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

SymPy представляет собой открытую библиотеку символьных вычислений на языке Python. SymPy полностью написан на языке Python и не требует сторонних библиотек.

Ниже, в познавательных целях (по материалам сайта http://www.asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html) , описаны *самые основные* возможностях пакета SymPy.

Более подробно смотрите http://www.sympy.org/en/index.html и https://www.sympy.org/ru/index.html

Математическая библиотека Python SymPy

SymPy — это библиотека Python для выполнения символьных вычислений. Это система компьютерной алгебры, которая может выступать как отдельное приложение, так и в качестве библиотеки для других приложений. Поработать с ней онлайн можно на https://live.sympy.org/. Поскольку это чистая библиотека Python, ее можно использовать даже в интерактивном режиме.

В SymPy есть разные функции, которые применяются в сфере символьных вычислений, математического анализа, алгебры, дискретной математики, квантовой физики и так далее. SymPy может представлять результат в разных форматах: LaTeX, MathML и так далее. Распространяется библиотека по лицензии New BSD. Первыми эту библиотеку выпустили разработчики Ondřej Čertík и Aaron Meurer в 2007 году.

Вот где применяется SymPy:

- Многочлены
- Математический анализ
- Дискретная математика
- Матрицы
- Геометрия
- Построение графиков
- Физика
- Статистика
- Комбинаторика

Установка SymPy

Для работы SymPy требуется одна важная библиотека под названием `mpmath`. Она используется для вещественной и комплексной арифметики с числами с плавающей точкой произвольной точности. Однако pip установит ее автоматически при загрузке самой SymPy:

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

Использование SymPy в качестве калькулятора

SymPy поддерживает три типа численных данных: **Float**, **Rational** и **Integer**.

Rational представляет собой обыкновенную дробь, которая задается с помощью двух целых чисел: числителя и знаменателя. Например, Rational(1, 2) представляет дробь 1/2, Rational(5, 2) представляет дробь 5/2, и так далее.

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

>>> a
1/2

>>> a*2
1

>>> Rational(2)**50/Rational(10)**50
1/888178419700125232338905334472656 25
```

Важная особенность Python-интерпретатора - при делении двух «питоновских» чисел типа int с помощью оператора “/” получается «питоновский» тип float. Этот же стандарт “true division” по умолчанию включен и в sympy:

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

Обратите внимание, что и в том и в другом случае вы имеете дело не с объектом Number из библиотеки SymPy, который представляет число в SymPy, а с питоновскими числами, которые создаются самим интерпретатором Python. Скорее всего, вам нужно будете работать с дробными числами из библиотеки SymPy, поэтому для того чтобы получать результат в виде объектов SymPy убедитесь, что вы используете класс Rational. Кому-то может показаться удобным обозначать Rational как **R**:

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

В модуле Sympy имеются особые константы, такие как **e** и **pi**, которые ведут себя как переменные (то есть выражение `1 + pi` не преобразуется сразу в число, а так и останется `1 + pi`):

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

```
>>> pi.evalf()
3.14159265358979

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

как видно, функция **evalf** переводит исходное выражение в число с плавающей точкой. Вычисления можно проводить с большей точностью. Для этого нужно передать в качестве аргумента этой функции требуемое число десятичных знаков:

```
>>> pi.evalf(100)
3.14159265358979323846264338327950288419716939937510582
09749445923078164062862089986280348253421170 68
>>>
```

Для работы с математической бесконечностью используется символ **oo**:

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

Переменные

В отличие от многих других систем компьютерной алгебры, нужно явно декларировать символьные переменные:

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

В левой части этого выражения находится переменная Python, которая питоновским присваиванием соотносится с объектом класса Symbol из SymPy.

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

Символьные переменные могут также задаваться и с помощью функций `symbols` или `var`. Они допускают указание диапазона. Их отличие состоит в том, что `var` добавляет созданные переменные в текущее пространство имен:

```
>>> from sympy 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)
```

Экземпляры класса `Symbol` взаимодействуют друг с другом. Таким образом, с помощью них конструируются алгебраические выражения:

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

>>> (x + y)**2
(x + y)**2
#раскрыть скобки
>>> ((x + y)**2).expand()
x**2 + 2*x*y + y**2
```

Переменные могут быть заменены на другие переменные, числа или выражения с помощью **функции подстановки `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
```

Теперь, с этого момента, для всех написанных ниже примеров мы будем предполагать, что запустили следующую команду по настройке системы отображения результатов (и используем процедуру pprint):

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

Она придаст более качественное отображение выражений. Подробнее по системе отображения и печати написано ниже в разделе Печать. Если же установлен шрифт с юникодом, то можно использовать опцию `use_unicode=True` для еще более красивого вывода.

Алгебра

Чтобы **разложить выражение на простейшие дроби** используется функция **`apart(expr, x)`:**

```
#apart(expr, x):
from sympy import apart
from sympy.abc import x, y, z
from sympy 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)
```

Вывод скрипта:

```
       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
```

Чтобы **привести дробь к общему знаменателю** используется функция **together(expr, x):**

```
>>> from 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)
```

Вычисления

Пределы

Для вычисления предела функции, используйте функцию **limit(function, variable, point).**

Например, чтобы вычислить предел f(x) при x -> 0, нужно ввести limit(f, x, 0):

```
>>> from sympy import limit, Symbol, sin, oo
>>> x = Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
```

также можно вычислять пределы при x, стремящемся к бесконечности:

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

>>> limit(1/x, x, oo)
0

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

Более сложные примеры вычисления пределов см. например - test_demidovich.py

Дифференцирование

Продифференцировать любое выражение SymPy, можно используя **diff(func, var).** Примеры:

```
>>> from sympy 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
```

Можно, через пределы проверить правильность вычислений производной:

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

Производные более высших порядков можно вычислить, используя дополнительный параметр этой же функции **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)
```

Разложение в ряд

Для разложения в ряд используйте метод **series(var, point, order):**

```
>>> from sympy import Symbol, cos
>>> x = Symbol('x')
>>> cos(x).series(x, 0, 10)

     2    4     6      8
    x    x     x      x       / 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
```

Еще один простой пример:

```
>>> from sympy import 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   x    x    x    x     / 5\
- - -- + -- - -- + -- + O\x /
y    2    3    4    5
    y    y    y    y
```

Суммы

Вычисляет значение суммы f(от заданной переменной) на заданных пределах

summation(f, (i, a, b)) – нахождение суммы слагаемых f, i изменяется от a до b:

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

If it cannot compute the sum, it prints the corresponding summation formula. Repeated sums can be computed by introducing additional limits:

Если сумма не «рассчитывается», то печатается соответствующая формула:

```
>>> from sympy import summation, oo, symbols, log
>>> i, n, m = symbols('i n 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
m    m    m
-- + -- + -
6    2    3

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

Интегрирование

SymPy поддерживает вычисление определенных и неопределенных интегралов с помощью функции `integrate()`. Она использует расширенный алгоритм Риша-Нормана и некоторые шаблоны и эвристики. Можно вычислять интегралы трансцендентных, простых и специальных функций:

```
>>> from sympy import integrate, erf, exp, sin, log,
                                        oo, pi, sinh, symbols
>>> x, y = symbols('x,y')
```

Вы можете интегрировать простейшие функции:

```
>>> 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)
```

Примеры интегрирования некоторых специальных функций:

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

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

Возможно также вычислить определенный интеграл:

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

Поддерживаются и несобственные интегралы:

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

Комплексные числа

Помимо мнимой единицы I, которое является мнимым числом, символы тоже могут иметь специальные атрибуты (real, positive, complex и т.д), которые определяют поведение этих символов при вычислении символьных выражений:

```
>>> from sympy import 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)
```

Функции

тригонометрические

```
>>> from 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
    x     x     x        x        / 10\
x - -- + --- - ---- + ------ + O\x  /
    6    120   5040   362880

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

     3     5     7        9
```

```
    x     x     x       x       / 10\
x + -- + --- + ---- + ------ + O\x  /
    6    120   5040   362880
```

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

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

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

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

сферические

```
>>> from sympy import Ylm
>>> from 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
```

факториалы и гамма-функции

```
>>> from 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) # i.e. factorial(x)

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

дзета-функции

```
>>> from 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
```

```
>>> from sympy 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
```

Дифференциальные уравнения

```
В isympy:
>>> from sympy import Function, Symbol, dsolve
>>> f = Function('f')
>>> x = Symbol('x')
>>> f(x).diff(x, x) + f(x)
f(x) + Derivative(f(x), x, x)
>>> from sympy import 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)
```

Алгебраические уравнения

```
>>> from 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}
```

Линейная алгебра

Матрицы

Матрицы задаются с помощью конструктора **Matrix**:

```
>>> from sympy import Matrix, Symbol
>>> Matrix([[1, 0], [0, 1]])
[1  0]
[    ]
[0  1]
```

В матрицах вы также можете использовать символьные переменные:

```
>>> 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*y    x*y + 1]
```

Для того, чтобы узнать о матрицах подробнее, прочитайте, пожалуйста, Руководство по Линейной Алгебре.

Сопоставление с образцом

Чтобы сопоставить выражения с образцами, используйте функцию **.match()** вместе со вспомогательным классом **Wild**. Эта функция вернет словарь с необходимыми заменами, например:

```
>>> from sympy import 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}
```

Если же сопоставление не удалось, функция вернет``**None**``:

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

Также можно использовать параметр **exclude** для исключения некоторых значений из результата:

```
>>> 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}
```

Печать

Реализовано несколько способов вывода выражений на экран.

Стандартный

Стандартный способ представлен функцией `str(expression)`, которая работает следующим образом:

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

«Красивая печать»

Этот способ печати выражений основан на ascii-графике и реализован через функцию **pprint**:

```
>>> from sympy import Integral, pprint
>>> from sympy.abc import x
>>> pprint(x**2)

 2
X

>>> pprint(1/x)

1
-
X

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

Если у вас установлен шрифт с юникодом, он будет использовать `Pretty-print` с юникодом по умолчанию. Эту настройку можно отключить, используя `use_unicode`:

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

Для изучения подробных примеров работы Pretty-print с юникодом вы можете обратится к статье Pretty Printing (https://github.com/sympy/sympy/wiki/Pretty-Printing) на Вики.

Для того, чтобы сделать **pprint** по умолчанию в стандартном интерпретаторе, производим следующую процедуру:

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

Примеры:

```
>>> from 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
 |
/
```

Печать объектов Python

```
>>> from sympy.printing.python import python
>>> from sympy import Integral
>>> from 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)
```

Печать в формате LaTeX

```
>>> from sympy import Integral, latex
>>> from 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
```

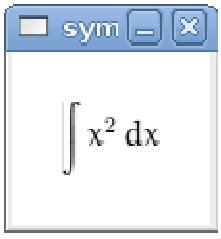
MathML

```
>>> from sympy.printing.mathml import mathml
>>> from sympy import Integral, latex
>>> from 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

```
>>> from sympy import Integral, preview
>>> from sympy.abc import x
>>> preview(Integral(x**2, x))
```

Появится окно pyglet с отрисованным выражением LaTeX:



Примечания

Если установлены дополнительные пакеты.

isympy вызывает **pprint** автоматически, по этой причине `Pretty-print` будет включен в **isympy** по умолчанию.

Также доступен модуль печати - **sympy.printing**. Через этот модуль доступны следующий функции печати:

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

Возвращает или выводит на экран, соответственно, “Красивое” представление выражения expr.

`latex(expr), print_latex(expr) –`

Возвращает или выводит на экран, соответственно, LaTeX - представление `expr`

`mathml(expr), print_mathml(expr) -`

Возвращает или выводит на экран, соответственно, MathML (http://www.w3.org/Math/) -представление `expr`.

Чтобы узнать о SymPy подробнее, обратитесь:

`Руководство пользователя SymPy`
`http://www.asmeurer.com/sympy_doc/dev-py3k/guide.html`
и `Описание модулей SymPy`.
`http://www.asmeurer.com/sympy_doc/dev-py3k/modules/index.html`

Также можно обратится на wiki.sympy.org - сайт, который содержит множество полезных примеров, руководств и советов.

Примеры применения пакета SymPy

SymPy - это пакет для символьных вычислений на питоне, подобный системе `Mathematica`. Он работает с выражениями, содержащими символы.

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

Основными кирпичиками, из которых строятся выражения, являются символы. Символ имеет имя, которое используется при печати выражений. Объекты класса Symbol нужно создавать и присваивать переменным питона, чтобы их можно было использовать.

В принципе, имя символа и имя переменной, которой мы присваиваем этот символ - две независимые вещи, и можно написать abc=Symbol('xyz').

Но тогда при вводе программы Вы будете использовать abc, а при печати результатов SymPy будет использовать xyz, что приведёт к ненужной путанице. Поэтому лучше, чтобы имя символа совпадало с именем переменной питона, которой он присваивается.

В языках, специально предназначенных для символьных вычислений, таких, как `Mathematica`, если Вы используете переменную, которой ничего не было присвоено, то она автоматически воспринимается как символ с тем же именем. Питон не был изначально предназначен для символьных вычислений. Если Вы используете переменную, которой ничего не было присвоено, Вы получите сообщение об ошибке. Объекты типа Symbol нужно создавать явно.

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

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

$$x^2 - 1$$

```
type(a)
```

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

Можно определить несколько символов одновременно. Строка разбивается на имена по пробелам.

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

Подставим вместо x выражение y+1

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

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

Многочлены и рациональные функции

SymPy не раскрывает скобки автоматически. Для этого используется функция **expand**.

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

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

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

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

Степень многочлена a по x **`degree`**

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

6

Соберём вместе члены с определёнными степенями x **`collect`**

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

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

Многочлен с целыми коэффициентами можно записать в виде произведения таких многочленов (причём каждый сомножитель уже невозможно расфакторизовать дальше, оставаясь в рамках многочленов с целыми коэффициентами).

Существуют эффективные алгоритмы для решения этой задачи - **`factor`**.

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

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

SymPy не сокращает отношения многочленов на их наибольший общий делитель автоматически. Для этого используется функция **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 приводит суммы рациональных выражений к общему знаменателю автоматически. Для этого используется функция **together**.

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

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

```
together(a)
```

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

Функция **simplify** пытается переписать выражение ***в наиболее простом виде***. Это понятие не имеет чёткого определения (в разных ситуациях ***наиболее простыми*** могут считаться разные формы выражения), и не существует алгоритма такого упрощения.

Функция **symplify** работает эвристически, и невозможно заранее предугадать, какие упрощения она попытается сделать. Поэтому её удобно использовать в интерактивных сессиях, чтобы посмотреть, удастся ли ей записать выражение в каком-нибудь разумном виде, но нежелательно использовать в программах. В них лучше применять более специализированные функции, которые выполняют одно определённое преобразование выражения.

```
simplify(a)
```

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

Разложение на элементарные дроби по отношению к x и y - **apart**

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

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

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

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

Подставим конкретные численные значения вместо переменных x и y **subs**

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

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

А сколько это будет численно? Метод **n()**

```
a.n()
```

$$-1.66666666666667$$

Элементарные функции

SymPy автоматически применяет упрощения элементарных функция (*которые справедливы во всех случаях*).

```
sin(-x)
```

$$-\sin(x)$$

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

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

SymPy может работать с числами с плавающей точкой, имеющими сколь угодно большую точность. Вот π с 100 значащими цифрами – метод `n(100)`.

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

$$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534$$

E - это основание натуральных логарифмов.

```
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)
```

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

Символы могут иметь некоторые свойства. Например, они могут быть положительными. Тогда SymPy может сильнее упростить квадратные корни.

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

$$p$$

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

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

Пусть символ *n* будет целым (I - это мнимая единица).

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

$$1$$

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

$$0$$

Метод **`rewrite`** пытается переписать выражение в терминах заданной функции.

```
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)
```

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

Функция **`trigsimp`** пытается переписать тригонометрическое выражение в наиболее простом виде. В программах лучше использовать более специализированные функции.

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

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

Функция **`expand_trig`** разлагает синусы и косинусы сумм и кратных углов.

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

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

Чаще нужно обратное преобразование - произведений и степеней синусов и косинусов в выражения, линейные по этим функциям.

Например, пусть мы работаем с отрезком ряда Фурье.

```
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
```

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

Мы хотим возвести его в квадрат и опять получить отрезок ряда Фурье.

```
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)])
```

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

Функция **expand_log** преобразует логарифмы произведений и степеней в суммы логарифмов (только для положительных величин);

logcombine производит обратное преобразование.

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

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

```
logcombine(a)
```

$$\log(pq^2)$$

Функция **expand_power_exp** переписывает степени, показатели которых - суммы, через произведения степеней.

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

$$x^px^q$$

Функция **expand_power_base** переписывает степени, основания которых - произведения, через произведения степеней.

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

$$x^ny^n$$

Функция **powsimp** выполняет обратные преобразования.

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

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

Можно вводить функции пользователя. Они могут иметь произвольное число аргументов.

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

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

Структура выражений

Внутреннее представление выражения - это дерево. Функция **srepr** возвращает строку, представляющую его.

```
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)))"
```

Вместо бинарных операций +, *, ** и т.д. можно использовать функции **Add**, **Mul**, **Pow** и т.д.

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

```
True
```

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

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

Атрибут **func** - это функция верхнего уровня в выражении, а **args** - список её аргументов.

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

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

```
a.args
```

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

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

$$2$$

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

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

Функция **subs** заменяет переменную на выражение.

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

$8x$

Она может заменить несколько переменных. Для этого ей передаётся список кортежей или словарь.

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

8π

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

8π

Она может заменить не переменную, а подвыражение - функцию с 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

Подстановки производятся последовательно. В данном случае сначала x заменился на y, получилось y^3+y^2; потом в этом результате y заменилось на x

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

x^3+x^2

Если написать эти подстановки в другом порядке, результат будет другим.

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

y^3+y^2

Но можно передать функции subs ключевой параметр **`simultaneous=True`**, тогда подстановки будут производиться одновременно. Таким образом можно, например, поменять местами x и y

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

x^3+y^2

Можно заменить функцию на другую функцию.

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

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

Метод **replace** ищет подвыражения, соответствующие образцу (содержащему произвольные переменные), и заменяет их на заданное выражение (оно может содержать те же произвольные переменные).

```
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)
```

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

Соответствовать образцу должно целое подвыражение, это не может быть часть сомножителей в произведении или меньшая степень в большей.

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

$$2xyz$$

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

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

Решение уравнений

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

Уравнение записывается как функция Eq с двумя параметрами. Функция **solve** возвращает список решений.

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

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

Впрочем, можно передать функции **solve** просто выражение. Подразумевается уравнение, что это выражение равно 0.

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

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

Квадратное уравнение имеет 2 решения.

```
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 линейных уравнений.

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

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

Функция **`roots`** возвращает корни многочлена с их кратностями.

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

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

Функция **`solve_poly_system`** решает систему полиномиальных уравнений, строя их базис Грёбнера.

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

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

Ряды

```
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 может начинаться с отрицательной степени.

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

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

И даже идти по полу целым степеням.

```
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)
```

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

Подготовим 3 ряда.

```
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
```

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

Произведения и частные рядов не вычисляются автоматически, к ним надо применить функцию **`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 этот ряд должен быть равен 1. Но поскольку **sinx** и **cosx** известны лишь с ограниченной точностью, мы получаем 1 с той же точностью.

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

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

Здесь первые члены сократились, и ответ можно получить лишь с меньшей точностью.

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

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

Ряды можно дифференцировать и интегрировать.

```
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)
```

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

Можно подставить ряд (если он начинается с малого члена) вместо переменной разложения в другой ряд. Вот ряды для **sin(tan(x))** и **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)
```

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

В ряд нельзя подставлять численное значение переменной разложения (а значит, нельзя и строить график). Для этого нужно сначала убрать $\mathcal{O}$ член, превратив отрезок ряда в многочлен.

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

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

0.0998334166468254

Производные

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

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

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

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

Вторая производная по x и первая по y

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

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

Можно дифференцировать выражения, содержащие неопределённые функции.

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

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

Что это за «зверь» такой получился?

```
print(b)
```

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

Функция **Derivative** представляет не вычисленную производную. Её можно вычислить методом doit.

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

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

Интегралы

```
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
```

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

Получился неопределённый интеграл.

```
print(a)
```

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

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

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

Определённые интегралы.

```
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))
```

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

Суммирование рядов

```
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))
```

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

Не вычисленная сумма обозначается Sum.

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

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

Пределы

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

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

Это простой предел, он считается разложением числителя и знаменателя в ряды. А если в x=0, то появляется особая точка.

Найдем односторонние пределы.

```
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,'-')
```

$$0$$

Дифференциальные уравнения

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

Первого порядка.

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

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

Второго порядка.

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

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

Система уравнений первого порядка.

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

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

Линейная алгебра

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

Матрицу можно построить из списка списков.

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

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

```
M.shape
```

$$(2, \ 3)$$

Матрица-строка.

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

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

Матрица-столбец.

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

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

Можно построить матрицу из функции.

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

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

Или из неопределённой функции.

```
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]
```

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

Единичная матрица.

```
eye(3)
```

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

Матрица из нулей.

```
zeros(3)
```

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

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

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

Диагональная матрица.

```
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)
```

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

Операции с матрицами.

```
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)
```

$$ad - bc$$

Собственные значения и векторы

```
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)
```

$$0$$

Значит, у этой матрицы есть нулевое подпространство (она обращает векторы из этого подпространства в 0). Базис этого подпространства.

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

$$1$$

Оно одномерно.

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

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

Проверим.

Проверим.

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

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

Собственные значения и их кратности.

```
M.eigenvals()
```

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

Если нужны не только собственные значения, но и собственные векторы, то нужно использовать метод **`eigenvects`**. Он возвращает список кортежей. В каждом из них нулевой элемент - собственное значение, первый - его кратность, и последний - список собственных векторов, образующих базис (их столько, какова кратность).

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

3

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

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

Проверим.

```
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 нормальная 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() возвращает пару матриц, матрицу преобразования P и собственно 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
```

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

Проверим.

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

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

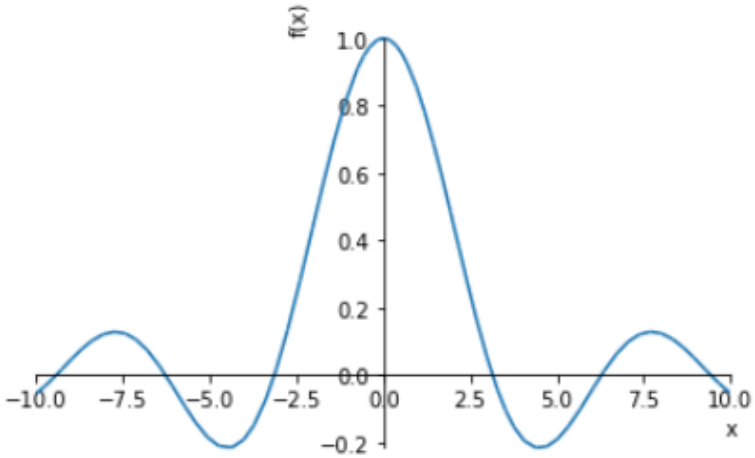
Графики

SymPy использует matplotlib. Однако он распределяет точки по x адаптивно, а не равномерно.

%matplotlib inline

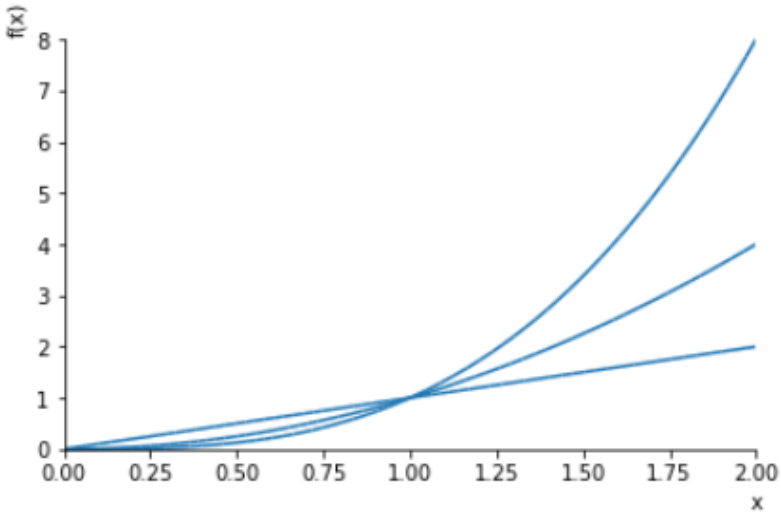
Одна функция.

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



Несколько функций.

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

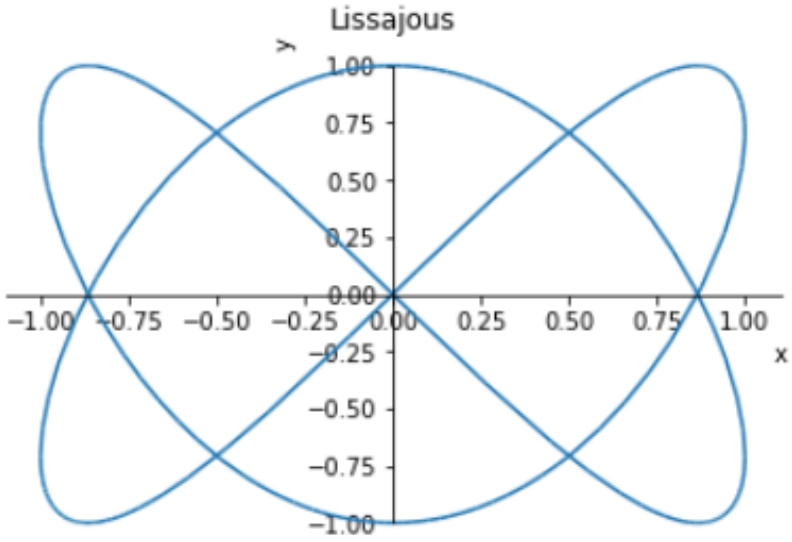


Другие функции надо **импортировать** из пакета **sympy.plotting.**

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

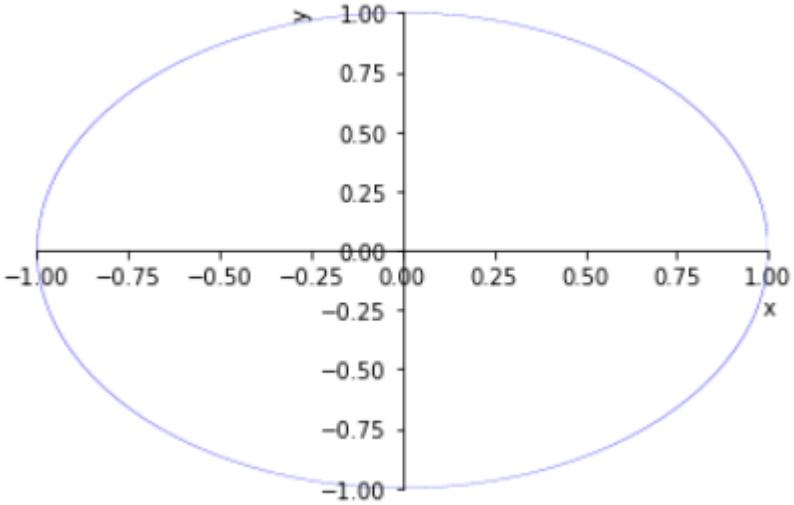
Параметрический график - фигура Лиссажу.

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



Неявный график - окружность.

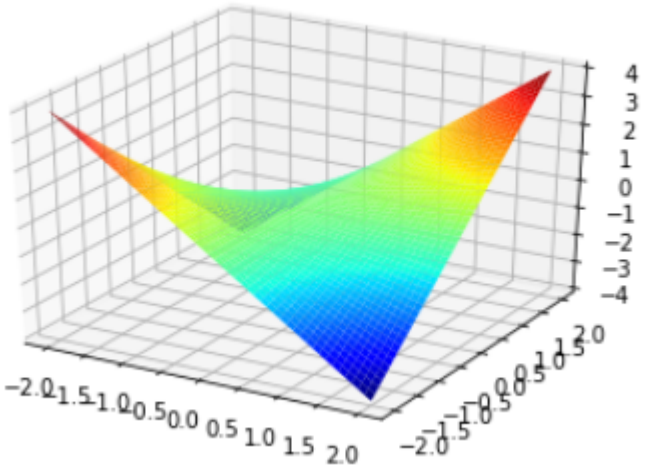
```
plot_implicit(x**2+y**2-1,(x,-1,1),(y,-1,1))
```



Поверхность.

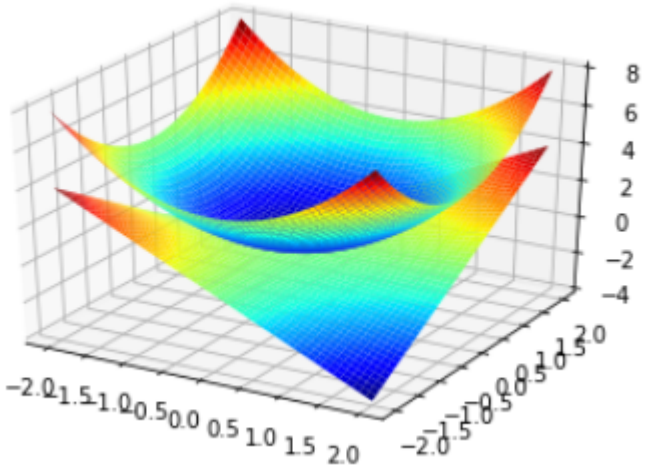
Если она строится не inline, а в отдельном окне, то её можно вертеть мышкой.

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



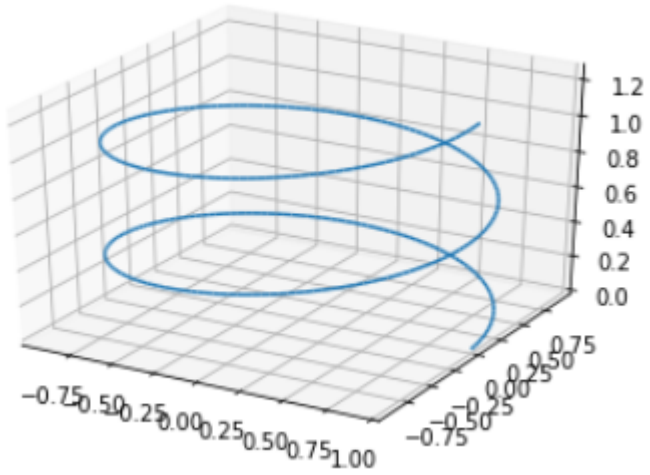
Несколько поверхностей.

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



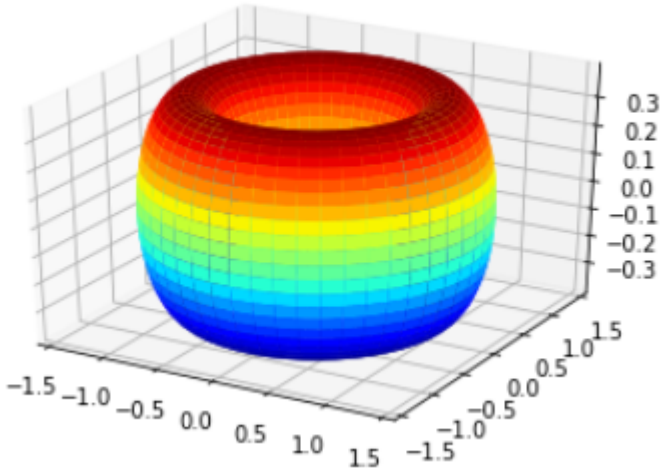
Параметрическая пространственная линия - спираль.

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



Параметрическая поверхность - 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))
```



(По https://www.inp.nsk.su/~grozin/python/b25_sympy.html)