Основы Maxima. Структура Maxima

Пакет **Maxima** состоит из интерпретатора макроязыка, написанного на **Lisp**, и нескольких поколений пакетов расширений, написанных на макроязыке пакета или непосредственно на **Lisp.** Maxima позволяет решать достаточно широкий круг задач, относящихся к различным разделам математики.

Области математики, поддерживаемые в Maxima

- Операции с полиномами (манипуляция рациональными и степенными выражениями, вычисление корней и т.п.)
- Вычисления с элементарными функциями, в том числе с логарифмами, экспоненциальными функциями, тригонометрическими функциями
- Вычисления со специальными функциями, в т.ч. эллиптическими функциями и интегралами
- Вычисление пределов и производных
- Аналитическое вычисление определённых и неопределённых интегралов
- Решение интегральных уравнений
- Решение алгебраических уравнений и их систем
- Операции со степенными рядами и рядами Фурье
- Операции с матрицами и списками, большая библиотека функций для решения задач линейной алгебры
- Операции с тензорами
- Теория чисел, теория групп, абстрактная алгебра

Используются дополнительные пакеты для **Maxima**, которые необходимо загружать перед использованием. Они существенно расширяют базовые возможности и круг решаемых задач СКА **Maxima.**

основные пакеты расширения Maxima

augmented_lagrangian - Минимизация функции нескольких переменных с ограничениями методом неопределённых множителей Лагранжа (используется совместно с lbfgs)

bode - Построение диаграмм Боде (узкоспециальный пакет)

contrib_ode - Дополнительные функции для аналитического решения обыкновенных дифференциальных уравнений

Descriptive - Описательная статистика, оценка параметров распределения (генеральной совокупности) по выборке

diag - Пакет для операций с некоторыми видами диагональных матриц

distrib - Пакет, содержащий функции для расчёта различных распределений вероятностей и их параметров (нормальное распределение, распределение Стьюдента и т.п.)

draw - Интерфейс Maxima-Gnuplot. Предназначен для подготовки иллюстраций полиграфического качества

Dynamics - Различные функции, в т.ч. графические, относящиеся к моделированию динамических систем и фракталов

f90 - Экспорт кода Maxima в код на Фортран90

Основные пакеты расширения Maxima

ggf - Пакет включает единственную функцию, позволяющую оперировать с производящими функциями последовательностей (узкоспециальный пакет)

graphs - Пакет, включающий функции для работы с графами

grobner - Функции для того, чтобы работать с базисом Грёбнера (Groebner)

lmpdiff - вычисление производных неявных функций нескольких переменных

implicit_plot - Графики неявных функций

interpol - Пакет, включающий функции интерполяции (линейной, полиномами Лагранжа, сплайнами)

lapack - Функции пакета Lapack для решения задач линейной алгебры

Lbfgs - пакет минимизации функций нескольких переменных квазиньютоновским методом (L-BFGS)

Lindstedt - Пакет, рассчитанный на интерпретацию некоторый типов начальных условий для ОДУ, описывающих колебания

Lsquares - Функции для оценки параметров различных зависимостей методом наименьших квадратов

основные пакеты расширения Maxima

makeOrders - Пакет включает одну функцию для операций с полиномами

mnewton - Метод Ньютона для решения систем нелинейных уравнений

numericalio - Чтение и запись файлов (преимущественно с матричными числовыми данными)

opsubst - Пакет содержит одну функцию opsubst , позволяющую выполнять замену в выражениях (мало отличается от **subst**)

orthopoly - Пакет, содержащий функции для операций с ортогональными полиномами (Лежандра, Чебышева и др.)

plotdf - Пакет, позволяющий строить поле направлений для решения автономных систем (интересный, но довольно узкоспециальный пакет)

romberg - Пакет, включающий ряд функций для численного интегрирования

simplex - Пакет, предназначенный для решения задач линейного программирования

solve_rec - Пакет, содержащий функции для упрощения рекуррентных выражений

stats - Пакет, включающий функции для статистической проверки гипотез (о равенстве математических ожиданий или дисперсий выборок и т.п.

stirling - Расчёт гамма-функции

stringproc - Пакет, включающий функции для обработки строк

unit - Пакет, включающий функции для операций с единицами измерения

zeilberger - Функции для гипергеометрического суммирования

Основные преимуществами программы Maxima

•возможность свободного использования (**Maxima** относится к классу свободных программ и распространяется на основе лицензии GNU);

•возможность функционирования под управлением различных ОС (в частности **Linux** и **Windows™**);

•размер программы (дистрибутив занимает порядка 130 мегабайт, в установленном виде со всеми расширениями потребуется около 80 мегабайт);

•широкий класс решаемых задач;

•возможность работы как в консольной версии программы, так и с использованием одного из графических интерфейсов (**xMaxima, wxMaxima** или как плагин (plug-in) к редактору **TexMacs**);

•расширение **wxMaxima** (входящее в комплект поставки) предоставляет пользователю удобный и понятный интерфейс, избавляет от необходимости изучать особенности ввода команд для решения типовых задач;

•интерфейс программы на русском языке;

•наличие справки и инструкций по работе с программой (русскоязычной версии справки нет, но в сети Интернет присутствует большое количество статей с примерами использования **Maxima**);

Установка и интерфейсы

Скачать последнюю версию программы можно с её сайта: http://maxima.sourceforge.net/.
Русская локализация сайта: http://maxima.sourceforge.net/ru/.

Интерфейс wxMaxima

Графический интерфейс **wxMaxima** - является наиболее дружественным для начинающих пользователей системы. Достоинствами **wxMaxima** являются:

- ✓ возможность графического вывода формул
- ✓ упрощённый ввод наиболее часто используемых функций (через диалоговые окна), а не набор команд, как в классической **Maxima.**
- ✓ разделение окна ввода данных и области вывода результатов (в классической **Maxima** эти области объединены, и ввод команд происходит в единой рабочей области с полученными результатами).
- ✓ Выполнение набранных команд инициализируется клавишами ctrl-Enter.

Интерфейс командной строки

Является классическим интерфейсом **Maxima.** Используется опытными пользователями.

Вызов программы с разными интерфейсами выполняется через меню ПУСК.

Ввод простейших команд Maxima

Все команды вводятся в поле ВВОД, разделителем команд является символ ; (**точка с запятой**).

После ввода команды необходимо нажать клавишу **Enter** для её обработки и вывода результата.

Завершение ввода **символом $** (вместо точки с запятой) позволяет вычислить результат введённой команды, но не выводить его на экран.

В случае, когда выражение надо отобразить, а не вычислить, перед ним необходимо поставить знак ' (**одинарная кавычка**). Но этот метод не работает, когда выражение имеет явное значение, например, выражение заменяется на значение равное нулю.

Две одинарных кавычки последовательно, применённые к выражению во входной строке, приводят к замещению входной строки результатом вычисления вводимого выражения.

Примеры ввода команд

```
(%i1) → aa:1024;¶
```
$$(\%o1) \quad 1024_{\P}$$
```
(%i2) → bb:19;¶
```
$$(\%o2) \quad 19_{\P}$$
```
(%i3) → sqrt(aa)+bb;¶
```
$$(\%o3) \quad 51_{\P}$$
```
(%i4) → '(sqrt(aa)+bb);¶
```
$$(\%o4) \quad bb + \sqrt{aa}\,{}_{\P}$$
```
(%i5) → ''%;¶
```
$$(\%o5) \quad 51_{\P}$$

Обозначение команд и результатов вычислений

После ввода, каждой команде присваивается порядковый номер. В рассмотренном примере, введённые команды имеют номера 1–5 и обозначаются соответственно (%i1), (%i2) и т.д.

Результат вычисления также имеет порядковый номер, например (%o1), (%o2) и т.д., где **i** — сокращение от англ. input (ввод), а **o** — англ. output (вывод).

Этот механизм позволяет избежать в последующих вычислениях повторения полной записи уже выполненных команд, например (%i1)+(%i2) будет означать добавление к выражению первой команды — выражения второй и последующего вычисления результата.

Также можно использовать и номера результатов вычислений, например (%o1) * (%o2). Для последней выполненной команды в Maxima есть специальное обозначение — %.

Обозначение команд и результатов вычислений

Вычислить значение производной функции $y(x) = x^2 \cdot e^{-x}x$:¶

```
(%i1) diff(x^2*exp(-x),x);¶
```

$$(\%o1) \qquad 2\,x\,e^{-x} - x^2\,e^{-x}$$ ¶

```
(%i2) f(x):=''%;¶
```

$$(\%o2) \qquad f(x) := 2\,x\,e^{-x} - x^2\,e^{-x}$$ ¶

Двойная кавычка перед символом предыдущей операции позволяет заместить этот символ значением, т.е. текстовой строкой, полученной в результате дифференцирования.¶

Другой пример (с очевидным содержанием):¶

```
(%i3)  →  x:4;¶
```

$$(\%o3) \qquad 4$$ ¶

```
(%i4) → sqrt(x);¶
```

$$(\%o4) \qquad 2$$ ¶

```
(%i5)  →  %^2;¶
```

$$(\%o5) \qquad 4$$ ¶

¶

Ввод числовой информации

Правила ввода чисел в **Maxima** точно такие, как и для многих других подобных программ.

Целая и дробная часть десятичных дробей разделяются символом точка.

Перед отрицательными числами ставится знак минус.

Числитель и знаменатель обыкновенных дробей разделяется при помощи символа **/** (прямой слэш).

Обратите внимание, что если в результате выполнения операции получается некоторое символьное выражение, а необходимо получить конкретное числовое значение в виде десятичной дроби, то решить эту задачу позволит применение **флага numer**. В частности он позволяет перейти от обыкновенных дробей к десятичным.

Преобразование к форме с плавающей точкой осуществляет также функция **float**.

Пример ввода числовой информации

```
(%i1) 3/7+5/3;
```

$$(\%o1) \quad \frac{44}{21}$$

```
(%i2) 3/7+5/3, float;
```

$$(\%o2) \quad 2.095238095238095$$

```
(%i3) 3/7+5/3, numer;
```

$$(\%o3) \quad 2.095238095238095$$

```
(%i4) float(5/7);
```

$$(\%o4) \quad 0.71428571428571$$

Арифметические и логические операции

Обозначение арифметических операций в **Maxima** ничем не отличается от классического представления: + , — , * , /.

Возведение в степень можно обозначать несколькими способами: ^, ^^, **.

Извлечение корня степени n записываем, как степень 1/n.

Операция нахождение факториала обозначается восклицательным знаком, например 5!.

Для увеличения приоритета операции, как и в математике, используются круглые скобки: ().

Список основных арифметических и логических операторов приведён далее.

Арифметические и логические операции

Арифметические операторы (операции)

+	оператор сложения
-	оператор вычитания или изменения знака
*	оператор умножения
/	оператор деления
^ или **	оператор возведения в степень

Логические операторы (операции)

<	оператор сравнения меньше
>	оператор сравнения больше
<=	оператор сравнения меньше или равно
>=	оператор сравнения больше или равно
#	оператор сравнения не равно
=	оператор сравнения равно
and	логический оператор и
or	логический оператор или
not	логический оператор не

Основные «встроенные» константы Maxima

Основные константы Maxima

Название	Обозначение
слева (в отношении пределов)	$minus$
справа (в отношении пределов)	$plus$
плюс бесконечность	inf
минус бесконечность	$minf$
число π	$\%pi$
e (экспонента)	$\%e$
Мнимая единица $\sqrt{-1}$	$\%i$
Истина	$true$
Ложь	$false$
Золотое сечение $(1+\sqrt{5})/2$	$\%phi$

Типы данных, переменные и функции

Для хранения результатов промежуточных расчётов применяются переменные. Заметим, что при вводе названий переменных, функций и констант важен регистр букв, так переменные x и X— две разные переменные.

Присваивание значения переменной осуществляется с использованием символа **: (двоеточие),** например x:5.

Если необходимо удалить значение переменной (очистить её), то применяется метод (функция) **kill**:

kill(x) — удалить значение переменной x;

kill(all) — удалить значения всех используемых ранее переменных.

Зарезервированные слова, использование которых в качестве имён переменных вызывает синтаксическую ошибку:

integrate, next, from, diff, in, at, limit, sum, for, and, elseif, then, else, do, or, if, unless, product, while, thtu, step.

Списки

Списки — базовые строительные блоки для **Maxima** и **Lisp.**

Все прочие типы данных (массивы, хэш-таблицы, числа) представляются как списки.

Чтобы задать список, достаточно записать его элементы через запятую и ограничить запись квадратными скобками. Список может быть пустым или состоять из одного элемента.

```
(%i1) list1:[1,2,3,x,x+y];
```

$$(\%o1) \qquad [1, 2, 3, x, y + x]$$

```
(%i2) list2:[];
```

$$(\%o2) \qquad []$$

```
(%i3) list3:[3];
```

$$(\%o3) \qquad [3]$$

Списки

Элементом списка может и другой список

```
(%i4) list4:[1,2,[3,4],[5,6,7]];
```

$$(\%o4) \qquad 1, 2, [3, 4], [5, 6, 7]]$$

Ссылка на элемент списка производится по номеру элемента списка:

```
(%i4) list4:[1,2,[3,4],[5,6,7]];
```

$$(\%o4) \qquad [1, 2, [3, 4], [5, 6, 7]]$$

```
(%i5) list4[1];
```

$$(\%o5) \qquad 1$$

```
(%i6) list4[3];
```

$$(\%o6) \qquad [3, 4]$$

```
(%i7) list4[3][2];
```

$$(\%o7) \qquad 4$$

Функции для элементарных операций со списками

Функция **length** возвращает число элементов списка (при этом элементы списка сами могут быть достаточно сложными конструкциями):

```
(%i8) length(list4);
(%o8)                          4
(%i9) length(list3);
(%o9)                          1
```

Функция **copylist(expr)** возвращает копию списка **expr**:

```
(%i1) list1:[1,2,3,x,x+y];
(%o1)                  [1, 2, 3, x, y + x]
(%i2) list2:copylist(list1);
(%o2)                  [1, 2, 3, x, y + x]
```

Функции для элементарных операций со списками

Функция **makelist** создаёт список, каждый элемент которого генерируется из некоторого выражения.

Возможны два варианта вызова этой функции:

$makelist(expr, i, i_0, i_1)$ — возвращает список, j -й элемент которого равен $ev(expr, i = j)$, при этом индекс j меняется от i_0 до i_1

$makelist(expr, x, list)$ — возвращает список, j -й элемент которого равен $ev(expr, x = list[j])$, при этом индекс j меняется от 1 до $length(list)$.

Примеры функции makelist

Примеры функции **makelist**:

```
(%i1)   makelist(concat(x,i),i,1,6);
```

$$(\%o1) \quad [x1, x2, x3, x4, x5, x6]$$

```
(%i2)   list:[1,2,3,4,5,6,7];
```

$$(\%o2) \quad [1, 2, 3, 4, 5, 6, 7]$$

```
(%i3)   makelist(exp(i),i,list);
```

$$(\%o3) \quad [e, e^2, e^3, e^4, e^5, e^6, e^7]$$

Функция createlist

функция **createlist**:

$$createlist(form, x_1, list_1, ..., x_n, list_n).$$

Эта функция строит список путём вычисления выражения $form$, зависящего от x_1, к каждому элементу списка $list_1$(аналогично form, зависящая и от x_2, применяется к $list_2$и т.д.).

Пример:

```
(%i1) create_list(x^i,i,[1,3,7]);
```

$$(\%o1) \qquad [x, x^3, x^7]$$

```
(%i2) create_list([i,j],i,[a,b],j,[e,f,h]);
```

$$(\%o2) \qquad [[a,e],[a,f],[a,h],[b,e],[b,f],[b,h]]$$

Функция append

функция **append**

Функция **append** позволяет склеивать списки. При вызове

```
append (list_1, . . ., list_n)
```

возвращается один список, в котором за элементами $list_1$ следуют элементы $list_2$ и т.д. вплоть до $list_n$.

Пример:

```
(%i1) append([1],[2,3],[4,5,6,7]);
```

$$(\%o1) \qquad [1, 2, 3, 4, 5, 6, 7]$$

Функция join

Создать новый список, компонуя элементы двух списков поочерёдно в порядке следования, позволяет функция **join**(l,m).

Новый список содержит l_1, затем m_1, затем l_2, m_2 и т.д.

Пример:

```
(%i1)   join([1,2,3],[10,20,30]);
```

$$(\%o1) \qquad [1, 10, 2, 20, 3, 30]$$

```
(%i2)  join([1,2,3],[10,20,30,40]);
```

$$(\%o2) \qquad [1, 10, 2, 20, 3, 30]$$

Длина полученного списка ограничивается минимальной длиной списков l и m.

Функции cons и endcons

Функция $cons(expr, list)$ создаёт новый список, первым элементом, которого будет $expr$, а остальные — элементы списка $list$.

Функция $endcons(expr, list)$ также создаёт новый список, первые элементы которого — элементы списка $list$, а последний — новый элемент $expr$.

Пример:

```
(%i1) cons(x,[1,2,3]);
```

$$(\%o1) \qquad [x, 1, 2, 3]$$

```
(%i2) endcons(x,[1,2,3]);
```

$$(\%o2) \qquad [1, 2, 3, x]$$

Функции reverse и member

Функция **reverse** меняет порядок элементов в списке на обратный

```
(%i5) list1:[1,2,3,x];
```
$$(\%o5) \quad [1, 2, 3, x]$$
```
(%i6) list2:reverse(list1);
```
$$(\%o6) \quad [x, 3, 2, 1]$$

Функция $member(expr_1, expr_2)$ возвращает $true$, если $expr_1$ является элементом списка $expr_2$, и $false$ в противном случае.

Пример:

```
(%i1) member (8, [8, 8.0, 8b0]);
```
$$(\%o1) \quad true$$
```
(%i2) member (8, [8.0, 8b0]);
```
$$(\%o2) \quad false$$
```
(%i3) member (b, [[a, b], [b, c]]);
```
$$(\%o3) \quad false$$
```
(%i4) member ([b, c], [[a, b], [b, c]]);
```
$$(\%o4) \quad true$$

Функции rest, last и first

Функция $rest(expr)$ выделяет остаток после удаления первого элемента списка $expr$. Можно удалить первые n элементов, используя вызов $rest(expr, n)$.

Функция $last(expr)$ выделяет последний элемент $expr$, аналогично $first$ — первый элемент списка.

Примеры:

```
(%i1)    list1:[1,2,3,4,a,b];
```
$$(\%o1) \quad [1, 2, 3, 4, a, b]$$
```
(%i2)    rest(list1);
```
$$(\%o2) \quad [2, 3, 4, a, b]$$
```
(%i3)    rest(%);
```
$$(\%o3) \quad [3, 4, a, b]$$
```
(%i4)    last(list1);
```
$$(\%o4) \quad b$$
```
(%i5)    rest(list1,3);
```
$$(\%o5) \quad [4, a, b]$$

Функции суммирование и перемножение списков

Суммирование и перемножение списков, как и прочих выражений, осуществляется функциями sum и $product$.

Функция $sum(expr, i, in, ik)$ суммирует значения выражения expr при изменении индекса i от in до ik.

Функция $product(expr, i, in, ik)$ перемножает значения выражения $expr$ при изменении индекса i от in до ik.

Пример:

```
(%i1)    product (x + i*(i+1)/2, i, 1, 4);
```

$$(\%o1) \quad (x+1)\,(x+3)\,(x+6)\,(x+10)$$

```
(%i2)    sum (x + i*(i+1)/2, i, 1, 4);
```

$$(\%o2) \quad 4x+20$$

```
(%i3)    product (i^2, i, 1, 4);
```

$$(\%o3) \quad 576$$

```
(%i4)    sum (i^2, i, 1, 4);
```

$$(\%o4) \quad 30$$

Функция, оперирующая элементами списков - map

Функция $map(f, expr_1, \ldots, expr_n)$ позволяет применить функцию (оператор, символ операции) f к частям выражений $expr_1, expr_2, \ldots, expr_n$.

При использовании со списками применяет f к каждому элементу списка.

Следует обратить внимание, что f — именно имя функции без указания переменных, от которых она зависит.

Функция f может быть заданной пользователем, например `f(x):=x^2;`

Примеры функции map

```
(%i1)  map(ratsimp, x/(x^2+x)+(y^2+y)/y);
```

$$(\%o1) \qquad y + \frac{1}{x+1} + 1$$

```
(%i2)  map("=",[a,b],[-0.5,3]);
```

$$(\%o2) \qquad [a = -0.5, b = 3]$$

```
(%i3)  map(exp,[0,1,2,3,4,5]);
```

$$(\%o3) \qquad [1, e, e^2, e^3, e^4, e^5]$$

Функция f может быть заданной пользователем, например:

```
(%i5)  f(x):=x^2;
```

$$(\%o5) \qquad f(x) := x^2$$

```
(%i6)  map(f,[1,2,3,4,5]);
```

$$(\%o6) \qquad [1, 4, 9, 16, 25]$$

Функция apply

Функция $apply$ применяет заданную функцию ко всему списку.

Список становится списком аргументов функции.

При вызове $F, [x_1, ..., x_n]$ вычисляется выражение $F(arg_1, ..., arg_n)$.

Следует учитывать, что $apply$ не распознаёт ординарные функции и функции от массива.

Примеры функции apply

```
(%i1)    L : [1, 5, -10.2, 4, 3];
```
$$(\%o1) \qquad [1, 5, -10.2, 4, 3]$$
```
(%i2)    apply(max,L);
```
$$(\%o2) \qquad 5$$
```
(%i3)    apply(min,L);
```
$$(\%o3) \qquad -10.2$$

Чтобы найти максимальный или минимальный элемент набора чисел, надо вызвать функции max или min. Однако, обе функции в качестве аргумента ожидают несколько чисел, а не список, составленный из чисел. Применять подобные функции к спискам позволяет функция $apply$.

Массивы

Массивы в **Maxima** — совокупности однотипных объектов с индексами. Число индексов не должно превышать пяти. В **Maxima** существуют и функции с индексами (функции массива).

Возможно создание и использование переменных с индексами до объявления соответствующего массива. Такие переменные рассматриваются как элементы массивов с неопределёнными размерностями (так называемые хэш-массивы). Размеры неопределённых массивов растут динамически по мере присваивания значений элементам. Интересно, что индексы массивов с неопределёнными границами не обязательно должны быть числами.

Для повышения эффективности вычислений рекомендуется преобразовывать массивы с неопределёнными границами в обычные массивы (для этого используется функция **array**.

Создание массива

Создание массива производится функцией **array**.

array(**name, dim$_1$,..., dim$_n$**)

— создание массива с именем **name** и размерностями **dim$_1$, ..., dim$_n$**;

array(name, type, dim$_1$, ..., dim$_n$)

— создание массива с именем **name** и элементами типа **type**;

array([name$_1$, ..., name$_m$], dim$_1$, …., dim$_n$)

— создание нескольких массивов одинаковой размерности.

Индексы обычного массива — целые числа, изменяющиеся от 0 до **dim$_n$**

Пример array и функция listarray

```
(%i1)  array(a,1,1);
(%o1)                          a
(%i2)  a[0,0]:0; a[0,1]:1;
            a[1,0]:2; a[1,1]:3;
(%o5)                        0123
(%i6)  listarray(a);
(%o6)                     [0, 1, 2, 3]
```

Функция **listarray**, использованная в примере, преобразует массив в список. Синтаксис вызова: **listarray**(**a**). Аргумент **a** может быть определённым или неопределённым массивом, функцией массива или функцией с индексами. Порядок включения элементов массива в список — по строкам.

Функция arrayinfo

Функция **arrayinfo** выводит информацию о массиве A.

Синтаксис вызова: **arrayinfo(A)**

Аргумент A, как и в случае **listarray**, может быть определённым или неопределённым массивом, функцией массива или функцией с индексами

Функции **listarray** и **arrayinfo** применимы и к функциям массива

Пример использования arrayinfo и listarray

```
(%i1)    array (aa, 2, 3);
```
$$(\%o1) \quad aa$$
```
(%i2)    aa [2, 3] : %pi;
```
$$(\%o2) \quad \pi$$
```
(%i3)    aa [1, 2] : %e;
```
$$(\%o3) \quad e$$
```
(%i4)    arrayinfo (aa);
```
$$(\%o4) \quad [declared, 2, [2, 3]]$$
```
(%i5)    bb [FOO] : (a + b)^2;
```
$$(\%o5) \quad (b+a)^2$$
```
(%i6)    bb [BAR] : (c - d)^3;
```
$$(\%o6) \quad (c-d)^3$$
```
(%i7)    arrayinfo (bb);
```
$$(\%o7) \quad [hashed, 1, [BAR], [FOO]]$$
```
(%i8)    listarray (bb);
```
$$(\%o8) \quad [(c-d)^3, (b+a)^2]$$

Пример использования arrayinfo и listarray

Функции **listarray** и **arrayinfo** применимы к функциям массива:

```
(%i9)     cc [x, y] := y / x;
```

(%o9) $$cc_{x,y} := \frac{y}{x}$$

```
(%i10)    cc[1,2];
```

(%o10) $$2$$

```
(%i11)    cc[2,1];
```

(%o11) $$\frac{1}{2}$$

```
(%i12)    arrayinfo(cc);
```

(%o12) $$[hashed, 2, [1, 2], [2, 1]]$$

```
(%i13)    listarray(cc);
```

(%o13) $$[2, \frac{1}{2}]$$

пример — создание и вывод информации о функциях с индексами

```
(%i1)  dd [x] (y) := y ^ x;
```

$$(\%o1) \qquad dd_x(y) := y^x$$

```
(%i2)  dd[1](4);
```

$$(\%o2) \qquad 4$$

```
(%i3)  dd[a+b];
```

$$(\%o3) \qquad lambda\left([y], y^{b+a}\right)$$

```
(%i4)  arrayinfo(dd);
```

$$(\%o4) \qquad [hashed, 1, [1], [b+a]]$$

```
(%i5)  listarray(dd);
```

$$(\%o5) \qquad [lambda([y], y), lambda\left([y], y^{b+a}\right)]$$

Функция make_array

Функция **make_array(type, dim_1, …, dim_n)** создаёт и возвращает массив Lisp. Тип массива может быть **any, flonum, fixnum, hashed, functional**. Индекс i может изменяться в пределах от 0 до **dim_n-1**

Достоинство **make_array** по сравнению с **array**— возможность динамически управлять распределением памяти для массивов.

Присваивание **y: make_array(…)** создаёт ссылку на массив.

Когда массив больше не нужен, ссылка уничтожается присваиванием **y:false**, память освобождается затем сборщиком мусора.

Примеры функции make_array

```
(%i1)  A1 : make_array (fixnum, 8);
(%o1)      Lisp Array : #(0 0 0 0 0 0 0 0)
(%i2)  A1[1]:8;
(%o2)                          8
(%i3)  A3 : make_array (any, 8);
(%o3)
Lisp Array : #(NIL NIL NIL NIL NIL NIL NIL NIL)
(%i4)  arrayinfo(A3);
(%o4)                  [declared, 1, [7]]
```

Переменная arrays

Переменная **arrays** содержит список имён массивов первого и второго видов, определённых на данный момент.

Пример:

```
(%i1) array(a,1,1);
```

$$(\%o1) \qquad a$$

```
(%i2) array(b,2,3);
```

$$(\%o2) \qquad b$$

```
(%i3) arrays;
```

$$(\%o3) \qquad [a, b]$$

Функция fillarray

Функция **fillarray** позволяет заполнять массивы значениями из другого массива или списка. Заполнения производится по строкам.
Примеры:

```
(%i1)     array(a,1,1);
(%o1)                          a

(%i2)     fillarray(a,[1,2,3,4]);
(%o2)                          a

(%i3)     a[1,1];
(%o3)                          4

(%i4)     a2 : make_array (fixnum, 8);
(%o4)     Lisp Array #(0 0 0 0 0 0 0 0)

(%i5)     fillarray (a2, [1, 2, 3, 4, 5]);
(%o5)     Lisp Array #(1 2 3 4 5 5 5 5)
```

Изменение размерности массива

Длина списка может и не совпадать с размерностью массива. Если указан тип массива, он должен заполняться

элементами того же типа.

Удаление массивов из памяти осуществляется функцией **remarray**.

Для изменения размерности массива имеется функция **rarray(A, dim_1, …, dim_n).**

Новый массив заполняется элементами старого по строкам. Если размер старого массива меньше, чем нового, остаток нового заполняется нулями или **false** (в зависимости от типа массива).

Матрицы и простейшие операции с ними

В **Maxima** определены прямоугольные матрицы.

Основной способ создания матриц — использования функции **matrix**.

Синтаксис вызова:

matrix(row1, …, rown).

Каждая строка — список выражений, все строки одинаковой длины.

На множестве матриц *определены операции сложения, вычитания, умножения и деления*. ***Эти операции выполняются поэлементно***, если операнды — две матрицы, скаляр и матрица или матрица и скаляр.

Возведение в степень возможно, если один из операндов — скаляр.

Перемножение матриц (в общем случае некоммутативная операция) обозначается символом "." (**точка**).

Операция умножения матрицы самой на себя может рассматриваться как возведение в степень.

Возведение в степень -1 — как обращение (если это возможно).

Пример создания двух матриц

```
(%i1)  x: matrix ([17, 3], [-8, 11]);
```

$$(\%o1)\begin{bmatrix} 17 & 3 \\ -8 & 11 \end{bmatrix}$$

```
(%i2)  y: matrix ([%pi, %e], [a, b]);
```

$$(\%o2)\begin{bmatrix} \pi & e \\ a & b \end{bmatrix}$$

Выполнение арифметических операций с матрицами

```
(%i3)     x+y;
```

$$(\%o3)\begin{bmatrix} \pi+17 & e+3 \\ a-8 & b+11 \end{bmatrix}$$

```
(%i4)     x-y;
```

$$(\%o4)\begin{bmatrix} 17-\pi & 3-e \\ -a-8 & 11-b \end{bmatrix}$$

Выполнение арифметических операций с матрицами

```
(%i5) x*y;
```

$$(\%o5)\begin{bmatrix} 17\,\pi & 3\,e \\ -a\,8 & 11\,b \end{bmatrix}$$

```
(%i6) x/y;
```

$$(\%o6)\begin{bmatrix} \frac{17}{\pi} & 3\,e^{-1} \\ -\frac{8}{a} & \frac{11}{b} \end{bmatrix}$$

Обратите внимание — операции выполняются поэлементно.

При попытке выполнять арифметические операции, как представлено выше, над матрицами различных размеров, выдаётся ошибка.

Пример операций с матрицами и скалярами

```
(%i1) x: matrix([17, 3],[-8, 11]);
```

$$(\%o1)\begin{bmatrix} 17 & 3 \\ -8 & 11 \end{bmatrix}$$

```
(%i9) x^3;
```

$$(\%o9)\begin{bmatrix} 4913 & 27 \\ -512 & 1331 \end{bmatrix}$$

```
(%i10)     3^x;
```

$$(\%o10)\begin{bmatrix} 129140163 & 27 \\ \frac{1}{6561} & 177147 \end{bmatrix}$$

Умножение матрицы на матрицу

```
(%i1)  x: matrix([17, 3],[-8, 11]);
```

$$(\%o1)\begin{bmatrix} 17 & 3 \\ -8 & 11 \end{bmatrix}$$

```
(%i2)  y: matrix([%pi, %e], [a, b]);
```

$$(\%o2)\begin{bmatrix} \pi & e \\ a & b \end{bmatrix}$$

Умножение матрицы на матрицу:

```
(%i11)  x.y;
```

$$(\%o11)\begin{bmatrix} 3a+17\pi & 3b+17e \\ 11a-8\pi & 11b-8e \end{bmatrix}$$

```
(%i12)  y.x;
```

$$(\%o12)\begin{bmatrix} 17\pi-8e & 3\pi+11e \\ 17a-8b & 11b+3a \end{bmatrix}$$

Очевидно, что для успешного перемножения матрицы должны быть согласованы по размерам

обратная матрица

```
(%i1)   x: matrix([17, 3],[-8, 11]);
```

$$(\%o1)\begin{bmatrix} 17 & 3 \\ -8 & 11 \end{bmatrix}$$

Возведение в степень -1 даёт обратную матрицу:

```
(%i13)  x^^-1;
```

$$(\%o13)\begin{bmatrix} \frac{11}{211} & -\frac{3}{211} \\ \frac{8}{211} & \frac{17}{211} \end{bmatrix}$$

```
(%i14)  x.(x^^-1);
```

$$(\%o14)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

операции x^^-1 и x^-1

Обратите внимание, что операции x^^-1 и x^-1 дают разный результат!

```
(%i1) x: matrix([17, 3],[-8, 11]);
```

$$(\%o1)\begin{bmatrix} 17 & 3 \\ -8 & 11 \end{bmatrix}$$

```
(%i2) x^-1;
```

$$(\%o2)\begin{bmatrix} \frac{1}{17} & \frac{1}{3} \\ -\frac{1}{8} & \frac{1}{11} \end{bmatrix}$$

```
(%i3) x^^-1;
```

$$(\%o3)\begin{bmatrix} \frac{11}{211} & -\frac{3}{211} \\ \frac{8}{211} & \frac{17}{211} \end{bmatrix}$$

Функция genmatrix

Функция **genmatrix** возвращает матрицу заданной размерности, составленную из элементов двух индексного массива.

Синтаксис вызова:

- $genmatrix(a, i_2, j_2, i_1, j_1)$
- $genmatrix(a, i_2, j_2, i_1)$
- $genmatrix(a, i_2, j_2)$

Индексы i_1, j_1 и i_2, j_2 указывают левый и правый нижний элементы матрицы в исходном массиве.

Примеры функции genmatrix

```
(%i1)  h [i, j] := 1 / (i + j - 1);
```

$$(\%o1) \qquad h_{i,j} := \frac{1}{i+j-1}$$

```
(%i2)  genmatrix(h,3,3);
```

$$(\%o2) \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

```
(%i3)  array (a, fixnum, 2, 2);
```

$$(\%o3) \qquad a$$

```
(%i4)  a [1, 1] : %e;
```

$$(\%o4) \qquad e$$

```
(%i5)  a [2, 2] : %pi;
```

$$(\%o5) \qquad \pi$$

```
(%i6)  genmatrix (a, 2, 2);
```

$$(\%o6) \begin{bmatrix} e & 0 \\ 0 & \pi \end{bmatrix}$$

Функции zeromatrix и ident

Функция **zeromatrix** возвращает матрицу заданной размерности, составленную из нулей.

Синтаксис вызова: **zeromatrix(n,m).**

```
(%i7)  zeromatrix(2,2);
```

$$(\%o7)\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Функция **ident** возвращает единичную матрицу заданной размерности.

Синтаксис вызова **ident** (n).

```
(%i9)  ident(2);
```

$$(\%o9)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Функция copymatrix

Функция **copymatrix(M)** создаёт копию матрицы M.
Обратите внимание, что присваивание не создаёт копии матрицы (как и присваивание не создаёт копии списка).
Пример:

```
(%i1)    a:matrix([1,2],[3,4]);
```

$$(\%o1)\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

```
(%i2)    b:a;
```

$$(\%o2)\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

```
(%i3)    b[2,2]:10;
```

$$(\%o3)\qquad 10$$

```
(%i4)    a;
```

$$(\%o4)\begin{bmatrix}1 & 2\\ 3 & 10\end{bmatrix}$$

Присваивание нового значения элементу матрицы b изменяет и значение соответствующего элемента матрицы a.
Использование **copymatrix** позволяет избежать этого эффекта.

Функции row, col и addrow и addcol

Функции **row** и **col** позволят извлечь соответственно строку и столбец заданной матрицы, получая список.
Синтаксис вызова:

- $row(M, i)$ — возвращает i -ю строку;
- $col(M, i)$ — возвращает i -й столбец.

Функции **addrow** и **addcol** добавляют к матрице строку или столбец соответственно.
Синтаксис вызова:

- $addcol(M, list_1, ..., list_n)$
- $addrow(M, list_1, ..., list_n)$

Здесь $list_1, ..., list_n$ — добавляемые строки или столбцы.

Примеры функции row, col и addrow и addcol

```
(%i1)  a:matrix([1,2],[3,4]);
```

$$(\%o1)\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
(%i2)  b:addrow(a,[10,20]);
```

$$(\%o2)\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 10 & 20 \end{bmatrix}$$

```
(%i3)  addcol(b,[x,y,z]);
```

$$(\%o3)\begin{bmatrix} 1 & 2 & x \\ 3 & 4 & y \\ 10 & 20 & z \end{bmatrix}$$

Функция submatrix

Функция **`submatrix(M)`** возвращает новую матрицу, состоящую из подматрицы заданной.

Синтаксис вызова:

- $submatrix(i_1, ..., i_m, M, j_1, ..., j_n)$
- $submatrix(i_1, ..., i_m, M)$
- $submatrix(M, j_1, ..., j_n)$

Подматрица строится следующим образом:
из матрицы M удаляются строки и,
соответственно, столбцы - $i_1, \ldots, i_m$ и $j_1, \ldots, j_n$.

Пример функции submatrix

Пример: удаляем третью строку и третий столбец:

```
(%i2) M:matrix([1,2,x], [3,4,y], [10,20,z]);
                                  [ 1   2   x ]
                                  [           ]
(%o2)                             [ 3   4   y ]
                                  [           ]
                                  [ 10  20  z ]

(%i22) submatrix(3,M,3);
                                   [ 1  2 ]
(%o22)                             [      ]
                                   [ 3  4 ]
```

Функция matrixmap

Для заполнения матрицы значениями некоторой функции используется функция **matrixmap** (аналог **map, apply, fullmap**).
Синтаксис вызова: **matrixmap(f,M).**

Функция **matrixmap** возвращает матрицу с элементами i, j, равными $f(M[i,j])$.

Пример:

```
(%i1)  a:matrix([1,2],[3,4]);
```

$$(\%o1)\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

```
(%i2)  f(x):=x^2;
```

$$(\%o2)\qquad f(x) := x^2$$

```
(%i3)  matrixmap(f,a);
```

$$(\%o3)\begin{bmatrix}1 & 4\\ 9 & 16\end{bmatrix}$$

Для работы с матрицами существует ещё много функций, но они относятся к решению различных задач линейной алгебры и обсуждаются ниже.

Математические функции - 1

В **Maxima** имеется достаточно большой набор встроенных математических функций. Перечень основных классов встроенных функций приведён ниже:

- тригонометрические функции: sin (синус), cos (косинус), tan (тангенс), cot (котангенс);
- обратные тригонометрические функции: $asin$ (арксинус), $acos$ (арккосинус), $atan$ (арктангенс), $acot$ (арккотангенс);
- sec (секанс, $sec(x) = \frac{1}{cos(x)}$, csc (косеканс, $csc(x) = \frac{1}{sin(x)}$);

Математические функции -2

- $sinh$ (гиперболический синус),
  $cosh$ (гиперболический косинус),
  $tanh$ (гиперболический тангенс),
  $coth$ (гиперболический котангенс),
  $sech$ (гиперболический секанс),
  $cosh$ (гиперболический косеканс);
- log (натуральный логарифм);
- $sqrt$ (квадратный корень);
- mod (остаток от деления);
- abs (модуль);
- $min(x_1, \ldots, x_n)$ и $max(x_1, \ldots, x_n)$ — нахождение минимального и максимального значения в списке аргументов;

Математические функции -3

- $sign$ - определяет знак аргумента:
  pos — положительный,
  neg — отрицательный,
  pnz — не определён,
  $zero$ — значение равно нулю;
- Специальные функции — функции Бесселя, гамма-функция, гипергеометрическая функция и др.;
- Эллиптические функции различных типов.

Функция **ev** является основной функцией, обрабатывающей выражения.
Синтаксис вызова: **ev(expr, arg1, …., argn).**

Функция **ev** вычисляет выражение **expr** в окружении, определяемом аргументами **arg1, …., argn**.

Аргументы могут быть ключами, булевскими флагами, присваиваниями, уравнениями и функциями.

Функция **ev** возвращает результат - другое выражение.

Во многих случаях можно опускать имя функции **ev** (т.е. применять значения переменных к некоторому выражению, не указывая **ev**):

expr, flag1, flag2, ...;

expr, x=val1, y=val2,...;

expr, flag1, x=val1, y=val2,..., flag2, ...;

На выражение **expr** по умолчанию действует функция упрощения.

Необходимость выполнения упрощения регулируется флагом **simp** - если установить **simp** : **false** упрощение будет отключено.

Кроме того, используют флаги **float** и **number**, определяющие формат представления рациональных чисел - в виде дробей или с плавающей точкой и результатов вычисления математических функций.

Флаг **pred** определяет необходимость вычисления применительно к логическим выражениям.

Аргументами **ev** могут быть и встроенные функции, выполняющие упрощение или преобразование выражений **expand, factor, trigexpand, trigreduce** или функция **diff**.

При этом повторный вызов функции **ev** вполне способен ещё раз изменить выражение, т.е. обработка выражения не идёт до конца при однократном вызове функции **ev**.

Пример:

```
(%i1)     ev((a+b)^2,expand);
```

$$(\%o1) \qquad b^2 + 2\,a\,b + a^2$$

```
(%i2)     ev((a+b)^2,a=x);
```

$$(\%o2) \qquad (x + b)^2$$

```
(%i3)     ev((a+b)^2,a=x,expand,b=7)
```

$$(\%o3) \qquad x^2 + 14\,x + 49$$

Другой пример показывает применение **diff** к *отложенному вычислению* производной:

```
(%i1)    sin(x)+cos(y)+(w+1)^2+'diff (sin(w), w);
```

$$(\%o1) \quad cos(y) + sin(x) + \frac{d}{dw} sin(w) + (w+1)^2$$

```
(%i2)    ev (%, sin, expand, diff, x=2, y=1);
```

$$(\%o2) \quad cos(w) + w^2 + 2w + cos(1) + 1.909297426825682$$

Флаг **simp** разрешает либо запрещает упрощение выражений. Изначально он равен **true**, если установить его равным **false**, то упрощения производиться не будут:

```
(%i1)    f:a+2*a+3*a+4*a;
```

$$(\%o1) \qquad 10\,a$$

```
(%i2)    simp:false;
```

$$(\%o2) \qquad false$$

```
(%i3)    f:a+2*a+3*a+4*a;
```

$$(\%o3) \qquad a+2\,a+3\,a+4\,a$$

Функцию **ev** не обязательно указывать явно, например:

```
(%i3)    x+y, x: a+y, y: 2;
```

$$(\%o3) \qquad y+a+2$$

Оператор принудительного вычисления

Оператор, принудительного вычисления, обозначенный *двумя апострофами*, является синонимом к функции ev (выражение).

Сама функция **ev** предоставляет гораздо более широкие возможности, нежели простое принудительное вычисление заданного выражения:

она может принимать произвольное число аргументов, первый из которых — вычисляемое выражение, а остальные — специальные опции, которые как раз и влияют на то, как именно будет производиться вычисление.

Форма выражения

В терминологии **Maxima** невычисленная форма выражения называется ***"noun form",*** вычисленная — ***"verb form".***

Сохраняя лингвистические параллели, на русский это можно перевести как **"несовершенная форма"** и **"совершённая форма".**

Значение вводимого выражения в **Maxima** закономерно сохраняется до его вычисления (т. е. в несовершенной форме). А значение выводимого выражения — после (т. е. в совершённой).

Другими словами, тут имеется естественный порядок "**ввод — вычисление — вывод**".

Функция **factor** и **gfactor**

Функция **factor** факторизует, т.е. представляет в виде произведения некоторых сомножителей, заданное выражение.

Функция **gfactor** — действует аналогично, но на множестве комплексных чисел и выражений.

Пример:

```
(%i1)    x^3-1,factor;
```

$$(\%o1) \qquad (x-1)\left(x^2+x+1\right)$$

```
(%i2)    factor(x^3-1);
```

$$(\%o2) \qquad (x-1)\left(x^2+x+1\right)$$

Примеры факторизации различных выражений

```
(%i3)  factor (-8*y - 4*x + z^2*(2*y + x));
```

$$(\%o3) \qquad (2y+x)\,(z-2)\,(z+2)$$

```
(%i4)  factor (2^63 - 1);
```

$$(\%o4) \qquad 7^2\,73\,127\,337\,92737\,649657$$

```
(%i5)  factor (1 + %e^(3*x));
```

$$(\%o5) \qquad (e^x+1)\left(e^{2x}-e^x+1\right)$$

Пример использования функции **gfactor**:

```
(%i6)  gfactor(x^2+a^2);
```

$$(\%o6) \qquad (x-i\,a)\,(x+i\,a)$$

```
(%i7)  gfactor(x^2+2*%i*x*a-a^2);
```

$$(\%o7) \qquad (x+i\,a)^2$$

Функция **expand**

Функция **expand** раскрывает скобки,
выполняет умножение, возведение в степень, например:

```
(%i1)    expand((x-a)^3);
```

$$(\%o1) \qquad x^3 - 3\,a\,x^2 + 3\,a^2\,x - a^3$$

```
(%i2)    expand((x-a)*(y-b)*(z-c));
```

$$(\%o2) \quad x\,y\,z - a\,y\,z - b\,x\,z + a\,b\,z - c\,x\,y + a\,c\,y + b\,c\,x - a\,b\,c$$

```
(%i3)    expand((x-a)*(y-b)^2);
```

$$(\%o3) \qquad x\,y^2 - a\,y^2 - 2\,b\,x\,y + 2\,a\,b\,y + b^2\,x - a\,b^2$$

Функция **factorsum**

Функция **factorsum** факторизует отдельные слагаемые в выражении.

```
(%i8)    expand ((x + 1)*((u + v)^2 + a*(w + z)^2));
```

(%o8)

$$axz^2+az^2+2awxz+2awz+aw^2x+v^2x+2uvx+u^2x+aw^2+v^2+2uv+u^2$$

```
(%i9)    factorsum(%);
```

(%o9) $$(x+1)\left(a(z+w)^2+(v+u)^2\right)$$

Функция **gfactorsum** отличается от **factorsum** тем же,
чем **gfactor** отличается от **factor**:

```
   gfactorsum (a^3+3*a^2*b+3*a*b^2+b^3+x^2+2*%i*x*y-y^2 );
```

(%o10) $$(b+a)^3-(y-ix)^2$$

Функция **combine**

Функция **combine** объединяет слагаемые с идентичным знаменателем

```
(%i5)    combine(x/(1+x^2)+y/(1+x^2));
```

$$(\%o5) \qquad \frac{y+x}{x^2+1}$$

Функция **xthru**

Функция **xthru** приводит выражение
к общему знаменателю, не раскрывая скобок
и не пытаясь факторизовать слагаемые

```
(%i6)    xthru( 1/(x+y)^10+1/(x+y)^12 );
```

$$(\%o6) \qquad \frac{(y+x)^2+1}{(y+x)^{12}}$$

```
(%i1) ((x+2)^20 - 2*y)/(x+y)^20 +
                  (x+y)^(-19) - x/(x+y)^20;
```

$$(\%o1) \qquad \frac{1}{(y+x)^{19}} + \frac{(x+2)^{20}-2\,y}{(y+x)^{20}} - \frac{x}{(y+x)^{20}}$$

```
(%i2)    xthru (%);
```

$$(\%o2) \qquad \frac{(x+2)^{20}-y}{(y+x)^{20}}$$

Функция **multthru**

Функция **`multthru`** умножает каждое слагаемое в сумме на множитель, причём при умножении скобки в выражении не раскрываются.
Она допускает два варианта синтаксиса:

- **`multthru(mult, sum);`**
- **`multthru(expr);`**

В последнем случае выражение **`expr`** включает и множитель, и сумму (см. (%i4) в примере ниже).
Пример функция **`multthru`**:

```
(%i1)    x/(x-y)^2 - 1/(x-y) - f(x)/(x-y)^3;
```

$$(\%o1) \qquad -\frac{1}{x-y}+\frac{x}{(x-y)^2}-\frac{f(x)}{(x-y)^3}$$

Продолжение примера функция **multthru**

```
(%i2)  multthru ((x-y)^3, %);
```

$$(\%o2) \qquad -(x-y)^2 + x\,(x-y) - f\,(x)$$

```
(%i3)  ((a+b)^10*s^2 + 2*a*b*s +
                    (a*b)^2)/(a*b*s^2);
```

$$(\%o3) \qquad \frac{(b+a)^{10}\,s^2 + 2\,a\,b\,s + a^2\,b^2}{a\,b\,s^2}$$

```
(%i4)  multthru (%);
```

$$(\%o4) \qquad \frac{2}{s} + \frac{a\,b}{s^2} + \frac{(b+a)^{10}}{a\,b}$$

Функции **assume** - ввод ограничений и **forget** - снятие ограничений

Функции **assume** - ввод ограничений и **forget** - снятие ограничений

позволяют управлять условиями выполнения (контекстом) прочих функций и операторов.

Пример:

```
(%i20)     sqrt(x^2);
(%o20)                |x|
(%i21)     assume (x<0);
(%o21)                [ x< 0 ]
(%i22)     sqrt(x^2);
(%o22)                -x
(%i23)     forget(x<0);
(%o23)                [ x< 0 ]
(%i24)     sqrt (x^2);|
(%o24)                |x|
```

Функции **divide** и **gcd**

Функция **divide** позволяет вычислить частное и остаток от деления одного многочлена на другой:

```
(%i1)     divide(x^3-2,x-1);
```

(%o1) $[x^2 + x + 1, -1]$

Первый элемент полученного списка — частное, второй — остаток от деления.

Функция **gcd** позволяет найти наибольший общий делитель многочленов:

```
(%i27) gcd((x-2)*(x^2-2*x+1), x-1);
```

(%o27) [x – 1]

Функция подстановки **subst**

Подстановки осуществляются функцией **`subst`**.
Вызов этой функции: **`subst(a,b,c)`**
- подставляем **`a`** вместо **`b`** в выражении **`c`**.

Пример:

```
(%i1)    subst (a, x+y, x + (x+y)^2 + y);
```

$$(\%o1) \qquad y + x + a^2$$

Преобразование рациональных выражений: функции **num** и **denom**

Для выделения числителя и знаменателя дробных выражений используются функции **num** и **denom**:

```
(%i1)     expr:(x^2+1)/(x^3-1);
```

$$(\%o1) \qquad \frac{x^2+1}{x^3-1}$$

```
(%i2)     num(expr);
```

$$(\%o2) \qquad x^2+1$$

```
(%i3)     denom(expr);
```

$$(\%o3) \qquad x^3-1$$

Функция **rat**

Функция **rat** приводит выражение к каноническому представлению. Она упрощает любое выражение, рассматривая его как дробнорациональную функцию, т.е. работает с операциями **"+", "-", "*", "/"** и с **возведением в целую степень**.
Синтаксис вызова:

- **rat(expr)**
- **rat(expr, x_1, ..., x_n)**

Переменные упорядочиваются в соответствии со списком **x_1, ..., x_n**. При этом вид ответа зависит от способа упорядочивания переменных.
Изначально переменные упорядочены в алфавитном порядке.

Пример использования **rat**:

Пример использования **rat**:

```
(%i1)    ((x - 2*y)^4/(x^2 - 4*y^2)^2 + 1)*
          (y + a)*(2*y + x) / (4*y^2 + x^2);
```

$$(\%o1)\quad \frac{(y+a)\,(2\,y+x)\left(\frac{(x-2\,y)^4}{\left(x^2-4\,y^2\right)^2}+1\right)}{4\,y^2+x^2}$$

```
(%i2)    rat(%);
```

$$(\%o2)\quad \frac{2\,y+2\,a}{2\,y+x}$$

После указания порядка использования переменных получаем следующее выражение:

```
(%i3)    rat(%o1,y,a,x);
```

$$(\%o3)\quad \frac{2\,a+2\,y}{x+2\,y}$$

Функция **ratvars** и флаг **ratfac**

Функция **`ratvars`** позволяет изменить
алфавитный порядок предпочтения переменных,
принятый по умолчанию.
Вызов
$$ratvars(z, y, x, w, v, u, t, s, r, q, p, o, n, m, l, k, j, i, h, g, f, e, d, c, b, a)$$
меняет порядок предпочтения в точности на обратный,

а вызов
$$ratvars(m, n, a, b)$$
упорядочивает переменные
m, n, a, b в порядке возрастания приоритета.

Флаг **ratfac** включает или выключает частичную факторизацию выражений при сведении их к стандартной форме (CRE). Изначально установлено значение **false**. Если установить значение **true**, то будет производиться *частичная факторизация*.

Функция **ratsimp и** флаг **ratsimpexpons**

Функция **ratsimp** приводит все части (в том числе аргументы функций) выражения, которое не является дробно-рациональной функцией, к каноническому представлению, производя упрощения, которые не выполняет функция **rat**. Повторный вызов функции в общем случае может изменить результат, т.е. не обязательно упрощение проводится до конца.

Применением упрощения к экспоненциальным выражениям управляет флаг **ratsimpexpons**, по умолчанию равный **false**, если его установить в **true**, упрощение *применяется и к показателям степени или экспоненты*

Примеры функции **ratsimp**

```
(%i1)  sin (x/(x^2 + x)) = exp ((log(x) + 1)^2 -
                                log(x)^2);
```

(%o1) $$sin\left(\frac{x}{x^2+x}\right) = e^{(log(x)+1)^2-log(x)^2}$$

```
(%i2)  ratsimp(%);
```

(%o2) $$sin\left(\frac{1}{x+1}\right) = e\,x^2$$

```
(%i3)  ((x - 1)^(3/2) - (x + 1)*sqrt(x - 1))/
                  sqrt((x - 1)*(x + 1));
```

(%o3) $$\frac{(x-1)^{\frac{3}{2}} - \sqrt{x-1}\,(x+1)}{\sqrt{(x-1)\,(x+1)}}$$

```
(%i4)  ratsimp(%);
```

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

```
(%i5)  x^(a + 1/a), ratsimpexpons: true;
```

(%o5) $$x^{\frac{a^2+1}{a}}$$

Функция **fullratsimp**

Функция `fullratsimp` вызывает функцию `ratsimp` до тех пор, пока выражение не перестанет меняться.

Пример:

```
(%i1) expr: (x^(a/2) + 1)^2*(x^(a/2) - 1)^2/
                                    (x^a - 1);
```

$$(\%o1) \qquad \frac{\left(x^{\frac{a}{2}}-1\right)^2\left(x^{\frac{a}{2}}+1\right)^2}{x^a-1}$$

```
(%i2)     ratsimp(expr);
```

$$(\%o2) \qquad \frac{x^{2\,a}-2\,x^a+1}{x^a-1}$$

```
(%i3)     fullratsimp(expr);
```

$$(\%o3) \qquad x^a-1$$

```
(%i4)     rat(expr);
```

$$(\%o4) \qquad \frac{\left(x^{\frac{a}{2}}\right)^4-2\left(x^{\frac{a}{2}}\right)^2+1}{x^a-1}$$

Пример влияния флага **ratsimpexpons**

Пример влияния флага **ratsimpexpons** на результат вычислений:

```
(%i1)    fullratsimp( exp((x^(a/2)-1)^2
                  *(x^(a/2)+1)^2 /(x^a-1) ) );
```

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

```
(%i2)    ratsimpexpons:true;
```

$$(\%o2) \qquad true$$

```
(%i3)    fullratsimp( exp((x^(a/2)-1)^2
                  *(x^(a/2)+1)^2 /(x^a-1) ) );
```

$$(\%o3) \qquad e^{x^a-1}$$

Функция **ratexpand**

Функция **ratexpand** раскрывает скобки в выражении.
Отличается от функции **expand** тем, что приводит выражение к канонической форме,
поэтому ответ может отличаться от результата применения функции **expand**:

```
(%i1)  ratexpand ((2*x - 3*y)^3);
```

$$(\%o1) \quad -27\,y^3 + 54\,x\,y^2 - 36\,x^2\,y + 8\,x^3$$

```
(%i2)  expr: (x - 1)/(x + 1)^2 + 1/(x - 1);
```

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

```
(%i3)  expand(expr);
```

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

```
(%i4)  ratexpand(expr);
```

$$(\%o4) \quad \frac{2\,x^2}{x^3+x^2-x-1} + \frac{2}{x^3+x^2-x-1}$$

Функция **ratsubst**

Подстановка в рациональных выражениях осуществляется функцией **ratsubst**.

Синтаксис вызова: **ratsubst (a,b,c)**

Выражение **a** подставляется вместо выражения **b** в выражении **c**.

b может быть суммой, произведением, степенью и т.п.

Пример использования **ratsubst**:

```
(%i1)    ratsubst(a, x*y^2, x^4*y^3 + x^4*y^8);
```

$$(\%o1) \qquad a\,x^3\,y + a^4$$

```
(%i2)    cos(x)^4 + cos(x)^3 + cos(x)^2 + cos(x) + 1;
```

$$(\%o2) \qquad cos\,(x)^4 + cos\,(x)^3 + cos\,(x)^2 + cos\,(x) + 1$$

```
(%i3)    ratsubst(1 -- sin(x)^2, cos(x)^2, %);
```

$$(\%o3) \quad sin\,(x)^4 - 3\,sin\,(x)^2 + cos\,(x)\left(2 - sin\,(x)^2\right) + 3$$

Преобразование тригонометрических выражений 1

Функция **trigexpand** раскладывает все тригонометрические и гиперболические функции от сумм и произведений в комбинации соответствующих функций единичных углов и аргументов.

Для усиления пользовательского контроля один вызов **trigexpand** выполняет упрощение на одном уровне.

Для управления вычислением имеется флаг **trigexpand**. Изначально флаг **trigexpand** установлен в **false**.

Если флаг **trigexpand** установить в **true**, то функция **trigexpand** будет работать до тех пор, пока выражение не перестанет меняться.

Пример функции **trigexpand**

```
(%i1)  x+sin(3*x)/sin(x),trigexpand=true,expand;
```

$$(\%o1) \qquad -sin(x)^2+3\,cos(x)^2+x$$

```
(%i2)  trigexpand(sin(10*x+y));
```

$$(\%o2) \qquad cos(10\,x)\;sin(y)+sin(10\,x)\;cos(y)$$

```
(%i3)  trigexpand(sin(3*x)+cos(4*x));
```

$$(\%o3)$$

$$sin(x)^4-sin(x)^3-6\,cos(x)^2\,sin(x)^2+3\,cos(x)^2\,sin(x)+cos(x)^4$$

Функция **trigreduce**

Функция **trigreduce(expr)** свёртывает все произведения тригонометрических и гиперболических функций в комбинации соответствующих функции от сумм.

Функция работает не до конца, так что повторный вызов может изменить выражение.

При вызове функции в формате **trigreduce(expr,x)** преобразования осуществляются относительно функций **x**.

Пример функции **trigreduce**

```
(%i8) trigreduce(cos(x)^4 + cos(x)^3 + cos(x)^2 +
                                        cos(x) + 1);
```

(%o8)

$$\frac{\cos(4x)+4\cos(2x)+3}{8}+\frac{\cos(3x)+3\cos(x)}{4}+\frac{\cos(2x)+1}{2}+\cos(x)+1$$

```
(%i9) trigreduce(-sin(x)^2+3*cos(x)^2+x);
```

(%o9)

$$\frac{\cos(2x)}{2}+3\left(\frac{\cos(2x)}{2}+\frac{1}{2}\right)+x-\frac{1}{2}$$

Функция **trigsimp**

Функция **trigsimp** упрощает тригонометрические и гиперболические выражения, применяя к ним правила

$$sin(x)^2 + cos(x)^2 = 1$$ и

$$cosh(x)^2 - sinh(x)^2 = 1.$$

Пример :

```
(%i1)   trigsimp(sin(x)^2+3*cos(x)^2);
```

$$(\%o1) \qquad 2\,cos\,(x)^2 + 1$$

```
(%i2)   trigsimp(sinh(x)^2+3*cosh(x)^2);
```

$$(\%o2) \qquad 4\,cosh\,(x)^2 - 1$$

Функция **trigrat**

Функция **trigrat.**
Синтаксис вызова **trigrat(expr)**
приводит заданное тригонометрическое выражение **expr**
к канонической упрощённой квазилинейной форме.

Это выражение рассматривается как рациональное,
содержащее функции sin, cos, tan, аргументы которых
линейные формы некоторых переменных и $\frac{\pi}{n}$ (n — целое).
Всегда, когда возможно, заданное выражение линеаризуется.

Пример:

```
(%i1) trigrat((1+sin(2*b)-cos(2*b))/sin(b));
```

$$(\%o1) \qquad 2\,sin\,(b) + 2\,cos\,(b)$$

Функция **radcan** упрощает выражения, содержащие *экспоненты, логарифмы и радикалы*, путём преобразования к форме, которая является канонической для широкого класса выражений.

Переменные в выражении упорядочиваются.

Эквивалентные выражения в этом классе не обязательно одинаковы, но их разность упрощается применением **radcan** до нуля.

Примеры функции **radcan**

```
(%i1)  (log(x+x^2)-log(x))^a/log(1+x)^(a/2);
```

$$(\%o1) \quad \frac{\left(log\left(x^2+x\right)-log\left(x\right)\right)^a}{log\left(x+1\right)^{\frac{a}{2}}}$$

```
(%i2)  radcan(%);
```

$$(\%o2) \quad log\left(x+1\right)^{\frac{a}{2}}$$

```
(%i10) (%e^x-1)/(1+%e^(x/2));
```

$$(\%o10) \quad \frac{e^x-1}{e^{\frac{x}{2}}+1}$$

```
(%i11) radcan(%);
```

$$(\%o11) \quad e^{\frac{x}{2}}-1$$

Функция **logcontract** 1

Функция **`logcontract(expr)`** рекурсивно сканирует выражение **`expr`**, преобразуя выражения вида

$$a1 * log(b1) + a2 * log(b2) + c$$

к форме

$$log(ratsimp(b1^{a1} * b2^{a2})) + c$$

Пример:

```
(%i1)    2*(a*log(x)+3*b*log(y));
```

$$(\%o1) \qquad 2\,(3\,b\,log\,(y) + a\,log\,(x))$$

```
(%i2)    logcontract(%);
```

$$(\%o2) \qquad b\,log\left(y^6\right) + a\,log\left(x^2\right)$$

Функция **logcontract 2**

Если объявить переменную n целой используя **declare**(n,integer);), то функция **logcontract** позволяет включить эту переменную в показатель степени:

```
(%i1)     declare(n,integer);
```

$$(\%o1) \qquad done$$

```
(%i2)     logcontract(3*a*n*log(x));
```

$$(\%o2) \qquad a\,log\left(x^{3\,n}\right)$$

Пользовательские функции 1

Для записи функции необходимо указать её название, а затем, в круглых скобках записать через запятую значения аргументов.

Если значением аргумента является список, то он заключается в квадратные скобки, а элементы списка также разделяются запятыми.

Пример:

sin(x);

integrate(**sin**(x),x,-5,5);

plot2d([**sin**(x)+3,**cos**(x)],[x,-%pi,%pi],[y,-5,5]);

Пользовательские функции 2

Пользователь может задать собственные функции.
Для этого сначала указывается название функции,
в скобках перечисляются названия аргументов,
после знаков **:= (двоеточие и равно**) следует
описание функции.
После задания пользовательская функция вызывается
точно так, как и встроенные функции **Maxima.**

Пример:

```
(%i44)   f(x):=x^2;
```

(%o44) $f(x) := x^2$

```
(%i45)   f(3 + 7);
```

(%o45) 100

Функция **define**

Не следует использовать для функций названия, зарезервированные для встроенных функций **Maxima.**

Для создания функций используется также встроенная функция **`define`**, которая позволяет преобразовать выражение в функцию.
Синтаксис вызова **`define`** довольно многообразен:

- $define(f(x_1, ..., x_n), expr)$
- $define(f[x_1, ..., x_n], expr)$
- $define(funmake(f, [x_1, ..., x_n]), expr)$
- $define(arraymake(f, [x_1, ..., x_n]), expr)$
- $define(ev(expr_1), expr_2)$

Варианты вызова функции **define**

Варианты вызова функции **define** различаются, какой именно объект создаётся:
ординарная функция (аргументы в круглых скобках)
или массив (аргументы в квадратных скобках).
Если первый аргумент — операторы
$funmake, arraymake$,
то функция создаётся и вычисляется (аналогично и **ev**).
Примеры:
Ординарная функция:

```
(%i1)     expr : cos(y) - sin(x);
```
$$(\%o1) \qquad cos\,(y) - sin\,(x)$$

```
(%i2)     define (F1 (x, y), expr);
```
$$(\%o2) \qquad F1\,(x, y) := cos\,(y) - sin\,(x)$$

```
(%i3)     factor(F1(a,b));
```
$$(\%o3) \qquad cos\,(b) - sin\,(a)$$

Примеры вызова функции **define**

Создание функции-массива:

```
(%i1)    define (G2 [x, y], x.y - y.x);
```

$$(\%o1) \qquad G2_{x,y} := x.y - y.x$$

Создание массива:

```
(%i2) define (arraymake (F, [u]), cos(u)+1);
```

$$(\%o2) \qquad F_u := cos\,(u) + 1$$

Использование функции ev для задания пользовательской функции:

```
(%i3) define (ev(foo(x,y)), sin(x)-cos(y));
```

$$(\%o3) \qquad foo\,(x, y) := sin\,(x) - cos\,(y)$$

Нахождение корней уравнений и систем алгебраических уравнений

Решение алгебраических уравнений и их систем осуществляется при помощи функции **solve**:

solve(expr,x)— решение одного уравнения **expr** относительно переменной **x**;

solve(expr) — решение уравнения с одной неизвестной и числовыми коэффициентами;

solve([eqn1, eqn2, . . ., eqnn],[x1, x2, . . ., xn]) — решение системы уравнений.

В качестве параметров:

- в первых квадратных скобках указывается список уравнений через запятую – Lexpr1=Rexpr2, Lexpr1=Rexpr2, . . .;
- во вторых квадратных скобках указывается — список переменных, через запятую.

Примеры **solve**

Решение одного уравнения с одним неизвестным

```
(%i7)    solve(x^2-5*x+4);
```

$$(\%o7) \qquad [x = 1, x = 4]$$

Решение одного уравнения в символьном виде:

```
(%i2)    solve([x-a/x+b], [x]);
```

$$(\%o2) \qquad [x = -\frac{\sqrt{b^2+4\,a}+b}{2}, x = \frac{\sqrt{b^2+4\,a}-b}{2}]$$

Решение системы уравнений в символьном виде:

```
(%i10)solve([x*y/(x+y)=a,x*z/(x+z)=b,y*z/(y+z)=c],
                                        [x,y,z]);
```

$$(\%o10)[[x = 0, y = 0, z = 0], [x = \frac{2abc}{(b+a)\,c-ab}, y = \frac{2abc}{(b-a)\,c+ab}, z = -\frac{2abc}{(b-a)\,c-ab}]]$$

В последнем примере решений несколько,
и **Maxima** выдаёт результат в виде списка.

Решение тригонометрических уравнений

Функция **solve** применима и для решения тригонометрических уравнений. При этом в случае множества решений у тригонометрических уравнений выдаётся соответствующее сообщение только и одно из решений.

Пример:

```
(%i13) solve([sin(x)=0], [x]);

solve: using arc-trig functions to get a solution.
Some solutions will be lost.
```

$$(\%o13) \qquad [x = 0]$$

Также Maxima позволяет находить комплексные корни:

```
(%i18)  solve([x^2+x+1], [x]);
```

$$(\%o18) \qquad [x = -\frac{\sqrt{3}\,i+1}{2}, x = \frac{\sqrt{3}\,i-1}{2}]$$

Построение графиков и поверхностей

Для вывода графиков на экран или на печать при помощи **Maxima** существуют несколько вариантов форматов и, соответственно, программ вывода графики, а именно:

- *openmath* (Tcl/Tk программа с графическим интерфейсом пользователя; элемент xMaxima)
- *gnuplot* (мощная утилита для построения графиков, обмен с Maxima — через канал)
- *mgnuplot* (Tk-интерфейс к *gnuplot* с рудиментарным графическим интерфейсом пользователя; включён в дистрибутив Maxima)
- *wxMaxima* (встроенные возможности *frontend*-а к Maxima)

Построение графиков и поверхностей

Все варианты интерфейса (кроме **wxMaxima**) для построения графиков используют две базовых функции:

plot2d построение двумерных графиков и

plot3d (построение трехмерных графиков).

При использовании **wxMaxima** кроме них используются ещё две аналогичные команды:

wxplot2d и **wxplot3d.**

Все команды позволяют либо вывести график на экран, либо (в зависимости от параметров функции) в файл.

Построение графика явной функции y = f(x)

График функции **y=f(x**) на отрезке **[a,b]** можно построить с помощью функции:

plot2d(f(x), [x,a,b] [, опции])

несколько графиков:

plot2d([f1(x), f2(x), …], [x,a,b] [, опции])

или

plot2d(f(x), [x,a,b], [y,c,d] [, опции])

несколько графиков:

plot2d([f1(x), f2(x), …], [x,a,b], [y,c,d] [, опции])

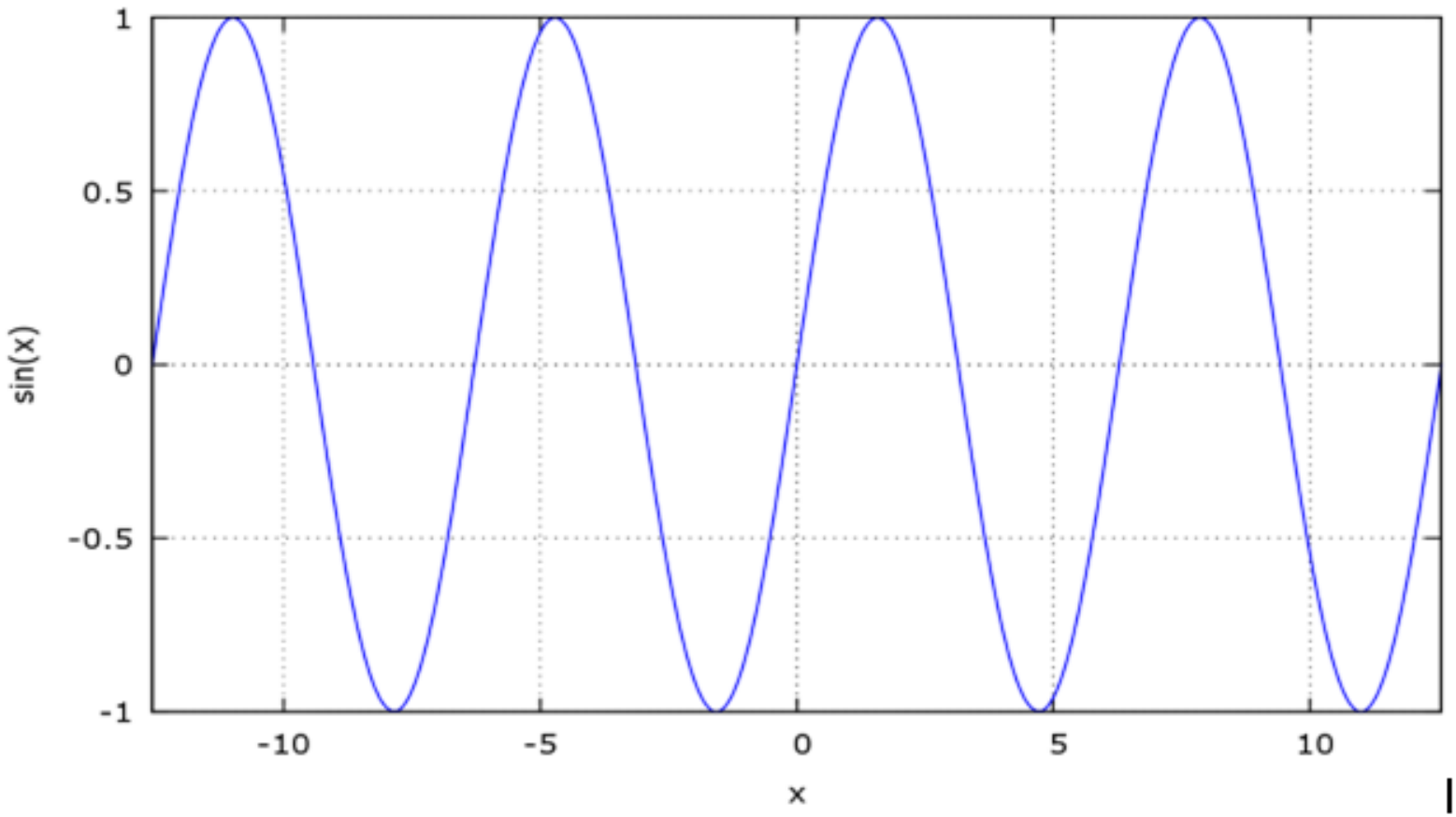
Опции не обязательны, однако, для изменения свойств графика их нужно задавать.

Параметр **[y,c,d]** можно не задавать, тогда высота графика выбирается по умолчанию.

Примеры построение графика явной функции

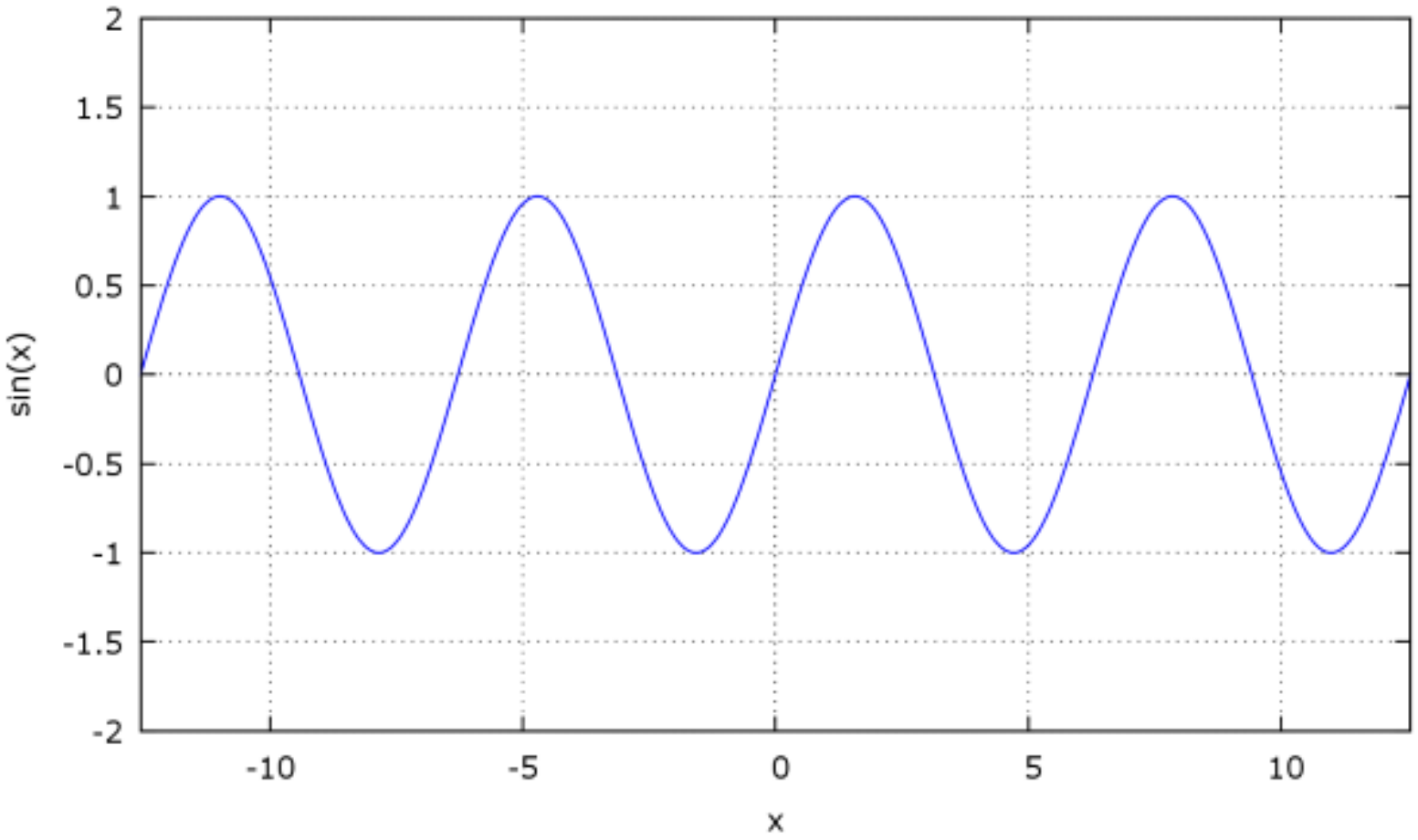
Построим график функции **y=sin(x)** на отрезке $[-4\pi, 4\pi]$.

```
(%i2) plot2d(sin(x), [x, -4*%pi, 4*%pi]);
```



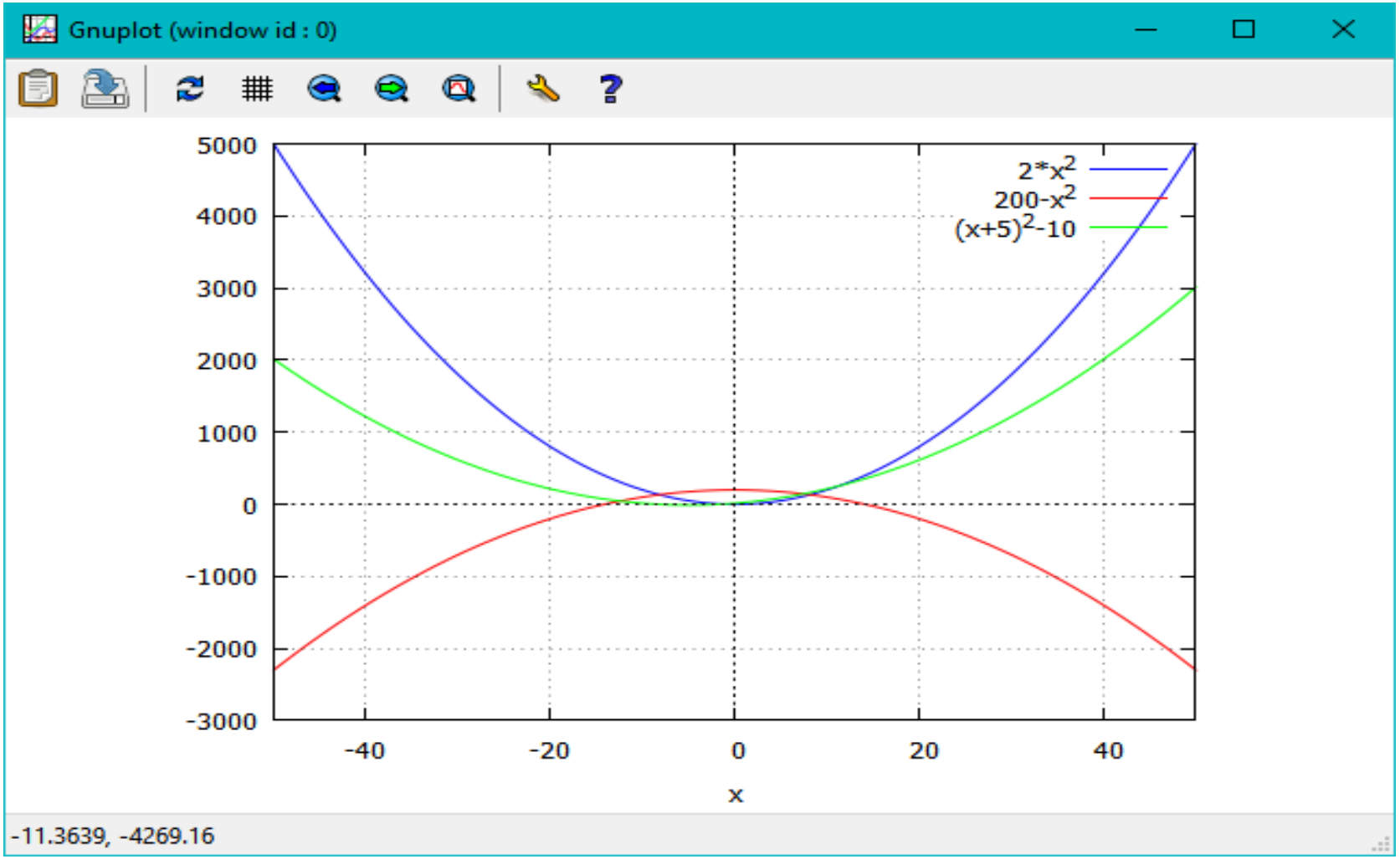
Примеры построение графика явной функции

```
(%i3) plot2d(sin(x), [x, -4*%pi, 4*%pi],[y,-2,2]);
```



Пример построения несколько графиков на одном рисунке

```
(%i7) plot2d([2*x^2,-x^2+200,(x+5)^2-10],[x,-50,50]);
```



Построение графиков функций, заданных параметрически

Для построения графиков функций, заданных параметрически, используется опция **parametric**. Для построения графика указывается область изменения параметра.

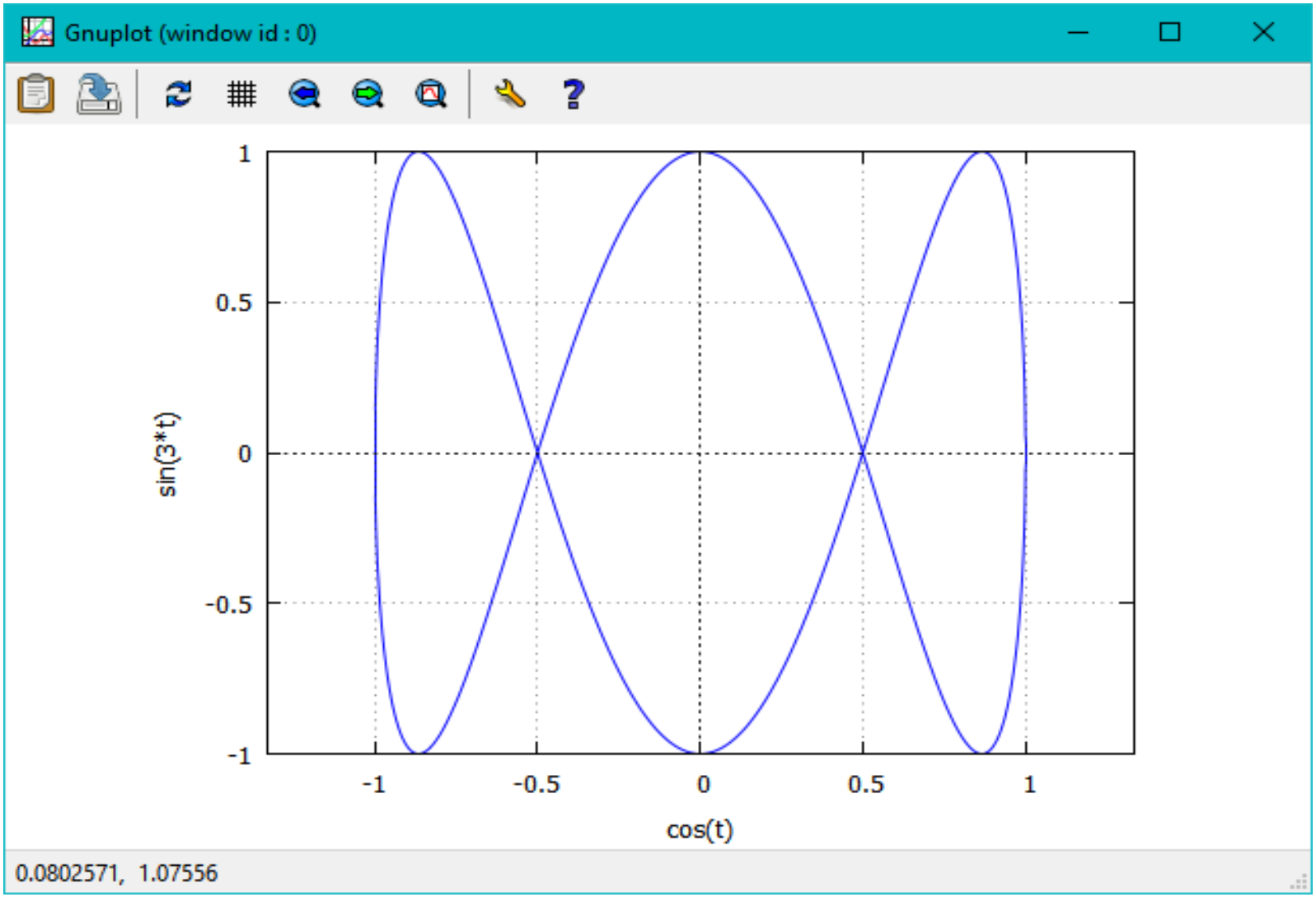
Команда построения графика:

```
plot2d([parametric,cos(t),sin(3*t),
   [t,-%pi,%pi], [nticks,80]],[x,-4/3,4/3]);
```

Опция **nticks** указывает число точек, по которым проводится кривая.

Построение графиков функций, заданных параметрически

plot2d([parametric,cos(t),sin(3*t),[t,-%pi,%pi], [nticks,80]],[x,-4/3,4/3]);



Некоторые опции построения графиков.

Опции указываются в виде аргументов функции **plot2d** в квадратных скобках.

Возможна установка легенды, меток на осях, цвета и стиля графика.

Применение нескольких опций характеризует следующий пример:

```
plot2d([[discrete,xy], 2*%pi*sqrt(l/980)],
  [l,0,50], [style, [points,5,2,6], [lines,1,1]],
  [legend, "эксперимент", "теория"],
  [xlabel,"длина маятника (см)"],
  [ylabel,"период (сек)"]);
```

Некоторые опции построения графиков

В данном примере в одних осях строятся два графика.

Первый график [**discrete**,xy] строится в виде точек по массиву xy указанием стиля **points**.

Второй график строится по уравнению функции 2*%pi*sqrt(l/980)] с указанием стиля **lines**.

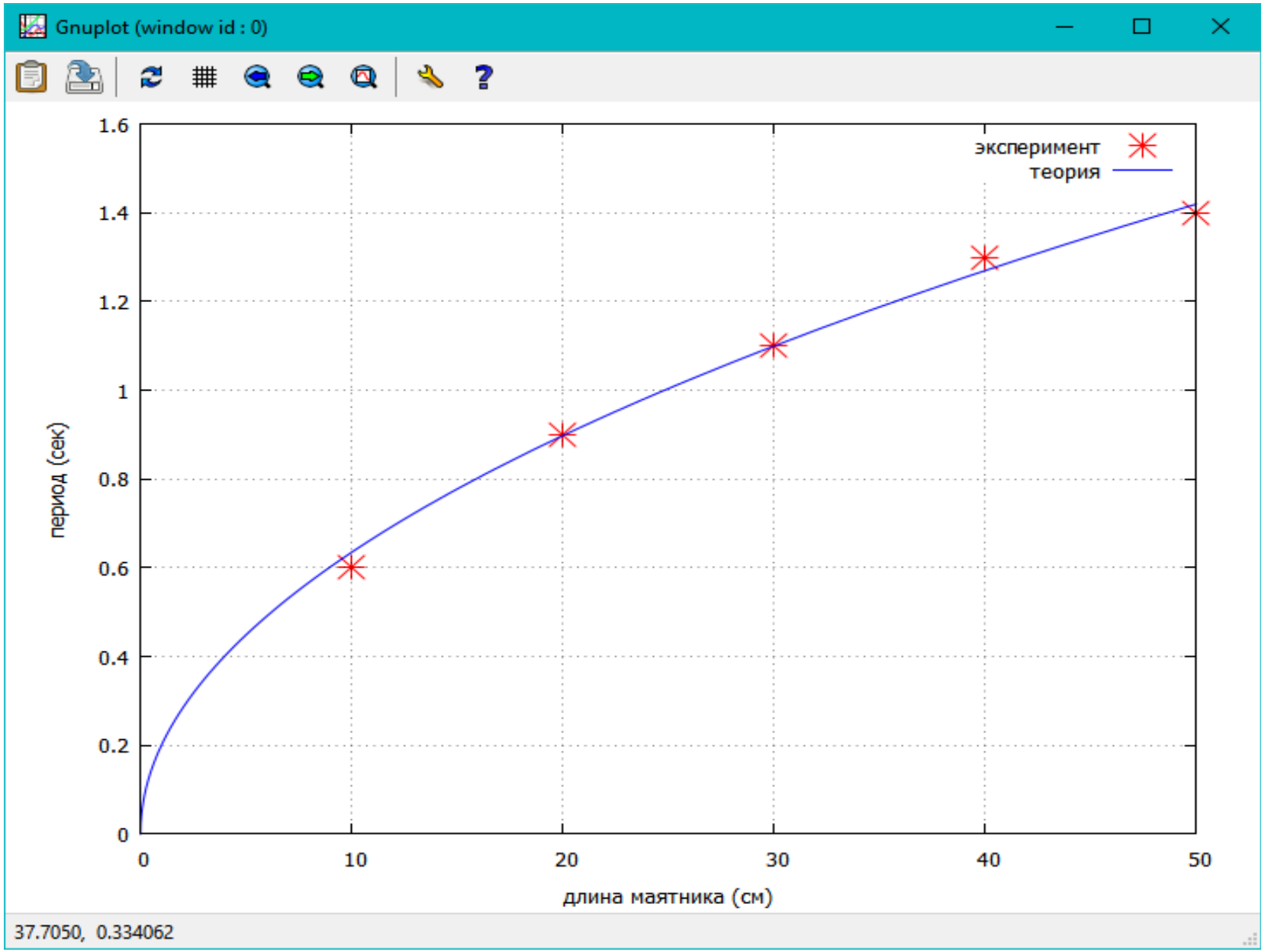
Опция **legend** указывает подписи кривых.

Опции **xlabel** и **ylabel** — подписи осей.

Формирование массива для построения графика осуществляется следующим образом:

xy:[[10,.6], [20,.9], [30,1.1], [40,1.3], [50,1.4]];

Некоторые опции построения графиков



Комбинированные графики

Можно комбинировать в одних осях графики кривых различного типа: функции **y=f(x)** или параметрические

$$\begin{cases} x = \varphi(t), \\ y = \psi(t). \end{cases}$$

например:

```
plot2d ([x^3+2, [parametric, cos(t),
                                sin(t), [t, -5, 5],
                                    [nticks,80]]],
          [x, -2, 2], [xlabel, "x"], [ylabel, "y"],
           [style, [linespoints,3,2], [lines,3,1]],
                                [gnuplot_term, ps],
                      [gnuplot_out_file, "test.eps"]);
```

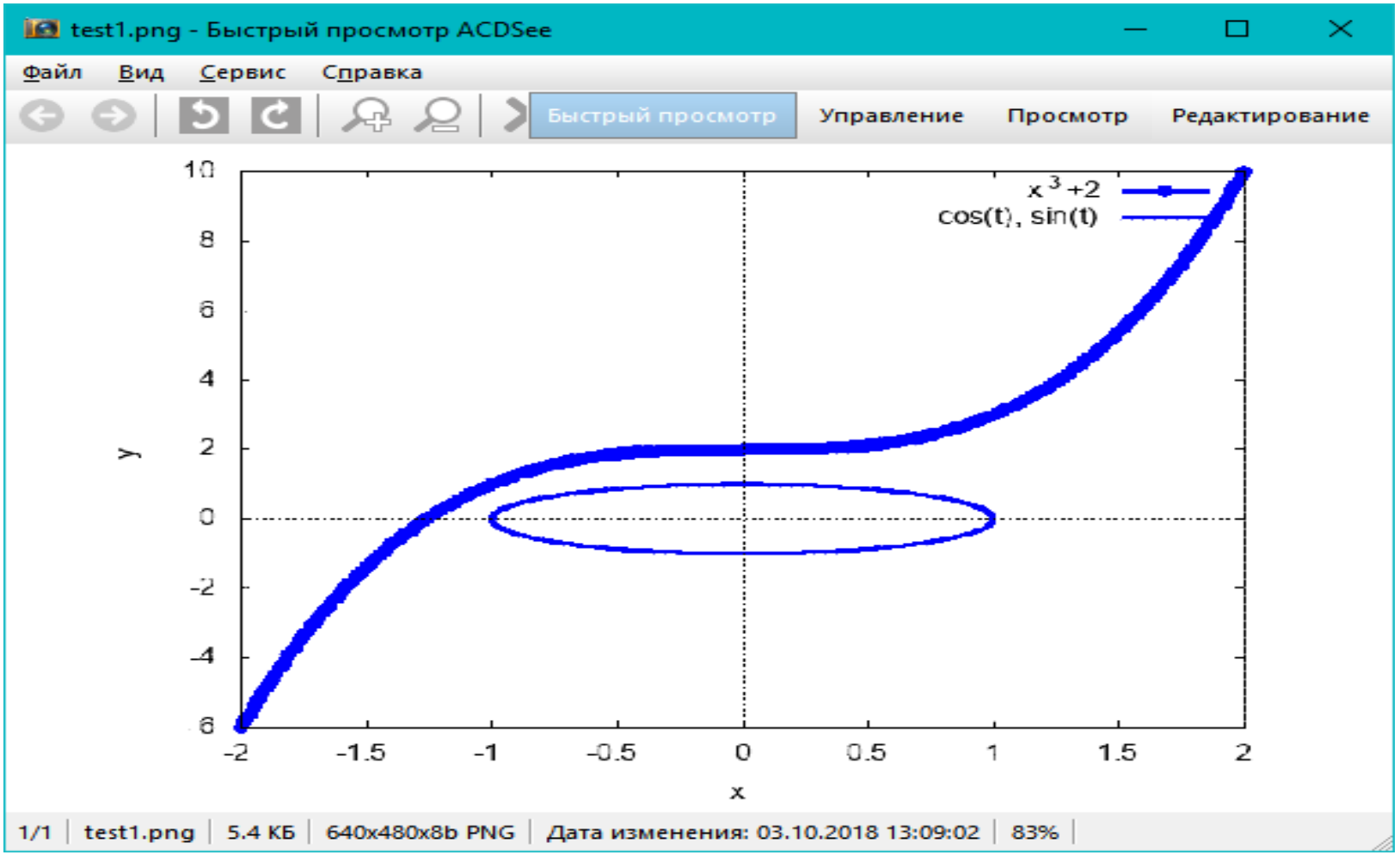
Некоторые опции

Опции
[**gnuplot_term**, **ps**],
[**gnuplot_out_file**, "test.eps"]
указывают, что графическая иллюстрация
выводится в файл "test.eps" в формате
postscript (бэкенд для вывода графиков — **gnuplot**).
Опции [**style**,[**linespoints**,3,2],**lines**,3,1]]
позволяют указать стиль линий на графике -
линия с точками или сплошная линия.

Для вывода результатов в формат **png**
можно использовать опции:
[**gnuplot_term**, **png**, **size**, 400,400],
[**gnuplot_out_file**, "test1.png"]
указание размеров **size**,400,400 в общем случае необязательно.

Совмещение на одном графике параметрической и заданной явно кривых (файл **"test1.png"**);

```
plot2d ([x^3+2, [parametric, cos(t),sin(t), [t, -5, 5], [nticks,80]]], [x, -2, 2],
        [xlabel, "x"], [ylabel, "y"], [style, [linespoints,3,2], [lines,3,1]],
        [gnuplot_term, png, size, 400,400], [gnuplot_out_file, "test1.png"]);
```



Построение кривых в полярной системе координат

Для построения графика в полярных координатах нужно задать изменение значений полярного радиуса и полярного угла.

Пусть **r=r(f) (a<= f <=b)** — зависимость полярного радиуса **r** от полярного угла **f**.

Тогда график этой функции в полярных координатах можно построить, задав у функции **plot2d** опцию [**gnuplot_postamble**, "**set polar**; **set zeroaxis**"].

Данная опция будет действовать лишь при условии, что выбран формат графика **gnuplot**.

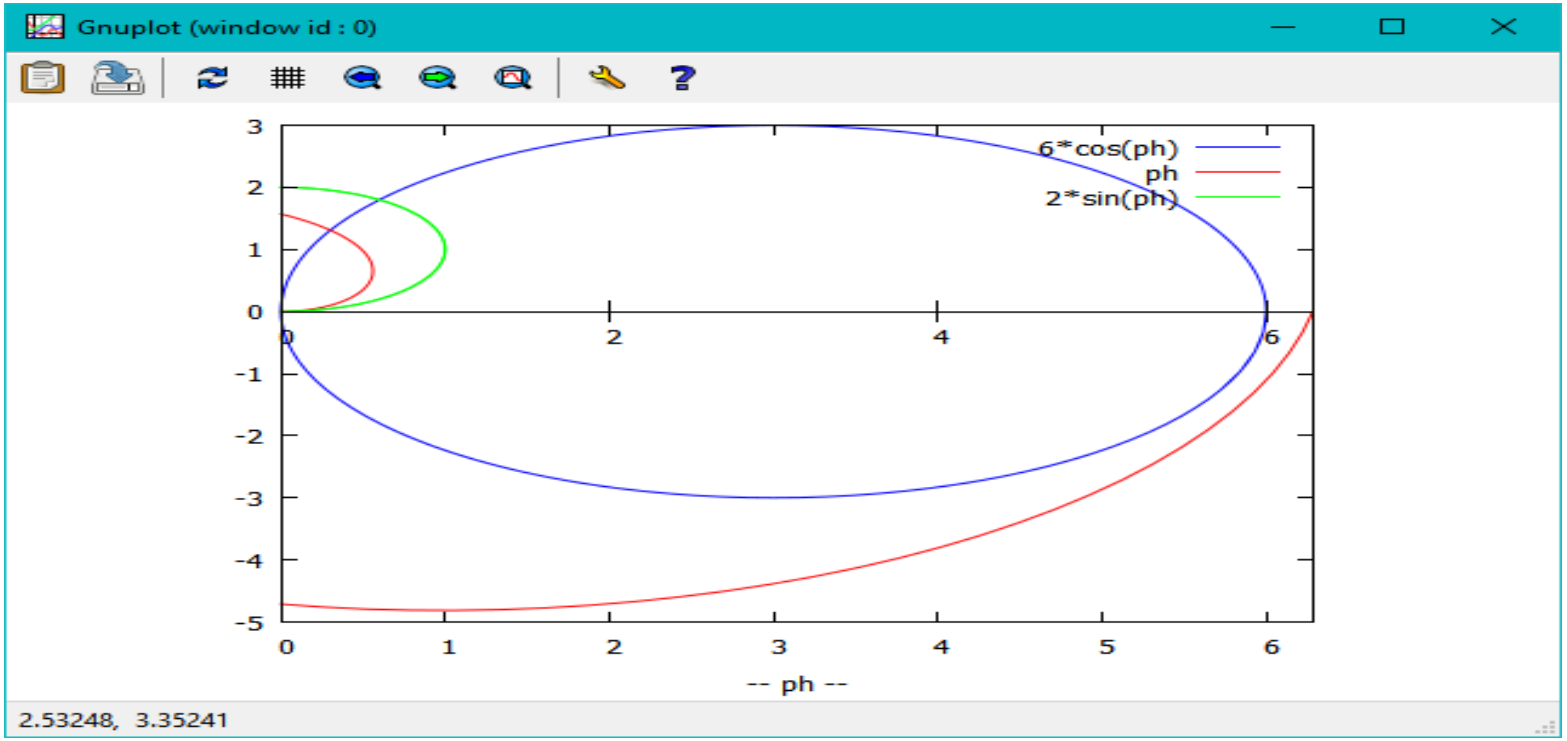
Пример построение кривых в полярной системе координат

Пример: построить в полярных координатах графики трёх функций

$$r = 6cos(\varphi), r = \varphi, r = 2sin(\varphi), 0 \leq \varphi \leq 2\pi$$

Для создания графика используем команду:

```
plot2d( [6*cos(ph),ph,2*sin(ph)],[ph,0,4*%pi],
                    [gnuplot_postamble,
                "set polar; set zeroaxis"],
                    [xlabel,"-- ph --"],
                  [plot_format, gnuplot]);
```



Построение трёхмерных графиков

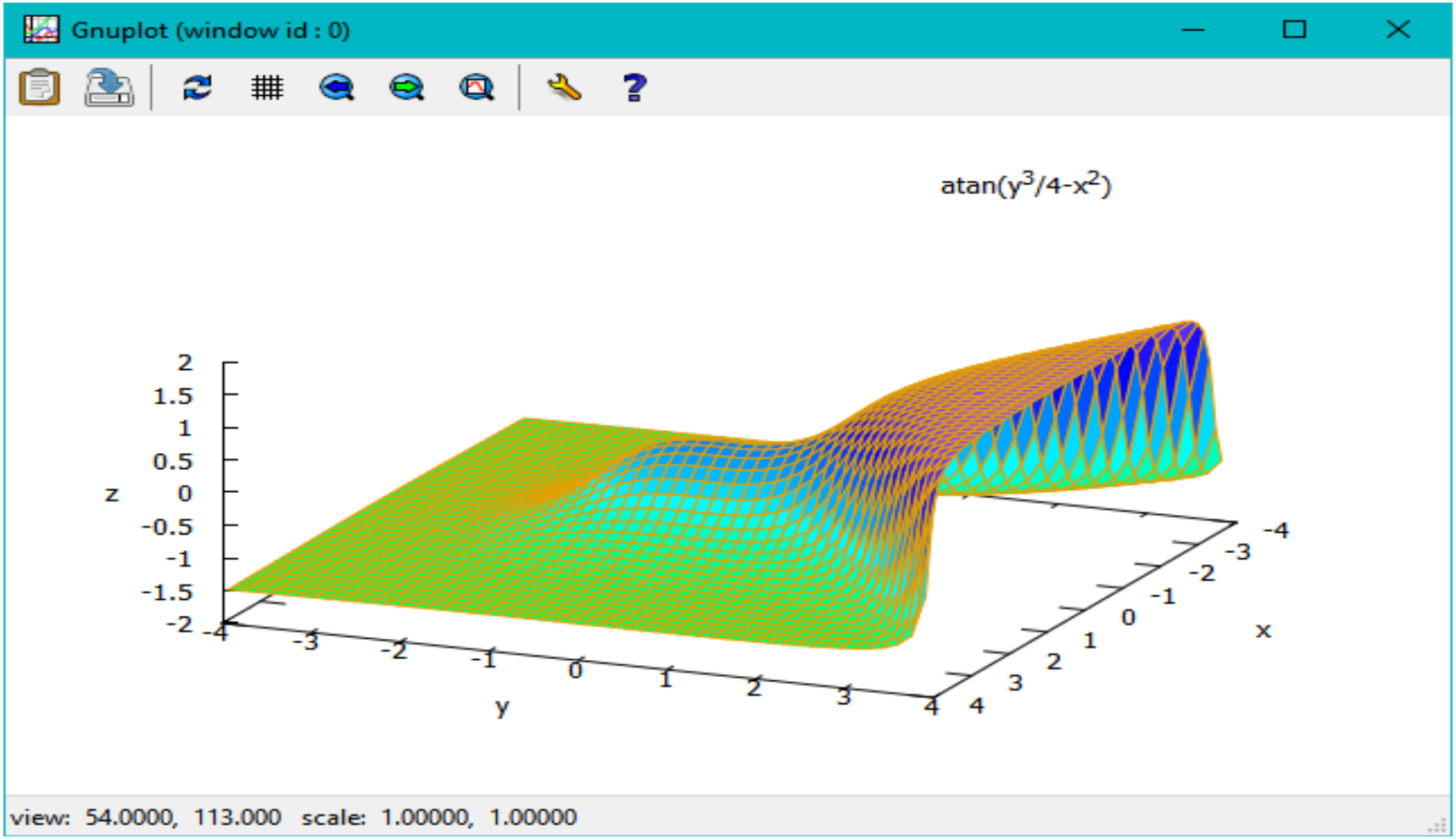
Основная команда для построения трёхмерных графиков — **plot3d**.

Рассмотрим технологию построения графиков с использованием интерфейса **plot3d**.

Поверхность функции в цветном изображении строится с использованием опции **gnuplot_pm3d:**

```
plot3d(atan(-x^2 + y^3/4), [x, -4, 4], [y, -4, 4],
    [grid, 50, 50], [gnuplot_pm3d,true]);
```

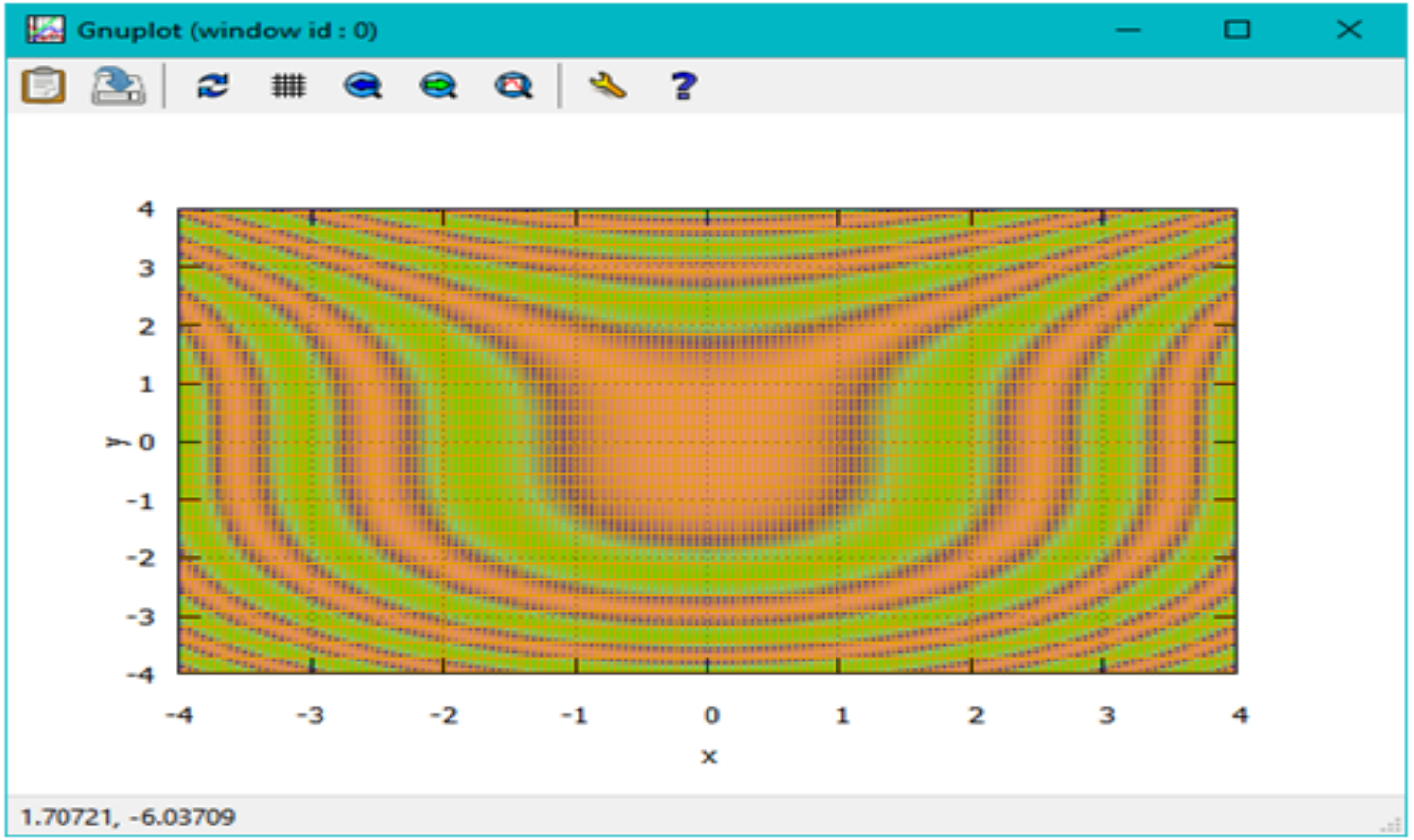
Пример построения трёхмерных графиков



Изображение линий уровня функции

Можно построить и изображение линий уровня функции.

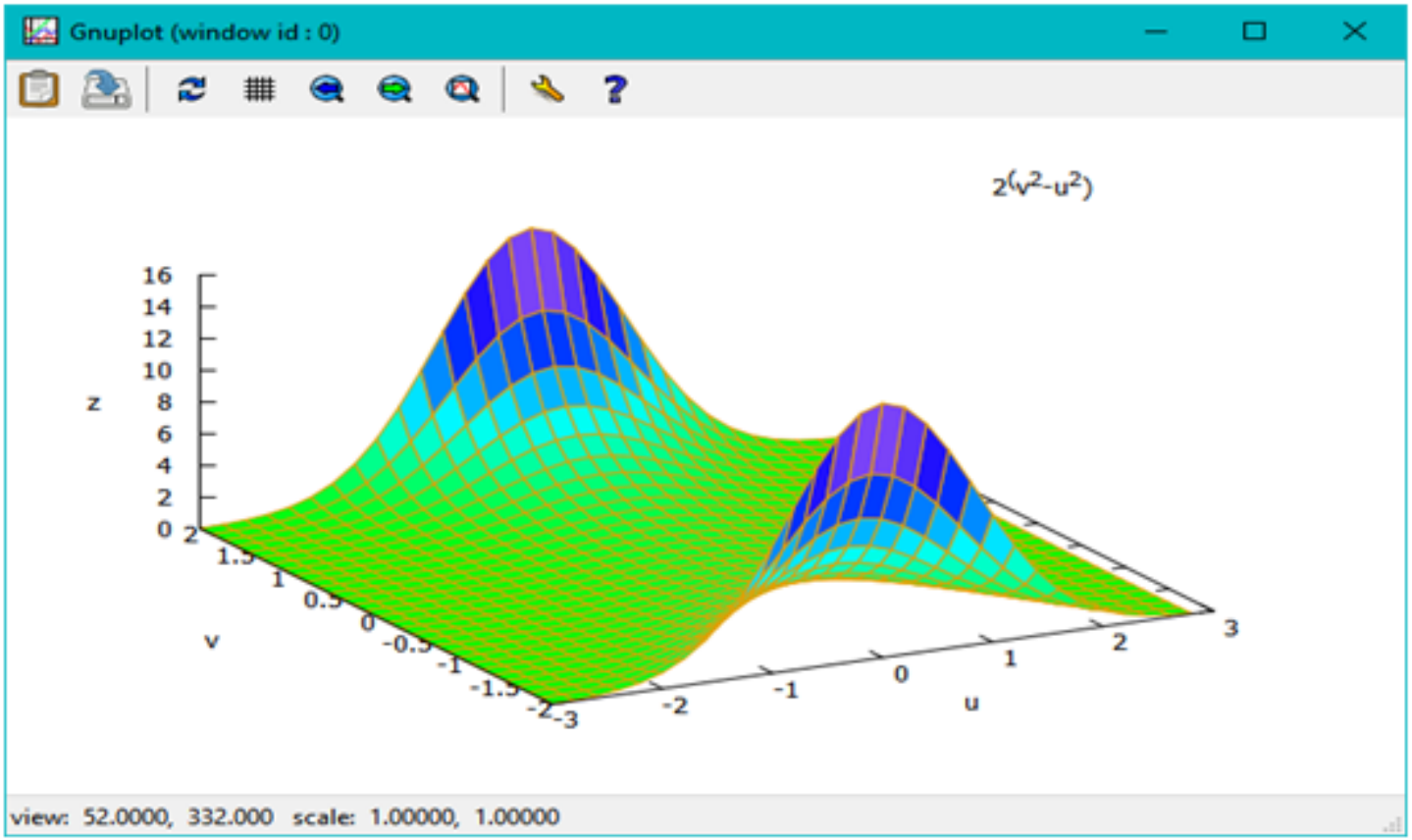
```
plot3d(cos(-x^2+y^3/4), [x,-4,4],[y, -4, 4],
    [gnuplot_preamble,"set view map"],
    [gnuplot_pm3d, true], [grid, 150, 150]);
```



Простой график функции двух переменных

Используя стандартный формат функции
plot3d, можно строить график функции двух переменных:

```
plot3d(2^(-u^2 + v^2),[u, -3, 3], [v, -2, 2]);
```



Вывод графика в файл

Для вывода графика в файл можно использовать опции **gnuplot -**

установить терминал **gnuplot**

и имя файла результата.

Необходимая команда:

plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2],
[gnuplot_term,ps],
[gnuplot_out_file,"plot33.eps"]);

Пример - вывод графики в формате **openmath**

Смена формата графики также возможна за счёт использования опций **plot3d**.

Пример - вывод графики в формате **openmath**:

```
plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2],
    [plot_format, openmath]);
```

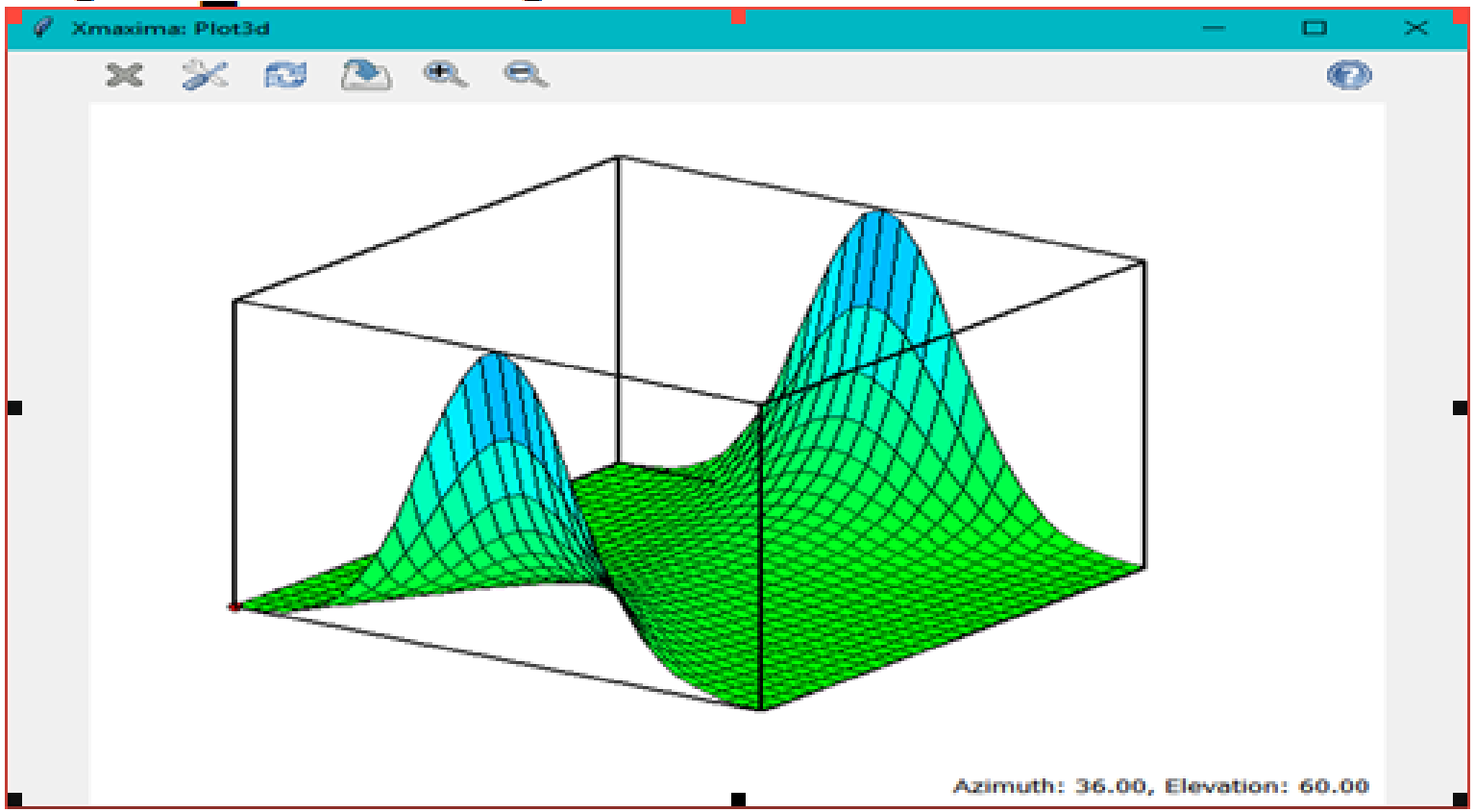


График параметрически заданной функции

График параметрически заданной функции строится так:

plot3d([выражение1, выражение2, выражение3],

[переменная1, начало, конец],

[переменная2, начало, конец]);

Где

выражения отвечают, по порядку

x(u, v), y(u, v), z(u, v)

Пример графика параметрически заданной функции

```
plot3d([cos(u),sin(u), v],[u, -%pi, %pi],[v,0,10]);
```

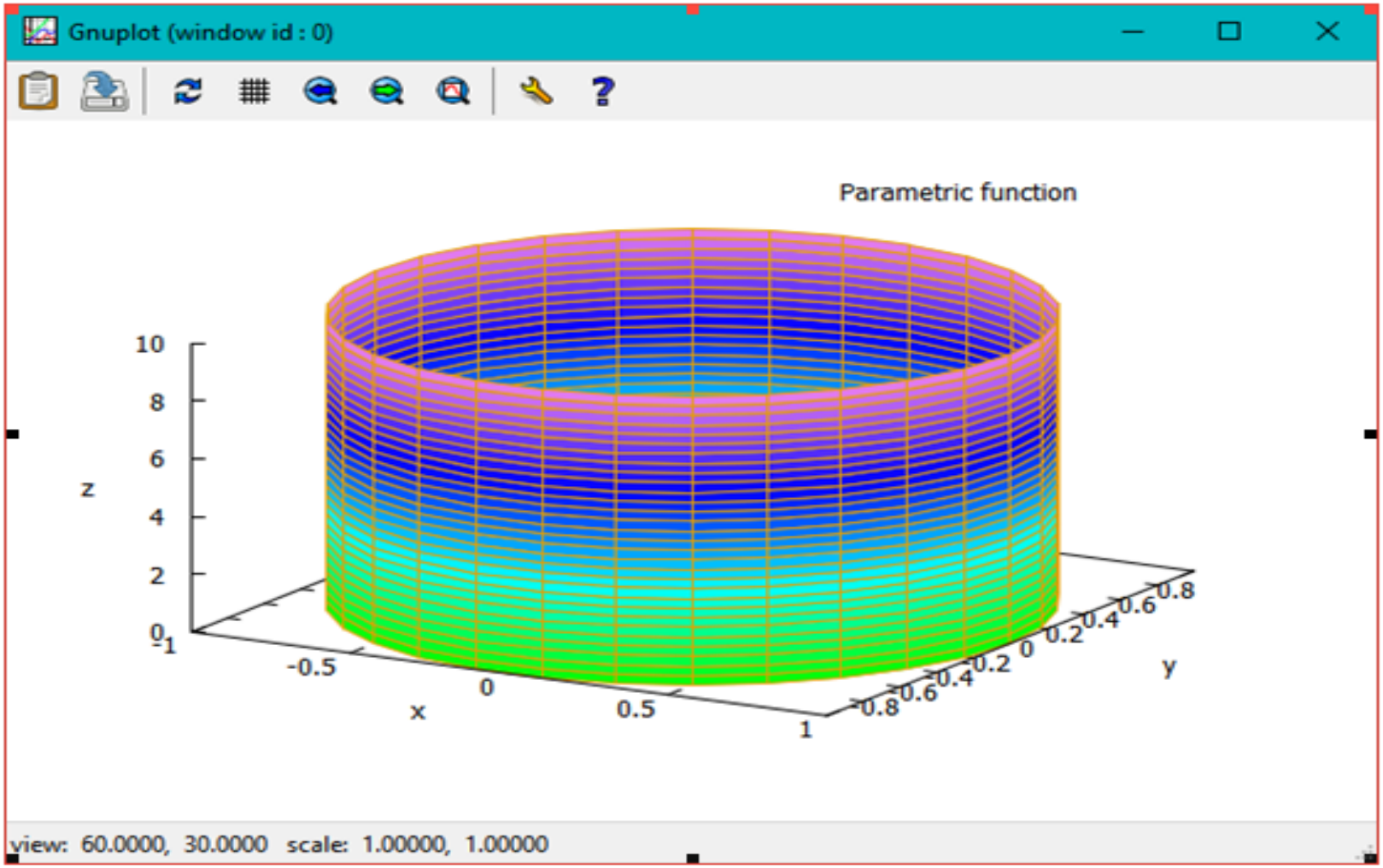


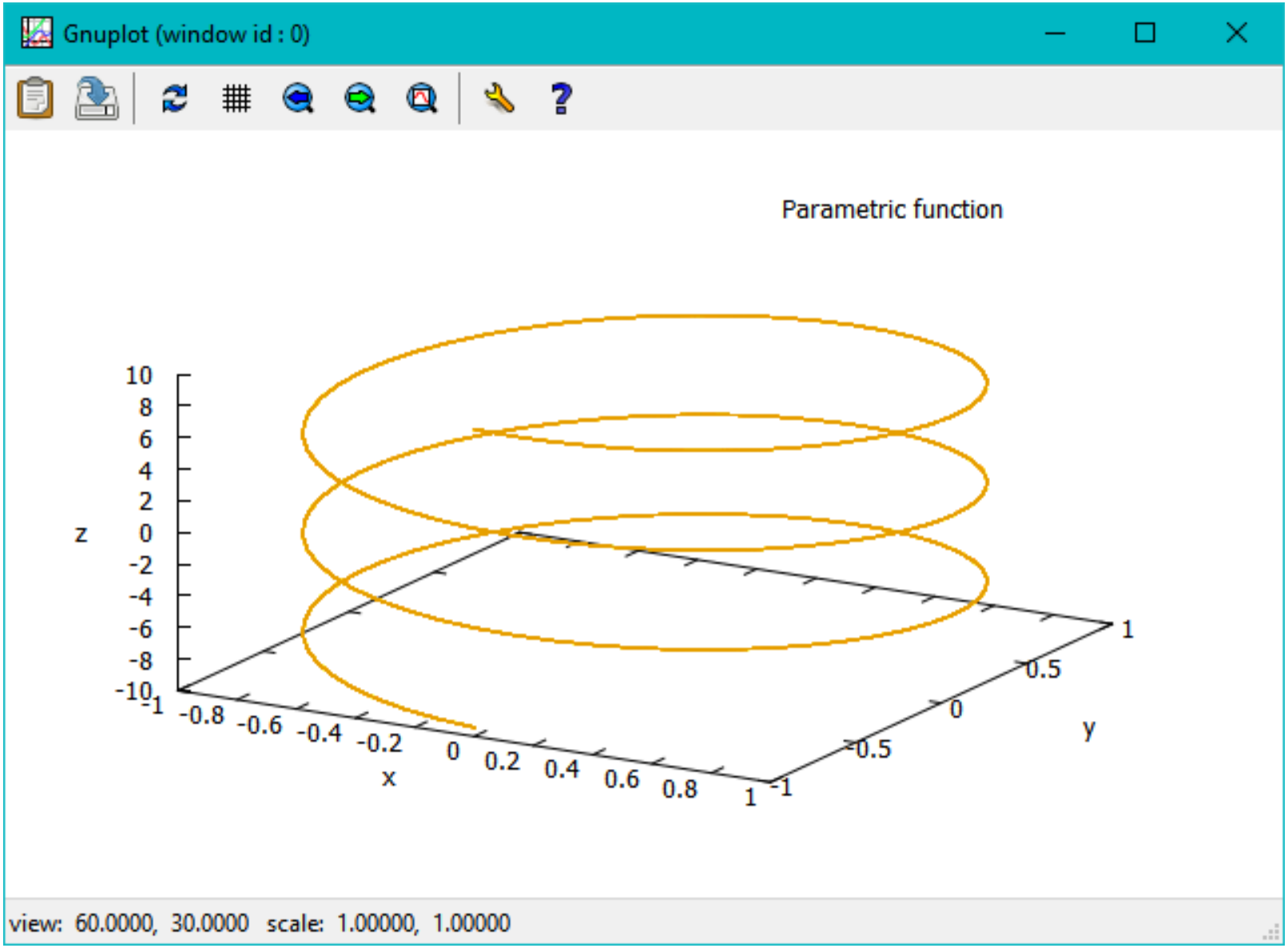
График пространственной кривой

С помощью параметрической формы можно строить и пространственные кривые.

Для этого просто нужно задать второй, фиктивный, параметр, чтобы Maxima «не ругалась» на неправильный синтаксис вызова функции:

```
plot3d([sin(t), cos(t), t], [t,-3*%pi, 3*%pi],
                    [v,0,1], [grid, 150, 150]);
```

Пример графика пространственной кривой



Draw графика - Сцены

gr2d (*graphic option*, ..., *graphic object*, ...)

Функция gr2d создает объект, описывающий 2D-сцену.

Аргументы - это графические параметры, графические объекты или списки, содержащие как графические параметры, так и объекты(*graphic options*, *graphic objects*). Эта сцена интерпретируется последовательно: графические параметры влияют на те графические объекты, которые расположены справа. Некоторые графические параметры влияют на глобальное появление сцены.

Список графических объектов (*graphic objects*), доступных для сцен в двух измерениях:

`bars, ellipse, explicit, image, implicit, label, parametric, points, polar, polygon, quadrilateral, rectangle, triangle, vector, and geomap (this one defined in package worldmap`).

Draw графика - Сцены

gr3d (*graphic option*, ..., *graphic object*, ...)

Функция gr3d создает объект, описывающий 3D-сцену.

Аргументы - это графические параметры, графические объекты или списки, содержащие как графические параметры, так и объекты. Эта сцена интерпретируется последовательно: графические параметры влияют на те графические объекты, которые расположены справа. Некоторые графические параметры влияют на глобальное появление сцены.

Список графических объектов, доступных для сцен в трех измерениях:

`cylindrical, elevation_grid, explicit, implicit, label, mesh, parametric, parametric_surface, points, quadrilateral, spherical, triangle, tube,vector, and geomap.`

Чтобы использовать эти объекты необходимо вначале загрузить пакет draw - **load(draw**).

draw (gr2d, ..., gr3d, ..., опции, ...)

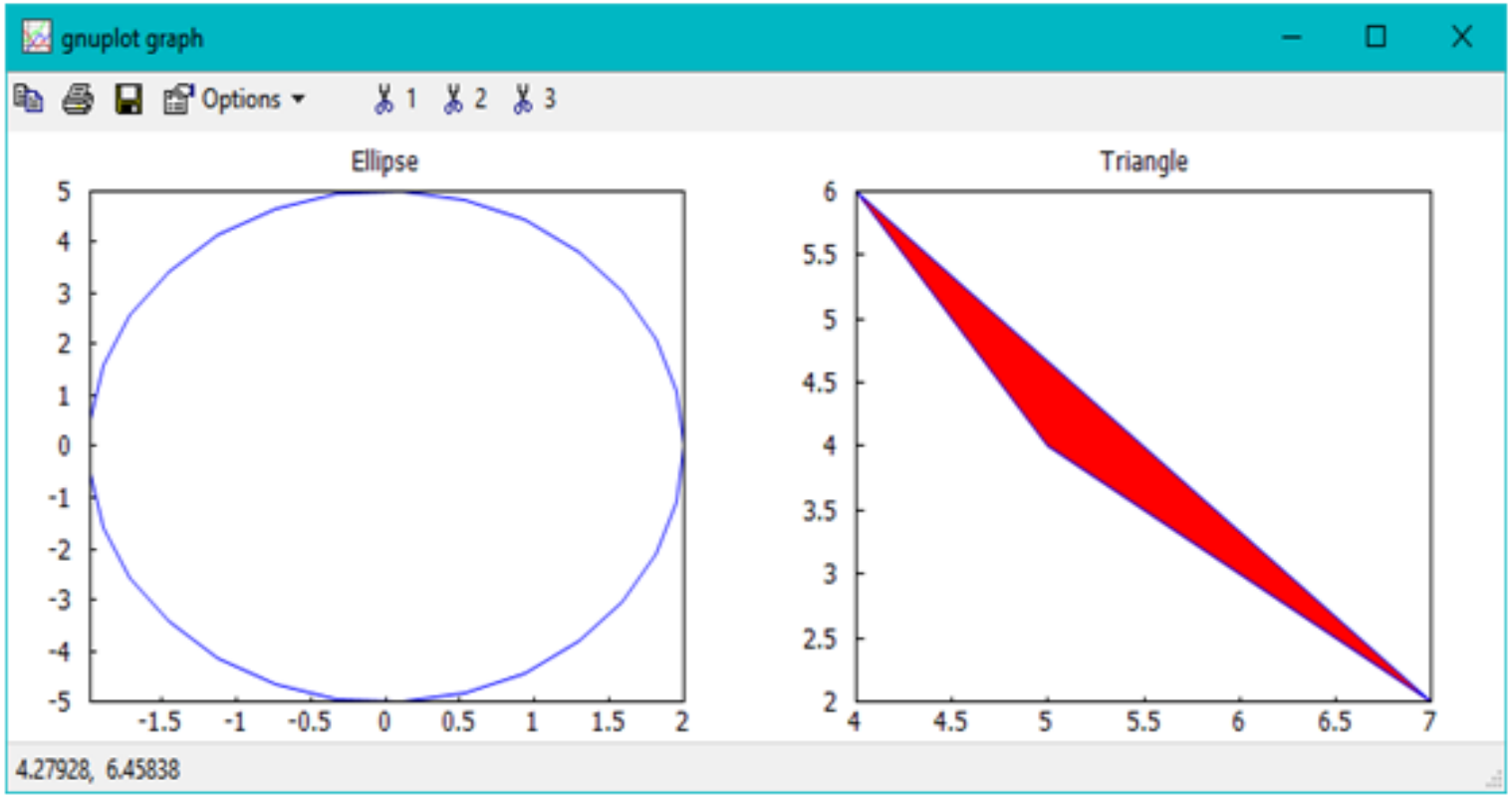
Прорисовка заданной сюжетной серии сцен. Аргументы представляют собой объекты **gr2d** и/или **gr3d** вместе с некоторыми опциями или списками сцен и опций. По умолчанию сцены объединяются в один столбец.

Функция **draw** принимает следующие глобальные параметры: ***терминал, столбцы, размеры, имя файла и задержку***.

Функции **draw2d** и **draw3d** - используются, когда требуется только одна сцена, в двух или трех измерениях соответственно.

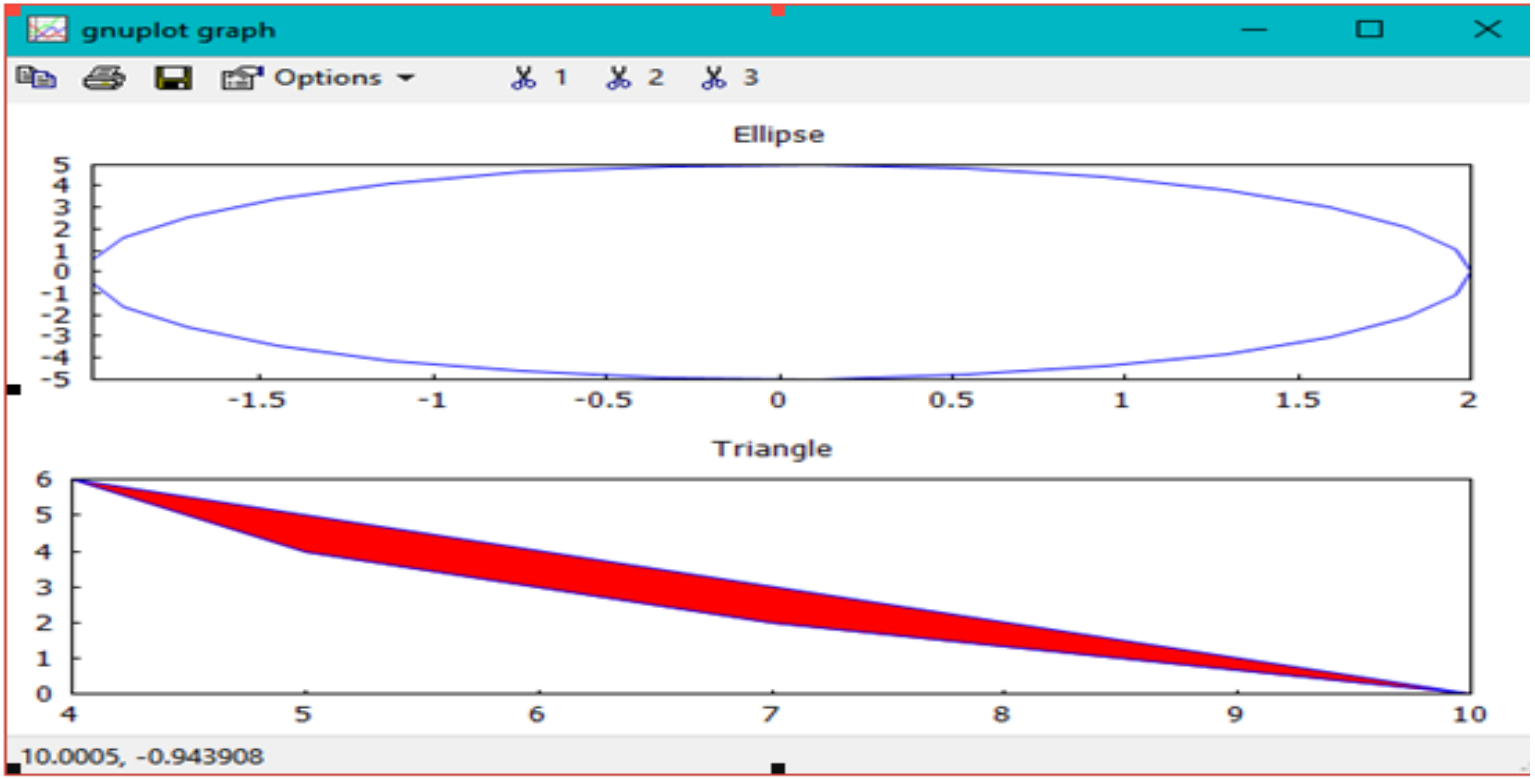
Основные функции и примеры

```
kill(all);
load(draw)$
scene1: gr2d(title="Ellipse",
                    nticks=30,
     parametric(2*cos(t),5*sin(t),t,0,2*%pi));
scene2: gr2d(title="Triangle",
               polygon([4,5,7],[6,4,2]));
draw(scene1, scene2, columns = 2);
```



Основные функции и примеры

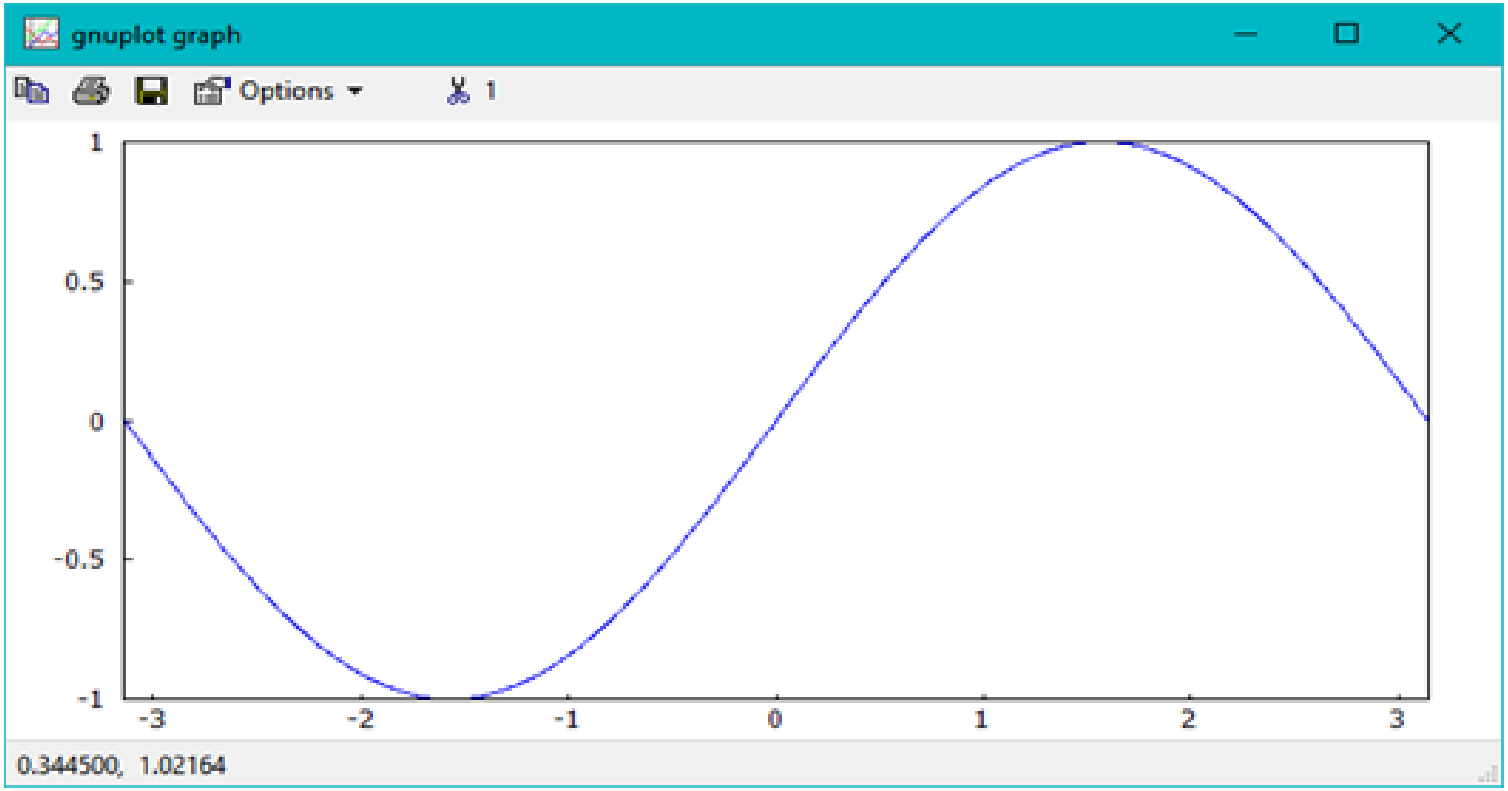
```
kill(all);
load(draw)$
scene1: gr2d(title="Ellipse",
   nticks=30,
parametric(2*cos(t),5*sin(t),t,0,2*%pi));
scene2: gr2d(title="Triangle",
                  polygon([4,5,7, 10 ],[6,4,2, 0]))
draw(scene1, scene2);
```



Построение простой (явной) функции при помощи draw2d

```
draw2d(nticks=300, explicit(sin(x),x,-%pi,%pi));
```

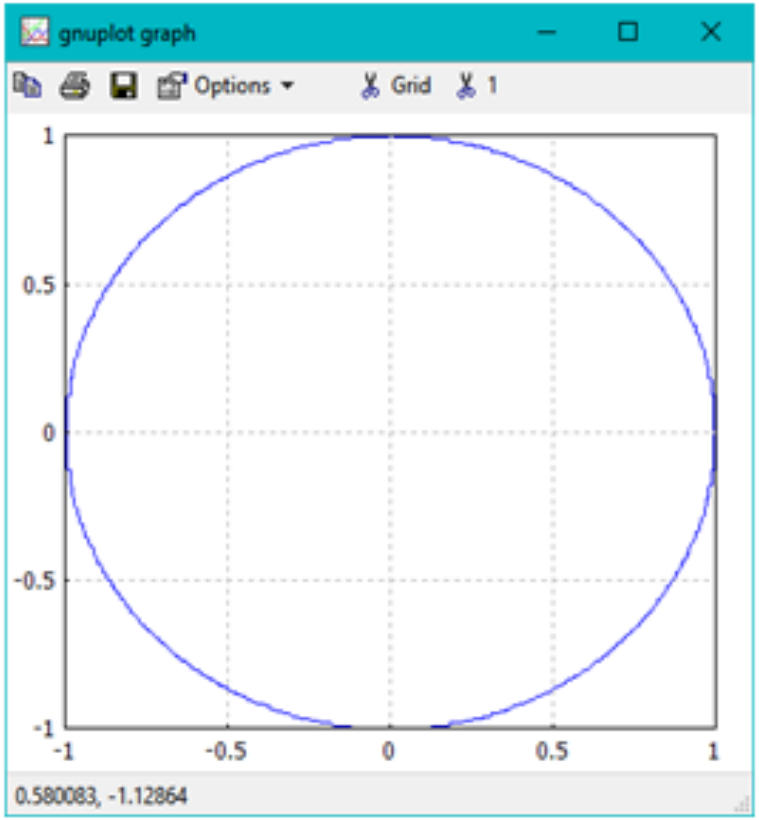
Здесь функция `f(x)=sin(x)` конструируется при помощи вызова **`explicit`**`(sin(x),x,-%pi,%pi)`.



Построение неявной функции - окружности

Уравнение функции задается через функцию **implicit** :

```
draw2d(nticks=300, grid=true,
                implicit(x^2+y^2=1,x,-1,1,y,-1,1));
```



Построения графика функции в полярных координатах

```
draw2d(nticks=300,  grid=true, polar(t,t,0,4*%pi));
```

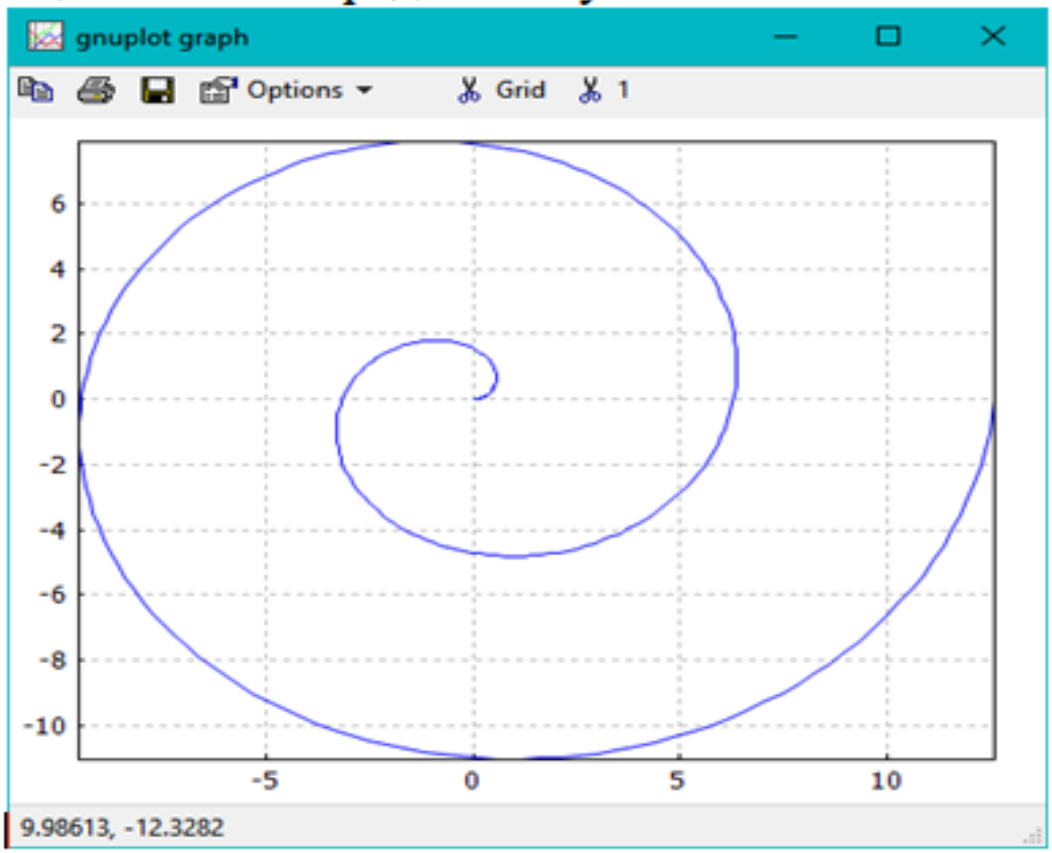
Здесь функция задается, как функция радиуса `r(t)`.

Функция конструируется в виде вызова **`polar`**`(t,t,0,4*%pi)`

первый параметр - это функция `r(t)`,

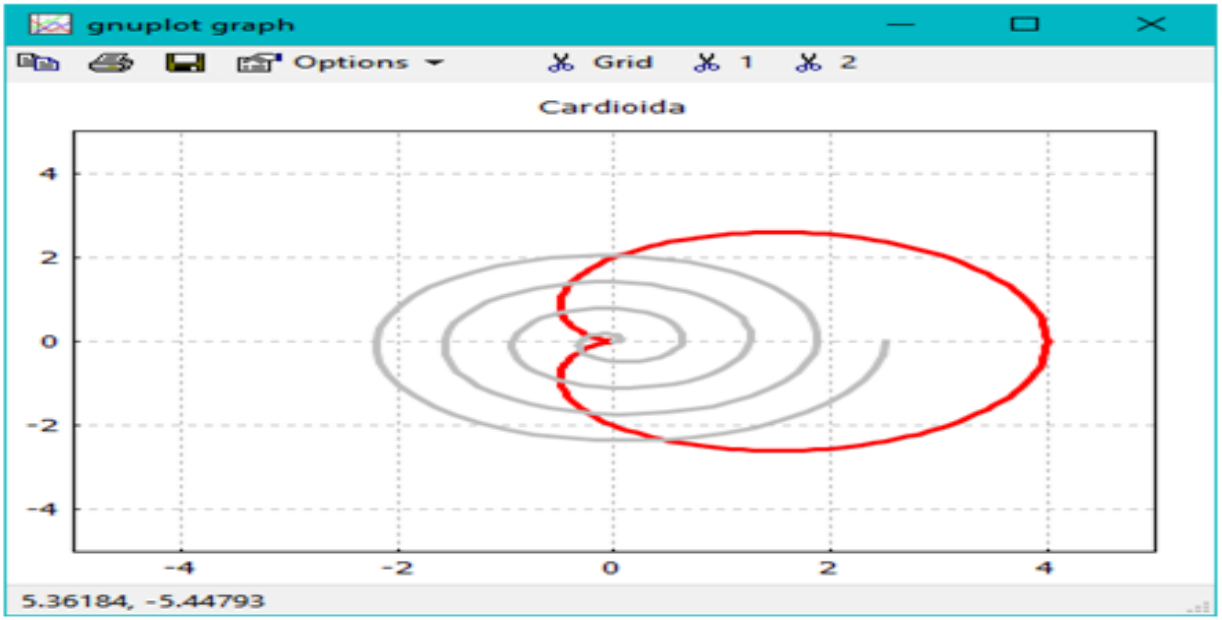
второй параметр - это обозначение полярного угла `t`,

далее границы области определения угла.



Пример - рисовать несколько графиков

```
load(draw);
draw2d(user_preamble="set grid",
    nticks = 200,
    xrange = [-5,5],
    yrange =[-5,5],
    color = red,
    line_width = 3,
    title = "Cardioida",
    polar ( 2 * (1+cos(theta)),theta, 0, 2*%pi),
    color=grey,
    polar ( 0.1*theta,theta, 0, 8*%pi)
 );
```

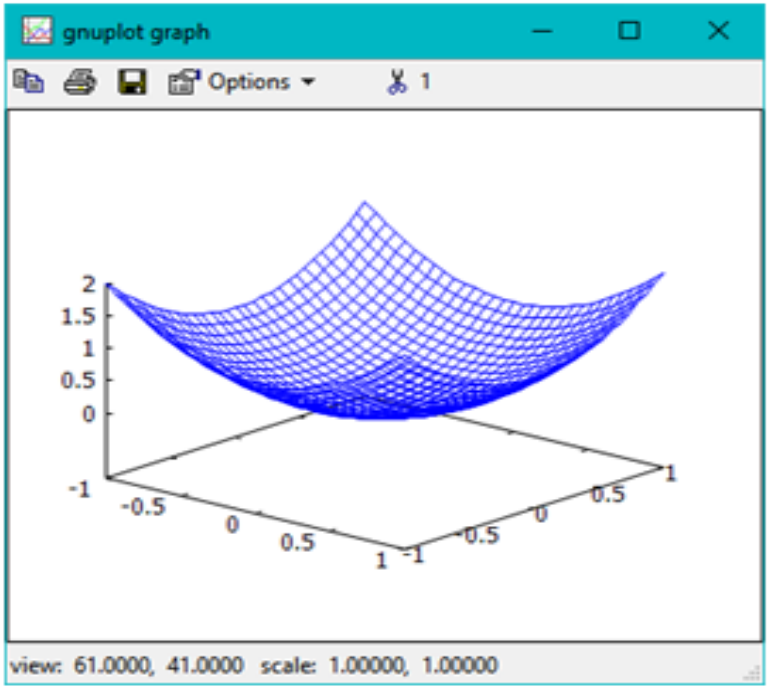


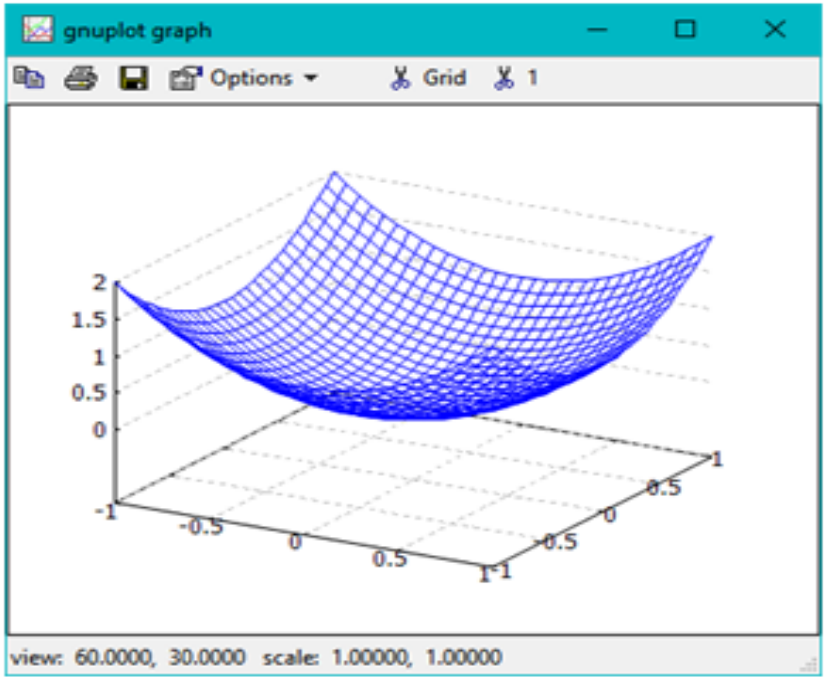
Пример построения поверхности двумя способами

Функция Z=f(x,y) конструируется при помощи вызова
`explicit(x^2+y^2,x,-1,1,y,-1,1)`

Второй график рисуется с «координатной сеткой»:

```
draw(gr3d(explicit(x^2+y^2,x,-1,1,y,-1,1)));
draw3d(grid=true, explicit(x^2+y^2,x,-1,1,y,-1,1));
```





Основные функции и примеры

Изображение графика можно «анимировать» и записать в файл:

```
kill(all);
load(draw);
draw(
        delay=100,
        file_name = "d:\\zzz",
        terminal  = animated_gif,
        gr2d(explicit(x^2,x,-1,1)),
        gr2d(explicit(x^3,x,-1,1)),
        gr2d(explicit(x^4,x,-1,1))
);
```

Функция **draw3d** строит трёхмерные графики

```
draw3d(zlabel = "Z variable",
       ylabel = "Y variable",
       explicit(sin(x^2+y^2),x,-2,2,y,-2,2),
       xlabel="X variable");
```