Variables simbólicas

Las variables `x` o `y` que vamos a definir no son números, ni tampoco pertenecen a los objetos definidos con **NumPy**. Todas las variables simbólicas son objetos de la clase `sp.Symbol` y sus atributos y métodos son completamente diferentes

```
x = sp.Symbol('x')          # define la variable simbólica x
y = sp.Symbol('y')

a, b, c = sp.symbols('a:c') # define como simbólicas las variables a, b, c.


p = sp.Symbol('p', positive = True)  # Natural
q = sp.Symbol('q', real = True)      # Real
```

```
x = sp.Symbol('x', nonnegative = True) # La raíz cuadrada de un número no negativo es real
y = sp.sqrt(x)
print(y.is_real) # La salida de una variable lógica es True o None

y = 4**sp.S(2)
print(y.is_integer)

b = sp.sqrt(2) # √2
print(f"Es b un número racional? {b.is_rational}")


b = 2**sp.S(-2) # 1/2²
print(f"Es b un número entero? {b.is_integer}")
```

Constantes simbólicas y numéricas

Por ejemplo, podemos definir la constante simbólica $1/3$. Si hacemos lo mismo con números representados por defecto en Python, obtendríamos resultados muy diferentes.

```
pi = sp.pi
E = sp.
raiz = sp.sqrt(p)
infinito = sp.oo

frac_simbolico = sp.Rational(1,3) # Número racional de sympy
frac_numerico = 1/3               # Float
```

Manipulación de expresiones

```
expr = (x-3)*(x-3)**2*(y-2)

expr_expandida = sp.expand(expr)    # Expandir

expr_factorizada = sp.factor(expr)  # Factorizar

expr_simplificada = sp.simplify((x**2 - 6*x + 9)/(x-3) - 3) # Simplificar
```

Solución de ecuaciones

El comando `solve` nos permite resolver una ecuación o un sistema de ecuaciones

```
# Ecuación simple / 1º: Expresión | 2º: Valor igualado
sol = sp.solve(sp.Eq(x**2 - 4, 0), x) # x² - 4 = 0

# Sistema de ecuaciones
x, y = sp.symbols('x y')
ec1 = sp.Eq(2*x + y, 1)
ec2 = sp.Eq(x - y, 3)

sol_sistema = sp.solve((ec1, ec2), (x, y))
```

Funciones

El comando `lambda` nos permite el paso de una expresión a una función

```
exprf = x**2+sp.exp(-3*x)+1

f = sp.Lambda((x),exprf)  # se define la función f

exprf.subs({x:3})         # evaluar la expresión
f(3) # evaluar la función
# Definir la función a trozos
```

```
f = sp.Piecewise(
    (x**2, x < -1),
    (x + 1, (x >= -1) & (x <= 2)),
    (sp.sin(x), x > 2)
)

display(f)
```

Dada una expresión en **SymPy** podemos sustituir unas variables simbólicas por otras o reemplazando las variables simbólicas por constantes. Empleamos la función `subs` y los valores a utilizar en la sustitución vienen definidos por un diccionario de Python:

```
expr = x*x + x*y + y*x + y*y

res = expr.subs({x:1, y:2})    # Sustitución de las variables simbólicas por valores
print(f"Valor de la expresión cuando x=1 e y=2: {res}\n")

expr_sub = expr.subs({x:1-y})  # Sustitución de variable simbólica por una expresión
print(f"Sustitución por otra expresión: {expr_sub}")
```

Funciones y lambdify

`lambdify` convierte la función en una función **NumPy** que puede ser evaluada en una matriz de valores x.

```
expr = x**2 + sp.exp(-x)          # Definir función simbólica

f = sp.lambdify(x, expr, 'numpy') # Convertir a función Python/NumPy

# Usar con NumPy
import numpy as np
import matplotlib.pyplot as plt

x_vals = np.linspace(-2, 2, 100)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals)
plt.grid(True)
plt.show()
```

Representación de funciones

```
from sympy.plotting import plot, plot3d

plot(x**2, (x, -3, 3))  # 2D
plot3d(x**2 + y**2, (x, -3, 3), (y, -3, 3))  # 3D
```

Cálculo de dominios

- Para las racionales, calculamos los valores para los que la función no está definida, el resto es el dominio.
- En funciones cuyos dominios son intervalos, calculamos la condición correspondiente, el resultado es el dominio

```
f1 = (x -1) / (x + 1)  # Racional ( Denominador ≠ 0 )

f2 = sp.sqrt(x + 3)    # Radical ( Lo de dentro ≥ 0 )

f3 = sp.log(x + 3)     # Logarítmica ( Lo de dentro > 0)

dom = sp.solve( x+1 , x )      # Valores para los que la función no está definida

dom = sp.solve( x+3 >= 0, x )  # Valores para los que la función está definida

dom = sp.solve( x+3 > 0, x )   # Valores para los que la función está definida
```

Simetría

Opción 1

Calculamos $f(-x)$ y $-f(-x)$ y comprobamos si es igual a $f(x)$

```
f = x**2   # f(x)

f2 = funcs[f].subs({x: -x})  # f(-x)

if f == f2:       # f(x) = f(-x)
    print("Simetría par")
elif f == -f2:    # f(x) = -f(-x)
    print("Simetría impar")
```

Opción 2

Comprobamos $f(-x) - f(x) = 0\$||\$f(-x) + f(x) = 0$

```
if f2 - f == 0: # f(-x) - f(x) = 0
    print("Simetría par")
elif f2 + f == 0: # f(-x) + f(x) = 0
    print("Simetría impar")
```

Composición de funciones

$f \circ g = f(g(x))$
$g \circ f = g(f(x))$
$Dom(f \circ g) = \{x \in Dom(f) / f(x) \in Dom(g)\}$

```
# Definimos las expresiones
f = (6*x) / (x**2 - 9)
g = sp.sqrt(3*x)

# Las transformamos en funciones para facilitar la composición
f = sp.Lambda(x , f)
g = sp.Lambda(x , g)

# Las componemos | f ◦ g
c1 = f(g(12))
c2 = f(g(x))

# Dominio:  1º Obtenemos la expresión compuesta | 2º Calculamos el dominio de esa función
```

Límites

```
x = sp.Symbol('x')          # Variable simbólica x

f = x**2                    # Expresión

a = -3                      # Punto en el que se calcula el límite

lim = sp.limit(f, x, a)     # Límite de la expresión cuando x se aproxima al punto a
```

Límites laterales

```
f1 = 1/x

limi = sp.limit(f1, x, 0,'+')   # Límite de f1 en a=0 por la DERECHA (+)

limd = sp.limit(f1, x, 0,'-')   # Límite de f1 en a=0 por la IZQUIERDA (-)

lim= sp.limit(f1, x, 0)         # Si no especificamos el lado, lo calcula por la DERECHA
```

Asíntotas

- La asíntota horizontal indica que la función se acerca a un **valor constante** cuando $x\to\infty$ o $x\to-\infty$.

$y = L\ es\ asíntota\ horizontal\ si\ \lim_{x\to\infty} f(x) = L\ o\ \lim_{x\to-\infty} f(x) = L$

- Una asíntota vertical ocurre cuando la función se acerca a **infinito** (o menos infinito) al acercarse a cierto valor x=a.

$\lim_{x\to-a} f(x) = \pm\infty\ \ y\ \lim_{x\to a} f(x) = \pm\infty$

- Asíntota oblicua es la recta con una **pendiente diferente de cero** a la que una función se aproxima indefinidamente cuando x tiende a infinito o menos infinito
  - Una función racional f(x) = $P(x)/Q(x)$ tiene una asíntota oblicua si y solo si el grado del **numerador (P)** es **una unidad mayor** que el **denominador (Q)**
  - Si existe una **asíntota horizontal, no habrá asíntota oblicua**

Es de la forma: $ y = mx +n$

1. $m = \lim_{x\to\pm\infty}[f(x)/x]$
2. $n = \lim_{x\to\pm\infty}[f(x) - mx]$

```
f2 = x*x/(x + 1)
```

Asíntota horizontal

```
ahd = sp.limit(f2, x, oo)
ahi = sp.limit(f2, x, -oo)
print('Límite en +oo =', ahd)
print('Límite en -oo =', ahi)

# Como los límites no son un valor constante ( 1 , 5 ...), no existe asíntota horizontal
```

Asíntota vertical

Comprobamos asíntotas verticales en los puntos para los que no está definida la función

```
avd = sp.limit(f2, x, -1,'+')
avi = sp.limit(f2, x, -1,'-')
print('Limite en a=-1 por la derecha = ',avd)
print('Limite en a=-1 por la izquierda = ',avi)

# Como ambos límites dan ±∞ , existe asíntota vertical en ese punto ( -1 )
```

Asíntota oblicua

```
aomd = sp.limit(f2/x, x, oo)          # Pendiente de la asíntota oblicua por la derecha
aond = sp.limit(f2 - aomd*x, x, oo)   # n de la asíntota oblicua por la derecha

ao = aomd*x + aond
```

```
aomi = sp.limit(f2/x, x, -oo)         # Pendiente de la asíntota oblicua por la izquierda
print('aomi = ',aomi)

aoni = sp.limit(f2 - aomi*x, x, -oo)  # n de la asíntota oblicua por la derecha
print('aoni = ',aoni)

# Como ambas m y n son iguales, hay una sola asíntota oblicua ( y = x -1 )
```

Derivadas

```
f = sp.Lambda((x), x**2 + sp.exp(-3*x) + 1) # Definimos la función

df = sp.diff(f(x), x) # Derivada de f con respecto a x

display(df)
df2 = sp.diff(f(x), x , 2) # Podemos calcular la derivada enésima

display(df2)
```

Tangente a una curva en un punto

La ecuación de la recta tangente a una curva $y = f(x)$ en el punto $(a, f(a))$ está dada por

$$y = f(a) + f'(a)(x - a).$$

A continuación, calculamos la recta tangente a la curva $y = x^3$ en el punto $a = 2$:

```
g = sp.Lambda((x),x**3) # g(x)=x³
dg = sp.diff(g(x),x) # g'(x)

r = g(2) + dg.subs({x:2}) * (x-2) # Recta tangente a g en el punto a=2

display(r)
```

Podemos emplear la recta tangente para aproximar valores de la función en puntos próximos al punto de desarrollo.

```
valor = g(2.1)

apr = r.subs({x:2.1})

print("g(2.1) = ", valor)
print("Aproximación: ", apr)
```

Cálculo de errores:

- Error absoluto:

$$E_a = |V_{real} - V_{medido}|$$

- Error relativo:

$$E_r = \frac{E_a}{V_{real}}$$

```
Eabs = abs(valor - apr) # Error absoluto

Erel = Eabs / valor     # Error relativo
```

Extremos y monotonía de una función

- Calcula la primera derivada de la función: $f'(x)$.
- Igualar la primera derivada a cero: $f'(x) = 0$ y resuelve la ecuación para encontrar los valores de x que la satisfacen.
- Calcula la segunda derivada de la función: $f''(x)$ y clasificar los posibles extremos según el signo

Si $f''(x) < 0$, es un **máximo** local.

Si $f''(x) > 0$, es un **mínimo** local.

Si $f''(x) = 0$, la prueba de la segunda derivada no es concluyente y se debe usar otro método (como el análisis del signo de la primera derivada).

¡¡ También hay que tener en cuenta los puntos donde la función no es derivable !!

```
f2 = x**3 -3*x**2 + 2

df = sp.diff(f2,x) # Calculamos la primera derivada

extr = sp.solve(df,x) # Resolvemos la ecuación f'(x)= 0 para determinar los POSIBLES extremos

df2 = sp.diff(f2,x,2) # Calculamos la segunda derivada

for e in extr:  # Calculamos el valor de la segunda derivada en cada punto y verificamos con el
    valor = df2.subs({x:e})
    if  valor > 0:
        print(f"x = {e} mínimo local de f(x)")
    elif valor < 0:
        print(f"x = {e} máximo local de f(x)")
    elif valor == 0:
        print("No es posible clasificar el extremo ( f''(x) = 0 )")
```

Monotonía (Crecimiento y decrecimiento)

```
g = (x+1)/(x-1)

dg = sp.diff(g , x)

# Solveset evalua mejor las inecuaciones, devuelve conjuntos ( COnjunto vacío, Reales, etc...)
crec = sp.solveset( dg > 0, x, domain=sp.Reals)
decrec = sp.solveset( dg < 0 , x, domain=sp.Reals)

display("Función creciente en: ", crec)  # Crecimiento = ∅ ( No crece )
display("Función decreciente en: ", decrec)
```

Concavidad y convexidad

Evaluamos el signo de la segunda derivada

- Si $f''(x) > 0$ es convexa
- Si $f''(x) < 0$ es cóncava.

```
h = 3 / (1 + x)

d2h = sp.diff(h, x, 2)
display("Segunda derivada:", d2h)

convexa = sp.solveset(d2h > 0, x, domain=sp.Reals)
concava = sp.solveset(d2h < 0, x, domain=sp.Reals)

display("Función convexa en:", convexa)
display("Función cóncava en:", concava)
t = x**3 - 4*x - 3

dt2 = sp.diff(t,x,2) # Segunda derivada
a = sp.solve(dt2,x)  # Resolvemos f''(x) = 0 -> Posibles puntos de inflexión

# Para comprobarlo tenemos que verificar el cambio de concavidad
```