PYCON
INDONESIA 2018

Better Science Understanding With Sympy

Tirtadwipa Manunggal

Overview

- **Introduction**
- Sympy Fundamentals
  - Numbers and Symbols
  - Math Expression and Evaluations
  - Lambdify, Solver, and Matrix
  - Plotting
  - Calculus
- Demonstration

Introduction

- **SymPy** is an open source Python library for symbolic mathematics
- **SymPy** was started in August 2006 by Ondrej Certik
- Why **Sympy**?
    - Standalone
    - Full featured
    - BSD licensed
    - Embraces Python
    - Usable as a library

Introduction : Features

- **Core Capabilities**
  - Basic arithmetic: Support for operators such as +, -, *, /, ** (power)
  - Simplification
  - Expansion
  - Functions: trigonometric, hyperbolic, exponential, roots, logarithms, absolute value, spherical harmonics, factorials and gamma functions, zeta functions, polynomials, special functions, . . .
  - Substitution
  - Numbers: arbitrary precision integers, rationals, and floats
  - Noncommutative symbols
  - Pattern matching
- **Polynomials**
  - Basic arithmetic: division, gcd, . . .
  - Factorization
  - Square-free decomposition
  - Gröbner bases
  - Partial fraction decomposition
  - Resultants
- **Calculus**
  - Limits: $\lim_{x\to 0} x \log(x) = 0$
  - Differentiation
  - Integration: It uses extended Risch-Norman heuristic
  - Taylor (Laurent) series
- **Solving equations**
  - Polynomial equations
  - Algebraic equations
  - Differential equations
  - Difference equations
  - Systems of equations
- **Combinatorics**
  - Permutations
  - Combinations
  - Partitions
  - Subsets
  - Permutation Groups: Polyhedral, Rubik, Symmetric, . . .
  - Prufer and Gray Codes

Introduction : Features (contd.)

- **Discrete math**
  - Binomial coefficients
  - Summations
  - Products
  - Number theory: generating prime numbers, primality testing, integer factorization, ...
  - Logic expressions
- **Matrices**
  - Basic arithmetic
  - Eigenvalues/eigenvectors
  - Determinants
  - Inversion
  - Solving
  - Abstract expressions
- **Geometric Algebra**
- **Geometry**
  - points, lines, rays, segments, ellipses, circles, polygons, ...
  - Intersection
  - Tangency
  - Similarity
- **Plotting**
  - Coordinate modes
  - Plotting Geometric Entities
  - 2D and 3D
  - Interactive interface
  - Colors
- **Physics**
  - Units
  - Mechanics
  - Quantum
  - Gaussian Optics
  - Pauli Algebra
- **Statistics**
  - Normal distributions
  - Uniform distributions
  - Probability
- **Printing**
  - Pretty printing: ASCII/Unicode pretty printing, LaTeX
  - Code generation: C, Fortran, Python

Introduction : Setup

The rest of the slides will be run on the virtual environment, so it won't ruin our existing python environment.

```
virtualenv --python=python3 --no-site-packages venv
source venv/bin/activate
pip install jupyter sympy numpy matplotlib
```

Please wait for the library installation. Once finished, start the **jupyter**.

```
jupyter notebook
```

Introduction : Setup (contd.)

The rest of code demonstration in this workshop will use library and setup as follows :

```
from sympy import *
import  matplotlib.pyplot as plt
import numpy as np

%matplotlib notebook
init_printing()
```

Overview

- Introduction
- **Sympy Fundamentals**
  - **Numbers and Symbols**
  - Math Expression and Evaluations
  - Lambdify, Solver, and Matrix
  - Plotting
  - Calculus
- Demonstration

Sympy Fundamentals : Numbers and Symbols

Symbolic computation deals with the computation of mathematical objects symbolically. In other words, a mathematical number is represented exactly, not by approximation. If the number or expression is not evaluated, it will remain in symbolic form.

```
# Irrational number by approximation
import math
print(144/12)          # rational number
print(121/33)          # irrational number
print(math.sqrt(4))    # rational number
print(math.sqrt(12))   # irrational number
print(math.pi)         # irrational number
```

```
# Irrational number in symbolic form
display(S('144/12')) # rational number
display(S('121/33')) # irrational number
display(sqrt(4))     # rational number
display(sqrt(12))    # irrational number
display(pi)          # irrational number
```

Sympy Fundamentals : Numbers and Symbols (contd.)

SymPy also allows us to define variables and compute them symbolically. Variables in SymPy must be declared beforehand. This can be done either using `symbols`, `var`, `sympify`, or `S` function.

```
x = symbols('x')
var('y')
z = S('z')
sigma = sympify('sigma')
rho = S('rho')
display(x, y, z, sigma, rho)
```

x

y

z

σ

ρ

Sympy Fundamentals : Numbers and Symbols (contd.)

Sympy has reserved common symbols name as defined in `sympy.abc`, so instead of declaring variables name by ourself, we can import them. Here are some of usable common symbols : `A`, `B`, `C`, `D`, `E`, `F`, `G`, `H`, `I`, `J`, `K`, `L`, `M`, `N`, `O`, `P`, `Q`, `R`, `S`, `T`, `U`, `V`, `W`, `X`, `Y`, `Z`, `a`, `alpha`, `b`, `beta`, `c`, `chi`, `d`, `delta`, `e`, `epsilon`, `eta`, `f`, `g`, `gamma`, `h`, `i`, `iota`, `j`, `k`, `kappa`, `l`, `m`, `mu`, `n`, `nu`, `o`, `omega`, `omicron`, `p`, `phi`, `pi`, `psi`, `q`, `r`, `rho`, `s`, `sigma`, `t`, `tau`, `theta`, `u`, `upsilon`, `v`, `w`, `x`, `xi`, `y`, `z`, `zeta`

```
from sympy.abc import delta, psi, m, i
Eq((i*delta-m)*psi,0)
```

> $$\psi(\delta i - m) = 0$$

Sympy Fundamentals : Numbers and Symbols (contd.)

Once variables are declared, we can do further calculation and manipulation upon these variables.

```
display(x + 1)
display(x + y**2)
display(sqrt(z))
display(factor(x**3 + 2*x**2*y))
display(Eq(x + y, z))
```

$$x + 1$$

$$x + y^2$$

$$\sqrt{z}$$

$$x^2(x + 2y)$$

$$x + y = z$$

Sympy Fundamentals : Numbers and Symbols (contd.)

Besides variables, SymPy also support mathematical and physical constants. Here are some of the common constant : `pi` , `oo` , `I` , `E`

```
display(pi) # pi
display(oo) # infinity
display(I)  # imaginary / complex number
display(E)  # euler's number
```

> π

> ∞

> i

> e

Sympy Fundamentals : Numbers and Symbols (contd.)

Exercise

- Expand $(x+1)^2$
- Expand $(\alpha+4)\cdot(\beta-4)$
- Compute $\sqrt{-1}$
- Compute $e^{i\pi}+1$

Overview

- Introduction
- **Sympy Fundamentals**
    - Numbers and Symbols
    - **Math Expression and Evaluations**
    - Lambdify, Solver, and Matrix
    - Calculus
    - Plotting
- Demonstration

Sympy Fundamentals : Math Expression and Evaluations

Mathematical operation in SymPy is no different with math operation in python. Sympy can both do numerical and symbolic computation. The basic mathematic equation can be expressed using following operator :

Operator	Operation	Example
`+`	addition	`x + 1`
`-`	subtraction	`x - 1`
`*`	multiplication	`x * 2`
`**`	exponent	`x ** 3`
`/`	division	`x / 2`
`//`	floor division	`x // 2`
`%`	modulo	`x % 2`

Sympy Fundamentals : Math Expression and Evaluations (contd.)

```
x,y = symbols('x,y')
expr = x + 2
expr + 2 * y
```

> $x + 2$

```
expr * y
```

> $x + y + 2$

```
expr % y
```

> $(x + 2) mod\, y$

```
expr**2
```

> $(x + 2)^2$

Sympy Fundamentals : Math Expression and Evaluations (contd.)

Mathematical expression can be assigned in to other variable and evaluated or manipulated afterward. There are dozen of function to perform various operations. Here are some of handy function :

Function	Operation
`expand`	expand mathematical expression
`factor`	find factor of an expression into irreducible factors
`simplify`	find the intelligent simplest representation
`collect`	collects common powers of a term in an expression
`cancel`	put expression into the standard canonical form
`apart`	performs a partial fraction decomposition on a rational function

Sympy Fundamentals : Math Expression and Evaluations (contd.)

```
#Expansion example
expr = (x + 2)**2
display(expr)
display(expand(expr))
```

$$(x+2)^2$$

$$x^2+4x+4$$

```
# Factorization example
expr = x**3 - x**2 + x -1
display(expr)
display(factor(expr))
```

$$x^3-x^2+x-1$$

$$(x-1)(x^2-1)$$

Sympy Fundamentals : Math Expression and Evaluations (contd.)

```
# Simplification example
expr = (x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
display(expr)
display(factor(expr))
```

$$\frac{x^3+x^2-x-1}{x^2+2x+1}$$

$$x - 1$$

```
# Collect example
expr = x*y + x - 3 + 2*x**2 - y*x**2 + x**3
display(expr)
display(collect(expr,x))
```

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

$$x^3 + x^2(2 - y) + x(y + 1) - 3$$

Sympy Fundamentals : Math Expression and Evaluations (contd.)

```
# Cancel example
expr = (x**2 + 2*x +1)/(x**2 + x)
display(expr)
display(cancel(expr))
```

$$\frac{x^2+2x+1}{x^2+x}$$

$$\frac{x+1}{x}$$

```
# Apart example
expr = (8*x+7)/(x**2 + x -2)
display(expr)
display(apart(expr))
```

$$\frac{8x+7}{x^2+x-2}$$

$$\frac{3}{x+2}+\frac{5}{x-1}$$

Sympy Fundamentals : Math Expression and Evaluations (contd.)

After manipulating the mathematical expression, one of the most common things we might want to do is substitution.There are at least two way to do substitution; `subs` for substituting symbolic expression and `evalf` for evaluating numerically.

```
# Substitute variable with other variable
expr = x**2
expr.subs(x, y + 2)
```

```
# Substitute variable with number
expr = x ** 3
expr.subs(x, -1)
```

```
# Substitute multivariate expression
expr = x ** 3 + y ** 2 + 2*x*y + 4
expr.subs({x:1, y:2})
```

Sympy Fundamentals : Math Expression and Evaluations (contd.)

To evaluate a numerical expression into a floating point number, use evalf. SymPy can evaluate floating point expressions to arbitrary precision. By default, 15 digits of precision are used, but you can pass any number as the argument to evalf. Letâ€™s compute the first 50 digits of π.

```
print(pi.evalf())
print(pi.evalf(20))
print(pi.evalf(50))
```

> 3.14159265358979

> 3.1415926535897932385

> 3.1415926535897932384626433832795028841971693993751

Sympy Fundamentals : Math Expression and Evaluations (contd.)

To evaluate an symbolic expression numerically, we need to combine `subs` and `evalf` .

```
expr = sin(x * pi / 180)
print(expr.subs(x, 45).evalf())
print(expr.evalf(subs={x:45}))  # preferred
```

> 0.707106781186548

> 0.707106781186548

Overview

- Introduction
- **Sympy Fundamentals**
  - Numbers and Symbols
  - Math Expression and Evaluations
  - **Lambdify, Solver, and Matrix**
  - Calculus
  - Plotting
- Demonstration

Sympy Fundamentals : Lambdify, Solver, and Matrix

Function `evalf` are good for evaluating a mathematical function at a single point, but it is typically more convenient to turn the symbolic expression to reusable numerical function. Let me show you the inconvenience as follows,

```
var('x')
inputs = np.linspace(1,10,10)

f = x**2
f(1) # error, not callable
f(inputs) # even not possible
```

> `Error object is not callable`

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

To overcome this, `lamdify` takes in a symbolic variable (or list of variables) and an expression, then returns a callable function that corresponds to the expression.

```
var('x, y')
g = lambdify(x, x**2)
h = lambdify((x, y), x+y)
print(g(0), g(10), g(100)) # become callable
print(h(2, 3), h(0.1, 3.14)) # much more convenient
```

> `0 100 10000`

> 5 3.24

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

`lambdify` has `module` input argument to define how the computation behaves. It maps the symbolic function in to numerical implementation. We can use either `sympy`, `numpy`, `math`, `numexpr`, or `mpmath`. If not specified differently by the user, SymPy functions are replaced as far as possible by either `python-math`, `numpy` (if available) or `mpmath` functions - exactly in this order. The choice also determine computational speed.

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

```
points = np.random.random(20000)
expr = sinh(x)

# Time using evalf() on each of the random points.
%time _ = [expr.subs(x, pt).evalf() for pt in points]

# Lambdify the expression and time using the resulting function.
f = lambdify(x, expr)
%time _ = [f(pt) for pt in points]

# Lambdify the expression and time using the default module order.
f = lambdify(x, expr)
%time _ = f(points)

# Lambdify the expression and time using only numpy module.
f = lambdify(x, expr, "numpy")
%time _ = f(points)
```

PYCON INDONESIA 2018

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

> CPU times: user 9.3 s, sys: 518 µs, total: 9.3 s
>
> Wall time: 9.31 s
>
> CPU times: user 24.5 ms, sys: 0 ns, total: 24.5 ms
>
> Wall time: 24.5 ms
>
> CPU times: user 855 µs, sys: 0 ns, total: 855 µs
>
> Wall time: 582 µs
>
> CPU times: user 738 µs, sys: 0 ns, total: 738 µs
>
> Wall time: 465 µs

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

A SymPy expression by itself is not an equation. However, `solve` equates an expression with zero and solves for a specified variable.

```
expr = x**2 - 4
display(expr) # right-hand side is equated as zero
solve(expr,x)
```

$$x^2 - 4$$

$$[-2, \quad 2]$$

We can also define both sides using `Eq` instead of assuming the right hand-side to be zero.

```
expr = Eq(x**2,4)
display(expr) # two sides equation
solve(expr,x)
```

$$x^2 - 4$$

$$[-2, \quad 2]$$

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

Sometimes we also calculate systems of equation. For example a system of linear equation as following

$$
\begin{array}{ccccccc}
x & + & y & + & z & = & 5 \\
2x & + & 3y & + & 2z & = & 2 \\
7x & + & y & + & 6z & = & 12
\end{array}
$$

We can rewrite in sympy and find the solutions.

```
var('x, y, z')
M = Matrix([
    [1, 1, 1, 5],
    [2, 3, 2, 2],
    [7, 1, 6, 12]
])
solve_linear_system(M, x, y, z)
```

> $\{x : -58, \quad y : -8, \quad z : 71\}$

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

SymPy matrices support the standard matrix operations of addition `+`, subtraction `-`, and multiplication `*`. Additionally, SymPy matrices are equipped with many useful methods, some of which are listed below. See http://docs.sympy.org/latest/modules/matrices/matrices.html for more methods and examples.

Method	Return
`det()`	The determinant.
`eigenvals()`	The eigenvalues and their multiplicities.
`eigenvects()`	The eigenvectors and their corresponding eigenvalues.
`inv()`	The matrix inverse.
`norm()`	The Frobenius, ∞, 1, or 2 norm.
`T`	The transpose.

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

```
# Example of matrix
A = Matrix([[1, 2],[3, 4]])

display(A, T)
display(A.T)
display(A.norm())
display(A * A.inv())
```

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

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

$$\sqrt{30}$$

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

Sympy Fundamentals : Lambdify, Solver, and Matrix (contd.)

```
# Example of symbolic matrix
var('a, b, c, d')
B = Matrix([[a, b],[c, d]])

display(B)
display(B.inv())
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\begin{bmatrix} \frac{d}{ad - bc} & -\frac{b}{ad - bc} \\ -\frac{c}{ad - bc} & \frac{a}{ad - bc} \end{bmatrix}$$

PYCON INDONESIA 2018

References

- www.sympy.org
- docs.sympy.org
- mattpap.github.io/scipy-2011-tutorial/html/
- www.sympy.org/scipy-2016-tutorial/