Cálculo Simbólico con SymPy

SymPy es una biblioteca de Python para cálculo simbólico. Su objetivo es convertirse en un sistema de álgebra computacional (CAS) con todas las funciones, manteniendo el código lo más simple posible para que sea comprensible y fácilmente extensible. SymPy está escrito completamente en Python y usa Python como lenguaje de programación. SymPy solo depende de mpmath, una biblioteca de Python para aritmética en punto flotante de longitud arbitraria, lo que facilita su uso. SymPy puede integrarse en otras aplicaciones y ampliarse con funciones personalizadas.

SymPy define tres tipos numéricos: Real, Racional y Entero.

La clase Racional representa un número racional como un par formado por dos enteros: el numerador y el denominador, por lo que Racional(1,2) representa 1/2, Racional(5,2) 5/2 y así sucesivamente:

In [1]:

```
import sympy
```

In [2]:

```
a = sympy.Rational(1,2)
a
```

Out[2]:

$$\frac{1}{2}$$

In [3]:

```
a*2
```

Out[3]:

$$1$$

In [4]:

```
# Si hicieramos ese cálculo en la forma habitual
x=1/2
x
```

Out[4]:

```
0.5
```

In [5]:

```
x*2
```

Out[5]:

```
1.0
```

In [6]:

```
#
import math
math.sqrt(9)
```

Out[6]:

```
3.0
```

In [7]:

```
math.sqrt(8)
```

Out[7]:

```
2.8284271247461903
```

In [8]:

```
sympy.sqrt(9)
```

Out[8]:

3

In [9]:

```
sympy.sqrt(8)
```

Out[9]:

$2\sqrt{2}$

In [10]:

```
sympy.sqrt(3)
```

Out[10]:

$\sqrt{3}$

SymPy usa mpmath en segundo plano, lo que hace posible realizar cálculos usando aritmética de precisión arbitraria. De esa forma, algunas constantes especiales, como e, π, ∞, se tratan como símbolos y se pueden evaluar con precisión arbitraria:

In [12]:

```
from sympy import pi
pi**2
```

Out[12]:

π^2

In [13]:

```
pi.evalf()
```

Out[13]:

3.14159265358979

In [15]:

```
from sympy import exp

(pi+exp(1)).evalf()
```

Out[15]:

5.85987448204884

Como se puede ver, evalf evalúa la expresión devolviendo un número de punto flotante.

También hay una clase que representa el infinito matemático, llamada Infinity

In [16]:

```
from sympy import oo, exp, limit, Symbol
1 + oo
```

Out[16]:

∞

In [17]:

```
oo > 99999
```

Out[17]:

True

In [18]:

```
oo + 1
```

Out[18]:

∞

In [19]:

```
42/oo
```

Out[19]:

0

En SymPy las variables se definen mediante símbolos. A diferencia de otros sistemas de manipulación simbólica, las variables en SymPy deben definirse antes de usarse.

Definamos una expresión simbólica, representando la expresión matemática

In [20]:

```
x = Symbol('x')
limit(exp(x), x, oo)
```

Out[20]:

∞

In [21]:

```
y = Symbol('y')
x+y+x-y
```

Out[21]:

$2x$

In [22]:

```
(x+y)**2
```

Out[22]:

$(x + y)^2$

In [23]:

```
# symbols permite definir varias variables a la vez

from sympy import symbols
x, y = symbols('x y')
expr = x + 2*y
expr
```

Out[23]:

$x + 2y$

Observe que escribimos $x + 2 * y$ tal como lo haríamos si x e y fueran variables ordinarias de Python. Pero en este caso, en lugar de evaluar algo, la expresión sigue siendo $x + 2 * y$.

In [24]:

```
expr + 1
```

Out[24]:

$x + 2y + 1$

In [25]:

```
expr - x
```

Out[25]:

$2y$

In [26]:

```
x*expr
```

Out[26]:

$x\left(x + 2y\right)$

Aquí, podríamos esperar que $x(x + 2y)$ se transformara en $x2 + 2xy$, pero en cambio vemos que la expresión se quedó como estaba. Este es un tema común en SymPy. Aparte de simplificaciones obvias como $x - x = 0$ y $\sqrt{8} = 2\sqrt{2}$, la mayoría de las simplificaciones no se realizan automáticamente. Esto se debe a que podríamos preferir la forma factorizada, o podríamos preferir la forma expandida. Ambas formas son útiles en diferentes circunstancias. En SymPy hay funciones para pasar de un formato al otro

In [27]:

```
from sympy import expand, factor
expanded_expr = expand(x*expr)
expanded_expr
```

Out[27]:

$x^2 + 2xy$

In [28]:

```
factor(expanded_expr)
```

Out[28]:

$x\left(x + 2y\right)$

In [29]:

```
expand((x+y)**3)
```

Out[29]:

$x^3 + 3x^2y + 3xy^2 + y^3$

In [30]:

```
expand(x+y, complex=True)
```

Out[30]:

$$\mathrm{re}\,(x) + \mathrm{re}\,(y) + i\,\mathrm{im}\,(x) + i\,\mathrm{im}\,(y)$$

In [32]:

```
from sympy import cos

expand(cos(x+y), trig=True)
```

Out[32]:

$$-\sin\,(x)\sin\,(y) + \cos\,(x)\cos\,(y)$$

In [33]:

```
from sympy import simplify

simplify((x+x*y)/x)
```

Out[33]:

$$y + 1$$

El poder real de un sistema de cálculo simbólico como SymPy es la capacidad de realizar todo tipo de cálculos simbólicamente. SymPy puede simplificar expresiones, calcular derivadas, integrales y límites, resolver ecuaciones, trabajar con matrices y mucho más y hacerlo todo simbólicamente. Incluye módulos para graficar, imprimir (como resultados impresos bonitos en 2D de fórmulas matemáticas o $\LaTeX$), generación de código, física, estadística, combinatoria, teoría de números, geometría, lógica y más.

In [34]:

```
x, t, z, nu = symbols('x t z nu')
```

In [36]:

```
from sympy import init_printing

# pretty print with unicode characters.
init_printing(use_unicode=True)
```

Calculemos la derivada de $sin(x)e^x$

In [39]:

```
from sympy import diff, sin

diff(sin(x)*exp(x), x)
```

Out[39]:

$$e^{x}\,\sin\,(x) + e^{x}\,\cos\,(x)$$

In [40]:

```
diff(sin(x), x)
```

Out[40]:

$\cos(x)$

In [41]:

```
diff(sin(2*x), x)
```

Out[41]:

$2\cos(2x)$

In [43]:

```
from sympy import tan

diff(tan(x), x)
```

Out[43]:

$\tan^2(x) + 1$

Las derivadas de orden superior se pueden calcular utilizando el método diff(func, var, n):

In [44]:

```
diff(sin(2*x), x, 2)
```

Out[44]:

$-4\sin(2x)$

In [45]:

```
diff(sin(2*x), x, 3)
```

Out[45]:

$-8\cos(2x)$

Para calcular $\int (e^x sin(x) + e^x cos(x))\, dx$

In [47]:

```
from sympy import integrate

integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
```

Out[47]:

$e^x \sin(x)$

Para calcular $\int_{\infty}^{\infty} sin(x^2)\, dx$

In [48]:

```
integrate(sin(x**2), (x, -oo, oo))
```

Out[48]:

$$\frac{\sqrt{2}\sqrt{\pi}}{2}$$

In [49]:

```
integrate(sin(x), (x, 0, pi/2))
```

Out[49]:

$$1$$

In [50]:

```
integrate(cos(x), (x, -pi/2, pi/2))
```

Out[50]:

$$2$$

In [51]:

```
integrate(exp(-x), (x, 0, oo))
```

Out[51]:

$$1$$

In [52]:

```
integrate(exp(-x**2), (x, -oo, oo))
```

Out[52]:

$$\sqrt{\pi}$$

Los límites son fáciles de utilizar en SymPy utilizando la sintaxis

```
límite(función, variable, punto),
```

por lo que, para calcular el límite de f (x) como x -> 0, se escribe límite (f, x, 0):

In [53]:

```
limit(sin(x)/x, x, 0)
```

Out[53]:

$$1$$

In [54]:

```
limit(1/x, x, oo)
```

Out[54]:

$$0$$

SymPy también sabe cómo calcular la serie de Taylor de una expresión en un punto utilizando

```
series (expr, var)
```

In [56]:

```
from sympy import series

series(cos(x), x)
```

Out[56]:

$$1 - \frac{x^2}{2} + \frac{x^4}{24} + O\left(x^6\right)$$

In [57]:

```
series(1/cos(x), x)
```

Out[57]:

$$1 + \frac{x^2}{2} + \frac{5x^4}{24} + O\left(x^6\right)$$

SymPy es capaz de resolver ecuaciones algebraicas, en una o varias variables.

Ejemplo: resolver $x^2 - 2 = 0$

In [59]:

```
from sympy import solve

solve(x**2 - 2, x)
```

Out[59]:

$$\left[-\sqrt{2},\ \sqrt{2}\right]$$

In [60]:

```
solve(x**4 - 1, x)
```

Out[60]:

$$[-1,\ 1,\ -i,\ i]$$

Como se puede ver, el primer argumento es una expresión que se supone que debe ser igual a 0. Es capaz de resolver una gran parte de las ecuaciones polinomiales, y también es capaz de resolver múltiples

Resolver la ecuación diferencial $y'' - y' = e^t$

In [65]:

```
from sympy import dsolve, Function, Eq

y = Function('y')
dsolve(Eq(y(t).diff(t, t) - y(t), exp(t)), y(t))
```

Out[65]:

$$y(t) = C_2 e^{-t} + \left(C_1 + \frac{t}{2}\right) e^t$$

Bibliografía

- SymPy (https://www.sympy.org/en/index.html)
- SymPy Tutorial (https://docs.sympy.org/latest/tutorials/intro-tutorial/index.html)