practica2_funciones_con_sympy

October 23, 2022

Funciones simbólicas con Sympy:

(i) cálculo de límites, derivadas, ...

(ii) solución de ecuaciones

(iii) gráficas de funciones

Para trabajar con sympy

Paso 1.- Cargar paquete sympy, y comandos básicos

Paso 2.- Definir símbolos x,... y función f= ...

Paso 3.- usar comandos limit, diff, solve, plot, etc...

[1]:
```
import sympy as sp
from sympy import symbols, limit, diff, pi, sin, cos, log, exp, sqrt


x=symbols('x')
f=sin(x)/x
f
```

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

A. Cálculo de límites: usar el comando limit...

[2]:
```
x=symbols('x')
f=sin(x)/x

limit(f,x,0)
```

[2]:
1

también pueden calcularse límites laterales, por ejemplo de la función parte entera (floor)

[3]:
```
f=sp.floor(x)

print(limit(f, x, 10, '+'))
limit(f, x, 10, '-')
```

```
10
```

[3]:
9

[4]:
```
f=x*sp.floor(1/x)

print(limit(f, x, 0, '+'))
print(limit(f, x, 0, '-'))
```

```
1
1
```

[5]:
```
# para limites en infty

from sympy import oo as infty

f=exp(-x)

display(f)
print(limit(f, x, infty))
limit(1/f, x, infty)

# Nota: el comando "display" es similar a "print", pero muestra el resultado␣
 ↪con texto de fórmula matemática
```

e^{-x}

```
0
```

[5]:
∞

B. Cálculo de derivadas: usar comando diff

[6]:
```
x=symbols('x')
f=sin(x)/x

diff(f)
```

[6]:
$$\frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$$

se pueden calcular derivadas de cualquier orden

[7]:
```
f=sin(x)/x
g=diff(f,x, 4)

display(g)
display(sp.simplify(g))
g.subs(x,pi/2)

# Nota: el comando "simplify" permite simplificar la expresión
```

$$\frac{\sin(x) + \frac{4\cos(x)}{x} - \frac{12\sin(x)}{x^2} - \frac{24\cos(x)}{x^3} + \frac{24\sin(x)}{x^4}}{x}$$

$$\frac{x^4\sin(x) + 4x^3\cos(x) - 12x^2\sin(x) - 24x\cos(x) + 24\sin(x)}{x^5}$$

[7]:
$$\frac{2\left(-\frac{48}{\pi^2} + 1 + \frac{384}{\pi^4}\right)}{\pi}$$

C. Resolver ecuaciones (simbólicas): usar solve

[8]:
```
from sympy import solve

x=symbols('x')
f=x**2+x-1

solve(f)
```

[8]:
```
[-1/2 + sqrt(5)/2, -sqrt(5)/2 - 1/2]
```

[9]:
```
x=symbols('x')

f=7*exp(-x)
g=2**x

solve(f-g)
```

[9]:
```
[log(7)/(log(2) + 1)]
```

[10]:
```
# A veces las raíces son números complejos

x=symbols('x')
f=x**2+4

solve(f)
```

[10]:
```
[-2*I, 2*I]
```

[11]:
```
x=symbols('x')

f=exp(x)+1

solve(f)
```

[11]:
```
[I*pi]
```

[12]:
```
# A veces dsolve no sabe encontrar la solución simbólica, o la indica en␣
 ↪términos de funciones "especiales".
# En esos casos hay buscar soluciones numéricas...
```

```
x=symbols('x')
f=exp(-x)-x

solve(f)
```

[12]: [LambertW(1)]

Para resolver ecuaciones numéricas: se puede usar nsolve(f, x0), donde x0 es un valor aproximado de la raíz (que se visualiza dibujando la gráfica)…

[13]:
```
from sympy import nsolve

x=symbols('x')
f=exp(-x)-x

nsolve(f, 1)
```

[13]: 0.567143290409784

[14]:
```
from sympy import nsolve

x=symbols('x')
f=sin(x)-x/3

nsolve(f,3.14)
```

[14]: 2.27886266007583

[15]:
```
# se puede añadir más decimales a la solución...

nsolve(f, 3.14, prec=50)
```

[15]: 2.2788626600758283126999511045618886288182747407398

Nota: Para resolver ecuaciones numéricas, es más común usar el paquete scipy.optimize y los comandos fsolve, bisect, etc… Abajo pongo algunos ejemplos

[16]:
```
import numpy as np
from scipy.optimize import fsolve

def f(x):
    return np.sin(x)-x/3

fsolve(f, 3)
```

[16]: array([2.27886266])

[17]:
```
import numpy as np
from scipy.optimize import bisect
```

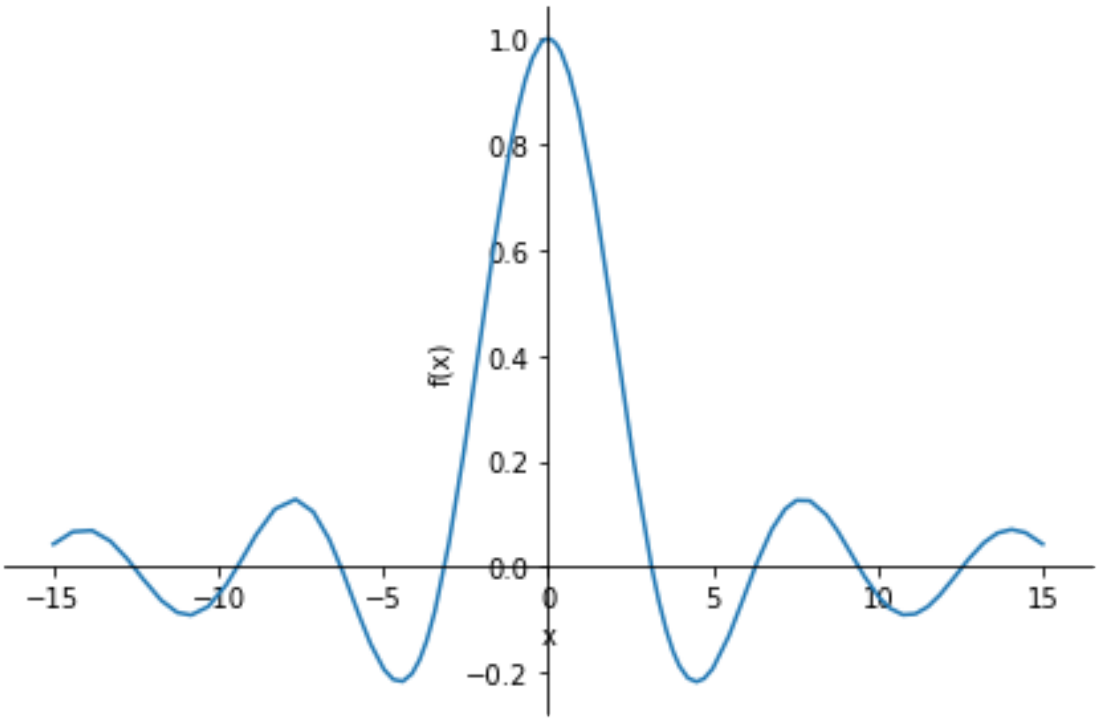
```
def f(x):
    return np.sin(x)-x/3

display(bisect(f, 1,3))
bisect(f, 1,3, xtol=10**(-5))
```

```
2.2788626600759017
```

[17]:
```
2.2788619995117188
```

D. Gráficas con sympy

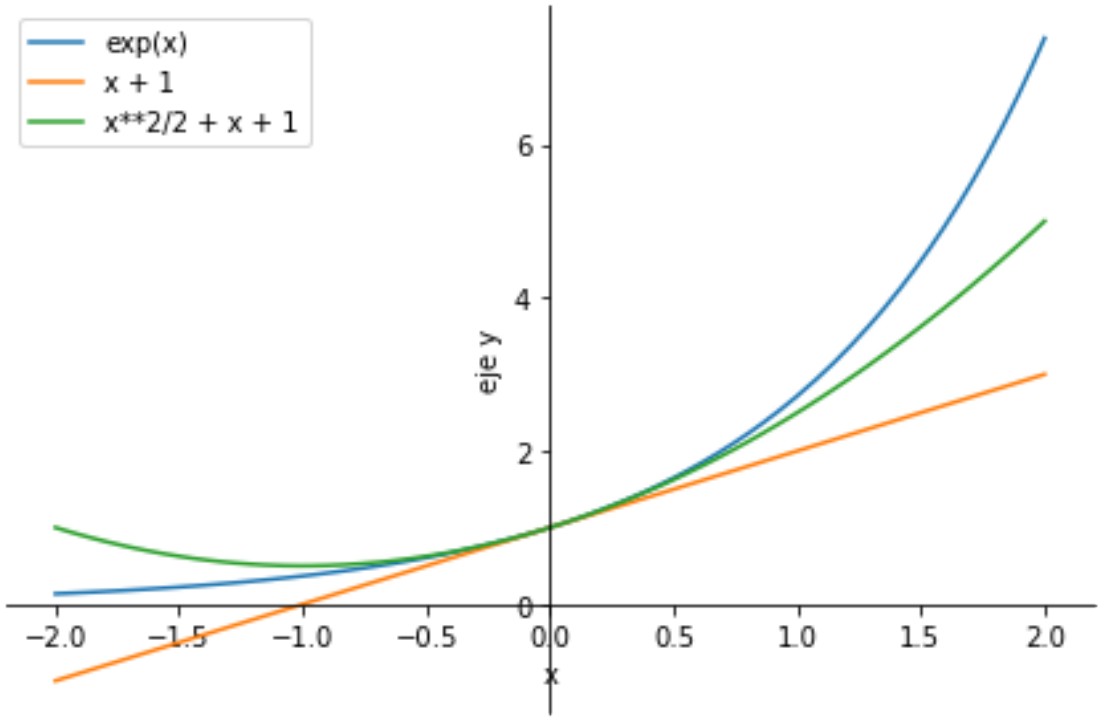
[18]:
```
from sympy.plotting import plot

x=symbols('x')
f=sin(x)/x

plot(f, (x, -15,15) )
```



[18]:
```
<sympy.plotting.plot.Plot at 0x216957cffa0>
```

[43]:
```
from sympy.plotting import plot
```

```
x=symbols('x')

f=exp(x)
g1=1+x
g2=1+x+x**2/2

plot(f,g1, g2, (x, -2,2) , ylabel='eje y', legend=True)
```

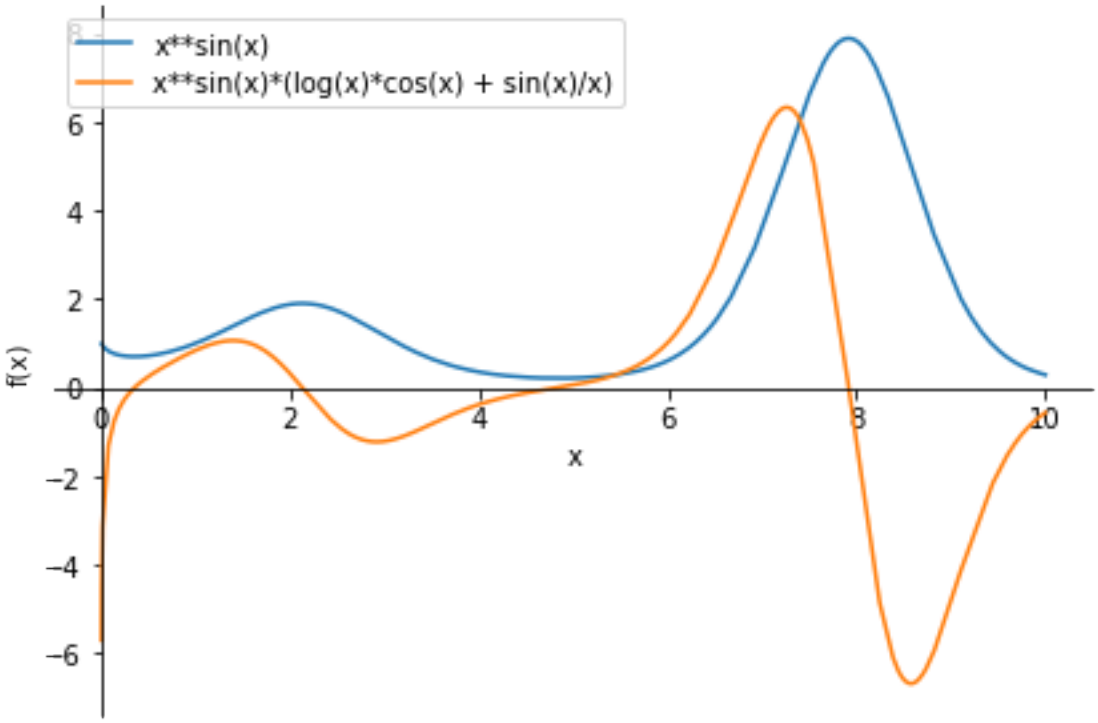


[43]: <sympy.plotting.plot.Plot at 0x2169c241460>

[75]:
```
import sympy as sp
from sympy import symbols, diff, sin

x=symbols('x')
f=x**sin(x)
display(f)

df=diff(f,x)
display(df)

plot(f, df, legend=True)
```

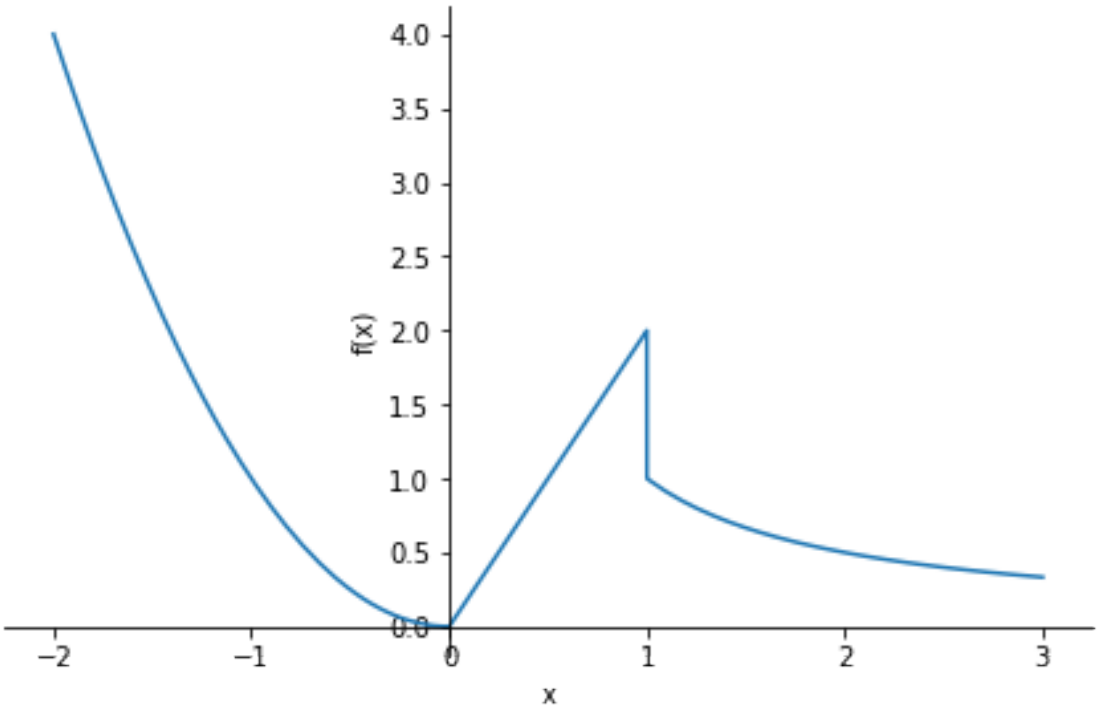
$x^{\sin(x)}$

$$x^{\sin(x)}\left(\log(x)\cos(x)+\frac{\sin(x)}{x}\right)$$



[75]: <sympy.plotting.plot.Plot at 0x216a019cee0>

[44]:
```
# un ejemplo de función definida a trozos: usar comando "Piecewise"

from sympy.plotting import plot
from sympy import Piecewise

x=symbols('x')

f1 = x**2
f2 = 2*x
f3= 1/x

f = Piecewise((f1, x <= 0), (f2, x < 1), (f3, True))

plot(f, (x, -2,3) )
```



[44]: `<sympy.plotting.plot.Plot at 0x2169c120a00>`

Nota: Aunque sympy.plot dibuja correctamente funciones simbólicas, en general es recomendable utilizar los paquetes matplotlib.pyplot y numpy, que son más eficientes en las operaciones numéricas, y ya son familiares de las otras prácticas …

E.- Gráficas con matplotlib.

Paso 1: definir la función

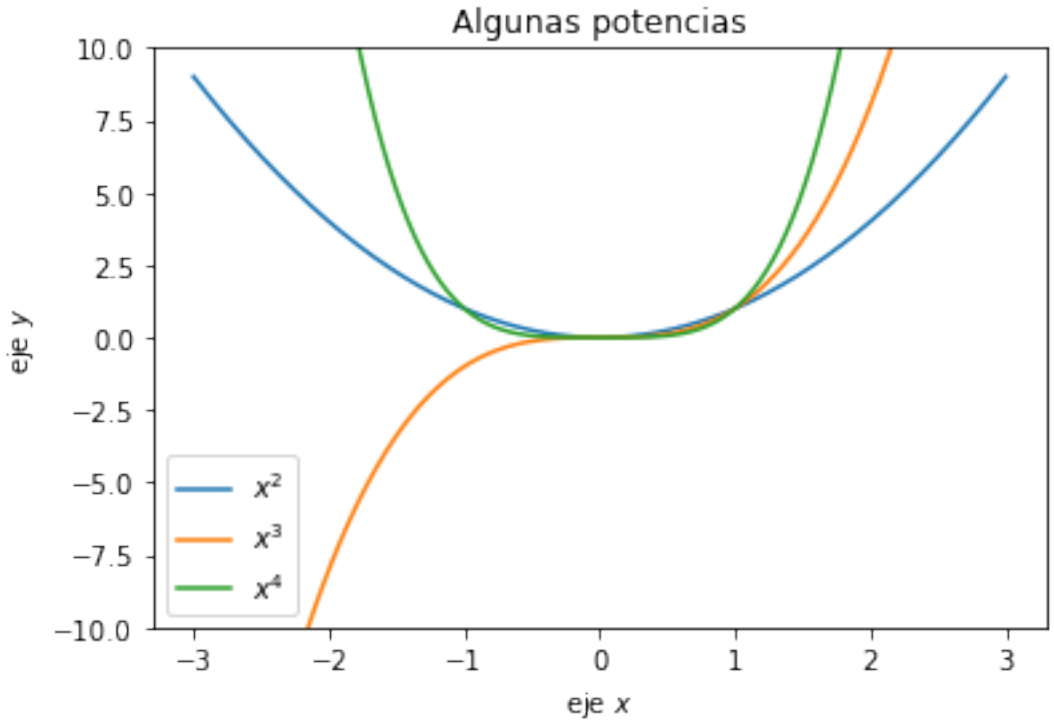
Paso 2: definir una lista de valores de x (con np.linspace o np.arange)

Paso 3: dibujar la pareja (x, f(x)) usando plt.plot

[88]:
```
import matplotlib.pyplot as plt
import numpy as np

def f(x,n):
    return x**n

# Plot de funciones numericas

x = np.linspace(-3, 3, 1000)
plt.plot(x, f(x,2), x, f(x,3), x, f(x,4))
plt.ylim([-10,10])
```

```
# adornos

plt.title("Algunas potencias")
plt.xlabel("eje $x$")
plt.ylabel("eje $y$")
plt.legend(["$x^2$", "$x^3$", "$x^4$"])
```

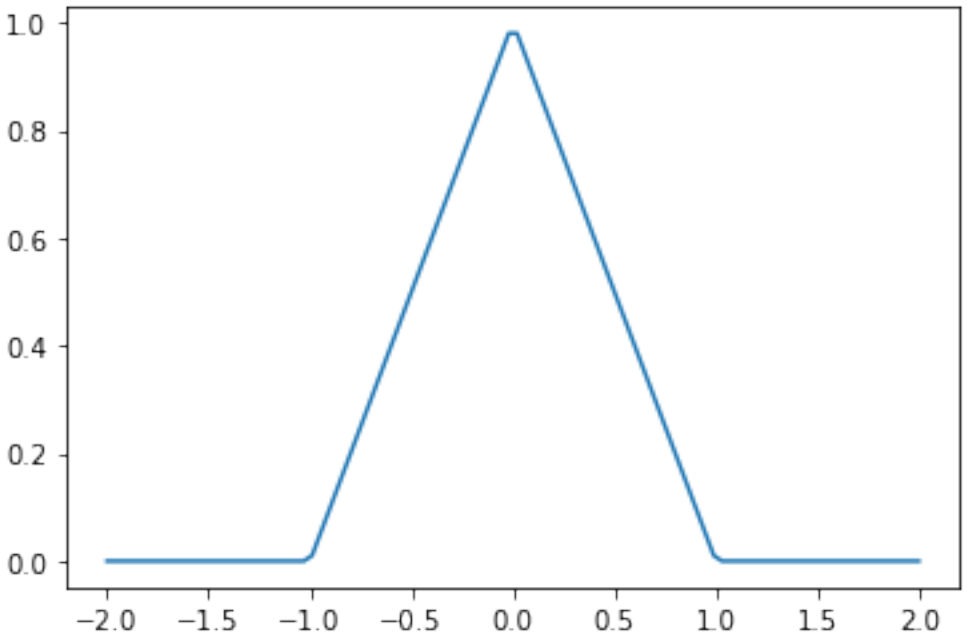
[88]: <matplotlib.legend.Legend at 0x216a0345be0>



[54]:
```
# Ejemplo de función a trozos.
# Opción 1: definir usando if... else...


import matplotlib.pyplot as plt
import numpy as np

def ramp(x):
    if 0<x<=1:
        ramp=1-x
    elif -1<x<=0:
        ramp=1+x
    else:
        ramp=0
```

```
    return ramp

x=np.linspace(-2,2,100)

# ramp(x) sólo tiene sentido si x es un número, no una lista
# para poder dibujar ramp(x), debemos creamos una lista "y" con los valores de␣
 ↪ramp en x...

y = []
for i in x:
    y.append(ramp(i))

plt.plot(x,y)
```

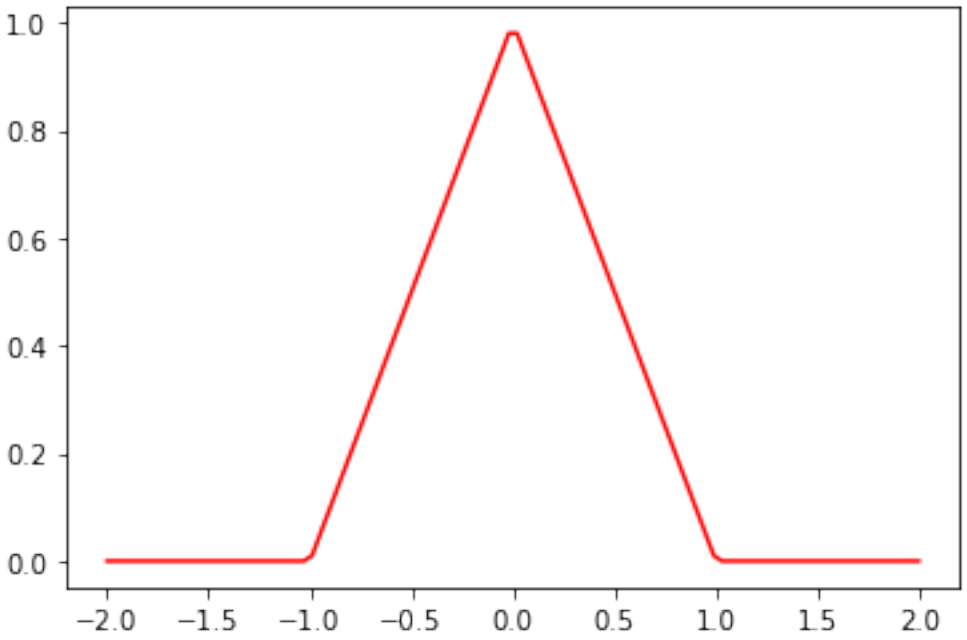
[54]: [<matplotlib.lines.Line2D at 0x2169c5024c0>]



[56]:
```
# Opción 2: usar el comando np.piecewise

import matplotlib.pyplot as plt
import numpy as np


x=np.linspace(-2,2,100)
```

```
y=np.piecewise(x, [x <= -1, (x>-1) & (x <= 0), (x>0) & (x<=1), x>1], [0, lambda␣
 ↪x: 1+x, lambda x: 1-x, 0])


plt.plot(x,y, 'r-')
```

[56]: [<matplotlib.lines.Line2D at 0x2169c57ffa0>]

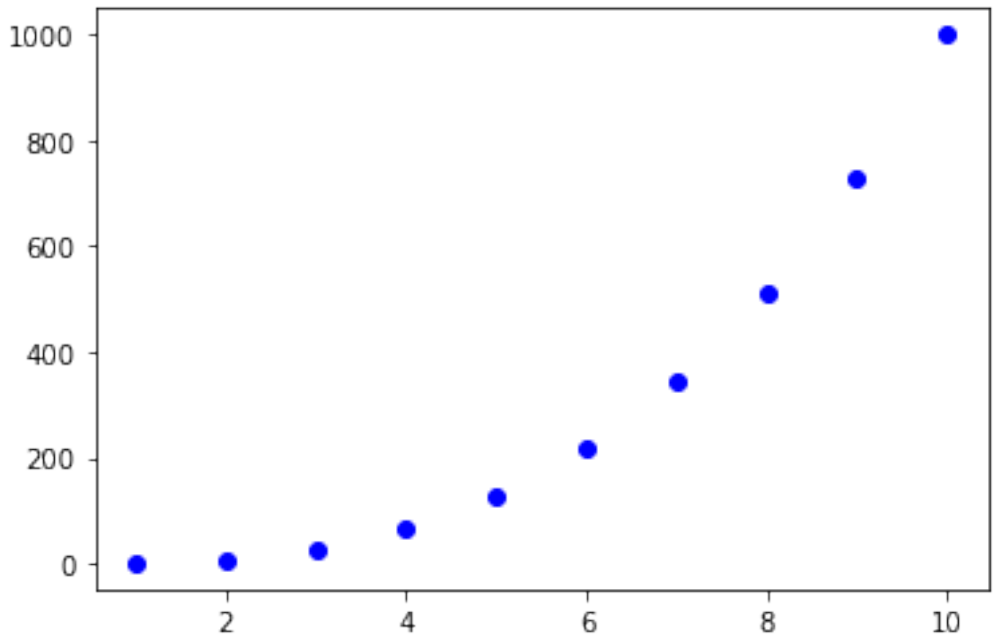


Nota sobre "funciones lambda": es una forma de definir funciones "anónimas" a las que no se asigna un nombre específico, generalmente porque aparecen dentro de otras expresiones donde sólo juegan un carácter auxiliar. Un ejemplo es el uso de np.piecewise arriba indicado...

[103]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import fsolve

cubo = lambda x: x**3
print(cubo(3))

x=np.linspace(1,10,10)
plt.plot(x, cubo(x), 'bo')
```

27

[103]: [<matplotlib.lines.Line2D at 0x216a1a8cbb0>]



NOTA: SYMPY vs NUMPY

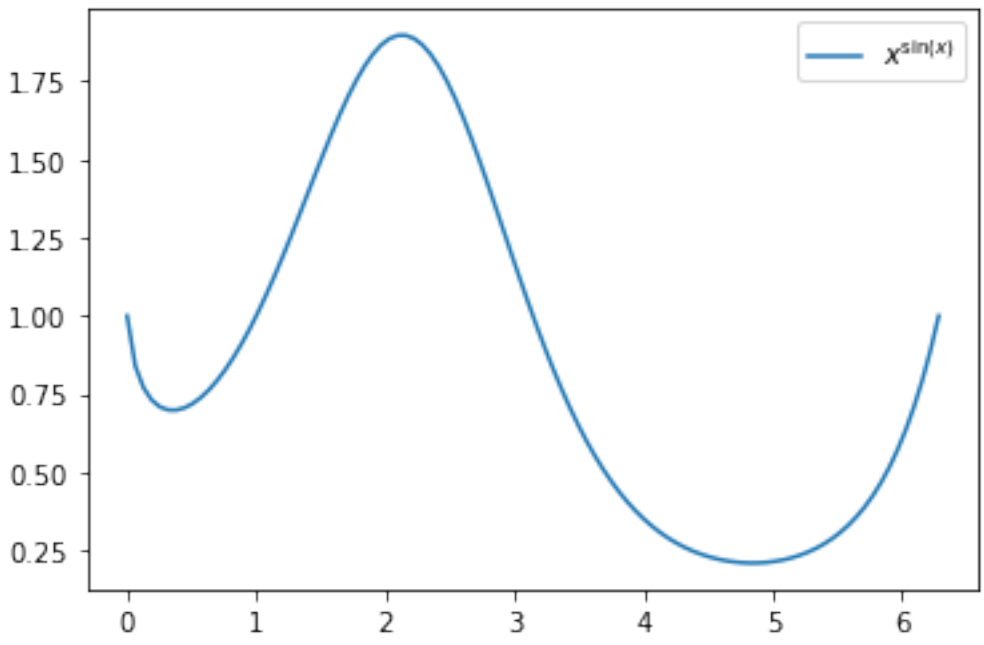
En general, NO se puede trabajar numéricamente en numpy, con funciones definidas en sympy. Para hacerlo, es necesario transformarlas previamente con el comando "lambdify"

[81]:
```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

from sympy import sin, cos, pi, symbols, lambdify

x = symbols('x')
f = x**sin(x)

display(f)

F = lambdify(x, f, 'numpy')

x = np.linspace(0,2*np.pi,100)

display(F(x))

plt.plot(x,F(x))
plt.legend([r'$x^{\sin(x)}$'])
```

$x^{\sin(x)}$

```
array([1.        , 0.83956184, 0.7700507 , 0.73059363, 0.70870853,
       0.69896577, 0.69849036, 0.7056001 , 0.7192511 , 0.73876995,
       0.7637077 , 0.79375206, 0.82866985, 0.868266  , 0.91235216,
       0.96072113, 1.01312522, 1.06925753, 1.12873595, 1.19109008,
       1.25575144, 1.32204753, 1.38920064, 1.45633182, 1.5224706 ,
       1.58657076, 1.64753208, 1.70422757, 1.75553551, 1.80037476,
       1.83774196, 1.86674845, 1.88665501, 1.89690209, 1.89713385,
       1.88721411, 1.86723337, 1.83750613, 1.79855879, 1.75110873,
       1.696036  , 1.63434928, 1.56714827, 1.49558458, 1.42082332,
       1.34400733, 1.26622546, 1.18848631, 1.11169787, 1.03665357,
       0.96402429, 0.89435616, 0.82807312, 0.76548359, 0.70679005,
       0.65210082, 0.60144296, 0.55477561, 0.51200315, 0.4729876 ,
       0.43755998, 0.40553048, 0.37669714, 0.35085325, 0.32779336,
       0.30731814, 0.28923807, 0.2733763 , 0.25957066, 0.24767511,
       0.23756073, 0.22911631, 0.2222488 , 0.21688363, 0.21296497,
       0.21045612, 0.20934002, 0.20961994, 0.21132044, 0.2144887 ,
       0.21919613, 0.22554055, 0.2336487 , 0.24367938, 0.25582718,
       0.27032676, 0.28745789, 0.30755108, 0.33099397, 0.35823826,
       0.38980729, 0.42630393, 0.46841866, 0.51693743, 0.57274885,
       0.63685006, 0.71035037, 0.79447182, 0.89054516, 1.        ])
```

[81]: <matplotlib.legend.Legend at 0x216a04ac6d0>



[ ]: