Capítulo 4: Álgebra y matemáticas simbólicas con SymPy

### **Trazar con SymPy**

En [el Capítulo 2](ch02.html#ch02) aprendimos a hacer gráficas en las que especificábamos explícitamente los números que queríamos representar. Por ejemplo, para trazar el gráfico de la fuerza gravitatoria frente a la distancia entre dos cuerpos, tenías que calcular la fuerza gravitatoria para cada valor de distancia y proporcionar las listas de distancias y fuerzas a matplotlib. Con SymPy, en cambio, sólo tienes que decirle la ecuación de la recta que quieres trazar y el gráfico se creará por ti. Vamos a trazar una recta cuya ecuación viene dada por *y* =*2x* + 3:

>>> from sympy.plotting import plot  
>>> from sympy import Symbol  
>>> x = Symbol('x')  
>>> plot(2\*x+3)

Todo lo que tuvimos que hacer fue importar plot y Symbol de sympy.plotting, crear un símbolo, x, y llamar a la función plot() con la expresión 2\*x+3. SymPy se encarga de todo lo demás y traza la gráfica de la función, como se muestra en la [Figura 4-1](ch04.html#ch4fig1).

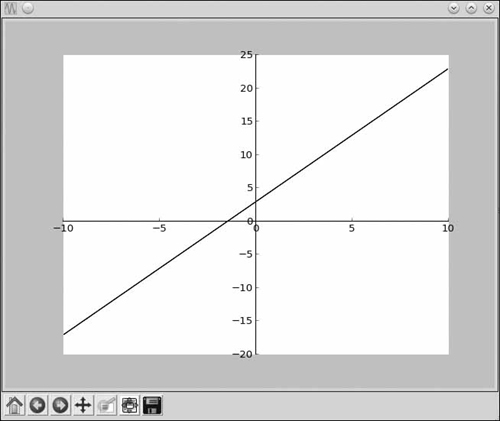


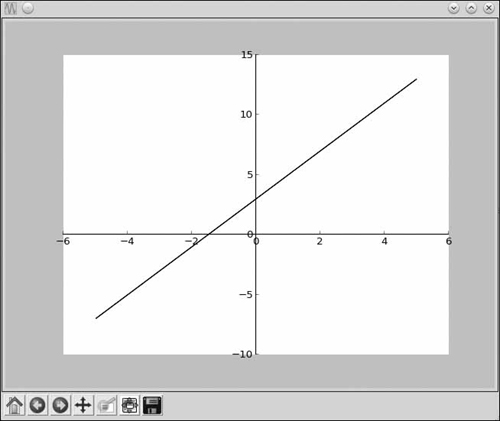
Figura 4-1*: Gráfico de la recta* y = *2x*+ *3*

La gráfica muestra que se ha elegido automáticamente un intervalo predeterminado de valores de *x*: de -10 a 10. Puede que notes que la ventana de la gráfica tiene un aspecto muy similar a las que viste en [los Capítulos 2](ch02.html#ch02) y [3](ch03.html#ch03). Esto se debe a que SymPy utiliza un lenguaje de programación muy sencillo. Esto se debe a que SymPy utiliza matplotlib entre bastidores para dibujar los gráficos. Observa también que no hemos tenido que llamar a la función show() para mostrar las gráficas, porque SymPy lo hace automáticamente.

Ahora, supongamos que quieres limitar los valores de 'x' en el gráfico anterior para que se encuentren en el intervalo de -5 a 5 (en lugar de -10 a 10). Lo harías de la siguiente manera

>>> plot((2\*x + 3), (x, -5, 5))

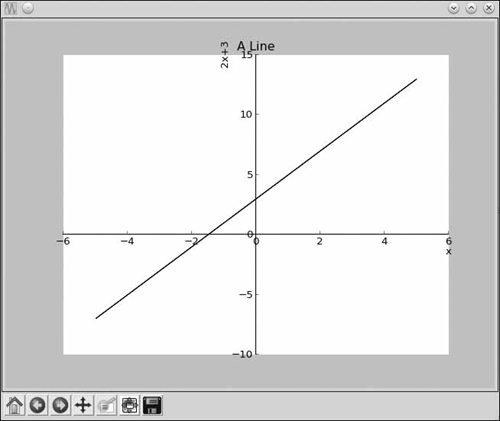
Aquí, una tupla formada por el símbolo, el límite inferior y el límite superior del intervalo -(x, -5, 5)- se especifica como segundo argumento de la función plot(). Ahora, el gráfico sólo muestra los valores de *y* correspondientes a los valores de *x* comprendidos entre -5 y 5 (ver [Figura 4-2](ch04.html#ch4fig2)).



*Figura 4-2: Gráfico de la recta* y = *2x*+ *3 con los valores de* x *restringidos al intervalo de -5 a* 5

Puedes utilizar otros argumentos de palabra clave en la función plot(), como title para introducir un título o xlabel y ylabel para etiquetar el eje *x* y el *eje y*, respectivamente. La siguiente función plot() especifica los tres argumentos de palabra clave anteriores (véase el gráfico correspondiente en la [Figura 4-3](ch04.html#ch4fig3)):

>>> plot(2\*x + 3, (x, -5, 5), title='A Line', xlabel='x', ylabel='2x+3')



*Figura 4-3: Gráfico de la recta* y = *2x*+ *3 con el rango de* x *y otros atributos especificados*

El gráfico de la Figura [4-3](ch04.html#ch4fig3) tiene ahora un título y etiquetas en el eje *x* y en el *eje y*. Puedes especificar otros argumentos de palabra clave a la función plot() para personalizar el comportamiento de la función y del propio gráfico. El argumento de palabra clave show nos permite especificar si queremos que se muestre la gráfica. Si pasas show=False, el gráfico no se mostrará cuando llames a la función plot():

>>> p = plot(2\*x + 3, (x, -5, 5), title='A Line', xlabel='x', ylabel='2x+3', show=False)

Verás que no se muestra ninguna gráfica. La etiqueta p hace referencia a la gráfica que se crea, por lo que ahora puedes llamar a p.show() para mostrar la gráfica. También puedes guardar la gráfica como un archivo de imagen utilizando el método save(), como se indica a continuación:

>>> p.save('line.png')

Esto guardará el gráfico en un archivo *line.png* en el directorio actual.

#### ***Trazado de expresiones introducidas por el usuario***

La expresión que pases a la función plot() debe expresarse sólo en términos de *x*. Por ejemplo, antes hemos trazado *y* =*2x* + 3, que hemos introducido en la función trazar como simplemente*2x* + 3. Si la expresión no estuviera originalmente en esta forma, tendríamos que reescribirla. Por supuesto, podríamos hacerlo manualmente, fuera del programa. Pero, ¿y si quieres escribir un programa que permita a sus usuarios representar gráficamente cualquier expresión? Si el usuario introduce una expresión en forma de*2x* + *3y* - 6, digamos, primero tenemos que convertirla. La función solve() nos ayudará en este caso. Veamos un ejemplo:

>>> expr = input('Enter an expression: ')  
Enter an expression: 2\*x + 3\*y - 6  
➊ >>> expr = sympify(expr)  
➋ >>> y = Symbol('y')  
>>> solve(expr, y)  
➌ [-2\*x/3 + 2]

En ➊, utilizamos la función sympify() para convertir la expresión de entrada en un objeto SymPy. En ➋, creamos un objeto Symbol para representar 'y', de modo que podamos decirle a SymPy para qué variable queremos resolver la ecuación. A continuación, resolvemos la expresión para encontrar y en términos de x especificando y como segundo argumento de la función solve(). En ➌, esto devuelve la ecuación en términos de x, que es lo que necesitamos para el trazado.

Observa que esta expresión final está almacenada en una lista, así que antes de poder utilizarla, tendremos que extraerla de la lista:

>>> solutions = solve(expr, 'y')  
➍ >>> expr\_y = solutions[0]  
>>> expr\_y  
-2\*x/3 + 2

Creamos una etiqueta, solutions, para referirnos al resultado devuelto por la función solve(), que es una lista con un solo elemento. Luego, extraemos ese elemento en ➍. Ahora podemos llamar a la función plot() para representar gráficamente la expresión. El siguiente listado muestra un programa completo de trazado de gráficos:

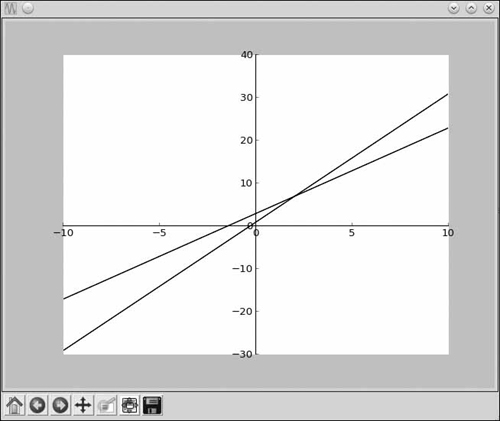
'''  
Plot the graph of an input expression  
'''  
  
from sympy import Symbol, sympify, solve  
from sympy.plotting import plot  
  
def plot\_expression(expr):  
  
y = Symbol('y')  
solutions = solve(expr, y)  
expr\_y = solutions[0]  
plot(expr\_y)  
  
if \_\_name\_\_=='\_\_main\_\_':  
  
expr = input('Enter your expression in terms of x and y: ')  
  
try:  
expr = sympify(expr)  
except SympifyError:  
print('Invalid input')  
else:  
plot\_expression(expr)

Observa que el programa anterior incluye un bloque try...except para comprobar si la entrada no es válida, como hemos hecho antes con sympify(). Cuando ejecutes el programa, te pedirá que introduzcas una expresión y creará el gráfico correspondiente.

#### ***Trazar múltiples funciones***

Puedes introducir varias expresiones al llamar a la función SymPy plot para trazar más de una expresión en el mismo gráfico. Por ejemplo, el siguiente código traza dos líneas a la vez (ver [Figura 4-4](ch04.html#ch4fig4)):

>>> from sympy.plotting import plot  
>>> from sympy import Symbol  
>>> x = Symbol('x')  
>>> plot(2\*x+3, 3\*x+1)

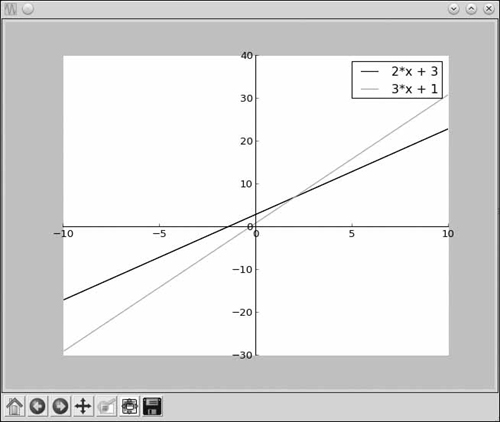


*Figura 4-4: Trazado de dos líneas en el mismo gráfico*

Este ejemplo pone de manifiesto otra diferencia entre el trazado en matplotlib y en SymPy. Aquí, con SymPy, las dos líneas tienen el mismo color, mientras que matplotlib las habría hecho automáticamente de colores diferentes. Para establecer colores diferentes para cada línea con SymPy, tendremos que realizar algunos pasos adicionales, como se muestra en el código siguiente, que también añade una leyenda al gráfico:

>>> from sympy.plotting import plot  
>>> from sympy import Symbol  
>>> x = Symbol('x')  
➊ >>> p = plot(2\*x+3, 3\*x+1, legend=True, show=False)  
➋ >>> p[0].line\_color = 'b'  
➌ >>> p[1].line\_color = 'r'  
>>> p.show()

En ➊, llamamos a la función plot() con las ecuaciones de las dos líneas, pero le pasamos dos argumentos clave adicionales:legend y show. Si establecemos el argumento leyenda en True, añadiremos una leyenda al gráfico, como vimos en el [Capítulo 2](ch02.html#ch02). Ten en cuenta, sin embargo, que el texto que aparezca en la leyenda coincidirá con las expresiones que hayas trazado; no puedes especificar ningún otro texto. También establecemos show=False porque queremos establecer el color de las líneas antes de dibujar el gráfico. La expresión en ➋, p[0], se refiere a la primera línea,*2x* + 3, y fijamos su atributo line\_color en 'b', lo que significa que queremos que esta línea sea azul. Del mismo modo, establecemos el color del segundo trazado en rojo utilizando la cadena 'r' ➌. Por último, llamamos a show() para que muestre el gráfico (véase [la Figura 4-5](ch04.html#ch4fig5)).



*Figura 4-5: Gráfico de las dos líneas con cada línea dibujada en un color diferente*

Además de rojo y azul, puedes trazar las líneas en verde, cian, magenta, amarillo, negro y blanco (utilizando la primera letra del color en cada caso).

[anterior](ch04_4.html)[Subtema 5 de 7: (Ver todo)](ch04.html)[siguiente](ch04_6.html)