sympy-moin-notes

March 17, 2024

[1]:
```
'''SymPy is a Python library for symbolic mathematics.
It is an open-source package that provides tools for performing
various symbolic operations, including algebraic manipulations,
calculus, equation solving, symbolic expressions, and more.
Sympy is designed to handle symbolic computation, which means it
works with mathematical symbols, expressions, and equations in their
symbolic form rather than numerical values'''
from sympy import *
```

[2]:
```
'''Creating Symbols:
A core feature in SymPy is to represent mathematical symbols as Python objects.
To create symbols in SymPy, you typically use the symbols function or the␣
 ↪Symbol class. i.e.'''
x=Symbol('x')
y=Symbol('y')
Exp=x**2+2*y+1
Exp
#or
```

Note: for nicely formatted output from SymPy, we also need to set up its printing system: sympy.init_printing()

[2]:
$$x^2 + 2y + 1$$

[3]:
```
x,y=symbols('x y')
Exp=x**2+2*y+1
Exp
```

[3]:
$$x^2 + 2y + 1$$

[9]:
```
'''The variable x now represents an abstract mathematical symbol x of which␣
 ↪very little
information is known by default. At this point, x could represent, for example,␣
 ↪a real number,
an integer, a complex number, a function, as well as a large number of other␣
 ↪possibilities.
In many cases it is sufficient to represent a mathematical symbol with this␣
 ↪abstract,
unspecified Symbol object, but sometimes it is necessary to give the SymPy␣
 ↪library more
hints about exactly what type of symbol a Symbol object is representing.
```

```
This may help SymPy to more efficiently manipulate or simplify analytical␣
 ↪expressions.
We can add on various assumptions that narrow down the possible properties of a␣
 ↪symbol
by adding optional keyword arguments to the symbol-creating functions. For␣
 ↪deatils Selected
Assumptions and Their Corresponding Keyword for Symbol Objects  see Table 3.1'''
x=Symbol('x',real='true')
y=Symbol('y')
z1=sqrt(x**2)
z2=sqrt(y**2)
z1
```

[9]: $|x|$

[7]:
```
z2
```

[7]: $\sqrt{y^2}$

[ ]:
```
'''For Selected Mathematical Constants and Special Symbols and Their
Corresponding Symbols in SymPy see Table 3.2'''
```

[15]:
```
#In SymPy, objects that represent functions can be created with sympy.Function.
#The resulting function is undefined(abstract) and unapplied:
x,y,z=symbols('x y z')
f=Function('f')(x,y,z)
f=x**2+y**2+z**2
f.free_symbols #Indp. Vars.
```

[15]: {x, y, z}

[16]:
```
f
```

[16]: $x^2 + y^2 + z^2$

[17]:
```
'''A special type of function in SymPy is lambda functions, or anonymous␣
 ↪functions,
which do not have names associated with them, but do have a specific function␣
 ↪body
that can be evaluated. Lambda functions can be created with sympy.Lambda:'''
h=Lambda(x,x**3)
h(3)
```

[17]: 27

[20]:
```
#Simplification:
x=symbols('x')
exp1=2*(x**2-x)-x*(x+1)
```

```
simplify(exp1)
```

[20]: $x\left(x-3\right)$

[21]:
```
#Expansion:
exp2=x*(x-3)
expand(exp2)
```

[21]: x^2-3x

[22]:
```
#Factorization:
exp3=x**2-4
factor(exp3)
```

[22]: $\left(x-2\right)\left(x+2\right)$

[23]:
```
#Roots finding(Solution):
eq=2*(x**2-x)-x*(x+1)
solve(eq)
```

[23]: [0, 3]

[35]:
```
'''Solving a system of equations for more than one unknown variable in SymPy is␣
 ↪a
straightforward generalization of the procedure used for univariate equations.␣
 ↪Instead of
passing a single expression as the first argument to sympy.solve, a list of␣
 ↪expressions that
represents the system of equations is used, and in this case the second␣
 ↪argument should
be a list of symbols to solve for. For example, the following two examples␣
 ↪demonstrate
how to solve two systems that are linear and nonlinear equations in x and y,␣
 ↪respectively:'''
x,y=symbols('x y')
eq1=x**2-y
eq2=y**2-x
solve([eq1,eq2],[x,y],dict='true')
'''Note that in both these examples, the function sympy.solve returns a list␣
 ↪where each
element represents a solution to the equation system. The optional keyword␣
 ↪argument
dict=True was also used, to request that each solution is returned in␣
 ↪dictionary format,
which maps the symbols that have been solved for to their values.'''
```

[35]:
```
[{x: 0, y: 0},
 {x: 1, y: 1},
 {x: (-1/2 - sqrt(3)*I/2)**2, y: -1/2 - sqrt(3)*I/2},
 {x: (-1/2 + sqrt(3)*I/2)**2, y: -1/2 + sqrt(3)*I/2}]
```

[37]:
```
#Substitution:
x,y=symbols('x y')
ex=x**3+y**3+10
ex.subs(x,10)
```

[37]: $y^3 + 1010$

[39]:
```
ex.subs({x:10,y:1}) #input as dict.
```

[39]: 1011

[42]:
```
'''Even when working with symbolic mathematics, it is almost invariably sooner␣
 ↪or later
required to evaluate the symbolic expressions numerically, for example, when␣
 ↪producing
plots or concrete numerical results. A SymPy expression can be evaluated using␣
 ↪either
the sympy.N function'''
c=1+pi
x=N(c,4) #upto 4 digits
x
```

[42]: 4.142

[49]:
```
x=N(pi,100000) #Pi upto 1lac digits
x
```

[49]: 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706

[85]:
```
#Limits:
x,y=symbols('x y')
h=1/(y**2-x)
fl=limit(limit(h,x,2),y,3)
fl
```

[85]: $\frac{1}{7}$

[56]:
```
#Derivation:
'''The derivative of a function describes its rate of change at a given point.␣
 ↪In SymPy we can
calculate the derivative of a function using sympy.diff or alternatively by␣
 ↪using the diff
```

```
method of SymPy expression instances. The argument to these functions is a␣
 ↪symbol,
or a number of symbols, with respect to which the function or the expression is␣
 ↪to be
derived. To represent the first-order derivative of an abstract function f(x)␣
 ↪with respect
to x, we can do'''
x=Symbol('x')
f= x**2+2*x+2
fd=diff(f,x)
fd
```

[56]: $2x + 2$

[58]:
```
fdd=diff(f,x,x)
fdd
```

[58]: 2

[59]:
```
fddd=diff(f,x,3) #equivalent to diff(f,x,x,x)
fddd
```

[59]: 0

[60]:
```
fd.subs(x,1)
```

[60]: 4

[66]:
```
#Partial Derivative:
x,y,z=symbols('x y z')
g=2*z*x**3+y**3*x+z**3*y
gdx=diff(g,x,2,z) #equivalent to diff(g,x,x,z)
gdx
```

[66]: $12x$

[77]:
```
#Integration:
x,y,z=symbols('x y z')
g=2*z*x**3+y**3*x+z**3*y
intx=integrate(g,x,(x,1,2),y) #First double-integrate w.r.t to x and input x as␣
 ↪limit (1,2) then int. w.r.t y
intx
```

[77]: $\frac{7y^4}{24} + \frac{3y^2z^3}{4} + \frac{31yz}{10}$

[79]:
```
#Defining Series:
x=Symbol('x')
f=E**x
```

```
f
```

[79]: e^x

[83]:
```
fs=series(f,x)
fs
```

[83]: $1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O\left(x^6\right)$

[90]:
```
'''Sums and products can be symbolically represented using the SymPy classes␣
 ↪sympy.
Sum and sympy.Product. They both take an expression as their first argument,␣
 ↪and as
a second argument, they take a tuple of the form (n, n1, n2), where n is a␣
 ↪symbol
and n1 and n2 are the lower and upper limits for the symbol n, in the sum or␣
 ↪product,
respectively. After sympy.Sum or sympy.Product objects have been created, they␣
 ↪can be
evaluated using the doit method:'''
n=symbols('n',integrer='true')
x=Sum(1/(n**2),(n,1,oo))  #infinity=double o
x
```

[90]: $\sum_{n=1}^{\infty} \frac{1}{n^2}$

[92]:
```
x.doit()
```

[92]: $\frac{\pi^2}{6}$

[93]:
```
n=symbols('n',integrer='true')
x=Product(1/(n**2),(n,1,8))  #infinity=double o
x
```

[93]: $\prod_{n=1}^{8} \frac{1}{n^2}$

[99]:
```
z=x.doit()
z
```

[99]: $\frac{1}{1625702400}$

[100]:
```
N(z,5)
```

[100]: $6.1512 \cdot 10^{-10}$

[105]:
```
#Matrix
A=Matrix([[1,2,2],[1,2,4],[3,4,5]])
B=transpose(A)
B
```

[105]:
$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 4 \\ 2 & 4 & 5 \end{bmatrix}$$

[106]:
```
C=A*B #Matrix Multilplication
C
```

[106]:
$$\begin{bmatrix} 9 & 13 & 21 \\ 13 & 21 & 31 \\ 21 & 31 & 50 \end{bmatrix}$$

[ ]:
```
#For other matrice operations see attached Table 3.2.
```

Table 3-4. *Selected Functions and Methods for Operating on SymPy Matrices*

Function/Method	Description
`transpose/T`	Compute the transpose of a matrix.
`adjoint/H`	Compute the adjoint of a matrix.
`trace`	Compute the trace (sum of diagonal elements) of a matrix.
`det`	Compute the determinant of a matrix.
`inv`	Compute the inverse of a matrix.
`LUdecomposition`	Compute the LU decomposition of a matrix.
`LUsolve`	Solve a linear system of equations in the form $Mx = b$, for the unknown vector x, using LU factorization.
`QRdecomposition`	Compute the QR decomposition of a matrix.
`QRsolve`	Solve a linear system of equations in the form $Mx = b$, for the unknown vector x, using QR factorization.
`diagonalize`	Diagonalize a matrix M, such that it can be written in the form $D = P^{-1}MP$, where D is diagonal.
`norm`	Compute the norm of a matrix.
`nullspace`	Compute a set of vectors that span the null space of a Matrix.
`rank`	Compute the rank of a matrix.
`singular_values`	Compute the singular values of a matrix.
`solve`	Solve a linear system of equations in the form $Mx = b$.

Table 3-2. *Selected Mathematical Constants and Special Symbols and Their Corresponding Symbols in SymPy*

Mathematical Symbol	SymPy Symbol	Description
π	`sympy.pi`	Ratio of the circumference to the diameter of a circle.
e	`sympy.E`	The base of the natural logarithm, $e = \exp(1)$.
γ	`sympy.EulerGamma`	Euler's constant.
i	`sympy.I`	The imaginary unit.
∞	`sympy.oo`	Infinity.

Table 3-1. *Selected Assumptions and Their Corresponding Keyword for* `Symbol` *Objects. For a complete list, see the docstring for* `sympy.Symbol`

Assumption Keyword Arguments	Attributes	Description
`real, imaginary`	`is_real, is_imaginary`	Specify that a symbol represents a real or imaginary number.
`positive, negative`	`is_positive, is_negative`	Specify that a symbol is positive or negative.
`integer`	`is_integer`	The symbol represents an integer.
`odd, even`	`is_odd, is_even`	The symbol represents an odd or even integer.
`prime`	`is_prime`	The symbol is a prime number and therefore also an integer.
`finite, infinite`	`is_finite, is_infinite`	The symbol represents a quantity that is finite or infinite.