy, scale=1)
vis.show()
```

```
heat_cond(Lx=1, Ly=1, Nx=4, Ny=4)
```



```
import ipywidgets as widgets
```

```
widgets.interact(heat_cond, Lx = (1,3,1), Ly = (1,3,1), Nx = (4, 8, 1), Ny = (4,
```

```
<function __main__.heat_cond(Lx, Ly, Nx, Ny)>
```



# 11 Conducción de calor en 2D: algoritmos de solución de sistemas de ecuaciones.

## Objetivo.

Comparar mediante un interactivo varios métodos de solución de sistemas de ecuaciones lineales.

[MACTI-Analisis\\_Numerico\\_01](#) by [Luis M. de la Cruz](#) is licensed under [Attribution-ShareAlike 4.0 International](#)



## Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

Al ejecutar la siguiente celda obtendrás un interactivo en donde podrás seleccionar el método de solución y el tamaño de la malla ( $M \times N$ ) para resolver numéricamente, por diferencias finitas, un problema de conducción de calor.

Analiza con cuidado los valores más óptimos para encontrar una buena solución.

**NOTA.** Para ejecutar el interactivo debes hacer clic en el botón de play (a la derecha).

```
%run "./zHeatCondSolvers.py"
```





## 2 La derivada de una función y su aproximación

**Objetivo.** - Revisar el concepto de derivada usando herramientas visuales que permitan comprender su sentido geométrico y comprender lo que significa el cambio instantáneo.

[MACTI-Analysis\\_Numerico\\_01](#) by [Luis M. de la Cruz](#) is licensed under

[Attribution-ShareAlike 4.0 International](#)

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

```
Importamos todas las bibliotecas
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sympy as sy
import macti.visual
from macti.evaluation import *
```

```
quizz = Quizz('q2', 'notebooks', 'local')
```

### ## Introducción

Si revisamos con cuidado, algunas definiciones matemáticas utilizan un tipo de figura literaria conocida como [oxímoron](#). En términos simples, un oxímoron consiste en usar dos conceptos de significado opuesto y con ello generar un tercer concepto.

Por ejemplo: **La razón de cambio instantáneo.** - Cuando se habla de un *cambio*, se requiere de la comparación entre dos o más estados y con ello analizar las diferencias entre un estado y otro; - por otro lado, la palabra *instantáneo* tiene que ver con algo que dura un solo instante, es decir un tiempo puntal.

Entonces el concepto “**cambio instantáneo**” representa un oxímoron. Pero ¿cuál es su significado? ¿Será importante este concepto en nuestra vida diaria?

En lo que sigue veremos que la razón de cambio instantáneo tiene que ver con un concepto muy importante en Cálculo: **la derivada**.

### ## La curva del olvido.

Un estudiante de lenguas participará en un concurso internacional cuyo principal reto es el conocimiento del vocabulario de un cierto idioma. Por ello, es importante que el estudiante utilice un método de estudio adecuado para recordar el significado del mayor número de palabras posible.

La [curva del olvido](#) puede ayudar al estudiante a generar un plan de estudio adecuado. La función que define esta curva es la siguiente:

$$R(t) = e^{-t/S}$$

donde  $R$  es cuanto recordamos,  $S$  es la intensidad del recuerdo y  $t$  el tiempo. Podemos definir  $S \in (0, 1]$ , donde 1 es la máxima intensidad de recuerdo y un valor cercano a 0 corresponde a algo que no nos interesa nada.

**Observación:**  $S$  no puede ser exactamente 0 por que en ese caso la función  $R(t)$  no está definida.

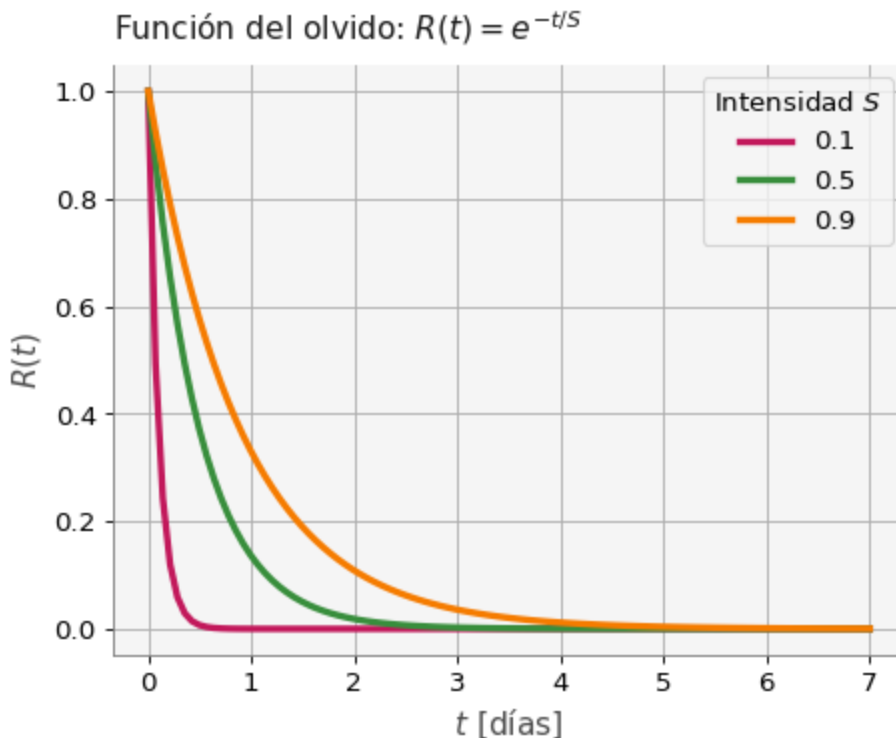
La siguiente gráfica muestra cómo decrecen nuestros recuerdos con el paso del tiempo.

```
Primero definimos la función del olvido
def R(t, S=0.9):
 return np.exp(-t/S)

Dominio de tiempo (hasta 7 días).
t = np.linspace(0,7,100)

Tres curvas del olvido para tres valores de S
plt.plot(t, R(t,0.1), lw=3, c='C3', label='{}'.format(0.1))
plt.plot(t, R(t,0.5), lw=3, c='C2', label='{}'.format(0.5))
plt.plot(t, R(t,0.9), lw=3, c='C1', label='{}'.format(0.9))

Configuración de la gráfica.
plt.title("Función del olvido: $R(t)=e^{-t/S}$ ")
plt.ylabel(" $R(t)$ ")
plt.xlabel(" t [días]")
plt.legend(title = 'Intensidad S ')
plt.grid()
plt.show()
```



### ¿Cuánto tiempo dura el recuerdo?

¿Será posible determinar cada cuanto tiempo el estudiante debe repasar las palabras para que no las olvide y pueda ganar el concurso? ¿De qué depende esto?

Tomemos por ejemplo el caso de  $S = 0.9$  (curva naranja). ¿En qué parte de la gráfica se incrementa el olvido? en otras palabras ¿en qué parte de la gráfica el descenso es más rápido?

Para conocer ese descenso, debemos calcular la pendiente  $m$  y eso lo podemos hacer con la siguiente fórmula:

$$m = \frac{R(t_2) - R(t_1)}{t_2 - t_1} \quad (1)$$

donde  $t_1$  y  $t_2$  son dos tiempos distintos.

Si definimos  $h = t_2 - t_1$  y  $t = t_1$  podemos escribir la fórmula (1) como sigue:

$$m(t) = \frac{R(t + h) - R(t)}{h} \quad (2)$$

En esta última fórmula vemos que la pendiente depende de  $t$ , es decir, en qué día nos encontramos.

Vamos a calcular  $R(t)$  y  $m(t)$  en  $t = [0, 1, 2, 3, 4, 5, 6, 7]$ , para  $h = 1$ :

```
h = 1.0
td = np.arange(0,8,h) # Definición de las t = 0,1,2,...,7
r = np.zeros(len(td)) # Arreglo para almacenar el valor de R
m = np.zeros(len(td)) # Arreglo para almacenar las pendientes

print('td = {}'.format(td))
print('r = {}'.format(r))
print('m = {}'.format(m))
```

```
td = [0. 1. 2. 3. 4. 5. 6. 7.]
r = [0. 0. 0. 0. 0. 0. 0. 0.]
m = [0. 0. 0. 0. 0. 0. 0. 0.]
```

```
Hacemos los cálculos en cada uno de los días
for i, t in enumerate(td):
 r[i] = R(t) # Función del olvido
 m[i] = (R(t + h) - R(t)) / h # Pendiente

Ponemos la información en un DataFrame y la mostramos
tabla = pd.DataFrame(np.array([td, r, m]).T,
 columns = ['t', '$R(t)$', '$m(t)$'])

tabla
```

	$t$	$R(t)$	$m(t)$
0	0.0	1.000000	-0.670807
1	1.0	0.329193	-0.220825

	$t$	$R(t)$	$m(t)$
2	2.0	0.108368	-0.072694
3	3.0	0.035674	-0.023930
4	4.0	0.011744	-0.007878
5	5.0	0.003866	-0.002593
6	6.0	0.001273	-0.000854
7	7.0	0.000419	-0.000281

Observa que la pendiente es negativa, lo cual indica un decrecimiento. También la magnitud de la pendiente (su valor absoluto) disminuye conforme  $t$  avanza. Vemos que el recuerdo disminuye mucho al principio, de tal manera que en el tercer día ya casi no se recuerda nada, alrededor del 3.5%. Esto se ve de manera gráfica como sigue:

```
Dominio de tiempo (hasta 7 días).
t = np.linspace(0,7,100)

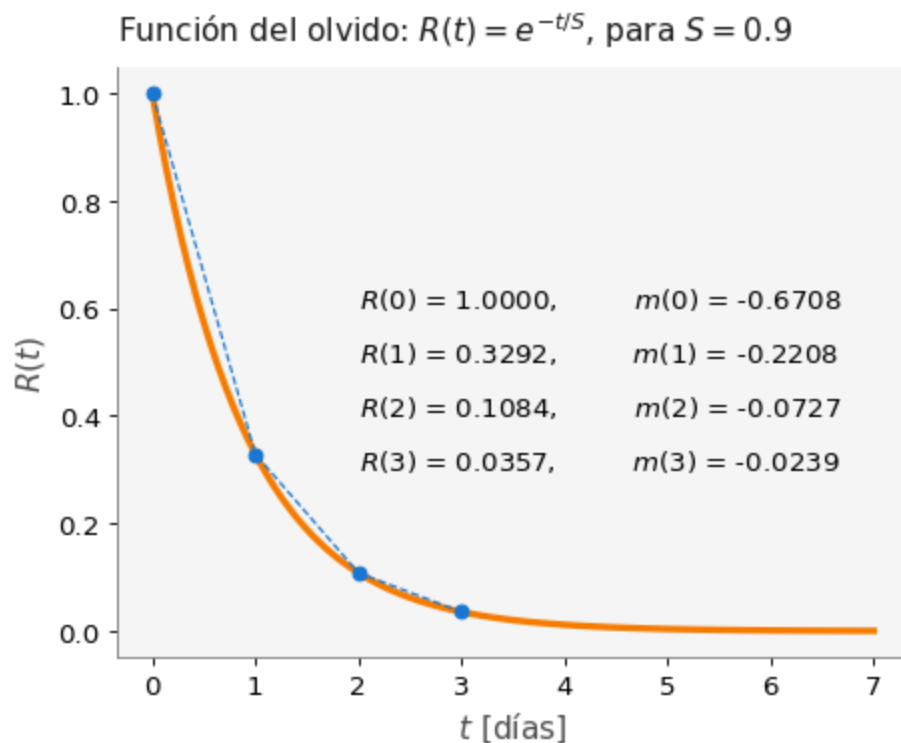
La curva del olvido para S = 0.9
plt.plot(t, R(t), lw=3, c='C1')

Línea punteada
plt.plot([0,1,2,3], [R(0), R(1), R(2), R(3)], 'o--', lw=1, zorder=5)

Configuración de la gráfica.
plt.title("Función del olvido: $R(t)=e^{-t/S}$, para $S = 0.9$ ")
plt.ylabel(" $R(t)$ ")
plt.xlabel(" t [días]")

Información de los primeros 3 días
plt.text(2,0.6,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \quad m(\{2:\}) = \{1:5.4f\}'.format(R(0), m[0]),
plt.text(2,0.5,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \quad m(\{2:\}) = \{1:5.4f\}'.format(R(1), m[1]),
plt.text(2,0.4,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \quad m(\{2:\}) = \{1:5.4f\}'.format(R(2), m[2]),
plt.text(2,0.3,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \quad m(\{2:\}) = \{1:5.4f\}'.format(R(3), m[3]),
plt.show()$$$$
```





En la gráfica anterior, la línea punteada nos muestra gráficamente el cambio en la pendiente de la recta que une los puntos negros, los cuales indican los días. Lo que estamos observando es la razón de cambio de  $R(t)$  en intervalos de tiempo de longitud  $h = 1$ . **Esto es justamente lo que expresa la fórmula (2).**

Los valores de  $R$  para los diferentes días indican como es que vamos olvidando lo que estudiamos en el día 0. Para el día 1 ya solo recordamos el 32.92%, en el segundo día el recuerdo es del 10.84% y para el día 3 el recuerdo es mínimo, del 3.57%. Por lo tanto, es conveniente repasar lo aprendido en el día 0 de manera frecuente, para este caso con  $S = 0.9$ , sería conveniente repasar todos los días.

¿Para este ejemplo, cómo podemos calcular **la razón de cambio instantáneo**? La respuesta es: haciendo  $h$  muy pequeña, es decir  $h \rightarrow 0$ .

Para ello, esta razón debería calcularse en un solo instante de tiempo, lo cual implica que  $t_1 = t_2 \implies h = 0$ , y esto nos lleva a que la fórmula de  $m(t)$  no está bien definida (¡división por cero!).

Pero, ¿qué pasa si  $h$  se hace muy pequeña? es decir:

$$\lim_{h \rightarrow 0} \frac{R(t+h) - R(t)}{h} = ? \quad (3)$$

¿Cuánto vale este límite? ¿Es posible calcularlo en cualquier caso y para cualquier tipo de función?

### 2.0.1 Ejemplo 1. ¿Qué pasa cuando $h \rightarrow 0$ para diferentes valores de $S$ ?

Ejecuta la siguiente celda de código para generar el interactivo en donde podrás modificar  $S$ ,  $h$  y  $t$ . Explora qué sucede para cada valor de los parámetros y posteriormente responde las preguntas del [Ejercicio 1](#).

Para ver los valores de  $R(t)$ ,  $R'(t)$  y  $m(t)$  haz clic sobre el botón **Muestra valores** sobre el interactivo

```
%run "./zinteractivo1.ipynb"
```

```
<function __main__.razonDeCambio(S, h, i0, anot)>
```

### Comentarios.

En la gráfica de la izquierda observamos que conforme  $h$  se hace más pequeño, observamos que la línea roja se aproxima cada vez mejor a la línea tangente (verde) que pasa por el punto rojo. La línea roja representa una aproximación a la razón de cambio instantánea en el punto rojo.

En la gráfica de la derecha observamos la gráfica de  $R'(t)$  (curva verde), un punto morado que representa el valor exacto de  $R'(t)$  y un punto negro que es la aproximación para una  $h$  dada.

Entonces, la tangente en el punto rojo, no es otra cosa que **la razón de cambio instantánea**. Veremos enseguida que ambas cosas representan un concepto conocido como **la derivada de la función** en el punto rojo.

## 2.0.2 Ejercicio 1.

El valor absoluto de la diferencia entre un valor exacto ( $v_e$ ) y un valor aproximado ( $v_a$ ) se conoce como el **error absoluto** y se escribe como sigue:

$$E_a = |v_e - v_a| \quad (4)$$

Usando esta definición, responde las siguientes preguntas.

1. ¿Cuál es la diferencia entre  $R'(1)$  y  $m(1)$  redondeada a 4 decimales para  $S = 0.9$ ,  $t = 1.0$  y  $h = 1.0$ ? Completa el código de la celda siguiente para obtener la respuesta.

**NOTA.** Para responder las preguntas, tienes que mover los parámetros en el interactivo a los valores correspondientes de  $S$ ,  $h$  y  $t$ ; posteriormente realiza los cálculos necesarios para obtener la respuesta correcta. No olvides revisar las reglas de redondeo.

**Hint:** calcula  $|R'(1) - m(1)|$ , usando las funciones `abs()` para calcular el valor absoluto y `round()` para redondear un número hasta un cierto número de dígitos. Puedes usar `help(abs)` y `help(round)` para obtener ayuda sobre el uso de estas funciones.

```
Define ve, va y Ea_1:
ve = ... # valor exacto R'(1)
va = ... # valor aproximado m(1) con h = 1.0
Ea_1 = ... # Error absoluto
BEGIN SOLUTION
ve = -0.36576999
va = -0.22082496
Ea_1 = round(abs(round(ve,4) - round(va,4)), 4)

file_answer = FileAnswer()
file_answer.write('1', Ea_1)
END SOLUTION

print(Ea_1)
```

0.145

```
quizz.eval_numeric('1', Ea_1)
```

-----

1 | Tu resultado es correcto.

-----

2. Cuál es la diferencia entre  $R'(1)$  y  $m(1)$  redondeada a 4 decimales para  $S = 0.9$ ,  $t = 1.0$  y  $h = 0.1$

```
Define ve, va y Ea_2:
ve = ... # valor exacto R'(1)
va = ... # valor aproximado m(1) con h = 0.1
Ea_2 = ... # Error absoluto
BEGIN SOLUTION
ve = -0.36576999
va = -0.34618159
Ea_2 = round(abs(round(ve,4) - round(va,4)), 4)

file_answer.write('2', Ea_2)
END SOLUTION

print(Ea_2)
```

0.0196

```
quizz.eval_numeric('2', Ea_2)
```

-----

2 | Tu resultado es correcto.

3. ¿Qué sucede con la diferencia entre  $R'(t)$  y  $m(t)$ , cuando  $h$  se hace más pequeño ( $h \rightarrow 0$ ), sin importar el valor de  $t$  ni de  $S$ ?

1. Se hace más grande.
2. Se mantiene constante.
3. Se hace más pequeña.
4. No es posible determinarlo.

```
Escribe tu respuesta como sigue
respuesta = ...

BEGIN SOLUTION
respuesta = 'c'

file_answer.write('3', respuesta)
END SOLUTION
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe  
Respuestas y retroalimentación almacenadas.

```
quizz.eval_option('3', respuesta)
```

3 | Tu respuesta: c, es correcta.

4. Escribe una función para calcular la fórmula (4) que reciba el valor exacto ( $v_e$ ), el valor aproximado ( $v_a$ ) y el número de decimales de la aproximación ( $p$ ) y que regrese el error absoluto entre  $v_e$  y  $v_a$ . Posteriormente pruebe la función con el valor de  $R'(t)$  (valor exacto) y el valor de  $m(t)$  (valor aproximado) con los siguientes parámetros:

1.  $S = 0.3, t = 0, h = 1, p = 6$
2.  $S = 0.3, t = 0, h = 0.1, p = 6$
3.  $S = 0.3, t = 6, h = 1.0, p = 10$
4.  $S = 0.3, t = 6, h = 0.1, p = 10$

```
Completa la función de error con la fórmula (4)
def error_absoluto(ve, va, p):
 #### BEGIN SOLUTION
 return round(abs(round(ve,p) - round(va,p)), p)
 #### END SOLUTION
```

```
Define ve, va y p con los valores del inciso A.
ve = ... # valor exacto R'(t)
va = ... # valor aproximado m(t)
p = ... # precisión

BEGIN SOLUTION
ve = -3.333333333
va = -0.9643260067
p = 6
EA = error_absoluto(ve, va, p) # S = 0.3, t = 0, h = 1

file_answer.write('4A', EA)
END SOLUTION

print(EA)
```

2.369007

```
quizz.eval_numeric('4A', EA)
```

-----  
4A | Tu resultado es correcto.  
-----

```
Define ve, va y p con los valores del inciso B.
ve = ... # valor exacto R'(t)
va = ... # valor aproximado m(t)
p = ... # precisión

BEGIN SOLUTION
ve = -3.333333333
va = -2.8346868943
p = 6
EB = error_absoluto(ve, va, p) # S = 0.3, t = 0, h = 0.1

file_answer.write('4B', EB)
END SOLUTION

print(EB)
```

0.498646

```
quizz.eval_numeric('4B', EB)
```

-----  
4B | Tu resultado es correcto.  
-----

```
Define ve, va y p con los valores del inciso C.
ve = ... # valor exacto R'(t)
va = ... # valor aproximado m(t)
p = ... # precisión

BEGIN SOLUTION
ve = -0.0000000069
va = -0.0000000020
p = 10
EC = error_absoluto(ve, va, p) # S = 0.3, t = 0, h = 0.1

file_answer.write('4C', EC)
END SOLUTION

print(EC)
```

4.9e-09

```
quizz.eval_numeric('4C', EC)
```

-----  
4C | Tu resultado es correcto.  
-----

```
Define ve, va y p con los valores del inciso D.
ve = ... # valor exacto R'(t)
va = ... # valor aproximado m(t)
p = ... # precisión
BEGIN SOLUTION
ve = -0.0000000069
va = -0.0000000058
p = 10
ED = error_absoluto(ve, va, p) # S = 0.3, t = 6, h = 0.1

file_answer.write('4D', ED)
END SOLUTION

print(ED)
```

1.1e-09

```
quizz.eval_numeric('4D', ED)
```

-----  
4D | Tu resultado es correcto.  
-----

## Definición de derivada

La fórmula (3) no es otra cosa que la definición formal de la derivada de una función. En casi todos los libros de cálculo encontrarás la siguiente notación para la derivada de la función  $f(x)$ :

$$\frac{df}{dx} = f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (5)$$

La derivada existe siempre y cuando exista este límite. ¿Puedes imaginar cuando este límite no existe?

Observe que en la definición anterior se está calculando la pendiente de la función  $f(x)$  en  $x$ . ¿Cuándo es que esta pendiente no se puede calcular?

## 2.0.3 Ejemplo 2. Aproximación de la derivada hacia adelante y hacia atrás.

Ejecuta la siguiente celda de código. Obtendrás un interactivo en donde podrás modificar  $h$  y  $x$ .

\* Explora los valores de  $f'$ ,  $m$  y del Error Absoluto cuando modificas  $x$  y  $h$ . \* Observa lo que sucede cuando activas el botón **Hacia atrás**. \* ¿El error absoluto es menor o mayor con el botón **Hacia atrás** activado? ¿De qué depende?

```
%run "./zinteractivo2.ipynb"
```

```
<function __main__.derivada(h, x0, back)>
```

Observamos en el interactivo anterior que también es posible calcular la derivada “hacia atrás” lo cual significa usar un punto a la izquierda del lugar donde se desea obtener la derivada (punto rojo). Esto se puede escribir analíticamente de la siguiente manera:

$$\frac{df}{dx} = f'(x) = \lim_{h \rightarrow 0} \frac{f(x) - f(x-h)}{h} \quad (6)$$

Entonces, las ecuaciones (5) y (6) indican dos maneras de calcular la derivada en un punto, pero que deben coincidir cuando  $h \rightarrow 0$ .

Se puede pensar en el límite por la derecha, ecuación (5) y el límite por la izquierda, ecuación (6). Ambos deben existir y deben ser iguales para que la derivada en un punto dado exista.

## ¿Cómo calcular la derivada analíticamente?

Consideremos la función  $f(x) = x^3$  y apliquemos la definición de derivada:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \rightarrow 0} \frac{(x+h)^3 - x^3}{h}$$

Si expandimos los términos del numerador obtenemos:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} (3x^2 + 3xh + h^2) \quad (7)$$

Al calcular el límite de la derecha obtenemos:

$$\frac{df}{dx} = 3x^2$$

Hemos calculado la derivada analítica de  $f(x) = x^3$ .

**¿Podrías calcular la derivada analíticamente usando la definición de la ecuación (6)?**

### 2.0.4 Ejercicio 3. Derivada analítica hacia atrás.

Escribe en la variable `respuesta` la fórmula que está entre paréntesis en la ecuación (7).

```
Escribe tu respuesta como sigue
respuesta = ...

BEGIN SOLUTION

x = sy.Symbol('x')
resultado = 3*x**2-3*x*h+h**2

file_answer.write('5', str(resultado))
file_answer.to_file('q2')
END SOLUTION
```

El directorio `:/home/jovyan/macti_notes/notebooks/.ans/Derivada/` ya existe  
Respuestas y retroalimentación almacenadas.

```
quizz.eval_expression('5', resultado)
```


-----  
5 | Tu respuesta:  
es correcta.  
-----

$$3x^2 - 3.0x + 1.0$$



# 12 Conducción de calor en sistemas terrestres.

**Objetivo.** Resolver numéricamente el flujo de calor de un sistema terrestre.

[MACTI NOTES](#) by Luis Miguel de la Cruz Salas is licensed under [CC BY-NC-SA 4.0](#) 

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

## Introducción

## Modelación Matemática y Computacional.

Cuatro modelos: 1. Modelo conceptual. 2. Modelo Matemático. 3. Modelo Numérico. 4. Modelo computacional.

## Modelo conceptual.

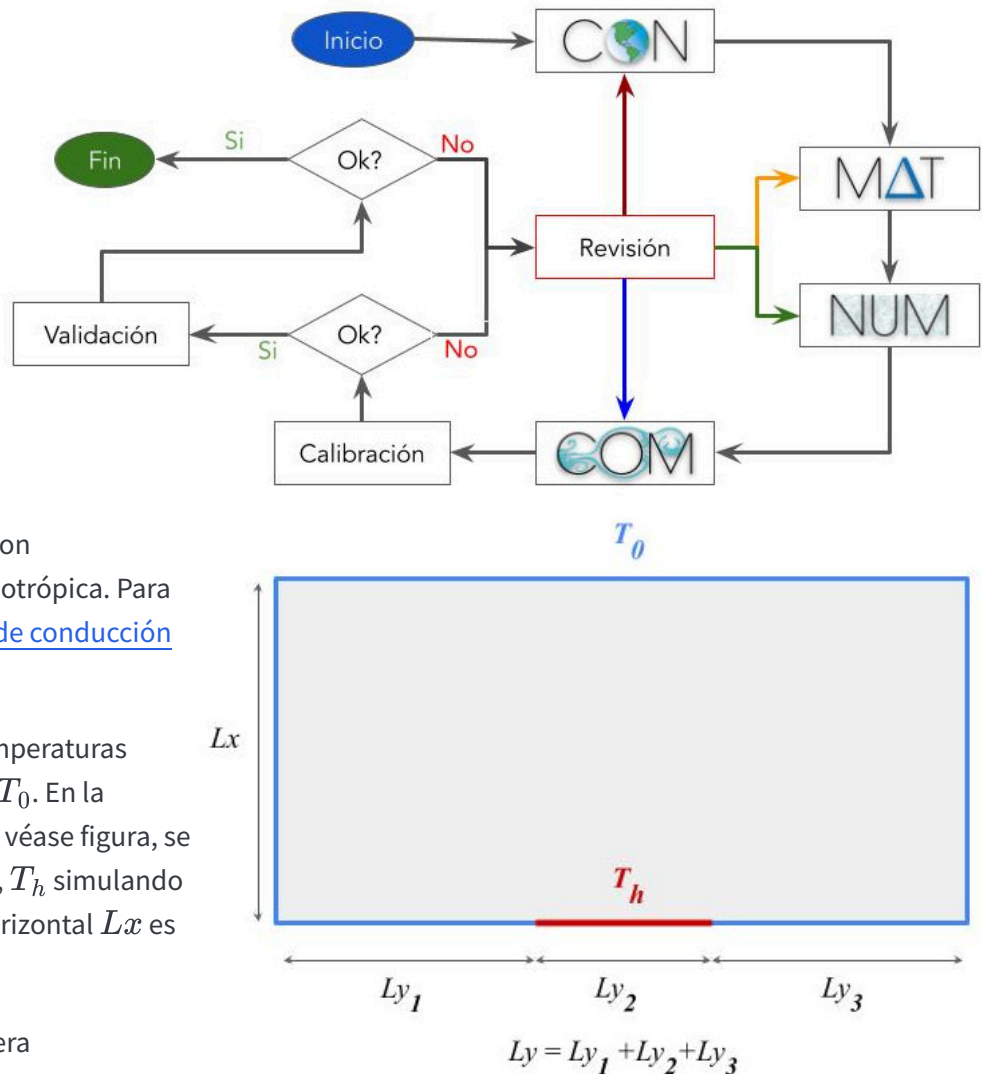
En este ejercicio vamos a aproximar la temperatura de una placa rectangular de metal con conductividad  $\kappa$  homogénea e isotrópica. Para ello, usaremos la [Ley de Fourier de conducción de calor](#).

Supondremos que se aplican temperaturas constantes en todas las paredes  $T_0$ . En la sección  $Ly_2$  de la pared inferior, véase figura, se aplica una temperatura más alta,  $T_h$  simulando un calentamiento. La longitud horizontal  $Lx$  es el doble que la horizontal  $Ly$ .

Trataremos el problema de manera adimensional de tal manera que tenemos los siguientes datos:  $\kappa = 1.0$ ,  $T_0 = 0$  y  $T_h = 20$ .

## Modelo matemático.

[Jean-Baptiste Joseph Fourier](#) fue un matemático y físico francés que ejerció una fuerte influencia en la ciencia a través de su trabajo *Théorie analytique de la chaleur*. En este trabajo mostró que es posible analizar la conducción de calor en cuerpos sólidos en términos de series matemáticas infinitas, las cuales ahora llevan su nombre: *Series de Fourier*. Fourier comenzó su trabajo en 1807, en Grenoble, y lo completó en París en 1822. Su trabajo le permitió



expresar la conducción de calor en objetos bidimensionales (hojas muy delgadas de algún material) en términos de la siguiente ecuación diferencial:

$$\frac{\partial T}{\partial t} = \kappa \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) \quad (1)$$

donde  $T$  representa la temperatura en un instante de tiempo  $t$  y en un punto  $(x, y)$  del plano Cartesiano y  $\kappa$  es la conductividad del material.

Para completar la ecuación (1) con condiciones iniciales y de frontera.

#### Condición inicial:


$$T(t = 0, x, y) = 0 \quad \text{para } (x, y) \in [0, Lx] \times [0, Ly]$$

#### Condiciones de frontera:

$$\begin{aligned} T(t, x = 0, y) &= T_0 & \text{para } y[0, Ly] \\ T(t, x = Lx, y) &= T_0 & \text{para } y[0, Ly] \\ T(t, x, y = 0) &= T_0 & \text{para } x \in Ly_1 \cup Ly_3 \\ T(t, x, y = 0) &= T_h & \text{para } x \in Ly_2 \\ T(t, x, y = Ly) &= T_0 & \text{para } y[0, Ly] \end{aligned}$$

para  $t = 0, T_{max}$

## Modelo numérico.

En ciertas condiciones existen soluciones analíticas de la ecuación (1), sin embargo, también es posible aproximar una solución usando el método numérico de diferencias finitas. 

El primer paso es discretizar el dominio como se ve en la figura.

El segundo paso es transformar la ecuación (1) en un conjunto de ecuaciones discretas, una para cada nodo donde se desea calcular la solución. Usando diferencias finitas y una fórmula explícita es posible escribir la forma discreta de (1) como sigue:

$$T_{i,j}^{n+1} = T_{i,j}^n + \frac{h_t \kappa}{h^2} (T_{i+1,j}^n + T_{i-1,j}^n + T_{i,j+1}^n + T_{i,j-1}^n - 4T_{i,j}^n) \quad (2)$$

donde: -  $T_{i,j}^n = T(t_n, x_i, y_j)$ , -  $T_{i+1,j}^n = T(t_n, x_{i+1}, y_j)$ , -  $T_{i-1,j}^n = T(t_n, x_{i-1}, y_j)$ , -  $T_{i,j+1}^n = T(t_n, x_i, y_{j+1})$ , -  $T_{i,j-1}^n = T(t_n, x_i, y_{j-1})$ . - El superíndice indica el instante de tiempo en el que se realiza el cálculo. - Se cumple que  $t_{n+1} = t_n + h_t$ , con  $h_t$  el paso de tiempo. - En este ejemplo  $h_x = h_y$ .

## Modelo computacional.

Usando la aproximación descrita en la sección anterior, vamos a realizar un ejemplo de conducción de calor. Para ello necesitamos conocer las herramientas de [Numpy](#) y [Matplotlib](#). Un tutorial de Numpy lo puedes ver [aquí](#) y uno de Matplotlib [por acá](#).

```
import sys
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis

print('Python', sys.version)
print(np.__name__, np.__version__)
print(plt.matplotlib.__name__, plt.matplotlib.__version__)
```

Python 3.11.5 | packaged by conda-forge | (main, Aug 27 2023, 03:34:09) [GCC 12.3.0]  
 numpy 1.25.2  
 matplotlib 3.7.2

### Algoritmo 1.

Los pasos a seguir son los siguientes.

### 1. Definir los parámetros físicos y numéricos del problema:

```
Parámetros físicos
κ = 1.0 # Conductividad
Lx = 2.0 # Longitud del dominio en dirección x
Ly = 1.0 # Longitud del dominio en dirección y
T0 = 0
Th = 20

Parámetros numéricos
Nx = 29 # Número de incógnitas en dirección x
Ny = 14 # Número de incógnitas en dirección y
h = Lx / (Nx+1) # Espaciamento entre los puntos de la rejilla
ht = 0.0001 # Paso de tiempo
N = (Nx + 2)* (Ny + 2) # Número total de puntos en la rejilla
```

```
print('Parámetros físicos' + '\n' + 40*'-')
print('Conductividad κ = {}'.format(κ))
print('Longitud en x = {} | Longitud en y = {}'.format(Lx, Ly))
print('T0 = {} | Th = {}'.format(T0, Th))
print('\nParámetros numéricos' + '\n' + 40*'-')
print('Nodos en x = {} | Nodos en y = {}'.format(Nx+2, Ny+2))
print('h = {} | ht = {}'.format(h, ht))
```

Parámetros físicos

```

Conductividad κ = 1.0
Longitud en x = 2.0 | Longitud en y = 1.0
T0 = 0 | Th = 20
```

Parámetros numéricos

Nodos en  $x = 31$  | Nodos en  $y = 16$

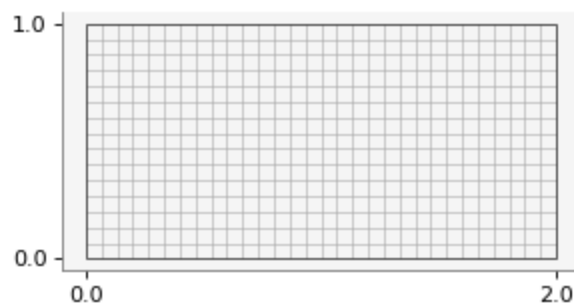
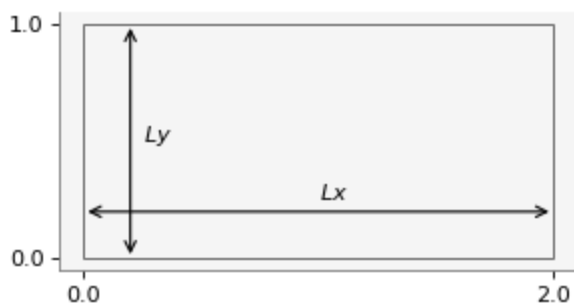
$h = 0.0666666666666667$  |  $ht = 0.0001$

## 2. Definir la rejilla donde se hará el cálculo (malla):

```
x = np.linspace(0,Lx,Nx+2) # Arreglo con las coordenadas en x
y = np.linspace(0,Ly,Ny+2) # Arreglo con las coordenadas en y
xg, yg = np.meshgrid(x,y,indexing='ij', sparse=False) # Creamos la rejilla para u
```

```
vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize=(10,10)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(2, xg, yg)
vis.plot_frame(2, xg, yg)
```



## 3. Definir las condiciones iniciales y de frontera:

```
T = np.zeros((Nx+2, Ny+2))
T[0,:] = 0 # Pared izquierda
T[Nx+1,:] = 0 # Pared derecha
T[:,0] = 0 # Pared inferior
T[:,Ny+1] = 0 # Pared superior
T[int(Nx*0.375):int(Nx*0.625),0] = 20 # Pedazo de la pared inferior en calentamiento
print(T)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [20. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [20. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [20. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [20. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

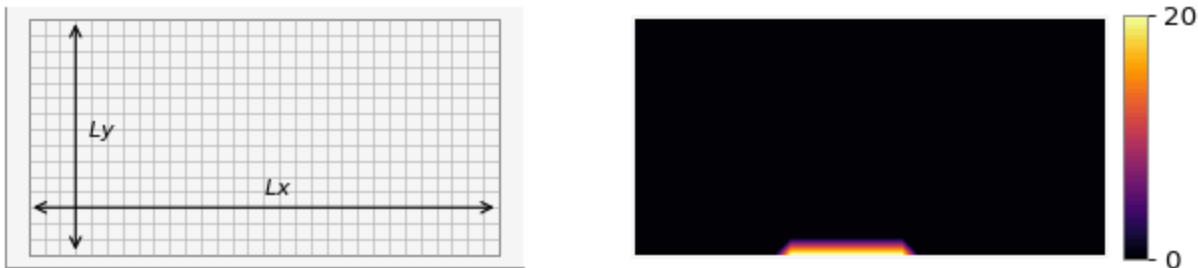
[illegible]

```
vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize=(10,10)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(1, xg, yg)

cax2 = vis.set_canvas(2,Lx,Ly)
c_T = vis.contourf(2, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c_T, cax=cax2, ticks = [T.min(), T.max()], shrink=0.5, orientation='vertical')

vis.show()
```



#### 4. Implementar el algoritmo de solución:

$$T_{i,j}^{n+1} = T_{i,j}^n + \frac{h_t \kappa}{h^2} (T_{i+1,j}^n + T_{i-1,j}^n + T_{i,j+1}^n + T_{i,j-1}^n - 4T_{i,j}^n)$$

```
ht = 0.001
r = κ * ht / h**2
T_new = T.copy()
tolerancia = 1.0e-4 #1.0e-3
error = 1.0
error_lista = []
iteracion = 1
```

```

while(error > tolerancia):
 for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 T_new[i,j] = T[i,j] + r * (T[i+1,j] + T[i-1,j] + T[i,j+1] + T[i,j-1])
 error = np.linalg.norm(T_new - T)
 error_lista.append(error)
 T[:] = T_new[:]
 print(iteracion, end = ' ')
 iteracion += 1

```

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293
294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337
338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469
470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535
536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579
580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601
602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623
624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645
646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667
668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711
712 713 714

```

```

vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize=

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(1, xg, yg)
vis.plot_frame(1, xg, yg)

cax2 = vis.set_canvas(2,Lx,Ly)

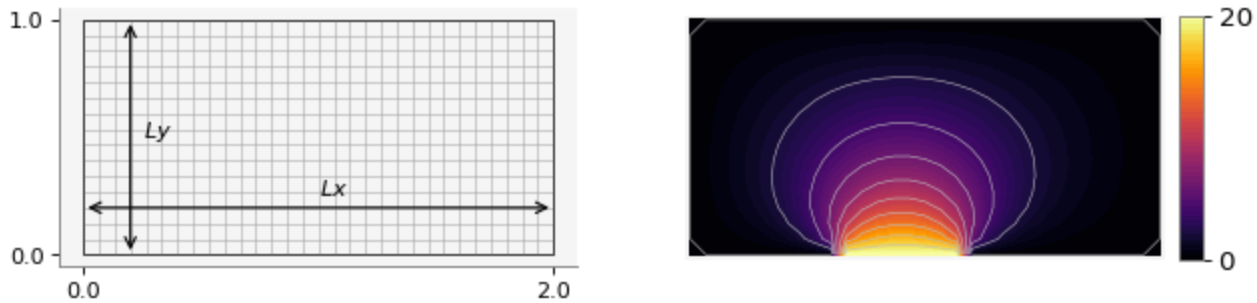
```

```

c_T = vis.contourf(2, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c_T, cax=cax2, ticks = [T.min(), T.max()], shrink=0.5, orientati
vis.contour(2, xg, yg, T, levels=10, colors='silver', linewidths=0.5)

vis.show()

```



Visualizamos la distribución de temperaturas y el *error* usando varias gráficas:

```

ax1 = dict(aspect='equal', title='Malla')
ax2 = dict(aspect='equal', title='Temperatura')
ax3 = dict(title='Error', yscale='log')

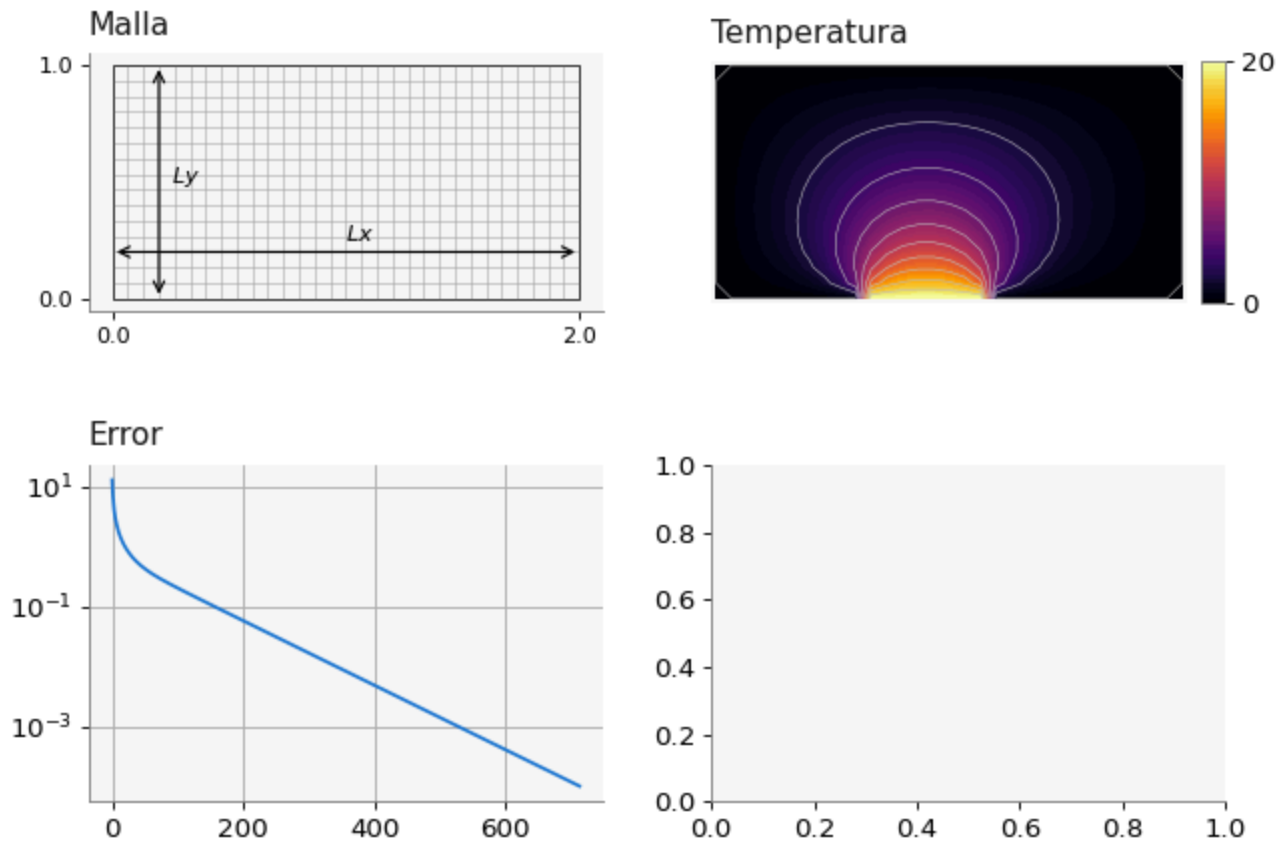
vis = mvis.Plotter(2,2,[ax1, ax2, ax3], dict(figsize=(8,6)))

vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(1, xg, yg)
vis.plot_frame(1, xg, yg)

cax2 = vis.set_canvas(2,Lx,Ly)
c_T = vis.contourf(2, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c_T, cax=cax2, ticks = [T.min(), T.max()], shrink=0.5, orientati
vis.contour(2, xg, yg, T, levels=10, colors='silver', linewidths=0.5)

vis.plot(3, [i for i in range(len(error_lista))], error_lista)
vis.grid(nlist=[3])
vis.show()

```



## ## Flujo de calor

Fourier también estableció una ley para el flujo de calor que se escribe como:

$$\vec{q} = -\kappa \nabla T = -\kappa \left( \frac{\partial T}{\partial x}, \frac{\partial T}{\partial y} \right)$$

Si usamos diferencias centradas para aproximar esta ecuación obtenemos:

$$\vec{q}_{i,j} = (qx_{i,j}, qy_{i,j}) = -\frac{\kappa}{2h} (T_{i+1,j} - T_{i-1,j}, T_{i,j+1} - T_{i,j-1})$$

La implementación de esta fórmula es directa y se muestra en la siguiente celda de código:

```
qx = np.zeros((Nx+2, Ny+2))
qy = qx.copy()

s = kappa / 2*h
for i in range(1,Nx+1):
 for j in range(1,Ny+1):
 qx[i,j] = -s * (T[i+1,j] - T[i-1,j])
 qy[i,j] = -s * (T[i,j+1] - T[i,j-1])
```

Visualización del flujo:



```
ax1 = dict(aspect='equal', title='Malla')
ax2 = dict(aspect='equal', title='Temperatura')
ax3 = dict(title='Error', yscale='log')
ax4 = dict(aspect='equal', title='Flujo de calor')

vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4], dict(figsize=(8,8)))

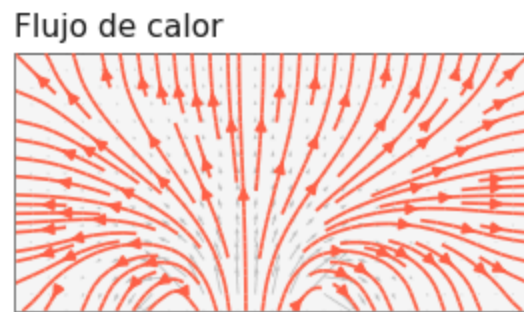
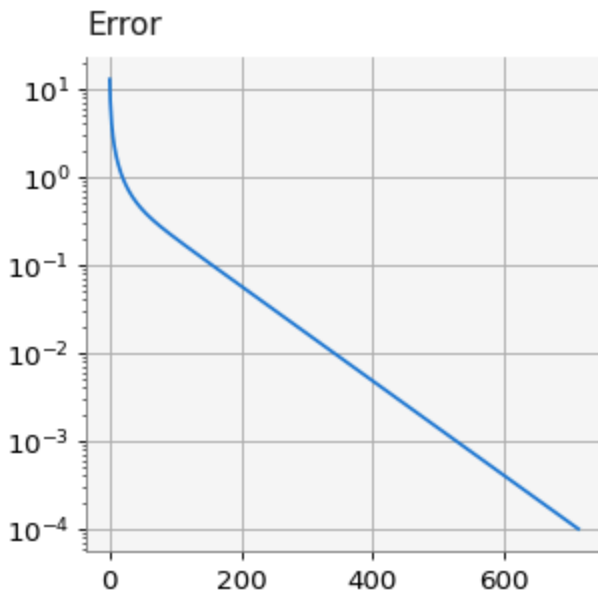
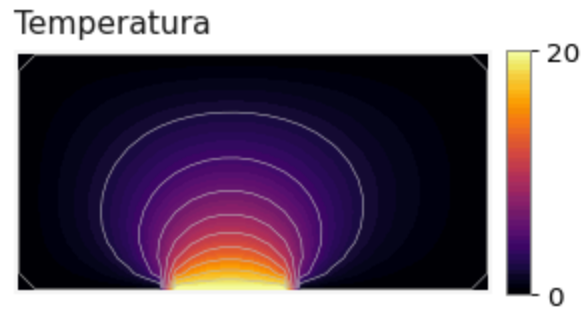
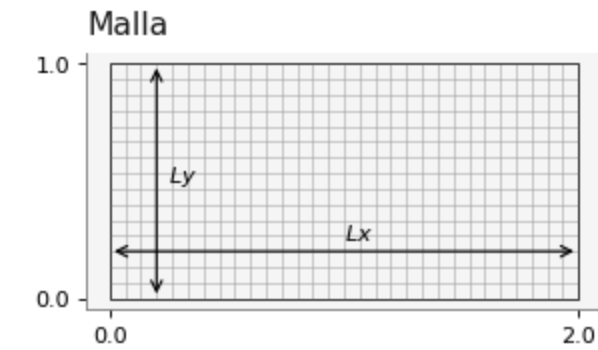
vis.draw_domain(1, xg, yg)
vis.plot_mesh2D(1, xg, yg)
vis.plot_frame(1, xg, yg)

cax2 = vis.set_canvas(2,Lx,Ly)
c_T = vis.contourf(2, xg, yg, T, levels=50, cmap='inferno')
vis.fig.colorbar(c_T, cax=cax2, ticks = [T.min(), T.max()], shrink=0.5, orientati
vis.contour(2, xg, yg, T, levels=10, colors='silver', linewidths=0.5)

vis.plot(3, [i for i in range(len(error_lista))], error_lista)
vis.grid(nlist=[3])

vis.plot_frame(4, xg, yg)
vis.streamplot(4, xg, yg, qx, qy, color = 'tomato')
vis.quiver(4, xg, yg, qx, qy, scale=4, color='silver')

vis.show()
```



### ## Seguimiento de partículas

Si soltamos una partícula en un flujo, dicha partícula seguirá la dirección del flujo y delinearé una trayectoria como se muestra en la siguiente figura. Para calcular los puntos de la trayectoria debemos resolver una ecuación como la siguiente:

$$\frac{\partial \vec{x}}{\partial t} = \vec{v} \quad \text{con} \quad \vec{x}(t=0) = \vec{x}_o$$

donde  $\vec{x} = (x, y)$   $\vec{x}$  representa la posición de la partícula y  $\vec{v} = (v_x, v_y)$  su velocidad. El método más sencillo para encontrar las posiciones de la partícula  $\vec{x}_i^{n+1}$ , en el instante  $n + 1$ , es conocido como de *Euler hacia adelante* y se escribe como:

$$\vec{x}_i^{n+1} = \vec{x}_i^n + h_t * \vec{v}_i^n$$

donde  $\vec{x}_i^n$  representa la posición de la partícula  $i$  en el instante  $n$ ,  $h_t$  es el paso de tiempo y  $\vec{v}_i^n$  es la velocidad de la partícula  $i$  en el instante  $n$ .

Escribimos la fórmula de *Euler hacia adelante* en componentes como sigue:

$$x_i^{n+1} = x_i^n + h_t * vx_i^n$$

$$y_i^{n+1} = y_i^n + h_t * vy_i^n$$

### Algoritmo 2. Para calcular la trayectoria de una partícula, dentro del flujo de calor, definimos el siguiente algoritmo.

### 1. Definimos un punto inicial:

```
xo = 0.75
yo = 0.25
print(xo)
print(yo)
```

0.75

0.25

### 2. Definimos los pasos de tiempo a calcular y los arreglos para almacenar las coordenadas de la trayectoria:

```
Pasos = 50
xp = np.zeros(Pasos)
yp = np.zeros(Pasos)
xp[0] = xo
yp[0] = yo
print(xp)
print(yp)
```

```
[0.75 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
[0.25 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
```

### 3. Interpolamos la velocidad en el punto donde está la partícula:

```
Interpolación de la velocidad
def interpolaVel(qx, qy, xpi, ypi, h):
 # localizamos la partícula dentro de la rejilla:
 li = int(xpi/h)
 lj = int(ypi/h)
 return (qx[li,lj], qy[li,lj])
```

### 4. Implementamos el método de Euler hacia adelante:

```
ht = 0.2
for n in range(1,Pasos):
```

```
vx, vy = interpolaVel(qx, qy, xp[n-1], yp[n-1], h)
xp[n] = xp[n-1] + ht * vx
yp[n] = yp[n-1] + ht * vy
```

```
print(xp)
print(yp)
```

```
[0.75 0.73267297 0.71623151 0.69979004 0.68334858 0.66690712
0.65517172 0.64257432 0.62997692 0.61737952 0.60478212 0.59218472
0.5802528 0.56832088 0.55638896 0.54708219 0.53777543 0.52846867
0.51988735 0.51130604 0.50272472 0.4941434 0.48556208 0.47698077
0.46839945 0.45981813 0.452232 0.44464587 0.43705974 0.4294736
0.42188747 0.41430134 0.40671521 0.39912908 0.39255738 0.38598568
0.37941398 0.37284229 0.36627059 0.35969889 0.35312719 0.34655549
0.34076944 0.33498338 0.32919732 0.3240257 0.31885409 0.31368247
0.30851085 0.30333924]
[0.25 0.2760231 0.29067962 0.30533614 0.31999265 0.33464917
0.34702799 0.35512954 0.36323109 0.37133264 0.37943419 0.38753574
0.39202761 0.39651948 0.40101135 0.40639468 0.41177802 0.41716136
0.4205113 0.42386125 0.4272112 0.43056115 0.43391109 0.43726104
0.44061099 0.44396093 0.44589069 0.44782044 0.4497502 0.45167995
0.45360971 0.45553946 0.45746922 0.45939897 0.46041839 0.46143782
0.46245724 0.46347666 0.46449609 0.46551551 0.46653494 0.46755436
0.46938765 0.47122094 0.47305423 0.47422718 0.47540013 0.47657308
0.47774603 0.47891898]
```

## 5. Graficamos el resultado.

```
ax1 = dict(aspect='equal', title='Temperatura y Flujo de calor')
ax2 = dict(aspect='equal', title='Trayectoria de una partícula')

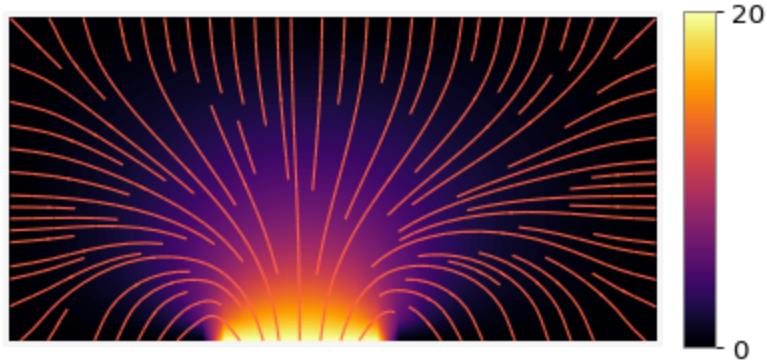
vis = mvis.Plotter(2,1,[ax1, ax2], dict(figsize=(8,6)))

cax1 = vis.set_canvas(1,Lx,Ly)
c_T = vis.contourf(1, xg, yg, T, levels=100, cmap='inferno')
vis.fig.colorbar(c_T, cax=cax1, ticks = [T.min(), T.max()], shrink=0.5, orientati
vis.streamplot(1, xg, yg, qx, qy, color = 'tomato', linewidth=1.0, arrowstyle='->')

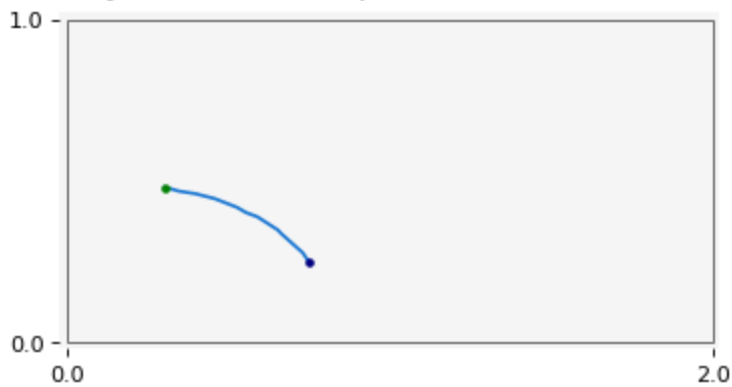
vis.plot_frame(2, xg, yg)
vis.set_canvas(2,Lx,Ly)
vis.plot(2, xp[0], yp[0], marker='.', color='navy', zorder=5)
vis.plot(2, xp, yp)
vis.plot(2, xp[-1], yp[-1], marker='.', color='g', zorder=5)

vis.show()
```

## Temperatura y Flujo de calor



## Trayectoria de una partícula



### Algoritmo 3. Dibuja varias trayectorias que inicien en sitios diferentes.

### 1. Definimos N posiciones aleatorias

```
from time import time
Transformación lineal
f = lambda x, a, b: (b-a)*x + a

Número de partículas
N = 50

Generación de partículas de manera aleatoria
np.random.seed(int(time()))
coord = np.random.rand(N,2)
coord[:,0] = f(coord[:,0], 0, Lx) # Transformación hacia el dominio de estudio
coord[:,1] = f(coord[:,1], 0, Ly) # Transformación hacia el dominio de estudio
```

```
coord
```

```
array([[1.80861083, 0.70284332],
 [0.19495729, 0.50500541],
 [0.29961371, 0.25697053],
 [0.05456327, 0.18328638],
 [1.70621773, 0.92613568],
 [0.68387895, 0.96914222],
```

```
[0.33004224, 0.76481451],
[1.14762037, 0.43546249],
[1.30818987, 0.93144918],
[0.76621098, 0.31536117],
[1.63299987, 0.56981932],
[0.05095166, 0.28699151],
[1.83355345, 0.71894018],
[0.75740415, 0.70825066],
[0.61753247, 0.97131374],
[1.14509942, 0.71906824],
[0.73257979, 0.8497072],
[1.56793562, 0.00572812],
[1.98044998, 0.37662571],
[0.04734427, 0.14621247],
[1.38006639, 0.60165316],
[0.19054804, 0.98702527],
[0.08633374, 0.53258502],
[1.47762183, 0.43874615],
[1.16836553, 0.99720001],
[1.11688811, 0.81774946],
[1.79934133, 0.72075091],
[1.67107078, 0.99296491],
[1.71089597, 0.07362231],
[0.92357449, 0.98310711],
[0.73241423, 0.30879331],
[1.14110394, 0.67605738],
[0.8252367 , 0.682654],
[0.7726115 , 0.43111325],
[1.63747706, 0.0103443],
[1.68862885, 0.46929049],
[0.65191397, 0.93185996],
[1.19352523, 0.85583507],
[1.64712482, 0.59986444],
[0.97560104, 0.49048682],
[1.93615182, 0.94317626],
[1.76533157, 0.75526127],
[1.14382069, 0.14581725],
[0.54453909, 0.89989101],
[1.8603346 , 0.05773824],
[1.57443738, 0.5371388],
[1.98317121, 0.37009536],
[1.79973185, 0.81160705],
[1.30302443, 0.27713148],
[0.30321612, 0.6030832]])
```

## 2. Definimos una función para el método de Euler hacia adelante.

```
def euler(x, v, h):
 return x + h * v
```

### 3. Definimos los arreglos para almacenar las posiciones de las trayectorias

```
Parámetros para el modelo numérico
Nt = 1000 # Número de pasos en el tiempo
ht = 0.2 # Tamaño del paso de tiempo

Arreglos para almacenar las N partículas en Nt pasos de tiempo
xn = np.zeros((N,Nt+1))
yn = np.zeros((N,Nt+1))

print('x : {}'.format(xn))
print('y : {}'.format(yn))
```

```
x : [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
y : [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

### 4. Inicializamos la primera posición de las trayectorias.

```
Inicialización
for i in range(0,N):
 xn[i, 0] = coord[i,0]
 yn[i, 0] = coord[i,1]

print('x : {}'.format(xn))
print('y : {}'.format(yn))
```

```
x : [[1.80861083 0. 0. ... 0. 0. 0.]
 [0.19495729 0. 0. ... 0. 0. 0.]
 [0.29961371 0. 0. ... 0. 0. 0.]
 ...
 [1.79973185 0. 0. ... 0. 0. 0.]
 [1.30302443 0. 0. ... 0. 0. 0.]
 [0.30321612 0. 0. ... 0. 0. 0.]]
y : [[0.70284332 0. 0. ... 0. 0. 0.]
 [0.50500541 0. 0. ... 0. 0. 0.]
 [0.25697053 0. 0. ... 0. 0. 0.]
 ...
 [0.81160705 0. 0. ... 0. 0. 0.]]
```

```
[0.27713148 0. 0. ... 0. 0. 0.]
[0.6030832 0. 0. ... 0. 0. 0.]]
```

### 5. Para cada posición inicial calculamos una trayectoria.

```
for n in range(1,Nt+1): # Ciclo en el tiempo.
 for i in range(0,N): # Ciclo para cada trayectoria.
 xi = xn[i,n-1]
 yi = yn[i,n-1]
 vx, vy = interpolaVel(qx, qy, xi, yi, h)
 xn[i,n] = euler(xi, vx, ht)
 yn[i,n] = euler(yi, vy, ht)

print('x : {}'.format(xn))
print('y : {}'.format(yn))
```

```
x : [[1.80861083 1.8104982 1.81238556 ... 2.00088212 2.00088212 2.00088212]
 [0.19495729 0.19068415 0.18641102 ... 0.06622123 0.06622123 0.06622123]
 [0.29961371 0.29408112 0.28854852 ... 0.06619201 0.06619201 0.06619201]
 ...
 [1.79973185 1.80101236 1.80220403 ... 2.00011212 2.00011212 2.00011212]
 [1.30302443 1.31817629 1.33332814 ... 1.66747644 1.66747644 1.66747644]
 [0.30321612 0.29930129 0.29538645 ... 0.06536666 0.06536666 0.06536666]]
y : [[0.70284332 0.70364223 0.70444114 ... 0.76646629 0.76646629 0.76646629]
 [0.50500541 0.50541443 0.50582344 ... 0.51408735 0.51408735 0.51408735]
 [0.25697053 0.25390652 0.25084252 ... 0.12645919 0.12645919 0.12645919]
 ...
 [0.81160705 0.81302338 0.81404905 ... 0.99836718 0.99836718 0.99836718]
 [0.27713148 0.27942968 0.28172788 ... 0.06210498 0.06210498 0.06210498]
 [0.6030832 0.6050664 0.60704961 ... 0.67477277 0.67477277 0.67477277]]
```

### 6. Graficamos el resultado final.

```
ax1 = dict(aspect='equal', title='Temperatura y Flujo de calor')
ax2 = dict(aspect='equal', title='Trayectoria de partículas')

vis = mvis.Plotter(2,1,[ax1, ax2], dict(figsize=(8,6)))

cax1 = vis.set_canvas(1,Lx,Ly)
c_T = vis.contourf(1, xg, yg, T, levels=100, cmap='inferno')
vis.fig.colorbar(c_T, cax=cax1, ticks = [T.min(), T.max()], shrink=0.5, orientati
vis.streamplot(1, xg, yg, qx, qy, color = 'tomato', linewidth=1.0, arrowstyle='->')

vis.plot_frame(2, xg, yg)

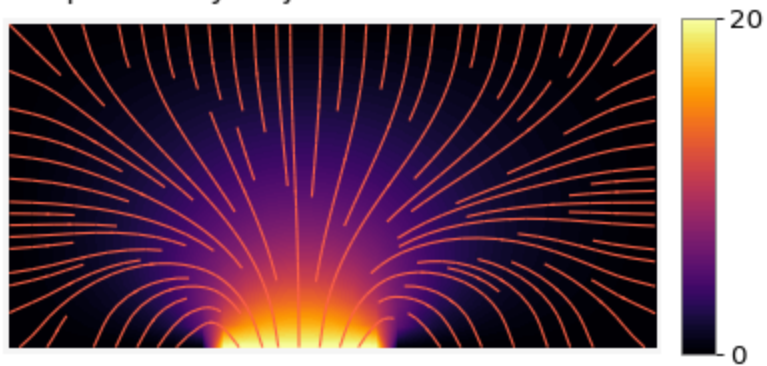
vis.set_canvas(2,Lx,Ly)
vis.quiver(2, xg, yg, qx, qy, scale=2, color='silver')

for i in range(0,N):
 vis.scatter(2, xn[i,0], yn[i,0], marker = '.', color='navy', alpha=0.75, s = 100)
 vis.plot(2, xn[i,:], yn[i,:], lw=1.0)
```

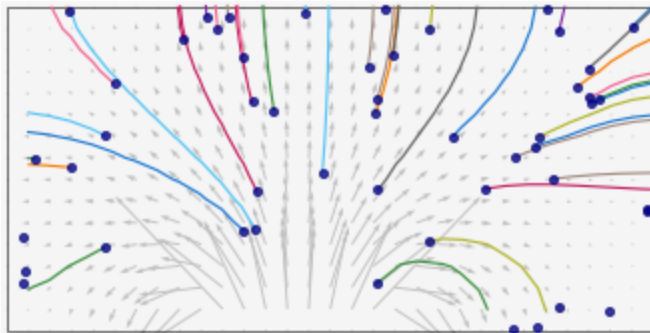


```
plt.savefig('flujo_calor.pdf')
vis.show()
```

### Temperatura y Flujo de calor



### Trayectoria de partículas





## 5 Derivadas numéricas: ecuación de calor 1D.

**Objetivo.** - Aplicar diferencias finitas centradas en la solución numérica de la transferencia de calor en 1D.

[MACTI-Analysis\\_Numerico\\_01](#) by [Luis M. de la Cruz](#) is licensed under

[Attribution-ShareAlike 4.0 International](#)

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import ipywidgets as widgets
import macti.visual as mvis
from macti.evaluation import *
```

```
quizz = Quizz('1', 'notebooks', 'local')
```

El modelo matemático para la conducción de calor en 1D con condiciones de frontera de tipo Dirichlet, con  $\kappa =$  constante se escribe como sigue:

$$\begin{aligned} -\kappa \frac{d^2 T}{dx^2} &= S \text{ para } x \in [0, L] \\ T(x=0) &= T_A \\ T(x=L) &= T_B \end{aligned}$$

La solución analítica de este modelo matemático se escribe como sigue:

$$T(x) = \left( \frac{T_B - T_A}{L} + \frac{S}{2\kappa}(L - x) \right) x + T_A \quad (1)$$

### 5.1 Ejercicio 1.

En la siguiente celda complete el código para implementar la fórmula (1). Posteriormente, define los siguientes valores para calcular la solución exacta:

```
x=np.linspace(0,1,10)
TA = 1.0
TB = 0.0
S = 1.0
L = 1.0
k = 1.0
```

```
Solucion exacta
def sol_exacta(x, TA, TB, S, L, k):
 """
 Calcula la temperatura usando la fórmula obtenida con Series de Taylor.

 Parameters

 x: np.array
 Coordenadas donde se calcula la temperatura.

 TA: float
 Es la condición de frontera a la izquierda.

 TB: float
 Es la condición de frontera a la derecha.

 S: float
 es la fuente.

 L: float
 L es la longitud del dominio.

 k: float
 es la conductividad del material.

 Return

 al final esta función dibuja la solución.
 """
 ### BEGIN SOLUTION
 return ((TB - TA)/L + S /(2*k) * (L - x)) * x + TA
 ### END SOLUTION
```

```
x=np.linspace(0,1,10)
TA = 1.0
TB = 0.0
S = 1.0
L = 1.0
k = 1.0

Cálculo de la solución exacta.
Te = ...
BEGIN SOLUTION
Te = sol_exacta(x, TA, TB, S, L, k)

file_answer = FileAnswer()
file_answer.write("1", Te, 'Checa el arreglo secciones')
END SOLUTION
```

```
print('T exacta = {}'.format(Te))
```

```
T exacta = {} [1. 0.9382716 0.86419753 0.77777778 0.67901235 0.56790123
 0.44444444 0.30864198 0.16049383 0.]
```

```
quizz.eval_numeric('1', Te)
```

-----  
1 | Tu resultado es correcto.  
-----

## 5.2 Ejercicio 2. Error absoluto y error relativo.

El error absoluto y el error relativo se definen como sigue.

$$Error_{absoluto} = ||v_e - v_a||$$

$$Error_{relativo} = \frac{||v_e - v_a||}{||v_e||}$$

donde  $v_e$  es el valor exacto y  $v_a$  es el valor aproximado.

Implementa las fórmulas del  $Error_{absoluto}$  y del  $Error_{relativo}$  en la funciones `error_absoluto()` y `error_relativo()`, respectivamente.

```
def error_absoluto(ve, va):
 """
 Calcula el error absoluto entre el valor exacto (ve) y el valor aproximado (va)
 """
 ### BEGIN SOLUTION
 return np.linalg.norm(ve - va)
 ### END SOLUTION
```

```
def error_relativo(ve, va):
 """
 Calcula el error relativo entre el valor exacto (ve) y el valor aproximado (va)
 """
 # BEGIN SOLUTION
 return np.linalg.norm(ve - va) / np.linalg.norm(ve)
 # END SOLUTION
```

## 5.3 Ejercicio 3. Solución numérica (interactivo).

Si todo lo realizaste correctamente, ejecuta la siguiente celda para generar un interactivo. Mueve los valores de  $k$ ,  $S$  y  $N$  y observa lo que sucede.

```
def conduccion_1d(k, S, L, TA, TB, N):
 """
 Calcula la temperatura en 1D mediante diferencias finitas.

 Parameters

 L: float
 L es la longitud del dominio.

 k: float
 es la conductividad del material.

 S: float
 es la fuente.

 TA: float
 Es la condición de frontera a la izquierda.

 TB: float
 Es la condición de frontera a la derecha.

 N: int
 Es el número de nodos internos (grados de libertad).

 Return

 al final esta función dibuja la solución.
 """

 # Cálculo de algunos parámetros numéricos
 h = L / (N+1)
 r = k / h**2

 # Definición de arreglos
 T = np.zeros(N+2)
 b = np.zeros(N)
 A = np.zeros((N,N))

 # Se inicializa todo el arreglo b con S/r
 b[:] = S / r
```

```

Condiciones de frontera en el arreglo de la Temperatura.
T[0] = TA
T[-1] = TB

Se ajusta el vector del lado derecho (RHS) con las condiciones de frontera.
b[0] += TA
b[-1] += TB

Se calculan las entradas de la matriz del sistema de ecuaciones lineales.
A[0,0] = 2
A[0,1] = -1
for i in range(1,N-1):
 A[i,i] = 2
 A[i,i+1] = -1
 A[i,i-1] = -1
A[-1,-2] = -1
A[-1,-1] = 2

Se resuelve el sistema lineal.
T[1:N+1] = np.linalg.solve(A,b)

Coordenadas para la solución exacta.
xe = np.linspace(0,L,100)

Coordenadas para la solución numérica.
xa = np.linspace(0,L,N+2)

Se calcula la solución exacta en las coordenadas xe.
Te = sol_exacta(xe, TA, TB, S, L, k)

Se calcula el error absoluto.
ea = error_absoluto(T, sol_exacta(xa,TA,TB,S,L,k))

Se calcula el error relativo
er = error_relativo(T, sol_exacta(xa,TA,TB,S,L,k))

Se imprime el error absoluto y el relativo.
print('Error absoluto = {:.65e}, Error relativo = {:.65e}'.format(ea, er))

Se realiza la gráfica de la solución.
plt.plot(xa, T, 'o-', lw = 0.5, c='k', label = 'Numérica', zorder=5)
plt.plot(xe, Te, lw=5, c='limegreen', label = 'Exacta')
plt.xlabel('x')
plt.ylabel('T')
plt.legend()
plt.grid()
plt.show()

Construcción del interactivo.
widgets.interactive(conduccion_1d,
 k = widgets.FloatSlider(max=1.0, min=0.02, value=0.02, step=0.02,

```

```
S = widgets.FloatSlider(max=10.0, min=0.0, value=0, step=1.0)
L = widgets.fixed(5.0),
TA = widgets.fixed(200),
TB = widgets.fixed(1000),
N = widgets.IntSlider(max=10, min=4, value=4))
```