Sympy: un sistema de álgebra computacional

SymPy es un sistema de álgebra computacional (CAS) escrito en Python puro, está desarrollado con un enfoque en la extensibilidad y facilidad de uso, tanto a través de aplicaciones interactivas como programáticas. Estas características le han permitido a SymPy convertirse en una librería de cómputo simbólico muy popular en el ecosistema científico de Python.

En todo este capítulo se asumirá que se ha importado la librería SymPy, con las siguientes instrucciones:

In [2]:
```
from sympy import *
```

Adicionalmente dentro del entorno de Jupyter podemos ejecutar la siguiente instrucción que permite *renderizar* las expresiones resultantes.

In [3]:
```
init_printing()
```

Variables simbólicas

Las variables simbólicas son el *alma* de SymPy, todas las operaciones de álgebra simbólica se basan en estas. A una variable simbólica se le asigna un símbolo que la representa mediante la función `symbols`:

In [4]:
```
x = symbols("x")
print(type(x))
```

```
<class 'sympy.core.symbol.Symbol'>
```

Con la función `symbols` se define una nueva variable simbólica que se guarda en `x`, se verifica que se crea un objeto de la clase `Symbol`. Con la variable `x` ya definida se puede comenzar a formar expresiones algebraicas y manipular matemáticamente.

In [5]:
```
x + 2
```

Out[5]:

$$x + 2$$

In [6]:
```
x**2 - 2*x - 10
```

Out[6]:

$$x^2 - 2x - 10$$

In [7]:
```
sqrt(x) - sin(x)
```

Out[7]:

$$\sqrt{x} - \sin(x)$$

Puede definir múltiples variables separando por comas cada representación simbólica:

In [9]:
```
p,q,r = symbols("p,q,r")
p**q + r/q
```

Out[9]:

$$p^q + \frac{r}{q}$$

Además de la forma anterior, también es posible tener variables simbólicas disponibles si se importan del módulo `abc` de SymPy:

In [10]:
```
from sympy.abc import a,b,c,d
```

In [11]:
```
(a + b)*(c + d)
```

Out[11]:

$$(a + b)(c + d)$$

Manipulación algebraica

SymPy es una poderosa herramienta de manipulación y simplificación algebraica, en lo subsiguiente se revisarán algunos casos elementales y se describirá el uso de las herramientas (funciones) que proporciona.

En primera instancia se definen algunas variables simbólicas a utilizar:

In [12]:
```
x,y,z = symbols("x,y,z")
a,b,c,d,k,m,n = symbols("a,b,c,d,k,m,n")
```

Para las expresiones algebraicas formadas en SymPy por default se *evalúan* y simplifican los términos semejantes. Vea los siguientes casos:

In [13]:
```
x**2 + 5*x**3 - 10*x**2 + 5*x - 10*(x + 1)
```

Out[13]:

$$5x^3 - 9x^2 - 5x - 10$$

In [14]:
```
a*b + c*d + x/y - 7*a*b
```

Out[14]:

$$-6ab + cd + \frac{x}{y}$$

Naturalmente para simplificaciones y operaciones un poco menos obvias, habrá que especificarle lo que se requiere.

Una de las operaciones elementales de simplificación en álgebra es la factorización, SymPy dispone de la función `factor`, la cual toma una expresión algebraica y la factoriza conforme sea posible. Por ejemplo, suponiendo que se tiene la expresión $ab + ac$, es sencillo identificar que se puede
factorizar como $a(b + c)$, SymPy hace lo mismo sin sobresalto alguno.

In [15]:
```
factor(a*b + a*c)
```

Out[15]:

$$a(b + c)$$

De igual forma se sabe que una expresión de la forma $ac + ad + bc + bd$ se puede factorizar como $(a + b)(d + c)$.

In [16]:
```
factor( a*c + a*d + b*c + b*d )
```

Out[16]:

$$(a + b)(c + d)$$

Así, para un binomio al cuadrado se sabe que, $(a + b)(a + b)$ es la factorización de una expresión del tipo $a^2 + 2ab + b^2$.

In [17]:
```
factor(a**2 + 2*a*b + b**2)
```

Out[17]:

$$(a + b)^2$$

¿Se puede hacer el proceso *inverso*, es decir, dados dos factores obtener su expresión *expandida*? Por supuesto, para ello se hace uso de la función `expand`.

In [18]:
```
expand( (x-7)*(x-3) )
```

Out[18]:

$$x^2 - 10x + 21$$

SymPy puede manejar cualquier cantidad de factores y devolver la expresión que resulta de realizar la multiplicación algebraica, sin estar limitada siquiera por la cantidad de variables simbólicas.

In [19]:
```
expand( (x + y)*(x - y) )
```

Out[19]:

$$x^2 - y^2$$

In [20]:
```
expand( (x + y)*(x+y) )
```

Out[20]:

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

In [21]:
```
expand( (x + y)**2 )
```

Out[21]:

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

In [22]:
```
expand( (x + y)**3 )
```

Out[22]:

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

In [23]:
```
expand( a*(m + n)**2 )
```

Out[23]:

$$am^2 + 2amn + an^2$$

De manera general, si requiere simplificar una expresión algebraica SymPy dispone de una función más o menos universal que funcionará en la mayoría de los casos: `simplify` . Por ejemplo, suponiendo que se tiene la siguiente expresión algebraica racional y se requiere reducir lo más posible:

$$\frac{x^2 - 3x}{x^2 + 3x}$$

Se puede notar que tanto en el numerador como en el denominador se puede factorizar x, lo cual conduce a:

$$\frac{x(x-3)}{x(x+3)} = \frac{x-3}{x+3}$$

Con SymPy:

In [25]:
```
simplify( (x**2 - 3*x)/(x**2 + 3*x) )
```

Out[25]:

$$\frac{x-3}{x+3}$$

Lo mismo para una expresión que involucre funciones trigonométricas, por ejemplo, cualquiera que haya cursado matemáticas del nivel secundario, al menos, sabe que $\sin^2 x + \cos^2 x = 1$, y también *lo sabe* SymPy:

In [26]:
```
simplify( sin(x)**2 + cos(x)**2 )
```

Out[26]:

$$1$$

Pero también \textit{sabe} que $\cos x \cos y - \sin x \sin y = \cos(x + y)$.

In [27]:
```
simplify( cos(x)*cos(y) - sin(x)*sin(y) )
```

Out[27]:

$$\cos(x + y)$$

Incluso simplificaciones que no son, en principio, evidentes:

In [29]:
```
simplify( (1+tan(x)**2)/(1+cot(x)**2) )
```

Out[29]:

$$\tan^2(x)$$

Habitualmente para la manipulación de expresiones que contienen funciones trigonométricas se suele utilizar la función `trigsimp` que en muchos casos lo que devuelva coincidirá con `simplify`. Sin embargo, si en lugar de reducir una expresión trigonométrica se requiere expandirla, como en el caso del coseno o seno de la suma de dos ángulos, probablemente por intuición se utilizaría `expand`, pero aquí no funciona como puede verificar.

In [31]:
```
expand( sin(x + y) )
```

Out[31]:

$$\sin(x + y)$$

En este caso puede utilizar la función `expand\_trig`, la cual maneja de mejor manera las manipulaciones trigonométricas:

In [32]:
```
expand_trig( sin(x+y) )
```

Out[32]:

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

Resolviendo ecuaciones e inecuaciones

SymPy dispone de la función `solve`, la cual resuelve desde ecuaciones polinomiales, sistemas de ecuaciones lineales, inecuaciones, hasta sistemas de ecuaciones no lineales. La sintaxis de `solve` es polimórfica y en general depende de lo que se requiera resolver, tendiendo siempre a que sea posible especificar el mínimo número de parámetros.

En las siguientes subsecciones se describen algunos casos, se asumirá en lo subsiguiente que además de importar SymPy se definieron las siguientes variables simbólicas:

In [33]:
```
x,y,z = symbols("x,y,z")
a,b,c = symbols("a,b,c")
```

Ecuaciones polinómicas

La sintaxis más elemental de `solve` es pasando un sólo argumento, el cual se espera sea una expresión algebraica y se considera que esta estará igualada a cero. Por ejemplo para resolver la ecuación lineal $x - 3 = 0$:

In [34]:
```
solve( x - 3 )
```

Out[34]:

$$[3]$$

Para una ecuación cuadrática $x^2 + 2x + 2 = 0$:

solve(x*2 + 2x + 2)

En este caso la ecuación tiene soluciones complejas, la unidad imaginaria en SymPy se especifica mediante el símbolo `I`.

Si se quisiera resolver la ecuación cuadrática en su forma general, por intuición y lo que sabemos hasta ahora, haríamos:

In [38]:
```
solve(a*x**2 + b*x + c)
```

Out[38]:

$$\left[\left\{a : -\frac{1}{x^2}(bx + c)\right\}\right]$$

Note que el problema está en que al haber más de una variable simbólica, SymPy no sabe con respecto a qué variable debe resolver y toma la primera en orden alfabético. Para especificar explícitamente respecto a qué variable resolver, se puede indicar mediante un segundo argumento:

In [39]:
```
solve(a*x**2 + b*x + c, x)
```

Out[39]:

$$\left[\frac{1}{2a}\left(-b + \sqrt{-4ac + b^2}\right), \quad -\frac{1}{2a}\left(b + \sqrt{-4ac + b^2}\right)\right]$$

Puede verificar que devuelve, en efecto, la tan conocida fórmula general.

Ecuaciones trigonométricas

Cálculo

Límites

Para calcular límites matemáticos SymPy dispone de la función `limit`, la cual requiere al menos tres argumentos de entrada: la función, variable y valor al que tiende. Por ejemplo, si se quiere calcular el siguiente límite:

$$\lim_{x \to 0} \frac{\sin x}{x}$$

Tendría que teclearse lo siguiente:

In [41]:
```
limit(sin(x)/x, x, 0)
```

Out[41]:

$$1$$

Para calcular limites laterales debe pasarse un cuarto argumento, por ejemplo:

In [42]:
```
limit(1/(x-5), x, 5, "-")
```

Out[42]:

$$-\infty$$

In [43]:
```
limit(1/(x-5), x, 5, "+")
```

Out[43]:

$$\infty$$

El símbolo `+` denota el cálculo de un límite lateral por la derecha y el símbolo `-` un límite lateral por la izquierda.

Derivadas

Las derivadas en Python se calculan utilizando la función `diff`, misma que en su forma más simple espera al menos dos argumentos: una expresión algebraica y una variable respecto a la cual derivar. Por ejemplo:

In [44]:
```
diff(exp(x)*cos(x), x)
```

Out[44]:

$$-e^x\sin(x) + e^x\cos(x)$$

Es posible también especificar una derivada de orden superior mediante un tercer argumento:

In [45]:
```
diff(5*x**2 + 3*x - 10, x, 2)
```

Out[45]:

$$10$$

La instrucción anterior calcula la segunda derivada de la función $f(x) = 5x^2 + 3x - 10$.

Integrales

Para calcular integrales vamos a utilizar la función `integrate`, la cual acepta al menos dos argumentos: la función a integrar y la expresión respecto a la cual se integra, por ejemplo:

In [46]:
```
integrate(x**2 + 3*x - 7, x)
```

Out[46]:

$$\frac{x^3}{3} + \frac{3x^2}{2} - 7x$$

Esa instrucción calcula la integral:

$$\int \left(x^2 + 3x - 7\right) dx = \frac{x^3}{3} + \frac{3x^2}{2} - 7x + C$$

Note que la expresión algebraica devuelta por Python no contiene la constante de integración, por default SymPy no la considera. Sí en algún caso específico necesita referir a la constante de integración puede adicionarla manualmente.

Las integrales definidas se pueden calcular si el segundo argumento se hace una tupla de la forma `(variable, a, b)`, donde `a` y `b` indican el límite inferior y superior a evaluar en la integral:

In [47]:
```
integrate(sin(x), (x, 0, pi))
```

Out[47]:

$$2$$

In [48]:
```
integrate(z**2, (z,-5,5))
```

Out[48]:

$$\frac{250}{3}$$

Vectores

Un vector denota una cantidad física que tiene magnitud y orientación en un determinado sistema de referencia. En SymPy los vectores se pueden definir mediante la clase \code{Matrix} del módulo \code{matrices}, de tal manera que en primera instancia haría falta importar dicho módulo, para esto hacemos:

In [50]:
```
from sympy.matrices import Matrix
```

Ahora vamos a definir dos vectores $\vec{u}$ y $\vec{v}$:

$$\vec{u} = \begin{bmatrix} 2 \\ 1 \\ -5 \end{bmatrix} \quad ; \quad \vec{v} = \begin{bmatrix} 4 \\ -1 \\ 3 \end{bmatrix}$$

In [51]:
```
u = Matrix([2,1,-5])
v = Matrix([4,-1,3])
```

Una suma y resta vectorial se pueden ejecutar sin muchas complicaciones, mediante los operadores aritméticos ya conocidos.

In [52]:
```
u + v
```

Out[52]:

$$\begin{bmatrix} 6 \\ 0 \\ -2 \end{bmatrix}$$

In [54]:
```
v - u
```

Out[54]:

$$\begin{bmatrix} 2 \\ -2 \\ 8 \end{bmatrix}$$

La magnitud de un vector se puede calcular utilizando el método `norm` .

In [55]:
```
u.norm()
```

Out[55]:

$$\sqrt{30}$$

In [56]:
```
v.norm()
```

Out[56]:

$$\sqrt{26}$$

Recuerde que si requiere ver las expresiones resultantes como fracciones decimales debe usar `evalf` .

El producto escalar de dos vectores puede calcularlo utilizando el método `dot` , por ejemplo $\vec{u} \cdot \vec{v}$ lo puede especificar como:

In [58]:
```
u.dot(v)
```

Out[58]:

$$-8$$

Para calcular el producto vectorial utilice el método `cross` , por ejemplo $\vec{u} \times \vec{v}$:

In [59]:
```
u.cross(v)
```

Out[59]:

$$\begin{bmatrix} -2 \\ -26 \\ -6 \end{bmatrix}$$

Recuerde que el producto vectorial no es conmutativo, por tanto, $\vec{v} \times \vec{u}$ resultará en un vector diferente al obtenido anteriormente, como puede verificar enseguida:

In [60]:
```
v.cross(u)
```

Out[60]:

$$\begin{bmatrix} 2 \\ 26 \\ 6 \end{bmatrix}$$

Matrices

Las matrices son arreglos rectangulares de números o cantidades simbólicas. En SymPy, se definen utilizando la clase `Matrix`, pasándole como argumentos una lista de listas, donde cada sublista corresponde a una fila de la matriz.

Por ejemplo, vamos a definir las matrices A, B y C, dadas por:

$$A = \begin{bmatrix} 20 & 50 & 100 \\ 10 & 35 & 200 \\ -30 & 20 & 80 \end{bmatrix} \quad B = \begin{bmatrix} 12 & 26 & 45 \\ 3 & -15 & 18 \\ 54 & 20 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 5 & 9 \\ 2 & 3 \\ -10 & 8 \end{bmatrix}$$

Primero, importamos la clase `Matrix` del módulo `matrices`:

In [61]:
```
from sympy.matrices import Matrix
```

Luego escribiríamos:

In [64]:
```
A = Matrix([[20,50,100], [10,35,200], [-30,20,80]])
B = Matrix([[12,26,45],[3,-15,18],[54,20,0]])
C = Matrix([[5,9], [2,3], [-10,8]])
```

De manera muy sencilla podríamos realizar operaciones matriciales básicas, por ejemplo, una suma:

In [65]:
```
A + B
```

Out[65]:

$$\begin{bmatrix} 32 & 76 & 145 \\ 13 & 20 & 218 \\ 24 & 40 & 80 \end{bmatrix}$$

Naturalmente, y como es esperable, SymPy *conoce* y aplica las reglas de álgebra de matrices, observe:

In [66]:
```
A + C
```

```
---------------------------------------------------------------------------
ShapeError                                Traceback (most recent call last)
<ipython-input-66-c3cb66996588> in <module>()
----> 1 A + C

~\Anaconda3\lib\site-packages\sympy\core\decorators.py in binary_op_wrapper(self, other)
    130                     else:
    131                         return f(self)
--> 132             return func(self, other)
    133         return binary_op_wrapper
    134     return priority_decorator

~\Anaconda3\lib\site-packages\sympy\matrices\common.py in __add__(self, other)
   1949             if self.shape != other.shape:
   1950                 raise ShapeError("Matrix size mismatch: %s + %s" % (
-> 1951                     self.shape, other.shape))
   1952
   1953         # honest sympy matrices defer to their class's routine

ShapeError: Matrix size mismatch: (3, 3) + (3, 2)
```

SymPy es muy explícito en ese tipo de situaciones y nos imprime un mensaje de error suficientemente descriptivo. De igual forma que es válido realizar el producto AC, pero no CA:

In [68]:
```
A*C
```

Out[68]:

$$\begin{bmatrix} -800 & 1130 \\ -1880 & 1795 \\ -910 & 430 \end{bmatrix}$$

In [69]:
```
C*A
```

```
---------------------------------------------------------------------------
ShapeError                                Traceback (most recent call last)
<ipython-input-69-cbae4c009e52> in <module>()
----> 1 C*A

~\Anaconda3\lib\site-packages\sympy\core\decorators.py in binary_op_wrapper(self, other)
    130                     else:
    131                         return f(self)
--> 132             return func(self, other)
    133         return binary_op_wrapper
    134     return priority_decorator

~\Anaconda3\lib\site-packages\sympy\matrices\common.py in __mul__(self, other)
   2006             if self.shape[1] != other.shape[0]:
   2007                 raise ShapeError("Matrix size mismatch: %s * %s." % (
-> 2008                     self.shape, other.shape))
   2009
   2010         # honest sympy matrices defer to their class's routine

ShapeError: Matrix size mismatch: (3, 2) * (3, 3).
```

El **determinante** de una matriz podemos calcularlo mediante la función `det`:

In [70]:
```
det(B)
```

Out[70]:

$$60102$$

O bien, mediante el propio método `det` :

In [71]:
```
B.det()
```

Out[71]:

$$60102$$

La **matriz inversa** podemos calcularla utilizando el método `inv` :

In [72]:
```
A.inv()
```

Out[72]:

$$\begin{bmatrix} \frac{6}{1195} & \frac{2}{239} & -\frac{13}{478} \\ \frac{34}{1195} & -\frac{23}{1195} & \frac{3}{239} \\ -\frac{5}{956} & \frac{19}{2390} & -\frac{1}{1195} \end{bmatrix}$$

In [73]:
```
A.inv()*A
```

Out[73]:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La **transpuesta** de una matriz se puede obtener accediendo al atributo `T`

In [74]:
```
C.T
```

Out[74]:

$$\begin{bmatrix} 5 & 2 & -10 \\ 9 & 3 & 8 \end{bmatrix}$$

Ejercicios

Utilice Python/SymPy para resolver los siguientes ejercicios:

1. Resuelva las siguientes ecuaciones (para la variable x):
   A. $x + 3x = 10$
   B. $2x^2 - 10x + 5 = 0$
   C. $\frac{a}{x} + bx - 8 = 0$
   D. $\cos x + \sin x = 10$
   E. $kx - 10 = x^2$
2. Calcule las siguientes derivadas:
   A. $\frac{d}{dx}(\cos x)$
   B. $\frac{d}{dt}\left(at^2 - 2\tan t - \frac{10}{t}\right)$
   C. $\frac{d}{dz}\left(z^5 - 10e^{-z}\right)$
3. Calcule las siguientes integrales indefinidas:
   A. $\int \cos x \, dx$
   B. $\int \left(x^3 - 8x\right) dx$

C. $\int \left(e^{-3x} \sin x \right) dx$

D. $\int \left(s^2 + ks - m \right) ds$

4. Calcule las siguientes integrales definidas:
   A. $\int_a^b \sin x \, dx$
   B. $\int_{-10}^{5} x^2 + 10x \, dx$
   C. $\int_0^2 10e^{-z} \, dz$
5. Resuelva la siguiente ecuación diferencial:

$$\frac{dS}{dr} = kS$$

6. Al sacar un pastel del horno, su temperatura es de 300 °F. Después de 3 minutos, 200 °F. ¿En cuanto tiempo se enfriará hasta la temperatura ambiente de 70 °F?
7. Escriba una función que dados como entrada dos vectores, determine sí estos son paralelos.
8. Desarrolle una función llamada \code{calcula_fuerza_resultante} que reciba como datos de entrada un conjunto de vectores fuerza y devuelva el vector de fuerza resultante.
9. Escriba un función llamada \code{calcula_momento_resultante} que reciba como datos de entrada un conjunto de vectores fuerza y un conjunto de vectores de posición del punto de aplicación de la fuerza con respecto a un cierto punto. Se deberá devolver el momento total producido por las fuerzas con respecto al punto.

In [ ]: