Тема 15. Виконання символьних обчислень на мові Python – бібліотека модулів SymPy

SymPy є відкритою бібліотекою символьних обчислень мовою Python. SymPy повністю написана мовою Python і не вимагає сторонніх бібліотек.

Нижче, у пізнавальній меті (за матеріалами сайту http://www.asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html) , описані основні можливості пакету SymPy.

Детальніше дивитьсяhttp://www.sympy.org/en/index.htmlihttps://www.sympy.org/ru/index.html

Математична бібліотека Python SymPy

SymPy – це бібліотека Python для виконання символьних обчислень. Це система комп'ютерної алгебри, яка може виступати як окрема програма, так і як бібліотека для інших програм. Попрацювати з нею онлайн можна наhttps://live.sympy.org/. Бо це чистабібліотека Python, можна використовувати навіть в інтерактивному режимі.

У SymPy є різні функції, які застосовуються у сфері символьних обчислень, математичного аналізу, алгебри, дискретної математики, квантової фізики тощо. SymPy може представляти результат у різних форматах: LaTeX, MathML і таке інше. Поширюється бібліотека ліцензії New BSD. Першими цю бібліотеку випустили розробники Ondřej Čertík та Aaron Meurer у 2007 році.

Ось де застосовується SymPy:

- Багаточлени
- Математичний аналіз
- Дискретна математика
- Матриці
- Геометрія
- Побудова графіків
- Фізика
- Статистика
- Комбінаторика

Установка SymPy

Для роботи SymPy потрібна одна важлива бібліотека під назвою mpmath. Вона використовується для речової та комплексної арифметики з числами з плаваючою точкою довільної точності. Однак pip встановить її автоматично під час завантаження самої SymPy:

```
python -m pip install sympy
```

Використання SymPy як калькулятор

SymPy підтримує три типи чисельних даних: Float, Rational та Integer.

Rationalявляє собою звичайний дріб, який задається за допомогою двох цілих чисел: чисельник і знаменник. Наприклад, Rational(1, 2) є дріб 1/2, Rational(5, 2) є дріб 5/2, і так далі.

```
>>> від sympy import Rational
>>> a = Rational(1, 2)

>>> a
1/2

>>> a*2
1

>>> Rational(2)**50/Rational(10)**50
1/88817841970012523233890533447265625
```

Важлива особливість Python-інтерпретатора - при розподілі двох "пітонівських" чисел типу int за допомогою оператора "/" виходить "пітонівський" тип float. Цей стандарт “true division” за замовчуванням включений і в sympy:

```
>>> 1/2
0.5
```

Зверніть увагу, що і в тому, і в іншому випадку ви маєте справу не з об'єктом Number з бібліотеки SymPy, який представляє число в SymPy, а з пітонівськими числами, які створюються самим інтерпретатором Python. Швидше за все, вам потрібно буде працювати з дробовими числами з бібліотеки SymPy, тому для того, щоб отримувати результат у вигляді об'єктів SymPy, переконайтеся, що ви використовуєте клас Rational. Комусь може здатися зручним позначати Rational як R:

```
import sympy
R = sympy.Rational
R(1, 2)
R(1)/2
```

У модулі Sympy є спеціальні константи, такі як e і pi, які поводяться як змінні (тобто вираз 1 + pi не перетворюється відразу в число, а так і залишиться 1 + pi):

```
>>> від sympy import pi, E
>>> pi**2
pi**2

>>> pi.evalf()
```

```
3.14159265358979

>>> (pi + E).evalf()
5.85987448204884
```

як видно, функція evalf переводить вихідний вираз у число з точкою, що плаває. Обчислення можна проводити з більшою точністю. Для цього потрібно передати як аргумент цієї функції необхідну кількість десяткових знаків:

```
>>> pi.evalf(100)
3.14159265358979323846264338327950288419716939937510582
09749445923078164062862089986280348253 82
>>>
```

Для роботи з математичною нескінченністю використовується символ oo:

```
>>> від sympy import oo
>>> oo > 99999
True
>>> oo + 1
oo
```

Змінні

На відміну від інших систем комп'ютерної алгебри, потрібно явно декларувати символьні змінні:

```
>>> від sympy import symbols
>>> x = symbols('x')
>>> y = symbols('y')
```

У лівій частині цього виразу знаходиться змінна Python, яка пітонівським присвоєнням співвідноситься з об'єктом класу Symbol із SymPy.

```
>>> з sympy.abc import x, theta
```

Символьні змінні можуть також задаватися і за допомогою функцій symbols або var. Вони допускають вказівку діапазону. Їхня відмінність полягає в тому, що var додає створені змінні в поточний простір імен:

```
>>> з символів import symbols, var
>>> a, b, c = symbols('a,b,c')
>>> d, e, f = symbols('d:f')
>>> var('g:h')
(g, h)
>>> var('g:2')
(g0, g1)
```

Примірники класу Symbol взаємодіють один з одним. Таким чином, за допомогою них конструюються алгебраїчні вирази:

```
>>> x + y + x - y
2*x

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

```
#розкрити дужки
>>> ((x + y)**2).expand()
x**2 + 2*x*y + y**2
```

Змінні можуть бути замінені іншими змінними, числами або виразами за допомогою функції підстановки subs(old, new):

```
>>> ((x + y)**2).subs(x, 1)
(y + 1)**2

>>> ((x + y)**2).subs(x, y)
4*y**2

>>> ((x + y)**2).subs(x, 1 - y)
1
```

Тепер, з цього моменту, для всіх наведених нижче прикладів ми будемо припускати, що запустили наступну команду з налаштування системи відображення результатів (і використовуємо процедуру pprint):

```
>>> від sympy import init_printing
>>> init_printing(use_unicode=False, wrap_line=False,
no_global=True)
```

Вона надасть якіснішого відображення виразів. Докладніше про систему відображення та друку наведено нижче в розділі Друк. Якщо ж шрифт встановлений з юнікодом, то можна використовувати опцію use_unicode=True для ще красивішого виведення.

Алгебра

Щоб розкласти вираз на найпростіші дроби, використовується функція apart(expr, x):

```
#apart(expr, x):
fromsympy import apart
fromsympy.abc import x, y, z
fromsympy import Integral, pprint
z= 1/( (x + 2)*(x + 1) )
pprint(z)
z=apart(1/( (x + 2)*(x + 1) ), x)
pprint(z)
z=apart((x + 1)/(x - 1), x)
pprint(z)
```

Висновок скрипту:

```
       1
--------------
(x + 1) * (x + 2)
```

```
    1 1
- ----- + -----
  x + 2 x + 1

      2
1 + -----
    x - 1
```

http://www.asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html

Щоб привести дріб до спільного знаменника, використовується функція together(expr, x):

```
>>> від sympy import together
>>> together(1/x + 1/y + 1/z)
x*y+x*z+y*z
---------------
     x*y*z

>>> together(apart((x + 1)/(x - 1), x), x)
x + 1
-----
x - 1

>>> together(apart(1/( (x + 2)*(x + 1) ), x), x)
       1
---------------
(x + 1) * (x + 2)
```

Обчислення

Межі

Для обчислення межі функції використовуйте функцію limit(function, variable, point).

```
Наприклад, щоб обчислити межу f(x) при x -> 0, потрібно
ввести limit(f, x, 0):
>>> від симпи import limit, Symbol, sin, oo
>>> x = Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
```

також можна обчислювати межі при x, що прагне нескінченності:

```
>>> limit(x, x, oo)
oo

>>> limit(1/x, x, oo)
```

```
0

>>> limit(x**x, x, 0)
1
```

Більш складні приклади обчислення меж див.test_demidovich.py

Диференціювання

Продиференціювати будь-який вираз SymPy можна за допомогою diff(func, var). Приклади:

```
>>> від симпи import diff, Symbol, sin, tan
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)

>>> diff(tan(x), x)
   2
tan(x) + 1
```

Можна через межі перевірити правильність обчислень похідної:

```
>>> від sympy import limit
>>> від sympy.abc import delta
>>> limit((tan(x + delta) - tan(x))/delta, delta, 0)
   2
tan(x) + 1
```

Похідні вищих порядків можна обчислити, використовуючи додатковий параметр цієї функції diff(func, var, n):

```
>>> diff(sin(2*x), x, 1)
2*cos(2*x)

>>> diff(sin(2*x), x, 2)
-4*sin(2*x)

>>> diff(sin(2*x), x, 3)
-8*cos(2*x)
```

Розкладання в ряд

Для розкладання в ряд використовуйте метод series(var, point, order):

```
>>> від симпи імпорту Symbol, cos
>>> x = Symbol('x')
>>> cos(x).series(x, 0, 10)

     2  4  6  8
    xxxx/10\
1 - -- + -- - --- + ----- + O\x /
```

```
    2 24 720 40320

>>> (1/cos(x)).series(x, 0, 10)

     2 4 6 8
    x 5*x 61*x 277*x / 10\
1 + -- + ---- + ----- + ------ + O\x /
    2 24 720 8064
```

Ще один простий приклад:

```
>>> від симпи імпорту Integral, pprint

>>> y = Symbol("y")
>>> e = 1/(x + y)
>>> s = e.series(x, 0, 5)

>>> print(s)
1/y - x/y**2 + x**2/y**3 - x**3/y**4 + x**4/y**5 +
O(x**5)
>>> pprint(s)
          2 3 4
1 xxxx/5\
- - - + - - - + - + O\x /
y 2 3 4 5
    yyyy
```

Суми

Обчислює значення суми f(від заданої змінної) на заданих межах

summation(f, (i, a, b))- Знаходження суми доданків f, і змінюється від a до b:

$$\mathbf{summation(f, (i, a, b)) = \sum_{i=a}^{b} f}$$

Якщо він не може compute sum, його деталі відповідають summation formula. Repeated sums can computed by introducing additional limits:

Якщо сума не розраховується, то друкується відповідна формула:

```
>>> з sympy import summation, oo, symbols, log
>>> i, n, m = symbols('in m', integer=True)

>>> summation(2*i - 1, (i, 1, n))
```

```
2
n


>>> summation(1/2**i, (i, 0, oo))
2


>>> summation(1/log(n)**n, (n, 2, oo))
  oo
____
\  `
  \-n
  /log(n)
/__,
n = 2


>>> summation(i, (i, 0, n), (n, 0, m))
      3 2
mmm
-- + -- + -
6 2 3

>>> від sympy.abc import x
>>> з sympy import factorial
>>> summation(x**n/factorial(n), (n, 0, oo))
x
e
```

Інтегрування

SymPy підтримує обчислення певних та невизначених інтегралів за допомогою функції integrate(). Вона використовує розширений алгоритм Ріша-Нормана та деякі шаблони та евристики. Можна обчислювати інтеграли трансцендентних, простих та спеціальних функцій:

```
>>> від симпи імпорту integrate, erf, exp, sin, log,
oo, pi, sinh, symbols
>>> x, y = symbols('x,y')
```

Ви можете інтегрувати найпростіші функції:

```
>>> integrate(6*x**5, x)
6
x
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(log(x), x)
x * log (x) - x
```

```
>>> integrate(2*x + sinh(x), x)
2
x + cosh(x)
```

Приклади інтегрування деяких спеціальних функцій:

```
>>> integrate(exp(-x**2)*erf(x), x)

  ____  2
\/ pi *erf (x)
--------------
      4
```

Можна також обчислити певний інтеграл:

```
>>> integrate(x**3, (x, -1, 1))
0
>>> integrate(sin(x), (x, 0, pi/2))
1
>>> integrate(cos(x), (x, -pi/2, pi/2))
2
```

Підтримуються і невласні інтеграли:

```
>>> integrate(exp(-x), (x, 0, oo))
1
>>> integrate(log(x), (x, 0, 1))
-1
```

Комплексні числа

Крім уявної одиниці I, що є уявним числом, символи теж можуть мати спеціальні атрибути (real, positive, complex тощо), які визначають поведінку цих символів при обчисленні символьних виразів:

```
>>> від симпи імпорту Symbol, exp, I
>>> x = Symbol("x") # a plain x with no attributes
>>> exp(I*x).expand()
I*x
e
>>> exp(I*x).expand(complex=True)

   -im(x) -im(x)
I * e * sin (re (x)) + e * cos (re (x))

>>> x = Symbol("x", real=True)
>>> exp(I*x).expand(complex=True)

I * sin (x) + cos (x)
```

Функції

тригонометричні

```
>>> від sympy import asin, asinh, cos, sin, sinh,
symbols, I
>>> x, y = symbols('x,y')

>>> sin(x + y).expand(trig=True)
sin(x)*cos(y) + sin(y)*cos(x)

>>> cos(x + y).expand(trig=True)
-sin(x)*sin(y) + cos(x)*cos(y)

>>> sin(I*x)
I*sinh(x)

>>> sinh(I*x)
I*sin(x)

>>> asinh(I)
I*pi
----
2

>>> asinh(I*x)
I*asin(x)

>>> sin(x).series(x, 0, 10)

     3 5 7 9
    xxxx/10\
x - -- + --- - ---- + ------ + Ox /
    6 120 5040 362880

>>> sinh(x).series(x, 0, 10)

     3 5 7 9
    xxxx/10\
x + -- + --- + ---- + ------ + Ox /
    6 120 5040 362880

>>> asin(x).series(x, 0, 10)

     3 5 7 9
    x 3*x 5*x 35*x / 10\
```

```
x + -- + ---- + ---- + ----- + Ox /
    6 40 112 1152

>>> asinh(x).series(x, 0, 10)

     3 5 7 9
    x 3*x 5*x 35*x / 10\
x - -- + ---- - ---- + ----- + Ox /
    6 40 112 1152
```

сферичні

```
>>> від sympy import Ylm
>>> з sympy.abc import theta, phi


>>> Ylm(1, 0, theta, phi)


  ___
\/ 3 *cos(theta)
----------------
      _____
    2*\/ pi

>>> Ylm(1, 1, theta, phi)

    ___  I*phi
-\/ 6 *e    *sin(theta)
------------------------
         _____
       4*\/ pi

>>> Ylm(2, 1, theta, phi)

     ____  I*phi
-\/ 30 *e    *sin(theta)*cos(theta)
-----------------------------------
               _____
             4*\/ pi
```

факторіали та гамма-функції

```
>>> від sympy import factorial, gamma, Symbol
>>> x = Symbol("x")
>>> n = Symbol("n", integer=True)

>>> factorial(x)
```

```
x!

>>> factorial(n)
n!

>>> gamma(x + 1).series(x, 0, 3) # ie factorial(x)

                      / 2  2  \
                    2 |EulerGamma pi | / 3\
1 - EulerGamma*x + x *|------------ + ---| + Ox /
                      \  2   12/
```

діта-функції

```
>>> від sympy import zeta
>>> zeta(4, x)
zeta(4, x)

>>> zeta(4, 1)

  4
pi
---
90

>>> zeta(4, 2)

       4
     pi
-1 + ---
     90

>>> zeta(4, 3)

         4
  17 pi
- -- + ---
  16 90
```

багаточлени

```
>>> з імпорту import assoc_legendre, chebyshevt,
legendre, hermite
>>> chebyshevt(2, x)

   2
2*x - 1
```

```
>>> chebyshevt(4, x)

   4 2
8*x - 8*x + 1

>>> legendre(2, x)

   2
3*x 1
---- - -
2 2

>>> legendre(8, x)

      8  6  4  2
6435*x 3003*x 3465*x 315*x 35
------- - ------- + ------- - ------ + ---
  128 32 64 32 128

>>> assoc_legendre(2, 1, x)

        ___________
       / 2
-3*x*\/ - x + 1


>>> assoc_legendre(2, 2, x)

     2
- 3*x + 3

>>> hermite(3, x)

   3
8*x - 12*x
```

Диференціальні рівняння

```
У isympy:
>>> від символів import Function, Symbol, dsolve
>>> f = Function('f')
>>> x = Symbol('x')
>>> f(x).diff(x, x) + f(x)
f(x) + Derivative(f(x), x, x)
>>> від симпи імпорту Integral, pprint
>>> uuu=f(x).diff(x, x) + f(x)
```

```
>>> pprint(uuu)

         2
        d
f(x) + ---(f(x))
         2
       dx

>>> dsolve(f(x).diff(x, x) + f(x), f(x))
Eq(f(x), C1*sin(x) + C2*cos(x))
>>> ud=dsolve(f(x).diff(x, x) + f(x), f(x))
>>> ud
Eq(f(x), C1*sin(x) + C2*cos(x))
>>> pprint(ud)
f(x) = C1*sin(x) + C2*cos(x)
```

Алгебраїчні рівняння

```
>>> від sympy import solve, symbols
>>> x, y = symbols('x,y')
>>> solve(x**4 - 1, x)
[-1, 1, -I, I]

>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{x: -3, y: 1}
```

Лінійна алгебра

Матриці

Матриці задаються за допомогою конструктора Matrix:

```
>>> з імпорту import Matrix, Symbol
>>> Matrix([[1, 0], [0, 1]])
[1 0]
[]
[0 1]
```

У матрицях ви також можете використовувати символьні змінні:

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> A = Matrix([[1, x], [y, 1]])
>>> A
[1 x]
[]
[y 1]

>>> A**2
[x * y + 1 2 * x]
```

```
[]
[ 2*yx*y + 1]
```

Для того, щоб дізнатися про матриці докладніше, прочитайте, будь ласка, Посібник з Лінійної Алгебри.

Зіставлення зі зразком

Щоб зіставити вирази зі зразками, використовуйте .match() разом із допоміжним класом Wild. Ця функція поверне словник із необхідними замінами, наприклад:

```
>>> від симпи імпорту Symbol, Wild
>>> x = Symbol('x')
>>> p = Wild('p')
>>> (5*x**2).match(p*x**2)
{p: 5}

>>> q = Wild('q')
>>> (x**2).match(p*x**q)
{p: 1, q: 2}
```

Якщо ж зіставлення не вдалося, функція поверне "None":

```
>>> print((x + 1).match(p**x))
None
```

Також можна використовувати параметр exclude для виключення деяких значень із результату:

```
>>> p = Wild('p', exclude=[1, x])
>>> print((x + 1).match(x + p)) # 1 is excluded
None
>>> print((x + 1).match(p + 1)) # x is excluded
None
>>> print((x + 1).match(x + 2 + p)) # -1 is not
excluded
{p_: -1}
```

Друк

Реалізовано кілька способів виведення виразів на екран.

Стандартний

Стандартний спосіб представлений функцією str(expression), яка працює наступним чином:

```
>>> from sympy import Integral
>>> від sympy.abc import x
>>> print(x**2)
x**2
>>> print(1/x)
1/x
>>> print(Integral(x**2, x))
```

```
Integral(x**2, x)
```

«Гарний друк»

Цей спосіб друку виразів заснований на ascii-графіці та реалізований через функцію pprint:

```
>>> від симпи імпорту Integral, pprint
>>> від sympy.abc import x
>>> pprint(x**2)

2
X

>>> pprint(1/x)

1
-
X

>>> pprint(Integral(x**2, x))
  /
|
| 2
| x dx
|
/
```

Якщо у вас встановлено шрифт із юнікодом, він буде використовувати Pretty-print із юнікодом за замовчуванням. Це налаштування можна вимкнути за допомогою use_unicode:

```
>>> pprint(Integral(x**2, x), use_unicode=True)
⌠
⎮ 2
⎮ x dx
⌡
```

Для вивчення детальних прикладів роботи Pretty-print з юнікодом ви можете звернутися до статті Pretty Printing (https://github.com/sympy/sympy/wiki/Pretty-Printing) на Вікі.

Для того, щоб зробити pprint за замовчуванням у стандартному інтерпретаторі, виконуємо таку процедуру:

```
>>>від sympy import *
>>>import sys
>>>sys.displayhook = pprint
```

Приклади:

```
>>> від sympy import init_printing, var, Integral
```

```
>>> init_printing(use_unicode=False, wrap_line=False,
no_global=True)

>>> var("x")
X

>>> x**3/3

3
x
-
3

>>> Integral(x**2, x) #doctest: +NORMALIZE_WHITESPACE
  /
|
| 2
| x dx
|
/
```

Друк об'єктів Python

```
>>> від sympy.printing.python import python
>>> from sympy import Integral
>>> від sympy.abc import x
>>> print(python(x**2))
x = Symbol('x')
e = x**2
>>> print(python(1/x))
x = Symbol('x')
e = 1/x
>>> print(python(Integral(x**2, x)))
x = Symbol('x')
e = Integral(x**2, x)
```

Друк у форматі LaTeX

```
>>> від імпорту import Integral, latex
>>> від sympy.abc import x
>>> latex(x**2)
x^{2}
>>> latex(x**2, mode='inline')
$x^{2}$
>>> latex(x**2, mode='equation')
\begin{equation}x^{2}\end{equation}
>>> latex(x**2, mode='equation*')
\begin{equation*}x^{2}\end{equation*}
```

```
>>> latex(1/x)
\frac{1}{x}
>>> latex(Integral(x**2, x))
\int x^{2}\, dx
```

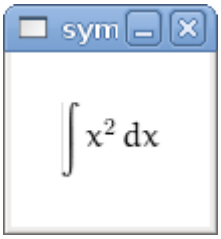
MathML

```
>>> від sympy.printing.mathml import mathml
>>> від імпорту import Integral, latex
>>> від sympy.abc import x
>>> print(mathml(x**2))
<apply><power/><ci>x</ci><cn>2</cn></apply>
>>> print(mathml(1/x))
<apply><power/><ci>x</ci><cn>-1</cn></apply>
```

Pyglet

```
>>> з імпорту import Integral, preview
>>> від sympy.abc import x
>>> preview(Integral(x**2, x))
```

З'явиться вікно pyglet з відмальованим виразом LaTeX:



Примітки

Якщо встановлені додаткові пакети.

isympyвикликає pprint автоматично, тому Pretty-print буде включено в isympy за замовчуванням.

Також доступний модуль друку – sympy.printing. Через цей модуль доступні наступні функції друку:

`pretty(expr), pretty_print(expr), pprint(expr)` –

Повертає або виводить на екран, відповідно, "Гарне" подання виразу expr.

`latex(expr), print_latex(expr)` –

Повертає або виводить на екран, відповідно,LaTeX-подання expr

`mathml(expr), print_mathml(expr)` –

Повертає або виводить на екран, відповідно, MathML (http://www.w3.org/Math/) -подання expr.

Щоб дізнатися про SymPy докладніше, зверніться:

Посібник користувача SymPy

http://www.asmeurer.com/sympy_doc/dev-py3k/guide.html

та Опис модулів
SymPy.http://www.asmeurer.com/sympy_doc/dev-py3k/modules/index.html

Також можна звернутися на wiki.sympy.org - сайт, який містить безліч корисних прикладів, посібників та порад.

Приклади застосування пакету SymPy

SymPy - це пакет для символьних обчислень на пітоні, подібний до системи Mathematica. Він працює з виразами, що містять символи.

```
from sympy import *
init_printing()
```

Основними цеглинами, з яких будуються вирази, є символи. Символ має ім'я, яке використовується для друку виразів. Об'єкти класу Symbol потрібно створювати та присвоювати змінним пітонам, щоб їх можна було використовувати.

В принципі, ім'я символу та ім'я змінної, якою ми присвоюємо цей символ, - дві незалежні речі, і можна написати abc=Symbol('xyz').

Але тоді при введенні програми Ви будете використовувати abc, а під час друкування результатів SymPy буде використовувати xyz, що призведе до непотрібної плутанини. Тому краще, щоб ім'я символу збігалося з ім'ям змінної пітона, якому він надається.

У мовах, спеціально призначених для символьних обчислень, таких як Mathematica, якщо Ви використовуєте змінну, якої нічого не було присвоєно, вона автоматично сприймається як символ з тим же ім'ям. Пітон був спочатку призначений для символьних обчислень. Якщо Ви використовуєте змінну, якій нічого не було надано, Ви отримаєте повідомлення про помилку. Об'єкти типу Symbol слід створювати явно.

```
x=Symbol('x')
```

```
a=x**2-1
a
```

$$x^2 - 1$$

```
type(a)
```

```
sympy.core.add.Add
```

Можна визначити кілька символів одночасно. Рядок розбивається на імена за пробілами.

```
y,z=symbols('y z')
```

Підставимо замість x вираз y+1

```
a.subs(x,y+1)
```

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

Багаточлени та раціональні функції

SymPy не розкриває дужки автоматично. Для цього використовується функція expand.

```
a=(x+y-z)**6
a
```

$$(x+y-z)^6$$

```
a=expand(a)
a
```

$$x^6+6x^5y-6x^5z+15x^4y^2-30x^4yz+15x^4z^2+20x^3y^3-60x^3y^2z+60x^3yz^2-20x^3z^3+15x^2y^4-60x^2y^3z+90x^2y^2z^2-60x^2yz^3+15x^2z^4+6xy^5-30xy^4z+60xy^3z^2-60xy^2z^3+30xyz^4-6xz^5+y^6-6y^5z+15y^4z^2-20y^3z^3+15y^2z^4-6yz^5+z^6$$

Ступінь многочлена a за x degree

```
degree(a,x)
```

6

Зберемо разом члени з певними ступенями x collect

```
collect(a,x)
```

$$x^6+x^5(6y-6z)+x^4(15y^2-30yz+15z^2)+x^3(20y^3-60y^2z+60yz^2-20z^3)+x^2(15y^4-60y^3z+90y^2z^2-60yz^3+15z^4)+x(6y^5-30y^4z+60y^3z^2-60y^2z^3+30yz^4-6z^5)+y^6-6y^5z+15y^4z^2-20y^3z^3+15y^2z^4-6yz^5+z^6$$

Багаточлен з цілими коефіцієнтами можна записати у вигляді добутку таких багаточленів (причому кожен співмножник вже неможливо розфакторизувати далі, залишаючись у рамках багаточленів з цілими коефіцієнтами).

Існують ефективні алгоритми для вирішення цієї задачі – factor.

```
a=factor(a)
a
```

$$(x+y-z)^6$$

SymPy не скорочує відносини багаточленів на їхній найбільший спільний дільник автоматично. Для цього використовується функція cancel.

```
a=(x**3-y**3)/(x**2-y**2)
a
```

$$\frac{x^3-y^3}{x^2-y^2}$$

```
cancel(a)
```

$$\frac{x^2+xy+y^2}{x+y}$$

SymPy не призводить до суми раціональних виразів до спільного знаменника автоматично. Для цього використовується функція доіншого.

```
a=y/(x-y)+x/(x+y)
a
```

$$\frac{x}{x+y}+\frac{y}{x-y}$$

```
together(a)
```

$$\frac{x(x-y)+y(x+y)}{(x-y)(x+y)}$$

Функція simplify намагається переписати вираз у найпростішому вигляді. Це поняття немає чіткого визначення (у різних ситуаціях найпростішими можуть вважатися різні форми висловлювання), немає алгоритму такого спрощення.

Функція symplify працює евристично, і неможливо передбачити, які спрощення вона спробує зробити. Тому її зручно використовувати в інтерактивних сесіях, щоб подивитися, чи вдасться їй записати вираз у якомусь розумному вигляді, але небажано використовувати у програмах. Вони краще застосовувати більш спеціалізовані функції, які виконують одне певне перетворення висловлювання.

```
simplify(a)
```

$$\frac{x^2+y^2}{x^2-y^2}$$

Розкладання на елементарні дроби по відношенню до x та y - apart

```
apart(a,x)
```

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

```
apart(a,y)
```

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

Підставимо конкретні чисельні значення замість змінних x та y subs

```
a=a.subs({x:1,y:2})
a
```

$$-\frac{5}{3}$$

А скільки це буде чисельно? Метод n()

```
a.n()
```

$$-1.66666666666667$$

Елементарні функції

SymPy автоматично застосовує спрощення елементарних функцій (які справедливі у всіх випадках).

```
sin(-x)
```

$$-\sin(x)$$

```
cos(pi/4),tan(5*pi/6)
```

$$\left(\frac{\sqrt{2}}{2}, \quad -\frac{\sqrt{3}}{3}\right)$$

SymPy може працювати з числами з плаваючою точкою, що мають як завгодно велику точність. Ось π із 100 значущими цифрами – метод n(100).

```
pi.n(100)
```

$$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534$$

E - це основа натуральних логарифмів.

```
log(1),log(E)
```

$$(0, \quad 1)$$

```
exp(log(x)),log(exp(x))
```

$$(x, \quad \log(e^x))$$

```
sqrt(0)
```

$$0$$

```
sqrt(x)**4,sqrt(x**4)
```

$$\left(x^2, \quad \sqrt{x^4}\right)$$

Символи можуть мати деякі властивості. Наприклад, вони можуть бути позитивними. Тоді SymPy може сильніше спростити квадратне коріння.

```
p,q=symbols('p q',positive=True)
sqrt(p**2)
```

$$p$$

```
sqrt(12*x**2*y),sqrt(12*p**2*y)
```

$$\left(2\sqrt{3}\sqrt{x^2y}, \quad 2\sqrt{3}p\sqrt{y}\right)$$

Нехай символ n буде цілим (I – це уявна одиниця).

```
n=Symbol('n',integer=True)
simplify(exp(2*pi*I*n))
```

$$1$$

```
sin(pi*n)
```

$$0$$

Метод rewrite намагається переписати вираз у термінах заданої функції.

```
cos(x).rewrite(exp),exp(I*x).rewrite(cos)
```

$$\left(\frac{e^{ix}}{2} + \frac{1}{2}e^{-ix}, \quad i\sin(x) + \cos(x)\right)$$

```
asin(x).rewrite(log)
```

$$-i\log\left(ix + \sqrt{-x^2+1}\right)$$

Функція trigsimp намагається переписати тригонометричний вираз у найбільш простому вигляді. У програмах краще використовувати спеціалізовані функції.

```
trigsimp(2*sin(x)**2+3*cos(x)**2)
```

$$\cos^2(x) + 2$$

Функція expand_trig розкладає синуси та косинуси сум та кратних кутів.

```
expand_trig(sin(x-y)),expand_trig(sin(2*x))
```

$$(\sin(x)\cos(y) - \sin(y)\cos(x), \quad 2\sin(x)\cos(x))$$

Найчастіше потрібно зворотне перетворення - творів і ступенів синусів і косінусів у вирази, лінійні за цими функціями.

Наприклад, нехай ми працюємо з відрізком низки Фур'є.

```
a1,a2,b1,b2=symbols('a1 a2 b1 b2')
a=a1*cos(x)+a2*cos(2*x)+b1*sin(x)+b2*sin(2*x)
a
```

$$a_1\cos(x) + a_2\cos(2x) + b_1\sin(x) + b_2\sin(2x)$$

Ми хочемо звести його у квадрат і знову отримати відрізок ряду Фур'є.

```
a=(a**2).rewrite(exp).expand().rewrite(cos).expand()
a
```

$$\frac{a_1^2}{2}\cos(2x) + \frac{a_1^2}{2} + a_1a_2\cos(x) + a_1a_2\cos(3x) + a_1b_1\sin(2x) + a_1b_2\sin(x) + a_1b_2\sin(3x) + \frac{a_2^2}{2}\cos(4x) + \frac{a_2^2}{2} - a_2b_1\sin(x) + a_2b_1\sin(3x) + a_2b_2\sin(4x) - \frac{b_1^2}{2}\cos(2x) + \frac{b_1^2}{2} + b_1b_2\cos(x) - b_1b_2\cos(3x) - \frac{b_2^2}{2}\cos(4x) + \frac{b_2^2}{2}$$

```
a.collect([cos(x),cos(2*x),cos(3*x),sin(x),sin(2*x),sin(3*x)])
```

$$\frac{a_1^2}{2} + a_1b_1\sin(2x) + \frac{a_2^2}{2}\cos(4x) + \frac{a_2^2}{2} + a_2b_2\sin(4x) + \frac{b_1^2}{2} - \frac{b_2^2}{2}\cos(4x) + \frac{b_2^2}{2} + \left(\frac{a_1^2}{2} - \frac{b_1^2}{2}\right)\cos(2x) + (a_1a_2 - b_1b_2)\cos(3x) + (a_1a_2 + b_1b_2)\cos(x) + (a_1b_2 - a_2b_1)\sin(x) + (a_1b_2 + a_2b_1)\sin(3x)$$

Функція expand_log перетворює логарифми творів та ступенів у суми логарифмів (тільки для позитивних величин);

logcombineздійснює зворотне перетворення.

```
a=expand_log(log(p*q**2))
a
```

$$\log(p) + 2\log(q)$$

```
logcombine(a)
```

$$\log(pq^2)$$

Функція expand_power_exp переписує ступеня, показники яких - суми через твори ступенів.

```
expand_power_exp(x**(p+q))
```

$$x^p x^q$$

Функція expand_power_base переписує ступеня, основи яких – твори, через твори ступенів.

```
expand_power_base((x*y)**n)
```

$$x^n y^n$$

Функція powsimp виконує зворотні перетворення.

```
powsimp(exp(x)*exp(2*y)),powsimp(x**n*y**n)
```

$$\left(e^{x+2y},\quad (xy)^n\right)$$

Можна вводити функції користувача. Вони можуть мати довільну кількість аргументів.

```
f=Function('f')
f(x)+f(x,y)
```

$$f(x) + f(x, y)$$

Структура виразів

Внутрішнє уявлення виразу – це дерево. Функція srepr повертає рядок, що його представляє.

```
srepr(x+1)
```

```
"Add(Symbol('x'), Integer(1))"
```

```
srepr(x-1)
```

```
"Add(Symbol('x'), Integer(-1))"
```

```
srepr(x-y)
```

```
"Add(Symbol('x'), Mul(Integer(-1), Symbol('y')))"
```

```
srepr(2*x*y/3)
```

```
"Mul(Rational(2, 3), Symbol('x'), Symbol('y'))"
```

```
srepr(x/y)
```

```
"Mul(Symbol('x'), Pow(Symbol('y'), Integer(-1)))"
```

Замість бінарних операцій +, *, ** і т.д. можна використовувати функції Add, Mul, Pow тощо.

```
Mul(x,Pow(y,-1))==x/y
```

```
True
```

```
srepr(f(x,y))
```

```
"Function('f')(Symbol('x'), Symbol('y'))"
```

Атрибут func – це функція верхнього рівня у виразі, а args – список її аргументів.

```
a=2*x*y**2
a.func
```

```
sympy.core.mul.Mul
```

```
a.args
```

$$(2,\quad x,\quad y^2)$$

```
a.args[0]
```

$$2$$

```
for i in a.args:
    print(i)
```

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

Функція subs замінює змінну вираз.

```
a.subs(y,2)
```

$$8x$$

Вона може замінити кілька змінних. Для цього їй передається список кортежів чи словник.

```
a.subs([(x,pi),(y,2)])
```

$$8\pi$$

```
a.subs({x:pi,y:2})
```

$$8\pi$$

Вона може замінити не змінну, а подвираженіе - функцію з аргументами.

```
a=f(x)+f(y)
a.subs(f(y),1)
```

$$f(x)+1$$

```
(2*x*y*z).subs(x*y,z)
```

$$2z^2$$

```
(x+x**2+x**3+x**4).subs(x**2,y)
```

$$x^3+x+y^2+y$$

Підстановки виконуються послідовно. У разі спочатку x замінився на y, вийшло y3+y2; потім у цьому результаті y замінилося на x

```
a=x**2+y**3
a.subs([(x,y),(y,x)])
```

$$x^3+x^2$$

Якщо написати ці підстановки в іншому порядку, результат буде іншим.

```
a.subs([(y,x),(x,y)])
```

$$y^3+y^2$$

Але можна передати функції subs ключовий параметр simultaneous = True, тоді підстановки будуть виконуватися одночасно. Таким чином можна, наприклад, поміняти місцями x та y

```
a.subs([(x,y),(y,x)],simultaneous=True)
```

$$x^3+y^2$$

Можна замінити функцію іншою функцією.

```
g=Function('g')
a=f(x)+f(y)
a.subs(f,g)
```

$$g(x)+g(y)$$

Метод replace шукає подвыражения, відповідні зразку (який містить довільні змінні), і замінює їх у заданий вираз (воно може містити самі довільні змінні).

```
a=Wild('a')
(f(x)+f(x+y)).replace(f(a),a**2)
```

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

```
(f(x,x)+f(x,y)).replace(f(a,a),a**2)
```

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

```
a=x**2+y**2
a.replace(x,x+1)
```

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

Відповідати зразку має ціле подвыражение, це може бути частина співмножників у творі чи менша ступінь більшою.

```
a=2*x*y*z
a.replace(x*y,z)
```

$$2xyz$$

```
(x+x**2+x**3+x**4).replace(x**2,y)
```

$$x^4 + x^3 + x + y$$

Розв'язання рівнянь

```
a,b,c,d,e,f=symbols('a b c d e f')
```

Рівняння записується як функція Eq із двома параметрами. Функція solve повертає перелік рішень.

```
solve(Eq(a*x,b),x)
```

$$\left[\frac{b}{a}\right]$$

Втім, можна передати функції solve просто вираз. Мається на увазі рівняння, що цей вираз дорівнює 0.

```
solve(a*x+b,x)
```

$$\left[-\frac{b}{a}\right]$$

Квадратне рівняння має два рішення.

```
solve(a*x**2+b*x+c,x)
```

$$\left[\frac{1}{2a}\left(-b+\sqrt{-4ac+b^2}\right),\quad -\frac{1}{2a}\left(b+\sqrt{-4ac+b^2}\right)\right]$$

Система лінійних рівнянь.

```
solve([a*x+b*y-e,c*x+d*y-f],[x,y])
```

$$\left\{x:\frac{-bf+de}{ad-bc},\quad y:\frac{af-ce}{ad-bc}\right\}$$

Функція roots повертає коріння багаточлена зі своїми кратностями.

```
roots(x**3-3*x+2,x)
```

$$\{-2:1,\quad 1:2\}$$

Функція solve_poly_system вирішує систему поліноміальних рівнянь, будуючи їх базис Гребнера.

```
p1=x**2+y**2-1
p2=4*x*y-1
solve_poly_system([p1,p2],x,y)
```

$$\left[\left(4\left(-1-\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(-\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad -\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right),\right.$$
$$\left(-4\left(-1+\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad \sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right),$$
$$\left(4\left(-1-\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(-\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad -\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right),$$
$$\left.\left(-4\left(-1+\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad \sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\right]$$

Ряди

```
exp(x).series(x,0,5)
```

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \mathcal{O}\left(x^5\right)$$

Ряд може починатися з негативного ступеня.

```
cot(x).series(x,n=5)
```

$$\frac{1}{x} - \frac{x}{3} - \frac{x^3}{45} + \mathcal{O}\left(x^5\right)$$

І навіть йти по підлозі цілими ступенями.

```
sqrt(x*(1-x)).series(x,n=5)
```

$$\sqrt{x} - \frac{x^{\frac{3}{2}}}{2} - \frac{x^{\frac{5}{2}}}{8} - \frac{x^{\frac{7}{2}}}{16} - \frac{5x^{\frac{9}{2}}}{128} + \mathcal{O}\left(x^5\right)$$

```
log(gamma(1+x)).series(x,n=6).rewrite(zeta)
```

$$-\gamma x + \frac{\pi^2 x^2}{12} - \frac{x^3\zeta\left(3\right)}{3} + \frac{\pi^4 x^4}{360} - \frac{x^5\zeta\left(5\right)}{5} + \mathcal{O}\left(x^6\right)$$

Підготуємо 3 ряди.

```
sinx=series(sin(x),x,0,8)
sinx
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \mathcal{O}\left(x^8\right)$$

```
cosx=series(cos(x),x,n=8)
cosx
```

$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \mathcal{O}\left(x^8\right)$$

```
tanx=series(tan(x),x,n=8)
tanx
```

$$x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \mathcal{O}\left(x^8\right)$$

Твори та приватні ряди не обчислюються автоматично, до них треба застосувати функцію series.

```
series(tanx*cosx,n=8)
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \mathcal{O}\left(x^8\right)$$

```
series(sinx/cosx,n=8)
```

$$x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \mathcal{O}\left(x^8\right)$$

А цей ряд повинен дорівнювати 1. Але оскільки sinx і cosx відомі лише з обмеженою точністю, ми отримуємо 1 з тією ж точністю.

```
series(sinx**2+cosx**2,n=8)
```

$$1 + \mathcal{O}\left(x^8\right)$$

Тут перші члени скоротилися, і відповідь можна отримати лише з меншою точністю.

```
series((1-cosx)/x**2,n=6)
```

$$\frac{1}{2} - \frac{x^2}{24} + \frac{x^4}{720} + \mathcal{O}\left(x^6\right)$$

Ряди можна диференціювати та інтегрувати.

```
diff(sinx,x)
```

$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \mathcal{O}\left(x^7\right)$$

```
integrate(cosx,x)
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \mathcal{O}\left(x^9\right)$$

Можна підставити ряд (якщо він починається з малого члена) замість змінного розкладання до іншого ряду. Ось ряди для sin(tan(x)) та tan(sin(x))

```
st=series(sinx.subs(x,tanx),n=8)
st
```

$$x + \frac{x^3}{6} - \frac{x^5}{40} - \frac{55x^7}{1008} + \mathcal{O}\left(x^8\right)$$

```
ts=series(tanx.subs(x,sinx),n=8)
ts
```

$$x + \frac{x^3}{6} - \frac{x^5}{40} - \frac{107x^7}{5040} + \mathcal{O}\left(x^8\right)$$

```
series(ts-st,n=8)
```

$$\frac{x^7}{30} + \mathcal{O}\left(x^8\right)$$

У ряд не можна підставляти чисельне значення змінної розкладання (отже, не можна будувати графік). Для цього потрібно спочатку прибрати $\mathcal{O}$ член, перетворивши відрізок низки багаточлен.

```
a=sinx.removeO()
```

```
a.subs(x,0.1)
```

0.0998334166468254

Похідні

```
a=x*sin(x+y)
diff(a,x)
```

$$x\cos\left(x+y\right) + \sin\left(x+y\right)$$

```
diff(a,y)
```

$$x\cos\left(x+y\right)$$

Друга похідна x і перша y

```
diff(a,x,2,y)
```

$$-x\cos\left(x+y\right) + 2\sin\left(x+y\right)$$

Можна диференціювати вирази, що містять певні функції.

```
a=x*f(x**2)
b=diff(a,x)
b
```

$$2x^2 \frac{d}{d\xi_1} f(\xi_1)\Big|_{\xi_1=x^2} + f\left(x^2\right)$$

Що це за «звір» такий вийшов?

```
print(b)
```

```
2*x**2*Subs(Derivative(f(_xi_1), _xi_1), (_xi_1,), (x**2,)) + f(x**2)
```

Функція Derivative є не обчисленою похідною. Її можна визначити методом doit.

```
a=Derivative(sin(x),x)
Eq(a,a.doit())
```

$$\frac{d}{dx}\sin(x) = \cos(x)$$

Інтеграли

```
integrate(1/(x*(x**2-2)**2),x)
```

$$\frac{1}{4}\log(x) - \frac{1}{8}\log(x^2 - 2) - \frac{1}{4x^2 - 8}$$

```
integrate(1/(exp(x)+1),x)
```

$$x - \log(e^x + 1)$$

```
integrate(log(x),x)
```

$$x\log(x) - x$$

```
integrate(x*sin(x),x)
```

$$-x\cos(x) + \sin(x)$$

```
integrate(x*exp(-x**2),x)
```

$$-\frac{e^{-x^2}}{2}$$

```
a=integrate(x**x,x)
a
```

$$\int x^x\,dx$$

Вийшов невизначений інтеграл.

```
print(a)
```

```
Integral(x**x, x)
```

```
a=Integral(sin(x),x)
Eq(a,a.doit())
```

$$\int \sin(x)\,dx = -\cos(x)$$

Певні інтеграли.

```
integrate(sin(x),(x,0,pi))
```

$$2$$

oo - это ∞.

```
integrate(exp(-x**2),(x,0,oo))
```

$$\frac{\sqrt{\pi}}{2}$$

```
integrate(log(x)/(1-x),(x,0,1))
```

$$-\frac{\pi^2}{6}$$

Підсумовування рядів

```
summation(1/n**2,(n,1,oo))
```

$$\frac{\pi^2}{6}$$

```
summation((-1)**n/n**2,(n,1,oo))
```

$$-\frac{\pi^2}{12}$$

```
summation(1/n**4,(n,1,oo))
```

$$\frac{\pi^4}{90}$$

Чи не обчислена сума позначається Sum.

```
a=Sum(x**n/factorial(n),(n,0,oo))
Eq(a,a.doit())
```

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x$$

Межі

```
limit((tan(sin(x))-sin(tan(x)))/x**7,x,0)
```

$$\frac{1}{30}$$

Це проста межа, він вважається розкладанням чисельника та знаменника в ряди. А якщо x=0, то з'являється особлива точка.

Знайдемо односторонні межі.

```
limit((tan(sin(x))-sin(tan(x)))/(x**7+exp(-1/x)),x,0,'+')
```

$$\frac{1}{30}$$

```
limit((tan(sin(x))-sin(tan(x)))/(x**7+exp(-1/x)),x,0,'-')
```

$$0$$

Диференціальні рівняння

```
t=Symbol('t')
x=Function('x')
p=Function('p')
```

Перший порядок.

```
dsolve(diff(x(t),t)+x(t),x(t))
```

$$x(t) = C_1 e^{-t}$$

Другий порядок.

```
dsolve(diff(x(t),t,2)+x(t),x(t))
```

$$x(t) = C_1 \sin(t) + C_2 \cos(t)$$

Система рівнянь першого ладу.

```
dsolve((diff(x(t),t)-p(t),diff(p(t),t)+x(t)))
```

$$[x(t) = C_1 \sin(t) + C_2 \cos(t), \quad p(t) = C_1 \cos(t) - C_2 \sin(t)]$$

Лінійна алгебра

```
a,b,c,d,e,f=symbols('a b c d e f')
```

Матрицю можна побудувати зі списку списків.

```
M=Matrix([[a,b,c],[d,e,f]])
M
```

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

```
M.shape
```

$$(2, \quad 3)$$

Матриця-рядок.

```
Matrix([[1,2,3]])
```

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

Матриця-стовпець.

```
Matrix([1,2,3])
```

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

Можна побудувати матрицю із функції.

```
def g(i,j):
    return Rational(1,i+j+1)
Matrix(3,3,g)
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

Або з невизначеної функції.

```
g=Function('g')
M=Matrix(3,3,g)
M
```

$$\begin{bmatrix} g(0,0) & g(0,1) & g(0,2) \\ g(1,0) & g(1,1) & g(1,2) \\ g(2,0) & g(2,1) & g(2,2) \end{bmatrix}$$

```
M[1,2]
```

$$g(1,2)$$

```
M[1,2]=0
M
```

$$\begin{bmatrix} g(0,0) & g(0,1) & g(0,2) \\ g(1,0) & g(1,1) & 0 \\ g(2,0) & g(2,1) & g(2,2) \end{bmatrix}$$

```
M[2,:]
```

$$\begin{bmatrix} g(2,0) & g(2,1) & g(2,2) \end{bmatrix}$$

```
M[:,1]
```

$$\begin{bmatrix} g(0,1) \\ g(1,1) \\ g(2,1) \end{bmatrix}$$

```
M[0:2,1:3]
```

$$\begin{bmatrix} g(0,1) & g(0,2) \\ g(1,1) & 0 \end{bmatrix}$$

Поодинока матриця.

```
eye(3)
```

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

Матриця з нулів.

```
zeros(3)
```

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

```
zeros(2,3)
```

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

Діагональна матриця.

```
diag(1,2,3)
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

```
M=Matrix([[a,1],[0,a]])
diag(1,M,2)
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 1 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Операції із матрицями.

```
A=Matrix([[a,b],[c,d]])
B=Matrix([[1,2],[3,4]])
A+B
```

$$\begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix}$$

```
A*B,B*A
```

$$\left(\begin{bmatrix} a+3b & 2a+4b \\ c+3d & 2c+4d \end{bmatrix}, \quad \begin{bmatrix} a+2c & b+2d \\ 3a+4c & 3b+4d \end{bmatrix}\right)$$

```
A*B-B*A
```

$$\begin{bmatrix} 3b-2c & 2a+3b-2d \\ -3a-3c+3d & -3b+2c \end{bmatrix}$$

```
simplify(A**(-1))
```

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

```
det(A)
```

$$ad - bc$$

Власні значення та вектори

```
x=Symbol('x',real=True)
```

```
M=Matrix([[(1-x)**3*(3+x),4*x*(1-x**2),-2*(1-x**2)*(3-x)],
          [4*x*(1-x**2),-(1+x)**3*(3-x),2*(1-x**2)*(3+x)],
          [-2*(1-x**2)*(3-x),2*(1-x**2)*(3+x),16*x]])
M
```

$$\begin{bmatrix} (-x+1)^3(x+3) & 4x\left(-x^2+1\right) & (-x+3)\left(2x^2-2\right) \\ 4x\left(-x^2+1\right) & -(-x+3)(x+1)^3 & (x+3)\left(-2x^2+2\right) \\ (-x+3)\left(2x^2-2\right) & (x+3)\left(-2x^2+2\right) & 16x \end{bmatrix}$$

```
det(M)
```

$$0$$

Значить, ця матриця має нульовий підпростір (вона звертає вектори з цього підпростору в 0). Базис цього підпростору.

```
v=M.nullspace()
len(v)
```

$$1$$

Воно одномірне.

```
v=simplify(v[0])
v
```

$$\begin{bmatrix} -\frac{2}{x-1} \\ \frac{2}{x+1} \\ 1 \end{bmatrix}$$

Проверим.

Перевіримо.

```
simplify(M*v)
```

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

Власні значення та його кратності.

```
M.eigenvals()
```

$$\left\{0 : 1, \quad -\left(x^2+3\right)^2 : 1, \quad \left(x^2+3\right)^2 : 1\right\}$$

Якщо потрібні не лише власні значення, а й власні вектори, то потрібно використовувати метод eigenvects. Він повертає список кортежів. У кожному їх нульовий елемент - власне значення, перший - його кратність, і останній - перелік власних векторів, що утворюють базис (їх стільки, яка кратність).

```
v=M.eigenvects()
len(v)
```

3

```
for i in range(len(v)):
    v[i][2][0]=simplify(v[i][2][0])
v
```

$$\left[\left(0, \quad 1, \quad \left[\begin{bmatrix} -\frac{2}{x-1} \\ \frac{2}{x+1} \\ 1 \end{bmatrix}\right]\right), \quad \left(-\left(x^2+3\right)^2, \quad 1, \quad \left[\begin{bmatrix} \frac{x}{2}+\frac{1}{2} \\ \frac{x+1}{x-1} \\ 1 \end{bmatrix}\right]\right), \quad \left(\left(x^2+3\right)^2, \quad 1, \quad \left[\begin{bmatrix} \frac{x-1}{x+1} \\ -\frac{x}{2}+\frac{1}{2} \\ 1 \end{bmatrix}\right]\right)\right]$$

Перевіримо.

```
for i in range(len(v)):
    z=M*v[i][2][0]-v[i][0]*v[i][2][0]
    pprint(simplify(z))
```

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

Жорданова нормальна форма

```
M=Matrix([[Rational(13,9),-Rational(2,9),Rational(1,3),Rational(4,9),Rational(2,3)],
          [-Rational(2,9),Rational(10,9),Rational(2,15),-Rational(2,9),-Rational(11,15)],
          [Rational(1,5),-Rational(2,5),Rational(41,25),-Rational(2,5),Rational(12,25)],
          [Rational(4,9),-Rational(2,9),Rational(14,15),Rational(13,9),-Rational(2,15)],
          [-Rational(4,15),Rational(8,15),Rational(12,25),Rational(8,15),Rational(34,25)]])
M
```

$$\begin{bmatrix} \frac{13}{9} & -\frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \frac{2}{3} \\ -\frac{2}{9} & \frac{10}{9} & \frac{2}{15} & -\frac{2}{9} & -\frac{11}{15} \\ \frac{1}{5} & -\frac{2}{5} & \frac{41}{25} & -\frac{2}{5} & \frac{12}{25} \\ \frac{4}{9} & -\frac{2}{9} & \frac{14}{15} & \frac{13}{9} & -\frac{2}{15} \\ -\frac{4}{15} & \frac{8}{15} & \frac{12}{25} & \frac{8}{15} & \frac{34}{25} \end{bmatrix}$$

Метод M.jordan_form() повертає пару матриць, матрицю перетворення P та власне жорданову форму J: M=PJP−1

```
P,J=M.jordan_form()
J
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1-i & 0 \\ 0 & 0 & 0 & 0 & 1+i \end{bmatrix}$$

```
P=simplify(P)
P
```

$$\begin{bmatrix} -2 & \frac{10}{9} & 0 & \frac{5i}{12} & -\frac{5i}{12} \\ -2 & -\frac{5}{9} & 0 & -\frac{5i}{6} & \frac{5i}{6} \\ 0 & 0 & \frac{4}{3} & -\frac{3}{4} & -\frac{3}{4} \\ 1 & \frac{10}{9} & 0 & -\frac{5i}{6} & \frac{5i}{6} \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Перевіримо.

```
Z=P*J*P**(-1)-M
simplify(Z)
```

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

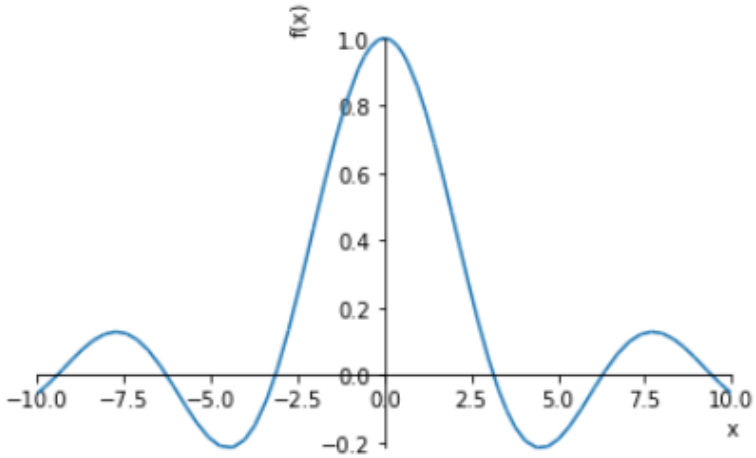
Графіки

SymPy використовує matplotlib. Однак він розподіляє точки по x адаптивно, а не рівномірно.

%matplotlib inline

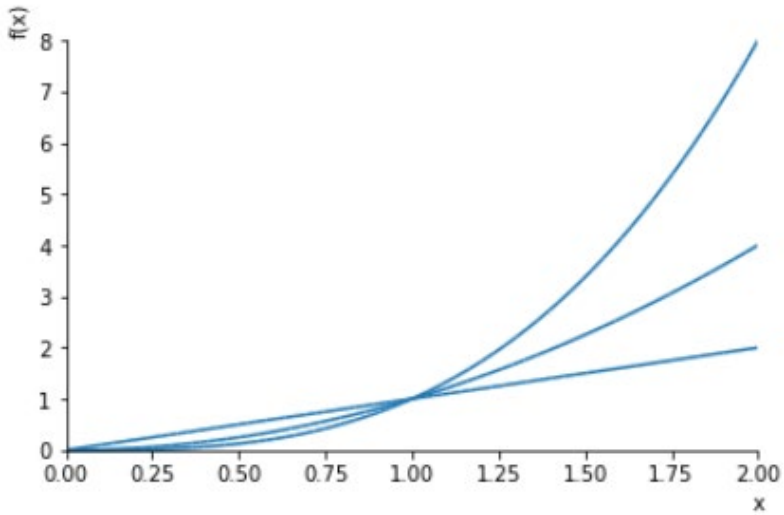
Одна функція.

plot(sin(x)/x,(x,-10,10))



Декілька функцій.

plot(x,x**2,x**3,(x,0,2))

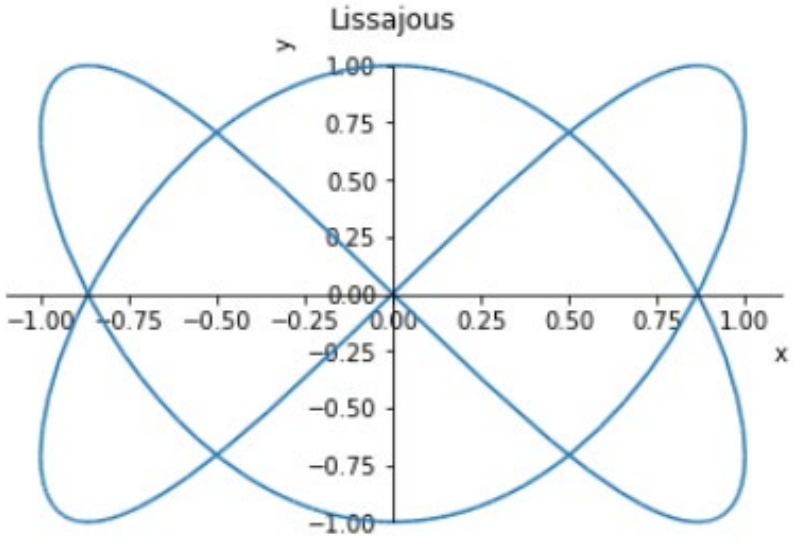


Інші функції слід імпортувати з пакету sympy.plotting.

```
від sympy.plotting import
(plot_parametric,plot_implicit,
plot3d,plot3d_parametric_line,
plot3d_parametric_surface)
```

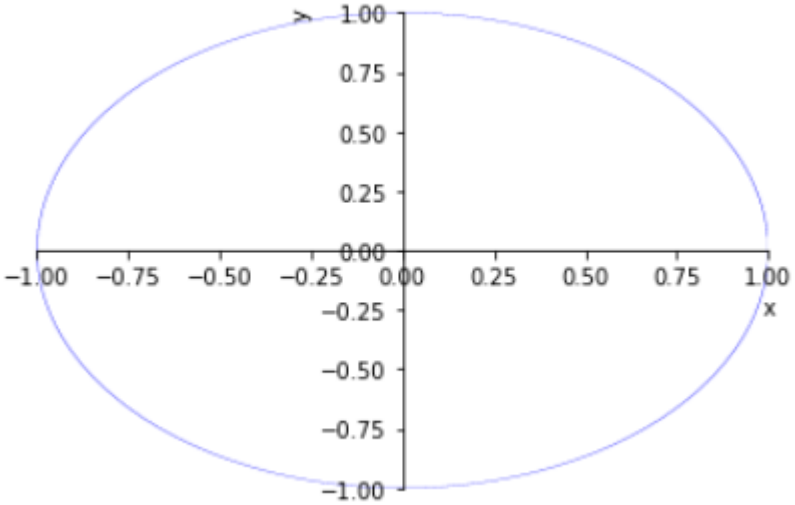
Параметричний графік- фігура Ліссажу.

```
t=Symbol('t')
plot_parametric(sin(2*t),cos(3*t),(t,0,2*pi),
          title='Lissajous',xlabel='x',ylabel='y')
```



Неявний графік- Коло.

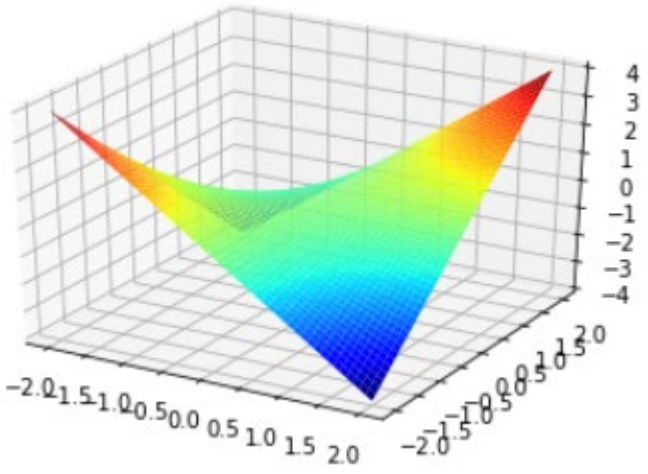
```
plot_implicit(x**2+y**2-1,(x,-1,1),(y,-1,1))
```



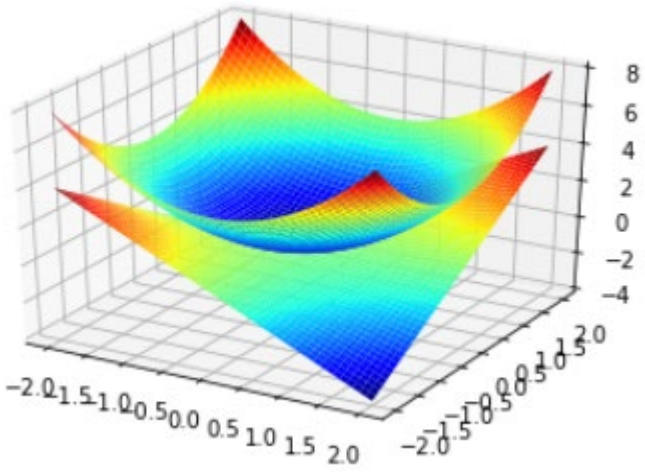
Поверхня.

Якщо вона будується не inline, а в окремому вікні, її можна крутити мишкою.

```
plot3d(x*y,(x,-2,2),(y,-2,2))
```

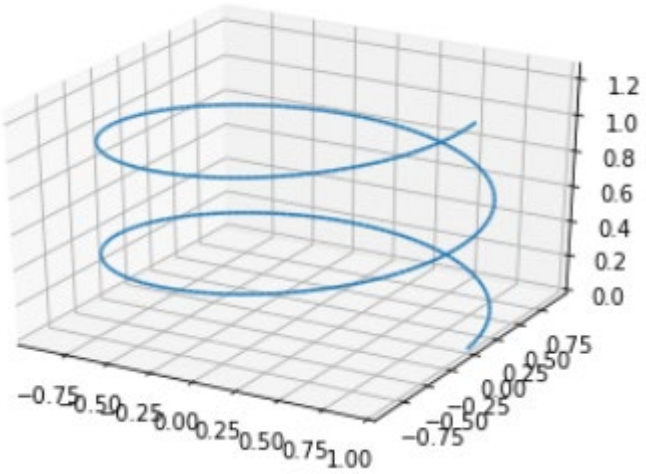


Декілька поверхонь.
plot3d(x**2+y**2,x*y,(x,-2,2),(y,-2,2))



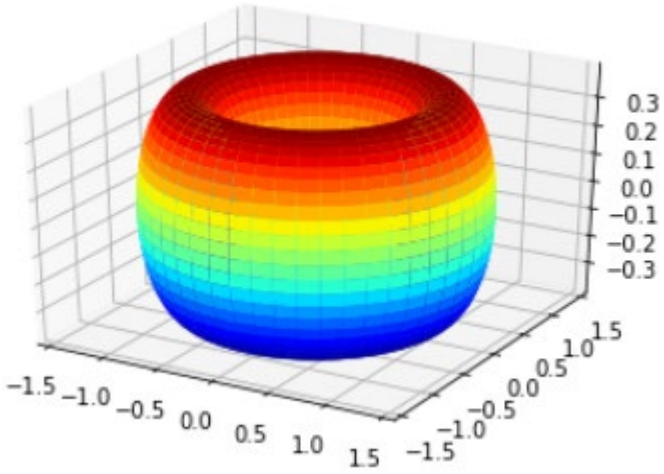
Параметрична просторова лінія- Спіраль.

```
a=0.1
plot3d_parametric_line(cos(t),sin(t),a*t,(t,0,4*pi))
```



Параметрична поверхня – тор.

```
u,v=symbols('u v')
a=0.4
plot3d_parametric_surface((1+a*cos(u))*cos(v),
                          (1+a*cos(u))*sin(v),a*sin(u),
                          (u, 0,2 * pi), (v, 0,2 * pi))
```



(3ahttps://www.inp.nsk.su/~grozin/python/b25_sympy.html)