



Screenshots



СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ

Maxima: укротитель выражений

ЧАСТЬ 3 Сегодня Тихон Тарнавский расскажет вам о возможностях Maxima по упрощению выражений, содержащих рациональные, иррациональные и тригонометрические функции.

**МЕСЯЦ
НАЗАД**

Мы изучали функции
и операторы Maxima



НА ДИСКЕ

- Дистрибутив Maxima и графические оболочки



В этот раз я расскажу о штатных возможностях Maxima по упрощению и прочим преобразованиям выражений. В частности, речь пойдет об автоматическом раскрытии скобок и вынесении за скобки; об упрощении как арифметических действий над некоторыми элементами, так и выражений с участием степенных, показательных и логарифмических функций; а также об обработке тригонометрических выражений. Все эти функции призваны облегчать читаемость математических формул и повышать простоту их восприятия, а посему стоит уделить этому уроку достаточно внимания: при верном использовании данные манипуляции позволят сэкономить в процессе работы значительное количество времени.

Выражаясь рационально...

Существенная часть интересующих нас сегодня функций предназначена для преобразования рациональных выражений. Напомним, рациональным называется выражение, состоящее только из арифметических операторов и возведения в натуральную степень; естественно, элементы такого выражения могут содержать и неарифметические и нестепенные функции — тогда такие элементы с точки зрения рационального выражения считаются атомарными, т.е. неделимыми и непреобразуемыми.

Функции, работающие с рациональными выражениями, описаны в разделе документации «Polynomials»; потому как рациональные функции с математической точки зрения рассматриваются как расширение многочленов (полиномов) — примерно так же, как рациональные числа считаются расширением целых (многочлены, кстати, тоже иногда называют целыми функциями; хотя общий математический смысл этого термина несколько шире).

Имена всех функций Maxima по обработке рациональных выражений содержат буквосочетание `rat`, но не от слова «крыса», а от слова «rational». И начнем мы знакомство с ними с функции, которая так и называется: `rat(выражение)`. Эта функция преобразовывает рациональное выражение к так называемой канонической форме (Canonical Rational Expression, CRE). То есть раскрывает все скобки, затем приводит все к общему знаменателю, суммирует и сокращает; кроме того, приводит все числа в конечной десятичной записи к рациональным.

$$(\%i1) \text{ rat}\left(\frac{(x-1)^2}{x^2+x} + \frac{1}{x+1} + 0.25\right)$$

‘rat’ replaced 0.25 by 1/4 = 0.25

$$(\%o1) \frac{5x^2 - 3x + 4}{4x^2 + 4x}$$

Тут надо заметить, что атомарные элементы, т.е. символы и числа, в канонической форме рационального выражения в Maxima имеют другое внутреннее представление. При работе в интерфейсах Maxima и xMaxima об этом напоминает приписка `/R/` после имени ячейки вывода (в `ixMaxima` и `TeXMaxcs` такая приписка отсутствует). При этом внешне, на видимом пользователю уровне, каноническая форма ничем, кроме

этого обозначения, от общей не отличается. Но один достаточно интересный момент здесь есть: если каноническая форма рационального выражения используется в других рациональных выражениях, то последние также автоматически приводятся к канонической форме:

$$(\%i2) \% - \frac{1}{x}$$

$$(\%o2) \frac{5x - 7}{4x + 4}$$

Это может быть достаточно удобно, если вам нужно пошагово проделать большое количество рациональных преобразований: вы можете, один раз вызвав `rat()`, сослаться на предыдущие ячейки и благодаря этому далее автоматически видеть на каждом шаге итоговое выражение в канонической, а значит, достаточно компактной и удобной к восприятию, форме. Если на каком-то этапе такое поведение станет вам мешать, вы можете вернуть выражение из канонической к общей форме с помощью функции `ratdisrep(выражение)`. Кроме того, каноническая форма автоматически «отменяется» и в случае любых преобразований, не являющихся рациональными:

$$(\%i3) \exp(\%)$$

$$(\%o3) e^{\frac{5x - 7}{4x + 4}}$$

Здесь, хотя `%o2` было выражением в канонической форме, `%o3` — уже выражение общего вида, так как оно не является рациональным.

Скажем пару слов о приведении конечной десятичной записи чисел к рациональной. Конечная десятичная запись считается по определению приближительной, что и понятно, т.к. при вычислениях самой Maxima такая запись может возникнуть исключительно при применении приближенных методов либо при ручном указании о переводе числа в десятичную запись из математической, в результате чего результат тоже, вероятнее всего, окажется приближительным. Эта приближительность учитывается и при переводе в рациональные числа, а ее уровень, то есть мера, на которую рациональное число при переводе может отклониться от конечной десятичной записи, регулируется переменной `ratepsilon`, равной по умолчанию `2.0e-8`, т.е. `0.00000002`. Если такое положение вещей вас не устраивает, вы можете убедить Maxima оставлять десятичную запись чисел как есть, установив в `true` значение флага `keepfloat` (по умолчанию он равен `false`).

Следующая функция раскрывает скобки в рациональном выражении и называется `ratexpand()` (одно из значений слова `expand` и есть «раскрыть скобки»). Здесь также действует опция `keepfloat`. Кроме нее, есть еще одна опция — `ratdenomdivide`; по умолчанию она установле-

на в *true*, что приводит к тому, что каждая дробь, в которой числитель является суммой, распадается на сумму дробей с одинаковым знаменателем. Если же сбросить эту опцию в *false*, тогда все дроби с одинаковым знаменателем будут, напротив, объединены в одну дробь с числителем в виде суммы числителей изначальных дробей. То есть внешне результат будет в этом случае выглядеть почти так же, как и у функции *rat()*; к тому же единственная видимая пользователю разница проявляется только в рациональных выражениях от нескольких переменных (или различных иррациональных выражений). Заключается эта разница в том, что после *ratexpand()* и в числителе, и в знаменателе дроби все скобки будут раскрыты, в случае же *rat()* слагаемые, где присутствуют, скажем, две переменных, будут сгруппированы, и одна из них будет вынесена за скобки (в документации такая форма записи называется «рекурсивной» (recursive):

```
(%i1) ratexpand((x+1)^2*(e^x-1)/(x+2)(2x+1), ratdenomdivide: false)
(%o1) (x^2 e^x + 2 x e^x + e^x - x^2 - 2 x - 1)/(2 x^2 + 5 x + 2)
(%i2) rat(%)
(%o2) (x^2 + 2 x + 1) e^x - x^2 - 2 x - 1/(2 x^2 + 5 x + 2)
(%i3) ratexpand(%)
(%o3) (x^2 e^x)/(2 x^2 + 5 x + 2) + (2 x e^x)/(2 x^2 + 5 x + 2) + (e^x)/(2 x^2 + 5 x + 2) - (x^2)/(2 x^2 + 5 x + 2) - (2 x)/(2 x^2 + 5 x + 2) - 1/(2 x^2 + 5 x + 2)
```

Кроме того, разница, конечно, заключается и во внутреннем представлении: с точки зрения программы, после *ratexpand()* выражение будет по-прежнему общего вида. Соответственно и все результаты дальнейших рациональных действий с выражением не будут автоматически «канонизироваться». Я специально обращаю ваше внимание на схожесть между результатами этих двух различных функций, поскольку в документации эта схожесть никак не обозначена: в описании обеих функций и примерах к ним нет вообще никаких ссылок друг на друга.

Помимо флага *ratdenomdivide*, есть также функция, собирающая воедино дроби с одинаковыми знаменателями; зовут ее *combine()*:

```
(%i4) combine(%)
(%o4) (x^2 e^x + 2 x e^x + e^x - x^2 - 2 x - 1)/(2 x^2 + 5 x + 2)
```

В дополнение к функции *ratexpand()* есть также флаг *ratexpand*, который по умолчанию равен *false*, а будучи установлен в *true*, приводит к тому, что все рациональные выражения в канонической форме отображаются и преобразовываются к общему виду сразу же с раскрытыми скобками:

```
(%i5) ratexpand: true $ %o2
(%o6) (x^2 e^x)/(2 x^2 + 5 x + 2) + (2 x e^x)/(2 x^2 + 5 x + 2) + (e^x)/(2 x^2 + 5 x + 2) - (x^2)/(2 x^2 + 5 x + 2) - (2 x)/(2 x^2 + 5 x + 2) - 1/(2 x^2 + 5 x + 2)
```

Обратите внимание, что при применении этого флага выражение сохраняет каноническую форму.

Действует в этом случае и флаг *ratdenomdivide* (напомню, что в строке *%i1* этот флаг был установлен локально, используя сокращенную запись функции *ev()*):

```
(%i7) ratdenomdivide: false $ %o2
(%o8) (x^2 e^x + 2 x e^x + e^x - x^2 - 2 x - 1)/(2 x^2 + 5 x + 2)
```

Иными словами, флаг *ratexpand* по своему действию аналогичен

одноименной функции, но действует он на все без исключения канонические рациональные выражения и при этом оставляет их в канонической внутренней записи и изменяет только внешнее отображение этой записи, сохраняя при этом и дальнейшую автоматическую «канонизацию».

...и не только рационально.

Помимо *ratexpand()* есть также и функция «просто» *expand()*. Различий между ними несколько, наиболее принципиальные таковы. Во-первых, *ratexpand()* раскрывает только рациональное выражение «верхнего уровня», все же подвыражения, не являющиеся рациональными, обрабатываются как атомарные, то есть внутрь них она не залезает; *expand()* же раскрывает скобки на всех уровнях вложенности:

```
(%i1) ratexpand((a+b)^(2-x)(x+2)+x^2)
(%o1) (b+a)^x^2+(2-x)(x+2)
(%i2) expand(%);
(%o2) b^4+4 a b^3+6 a^2 b^2+4 a^3 b+a^4
```

Во-вторых, *ratexpand()* приводит дроби-слагаемые к общему знаменателю, а *expand()* этого не делает; в-третьих, на функцию *expand* не действует переключатель *ratdenomdivide*:

```
(%i1) ratdenomdivide: false $ expand((x-2 y)^4/(x^2-4 y^2)^2+1)
(%o2) (16 y^4)/(16 y^4-8 x^2 y^2+x^4) - (32 x y^3)/(16 y^4-8 x^2 y^2+x^4) + (24 x^2 y^2)/(16 y^4-8 x^2 y^2+x^4) - (8 x^3 y)/(16 y^4-8 x^2 y^2+x^4) + 1/(16 y^4-8 x^2 y^2+x^4)
(%i3) ratexpand((x-2 y)^4/(x^2-4 y^2)^2+1)
(%o3) (8 y^2+2 x^2)/(4 y^2+4 x y+x^2)
```

И в-четвертых, *expand()* не преобразовывает к рациональным числам конечную десятичную запись — опять-таки, вне зависимости от флага *keepfloat*.

Функция *expand()*, в отличие от своего рационального сородича, имеет несколько вариаций — в виде отдельных функций с похожими названиями, которые раскрывают скобки несколько по-разному. Первую мы уже рассмотрели. Вторая называется *expandwrt(выражение, x, y, ..., v)*, где *wrt* расшифровывается как «with respect to...», то есть «относительно...». Она раскрывает скобки не везде, а только относительно тех символов, которые заданы в списке аргументов после выражения. Другими словами, только там, где из скобок можно вынести хотя бы один из перечисленных символов:

>>

```
(%i1) (x+y)(y+z)+x(v+w)$
(%i2) combine(expand(%))
(%o2) yz+xz+y^2+xy+x(w+v)/(vz+vy+vx)
(%i3) combine(expandwrt(%o1,x))
Warning - you are redefining the Maxima function intersection
(%o3) y(z+y)+x(z+y)+x(w+v)/(v(z+y+x))
(%i4) combine(expandwrt(%o1,z))
(%o4) (y+x)z+y(y+x)+x(w+v)/(v(z+y+x))
```

<< (На предупреждение, возникающее при первом вызове функций `expandwrt*`(), можете не обращать внимания — на функционале, о котором идет речь, оно никоим образом не отражается.)

Если в выражении встречаются дроби, то по умолчанию эта функция раскрывает скобки только в их числителях, оставляя знаменатели в покое. Изменить это поведение можно переключателем `expandwrt_denom`, установив его в `true` (по умолчанию он равен `false`):

```
(%i5) combine(expandwrt(%o1,z)),expandwrt_denom:true
```

```
(%o5) 
$$\frac{(y+x)z + y(y+x) + x(w+v)}{vz + v(y+x)}$$

```

И, наконец, последняя функция из этого семейства — `expandwrt_factored`(выражение, x, y, ..., v) — раскрывает скобки лишь в тех слагаемых, где упомянутые символы встречаются не в одном, а в каждом из сомножителей:

```
(%i1) expandwrt((x+a)(b+y),x)
```

Warning - you are redefining the Maxima function intersection

```
(%o1) x(y+b)+a(y+b)
```

```
(%i2) expandwrt_factored((x+a)(b+y),x)
```

```
(%o2) (x+a)(y+b)
```

```
(%i3) expandwrt_factored((x+a)(b+y),x,b)
```

```
(%o3) xy+ay+bx+ab
```

```
(%i4) expandwrt((x+a)(b+y),x,b)
```

```
(%o4) xy+ay+bx+ab
```

Раскрытием возведения в целую степень можно управлять как в контексте функции `expand()`, так и отдельно. В первом случае применяются переменные `maxposex` и `maxnegex`, определяющие соответственно максимальные положительный и отрицательный показатель степени, которые будут раскрываться этой функцией. По умолчанию оба параметра равны `1000`. Переназначить их можно не только глобально, но и в контексте одного конкретного вызова функции `expand()` — в таком случае это делается с помощью дополнительных аргументов, задаваемых после выражения:

```
(%i1) combine(expand((a+b)^2+(a+b)^4/(a-b)^6,3,3))
```

```
(%o1) 
$$\frac{(b+a)^4}{-b^3+3ab^2-3a^2b+a^3} + \frac{b^2+2ab+a^2}{(a-b)^6}$$

```

В противовес `maxposex` и `maxnegex` можно задать максимальные положительную и отрицательную степени, которые будут раскрываться автоматически, без вызова функций группы `expand`. За это отвечают переменные `exprop` и `exrpn`, и по умолчанию они равны нулю, то есть автоматически степени не раскрываются вообще.

Кроме самостоятельной функции `expand()`, существуют также флаги `expand` и `expand(p, n)` у функции `ev()`. Запись `выражение, expand` равносильна `expand(ev(выражение))`, а `выражение, expand(p, n)` — `expand(ev(выражение, p, n))`.

Возможности управлять раскрытием скобок на этом не заканчиваются. Еще одна функция — `distrib()` — представляет как бы облегченный вариант `expand()`. Она действует аналогично `expand()`, но только на один уровень в глубину:

```
(%i1) ((a+b)c+(b+c)d)(x+y)$
```

```
(%i2) distrib(%)
```

```
(%o2) (c+b)dy+(b+a)cy+(c+b)dx+(b+a)cx
```

```
(%i3) expand(%o1)
```

```
(%o3) cdy+bdy+bcy+acy+cdx+bdx+bcx+acx
```

В противоположность функциям `*expand*`(), раскрывающим скобки, можно также и разложить выражение на множители, то есть макси-

мально выносить все за скобки. Делается это с помощью функции `factor()`:

```
(%i1) factor(x^2-1)
```

```
(%o1) (x-1)(x+1)(x^2+1)(x^2-x+1)(x^2+x+1)(x^4+1)(x^4-x^2+1)(x^8-x^4+1)
```

Если функции `factor()` передать целое число, она разложит его на простые множители; если же передать рациональное число — на множители будут разложены его числитель и знаменатель:

```
(%i1) factor(2^100-1)
```

```
(%o1) 3^3 11 31 41 101 251 601 1801 4051 8101 268501
```

```
(%i2) factor(123456789/987654321)
```

```
(%o2) 
$$\frac{36073803}{17^2 379721}$$

```

Если многочлен не может быть представлен в виде произведения нескольких сомножителей, его можно попытаться преобразовать в сумму таких произведений с помощью функции `factorsum()`:

```
(%i1) factorsum(4yz+4xz+y^2+xy-vw-uw+tw)
```

```
(%o1) (y+x)(4z+y)-(v+u-t)w
```

Функция `factorsum()` умеет раскладывать на множители только независимые слагаемые, то есть такие, которые не содержат одинаковых переменных. Если мы раскроем скобки в выражении, содержащем в двух разных местах один и тот же символ, то так как коэффициенты при этом символе после раскрытия сгруппируются, `factorsum()` не сможет понять, каким именно образом разгруппировать их обратно:

```
(%i1) expand((x+y)^2+(x+z)^2)
```

```
(%o1) z^2+2xz+y^2+2xy+2x^2
```

```
(%i2) factorsum(%)
```

```
(%o2) z^2+2xz+y^2+2xy+2x^2
```

Нужно заметить, что функции `factor()` и `factorsum()`, хотя и не имеют в имени приставки `rat`, все же ведут себя в смысле разбора передаваемых им выражений не как `expand()` и сопутствующие, а как `ratexpand()`; то есть на любой не-рациональной функции останавливаются и внутрь не идут:

```
(%i1) factor(log(x^2+2x+1))
```

```
(%o1) log(x^2+2x+1)
```

```
(%i2) log(factor(x^2+2x+1))
```

```
(%o2) 2log(x+1)
```

Впрочем, об этом можно догадаться из документации, так как функции `factor*` описаны не в разделе `Simplification`, куда относятся `expand*`, а, так же, как и `rat*`, в разделе `Polynomials`.

Выносить за скобки, а также раскрывать эти скобки можно не только специальной функцией, но и дополнительным флагом ко все той же канонической форме рациональных выражений. Флаг этот зовут `ratfac`, и по умолчанию он равен `false`, то есть вынесение за скобки не происходит. Если же его установить в `true`, то в каждом рациональном выражении, приведенном к канонической форме, все будет максимально вынесено за скобки, но без вызова функции `factor()`; например, в примере ниже не произошло обратного свертывания $(x+1)^2$, хотя, будучи применен к первоначальному выражению, флаг `ratfac` сохранил и этот множитель нераскрытым (также можете сравнить этот

пример с аналогичным примером к функциям `ratexpand()` и `rat()`:

$$(\%i1) \text{ rat}\left(\frac{(x+1)^2 (e^x - 1)}{(x+2)(2x+1)}\right)$$

$$(\%o1) \frac{(x^2 + 2x + 1)e^x - x^2 - 2x - 1}{2x^2 + 5x + 2}$$

(%i2) `%,ratfac: true`

$$(\%o2) \frac{(e^x - 1)(x^2 + 2x + 1)}{2x^2 + 5x + 2}$$

$$(\%i3) \text{ rat}\left(\frac{(x+1)^2 (e^x - 1)}{(x+2)(2x+1)}\right), \text{ratfac: true}$$

$$(\%o3) \frac{(e^x - 1)(x+1)^2}{(2x+1)(x+2)}$$

Проще простого

Итак, о преобразованиях выражений мы уже говорили достаточно – теперь перейдем к их упрощению. Об элементарных упрощениях мы уже говорили в предыдущий раз: они могут производиться автоматически, на что влияет установленный флаг `simp`; и по умолчанию именно так и происходит.

Здесь тоже все начинается с рациональных выражений, которыми занимается функция `ratsimp(выражение)`. Она упрощает выражение за счет рациональных преобразований, но, в отличие от остальных функций по обработке рациональных выражений, работает в том числе и «вглубь», то есть иррациональные части выражения не рассматриваются как атомарные, а упрощаются, в том числе, и все рациональные элементы внутри них:

$$(\%i1) \text{ ratsimp}\left(\%e^{\frac{x^3+1}{x+1}}\right)$$

$$(\%o1) e^{x^2-x+1}$$

На `ratsimp()` действуют те же флаги, что и на `rat()`: и `ratexpand`, и `keepfloat`, и `ratfac`. Но отличается она от `rat()` или `ratexpand()` не только умением работать «в глубину», но и некоторыми дополнительными рациональными преобразованиями, которые не поддерживаются этими двумя функциями:

$$(\%i1) \frac{\sqrt{(x-a)^3} - (x+a)\sqrt{x-a}}{\sqrt{(x-a)(x+a)}} \$$$

(%i2) `rat(%)`

$$(\%o2) \frac{\sqrt{x-a^3} + (-x-a)\sqrt{x-a}}{\sqrt{(x-a)(x+a)}}$$

(%i3) `ratexpand(%o1)`

$$(\%o3) \frac{(x-a)^{\frac{3}{2}} - x\sqrt{x-a} - a\sqrt{x-a}}{\sqrt{(x-a)(x+a)}}$$

(%i4) `ratsimp(%o1)`

$$(\%o4) -\frac{2a\sqrt{x-a}}{\sqrt{x^2-a^2}}$$

Кроме функции `ratsimp()`, есть еще и дополнительный переключатель – `ratsimpexpons`. По умолчанию он установлен в `false`; если же назначить ему значение `true` – это приведет к автоматическому упрощению показателей степени:

$$(\%o1) x^{\frac{a^2+a+\frac{1}{4}}{2a+1}}$$

(%i2) `%,ratsimpexpons: true`

$$(\%o2) x^{\frac{2a+1}{4}}$$

Функция `ratsimp()` – это уже достаточно мощный, и в то же время весьма быстрый, механизм упрощения; но, конечно, не достаточный: ведь те действия, которые можно упростить в разнообразных математических выражениях, не ограничиваются рациональными. Поэтому все же основной плюс этой функции – это скорость. А для более серьезных упрощений существует расширенный вариант – `fullratsimp(выражение)`. Эта функция последовательно применяет к переданному выражению функцию `ratsimp()`, а также некоторые нерациональные преобразования – и повторяет эти действия в цикле до тех пор, пока выражение не перестанет в процессе них изменяться. За счет этого функция работает несколько медленнее, чем `ratsimp()`, зато дает более надежный результат – к некоторым выражениям, которые она может упростить с ходу, `ratsimp()` пришлось бы применять несколько раз, а иногда та и вообще не справилась бы с задачей.

$$(\%i1) \frac{(x^{\frac{a}{2}} - 1)^2 (x^{\frac{a}{2}} + 1)^2}{x^a - 1} \$$$

(%i4) `ratsimp(%)`

$$(\%o4) \frac{x^{2a} - 2x^a + 1}{x^a - 1}$$

(%i5) `ratsimp(%)`

$$(\%o5) x^a - 1$$

(%i6) `fullratsimp(%o1)`

$$(\%o6) x^a - 1$$

И третья основная функция упрощения выражений – уже никак с предыдущими двумя не соотносящаяся – `radcan(выражение)`. Если `ratsimp()` и `fullratsimp()` ориентированы на упрощение рациональных действий, то `radcan()` занимается упрощением логарифмических, экспоненциальных функций и степенных с нецелыми рациональными показателями, то есть корней (радикалов). Например, выражение из второго примера в этом разделе `radcan()` сможет упростить сильнее, чем `ratsimp()/fullratsimp()`:

$$(\%i1) \text{ ratsimp}\left(\frac{\sqrt{(x-a)^3} - (x+a)\sqrt{x-a}}{\sqrt{(x-a)(x+a)}}\right)$$

$$(\%o1) -\frac{2a\sqrt{x-a}}{\sqrt{x^2-a^2}}$$

(%i2) `fullratsimp(%)`

$$(\%o2) -\frac{2a\sqrt{x-a}}{\sqrt{x^2-a^2}}$$

(%i3) `radcan(%)`

$$(\%o3) -\frac{2a}{\sqrt{x+a}}$$

>>

(%i3) `trigreduce(%)`

(%o3) $\frac{\cos(2y+x)}{2} - \frac{\cos(2y-x)}{2} + \frac{\sin(y+2x)}{2} - \frac{\sin(y-2x)}{2} + \cos x \cos(2y) + \cos(2x) \sin y$

(%i4) `trigreduce(%)`

(%o4) $\cos(2y+x) + \sin(y+2x)$

<<

В некоторых случаях наилучшего результата можно добиться, комбинируя `radcan()` с `ratsimp()` или `fullratsimp()`.

С функцией `radcan()` смежны по действию еще два управляющих ключа. Один из них называется `%e_to_numlog`. Влияет он не на саму функцию, а на автоматическое упрощение. Если выставить его в `true`, то выражения вида $e^{(r \cdot \log(\text{выражение}))}$, где r – рациональное число, будут автоматически раскрываться в `выражение`. Функция `radcan()` делает такие преобразования независимо от значения ключа. Второй ключ – `radexpand` (от `radical`, не путать с `ratexpand`) – влияет на упрощение квадратного корня из четной степени какого-либо выражения. Он, в отличие от большинства переключателей, имеет не два, а три значения: при значении `all`, `sqrt(x^2)` будет раскрываться в x – как для действительных, так и для комплексных чисел; при значении `true` (по умолчанию), `sqrt(x^2)` для действительных чисел превращается в $|x|$, а для комплексных не преобразуется; а при значении `false`, `sqrt(x^2)` не будет упрощаться вообще.

Следующие две функции и один флаг относятся к упрощению факториалов. Функция `factcomb(выражение)` проводит упрощения вида $n!(n+1) = (n+1)!$ и тому подобные. Функция `minfactorial`, напротив, сокращает факториалы, то есть действует по принципу $n!/(n-1)! = n$. И флаг `sumsplitfact`, который изначально установлен в `true`, находясь в состоянии `false`, приводит к тому, что после того, как отработает `factcomb`, `minfactorial` вызывается автоматически.

Вот под таким углом...

И напоследок поговорим о функциях для преобразования тригонометрических формул. Здесь так же, как и у рациональных функций, присутствует общая для всех приставка – `trig`; расшифровывать ее, думаю, особой нужды нет. Начнем по традиции с функции `trigexpand(выражение)`. Она, как нетрудно догадаться, раскрывает скобки в тригонометрических выражениях:

(%i1) `trigexpand(sin(2x+y)+cos(x+2y))`

(%o1) $-\sin x \sin(2y) + \cos x \cos(2y) + \cos(2x) \sin y + \sin(2x) \cos y$

Здесь, как обычно, есть несколько управляющих флагов, первый из которых опять же является тезкой самой функции. Он приводит к повторному раскрытию всех синусов-косинусов, то есть фактически равнозначен повторному вызову самой функции:

(%i2) `%, trigexpand: true`

(%o2) $\cos x ((\cos y)^2 - (\sin y)^2) - 2 \sin x \cos y \sin y + ((\cos x)^2 - (\sin x)^2) \sin y + 2 \cos x \sin x \cos y$

Второй флаг – `halfangles` – управляет раскрытием формул половинных углов. Оба эти флага по умолчанию сброшены. А следующие два флага – `trigexpandplus` и `trigexpandtimes` – отвечают соответственно за применение формул сумм углов и кратных углов. То есть в примере выше сначала сработал флаг `trigexpandplus`, а затем – `trigexpandtimes`. Эти флаги по умолчанию установлены, что и видно из примера.

Кроме всего уже упомянутого, есть еще флаги `trigsign` и `triginverses`. Первый принимает традиционные два значения (по умолчанию – `true`) и регулирует вынос знака за пределы тригонометрической функции, то есть, к примеру, `sin(-x)` упростится до $-\sin(x)$, а `cos(-x)` – до $\cos(x)$. Флаг `triginverses` – трехзначный, и умолчательное его значение

равно `all`. Он отвечает за обработку сочетаний вида `sin(asin(x))` или `atan(tan(x))`. Значение `all` позволяет раскрывать эти сочетания в обоих направлениях (напомню, что при этом часть корней будет теряться); значение `true` оставляет разрешенным раскрытие только вида `sin(asin(x))`, то есть блокирует вариант с потерями периодических значений; а случай `false` запрещает оба направления преобразований.

Функция, обратная `trigexpand()`, называется `trigreduce(выражение)` – здесь, в полном соответствии со значением слова `reduce`, действуют формулы понижения степени. Например, применив дважды эту функцию к результату предыдущего примера, мы получим его в исходном виде (см. рис. наверху страницы).

Эту функцию можно вызвать с более полным списком аргументов: `trigreduce(выражение, переменная)` – тогда формулы понижения степени будут применяться только по отношению к заданной переменной (переменная может быть, как и почти везде, не только отдельным символом, но и выражением).

Третья функция занимается уже упрощением, и зовут ее, соответственно, `trigsimp(выражение)`. Она старается упростить любое тригонометрическое выражение, используя известные формулы, такие как $\sin^2(x) + \cos^2(x) = 1$ и тому подобные. Для наилучшего результата ее можно комбинировать с `trigreduce()`, `ratsimp()/fullratsimp()` и `radcan()`.

Этим возможности *Maxima* по преобразованию и упрощению разнообразных выражений еще не совсем исчерпаны, но основные из них мы рассмотрели в полной мере. В следующий раз поговорим немного о применении некоторых встроенных функций, о работе с векторами, матрицами и множествами и, возможно, о работе с логикой, с уравнениями и неравенствами, а также их системами. **LXF**

ЧЕРЕЗ МЕСЯЦ

Мы поговорим о линейной алгебре, теории множеств и уравнениях.