Символьные выражения. Решение уравнений и неравенств

Eugene Strakhov

Please report bugs to strakhov.e.m@onu.edu.ua

1 Символьные переменные и выражения

Помимо стандартных объектов языка Python, SageMath может оперировать с объектами специального вида — **символьными** константами, переменными, выражениями, функциями. Символьный объект «ведёт» себя так, как если бы вычисления с ним производились на бумаге. С примерами символьных констант мы уже знакомы: это, например, `pi`, `pi/3`, `2/5`, `sqrt(2)`, `sin(pi/4)`. Производя вычисления с символьными константами (т. е. конструируя из них выражение), мы получаем снова символьную константу. Получить числовое значение символьного выражения можно с помощью функций `float()`, `round()` либо `n()`.

Существует возможность задания символьной переменной, не присваивая ей конкретного значения (т. е. в общем виде). Такая переменная затем может входить в определение символьной функции, использоваться в качестве аргумента встроенных функций и т. д. Символьные переменные объявляются командой `var()`. Вызовем справку по этой команде.

```
var?
```

```
File: /projects/sage/sage-7.5/src/sage/calculus/var.pyx
Signature : var(*args, **kwds)
Docstring :
Create a symbolic variable with the name *s*.

INPUT:
```

```
* "args" -- A single string "var('x y')", a list of strings
  "var(['x','y'])", or multiple strings "var('x', 'y')". A single
  string can be either a single variable name, or a space or comma
  separated list of variable names. In a list or tuple of strings,
  each entry is one variable. If multiple arguments are specified,
  each argument is taken to be one variable. Spaces before or after
  variable names are ignored.

* "kwds" -- keyword arguments can be given to specify domain and
  custom latex_name for variables. See EXAMPLES for usage.

Note: The new variable is both returned and automatically
  injected into the global namespace. If you need a symbolic
  variable in library code, you must use either "SR.var()" or
  "SR.symbol()".

OUTPUT:

If a single symbolic variable was created, the variable itself.
Otherwise, a tuple of symbolic variables. The variable names are
checked to be valid Python identifiers and a "ValueError" is raised
otherwise.

...
```

Примеры: создание символьных переменных с помощью `var()`

```
x = var('x') # символьная переменная x
f = x * sin(x) # символьная функция от x
f(x=pi/6) # значение функции в точке pi/6
x, y, z = var('x y z') # множественное присваивание
x, y = var('x y', domain=RR) # ключевое слово domain определяет тип
                             # переменной (RR, ZZ, QQ и др.);
                             # по умолчанию - CC (комплексное поле)
expr = (x+y)^2 # символьное выражение в общем виде
type(expr)
eqn = x^2 == y + 5 # выражение с == является уравнением
show(eqn)
ineq = x + 5 <= y # неравенство
show(ineq)
a1 = var('a1', latex_name='a_1') # удобно при выводе на экран
```

```
show(a1)
eps = var('eps', latex_name=r'\varepsilon')
show(eps + eps^2/2)
```

```
1/12*pi
<type 'sage.symbolic.expression.Expression'>
```

$$x^2 = y + 5$$
$$x + 5 \leq y$$
$$a_1$$
$$\frac{1}{2}\varepsilon^2 + \varepsilon$$

Обратите внимание!

```
x = var('a')
show(x)
```

$$a$$

2 Преобразования выражений

SageMath «умеет» выполнять некоторые преобразования над выражениями: складывать, вычитать, умножать, делить, раскрывать скобки, приводить подобные слагаемые, раскладывать на множители, упрощать выражение, делать подстановки и т. д. Чтобы увидеть полный список доступных методов, создайте любое выражение, наберите его имя, затем точку и нажмите клавишу Tab.

Как правило, суть того или иного метода сразу ясна из названия. Описание методов и примеры см. в документации:
http://doc.sagemath.org/html/en/reference/calculus/sage/symbolic/expression.html

Отметим, что все методы, преобразующие выражение, возвращают новое выражение. Т. е. мы можем использовать конструкции вида

```
expr.func_name() # вывод на экран;
                 # то же самое, что print expr.func_name()
show(expr.func_name()) # красивый вывод на экран
expr1 = expr.func_name() # результат преобразования присваивается
                         # другому выражению
```

Чтобы не набирать названия длинных методов полностью, можно также пользоваться Tab-дополнениями команд (набрать несколько первых символов + Tab).

2.1 Замечание относительно неравенств

Применяя функцию умножения (деления) обеих частей неравенства на некоторое число, можно было бы ожидать, что в случае отрицательного множителя знак неравенства будет меняться. Однако...

```
x = var('x')
ineq = x^2 >= 7
ineq*(-1) # :(
ineq.multiply_both_sides(-1) # :(
```

```
-x^2 >= -7
-x^2 >= -7
```

3 Аналитическое решение уравнений и неравенств

Для решения уравнений и неравенств **в символьном виде** служит функция `solve()`. Вызовем справку по ней.

Обратите внимание на опции `explicit_solutions` и `to_poly_solve`. Довольно часто они «выручают». Однако эти опции не применимы при решении неравенств.

Функция `solve()` выдаёт ответ в виде **списка выражений** (для систем — список списков). На структуру ответа следует обращать внимание в тех случаях, когда требуется дальнейшее исследование полученных решений (например, проверить, есть ли среди решений целые корни, все ли решения действительны и т. п.). «Добраться» до левой и правой частей выражений помогут методы `left()` и `right()`.

```
solve?
```

```
...
```

`solve()`: пример 1

```
x = var('x')
eq = x^3 -7*x + 6 == 0
sol = eq.solve(x)
print sol
show(sol)
latex(sol)
sol[0]
```

```
# Обратите внимание на использование left() и right():
sol[0].left()
sol[0].right()
# Проверим, все ли решения являются целыми:
for xx in sol :
    if xx.right() not in ZZ :
        print('Неверно')
        break
else :
    print('Верно')
```

```
[
x == 1,
x == 2,
x == -3
]
```

$$[x = 1, x = 2, x = (-3)]$$

```
\left[x = 1, x = 2, x = \left(-3\right)\right]
x == 1
x
1
Верно
```

solve(): пример 2

```
x = var('x', domain=ZZ)
eq = x^3 -7*x + 6 == 0
sol = eq.solve(x)
# Верно ли, что у уравнения есть целые корни?
if sol == [] :
    print('No')
else :
    print('Yes')
```

```
Yes
```

solve(): пример 3

```
x, y = var('x y')
eq1 = x^2 + y == 6
eq2 = 7*x - 22*y == 15
sols = solve([eq1, eq2], x, y, solution_dict=True) # решение в виде
                    # списка словарей
show(sols)
# Выведем только целые решения
flag = False
for s in sols :
    if s[x] in ZZ and s[y] in ZZ : # ключи - не строки, а сами переменные!
        print(s)
        flag = True
if not flag :
    print 'Целых решений нет'
```

$$\left[\left\{y : -\frac{49}{968}\sqrt{53}\sqrt{5} - \frac{709}{968}, x : -\frac{7}{44}\sqrt{265} - \frac{7}{44}\right\}, \left\{y : \frac{49}{968}\sqrt{53}\sqrt{5} - \frac{709}{968}, x : \frac{7}{44}\sqrt{265} - \frac{7}{44}\right\}\right]$$

```
Целых решений нет
```

3.1 Опция to_poly_solve

Пример: опция to_poly_solve

```
x = var('x')
eqn = sin(x) + cos(x) == 1
eqn.solve(x) # ответ в неявном виде...
eqn.solve(x, explicit_solutions=True) # не помогло...
eqn.solve(x, to_poly_solve=True) # есть!
```

```
[sin(x) == -cos(x) + 1]
[]
[x == 2*pi*z38, x == 1/2*pi + 2*pi*z40]
```

3.2 Фильтрование решений

Пример: фильтрование решений

```
x = var('x')
eqn = (x-1)*(x-sqrt(2))^2*(x-2/3) == 0
eqn.solve(x)
# Профильтруем список решений и оставим только целые значения
sol = eqn.solve(x) # исходный список
z_sol = filter(lambda x: x.right().is_integer(), sol) # с помощью filter()
print z_sol
z_sol = [x for x in sol if x.right() in ZZ] # с помощью конструктора
print z_sol
# Оставим в sol только значения (без x==)
print [x.right() for x in sol]
```

```
[x == (2/3), x == sqrt(2), x == 1]
[x == 1]
[x == 1]
[2/3, sqrt(2), 1]
```

3.3 Некоторые ограничения функции `solve()`

Функция `solve()` лучше всего справляется с алгебраическими уравнениями и неравенствами. Что касается других видов, могут возникать непредвиденные проблемы. Например, попытка решить даже такое простое иррациональное неравенство, как $\sqrt{x-5} > 2$, приводит к разочарованию:

```
x = var('x', domain='real')
solve(sqrt(x-5)>2, x) # получен ответ в неявной форме...
```

```
[[sqrt(x - 5) - 2 > 0]]
```

4 Предположения относительно переменных

Мы уже видели, что при создании символьной переменной существует возможность указать для неё область допустимых значений. Однако этого не всегда достаточно. Например, мы хотим решить уравнение в предположении, что `x` не только целое число, но и большее 5. Сделать это можно, **введя предположение** (assumption) для переменной `x`. Предположения вводятся с помощью функции `assume()`. Для от-

мены предположений существует функция `forget()`. Просмотреть список активных предположений можно с помощью `assumptions()`. Примеры:

```
assume(a>0)
assume(y, 'integer')
assume(a>=1, a, 'real', x, 'even') # несколько предположений одновременно
assumptions()
forget(a>0)
forget(a>=1, x, 'even')
forget() # для отмены всех предположений
```

Список доступных опций `feature` для предположений вида `assume(x, 'feature')`:

```
[integer, noninteger, even, odd, rational,       irrational, real, imaginary, complex,
 analytic, increasing, decreasing, oddfun, evenfun, posfun, constant, commutative,
 lassociative, rassociative, symmetric, antisymmetric, integervalued, one_to_one]
```

См. также
http://doc.sagemath.org/html/en/reference/calculus/sage/symbolic/assumptions.html

Предположения: пример 1

```
# Рассмотрим решение уравнения x^2 == -1
x = var('x') # по умолчанию x - комплексное число
x.is_real()
eqn = x^2 == -1
eqn.solve(x) # получаем два комплексных решения
assume(x, 'real') # ввели предположение
eqn.solve(x) # теперь решений нет
forget(x, 'real') # отменили предположение
assumptions() # список текущих предположений
```

```
False
[x == -I, x == I]
[]
[]
```

Предположения: пример 2

```
a = var('a', domain='real')
assume(a<0)
bool(abs(a) == a)
forget(a<0)
```

```
assume(a>=0)
bool(abs(a) == a)
bool(abs(a) == -a)
forget()
```

Больше примеров использования команд `assume()` и `forget()` можно увидеть, вызвав справку:

```
assume?
```

```
...
```

```
forget?
```

```
...
```