Capítulo 4: Álgebra y matemáticas simbólicas con SymPy

### **Resolución de ecuaciones**

La función solve() de SymPy puede utilizarse para encontrar soluciones a ecuaciones. Cuando introduces una expresión con un símbolo que representa una variable, como *x*, solve() calcula el valor de ese símbolo. Esta función siempre realiza su cálculo suponiendo que la expresión que introduces es igual a cero, es decir, imprime el valor que, al sustituir al símbolo, hace que toda la expresión sea igual a cero. Empecemos con la sencilla ecuación *x* - 5 = 7. Si queremos utilizar solve() para hallar el valor de x, primero tenemos que hacer que uno de los lados de la ecuación sea igual a cero*(x* - 5 - 7 = 0). Entonces, estamos preparados para utilizar solve(), como se indica a continuación:

>>> from sympy import Symbol, solve  
>>> x = Symbol('x')  
>>> expr = x - 5 - 7  
>>> solve(expr)  
[12]

Cuando utilizamos solve(), calcula el valor de 'x' como 12 porque ése es el valor que hace que la expresión*(x* - 5 - 7) sea igual a cero.

Observa que el resultado 12 se devuelve en una lista. Una ecuación puede tener varias soluciones; por ejemplo, una ecuación cuadrática tiene dos soluciones. En ese caso, la lista tendrá todas las soluciones como miembros. También puedes pedir a la función solve() que devuelva el resultado de forma que cada miembro sea un diccionario. Cada diccionario está compuesto por el símbolo (nombre de la variable) y su valor (la solución). Esto es especialmente útil cuando resolvemos ecuaciones simultáneas en las que tenemos más de una variable que resolver, porque cuando la solución se devuelve como diccionario, sabemos qué solución corresponde a cada variable.

#### ***Resolver ecuaciones cuadráticas***

En el [Capítulo 1](ch01.html#ch01), encontramos las raíces de la ecuación cuadrática ax2 + *bx* + *c* = 0 escribiendo las fórmulas de las dos raíces y sustituyendo después los valores de las constantes *a*, *b* y *c*. Ahora aprenderemos a utilizar la función solve() de SymPy para encontrar las raíces sin necesidad de escribir las fórmulas. Veamos un ejemplo:

➊ >>> from sympy import solve  
>>> x = Symbol('x')  
➋ >>> expr = x\*\*2 + 5\*x + 4  
➌ >>> solve(expr, dict=True)  
➍ [{x: -4}, {x: -1}]

Primero importamos la función solve() en ➊. A continuación, definimos un símbolo, x, y una expresión correspondiente a la ecuación cuadrática, x\*\*2 + 5\*x + 4, en ➋. A continuación, llamamos a la función solve() con la expresión anterior en ➌. El segundo argumento de la función solve() (dict=True) especifica que queremos que el resultado se devuelva como una lista de diccionarios Python.

Cada solución de la lista devuelta es un diccionario que utiliza el símbolo como clave emparejado con su valor correspondiente. Si la solución está vacía, se devuelve una lista vacía. Las raíces de la ecuación anterior son -4 y -1, como puedes ver en ➍.

En el primer capítulo descubrimos que las raíces de la ecuación

x2 + *x* + 1 = 0

son números complejos. Intentemos hallarlas utilizando solve():

>>> x=Symbol('x')  
>>> expr = x\*\*2 + x + 1  
>>> solve(expr, dict=True)  
[{x: -1/2 - sqrt(3)\*I/2}, {x: -1/2 + sqrt(3)\*I/2}]

Ambas raíces son imaginarias, como era de esperar, con la componente imaginaria indicada por el símbolo I.

#### ***Resolución de una variable en función de otras***

Además de hallar las raíces de las ecuaciones, podemos aprovechar la matemática simbólica para utilizar la función solve() para expresar una variable de una ecuación en términos de las demás. Veamos cómo encontrar las raíces de la ecuación cuadrática genérica ax2 + *bx* + *c* = 0. Para ello, definiremos *x* y tres símbolos *adicionales-a*, *b* y *c*, que corresponden a las tres constantes:

>>> x = Symbol('x')  
>>> a = Symbol('a')  
>>> b = Symbol('b')  
>>> c = Symbol('c')

A continuación, escribimos la expresión correspondiente a la ecuación y utilizamos sobre ella la función solve():

>>> expr = a\*x\*x + b\*x + c  
>>> solve(expr, x, dict=True)  
[{x: (-b + sqrt(-4\*a\*c + b\*\*2))/(2\*a)}, {x: -(b + sqrt(-4\*a\*c + b\*\*2))/(2\*a)}]

Aquí, tenemos que incluir un argumento adicional, x, a la función solve(). Como hay más de un símbolo en la ecuación, tenemos que decirle a solve() qué símbolo debe resolver, que es lo que indicamos pasando x como segundo argumento. Como era de esperar, solve() imprime la fórmula cuadrática: la fórmula genérica para encontrar el valor o valores de *x* en una expresión polinómica.

Para que quede claro, cuando utilizamos solve() en una ecuación con más de un símbolo, especificamos el símbolo a resolver como segundo argumento (y ahora el tercer argumento especifica cómo queremos que se devuelvan los resultados).

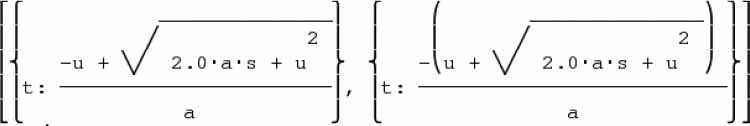
A continuación, veamos un ejemplo de física. Según una de las ecuaciones del movimiento, la distancia recorrida por un cuerpo que se mueve con una aceleración constante *a*, con una velocidad inicial *u*, en un tiempo *t*, viene dada por

image

Sin embargo, dados *u* y *a*, si quisieras hallar el tiempo necesario para recorrer una distancia dada, *s*, tendrías que expresar primero *t* en términos de las otras variables. He aquí cómo podrías hacerlo utilizando la función solve() de SymPy:

>>> from sympy import Symbol, solve, pprint  
>>> s = Symbol('s')  
>>> u = Symbol('u')  
>>> t = Symbol('t')  
>>> a = Symbol('a')  
>>> expr = u\*t + (1/2)\*a\*t\*t - s  
>>> t\_expr = solve(expr,t, dict=True)  
>>> pprint(t\_expr)

El resultado tiene este aspecto:



Ahora que tenemos la expresión para *t* (a la que se refiere la etiqueta t\_expr), podemos utilizar el método subs() para sustituir los valores de *s*, *u* y *a* y encontrar los dos valores posibles de *t*.

#### ***Resolución de un sistema de ecuaciones lineales***

Considera las dos ecuaciones siguientes

*2x* + *3y* = 6

*3x* + *2y* = 12

Supongamos que queremos encontrar el par de valores*(x*, *y*) que satisface las dos ecuaciones. Podemos utilizar la función solve() para encontrar la solución de un sistema de ecuaciones como éste.

En primer lugar, definimos los dos símbolos y creamos las dos ecuaciones:

>>> x = Symbol('x')  
>>> y = Symbol('y')  
>>> expr1 = 2\*x + 3\*y - 6  
>>> expr2 = 3\*x + 2\*y – 12

Las dos ecuaciones están definidas por las expresiones expr1 y expr2, respectivamente. Observa cómo hemos reordenado las expresiones para que ambas sean iguales a cero (hemos desplazado el lado derecho de las ecuaciones dadas al lado izquierdo). Para encontrar la solución, llamamos a la función solve() con las dos expresiones formando una tupla:

>>> solve((expr1, expr2), dict=True)  
[{y: -6/5, x: 24/5}]

Como he dicho antes, obtener la solución en forma de diccionario es útil en este caso. Podemos ver que el valor de x es 24/5 y el valor de y es -6/5. Verifiquemos si la solución que hemos obtenido satisface realmente las ecuaciones. Para ello, primero crearemos una etiqueta, soln, para referirnos a la solución que hemos obtenido y, a continuación, utilizaremos el método subs() para sustituir los valores correspondientes de x y y en las dos expresiones:

>>> soln = solve((expr1, expr2), dict=True)  
>>> soln = soln[0]  
>>> expr1.subs({x:soln[x], y:soln[y]})  
0  
>>> expr2.subs({x:soln[x], y:soln[y]})  
0

El resultado de sustituir los valores de x y y correspondientes a la solución en las dos expresiones es cero.

[anterior](ch04_3.html)[Subtema 4 de 7: (Ver todo)](ch04.html)[siguiente](ch04_5.html)