Introducción a SymPy

February 2, 2021

Elaboró Hernández Martínez Erick Daniel
Github: Erickkhm

1 Introducción a SymPy

1.1 ¿Qué es SymPy?

SymPy es un sistema de álgebra informática (CAS) escrito en el lenguaje de programación Python . SymPy es fácil de usar e instalar y funciona en todos los lugares donde está instalado Python 2.7 o posterior.

Simpy es una paquetería que nos permite hacer cálculo simbólico y numérico, contiene funciones y aplicaciones que son muy útiles para elaborar cálculo complejo

Un sistema de álgebra de computadora (CAS) es un programa de computadora diseñado para manipular matemáticas , expresiones temáticas en forma simbólica en lugar de numérica, una tarea que se lleva a cabo por científicos teóricos. Hay una serie de programas especializados y algunos sistemas de propósito aplicables a una amplia gama de problemas, y es este último el que considerar aquí.

La característica más sorprendente de SymPy es que está escrito completamente en Python y, de hecho, es solo un módulo adicional. Es pequeño y funciona en cualquier sistema Python. Interfaces Bueno, a NumPy para números y a Matplotlib para gráficos. El modo terminal estándar tiene una salida poco sofisticada, pero, naturalmente, se puede utilizar dentro del portátil Jupyter, y allí ofrece quizás el mejor formato disponible actualmente de complicados simbólicos salida.

Simpy es una paquetería de Python que utiliza y recibe el lenguaje de python, en la paquetería podemos encontrar la creación e interpretación de símbolos y funciones e incluye funciones que nos permiten crear funciones matemáticas más complejas ya que simpy cuanta con funciones como derivar, integrar, simplificar expresiones e incluye el reconocimiento e interpretación de números enteros, racionales, flotantes y números complejos, entre otras características más, todo ello da como resultado que simpy sea una paquetería que nos brinda múltiples herramientas para realizar calculo y programas matemáticos de manera mucho más sencilla y eficaz.

2 Símbolos

El verdadero poder de las matemáticas viene del uso de variables algebraicas. Para ello, debe utilizar los objetos Symbol. Para crearlos, debe especificar su nombre y, opcionalmente, puede especificar varios “supuestos” sobre el variable. Los supuestos disponibles incluyen “entero”,

"real", "positivo", etc. Si ninguno es especificado, el objeto Símbolo se trata como una variable compleja. Implicaciones entre propiedades se tienen en cuenta automáticamente y las combinaciones imposibles provocan un error.

Hay varias formas de crear instancias de clases, y quizás la forma más común de manejar insertar la línea x, y = sy.symbols ("x y")

La función de símbolos () facilita la creación de muchos símbolos a la vez. La palabra clave los argumentos que le da se pasan al constructor Symbol (). Para el primer argumento, acepta un mini-lenguaje inspirado en la notación de segmento de Python, que se describe mejor con el siguiente ejemplo:

[1]:
```
from sympy import *
```

Podemos crear símbolos usando la función symbols() y podemos crear varios símbolos a la vez como se muestra a continuación

[2]:
```
x = symbols('x')
t, y, z = symbols('t y z')
```

[3]:
```
z = Symbol('z') # variable compleja
x = Symbol('x', real=True) #simbolo entre los reales
a = Symbol('a', positive=True) # simbolo en los positivos
```

[4]:
```
print(type(x))
print(type(t))
```

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

También podemos crear una serie de símbolos usando literales indicando la letra de inicio y el límite.

[5]:
```
symbols('a:c, i:k')
```

[5]:
```
(a, b, c, i, j, k)
```

Al igual que con las literales se puede utilizar un símbolo con un subíndice colocando primero el símbolo y después numero de símbolos que se desea crear, se puede indicar desde que subíndice se desea empezar.

[6]:
```
symbols('x:5')
```

[6]:
```
(x0, x1, x2, x3, x4)
```

[7]:
```
symbols('x3:12')
```

[7]:
```
(x3, x4, x5, x6, x7, x8, x9, x10, x11)
```

Lo importante de definir y trabajar con símbolos es porque SymPy utiliza la sintaxis de Python y si nosotros no definimos como símbolo a una variable Python lo va a reconocer como una variable

que no está definida y nos va a saltar un error, al definir los símbolos Python los reconoce como una variable aunque aún no se le defina un valor en concreto.

A continuación se muestra el ejemplo:

[8]:
```
s**2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-8-8a329be56f67> in <module>
----> 1 s**2


NameError: name 's' is not defined
```

[9]:
```
s = symbols('S')
```

[10]:
```
s**2
```

[10]: S^2

Al definir los símbolos también podemos trabajar operaciones aritméticas y SymPy va a simplificar los términos semejantes

[11]:
```
3*x**2 + 7*y + 2*x**2 + (-5*y) - 4*x**2
```

[11]: $x^2 + 2y$

[ ]:
```

```

3 Funciones

En SymPy podemos usar y ejecutar funciones matemáticas y dado a que ya hemos creado símbolos antes podemos aplicarle funciones a nuestros símbolos de esta manera obtenemos una expresión resumida o resuelta pero de forma simbólica es decir que aún no evaluamos valores numéricos, prácticamente como si de cálculo teórico se tratase.

Dentro de la creación de funciones al utilizar el lenguaje de Python, podemos definir funciones como previamente ya conocemos y SymPy va a trabajar con ellas sin problemas, además también podemos definir funciones simbólicas esto quiere decir que se expresa que es una función pero no se tiene que definir.

[12]:
```
f = Function('f')
```

[13]:
```
print(type(f))
f(0)
```

```
<class 'sympy.core.function.UndefinedFunction'>
```

[13]: $f(0)$

Aquí definimos una función utilizando el recurso de función anónima lambda que reciba de argumento a 'x'

[14]:
```
g =lambda x: x**2 + 3*x - 21
g(x)
```

[14]: $x^2 + 3x - 21$

También podemos introducir otros símbolos o variables matemáticas que ya hayamos definido antes

[15]:
```
g(x*y**2/3)
```

[15]: $\frac{x^2y^4}{9} + xy^2 - 21$

[16]:
```
g(x**2 + 3)
```

[16]: $3x^2 + \left(x^2 + 3\right)^2 - 12$

[17]:
```
g(3)
```

[17]: -3

SymPy tiene dentro de su paquetería funciones como sin, cos, tan, pi, exp entre otras por esto mismo es que Sympy nos puede simplificar bastante el cálculo.

[18]:
```
f = cos(exp(sin(x**pi)))
f
```

[18]: $\cos\left(e^{\sin\left(x^{\pi}\right)}\right)$

Dentro de SymPy podemos crear expresiones matemáticas solo usando la sintaxis de Python y con ellas ya podremos realizar el cálculo en SymPy

4 Manipulación de expresiones

5 Substitución

Los objetos y expresiones de SymPy son inmutables. Transformarlos en realidad significa crear un nuevo objeto que difiere del inicial de alguna manera. La forma más directa de hacer esto es utilice el método .subs (). subs (antiguo, nuevo) devuelve una nueva expresión donde todos los las ocurrencias de lo antiguo en la expresión han sido reemplazadas por nuevas. Para reemplazar varias expresiones

diferentes de una sola vez se debe pasar como un solo argumento a un diccionario de los pares {antiguo: nuevo}, como se muestra en el siguiente ejemplo:

[19]:
```
x, y = symbols('x, y', real=True)
f = lambda x, y:(cos(x) * exp(y))
f(x, y)
```

[19]: $e^{y}\cos\left(x\right)$

Si colocamos dentro de la función subs() un diccionario con los símbolos que deseamos sustituir podemos cambiar los varios valores a la vez

[20]:
```
f(x,y).subs({x: pi, y: 2})
```

[20]: $-e^{2}$

[21]:
```
funcion = exp(x+1)/(x**2-2)
funcion
```

[21]: $\frac{e^{x+1}}{x^{2}-2}$

A continuación tenemos un ejemplo de como sympy con el métodos .subs() puede cambiar de símbolos que son variables complejas a valores u otras variables

[22]:
```
funcion.subs(x,2)
```

[22]: $\frac{e^{3}}{2}$

6 Simplificación

Simplificación Cuando tienes una expresión simbólica, una necesidad común es ponerla en una forma más corta o que es más fácil de entender. Casi siempre es más conveniente tratar con 1 que con $cos(x^2) + sin(x^2)$. Aquí es donde entra en uso la función simplify (). simplify () aplica una combinación heurística de algoritmos de simplificación. Puede usar relaciones trigonométricas, aplicar propiedades de logaritmos y exponenciales, simplificar fracciones y raíces cuadradas, etc. En todas estas operaciones, se tienen en cuenta las propiedades de los objetos, particularmente is_positive e is_real, como se muestra en el siguiente ejemplo:

[23]:
```
f = cos(x)**2 + sin(x)**2
f
```

[23]: $\sin^{2}\left(x\right)+\cos^{2}\left(x\right)$

La función simplify() de sympy puede reconocer identidades trigonométricas y simplificarlas a la expresión más sencilla aquí unos ejemplos.

[24]:
```
simplify(f)
```

[24]: 1

[25]:
```
alpha = symbols('alpha')
alpha
```

[25]:
$$\alpha$$

Fórmulas de doble ángulo

[26]:
```
s = cos(alpha)**2 - sin(alpha)**2
s
```

[26]:
$$-\sin^2(\alpha) + \cos^2(\alpha)$$

[27]:
```
simplify(s)
```

[27]:
$$\cos(2\alpha)$$

Identidades de productos

[28]:
```
d = sin(alpha)*cos(alpha)
d
```

[28]:
$$\sin(\alpha)\cos(\alpha)$$

[29]:
```
simplify(d)
```

[29]:
$$\frac{\sin(2\alpha)}{2}$$

Identidades fundamentales

[30]:
```
t = sin(alpha)/cos(alpha)
t
```

[30]:
$$\frac{\sin(\alpha)}{\cos(\alpha)}$$

[31]:
```
simplify(t)
```

[31]:
$$\tan(\alpha)$$

Logaritmos y exponeciales

[32]:
```
c = (cos(log(a**2))*tan(log(a**2)))
c
```

[32]:
$$\cos\left(\log\left(a^2\right)\right)\tan\left(\log\left(a^2\right)\right)$$

[33]:
```
simplify(c)
```

[33]:
$$\sin(2\log(a))$$

Aquí hay una lista de los métodos más útiles (hay más en el módulo sympy.simplify):
+ radsimp (): esto simplifica expresiones con raíces cuadradas + trigsimp ():

esto simplifica combinaciones de funciones trigonométricas + cancel (): Esto elimina los factores comunes entre el numerador y el denominador de una fracción + together (): Esto coloca las sumas de una fracción sobre el mismo denominador + apart (): esto aplica la descomposición de fracciones parciales

Aunque hay que aclarar que la función simplify() engloba todas estas, pero en caso de que solo se busque simplificar una en específico se pueden utilizar estas variantes que solo simplificaran la parte de la expresión que les corresponde.

Raíz cuadrada

[34]:
```
raiz = 1/(sqrt(5) - 2)
raiz
```

[34]:
$$\frac{1}{-2+\sqrt{5}}$$

[35]:
```
radsimp(raiz)
```

[35]:
$$2+\sqrt{5}$$

Identidades trigonometricas

[36]:
```
trigo = (sin(x)**2 + cos(x)**2)
trigo
```

[36]:
$$\sin^2(x)+\cos^2(x)$$

[37]:
```
trigsimp(trigo)
```

[37]:
$$1$$

Factores comunes

[38]:
```
factor_comun =((x**2 - 1)/(x - 1))
factor_comun
```

[38]:
$$\frac{x^2-1}{x-1}$$

[39]:
```
cancel(factor_comun)
```

[39]:
$$x+1$$

Suma de fracciones

[40]:
```
sum_frac = (1/(x-1) - 1/(x+1))
sum_frac
```

[40]:
$$-\frac{1}{x+1}+\frac{1}{x-1}$$

[41]:
```
together(sum_frac)
```

[41]:
$$\frac{2}{(x-1)(x+1)}$$

fracciones parciales

[42]:
```
frac_parciales = (2/(x**2 - 1))
frac_parciales
```

[42]:
$$\frac{2}{x^2-1}$$

[43]:
```
apart(frac_parciales)
```

[43]:
$$-\frac{1}{x+1}+\frac{1}{x-1}$$

Si tenemos alguna expresión que sympy pueda simplificar y usamos el método simplify() va a simplificar todo o que encuentre.

7 Evaluación númerica

La evaluación numérica sirve para obtener resultados numéricos para las expresiones simbólicas o para las funciones que hemos creado, el método de evaluación numérica nos va a definir la expresión en valores reales numéricos eso quiere decir que deja de ser simbólico el cálculo y la función para evaluar numéricamente es N() que recibe la expresión o función como primer argumento y como segundo recibe las el número de cifras de precisión

[44]:
```
pi
```

[44]:
$$\pi$$

[45]:
```
N(pi, 100)
```

[45]:
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706

Aquí tenemos una expresión compleja que al correr la celda nos entrega la expresión pero de manera simbólica

[46]:
```
(exp(pi*sqrt(163)))
```

[46]:
$$e^{\sqrt{163}\pi}$$

Aquí podemos ver una expresión que como resultado va a dar un numero flotante y para ello podemos agregar a la función N() un segundo argumento que nos de el número de digitos de precisión del resultado

[47]:
```
N(exp(pi*sqrt(163)), 50)
```

[47]:
262537412640768743.99999999999925007259719818568888

[48]:
```
len('262537412640768743.99999999999925007259719818568888')
```

[48]: 51

Aquí comprobamos la longitud de la cifra y nos da 51 pero phyton cuenta el punto como elemento por ello da 51 - 1 = 50 cifras de precisión

8 Calcúlo con SymPy

La principal aplicación de SymPy es que puede realizar calculo computacional, anterior mente ya definimos que es el cálculo simbólico, creamos símbolos y funciones además vimos cómo se manipulan y ejecutan.

Ahora toca ver como son las funciones que SymPy nos ofrece en el apartado del cálculo matemático y como es que se ejecutan

9 Derivadas

Para realizar el cálculo de una derivada de una función tenemos el método diff() este método recibe de argumento a la función o expresión como primer argumento, el símbolo o variable con respecto a al cual se quiere derivar y como último argumento al número u orden al cual se quiere derivar en caso de no poner este ultimo la función va a entregar la primer derivada.

[49]:
```
f = Function('f')
```

[50]:
```
diff(f(x))
```

[50]: $\frac{d}{dx} f(x)$

[51]:
```
f = x**3 + 3*x**2 + 2*x + 11
f
```

[51]: $x^3 + 3x^2 + 2x + 11$

[52]:
```
diff(f, x)
```

[52]: $3x^2 + 6x + 2$

Aquí obtenermos la tercera derivada de la función con respecto a 'x'

[53]:
```
diff(f,x,3)
```

[53]: 6

Cuando se tienen más de una variable en la función solo hay que especificar con respecto a que se quiere derivar, si lo que queremos es derivar con respecto a más de una variable solo introducimos en la función diff() la función seguido de la variable y después el número de veces que se quiere derivar con respecto a esta variable para después introducir la siguiente variable seguido del numero.

A contiuacion un ejemplo:

[54]:
```
f = x**2 * y**2
f
```

[54]: x^2y^2

[55]:
```
diff(f, x, 2, y, 2)
```

[55]: 4

[56]:
```
f = x**4 + sin(y) - x**3*y**2
f
```

[56]: $x^4 - x^3y^2 + \sin(y)$

[57]:
```
diff(f, x, 3, y, 1)
```

[57]: $-12y$

[58]:
```
f = exp(x*y*z)
f
```

[58]: e^{xyz}

En caso de que solo queramos expresar la derivada sin evaluar tenemos el método Derivative()

[59]:
```
deriv = Derivative(f, x, y, x, z, 4)
deriv
```

[59]: $\frac{\partial^7}{\partial z^4 \partial x \partial y \partial x} e^{xyz}$

Para evaluar una derivada no evaluada, use el método .doit()

[60]:
```
deriv.doit()
```

[60]: $x^2y^3\left(x^3y^3z^3 + 14x^2y^2z^2 + 52xyz + 48\right)e^{xyz}$

10 Integrales

SymPy tiene potentes algoritmos de integración y, en particular, puede encontrar la mayoría de integrales de funciones logarítmicas y exponenciales expresables con funciones especiales, y muchas más

Para calcular una integral, use la función de integrate(). Hay dos tipos de integrales, definidas e indefinidas. Para calcular una integral indefinida, es decir, una antiderivada o función primitiva, simplemente pase la variable después de la expresión.

Tenga en cuenta que SymPy no incluye la constante de integración.

[61]:
```
f = cos(x) + (x)
f
```

[61]: $x + \cos(x)$

[62]:
```
integrate(f)
```

[62]: $\frac{x^2}{2} + \sin(x)$

En el caso de que la integral sea definida solo tenemos que agregar los límites de integración. Para calcular una integral definida, pase el argumento (variable_integración, límite_inferior, límite_superior).

Ejemplo:

[63]:
```
f = 2*sqrt(x -1)
f
```

[63]: $2\sqrt{x-1}$

[64]:
```
integrate(f,(x, 1, 3))
```

[64]: $\frac{8\sqrt{2}}{3}$

[65]:
```
f = exp(-x)
f
```

[65]: e^{-x}

[66]:
```
integrate(f,(x, 0, oo))
```

[66]: 1

Las integrales simbólicas y las antiderivadas no evaluadas están representadas por la clase Integral.

El método Integral() puede devolver estos objetos si no puede calcular la integral, nos devuelve la integral simbólica o no evaluada

[67]:
```
intg = Integral(sin(x), (x, 0, pi))
intg
```

[67]: $\int_0^{\pi} \sin(x)\, dx$

De igual manera que la derivada no evaluada tenemos que para evaluar una integral podemos utilizar el método .doit()

[68]:
```
intg.doit()
```

[68]:

2

11 Límites

Los límites de una función en SymPy se obtienen con el método limit() que recibe como argumentos a la función a evaluar, la variable y el valor hacia el cual va a la tendencia del limite

ejemplo:

[69]:
```
f = (3-x)/(x**2 - 2*x - 8)
f
```

[69]:
$$\frac{3-x}{x^2-2x-8}$$

[70]:
```
limit(f, x, 4)
```

[70]:
$$-\infty$$

[71]:
```
f = (x - 9)/(sqrt(x) - 3)
f
```

[71]:
$$\frac{x-9}{\sqrt{x}-3}$$

[72]:
```
limit(f, x, 6)
```

[72]:
$$-\frac{3}{-3+\sqrt{6}}$$

[73]:
```
simplify(limit(f, x, 6))
```

[73]:
$$\sqrt{6}+3$$

[74]:
```
N(simplify(limit(f, x, 6)),10)
```

[74]:
5.449489743

limites cuanado x tiende a infinito, para definir limites a infinito solo tenemos que colocar a infinito en el tercer argumento que recibe la función y en SymPy el infinito se representa como una doble 'o' (oo) y en el caso en donde el valor tienda a menos infinito solo se agrega un - antes (-oo).

[75]:
```
f = (3*x + 1)/(2*x - 3)
f
```

[75]:
$$\frac{3x+1}{2x-3}$$

[76]:
```
limit(f , x , oo)
```

[76]: $\frac{3}{2}$

[77]:
```
f = (1 - exp(x))/(1 + exp(x))
f
```

[77]: $\frac{1 - e^x}{e^x + 1}$

[78]:
```
limit(f, x, -oo)
```

[78]: 1

También hay un cuarto parámetro opcional, para especificar la dirección de aproximación al objetivo límite. ‘‘+’’ (el valor predeterminado) da el límite desde arriba y ‘‘-’’ es desde abajo.

Ejemplo:

[79]:
```
f = 1/x
f
```

[79]: $\frac{1}{x}$

[80]:
```
limit(f, x, 0, "-")
```

[80]: $-\infty$

Al igual que las funciones de cálculo anteriores tenemos para expresar a los limites sin evaluar el método Limit() y para evaluarlo la función .doit().

[81]:
```
f = log(2/x**2)
f
```

[81]: $\log\left(\frac{2}{x^2}\right)$

[82]:
```
Limit(f, x, -oo)
```

[82]: $\lim_{x \to -\infty} \log\left(\frac{2}{x^2}\right)$

[83]:
```
Limit(f, x, -oo).doit()
```

[83]: $-\infty$

12 Taylor series

Una aproximación de la serie de Taylor es una aproximación de una función obtenida a partir de su serie Taylor. Para calcularlo, use series (expr, x, x0, n), donde x

es la variable relevante, x0 es el punto donde se realiza la expansión (por defecto es 0), y n es el orden de expansión (por defecto a 6)

[84]:
```
series(cos(x), x)
```

[84]:
$$1 - \frac{x^2}{2} + \frac{x^4}{24} + O\left(x^6\right)$$

[85]:
```
f = ln(x)
f
```

[85]:
$$\log(x)$$

[86]:
```
series(f, x, 1, 6)
```

[86]:
$$-1 - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \frac{(x-1)^5}{5} + x + O\left((x-1)^6; x \to 1\right)$$

El término O al final representa el término de orden de Landau en x = 0,Significa que se omiten todos los términos x con potencia mayor o igual que que se señala(la potencia posterior a O). Los términos del pedido se pueden crear y manipular fuera de la serie. Absorben automáticamente los términos de orden superior.

[87]:
```
f = sqrt(x)
f
```

[87]:
$$\sqrt{x}$$

[88]:
```
diff(f, x, 5)
```

[88]:
$$\frac{105}{32x^{\frac{9}{2}}}$$

[89]:
```
series(f, x, 4, 6)
```

[89]:
$$1 - \frac{(x-4)^2}{64} + \frac{(x-4)^3}{512} - \frac{5(x-4)^4}{16384} + \frac{7(x-4)^5}{131072} + \frac{x}{4} + O\left((x-4)^6; x \to 4\right)$$

13 Resolver ecuaciones

La función principal que se utiliza para resolver ecuaciones es solve (). Su interfaz es algo complicado ya que acepta muchos tipos de entradas diferentes y puede generar resultados en varias formas dependiendo de la entrada. Para ecuaciones simples en donde igualamos a 0 tenemos la siguiente sintaxis solve(expresión, variable). Esto puede resolver ecuaciones algebraicas y ecuaciones trascendentales que involucran fracciones racionales, raíces cuadradas, valores absolutos, exponenciales, logaritmos, funciones trigonométricas, etc. El resultado es entonces una lista de los valores de las variables que satisfacen la ecuación. En es caso de tener una igualdad de expresiones colocar así solve(expr A - expr B, x)

A continuacion se muestra como

[90]:
```
solve(x**2 - 1, x)
```

[90]: [-1, 1]

[91]:
```
expr_a = 2*x**2
expr_a
```

[91]: $2x^2$

[92]:
```
expr_b = 3*x
expr_b
```

[92]: $3x$

[93]:
```
solve(expr_a - expr_b, x)
```

[93]: [0, 3/2]

[ ]:
```

```

14 Algunas funciones creadas con SymPy

A continuación crearemos un par de funciones implementando los conocimientos que hemos adquirido en el nootebook utilizando el módulo de cálculo de SymPy con el objetivo de ejemplificar las oportunidades que SymPy nos ofrece, en donde yo quiero destacar que esto solo es una parte muy resumida y breve de todas la funciones y aplicaciones que SymPy tiene y que nos permite realizar, ya que con esta paquetería la cantidad de herramientas aplicaciones, funciones, métodos y uso se extiende más a de algunas funciones y aplicaciones para el cálculo.

15 Graficas de función, derivada y función primitiva

A continuación se vamos a realizar una función que va a graficar a las funciones y utilizaremos el módulo de graficas que SymPy tiene si quieres saber cómo graficar de manera rápida y sencilla "Como graficar de manera simple y facíl en SymPy".
Si quieres conocer más acerca del módulo de graficas de SymPy entra al sitio web oficial de "SymPy plotting"

[94]:
```
def visual_grafic(function, a=-10, b=10):
    '''Esta función recibe a una función matemática y a un intervalo en donde se␣
 ↪desea graficar, la función regresa las gráficas de la función, la derivada de␣
 ↪la función y la función primitiva o anti-derivada de la función. \n \n␣
 ↪visual_grafic(funcion, limite de graficación a, limite de graficación b) '''
    print((function))

    dx = diff(function, x)
```

```
    print(dx)
    dx
    integ = (integrate(function, x))
    print(integ)

    plot(function, (x, a, b),line_color='cyan', title='Función' )
    plot(dx, (x, a, b), line_color='purple', title='Derivada de la función')
    plot(integ, (x, a, b), line_color='pink', title='Función primitiva')
```

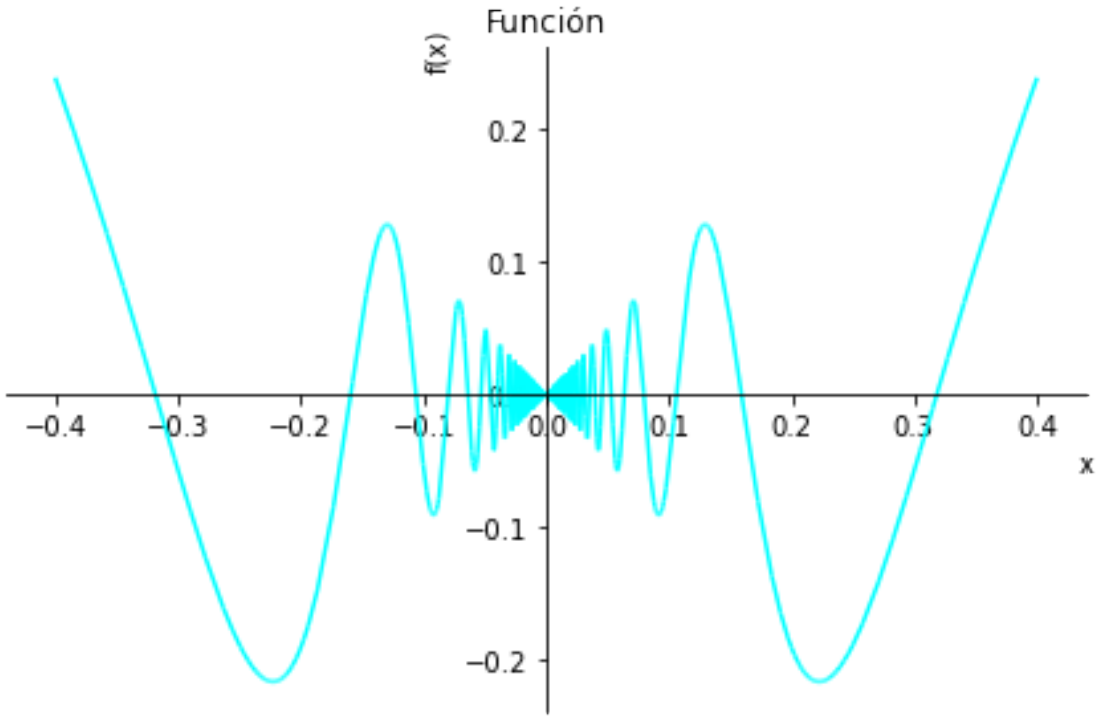
[95]:
```
visual_grafic?
```

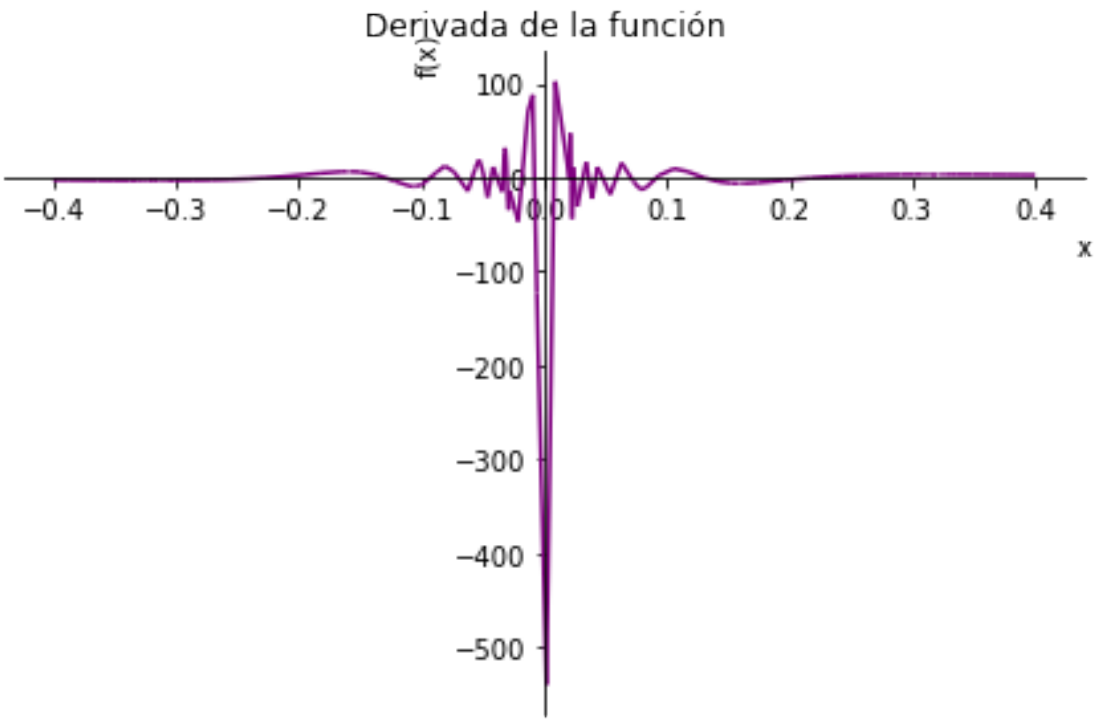
[96]:
```
f = x*sin(1/x)
f
```

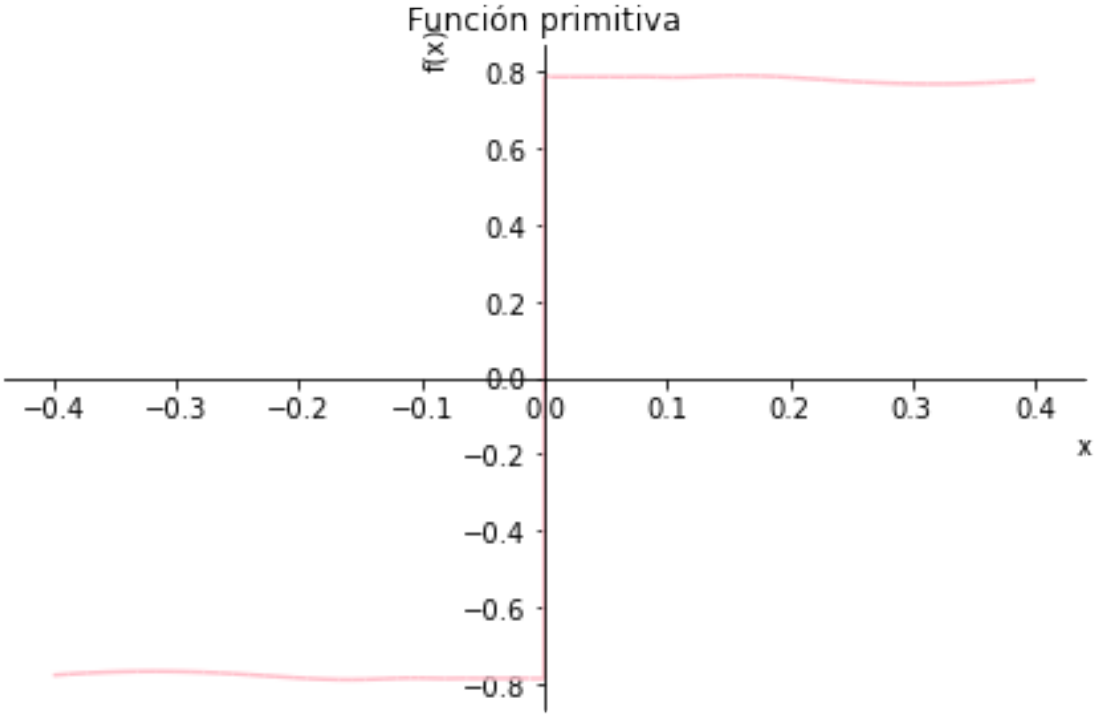
[96]:
$$x \sin\left(\frac{1}{x}\right)$$

[105]:
```
visual_grafic(f, -0.4, 0.4)
```

```
x*sin(1/x)
sin(1/x) - cos(1/x)/x
x**2*sin(1/x)/2 + x*cos(1/x)/2 + Si(1/x)/2
```




Derivada de la función
f(x)
100
0
−100
−200
−300
−400
−500
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
x


Función primitiva
f(x)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
x

16 Calculo de área entre dos funciones

En esta función vamos a calcular el área que se limita por la curva de dos funciones en donde la función va a recibir 2 argumentos la función o expresión matemática inferior y la función o expresión matemática superior.

Esta función recibe dos expresiones o funciones creadas en sympy y nos regresa los puntos en donde se intersectan las funciones y el área entre las dos funciones.

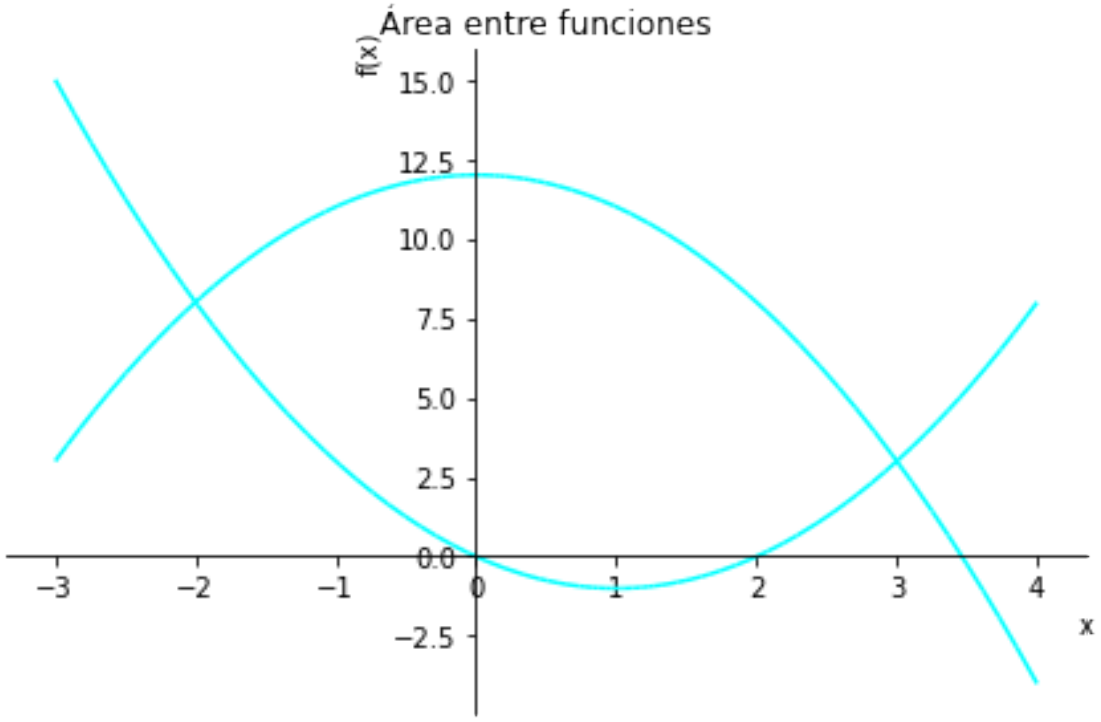
[98]:
```
def area_entre_funciones(fi, fs):
    '''Esta función recibe dos expresiones o funciones creadas en sympy y nos␣
↪regresa los puntos en donde se intersectan las funciones y el área entre las␣
↪dos funciones '''
    #fis = fi - fs
    #print(fis)
    puntos = solve(fi - fs, x)
    print('Los puntos donde se intersectan las funciones son:', puntos)
    a = puntos[0]
    b = puntos[1]

    A = integrate(fi - fs, (x, a, b))
    #print(A)
    #f = A.subs(x, a)
    at = N(simplify(A), 10)
    abs(at)
    #print(at)
    if at > 0:
        print(at)
        #print('el valor llega aqui')
        plot(fi, fs, (x, a-1, b+1), line_color='cyan', title='Área entre␣
↪funciones')

    elif at < 0:
        nat = at*(-1)
        print('El valor de el área señalada es igual a', nat)
        #print('el valor llega aqui')
        plot(fi, fs, (x, a-1, b+1), line_color='cyan', title='Área entre␣
↪funciones')


    #at = positive=true
    #print(at)
    #print(f)
    #g = A.subs(x, b)
    #print(g)
    #plot(fi, fs, (x, a-1, b+1), line_color='cyan', title='Área entre funciones')
```

[99]:
```
fi = x**2 - 2*x
fs = 12 - x**2
```

[100]:
```
area_entre_funciones?
```

[101]:
```
area_entre_funciones(fi, fs)
```

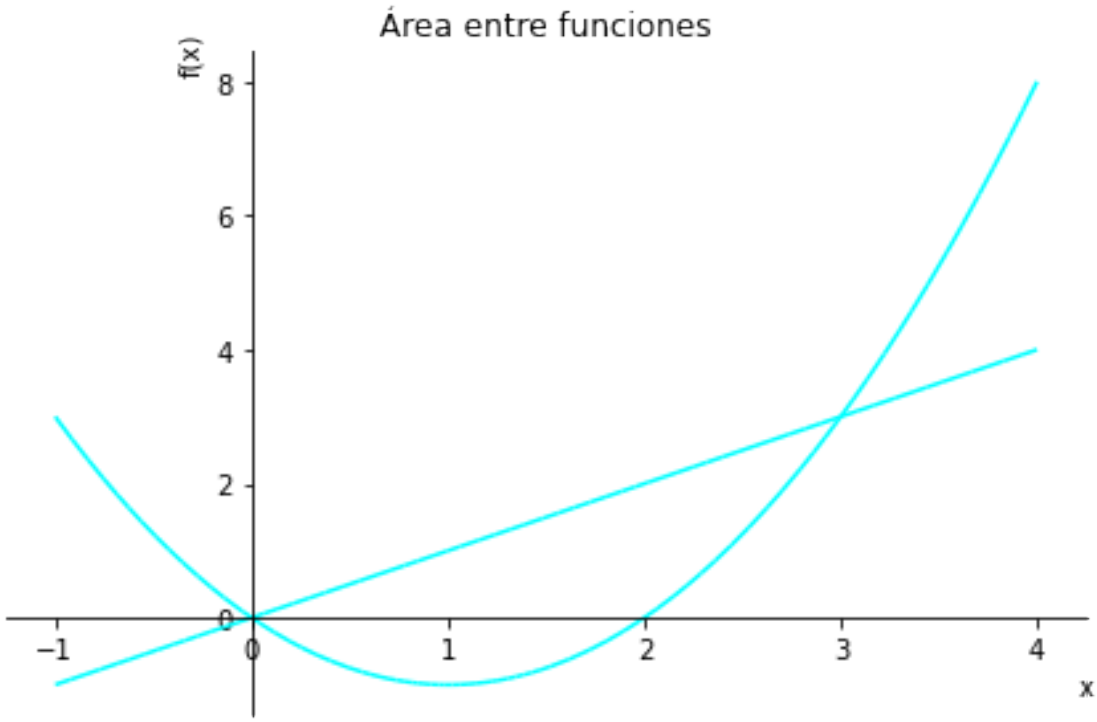
```
Los puntos donde se intersectan las funciones son: [-2, 3]
El valor de el área señalada es igual a 41.66666667
```



[102]:
```
fi = x**2 - 2*x
fs = x
```

[103]:
```
area_entre_funciones(fi, fs)
```

```
Los puntos donde se intersectan las funciones son: [0, 3]
El valor de el área señalada es igual a 4.500000000
```



Si quieres conocer más acerca de SymPy entra en su página web y ahí podrás encontrar toda la información sobre SymPy y también dentro de su página se encuentra toda la documentación y está dividida y estructurada por temas y subtemas así como sus funciones y aplicaciones.

Pagína web SymPy

17 Bibliografía

- Ronan Lamy, R. L. (2013). Instant SymPy Starter (1.a ed., Vol. 1). http://www.packtpub.com/. https://www.packtpub.com/big-data-and-business-intelligence/inst
- Stewart, J. M. (2014). Python for Scientists (Segunda Edición, Vol. 1). Department of Applied Mathematics & Theoretical Physics University of Cambridge. https://doi.org/10.1017/9781108120241
- Welcome to SymPy's documentation! -- SymPy 1.7.1 documentation. (s. f.). SymPy. Recuperado 16 de enero de 2021, de https://docs.sympy.org/latest/index.html

Elaboró Hernández Martínez Erick Daniel
Github: Erickkhm

Correo electrónico: erickhernandez@encit.unam.mx

Licenciatura en ciencias de la tierra (ENCiT)

Curso de Herramientas Computacioneales