


# 1 Aritmética de Punto flotante

**Objetivo general.** - Revisar y entender cómo funcionan los números y la aritmética de punto flotante.

[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

## 2 Introducción.

La aritmética que se realiza en una computadora digital es diferente de la que se usa en matemáticas, por ejemplo:  $* 2 + 2 = 4$

`2 + 2`

4

- $4^2 = 16$

`4**2`

16

- $\$ ()^2 = 3 \$$

```
import math
math.sqrt(3)**2
```

2.9999999999999996

Observa que en este último ejemplo, el resultado no es exacto, solo aproximado al valor real.

En la aritmética continua se permite que un número real pueda tener un número infinito de dígitos.

$$\frac{1}{3} = 0.3333333 \dots 33333 \dots$$

Pero en una computadora solo puede representar un subconjunto de los números reales, el cual solo contiene números racionales (positivos y negativos).

`1/3`

0.3333333333333333

```
format(1/3, '.55f')
```

'0.3333333333333333148296162562473909929394721984863281250'

En el hardware de una computadora, los números se aproximan en base binaria; particularmente los números de punto flotante se aproximan con fracciones binarias. Por ejemplo, la fracción binaria  $0.101$  se aproxima como sigue:

$$0.101 = \frac{1}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} = 0.5 + 0.125 = 0.625$$

Observe que la suma de fracciones anterior se puede escribir como:

$$\frac{1 * 2^2 + 1}{2^3} = \frac{5}{2^3} = 0.625$$

es decir, la representación del número 0.625 está dada en la forma:  $\frac{J}{2^N}$ .

En base 10 el número anterior se puede representar como sigue:

$$0.625 = \frac{6}{10} + \frac{2}{10^2} + \frac{5}{10^3}$$

**Pero no siempre es posible representar los números de punto flotante con fracciones binarias.**

Como se puede observar anteriormente, la fracción  $1/3$  se puede aproximar como una fracción en base 10. Por ejemplo:

$$\begin{aligned} 0.3 &= \frac{3}{10} \\ 0.33 &= \frac{3}{10} + \frac{3}{100} \\ 0.333 &= \frac{3}{10} + \frac{3}{100} + \frac{3}{1000} \end{aligned}$$

Pero en binario no es posible representar  $1/3$  de manera exacta.

Otro ejemplo es el número  $0.1 = 1/10$ :

```
print(0.1)
```

0.1

```
print(1/10) # El resultado que imprime se ve correcto
            # pero el resultado de la operación es
            # la fracción binaria representable más cercana
            # al valor correcto.
```

0.1

```
format(0.1, '.55f')
```

```
'0.1000000000000000055511151231257827021181583404541015625'
```

```
format(1/10, '.55f')
```

```
'0.1000000000000000055511151231257827021181583404541015625'
```

Existen varios números decimales que comparten la misma fracción binaria más aproximada.

Por ejemplo, los siguientes números

0.1  
 0.100000000000000001  
 0.1000000000000000055511151231257827021181583404541015625

son todos aproximados por la siguiente fracción:

$$\frac{3602879701896397}{2^{55}}$$

```
format(3602879701896397 / 2 ** 55, '.55f')
```

```
'0.1000000000000000055511151231257827021181583404541015625'
```

```
format(0.100000000000000001, '.55f')
```

```
'0.1000000000000000055511151231257827021181583404541015625'
```

```
format(0.1000000000000000055511151231257827021181583404541015625, '.55f')
```

```
'0.1000000000000000055511151231257827021181583404541015625'
```

Esto es muy importante cuando se usan constantes matemáticas en cálculos numéricos

```
print(str(math.pi))          # Se redondea el valor real almacenado
print(repr(math.pi))         # para propósitos de despliegue
print(format(math.pi, '.55f')) # muestra el valor real almacenado
```

```
3.141592653589793
3.141592653589793
3.1415926535897931159979634685441851615905761718750000000
```

**Observación:** Esto puede crear ciertas ilusiones sobre el valor real de un número.

## 2.1 Ejemplo 1.

¿Que resultará de las siguientes evaluaciones?:

```
0.1 == 1/10
0.1 == repr(1/10)
repr(0.1) == 1/10
.1 + .1 + .1 == 0.3
round(.1, 1) + round(.1, 1) + round(.1, 1) == round(.3, 1)
round(.1 + .1 + .1, 10) == round(.3, 10)
```

Explique el resultado de las evaluaciones.

**Hint:** Checar el valor más aproximado almacenado en memoria usando por ejemplo `format(x, '.52f')` y también usar el comando `type` para conocer el tipo de dato.

```
0.1 == 1/10
```

True

```
0.1 == repr(1/10)
```

False

```
repr(0.1) == 1/10
```

False

```
0.1 + 0.1 + 0.1 == 0.3
```

False

```
format(0.3, '.55f')
```

```
'0.2999999999999999888977697537484345957636833190917968750'
```

```
round(.1, 1) + round(.1, 1) + round(.1, 1) == round(.3, 1)
```

False

```
round(.1 + .1 + .1, 10) == round(.3, 10)
```

True

### 3 Sistema Numérico de Punto Flotante

- Las computadoras cuentan con una cierta capacidad finita para almacenar información.
- Los números reales se representan mediante los llamados **números de punto flotante** (*floating point numbers*) usando las siguientes características:

- Signo + o - .
- Mantisa con  $t$  dígitos; donde  $t$  es un entero positivo mayor o igual a 1.
- Base  $\beta$ ; donde  $\beta$  es un entero positivo mayor que 1
- Exponente  $e$ ; donde  $m \leq e \leq M$  con  $m \leq 0$  y  $M > t$

- Cada número de punto flotante se representa como:

$$\pm .d_1d_2d_3 \dots d_t \times \beta^e$$

donde  $0 \leq d_i \leq \beta - 1$  ( $i = 1, 2, 3, \dots, t$ )

- La forma normalizada ocurre cuando  $d_1 \neq 0$
- El número de dígitos en la mantisa es finito, lo que propicia un error en la representación y en las operaciones aritméticas

### 3.1 Ejemplo 2.

IBM 3000 series

- Sistema numérico de punto flotante (SNPF) de simple precisión: 1 dígito binario (bit) para el signo, 7 bits para el exponente en base 16, y 24 bits para la mantisa.
  - 24 dígitos binarios corresponde a  $\approx 6$  dígitos decimales.
  - El exponente va de 0000000 = 0 a 1111111 = 127.
  - Para asegurar la representación de números de magnitud pequeña, se resta 64 al exponente, de tal manera que el rango en realidad es de -64 a 63.

Signo	$e$	$t$
0	1000010	101100110000010000000000

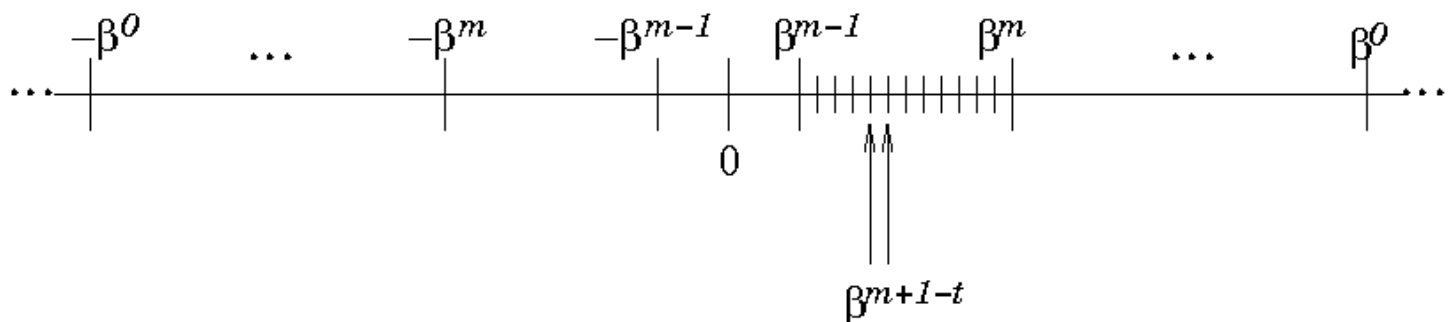
$$1000010 = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 66 \implies 16^{66-64}.$$

$$\left[ \left( \frac{1}{2} \right)^1 + \left( \frac{1}{2} \right)^3 + \left( \frac{1}{2} \right)^4 + \left( \frac{1}{2} \right)^7 + \left( \frac{1}{2} \right)^8 + \left( \frac{1}{2} \right)^{14} \right] 16^{66-64} = 179.015625$$

- En este sistema, el siguiente número más pequeño y el siguiente más grande son:

0	1000010	101100110000001111111111	=	179.0156097412109375
0	1000010	1011001100000100000000001	=	179.0156402587890625

$$\implies 179.015625 \text{ representa } [179.0156097412109375, 179.0156402587890625]$$



En 1985, el IEEE (Institute for Electrical and Electronic Engineers) publicó: *Binary Floating Point Arithmetic Standard 754, 1985*. Se especifican los formatos para precisión simple, doble y extendida, y esos estándares son usados por muchos constructores de CPUs.

## 4 Error de representación

Recordemos que muchas fracciones decimales no pueden representarse exactamente como fracciones binarias. Por esta razón, todos los lenguajes de programación no muestran el número decimal exacto que se espera.

La mayoría de las computadoras utilizan el estándar IEEE-754 para representar números de punto flotante. Los números de doble precisión en el IEEE-754 tienen 53 bits de precisión, por lo tanto, la representación del número 0.1 en este sistema requiere de un número  $J$ , de 53 bits, tal que:

$$\frac{J}{2^N} \approx 0.1$$

Usando la fórmula anterior tenemos que:

$$J \approx \frac{2^N}{10}$$

Dado que  $J$  debe tener exactamente 53 bits, entonces se cumple que  $2^{52} \leq J \leq 2^{53}$ . Se puede comprobar fácilmente que para que esto ocurra, entonces  $N = 56$ :

```
J = 2**56 // 10 # división entera
print('{} < {} < {}'.format(2**52, J, 2**53))
print(2**52 < J < 2**53)
```

```
4503599627370496 < 7205759403792793 < 9007199254740992
True
```

Observe que en la celda anterior realizamos la división entera con el propósito de comparar con otros valores enteros. Si hacemos el cálculo usando flotantes y luego convertimos a entero obtenemos el valor requerido de  $J$ :

```
J = int(2**56 / 10)
J
```

```
7205759403792794
```

Entonces, la representación de 0.1 en este sistema es  $\frac{7205759403792794}{2^{56}}$  que se puede reducir a  $\frac{3602879701896397}{2^{55}}$ , por lo tanto:

```
representacion = 3602879701896397 / 2**55
```

```
0.1 == representacion
```

```
True
```

```
print('0.1 ~ {:.55f} en el sistema en IEEE-754'.format(representacion))
```

```
0.1 ~ 0.1000000000000000055511151231257827021181583404541015625 en el sistema en IEEE-754
```

La mayoría de los lenguajes utilizan 17 dígitos significativos para desplegar los números de punto flotante:

```
print('0.1 ~ {:.17f} en el sistema en IEEE-754'.format(representacion))
```

0.1 ~ 0.10000000000000001 en el sistema en IEEE-754

## 5 Algunas funciones útiles

```
from decimal import Decimal
from fractions import Fraction
```

```
Fraction.from_float(0.1) # regresa los enteros usados para
                        # la representación del número flotante
```

```
Fraction(3602879701896397, 36028797018963968)
```

```
3602879701896397 / 36028797018963968
```

0.1

```
2**55
```

36028797018963968

```
print('0.1 ~ {:.55f}'.format(3602879701896397 / 36028797018963968))
```

0.1 ~ 0.1000000000000000055511151231257827021181583404541015625

```
(0.1).as_integer_ratio() # regresa los enteros usados para
                        # la representación del número flotante
```

```
(3602879701896397, 36028797018963968)
```

```
Decimal.from_float(0.1)
```

```
Decimal('0.1000000000000000055511151231257827021181583404541015625')
```

```
## Cualquier otro número
x = 3.14159
x.as_integer_ratio()
```

```
(3537115888337719, 1125899906842624)
```

```
3537115888337719 / 1125899906842624
```

3.14159

### Conversión decimal a hexadecimal

```
x.hex() # Convierte el flotante a hexadecimal
```

```
'0x1.921f9f01b866ep+1'
```

```
float.fromhex('0x1.921f9f01b866ep+1') # convierte el hexadecimal a flotante
```

3.14159

```
x == float.fromhex('0x1.921f9f01b866ep+1')
```

True

El uso de hexadecimales es útil para portabilidad de valores entre diferentes versiones de Python y para el intercambio de información con otros lenguajes.

## Mitigación del error

Es posible que en algunos casos se pierda precisión de tal manera que el resultado se vea afectado, por ejemplo:

```
lista = [0.1] * 10
lista
```

```
[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
```

```
suma = sum(lista)
print('Suma = {:.52f}'.format(suma))
```

Suma = 0.999999999999998889776975374843459576368331909179688

```
suma = math.fsum(lista)
print('Suma = {:.52f}'.format(suma))
```

Suma = 1.000

La función `math.fsum()` ayuda a **mitigar** la pérdida de precisión durante la suma.

## Decimal

Con esta biblioteca, los números se pueden representar de manera exacta y es muy útil cuando se requiere de una “aritmética exacta”.

```
from decimal import getcontext, Decimal
getcontext()
```

```
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=[InvalidOperation, DivisionByZero, Overflow])
```

```
getcontext().prec = 10
print(getcontext())

r = Decimal(0.1) + Decimal(0.1) + Decimal(0.1)
print('\n 0.3 ~ {:.55f}'.format(r))
```



```
Context(prec=10, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=[InvalidOperation, DivisionByZero, Overflow])
```

$0.3 \sim 0.300$

## Truncamiento y redondeo

Dado el número  $\pi = 3.1415926535897931159979634 \dots = 0.31415926535897931159979634 \dots \times 10^1$ , podemos hacer lo siguiente:

- **Truncamiento:** Usando 5 dígitos:  $\$0.31415 \times 1 = 3.1415 \$$
- **Redondeo:** Usando 5 dígitos:  $(0.31415 + 0.00001) \times 10^1 = 3.1416$

Opciones de redondeo: - ROUND\_CEILING, - ROUND\_DOWN, - ROUND\_FLOOR, - ROUND\_HALF\_DOWN, - ROUND\_HALF\_EVEN, - ROUND\_HALF\_UP, - ROUND\_UP, - ROUND\_05UP.

```
from decimal import setcontext, Context, ROUND_CEILING, ROUND_FLOOR, ROUND_HALF_DOWN, ROUND_H
redondeo = ['ROUND_CEILING', 'ROUND_FLOOR', 'ROUND_HALF_DOWN',
            'ROUND_HALF_EVEN', 'ROUND_HALF_UP', 'ROUND_UP', 'ROUND_05UP']
for r in redondeo:
    setcontext(Context(prec=6, rounding=r))
    print(getcontext())
    print(Decimal(3.1415926535897931159979634) * 1, '\n')
```

```
Context(prec=6, rounding=ROUND_CEILING, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[],
traps=[InvalidOperation, DivisionByZero, Overflow])
3.14160
```

```
Context(prec=6, rounding=ROUND_FLOOR, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[],
traps=[InvalidOperation, DivisionByZero, Overflow])
3.14159
```

```
Context(prec=6, rounding=ROUND_HALF_DOWN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[],
traps=[InvalidOperation, DivisionByZero, Overflow])
3.14159
```

```
Context(prec=6, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[],
traps=[InvalidOperation, DivisionByZero, Overflow])
3.14159
```

```
Context(prec=6, rounding=ROUND_HALF_UP, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[],
traps=[InvalidOperation, DivisionByZero, Overflow])
3.14159
```

```
Context(prec=6, rounding=ROUND_UP, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=
[InvalidOperation, DivisionByZero, Overflow])
3.14160
```

```
Context(prec=6, rounding=ROUND_05UP, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=[InvalidOperation, DivisionByZero, Overflow])
3.14159
```

## Fractions

```
from fractions import Fraction

num1 = Fraction(2,3)
```

```

num2 = Fraction(1,3)

print("num1 = {} and num2 = {}".format(num1,num2))

print(num1 + num2)

print(num1 - num2)

print(num1*10)

print(num1/num2)

```

num1 = 2/3 and num2 = 1/3

1  
1/3  
20/3  
2

## 6 Características del SNPF en Python 3

**NumPy** soporta una variedad más amplia de tipos numéricos.

Data type	Description
<code>bool_</code>	Boolean (True or False) stored as a byte
<code>int_</code>	Default integer type (same as C <code>long</code> ; normally either <code>int64</code> or <code>int32</code> )
<code>intc</code>	Identical to C <code>int</code> (normally <code>int32</code> or <code>int64</code> )
<code>intp</code>	Integer used for indexing (same as C <code>ssize_t</code> ; normally either <code>int32</code> or <code>int64</code> )
<code>int8</code>	Byte (-128 to 127)
<code>int16</code>	Integer (-32768 to 32767)
<code>int32</code>	Integer (-2147483648 to 2147483647)
<code>int64</code>	Integer (-9223372036854775808 to 9223372036854775807)
<code>uint8</code>	Unsigned integer (0 to 255)
<code>uint16</code>	Unsigned integer (0 to 65535)
<code>uint32</code>	Unsigned integer (0 to 4294967295)
<code>uint64</code>	Unsigned integer (0 to 18446744073709551615)
<code>float_</code>	Shorthand for <code>float64</code> .
<code>float16</code>	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
<code>float32</code>	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
<code>float64</code>	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
<code>complex_</code>	Shorthand for <code>complex128</code> .
<code>complex64</code>	Complex number, represented by two 32-bit floats (real and imaginary components)
<code>complex128</code>	Complex number, represented by two 64-bit floats (real and imaginary components)

```

import numpy as np
x = np.float64(0.1)
print(format(x, '.52f'))

```

```
print(type(x))

y = np.int_([1,2,4])
print(y)
print(type(y))
print(type(y[1]))

z = np.arange(3, dtype=np.uint8)
print(z)
print(type(z))
print(type(z[0]))
```

```
0.1000000000000000055511151231257827021181583404541016
<class 'numpy.float64'>
[1 2 4]
<class 'numpy.ndarray'>
<class 'numpy.int64'>
[0 1 2]
<class 'numpy.ndarray'>
<class 'numpy.uint8'>
```

```
xd = math.pi
print(format(xd, '.52f'))
print(type(xd))

xdd = np.float64(xd)
print(format(xdd, '.52f'))
print(type(xdd))
```

```
3.1415926535897931159979634685441851615905761718750000
<class 'float'>
3.1415926535897931159979634685441851615905761718750000
<class 'numpy.float64'>
```

```
np.finfo(float)
```

```
finfo(resolution=1e-15, min=-1.7976931348623157e+308, max=1.7976931348623157e+308, dtype=float64)
```

```
np.finfo(np.float64)
```

```
finfo(resolution=1e-15, min=-1.7976931348623157e+308, max=1.7976931348623157e+308, dtype=float64)
```

```
np.finfo(np.float64).eps
```

```
2.220446049250313e-16
```

```
np.finfo(np.float64).nmant
```

```
print(np.finfo(float))
print(np.finfo(np.float32))
print(np.finfo(np.float16))
```

#### Machine parameters for float64

```
-----
precision = 15    resolution = 1.000000000000001e-15
machep = -52     eps = 2.2204460492503131e-16
negexp = -53     epsneg = 1.1102230246251565e-16
minexp = -1022   tiny = 2.2250738585072014e-308
maxexp = 1024    max = 1.7976931348623157e+308
nexp = 11        min = -max
smallest_normal = 2.2250738585072014e-308    smallest_subnormal = 4.9406564584124654e-324
-----
```

#### Machine parameters for float32

```
-----
precision = 6    resolution = 1.0000000e-06
machep = -23     eps = 1.1920929e-07
negexp = -24     epsneg = 5.9604645e-08
minexp = -126    tiny = 1.1754944e-38
maxexp = 128     max = 3.4028235e+38
nexp = 8         min = -max
smallest_normal = 1.1754944e-38    smallest_subnormal = 1.4012985e-45
-----
```

#### Machine parameters for float16

```
-----
precision = 3    resolution = 1.00040e-03
machep = -10     eps = 9.76562e-04
negexp = -11     epsneg = 4.88281e-04
minexp = -14     tiny = 6.10352e-05
maxexp = 16      max = 6.55040e+04
nexp = 5         min = -max
smallest_normal = 6.10352e-05    smallest_subnormal = 5.96046e-08
-----
```

- Para entender esto con mayor detalle véase:
  - <http://www.lahey.com/float.htm>
  - [https://docs.oracle.com/cd/E19957-01/806-3568/ncg\\_goldberg.html](https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html)

## 5 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

### 5.1 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)
```

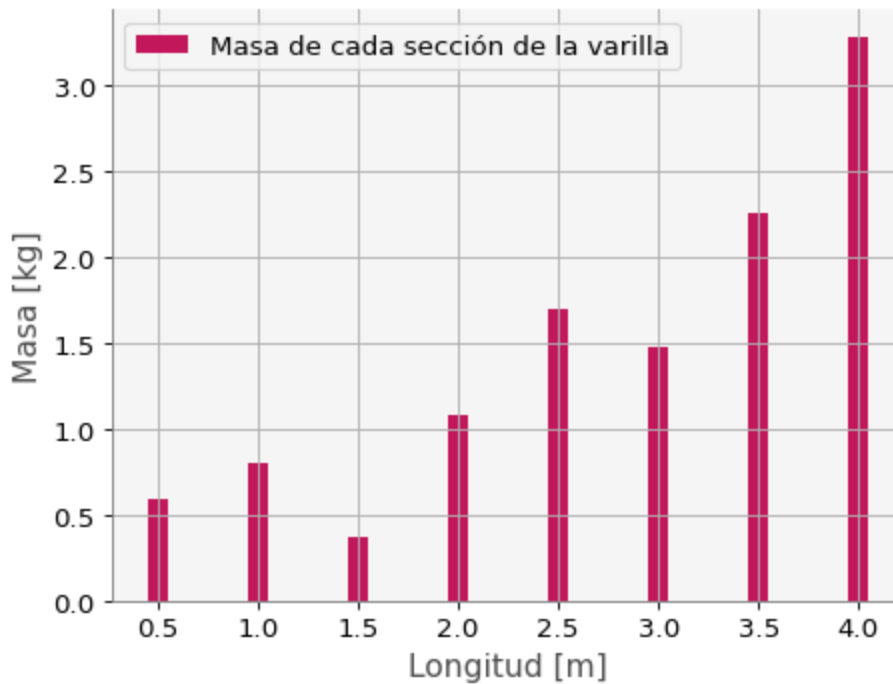
-----  
 1b | Tu resultado es correcto.  
 -----

```

# 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()

```



## 5.2 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]

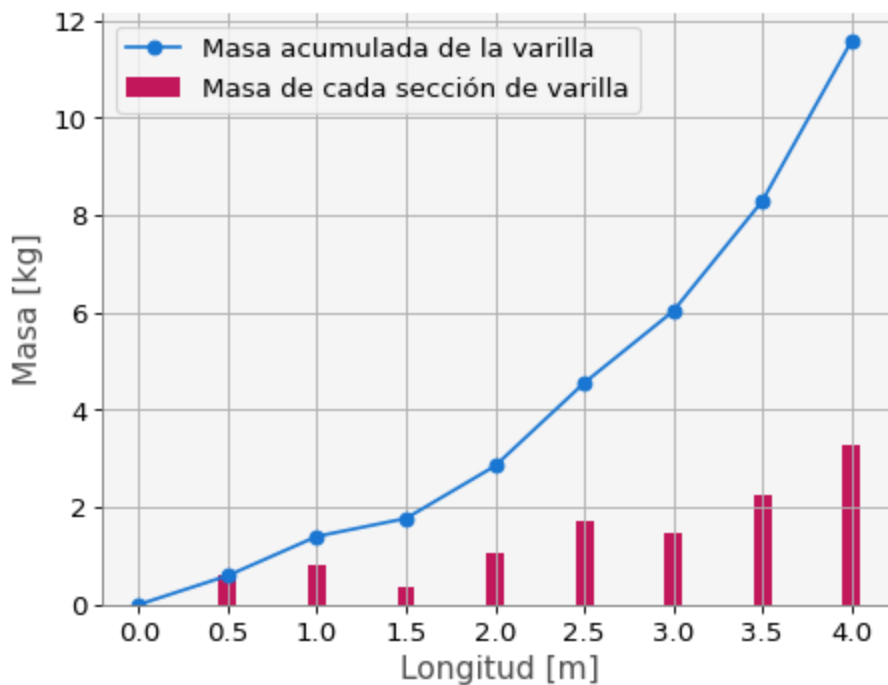
```
quizz.eval_numeric('2', masa)
```

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.



```

:::
:::

```

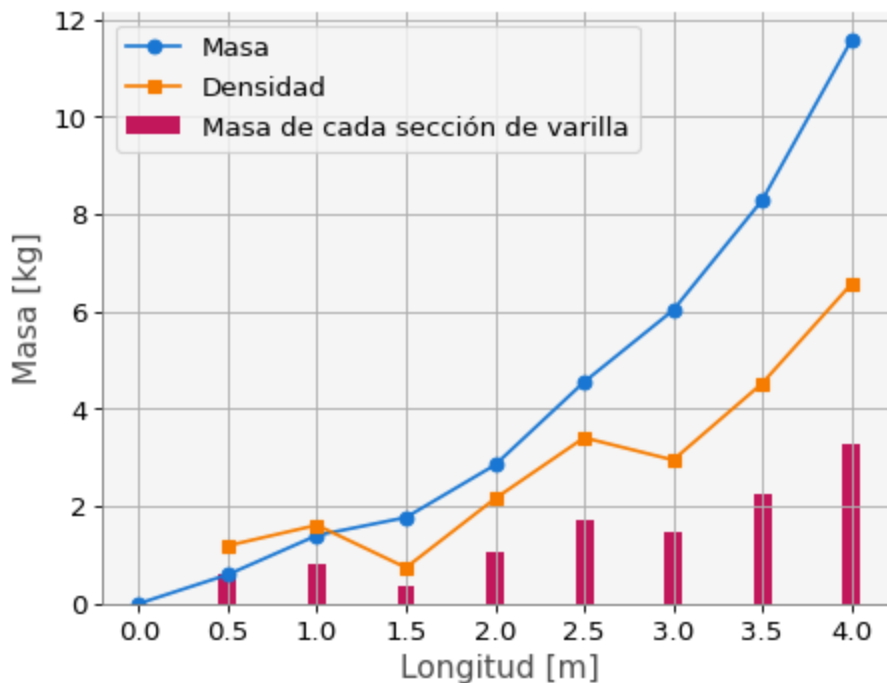
```

::: {#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.

## 5.3 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.  
 -----

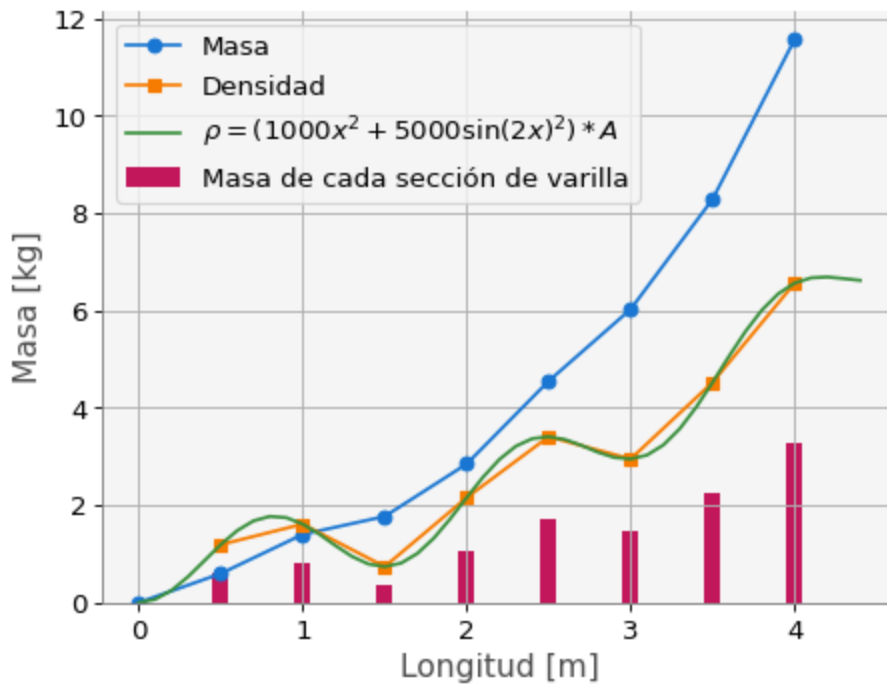
```
# Gráfica de la masa como función de las secciones
plt.plot(longitud, masa, 'o-', label='Masa')

# 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.

## 5.4 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.  
-----

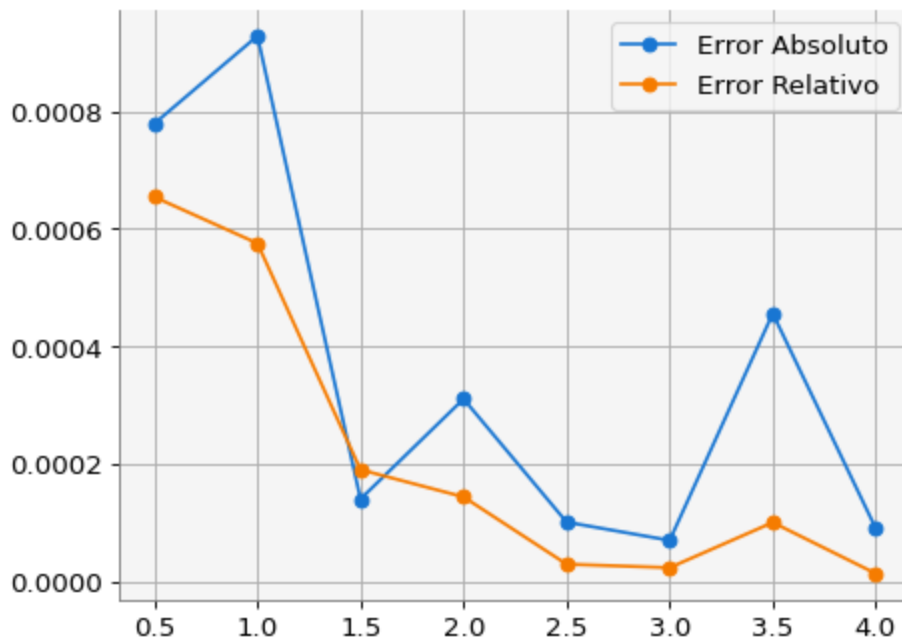
```
quizz.eval_numeric('5b', error_a)
```

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.

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

## 5.5 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)
```

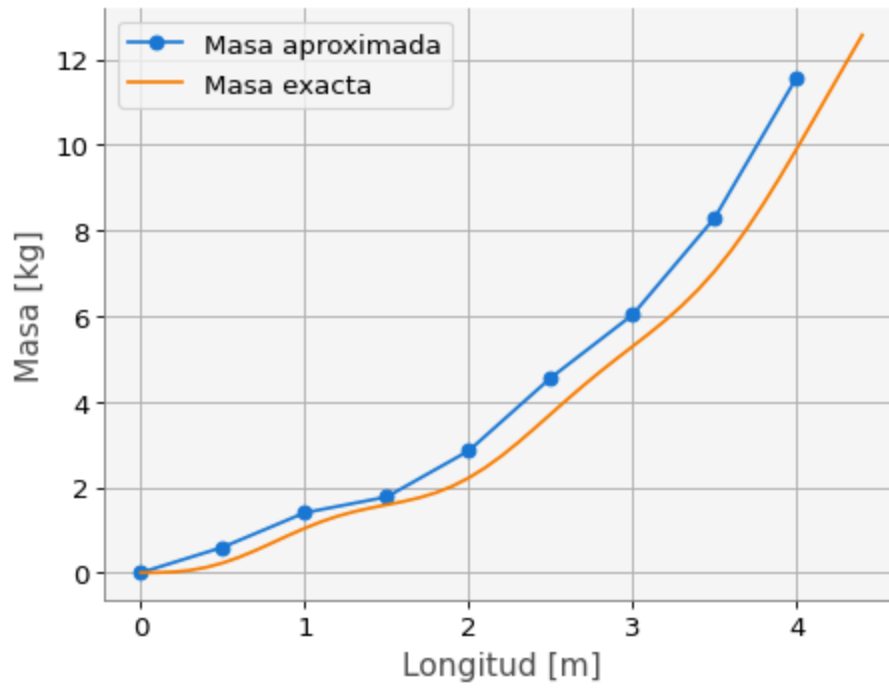
-----  
6b | Tu resultado es correcto.  
-----

```
# 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()
```





## 2 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')
```

### 2.1 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
```

#### 2.1.1 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.

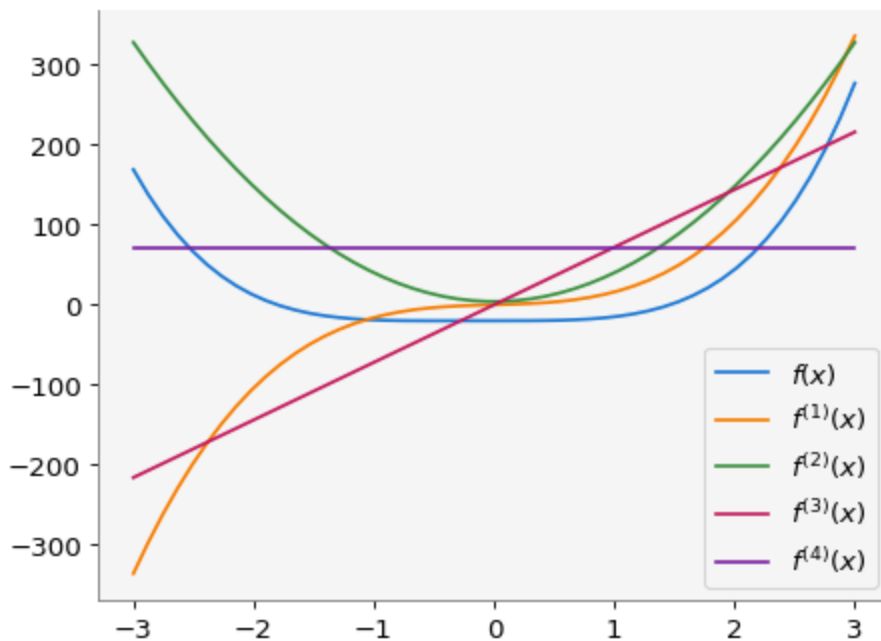
```
# Definimos la función y sus cuatro derivadas
f = lambda x: 3*x**4 + 2*x**3 -20

### 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)

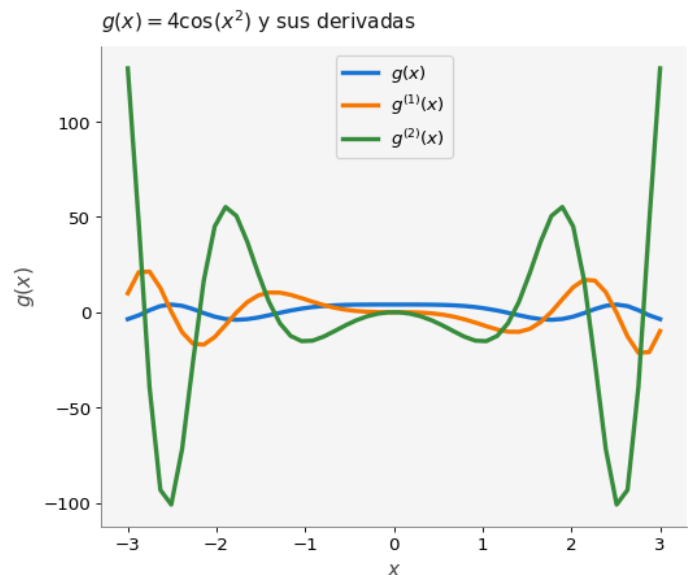
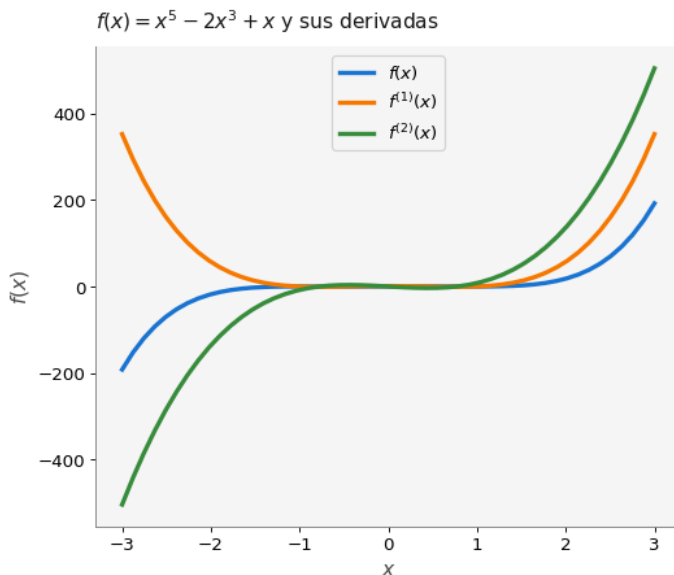
ax1.plot(xc, f(xc), label='$f(x)$',lw=3)
ax1.plot(xc, f1(xc), label='$f^{(1)}(x)$',lw=3)
ax1.plot(xc, f2(xc), label='$f^{(2)}(x)$',lw=3)
ax1.legend(loc='upper center')
ax1.set_title('$f(x)=x^5 - 2x^3 + x$ y sus derivadas')
ax1.set_xlabel

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.

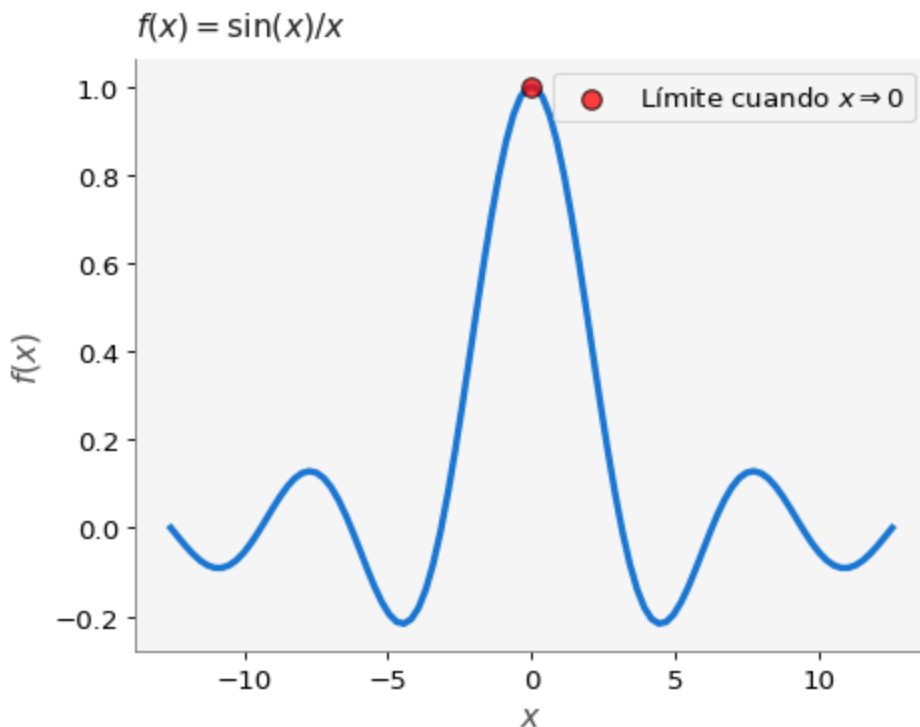
Al cumplirse las condiciones de la regla podemos asegurar que:

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

```
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("$f(x)=x^{2}+5$")

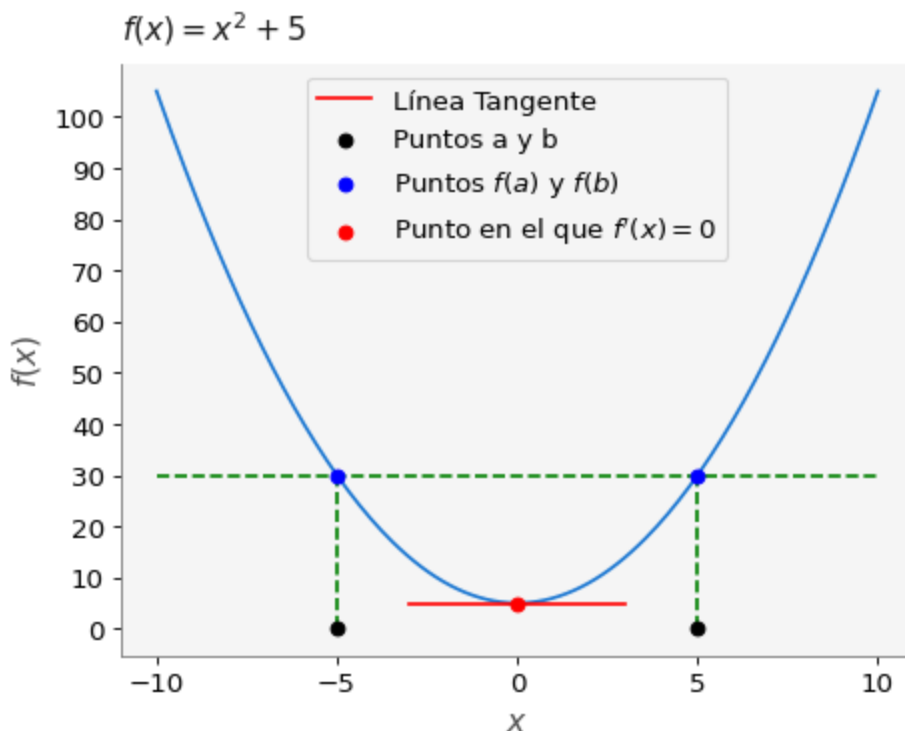
# Función
plt.plot(xc,f(xc))

# 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 com  $f^{(2)}(x)$ .


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

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

## 4 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`

## 4.1 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]
```

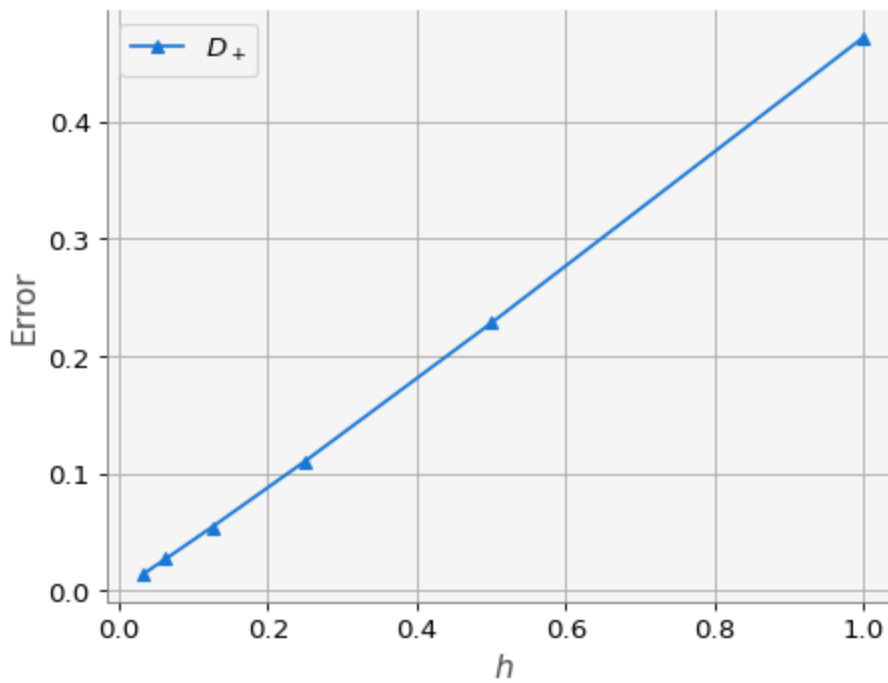
```
# Colocamos la información de h y del error en un Dataframe y mostramos el result
Error = pd.DataFrame(np.array([h, ef]).T,
                      columns=['$h$', '$D_+$'])

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
```

## 4.2 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)
```

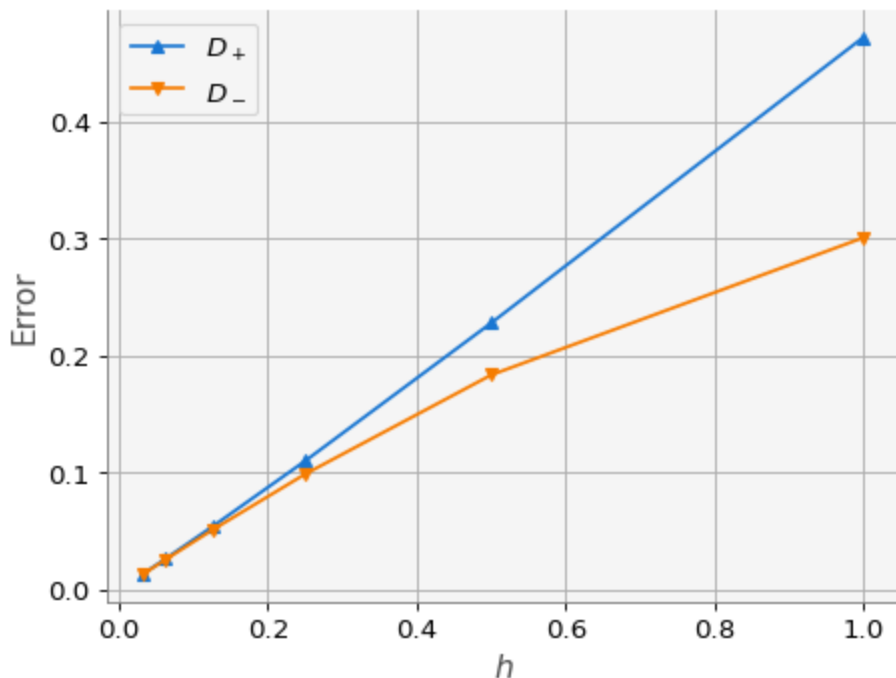
-----  
1 | Tu resultado es correcto.  
-----

```
# 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.

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

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)

## 4.3 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.  
-----

## 4.4 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['...'] = ...

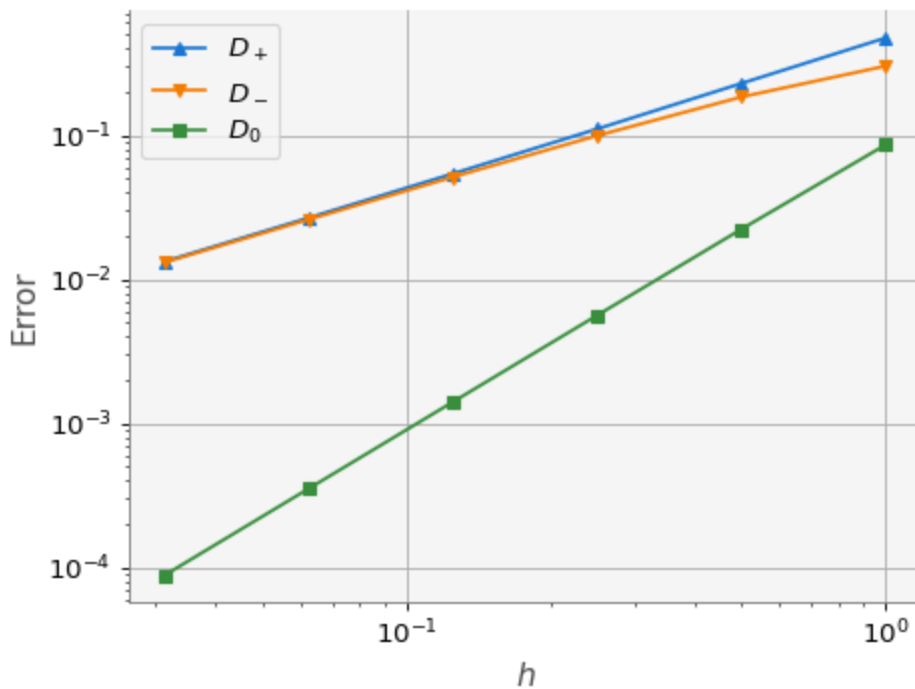
#### BEGIN SOLUTION
Error['$D_0$'] = ec
Error
#### END SOLUTION
Error
```

	<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.

## 4.5 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.  
-----

## 4.6 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

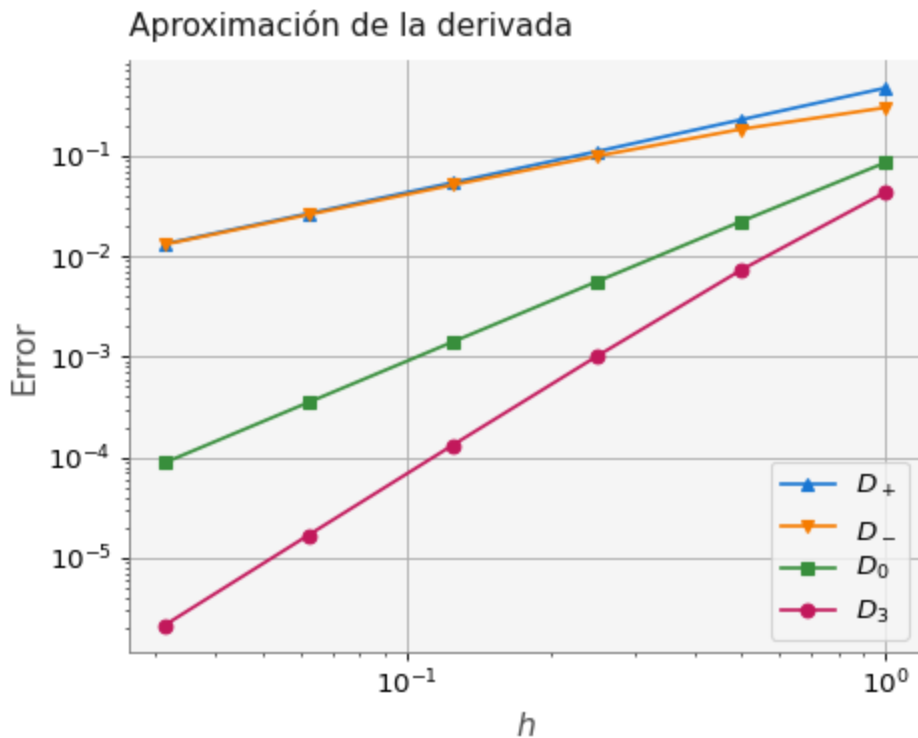
#### END SOLUTION
Error
```

	\$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()
```



## 4.7 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.

## 4.8 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

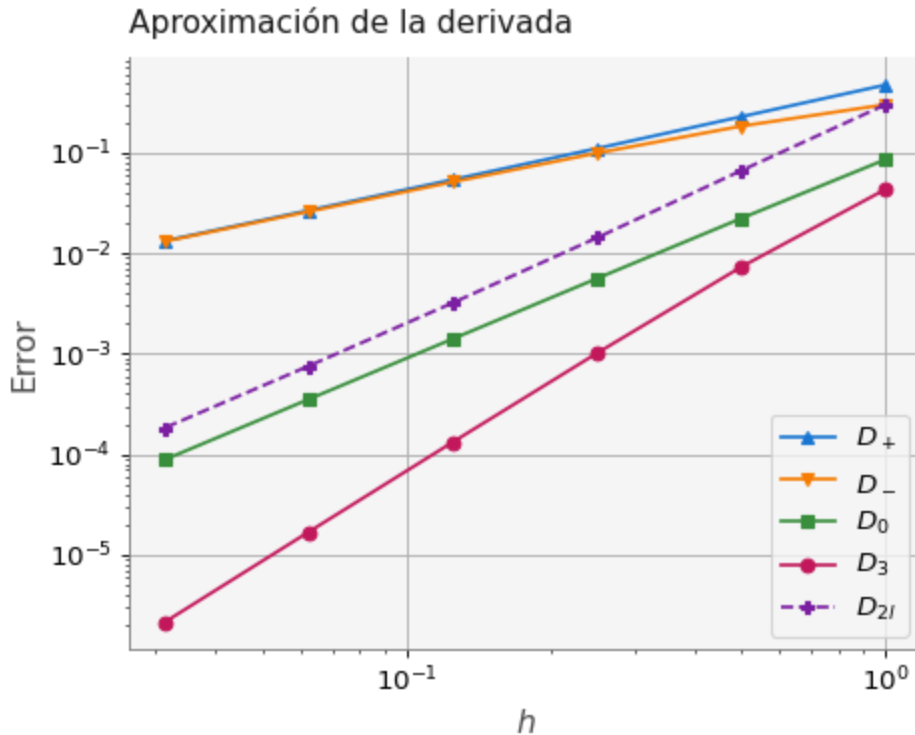
#### END SOLUTION
Error
```

	<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()
```



## 4.9 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.  
-----

## 4.10 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

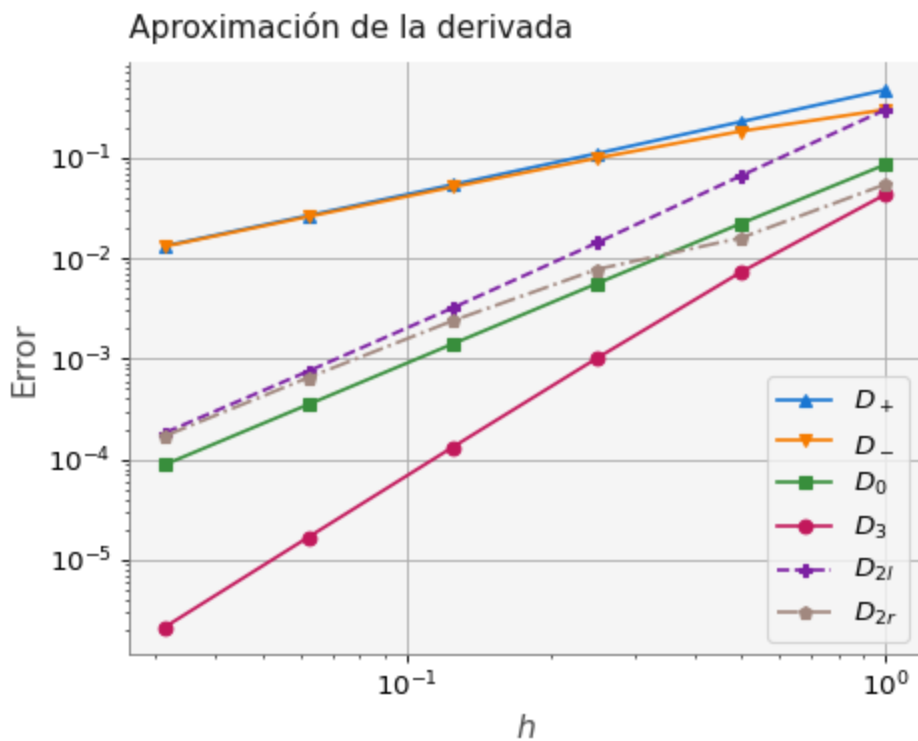
### END SOLUTION
Error
```

	\$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='$D_0$')
plt.plot(h, e3, 'o-', label='$D_3$')
plt.plot(h, e2l, 'P--', label='$D_{2l}$')
plt.plot(h, e2r, 'p-.', label='$D_{2r}$')

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()
```





# 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: *Séries 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  $h_x = h_y$ .

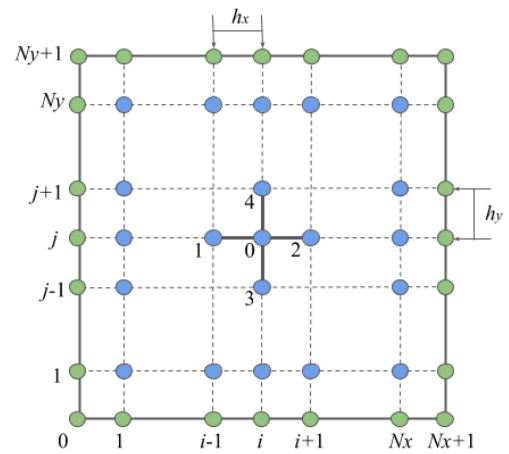
Usando esta aproximación, vamos a realizar una ejemplo de conducción de calor, pero para ello necesitamos conocer las herramientas de [numpy](#) y de [matplotlib](#).

## 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) # Espaciamiento 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. ]
```

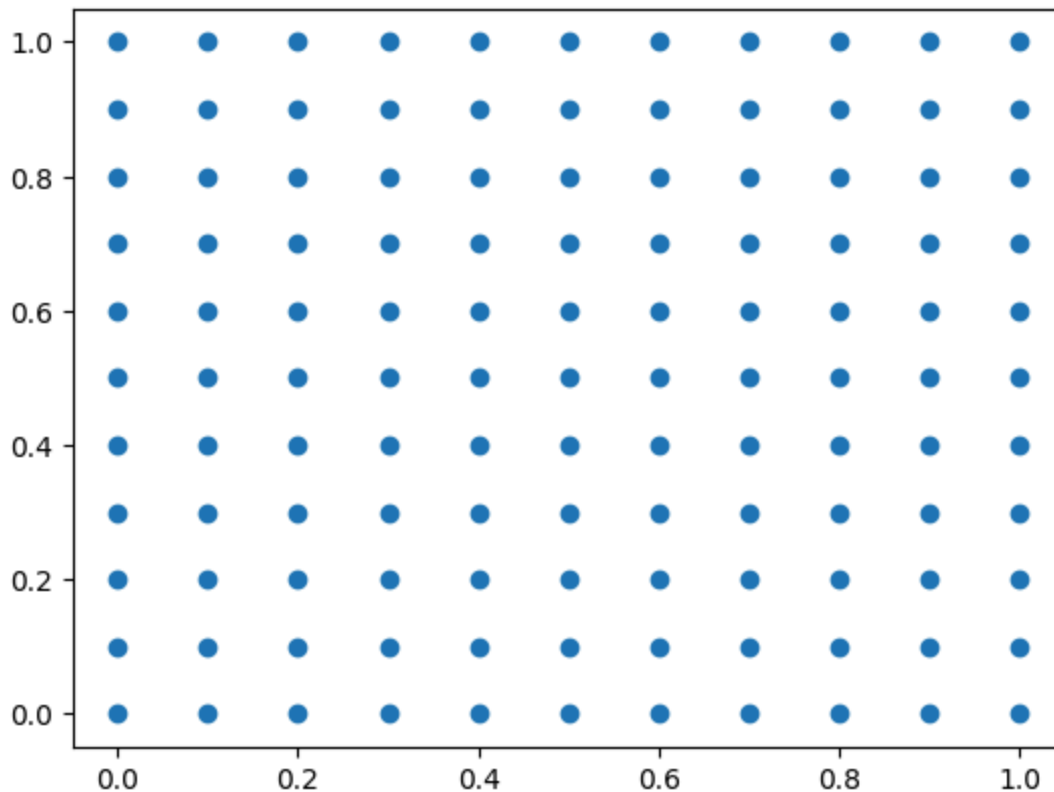
```
xg, yg = np.meshgrid(x, y) # Creamos la rejilla para usarla en Matplotlib
print(xg)
print(yg)
```

```
[[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)

```

```

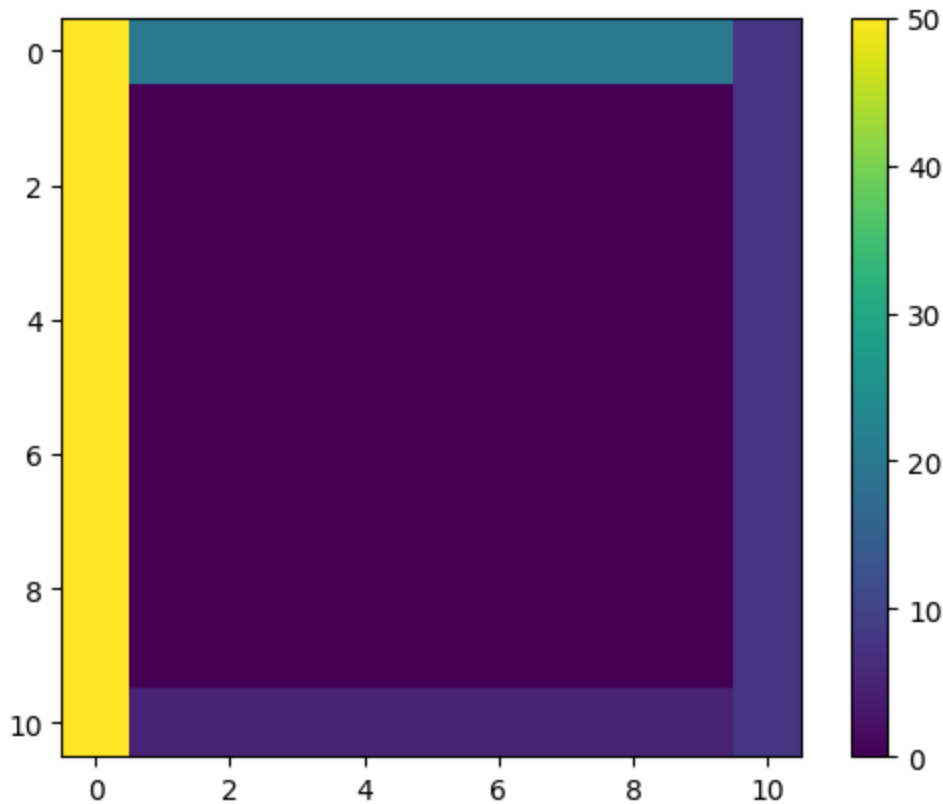
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 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[:]

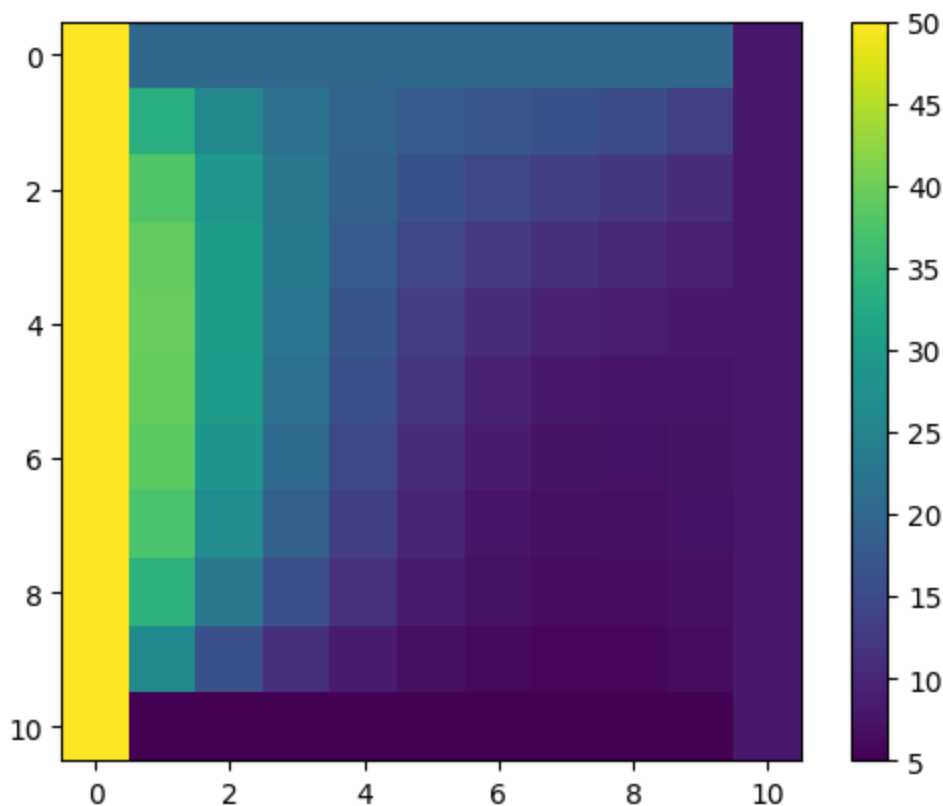
print(error_lista)

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

```

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

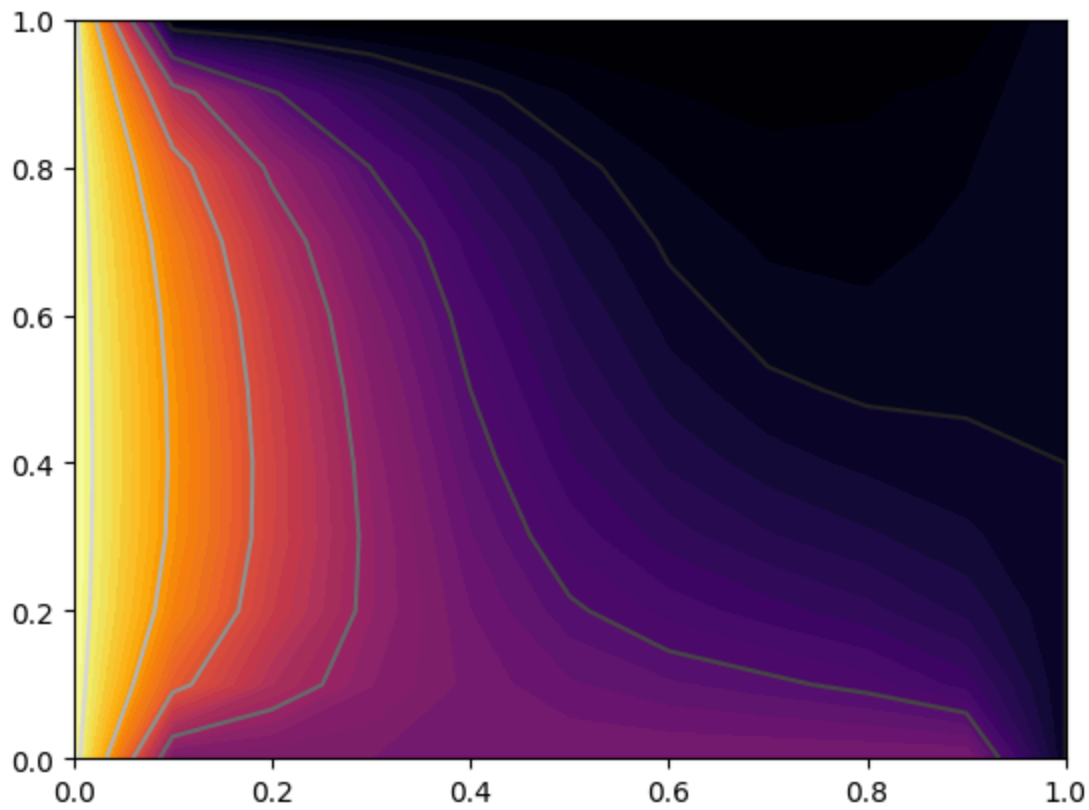
5.275727580531657, 4.867291496014421, 4.522054027452314, 4.226261372943586,  
 3.9698959855402296, 3.745493517737013, 3.547373613271126, 3.3711295389631717,  
 3.21328287724892, 3.0710454523767496, 2.9421521475712167, 2.824741357872151,  
 2.7172679513259683, 2.6184387521297463, 2.5271638648303663, 2.4425193146763595,  
 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]



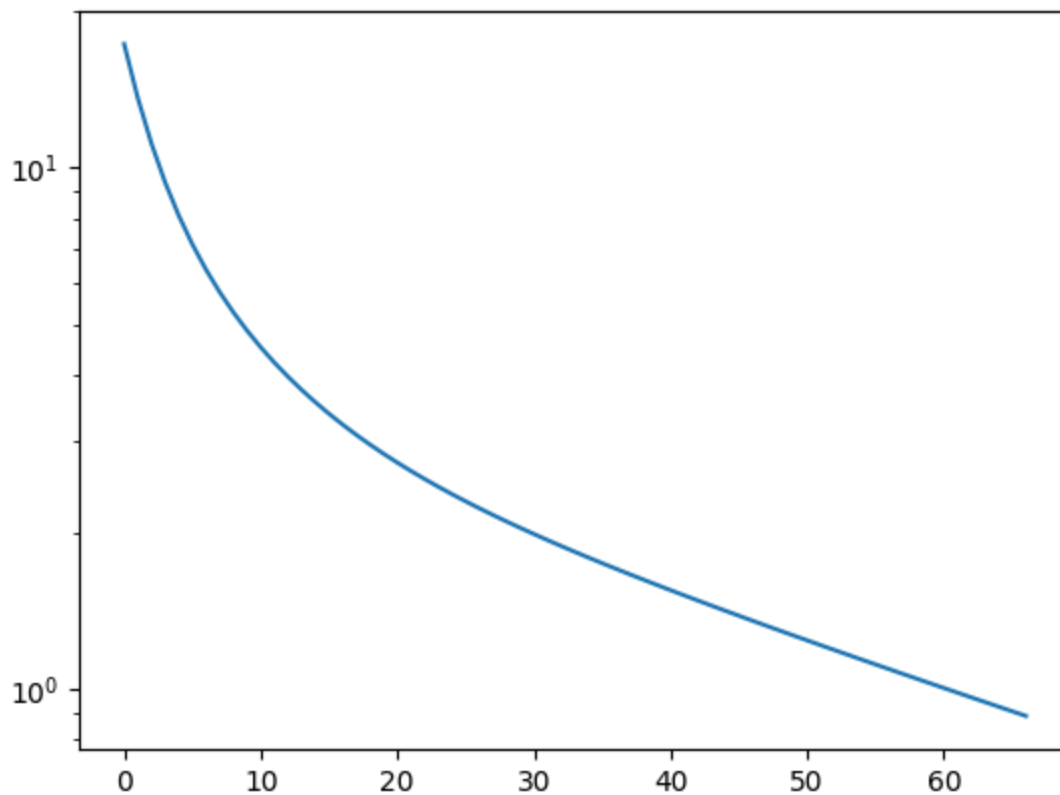
## 11.2 Ejercicio 2.

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 gráfícalo en semi-log. 3. Realiza las dos gráficas 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()

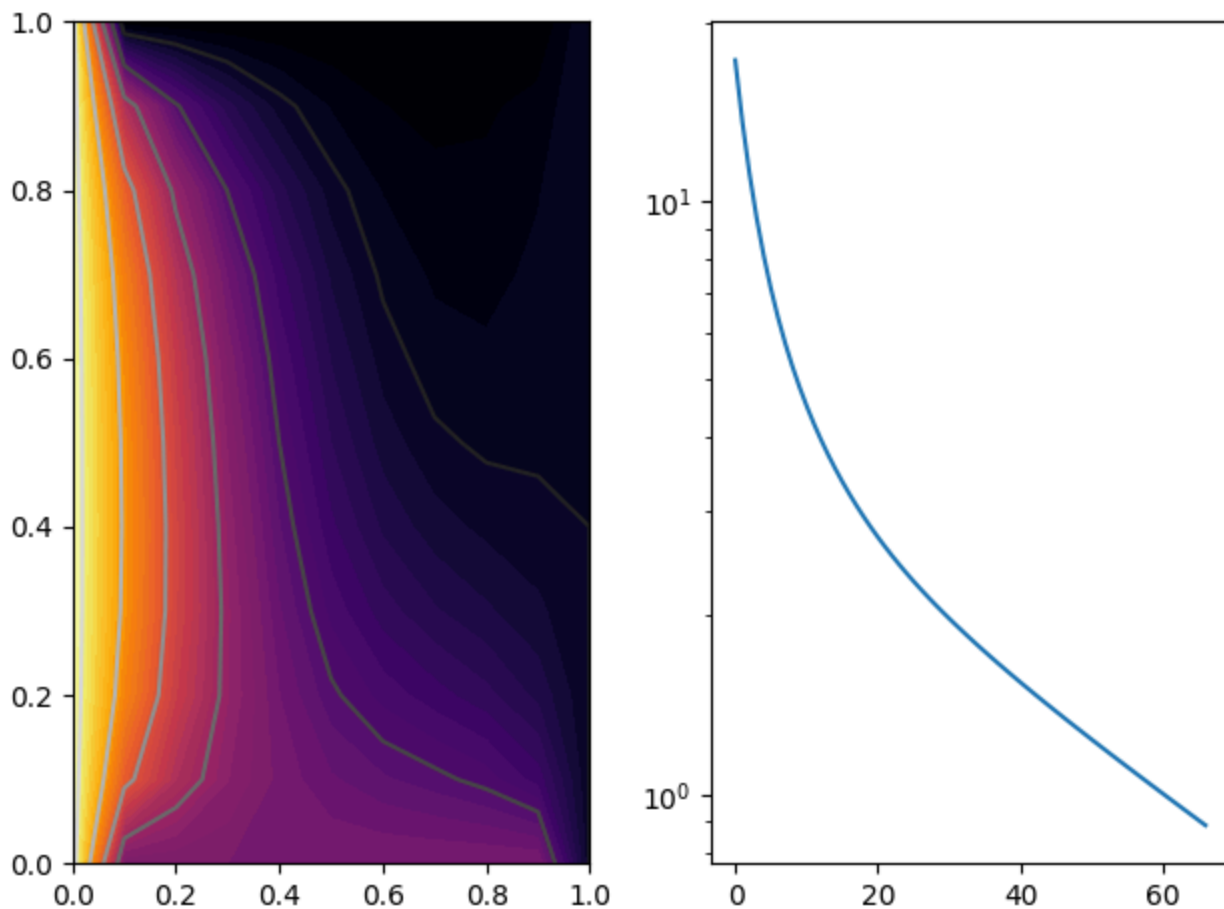
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

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.

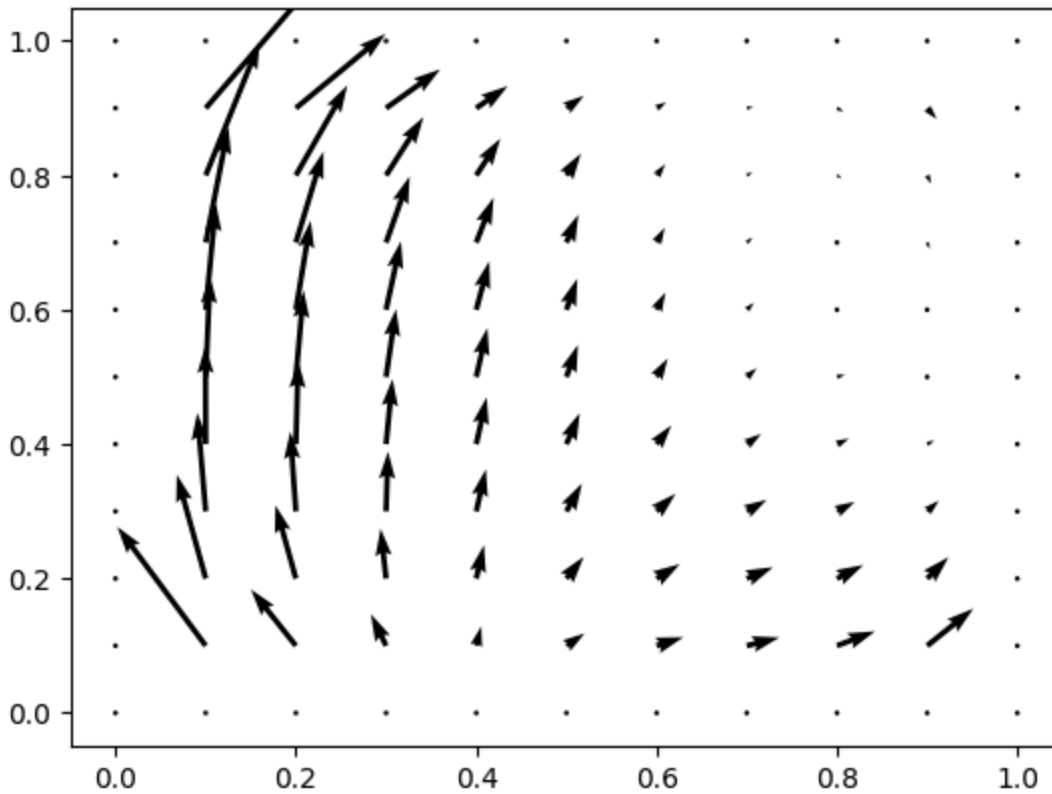
Usando la información calculada de la temperatura (almacenada en el arreglo `u`), vamos a calcular el flujo de calor usando la siguiente fórmula en diferencias:

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

```
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])

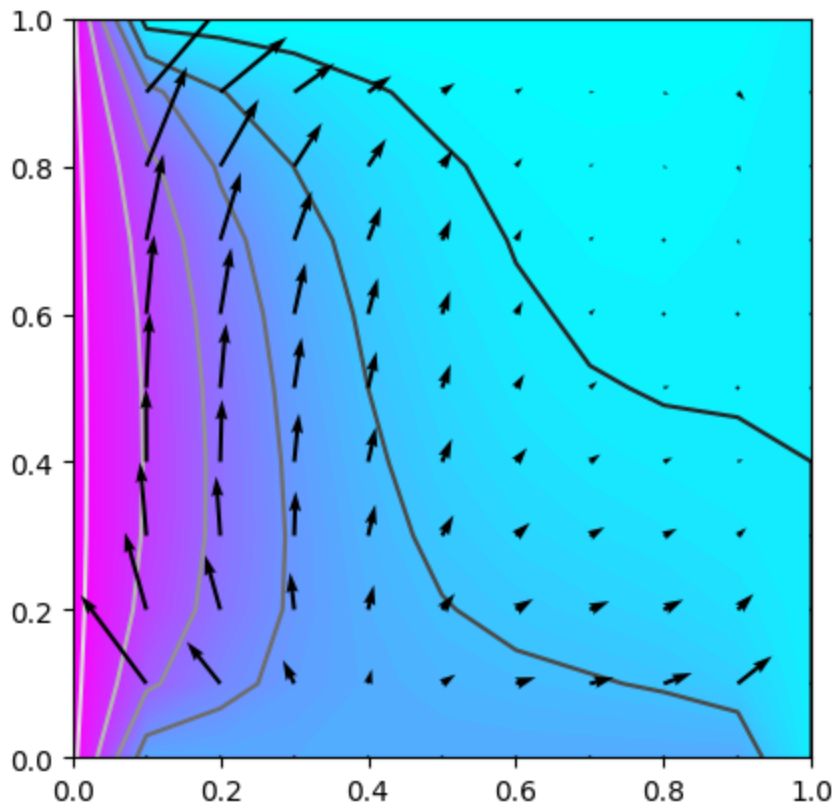
plt.quiver(xg, yg, qx, qy, scale=10, zorder=10)
```



## 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:

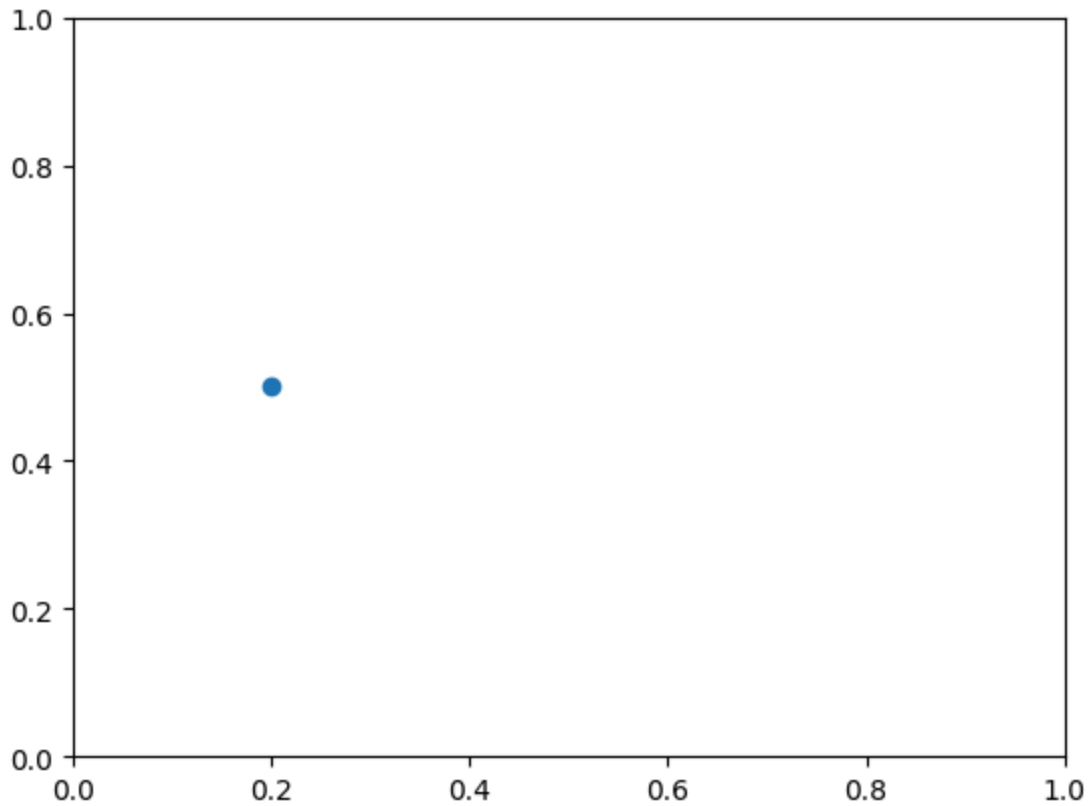
```
Pasos = 10
xp = np.zeros(Pasos)
yp = np.zeros(Pasos)
xp[0] = xo
yp[0] = yo
print(xp)
print(yp)
```

```
[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])
```

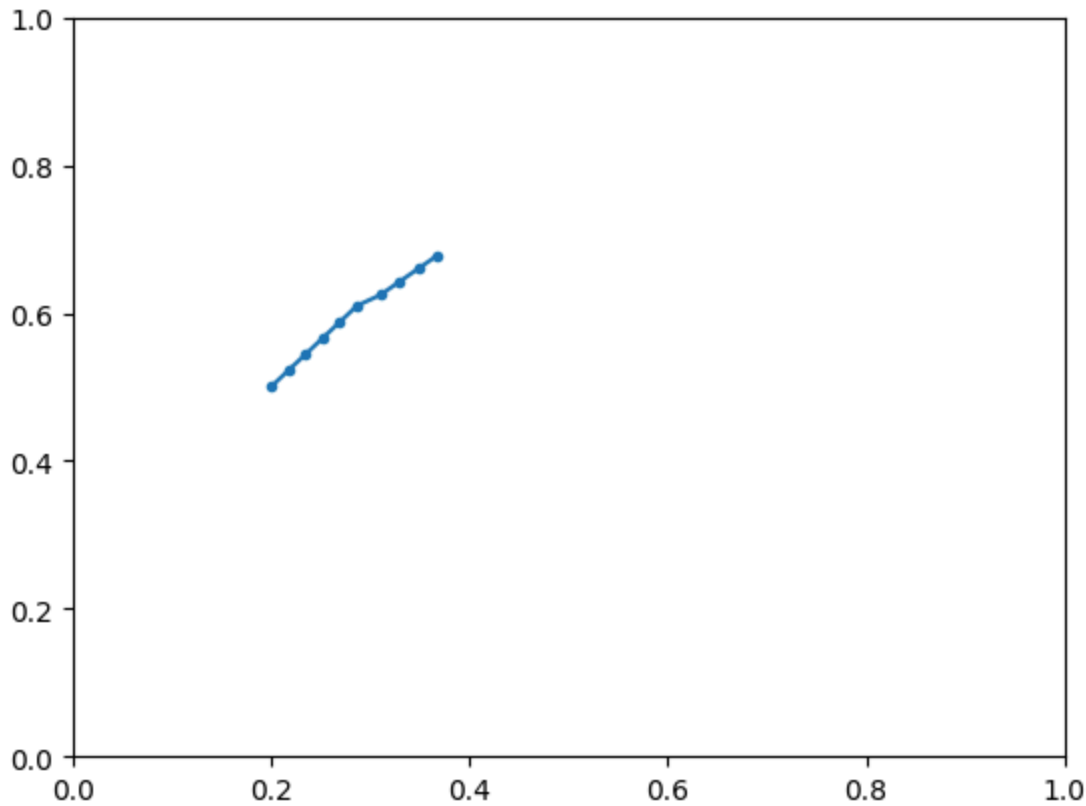
```
ht = 0.1
for n in range(1,Pasos):
    vx, vy = interpolaVel(qx, qy, xp[n-1], yp[n-1], h)
    xp[n] = xp[n-1] + ht * vx
    yp[n] = yp[n-1] + ht * vy
```

```
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)
```

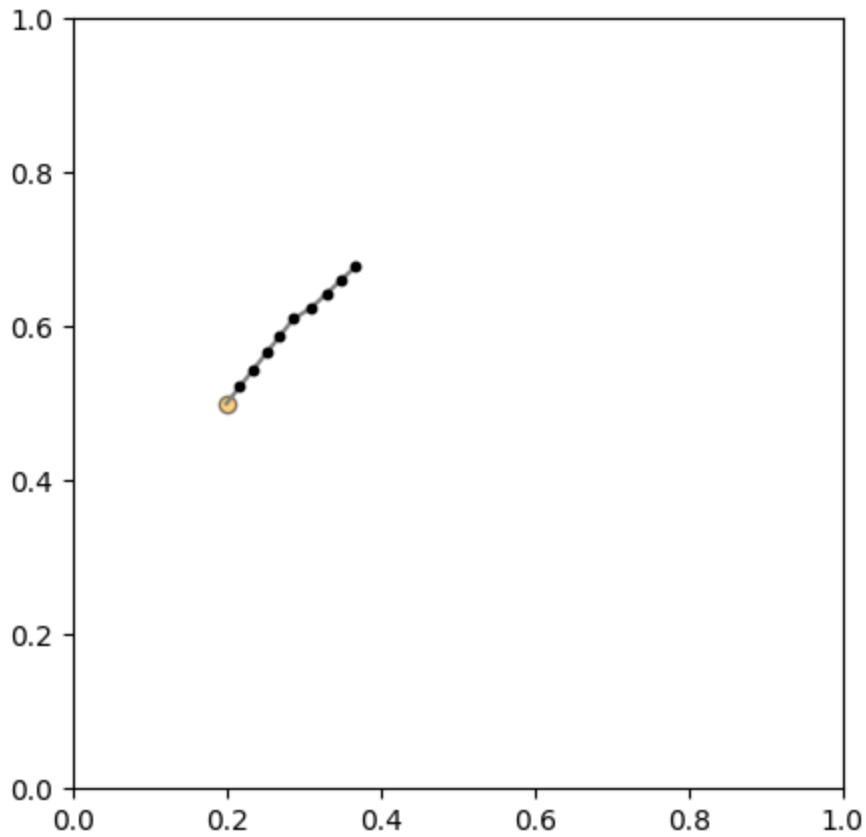
```
plt.ylim(0,1)
```



## 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.



# 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"
```





## 7 Método de Euler hacia atrás.

### Objetivo.

Resolver la ecuación de calor no estacionaria y sin fuentes en 1D usando el Método de Euler hacia atrás (implícito).

[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

```

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

def plot_initial_status(ax, x, u):
    ax.plot(x,[0 for i in x], '-', c='gray', lw=5)#, label='Malla')
    ax.plot(x,u,'r-',lw=2, label='Cond. inicial')
    ax.plot([0,0],[0,-1], 'k--', lw=1.0)
    ax.plot([1,1],[0,1], 'k--', lw=1.0)
    ax.scatter([0,1],[u[0], u[-1]], fc='blue', ec='k', alpha=0.75, label='Cond. c
    ax.grid()

def buildMatrix(N, r):
    # Matriz de ceros
    A = np.zeros((N,N))

    # Primer renglón
    A[0,0] = 1 + 2 * r
    A[0,1] = -r

    # Renglones interiores
    for i in range(1,N-1):
        A[i,i] = 1 + 2 * r
        A[i,i+1] = -r
        A[i,i-1] = -r

    # Último renglón
    A[N-1,N-2] = -r
    A[N-1,N-1] = 1 + 2 * r

    return A

```

```
quizz = Quizz('08', 'notebooks', 'local')
```

## 7.1 Ejercicio 1.

Definir:

- Coordenadas de la malla:  $x$
- Arreglo para la solución final:  $u$
- Valores de  $u$  en la frontera.

```
# Parámetros físicos
L = 1.0 # Longitud del dominio
bA = -1 # Dirichlet en A
bB = 1 # Dirichlet en B
alpha = 1 # Parámetro físico

# Parámetros numéricos
N = 49 # Número de incógnitas
h = L / (N+1) # Tamaño de la malla
ht = 0.0001 # Paso del tiempo
Tmax = 1.0 # Tiempo total de simulación
Nt = int(Tmax / ht) # Número total de pasos
r = ht * alpha / h**2
tolerancia = 1e-6 # Criterio de termino anticipado

# Variables para medir el rendimiento
suma_tiempos = 0.0 # Tiempo total
error = [] # Errores

print(" h = {}, ht = {}, Tmax = {}, Nt = {}, r = {}".format(h, ht, Tmax, Nt, r))

# Preparación de arreglos (malla, solución)
# x = ...
# u = ...
# Condiciones de frontera
# u[0] = ...
# u[N+1] = ...

### BEGIN SOLUTION
# Preparación de arreglos (malla, solución)
x = np.linspace(0,L,N+2) # Coordenadas de la malla
u = np.zeros(N+2) # Arreglo para la solución

# Condiciones de frontera
u[0] = bA
u[N+1] = bB

file_answer = FileAnswer()
file_answer.write('1', x, 'Las coordenadas de la malla están incorrectas.')
```

```
file_answer.write('2', u, 'El arreglo para la solución no está bien definido.')
#### END SOLUTION
```

$h = 0.02$ ,  $ht = 0.0001$ ,  $Tmax = 1.0$ ,  $Nt = 10000$ ,  $r = 0.25$

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

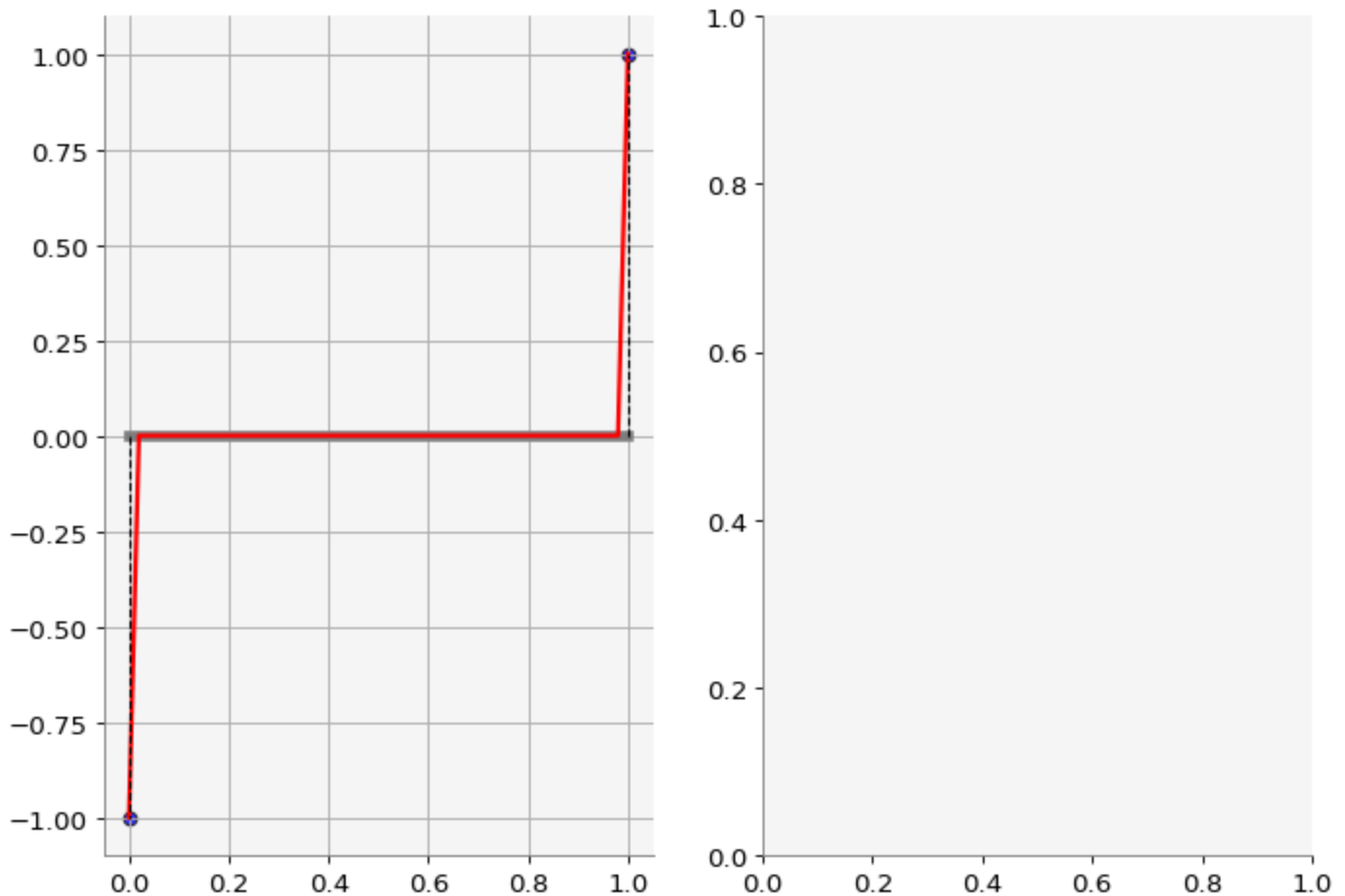
```
quizz.eval_numeric('1',x)
```

1 | Tu resultado es correcto.

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

2 | Tu resultado es correcto.

```
# Visualización de las condiciones iniciales y de frontera
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7))
plot_initial_status(ax1, x, u)
plt.show()
```





## 7.2 Ejercicio 2.

Completar el código con el algoritmos de Euler hacia adelante.

```
# Visualización de las condiciones iniciales y de frontera
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7))
plot_initial_status(ax1, x, u)

# Lado derecho del sistema, contiene la condicion inicial u
f = np.copy(u[1:N+1])
# Copia de la solución para mantener el resultado en el paso previo.
uold = np.copy(u)

# Construcción de la matriz
A = buildMatrix(N,r)

# Ciclo en el tiempo, desde 1 hasta Nt-1
for n in range(1, Nt):
    ### BEGIN SOLUTION
    t1 = time.perf_counter()
    f[0] += r * bA
    f[N-1] += r * bB
    u[1:N+1] = np.linalg.solve(A,f) # Sol. del sistema lineal
    t2 = time.perf_counter()
    suma_tiempos += (t2 - t1)

    e = np.sqrt(h) * np.linalg.norm(uold-u)
    error.append(e)
    ### END SOLUTION

# Graficación cada 25 pasos
if n % 25 == 0:
    ax1.plot(x,u,'-', lw = 1.0, alpha = 0.75, zorder=1)

# Actualizacion de la solucion para dar el siguiente paso
t1 = time.perf_counter()
f = np.copy(u[1:N+1])
uold = np.copy(u)
t2 = time.perf_counter()
suma_tiempos += (t2 - t1)

# Terminación anticipada si se cumple la tolerancia
if e < tolerancia:
    break

file_answer.write('3', error[-1], 'El error no está correctamente calculado.')
file_answer.write('4', n, 'El número de pasos no es el correcto, checka tu algoritmo')
```

```
# Gráfica de resultados
titulo = 'Backward Euler: Error = {:.4e}, Pasos = {:4d}, CPU = {:.5.4} [s]'.format(
fig.suptitle(titulo, fontsize=20)

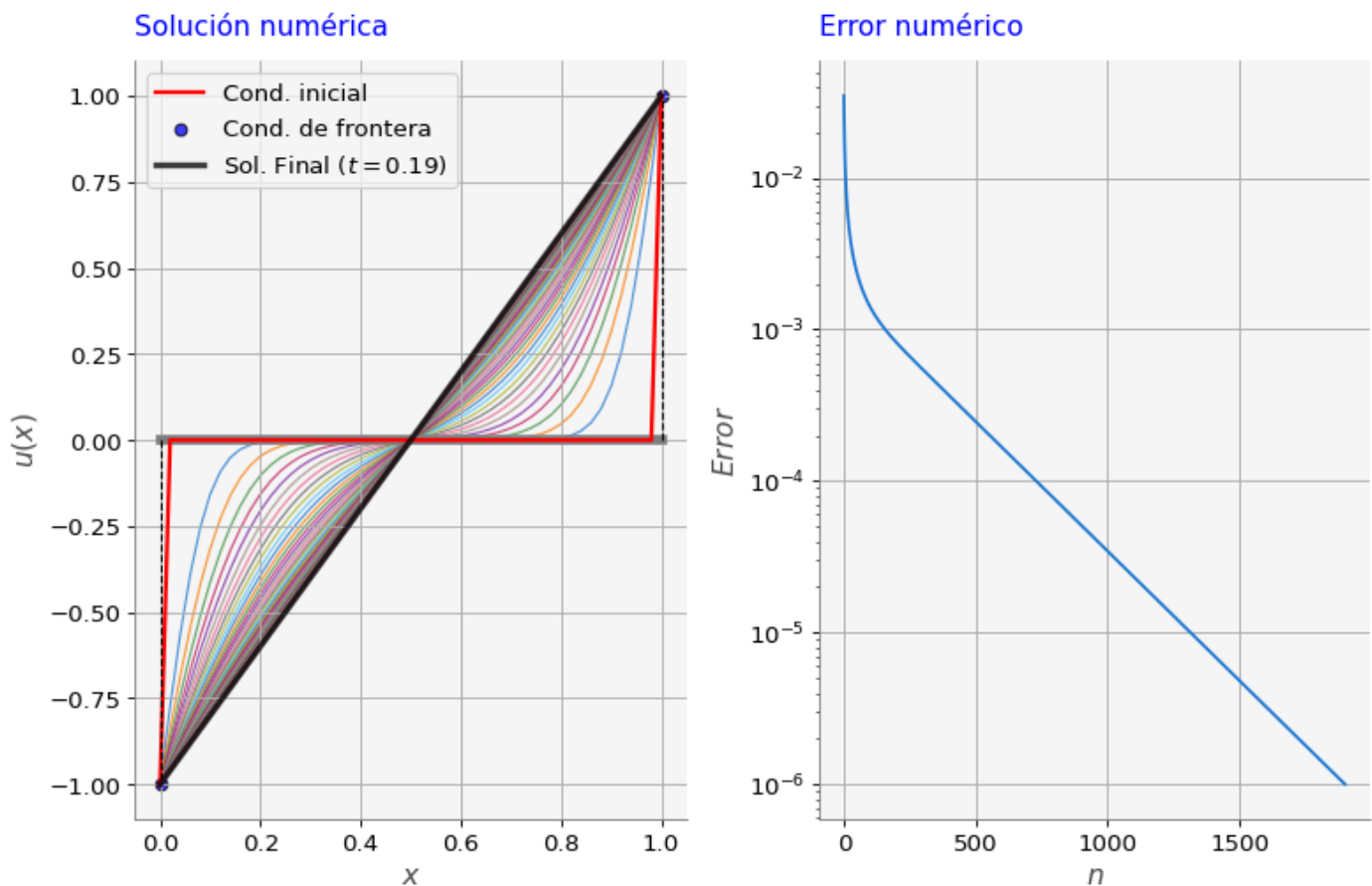
ax1.plot(x,u,'-k',lw=3,alpha=0.75,label='Sol. Final ($t=${:3.2f})'.format(n*ht))
ax1.set_xlabel('$x$')
ax1.set_ylabel('$u(x)$')
ax1.set_title('Solución numérica', color='blue')
ax1.legend()

ax2.plot(error)
ax2.set_yscale('log')
ax2.set_xlabel('$n$')
ax2.set_ylabel('$Error$')
ax2.set_title('Error numérico', color='blue')
ax2.grid()

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

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

**Backward Euler: Error = 9.9989e-07, Pasos = 1901, CPU = 7.784 [s]**



```
quizz.eval_numeric('3',error[-1])
```

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


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

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

## 8 Comparación de método de Euler hacia atrás y hacia adelante.

### Objetivo.

Construir un interactivo que permita comparar en tiempo real los métodos de Euler hacia adelante (explícito) y hacia atrás (implícito).

[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

```
import ipywidgets as widgets
import macti.visual as mvis
from Euler_FB import FEuler, BEuler, FE_vs_BE
```

## 9 Definimos los datos físicos y numéricos

```
# Parámetros físicos
L = 1.0 # Longitud del dominio
bA = -1 # Dirichlet en A
bB = 1 # Dirichlet en B
alpha = 1 # Parámetro físico

# Parámetros numéricos
N = 49 # Número de incógnitas
h = L / (N+1) # Tamaño de la malla
ht = 0.0001 # Paso del tiempo
Tmax = 1.0 # Tiempo total de simulación
Nt = int(Tmax / ht) # Número total de pasos

tolerancia = 1e-6 # Criterio de término anticipado

print(" h = ", h, ", ht = ", ht, ", Tmax = ", Tmax, ", Nt = ", Nt)
```

```
h = 0.02 , ht = 0.0001 , Tmax = 1.0 , Nt = 10000
```

## 10 Método de Euler hacia adelante (Forward)

```
widgets.interact_manual(FEuler,
                        L = widgets.fixed(L),
                        N = widgets.IntSlider(min=10, max = 49, value=49, step=1),
                        alpha = widgets.fixed(alpha),
```

```

bA = widgets.fixed(bA),
bB = widgets.fixed(bB),
Nt = widgets.IntSlider(min=10, max=10000, value=10, step=10),
ht = widgets.FloatSlider(min=1e-4, max=1e-3, value=1e-4, step=1e-4),
tol = widgets.fixed(tolerancia),
compara=widgets.fixed(False),
ax=widgets.fixed(None))

```

```
<function Euler_FB.FEuler(L, N, alpha, bA, bB, Nt, ht, tol, compara=False, ax=None)>
```

## 11 Método de Euler hacia atrás (Backward)

```

widgets.interact_manual(BEuler,
    L = widgets.fixed(L),
    N = widgets.IntSlider(min=10, max = 49, value=49, step=1),
    alpha = widgets.fixed(alpha),
    bA = widgets.fixed(bA),
    bB = widgets.fixed(bB),
    Nt = widgets.IntSlider(min=10, max=10000, value=10, step=10),
    ht = widgets.FloatSlider(min=1e-4, max=1e-3, value=1e-4, step=1e-4),
    tol = widgets.fixed(tolerancia),
    compara=widgets.fixed(False),
    ax=widgets.fixed(None))

```

```
<function Euler_FB.BEuler(L, N, alpha, bA, bB, Nt, ht, tol, compara=False, ax=None)>
```

## 12 Comparación de los métodos de Euler

```

widgets.interact_manual(FE_vs_BE,
    L = widgets.fixed(L),
    N = widgets.IntSlider(min=10, max = 49, value=49, step=1),
    alpha = widgets.fixed(alpha),
    bA = widgets.fixed(bA),
    bB = widgets.fixed(bB),
    Nt = widgets.IntSlider(min=10, max=10000, value=10, step=10),
    ht = widgets.FloatSlider(min=1e-4, max=1e-3, value=1e-4, step=1e-4),
    tol = widgets.fixed(tolerancia))

```

```
<function Euler_FB.FE_vs_BE(L, N, alpha, bA, bB, Nt, ht, tol)>
```

## 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:

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

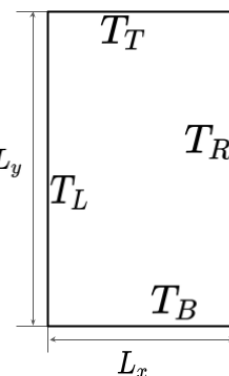
donde  $u$  representa la temperatura en un instante de tiempo  $t$  y en un punto  $(x, y)$  del plano Cartesiano,  $\kappa$  es la conductividad del material y  $S$  una fuente de calor.

### 9.2 Conducción estacionaria en 2D.

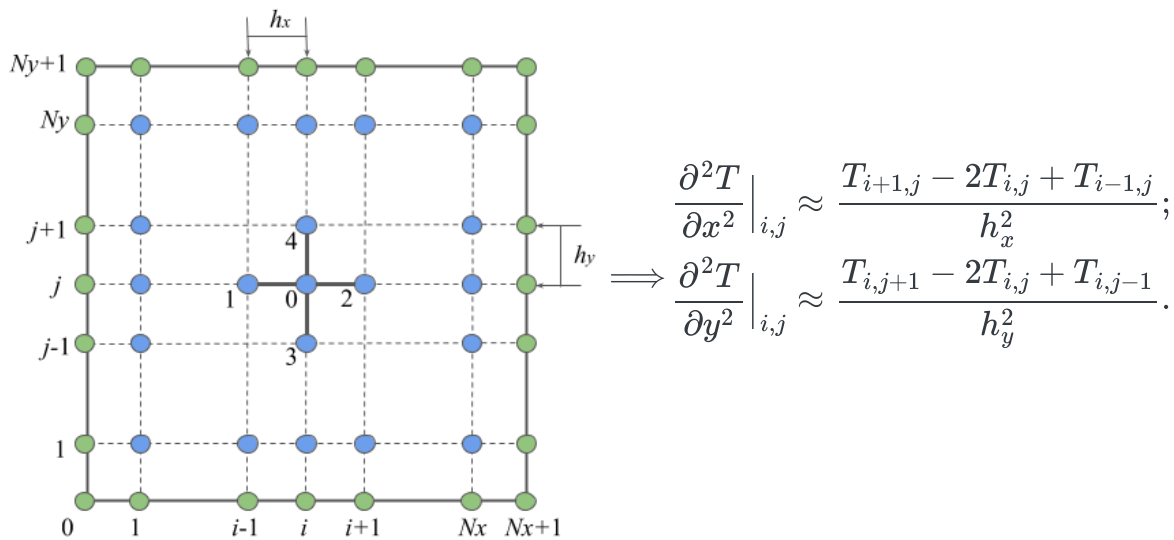
Cuando el problema es estacionario, es decir no hay cambios en el tiempo, y el dominio de estudio es una placa en dos dimensiones, como la que se muestra en la figura, podemos escribir el problema como sigue:

$$-\kappa \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) = S \quad (1)$$

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:

$$\begin{bmatrix}
 -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\
 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\
 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\
 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\
 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & \dots \\
 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & \dots \\
 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & \dots \\
 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & \dots \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & \dots \\
 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & \dots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
 \end{bmatrix}
 \begin{bmatrix}
 u_{1,1} \\
 u_{2,1} \\
 u_{3,1} \\
 u_{4,1} \\
 u_{1,2} \\
 u_{2,2} \\
 u_{3,2} \\
 u_{4,2} \\
 u_{1,3} \\
 u_{2,3} \\
 u_{3,3} \\
 \vdots
 \end{bmatrix}
 =
 \begin{bmatrix}
 f_{1,1} \\
 f_{2,1} \\
 f_{3,1} \\
 f_{4,1} \\
 f_{1,2} \\
 f_{2,2} \\
 f_{3,2} \\
 f_{4,2} \\
 f_{1,3} \\
 f_{2,3} \\
 f_{3,3} \\
 \vdots
 \end{bmatrix}$$

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

```

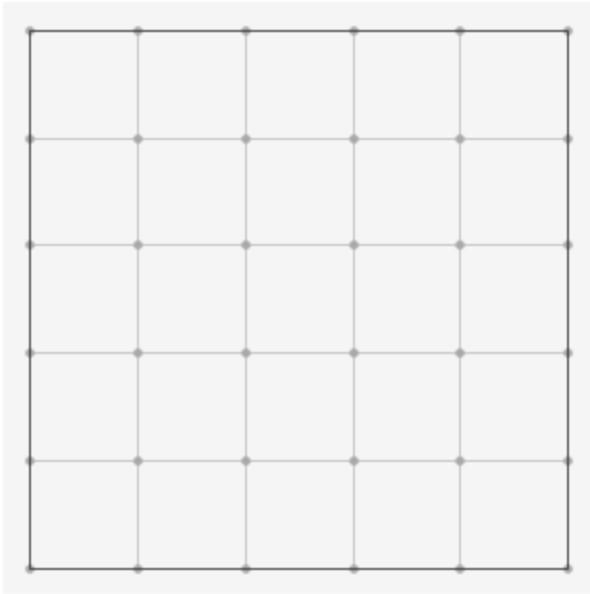
```

fig = plt.figure()
ax = plt.gca()

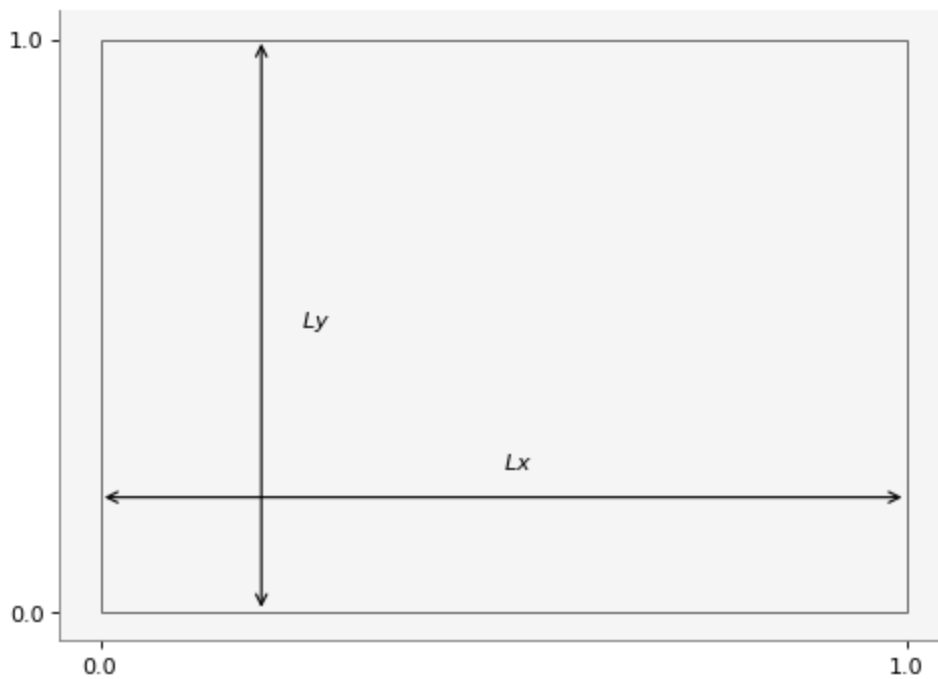
# Ejecutamos la función plot_mesh(...)
plot_mesh(ax, xg, yg)

```

```
# Dibujamos el recuadro con la función plot_fame(...)
plot_frame(ax, xn, yn)
```

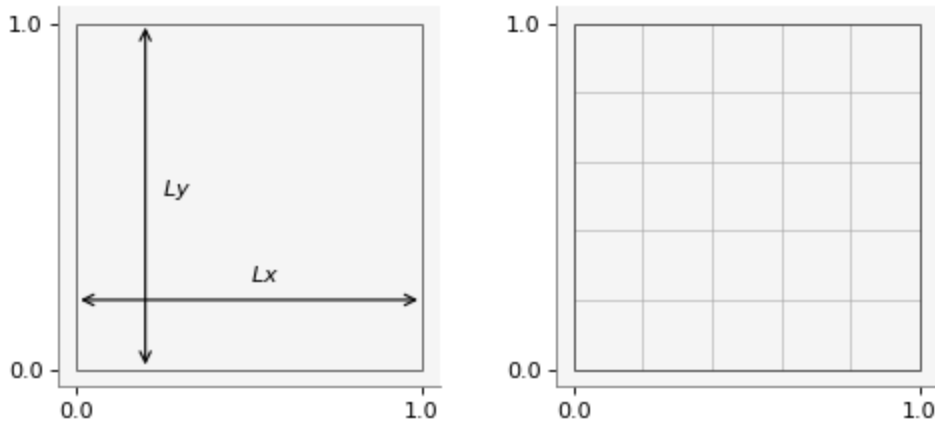


```
vis = mvis.Plotter(1,1)
vis.draw_domain(1, xg, yg)
```



```
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

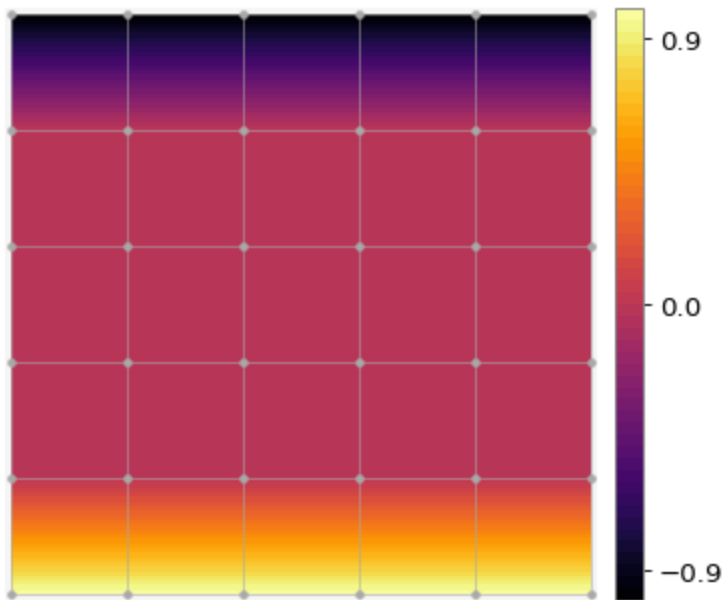
```
fig = plt.figure()
ax = plt.gca()
cax = set_canvas(ax, Lx, Ly)

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()

```

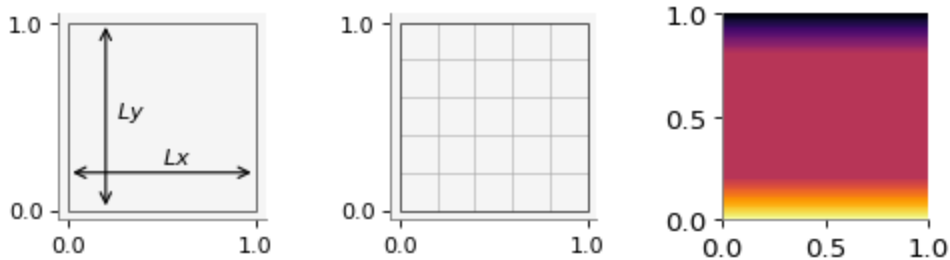


```

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()

```

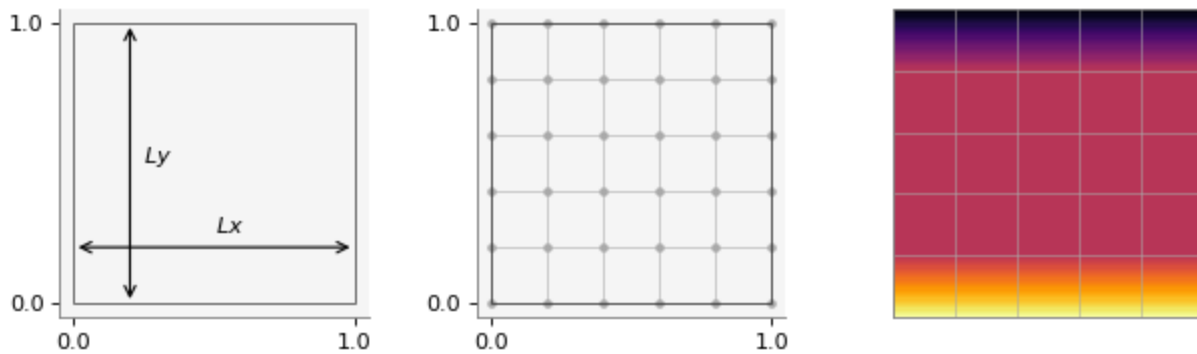


```

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)
```

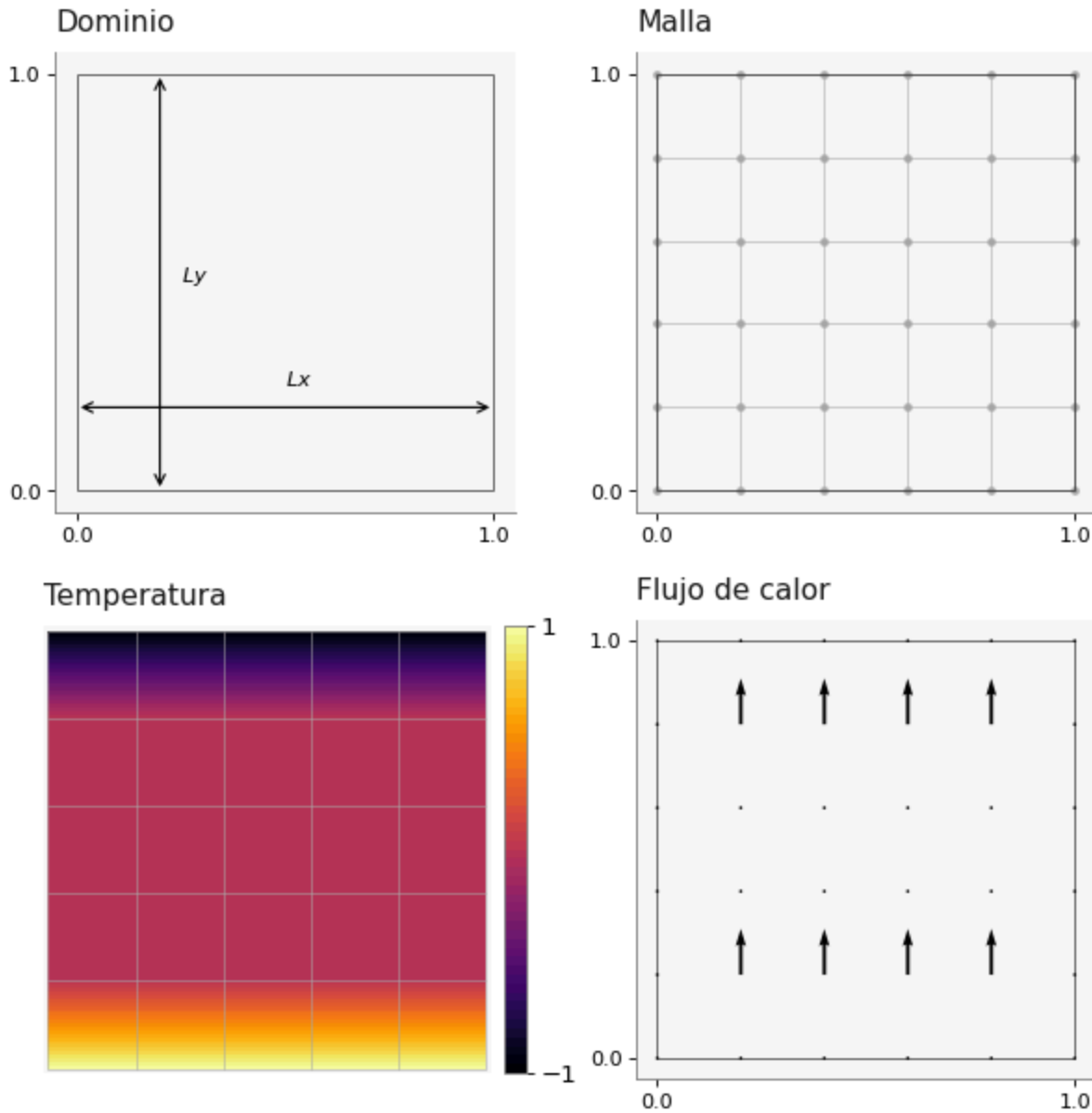
```
ax1 = dict(aspect='equal', title='Dominio')
ax2 = dict(aspect='equal', title='Malla')
ax3 = dict(aspect='equal', title='Temperatura')
ax4 = dict(aspect='equal', title='Flujo de calor')

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., -4.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  1., -4.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1., -4.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1., -4.,  1.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,  1.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,  1.,  0.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,  1.,  0.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,  1.,  0.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,  1.,
         0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,
         1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
        -4.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1., -4.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  1., -4.]])
```

```

    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

```

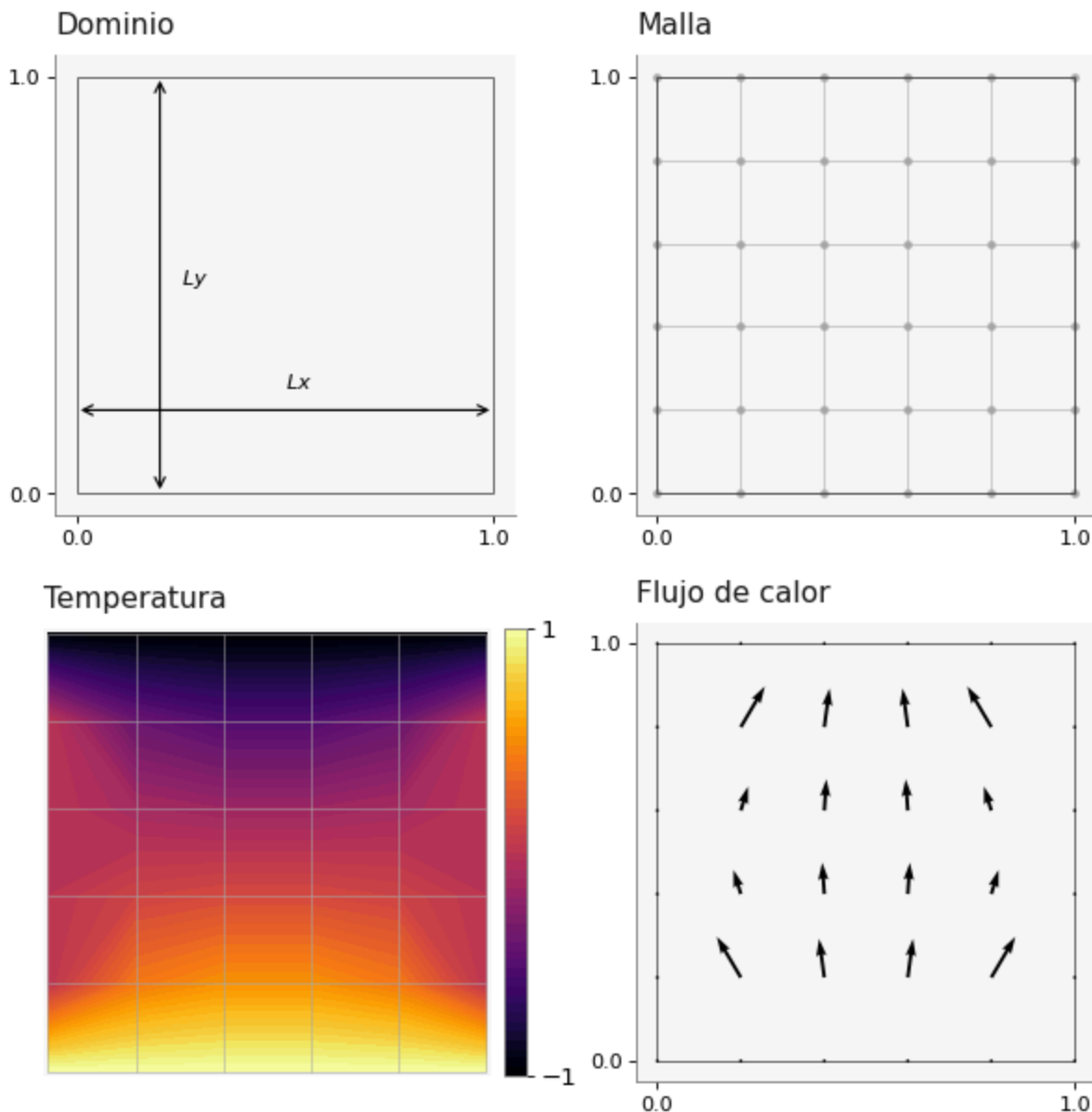
vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
                  dict(figsize=(8,8)))

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
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')

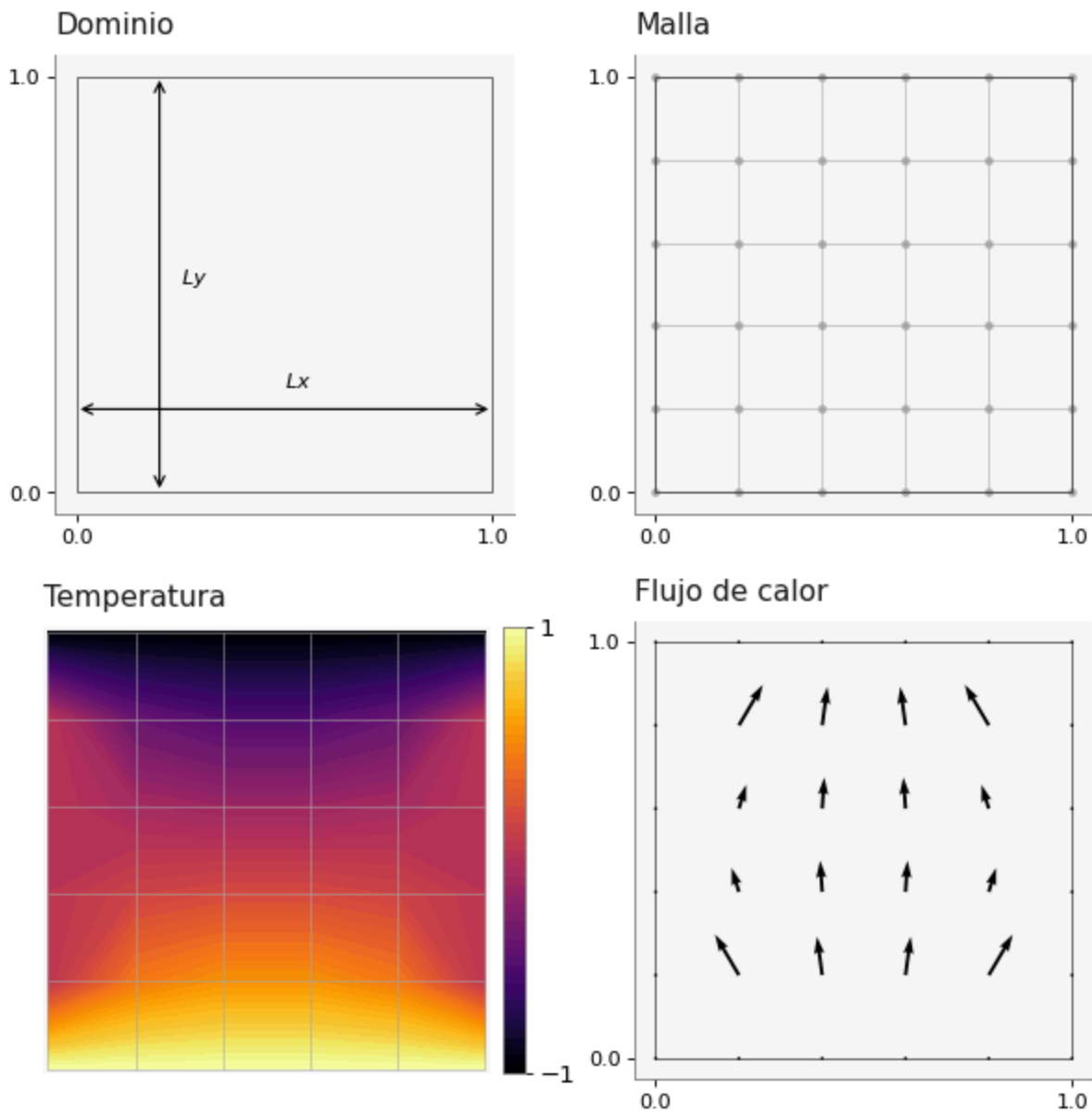
vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
                  dict(figsize=(8,8)))

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)>
```

## 3 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.

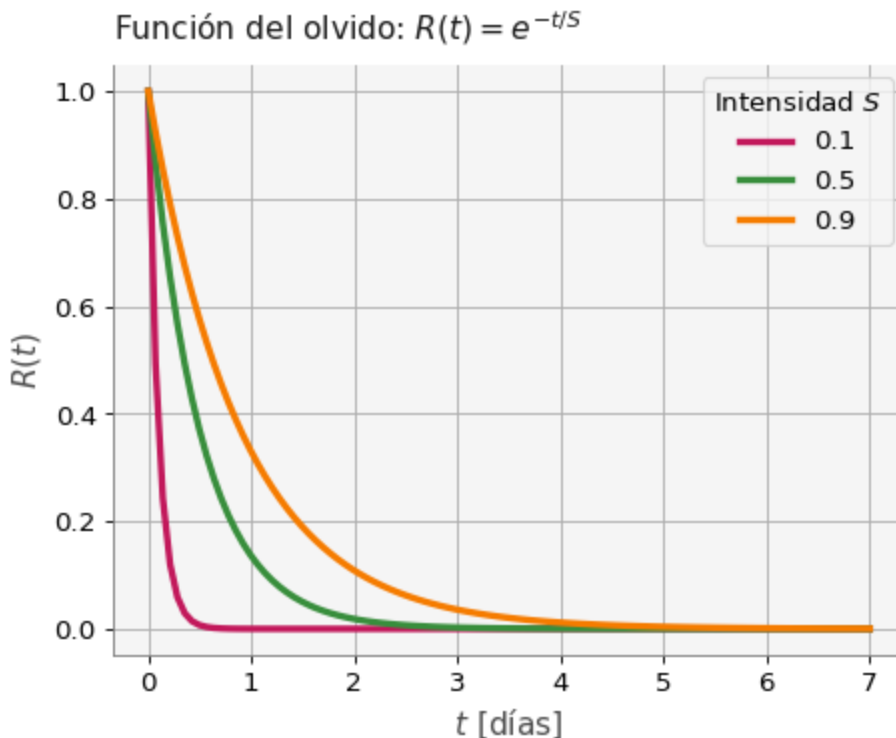
La siguiente gráfica muestra cómo decrecen nuestros recuerdos con el paso del tiempo.

```
# 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:

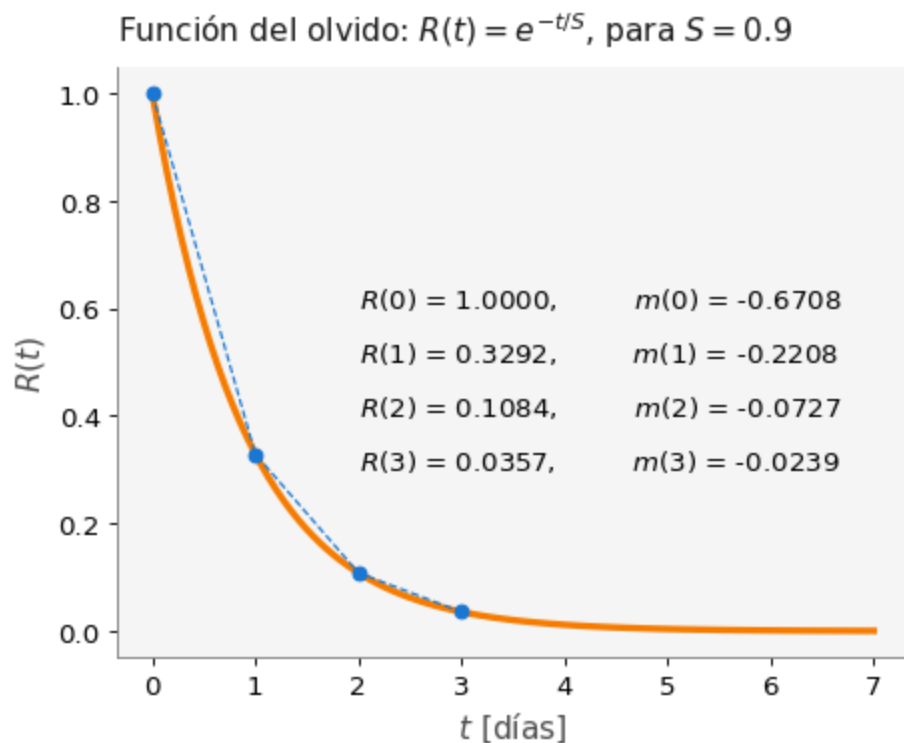
```
# Dominio de tiempo (hasta 7 días).
t = np.linspace(0,7,100)

# 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 \ m(\{2:\}) = \{1:5.4f\}'.format(R(0), m[0]),
plt.text(2,0.5,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \ m(\{2:\}) = \{1:5.4f\}'.format(R(1), m[1]),
plt.text(2,0.4,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \ m(\{2:\}) = \{1:5.4f\}'.format(R(2), m[2]),
plt.text(2,0.3,' $R(\{2:\}) = \{0:5.4f\}, \backslash t \ 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?

### 3.0.1 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.

## 3.0.2 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?

### 3.0.3 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)?**

### 3.0.4 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.  
-----

$$3x^2 - 3.0x + 1.0$$