FREE eBook

LEARNING

sympy

Free unaffiliated eBook created from
Stack Overflow contributors.

#sympy

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sympy

It is an unofficial and free sympy ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sympy.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with sympy

Remarks

This section provides an overview of what sympy is, and why a developer might want to use it.

It should also mention any large subjects within sympy, and link out to the related topics. Since the Documentation for sympy is new, you may need to create initial versions of those related topics.

Examples

Installing SymPy

The easiest and recommended way to install SymPy is to install Anaconda.

If you already have Anaconda or Miniconda installed, you can install the latest version with conda:

```
conda install sympy
```

Another way of installing SymPy is using pip:

```
pip install sympy
```

Note that this might require root privileges, so one might acually need

```
sudo pip install sympy
```

Most linux distributions also offer SymPy in their package repositories. For Fedora one would install SymPy with

```
sudo dnf install python-sympy
sudo dnf install python3-sympy
```

The first one installs the python 2 version of the package, the latter python 3.

On OpenSuse the respective commands are:

```
sudo zypper install python-sympy
sudo zypper install python3-sympy
```

The packages for OpenSuse 42.2 seem rather outdated, so one of the first methods should be prefered.

Alternate installation (not conda)

Alternate ways to install SymPy from conda. conda is the recommended way, but these are some alternate ways. Including: git, pip, etc.

'Hello World'

Sympy is a Python library for doing symbolic — rather than numeric — calculations.

For instance, consider the quadratic equation in **x**,

x**2 + HELLO * x + WORLD = 0

where HELLO and WORLD are constants. What's the solution of this equation?

In Python, using Sympy we can code,

```
from sympy import symbols, solve, latex

x, HELLO, WORLD = symbols('x, HELLO, WORLD')
print ( latex ( solve ( x**2 + HELLO * x + WORLD, x ) ) )
```

Since I made a call to Latex the solutions are almost ready for publication! Sympy provides the two of them packed in a list. Here's one:

$$-\frac{HELLO}{2}-\frac{1}{2}\sqrt{HELLO^2-4WORLD}$$

If you need to do more work on an expression then you would leave out the call to latex.

Integration and Differentiation

Sympy is made for symbolic math, so let's have a look at some basic integration and differentiation.

```
from sympy import symbols, sqrt, exp, diff, integrate, pprint


x, y = symbols('x y', real=True)


pprint(diff(4*x**3+exp(3*x**2*y)+y**2,x))

pprint(diff(4*x**3+exp(3*x**2*y)+y**2,y))

pprint(integrate(exp(x*y**2)+sqrt(x)*y**2,x))

pprint(integrate(exp(x*y**2)+sqrt(x)*y**2,y))
```

First we import the necessary functions from sympy. Next we define our variables x and y. Note that these are considered complex by default, so we tell sympy that we want a simple example by making them real. Next we differentiate some expression with respect to x and then y. Finally we

integrate some expression, again with respect to x and then y. The call of `pprint` ensures that our functions get written in some nice human readable style.

Read Getting started with sympy online: https://riptutorial.com/sympy/topic/5450/getting-started-with-sympy

Chapter 2: Differential Calculus

Examples

Constrained Non-Linear Optimization

Problem statement:

Find the minimum (over `x`, `y`) of the function `f(x,y)`, subject to `g(x,y)=0`, where `f(x,y) = 2 * x**2 + 3 * y**2` and `g(x,y) = x**2 + y**2 - 4`.

Solution: We will solve this problem by performing the following steps:

1. Specify the Lagrangian function for the problem
2. Determine the Karush-Kuhn-Tucker (KKT) conditions
3. Find the `(x,y)` tuples that satisfy the KKT conditions
4. Determine which of these `(x,y)` tuples correspond to the minimum of `f(x,y)`

First, define the optimization variables as well as objective and constraint functions:

```
import sympy as sp
x, y = sp.var('x,y',real=True);
f = 2 * x**2 + 3 * y**2
g = x**2 + y**2 - 4
```

Next, define the Lagrangian function which includes a Lagrange multiplier `lam` corresponding to the constraint

```
lam = sp.symbols('lambda', real = True)
L = f - lam* g
```

Now, we can compute the set of equations corresponding to the KKT conditions.

```
gradL = [sp.diff(L,c) for c in [x,y]] # gradient of Lagrangian w.r.t. (x,y)
KKT_eqs = gradL + [g]
KKT_eqs
```

```
[-2*lambda*x + 4*x, -2*lambda*y + 6*y, x**2 + y**2 - 4]
```

The potential minimizers of `f` (given `g=0`) are obtained by solving the `KKT_eqs` equations overs `x`, `y`, `lam`:

```
stationary_points = sp.solve(KKT_eqs, [x, y, lam], dict=True) # solve the KKT equations
stationary_points
```

```
[{x: -2, y: 0, lambda: 2},
 {x: 2, y: 0, lambda: 2},
 {x: 0, y: -2, lambda: 3},
```

```
{x: 0, y: 2, lambda: 3}]
```

Finally, check the objective function for each of the above points to determine the minimum

```
[f.subs(p) for p in stat_points]
```

```
[8, 8, 12, 12]
```

It follows that the constrained minimum of `f` equals `8` and is achieved at `(x,y)=(-2,0)` and `(x,y)=(2,0)`.

Read Differential Calculus online: https://riptutorial.com/sympy/topic/6867/differential-calculus

Chapter 3: Equations

Examples

Solve system of linear equations

```
import sympy as sy

x1, x2 = sy.symbols("x1 x2")

equations = [
    sy.Eq( 2*x1 + 1*x2 ,  10 ),
    sy.Eq( 1*x1 - 2*x2 ,  11 )
]

print sy.solve(equations)
# Result: {x1: 31/5, x2: -12/5}
```

Solve nonlinear set of equations numerically

```
import sympy as sy

x, y = sy.symbols("x y")

# nsolve needs the (in this case: two) equations, the names of the variables
# (x,y) we try to evaluate solutions for, and an initial guess (1,1) for the
# solution
print sy.nsolve((x**3+sy.exp(y)-4,x+3*y),(x,y),(1,1))
```

The result shown will be the solution for x and y:

```
[  1.50281519319939]
[-0.500938397733129]
```

Solve a single equation

```
import sympy as sy

# Symbols have to be defined before one can use them
x = sy.S('x')

# Definition of the equation to be solved
eq=sy.Eq(x**2 + 2, 6)

#Print the solution of the equation
print sy.solve(eq)
```

The result printed will be:

```
[-2, 2]
```

Read Equations online: https://riptutorial.com/sympy/topic/6833/equations

Chapter 4: Solvers

Remarks

As of version 1.0 of Sympy perhaps the main thing to understand about using its solvers is that ' **solveset** will take over **solve** either internally or externally'. At this point solveset should already be used for solving univariate equations and systems of linear equations.

Examples

Solving a univariate inequality

```
>>> from sympy.solvers.inequalities import solve_univariate_inequality
>>> from sympy import var
>>> x=var('x')
>>> solve_univariate_inequality(2*x**2-6>1,x,relational=False)
(-oo, -sqrt(14)/2) U (sqrt(14)/2, oo)
```

The **relational=False** parameter simply indicates how the results are to be rendered. The default (**relational=True**) produces a result like this.

```
>>> solve_univariate_inequality(2*x**2-6>1,x)
Or(And(-oo < x, x < -sqrt(14)/2), And(sqrt(14)/2 < x, x < oo))
```

Solving a linear Diophantine equation

[![Sample equation][1]][1]

sympy provides its solution as a Python set of expressions in terms of parametric variables, as shown here in the final line.

```
>>> from sympy.solvers.diophantine import diophantine
>>> from sympy import var
>>> x,y,z=var('x y z')
>>> diophantine(2*x+3*y-5*z-77)
{(t_0, -9*t_0 - 5*t_1 + 154, -5*t_0 - 3*t_1 + 77)}
```

Read Solvers online: https://riptutorial.com/sympy/topic/7199/solvers

Credits

S. No	Chapters	Contributors
1	Getting started with sympy	asmeurer, Bill Bell, Community, Hannebambel
2	Differential Calculus	Stelios
3	Equations	tfv
4	Solvers	Bill Bell