EBook Gratis

APRENDIZAJE sympy

Free unaffiliated eBook created from
Stack Overflow contributors.

#sympy

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con sympy

Observaciones

Esta sección proporciona una descripción general de qué es sympy y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de Sympy y vincular a los temas relacionados. Dado que la Documentación para Sympy es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Instalando SymPy

La forma más fácil y recomendada de instalar SymPy es instalar Anaconda .

Si ya tiene Anaconda o Miniconda instalada, puede instalar la última versión con conda:

```
conda install sympy
```

Otra forma de instalar SymPy es usando pip:

```
pip install sympy
```

Tenga en cuenta que esto puede requerir privilegios de raíz, por lo que uno podría necesitar realmente

```
sudo pip install sympy
```

La mayoría de las distribuciones de Linux también ofrecen SymPy en sus repositorios de paquetes. Para Fedora se instalaría SymPy con

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

El primero instala la versión python 2 del paquete, el último python 3.

En OpenSuse los comandos respectivos son:

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

Los paquetes para OpenSuse 42.2 parecen bastante obsoletos, por lo que se debe preferir uno de los primeros métodos.

Instalación alternativa (no conda)

Formas alternativas de instalar SymPy desde conda. Conda es la forma recomendada, pero estas son algunas formas alternativas. Incluyendo: git, pip, etc.

'Hola Mundo'

Sympy es una biblioteca de Python para realizar cálculos simbólicos, en lugar de numéricos.

Por ejemplo, considere la ecuación cuadrática en **x** ,

x ** 2 + HOLA * x + MUNDO = 0

donde HOLA y MUNDO son constantes. ¿Cuál es la solución de esta ecuación?

En Python, usando Sympy podemos codificar,

```
from sympy import symbols, solve, latex

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

¡Desde que hice una llamada a Latex, las soluciones están casi listas para su publicación! Sympy proporciona los dos empaquetados en una lista. Aquí hay uno:

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

Si necesita trabajar más en una expresión, entonces dejaría de lado la llamada al látex.

Integración y diferenciación

Sympy está hecho para matemáticas simbólicas, así que echemos un vistazo a algunas integraciones y diferenciaciones básicas.

```
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))
```

Primero importamos las funciones necesarias desde sympy. A continuación definimos nuestras

variables x y y. Tenga en cuenta que estos se consideran complejos de manera predeterminada, por lo que le decimos a Sympy que queremos un ejemplo simple al hacerlos reales. A continuación diferenciamos alguna expresión con respecto a x y luego y. Finalmente integramos alguna expresión, nuevamente con respecto a x y luego y. El call of `pprint` garantiza que nuestras funciones se escriban con un estilo agradable y legible para los humanos.

Lea Empezando con sympy en línea: https://riptutorial.com/es/sympy/topic/5450/empezando-con-sympy

Capítulo 2: Calculo diferencial

Examples

Optimización no lineal restringida

Enunciado del problema :

Encuentre el mínimo (sobre `x` , `y`) de la función `f(x,y)` , sujeto a `g(x,y)=0` , donde `f(x,y) = 2 * x**2 + 3 * y**2` y `g(x,y) = x**2 + y**2 - 4` .

Solución : Resolveremos este problema realizando los siguientes pasos:

1. Especifique la función lagrangiana para el problema.
2. Determinar las condiciones de Karush-Kuhn-Tucker (KKT)
3. Encuentra las tuplas `(x,y)` que satisfacen las condiciones KKT
4. Determine cuál de estas `(x,y)` tuplas corresponde al mínimo de `f(x,y)`

Primero, defina las variables de optimización, así como las funciones de objetivo y restricción:

```
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
```

A continuación, defina la función Lagrangiana que incluye un `lam` multiplicador de Lagrange correspondiente a la restricción

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

Ahora, podemos calcular el conjunto de ecuaciones correspondientes a las condiciones KKT.

```
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]
```

Los minimizadores potenciales de `f` (dado `g=0`) se obtienen resolviendo las ecuaciones de `KKT_eqs` sobre `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}]
```

Finalmente, verifique la función objetivo para cada uno de los puntos anteriores para determinar el mínimo

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

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

De ello se deduce que el mínimo restringido de `f` es `8` y se alcanza en `(x,y)=(-2,0)` y `(x,y)=(2,0)` .

Lea Calculo diferencial en línea: https://riptutorial.com/es/sympy/topic/6867/calculo-diferencial

Capítulo 3: Ecuaciones

Examples

Sistema de resolución de ecuaciones lineales.

```
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}
```

Resuelve numéricamente un conjunto de ecuaciones no lineales.

```
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))
```

El resultado mostrado será la solución para x y y:

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

Resuelve una sola ecuación

```
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)
```

El resultado impreso será:

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

Lea Ecuaciones en línea: https://riptutorial.com/es/sympy/topic/6833/ecuaciones

Capítulo 4: Solucionadores

Observaciones

A partir de la versión 1.0 de Sympy, quizás lo más importante que se debe entender sobre el uso de sus solucionadores es que ' **solveset** se encargará de **resolver** ya sea interna o externamente'. En este punto, solveset debería ya usarse para resolver ecuaciones univariadas y sistemas de ecuaciones lineales.

Examples

Resolviendo una desigualdad univariada.

```
>>> 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)
```

El parámetro **relacional = Falso** simplemente indica cómo se deben representar los resultados. El valor predeterminado (**relacional = Verdadero**) produce un resultado como este.

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

Resolviendo una ecuación diofántica lineal.

[! [Ecuación de muestra] [1]] [1]

sympy proporciona su solución como un conjunto de expresiones de Python en términos de variables paramétricas, como se muestra aquí en la línea final.

```
>>> 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)}
```

Lea Solucionadores en línea: https://riptutorial.com/es/sympy/topic/7199/solucionadores

Creditos

S. No	Capítulos	Contributors
1	Empezando con sympy	asmeurer, Bill Bell, Community, Hannebambel
2	Calculo diferencial	Stelios
3	Ecuaciones	tfv
4	Solucionadores	Bill Bell