Sympy Introduction

September 8, 2023

[1]:
```
# Notebook Preamble
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp

plt.style.use('../maroon_ipynb.mplstyle')
```

Contents

1 Introduction **3**

2 General Functionality **3**
- 2.1 Defining Symbols 3
- 2.2 Substitution 4
- 2.3 Converting from Sympy to Python Function 5

3 Solving Systems of Equations **5**

4 Calculus **7**
- 4.1 Differentiation 7
- 4.2 Integration 7

5 Differential Equations **8**
- 5.1 Solving ODE's 8
- 5.2 Laplace Transforms 10

6 Linear Algebra **12**

1 Introduction

- The sympy package is a Computer Algebra System (CAS), which has the capabilities of solving math problems symbolically.
- It essentially is an alternative to mathematica and is useful for showing mathematical procedures, especially in jupyter notebook.
- However, it is very slow in some applications; therefore, it is usually not good to use it for automation. Instead, use numerical methods from the numpy and scipy packages when doing math behind the scenes.

2 General Functionality

- sympy primarily works by defining symbols and functions of some variable, then doing some operation on them.

2.1 Defining Symbols

Example: Define the following symbols and functions:

- Symbols: $x, \ t, \ s, \ \dot{x}$
- Functions: $x(t), \ x(t,s), \ X(s)$

[2]:
```
# Use the sp.Symbol() to define a symbol one at a time.
# Use sp.symbols() define multiple symbols at one time
x, t, s, x_dot = sp.symbols(r'x t s \dot{x}')
display(x, t, s, x_dot)
```

x

t

s

$\dot{x}$

[3]:
```
# For functions, we create instances of the sp.Function() class
x_t, x_ts, X = sp.Function('x')(t), sp.Function('x')(t, s), sp.Function('X')(s)
display(x_t, x_ts, X)
```

$x(t)$

$x(t,s)$

$X(s)$

- You define the function, then determine what it is a function of by passing in the parameters in the function call.

2.2 Substitution

Example: Create an expression for $x^2y + 5yx + 10$, then substitute values $x = 5,\ y = 3$.

[4]:
```
x, y = sp.symbols('x y')
expr = x**2*y + 5*x*y + 10
expr
```

[4]:

$x^2y + 5xy + 10$

- You use the `.subs` method for substituting in parameters.

[5]:
```
# If multiple arguments, it takes a list of tuples.
expr.subs([
    (x, 5),
    (y, 3)
])
```

[5]: 160

[6]:
```
# If you wanted to show substitutions without simplifying, this is what you␣
↪would do
with sp.evaluate(False):
    # Everything within this context manager will not simplify
    expr_subs = expr.subs([
        (x, 5),
        (y, 3)
    ])
display(expr_subs, expr_subs.simplify())
```

$10 + 3 \cdot 5^2 + 5 \cdot 5 \cdot 3$

160

[7]:
```
# If you wanted to show it as an equation
with sp.evaluate(False):
    expr_subs = expr.subs([
        (x, 5),
        (y, 3)
    ])

# This is called the Equality class (think of it as an equation class too)
# If sympy sees that there are no symbols, it will try to evaluate it as true␣
↪or false
sp.Eq(expr_subs, expr_subs.simplify(), evaluate=False)
```

[7]:

$10 + 3 \cdot 5^2 + 5 \cdot 5 \cdot 3 = 160$

2.3 Converting from Sympy to Python Function

- The `.subs` method is good for showing basic substitutions, but if you needed to perform many different substitutions (like you would when you are plotting points), then you need to *lambdify* the expression.
- This just means that we need to convert it from a `sympy` object to python function for fast computation.

Example: Convert the `sympy` expression $f(x) = x^2$ into a python function.

[8]:
```
f = x**2  # the variable x was defined above
f
```

[8]: x^2

[9]:
```
# Use sp.lambdify() to generate the new function
f_lamb = sp.lambdify(x, f, modules='numpy')  # modules='numpy' tells it to use␣
 ↪numpy functions if necessary
f_lamb
```

[9]:
```
<function _lambdifygenerated(x)>
```

[10]:
```
# f_lamb is essentially equivalent to the following function
# def f_lamb(x):
#     return x**2

# Now we can use it like a normal python function
f_lamb(5)
```

[10]:
```
25
```

[11]:
```
x_values = np.array([1, 2, 3, 4])
f_lamb(x_values)
```

[11]:
```
array([ 1,  4,  9, 16])
```

---

3 Solving Systems of Equations

- Systems can be solved both symbolically and numerically if needed.

Example: Solve the following system for x and y:

$$\begin{cases} xy + 3y + a = 7 \\ y + 5x = 2 \end{cases}$$

[12]:
```python
x, y, a = sp.symbols('x y a')
eq1 = sp.Eq(x*y + 3*y + a, 7)
eq2 = sp.Eq(y + 5*x, 2)
display(eq1, eq2)
```

$a + xy + 3y = 7$

$5x + y = 2$

[13]:
```python
sol = sp.solve([eq1, eq2], (x, y), dict=True)
sol
```

[13]:
```
[{x: -sqrt(20*a + 149)/10 - 13/10, y: sqrt(20*a + 149)/2 + 17/2},
 {x: sqrt(20*a + 149)/10 - 13/10, y: 17/2 - sqrt(20*a + 149)/2}]
```

- Specifying `dict=True` returns a list of dictionaries where the keys are the variable and the value is the solution.

[14]:
```python
for d in sol:
    for key, value in d.items():
        display(sp.Eq(key, value))
```

$$x = -\frac{\sqrt{20a + 149}}{10} - \frac{13}{10}$$

$$y = \frac{\sqrt{20a + 149}}{2} + \frac{17}{2}$$

$$x = \frac{\sqrt{20a + 149}}{10} - \frac{13}{10}$$

$$y = \frac{17}{2} - \frac{\sqrt{20a + 149}}{2}$$

- You can check the solution by substituting it, then simplifying the expression.

[15]:
```python
# The .lhs method returns the left hand side of the equation
check = eq1.lhs.subs([
    (x, sol[0][x]),
    (y, sol[0][y])
])
check
```

[15]:
$$a + \frac{3\sqrt{20a + 149}}{2} + \left(-\frac{\sqrt{20a + 149}}{10} - \frac{13}{10}\right)\left(\frac{\sqrt{20a + 149}}{2} + \frac{17}{2}\right) + \frac{51}{2}$$

[16]:
```python
check.simplify()
```

[16]:
7

Example: The following equation cannot be solved algebraically. Solve using numerical methods.

$$e^x + x = 3$$

[17]:
```
eq = sp.Eq(sp.exp(x) + x, 3)
eq
```

[17]: $x + e^x = 3$

[18]:
```
sol = sp.nsolve(eq, x, 1)  # equation, variable, guess
sol
```

[18]: 0.792059968430677

---

4 Calculus

4.1 Differentiation

Example: Find the first and second order derivative with respect to x of

$$f(x) = x^3 + 3xy + x^2$$

[19]:
```
f = x**3 + 3*x*y + x**2
f
```

[19]: $x^3 + x^2 + 3xy$

[20]:
```
f.diff(x)
```

[20]: $3x^2 + 2x + 3y$

[21]:
```
# For second order derivative:
f.diff(x, 2)
```

[21]: $2 \cdot (3x + 1)$

4.2 Integration

Example: Find $\int \ln(x)dx$

[22]:
```
integral = sp.Integral(sp.log(x), x)
integral
```

[22]: $\int \log(x)\, dx$

- Note that the $\log(x)$ function is equivalent to the $\ln(x)$ function in `sympy`.
- The above example shows the `Integral` class, but you can evaluate it by calling the `.doit()` method. This way of doing things may be desired for making sure that you set it up appropriately. The same concept can be done for other operations like the `Derivative` class.

[23]:
```
integral.doit()
```

[23]:
$x \log(x) - x$

[24]:
```
# Alternatively, you can use the integrate() method
(sp.log(x)).integrate(x)
```

[24]:
$x \log(x) - x$

---

5 Differential Equations

5.1 Solving ODE's

Example: Solve $y'' + y = \tan(x)$

[25]:
```
y = sp.Function('y')(x)
eq = sp.Eq(y.diff(x, 2) + y, sp.tan(x))
eq
```

[25]:
$y(x) + \frac{d^2}{dx^2} y(x) = \tan(x)$

[26]:
```
sol = sp.dsolve(eq)
sol
```

[26]:
$y(x) = C_2 \sin(x) + \left(C_1 + \frac{\log(\sin(x) - 1)}{2} - \frac{\log(\sin(x) + 1)}{2}\right) \cos(x)$

[27]:
```
# Checking solution
check = sol.rhs.diff(x, 2) + sol.rhs
check.simplify()
```

[27]:
$\tan(x)$

Example: Solve the system of ODE's with $x(0) = 0$ and $y(0) = 1$:

$$\begin{cases} \frac{dx}{dt} = -x + y \\ \frac{dy}{dt} = 2x \end{cases}$$

[28]:
```
t = sp.Symbol('t')
x, y = sp.Function('x')(t), sp.Function('y')(t)
```

```python
eq1 = sp.Eq(x.diff(), -x + y)
eq2 = sp.Eq(y.diff(), 2*x)
display(eq1, eq2)
```

$$\frac{d}{dt}x(t) = -x(t) + y(t)$$

$$\frac{d}{dt}y(t) = 2x(t)$$

[29]:
```python
sol = sp.dsolve([eq1, eq2], ics={
    x.subs(t, 0): 0,
    y.subs(t, 0): 1
})
sol
```

[29]: `[Eq(x(t), exp(t)/3 - exp(-2*t)/3), Eq(y(t), 2*exp(t)/3 + exp(-2*t)/3)]`

[30]:
```python
display(*sol)  # unpacking is the same as display(sol[0], sol[1])
```

$$x(t) = \frac{e^t}{3} - \frac{e^{-2t}}{3}$$

$$y(t) = \frac{2e^t}{3} + \frac{e^{-2t}}{3}$$

Example: Solve $y'' - 10y' + 25y = 30x + 3$ with $y(0) = 1$ and $y'(0) = 3$ and plot the function by `lambdifying` the solution.

[31]:
```python
x = sp.Symbol('x')  # re-defining as symbol because it was previously defined
 ↪as a function
y = sp.Function('y')(x)
eq = sp.Eq(y.diff(x, 2) - 10*y.diff() + 25*y, 30*x + 3)
eq
```

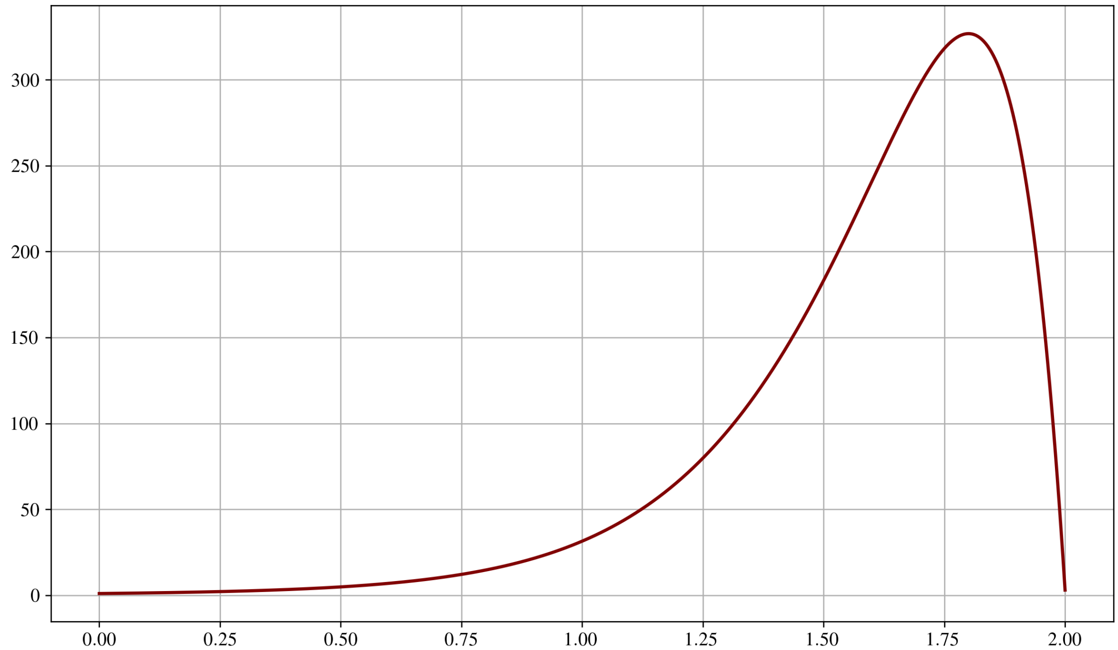
[31]:
$$25y(x) - 10\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 30x + 3$$

[32]:
```python
sol = sp.dsolve(eq, ics={
    y.subs(x, 0): 1,
    y.diff().subs(x, 0): 3
})
sol
```

[32]:
$$y(x) = \frac{6x}{5} + \left(\frac{2}{5} - \frac{x}{5}\right)e^{5x} + \frac{3}{5}$$

[33]:
```python
y_lamb = sp.lambdify(x, sol.rhs, modules='numpy')
t_ = np.linspace(0, 2, 500)  # array from 0 to 2 with a size of 500
```

```
plt.plot(t_, y_lamb(t_))
plt.show()
```



5.2 Laplace Transforms

- Laplace transforms in `sympy` as of version 1.12 are lacking. A re-design of this part of the package is coming in a later version as seen here.

Example: Find the laplace transform of $f(t) = 2\cos(5t)$.

[34]:
```
s, t = sp.symbols('s t')
sp.laplace_transform(2*sp.cos(5*t), t, s)[0]
```

[34]: $\frac{2s}{s^2 + 25}$

Example: Solve the following ODE using laplace transforms:

$$\ddot{x} + 20\dot{x} + 1000 = \begin{cases} t & 0 \leq t < 1 \\ 1 & t \geq 1 \end{cases}$$

The initial conditions are zero.

[35]:
```python
# sympy cannot do laplace transforms of piecewise functions yet, but that is in
↪the works
# Instead, use the answer that was found by hand in class
X = sp.Function('X')(s)
eq = sp.Eq(s**2*X + 20*s*X + 1000*X, 1/s**2 - 1/s**2*sp.exp(-s))
eq
```

[35]:
$$s^2X(s) + 20sX(s) + 1000X(s) = \frac{1}{s^2} - \frac{e^{-s}}{s^2}$$

[36]:
```python
sol = sp.solve(eq, X)[0]
sol
```

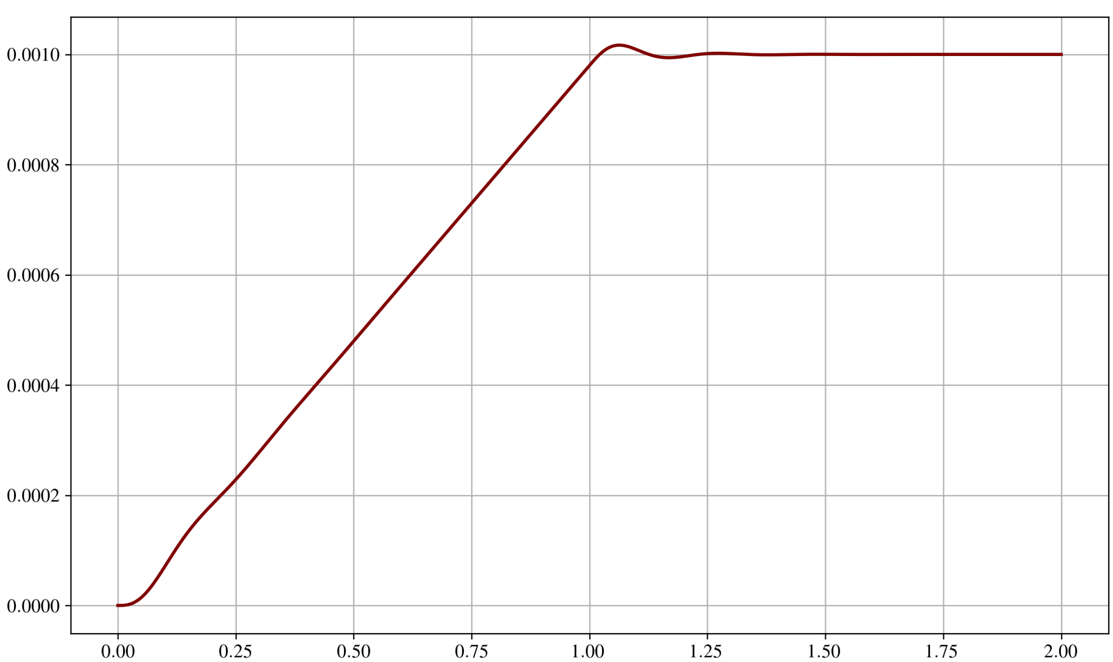
[36]:
$$\frac{(e^s - 1)\,e^{-s}}{s^2\,(s^2 + 20s + 1000)}$$

[37]:
```python
x_t = sp.inverse_laplace_transform(sol, s, t).simplify()
x_t
```

[37]:
$$\frac{t\theta\,(t)}{1000} + \left(-\frac{e^{-10t}\sin\,(30t)}{37500} + \frac{e^{-10t}\cos\,(30t)}{50000}\right)\theta\,(t) - \frac{((150t - 153)\,e^{10t-10} - 4\sin\,(30t - 30) + 3\cos\,(30t - 30))\,e^{10-10t}}{150000}$$
$$\frac{\theta\,(t)}{50000}$$

- Note that the $\theta(t)$ is the heaviside function (or unit step function).

[38]:
```python
x_lamb = sp.lambdify(t, x_t, modules='numpy')
t_ = np.linspace(0, 2, 500)
plt.plot(t_, x_lamb(t_))
plt.show()
```



6 Linear Algebra

- `sympy` is wonderful for visualizing matrices as it is able to output LaTeX matrices through jupyter notebook.

Example: Solve the following system by converting it to the matrix form, then augment the solution vector and put the matrix in the reduced row echelon form.

$$\begin{cases} x_1 - x_2 + 2x_3 = 4 \\ x_2 - 3x_3 = 2 \end{cases}$$

[39]:
```
x1, x2, x3 = sp.symbols('x1:4')  # defines sequence of symbols from 1 to 3
eq1 = sp.Eq(x1 - x2 + 2*x3, 4)
eq2 = sp.Eq(x2 - 3*x3, 2)
eq3 = sp.Eq(2*x1 + x2 - 4*x3, 2)
display(eq1, eq2, eq3)
```

$x_1 - x_2 + 2x_3 = 4$

$x_2 - 3x_3 = 2$

$2x_1 + x_2 - 4x_3 = 2$

[40]:
```
# Convert it to the matrix form
A, b = sp.linear_eq_to_matrix([eq1, eq2, eq3], (x1, x2, x3))
sp.Eq(A*sp.Matrix([x1, x2, x3]), b)
```

[40]:

$$\begin{bmatrix} x_1 - x_2 + 2x_3 \\ x_2 - 3x_3 \\ 2x_1 + x_2 - 4x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 2 \end{bmatrix}$$

[41]:
```
augmented = A.col_insert(3, b)
augmented
```

[41]:

$$\begin{bmatrix} 1 & -1 & 2 & 4 \\ 0 & 1 & -3 & 2 \\ 2 & 1 & -4 & 2 \end{bmatrix}$$

[42]:
```
augmented.rref()[0]
```

[42]:

$$\begin{bmatrix} 1 & 0 & 0 & -6 \\ 0 & 1 & 0 & -34 \\ 0 & 0 & 1 & -12 \end{bmatrix}$$