Symbolic Calculus with SymPy

SymPy stands for Symbolic Python. It is a library designed to perform symbolic computations, meaning it can expand, simplify, differentiate, and solve equations symbolically.

1 Getting Started with SymPy

Before exploring the advanced features of SymPy, please make sure it is properly set up.

1.1 Installation and Setup

The simplest method to install SymPy, which is also the recommended method, is to use pip:

```
pip install sympy
```

If you are using Conda, the command is as follows:

```
conda install sympy
```

You can also install directly from the source code (this is particularly useful for contributors):

```
git clone https://github.com/sympy/sympy.git
```

1.2 Verifying the Installation

To verify the installation, please open Python and enter:

```
import sympy
print(sympy.__version__)
```

```
1.14.0
```

The output displays the installed version of the package. You can now test the package using a simple expression:

```
from sympy import symbols

# Define x as a symbolic variable
x = symbols('x')
print((x + 2)**2)
```

```
(x + 2)**2
```

Regarding troubleshooting:

- If you encounter the error "ModuleNotFoundError", please rerun `pip install sympy`.
- In case of version conflicts, please update pip: `pip install -upgrade pip`

It is also important to know that: If you think there is a bug or if you wish to request a feature, you can also open a ticket. For more details on installation methods, please consult the official documentation.

2 Basic Usage

SymPy offers an interactive environment in which you can symbolically manipulate algebraic expressions.

```
from sympy import symbols, expand

# define x and y as symbolic variables
x, y = symbols('x y')

# create an algebraic expression
expr = (x + y)**2
expanded = expand(expr)
print(expanded)
```

```
x**2 + 2*x*y + y**2
```

The printed result above `x**2 + 2*x*y + y**2` is the fully expanded form, demonstrating how SymPy performs exact symbolic algebra rather than numerical computation.

3 Main Features and Functionalities

The main features are focused on symbolic computation, which involves working with functions, equations, and expressions as symbols rather than as numbers.

3.1 Arithmetic Operations

You can treat symbolic variables as algebraic variables. Moreover, functions can pass from one form to another, as shown in the code below.

Using `expand()` for distribution and `factor()` for the inverse simplification, SymPy performs the transformation between expanded and factored forms of an algebraic expression.

```
# Define symbolic variables x and y
x, y = symbols('x y')

# Create an algebraic expression
expr = (x + y)**2

# Expand the expression to its full polynomial form
print(expand(expr))

# Factor the expanded polynomial back into its compact form
print(factor(x**2 + 2*x*y + y**2))
```

```
x**2 + 2*x*y + y**2
(x + y)**2
```

3.2 Simplification Techniques

Simplification is essential in symbolic computation. SymPy has a general function, called `simplify()`, which makes it easy to simplify rational, trigonometric, or algebraic expressions.

```
from sympy import simplify, trigsimp, sin, cos

# Create a trigonometric expression
expr = sin(x)**2 + cos(x)**2
```

```
# General simplification -- recognizes that sin^2(x) + cos^2(x) = 1
print(simplify(expr))

# Trigonometric-specific simplification -- also simplifies to 1
print(trigsimp(expr))
```

The result of the `simplify()` and `trigsimp()` functions above is equal to one.

4 Calculus Operations

SymPy is particularly suited to differential and integral calculus, as it allows derivatives, integrals, limits, and even series expansions to be performed symbolically, as if doing the calculations by hand, but with perfect precision.

4.1 Differentiation

To compute derivatives, please use the `diff()` function.

```
from sympy import diff

# Define a symbolic polynomial expression
f = x**3 + 2*x**2 + x

# Compute the derivative of the expression with respect to x
df = diff(f, x)
print(df)
```

```
3*x**2 + 4*x + 1
```

4.2 Integration

The code below computes the indefinite integral of the symbolic expression f with respect to the variable x using SymPy's `integrate()` function. The resulting antiderivative is then displayed.

```
from sympy import integrate

# Compute the indefinite integral of the expression f with respect to x
integral = integrate(f, x)
print(integral)
```

```
x**4/4 + 2*x**3/3 + x**2/2
```

4.3 Limits

Limit computation is a classic operation in differential calculus, which can be performed using SymPy's `limit()` function. The syntax is as follows: `limit(f(x), x, x0)`. For example, the code below computes the limit of the expression $\sin(x)/x$ as x approaches zero and displays the exact result, one.

```
from sympy import limit, sin

# Compute the limit of sin(x)/x as x approaches 0
limit_expr = limit(sin(x)/x, x, 0)
print(limit_expr)
```

The `series()` function allows the computation of the Taylor series or power series expansion of a mathematical expression around a given point.

```
from sympy import series, exp

# Generate the Taylor series expansion of e^x around 0 up to 5 terms
print(series(exp(x), x, 0, 5))
```

```
1 + x + x**2/2 + x**3/6 + x**4/24 + O(x**5)
```

4.4 Summations

The `summation()` function in SymPy performs exact symbolic summation over a certain range, just like the sigma notation in mathematics.

```
from sympy import summation, symbols

# Compute the symbolic sum of 1/n^2 from n = 1 to 5
n = symbols('n')
print(summation(1/n**2, (n, 1, 5)))
```

```
5269/3600
```

5 Solving Equations

SymPy facilitates the solving of symbolic equations, not only simple algebraic equations but also systems of equations, differential equations, and even Diophantine equations, which have only integer solutions.

It has powerful solvers such as:

- `solve()` for algebraic equations,
- `linsolve()` for linear systems, and
- `dsolve()` for differential equations that provide exact solutions when possible.

5.1 Algebraic Equations

```
from sympy import Eq, solve

# Define and solve the equation x^2 - 4 = 0 symbolically
eq = Eq(x**2 - 4, 0)
print(solve(eq, x))
```

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

5.2 Systems of Equations

```
from sympy import linsolve, symbols

# Solve a system of linear equations:
# x + y = 5 and x - y = 1
x, y = symbols('x y')
sol = linsolve([x + y - 5, x - y - 1], (x, y))
print(sol) # Outputs the solution as a set (x, y)
```

```
{(3, 2)}
```

5.3 Differential Equations

```python
from sympy import Function, dsolve

# Define y as a symbolic function of x
y = Function('y')

# Define a second-order differential equation
diff_eq = Eq(y(x).diff(x, 2) - y(x), 0)

# Solve the differential equation symbolically
print(dsolve(diff_eq))
```

```
Eq(y(x), C1*exp(-x) + C2*exp(x))
```