МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Символьные вычисления в системе компьютерной математики Maxima

Учебное пособие для вузов

пособие для студентов, обучающихся по направлениям 01.03.01 Математика, 02.03.01 Математика и компьютерные науки, 01.03.04 Прикладная математика и по специальности 01.05.01 Фундаментальная математика и механика

Воронеж
2015

Утверждено научно-методическим советом математического факультета

29 октября 2015 года

Протокол № 0500-09

Составители: С.А. Ткачева, Л.В. Безручкина, П.В. Садчиков

Рецензент: к.ф-м. н., доцент Бурлуцкая М.Ш.

Учебно-методическое пособие подготовлено на кафедре уравнений в частных производных и теории вероятностей математического факультета Воронежского государственного университета

Рекомендуется для студентов 2-5 курсов очной формы обучения математического факультета, обучающихся *по направлениям 01.03.01 Математика, 02.03.01 Математика и компьютерные науки, 01.03.04 Прикладная математика и по специальности 01.05.01 Фундаментальная математика и механика*

Что такое символьные вычисления

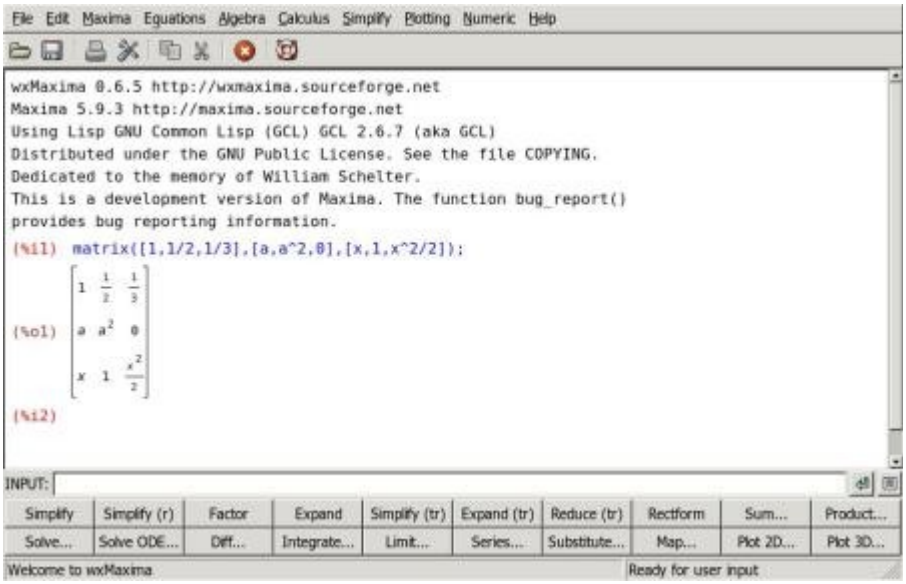
Что представляют эти самые символьные или, как их еще называют, аналитические вычисления, в отличие от численных расчетов. Компьютеры, как известно, оперируют с числами (целыми и с плавающей запятой). К примеру, решения уравнения $x^2 = 2x + 1$ можно получить как −0.41421356 и 2.41421356, а $3x = 1$ — как 0.33333333. А ведь хотелось бы увидеть не приближенную цифровую запись, а точную величину, т. е. $1\pm\sqrt{2}$ в первом случае и 1/3 во втором. С этого простейшего примера и начинается разница между численными и символьными вычислениями. Но кроме этого, есть еще задачи, которые вообще невозможно решить численно. Например, параметрические уравнения, где в виде решения нужно выразить неизвестное через параметр; или нахождение производной от функции; да практически любую достаточно общую задачу можно решить только в символьном виде. Поэтому неудивительно, что и для такого класса задач появились компьютерные программы, оперирующие не только числами, а почти любыми математическими объектами, от векторов до тензоров, от функций до дифференциальных уравнений и т.д.

Среди математического ПО для аналитических (символьных) вычислений наиболее широко известно коммерческое (*Maple*, *Mathematica*); это очень мощный инструмент для ученого или преподавателя, аспиранта или студента, позволяющий автоматизировать наиболее рутинную и требующую повышенного внимания часть работы, оперирующий при этом аналитической записью данных, т. е. фактически математическими формулами. Такую программу можно назвать средой программирования, с той разницей, что в качестве элементов языка программирования выступают привычные человеку математические обозначения. История проекта, известного ныне под именем Maxima, началась еще в конце 60-х годов в легендарном MIT (Massachusetts Institute of Technology — Массачусетский Технологический институт), когда в рамках существовавшего в те годы большого проекта MAC началась работа над программой символьных вычислений, которая получила имя Macsyma (от MAC Symbolic MAnipulation). Архитектура системы была разработана к июлю 1968 г., непосредственно программирование началось в июле 1969. в качестве языка для разработки системы был выбран Lisp, и история показала, насколько это был правильный выбор: из существующих в то время языков программирования он единственный продолжает развиваться и сейчас — спустя почти полвека после старта проекта. Принципы, положенные в основу проекта, позднее были заимствованы наиболее активно развивающимися ныне коммерческими программами — Mathematica и Maple; таким образом, Macsyma фактически стала родоначальником всего направления программ символьной математики. Естественно, Macsyma была закрытым коммерческим проектом; его финансировали государственные и частные организации, среди которых были вошедшее в историю ARPA (Advanced Research Projects Agency; помните ARPAnet — предок интернета?), Энергетический и Оборонный Департаменты США (Departments of Energy & Defence, DOE and DOD). Проект активно развивался, а организации, контролирующие его, менялись не раз, как это всегда бывает с долгоживущими закрытыми проектами. в 1982 году профессор Уильям Шелтер (William Schelter) начал разрабатывать свою версию на основе этого же кода, под названием Maxima. в 1998 году Шелтеру удалось получить от DOE права на публикацию кода по лицензии GPL. Первоначальный проект Macsyma прекратил свое существование в 1999 году. Уильям Шелтер продолжал заниматься разработкой Maxima вплоть до своей смерти в 2001 году. Но, что характерно для открытого ПО, проект не умер вместе со своим автором и куратором. Сейчас проект продолжает активно развиваться, и участие в нем является лучшей визитной карточкой для математиков и программистов всего мира. На данный момент Maxima выпускается под две платформы: Unix-совместимые системы, т. е. Linux и *BSD, и MS Windows.

Замечание. *Имена функций и переменных в Максиме чувствительны к регистру, то есть прописные и строчные буквы в них различаются. Это не будет в новинку любому, кто уже имел дело с POSIX-совместимыми системами или с такими языками программирования, как, скажем, C или Perl. Удобно это и с точки зрения математика, для которого тоже привычно, что заглавными и строчными буквами могут обозначаться разные объекты (например, множества и их элементы, соответственно).*

Графические интерфейсы Maxima

С точки зрения ознакомления с самой Maxima наибольший интерес представляют два интерфейса. Первый — это отдельная самостоятельная графическая программа по имени wxMaxima. Она, как и сама Maxima, помимо Linux/*BSD существует еще и в версии для MS Windows. В wxMaxima вы вводите формулы в текстовом виде, а вывод Максимы отображается графически, привычными математическими символами. Кроме того, большой упор здесь сделан на удобство ввода: командная строка отделена от окна ввода-вывода, а дополнительные кнопки и система меню позволяют вводить команды не только в текстовом, но и в диалоговом режиме. Так называемое «автодополнение» в командной строке на самом деле с таковым имеет лишь то сходство, что вызывается клавишей «`Tab`». Ведет же оно себя, к сожалению, всего лишь как умная история команд, т. е. вызывает ту команду из уже введенных в этой сессии, которая начинается с заданных в командной строке символов, но не дополняет до имен команд и их параметров. Таким образом, этот интерфейс наиболее удобен в том случае, когда вам нужно много вычислять и видеть результаты на экране; и еще, возможно, в том случае, если вы не очень любите вводить все команды с клавиатуры. Кроме того, wxMaxima предоставляет удобный интерфейс к документации по системе; хотя, так как документация поставляется в формате html, вместо этого можно использовать обычный браузер.



Второй достаточно интересный интерфейс к Maxima — это дополнительный режим в редакторе TeXmacs. Хотя этот редактор имеет общее историческое прошлое с широко известным Emacs, что явствует из названия, но практического сходства между ними мало. TeXmacs разрабатывается для визуального редактирования текстов научной тематики, при котором вы видите на экране редактируемый текст практически в том же виде, в котором он будет распечатан. В частности, он имеет так называемый математический режим ввода, очень удобный для работы с самыми разнообразными формулами, и умеет импортировать/экспортировать текст в LaTeX и XML/HTML. Именно возможностями по

работе с формулами пользуется Maxima, вызванная из TeXmacs'а. Фактически, формулы отображаются в привычной математической нотации, но при этом их можно редактировать и копировать в другие документы наподобие обыкновенного текста. Maxima-сессия вызывается из меню: «*вставить* → *Сессия* → *Maxima*», при этом появляется дополнительное меню с командами Максимы. После запуска сессии можно уже внутри нее перейти в математический режим ввода (меню режимов ввода вызывается первой кнопкой на панели ввода). Этот интерфейс будет наиболее удобен тем, кто хочет использовать результаты вычислений в своих текстах и любит редактировать их в визуальном режиме.

Система символьной математики Maxima

Maxima - еще одна программа для выполнения математических вычислений, символьных преобразований, а также построения разнообразных графиков. Сложные вычисления оформляются в виде отдельных процедур, которые затем могут быть использованы при решении других задач. Система Maxima распространяется под лицензией GPL и доступна как пользователям ОС Linux, так и пользователям MS Windows.

Система компьютерной математики Maxima -- настоящий ветеран среди программ этого класса. Она старше многих своих известных коммерческих собратьев по крайней мере на два десятка лет. Первоначально носившая имя Macsyma, она была создана в конце 1960-х годов в знаменитом Массачусетском технологическом институте и почти 20 лет (с 1982 по 2001) поддерживалась Биллом Шелтером (William Schelter), благодаря которому и приобрела свои замечательные качества и известность в научном мире. Подробности по истории системы, инсталляционный модуль (размером всего в 10 MB), документацию, исходный код и другую сопутствующую информацию можно найти на Web-узле пакета http://maxima.sourceforge.net. Текущая версия (5.15.0) работает под управлением Windows и Linux.

Выбор программы в меню **Пуск** vxmaxima позволит начать сеанс работы с этой программой.

При *старте программы* выводится некоторая информация о системе и "метка" (%i1).

wxMaxima 0.7.5 http://wxmaxima.sourceforge.net

Maxima 5.15.0 http://maxima.sourceforge.net

Using Lisp GNU Common Lisp (GCL) GCL 2.6.8 (aka GCL)

Distributed under the GNU Public License. See the file COPYING.

Dedicated to the memory of William Schelter.

The function bug_report() provides bug reporting information.

```
(%i1)
```

Это сообщение о том, что ядро Максимы только что запустилось (вместо нее, в зависимости от версии и конкретной сборки, (например: версия *Maxima* 5.31.2 (и выше)), может выводиться краткая информация о программе); и приглашение к вводу первой

команды. Команда в Максиме — это любая комбинация математических выражений и встроенных функций, завершенная, в простейшем случае, точкой с запятой. После ввода команды и нажатия «Enter» (в более поздних версиях, например 5.31.2, ввод данных осуществляется с помощью комбинации клавиш «Shift»+«Enter») Maxima выведет результат и будет ожидать следующей команды:

```
(%i1)  (1/2+1/3+1/4)/(1/5+1/6+1/8);

(%o1)  130/59
```

Как видите, каждая ячейка имеет свою метку; эта метка — заключенное в скобки имя ячейки. Каждый ввод и вывод помечаются системой и затем могут быть использованы снова. Символ (%i1) используется для обозначения команд, введенных пользователем, а (%o1) - при выводе результатов вычислений. Ячейки ввода именуются как %i с номером (i от *input* — ввод), ячейки вывода — как %o с соответствующим номером (o от *output* — вывод). Со знака % начинаются все встроенные служебные имена: чтобы, с одной стороны сделать их достаточно короткими и удобными в использовании, а с другой — избежать возможных накладок с пользовательскими именами, которые тоже часто удобно делать короткими. Благодаря такому единообразию вам не придется запоминать, как часто бывает в других системах, какие из таких коротких и удобных имен зарезервированы программой, а какие вы можете использовать для своих нужд. К примеру, внутренними именами **%e** и **%pi** обозначены общеизвестные математические постоянные; а через %c с номером обозначаются константы, используемые при интегрировании, для которых использование буквы «c» традиционно в математике. При вводе мы можем обращаться к любой из предыдущих ячеек по ее имени, подставляя его в любые выражения. Кроме того последняя ячейка вывода обозначается через %, а последняя ячейка ввода — через _. Это позволяет обращаться к последнему результату, не отвлекаясь на то, каков его номер.

```
(%i2)  %+47/59;

(%o2)  3
```

Здесь %+47/59 — то же самое, что %o1+47/59. Вывод результата вычисления не всегда нужен на экране; его можно заглушить, завершив команду символом $ вместо ;. Заглушенный результат при этом все равно вычисляется; как видите, в этом примере ячейки %o1 и %o2 доступны, хотя и не показаны (к ячейке %o2 обращение идет через символ %, смысл которого расшифрован выше):

```
(%i1)  sqrt(2)+3$
(%i2)  2*sqrt(2)+1$
(%i3)  %-%o1;

(%o3)  sqrt(2)-2
```

Каждую следующую команду не обязательно писать с новой строки; если ввести несколько команд в одну строчку, каждой из них все равно будет соответствовать свое имя ячейки. К примеру, здесь в строке после метки %i1 введены ячейки от %i1 до %i4; в ячейке %i3 используются %i1 и %i2 (обозначенная как _ — предыдущий ввод):

```
(%i3)  asin(1/2)$acos(1/2);%i1+_;%o1+%;

(%o4)  %pi/3

(%o5)  %pi/3+130/59

(%o6)  %pi/3+260/59
```

В wxMaxima и TeXmacs последнюю или единственную команду в строке можно не снабжать завершающим символом **;** — это сработает так же, как если бы она была **завершена ;** т. е. вывод заглушен не будет. Если вы выберете другой интерфейс, не забывайте ее добавлять.

Помимо использования имен ячеек, мы, естественно, можем и сами давать имена любым выражениям. По-другому можно сказать, что мы присваиваем значения переменным, с той разницей, что в виде значения такой переменной может выступать любое математическое выражение. Делается это с помощью двоеточия — знак равенства оставлен уравнениям, которые, учитывая общий математический контекст записи, проще и привычнее так читаются. И к тому же, так как основной конек Максимы — символьная запись и аналитические вычисления, уравнения достаточно часто используются. Например:

```
(%i1)  equation:x^3-x=0$

(%i2)  solve(equation);
(%o2)  [x=-1,x=1,x=0]
```

В каком-то смысле двоеточие даже нагляднее в таком контексте, чем знак равенства: это можно понимать так, что мы задаем некое обозначение, а затем через двоеточие расшифровываем, что именно оно обозначает. После того, как выражение поименовано, мы в любой момент можем вызвать его по имени:

```
(%i3)  diff(equation,x);
(%o3)  3x^2-1=0

(%i4) solve([x^2+6*x+9], [x]);

(%o4) [x=-3]
```

Таким образом: *для инициализации процесса вычислений следует ввести команду, затем символ ; (точка с запятой) и нажать клавишу* ***Enter (или Shift + Enter).*** *Если не требуется вывод полученной информации на экран, то вместо точки с запятой используется символ* ***$.*** *Обратиться к результату последней команды можно с помощью*

символа `%`*. Для повтора ранее введенной команды, скажем* `(%i2)`*, достаточно ввести два апострофа и затем метку требуемой команды, например,* `'(%i2) '`

Система Maxima обращает внимание на регистр введенных символов в именах встроенных констант и функций. Регистр букв важен при использовании переменных, например, Maxima считает x и X разными *переменными.*

Для стандартных математических констант используются следующие обозначения: `%e` для основания натуральных логарифмов, **%i** для мнимой единицы (квадратный корень из числа -1) и **%pi** для числа π.

Присваивание значения какой-либо переменной осуществляется с помощью знака **:** (двоеточие), а символ **=** (равно) используется при задании уравнений или подстановок.

```
(%i1) x:2;
(%o1)                                 2
(%i2) y:3;
(%o2)                                 3
(%i3) x + y;
(%o3)                                 5
```

Любое имя можно очистить от присвоенного ему выражения функцией kill(), и освободить занимаемую этим выражением память. Для этого нужно просто набрать kill(name), где name — имя уничтожаемого выражения; причем это может быть как имя, назначенное вами, так и любая ячейка ввода или вывода. Точно так же можно очистить разом всю память и освободить все имена, введя kill(all). В этом случае очистятся в том числе и все ячейки ввода-вывода, и их нумерация опять начнется с единицы. В дальнейшем, если по контексту будет иметься в виду логическое продолжение предыдущих строк ввода-вывода, я буду продолжать нумерацию (этим приемом я уже воспользовался выше). Когда же новый «сеанс» будет никак не связан с предыдущим, буду начинать нумерацию заново; это будет косвенным указанием сделать «kill(all)», если вы будете набирать примеры в Maxima, так как имена переменных и ячеек в таких «сеансах» могут повторяться. Для этого можно использовать основное меню. Кликаем ЛКМ по кнопке **Maxima** основного меню, переходим на строку **Очистить память** и еще раз кликнем ЛКМ. В этом меню можно также **Прервать** вычисления, **Перезапустить Maxima** и др. операции.

Функция **`kill`** *аннулирует присвоенные ранее значения переменных. Параметр* `all` *этой функции приводит к удалению значения всех переменных, включая метки* `(%i)` *и* `(%o)`*.*

```
(%i4) kill(x);
(%o4)                                done
(%i5) x + y;
(%o5)                              x + 3
(%i6) kill(all);
(%o0)                                done
(%i2) x + y;
(%o2)                              y + x
```

Для завершения работы с системой применяется функция **quit();**, а прерывание процесса вычислений осуществляется путем нажатия комбинации клавиш Ctrl- (после чего следует ввести :q для возврата в обычный режим работы), или команды в меню: **Maxima ->Прервать.** Можно использовать также команды: **Очистить память,** а также **Перезапустить Maxima.**

Работа с выражениями

При записи математических выражений могут использоваться четыре стандартные арифметрические операции (+, -, *, /) и операция возведения в степень (^, ^^ или **). Приоритет этих операций традиционен, для изменения порядка вычислений следует использовать круглые скобки. Кроме чисел выражения могут содержать результаты вычислений математических функций. Аргументы функций указываются в круглых скобках, например, запись sqrt(5) означает корень квадратный из числа 5. Если в результате расчета получается дробное выражение, то оно и выводится в виде обыкновенной дроби. Иррациональные числа, входящие в выражение, представляются в символьном виде.

Примеры работы программы с арифметическими операциями:

Сложение и умножение коммутирующие операции. Вычитание a- b представлено в Максима как сложение a+ (-b). Деление a/ b представлено в Максима как умножение, a * b ^ (-1).

```
(%i1) c + g + d + a + b + e + f;
(%o1)                g + f + e + d + c + b + a
(%i2) [op (%), args (%)];
(%o2)              [+, [g, f, e, d, c, b, a]]
(%i3) c * g * d * a * b * e * f;
(%o3)                       a b c d e f g
(%i4) [op (%), args (%)];
(%o4)               [*, [a, b, c, d, e, f, g]]
(%i5) apply ("+", [a, 8, x, 2, 9, x, x, a]);
(%o5)                      3 x + 2 a + 19
(%i6) apply ("*", [a, 8, x, 2, 9, x, x, a]);
                                  2  3
(%o6)                        144 a  x
```

op (операция), args(аргументы), apply(применить).

Деление и возведение в степень не коммутирующие операции. Операции обозначаются как "/" и "^".

```
(%i1) [a / b, a ^ b];
                               a   b
(%o1)                         [-, a ]
                               b
(%i2) [map (op, %), map (args, %)];
(%o2)               [[/, ^], [[a, b], [a, b]]]
(%i3) [apply ("/", [a, b]), apply ("^", [a, b])];
                               a   b
(%o3)                         [-, a ]
                               b
```

```
(%i1) 17 + b - (1/2)*29 + 11^(2/4);
                                        5
(%o1)                  b + sqrt(11) + -
                                        2
(%i2) [17 + 29, 17 + 29.0, 17 + 29b0];
(%o2)                   [46, 46.0, 4.6b1]
```

Для раскрытия скобок используется функция **expand**. Команда **ev** позволит получить численное значение выражения. Ее первый аргумент есть вычисляемое выражение, а второй - опция **numer**. Напомним, что символ **%** означает результат предыдущего вычисления.

```
(%i3) 1/100+1/101;

(%o3) 201/10100

(%i4) (1+sqrt(2))^5;

(%o4) (sqrt(2)+1)^5

(%i5) expand(%i14);

(%o5) 29*sqrt(2)+41

(%i6) ev(%i15,numer);

(%o6) 82.01219330881975
```

Допускается более удобная форма функции **ev,** требующая указания только ее аргументов:

```
(%i7) 29*sqrt(2) + 41, numer;
(%o7)                        82.01219330881976
```

По умолчанию результат содержит 16 значащих цифр. Для вывода числа в экспоненциальной форме используется функция **bfloat**:

```
(%i8) bfloat(%i7);
(%o8)                        ev(8.201219330881976b1,numer)
```

Запись mbn есть сокращенная форма выражения $m*10^n$.

Количество значащих цифр в представлении числа определяется специальной переменной **FPPREC.** Увеличение ее значения приводит к возрастанию точности результата, например,

```
(%i9) fpprec;
(%o9)                        16

(%i10) fpprec:100;
(%o10)                        100

(%i11) ''(%i7);
(%o11)                        8.20121933088197607657604923686248525#
03077530516721861648463104778270702443495 4#
83506838511144226151 55B1
```

Символ # в конце выводимой строки означает, что число не уместилось на одной строке и его оставшаяся часть переносится на следующую. В последнем примере мы использовали повторение ранее введенной команды "(%i7);

Система Maxima может работать с числами произвольной длины и точности:

```
(%i12) 100!;
(%o12)                        933262154439441526816992388562667004#
907159682643816214685929638952175999932299 1#
560894146397615651828625369792082722375825 1#
185210916864000000000000000000000000
```

Пример

Вычислим число pi с точностью 200 знаков после запятой:

```
(%i13) %pi, numer;
(%o13)               3.141592653589793

(%i14) fpprec:200;
(%o14)                      200

(%i15) bfloat(%pi);
(%o15)          3.14159265358979323846264338327950288#
419716939937510582097494459230781640628620899#
862803482534211706798214808651328230664709384#
460955058223172535940812848111745028410270l#
938521105559644622948954930381980
```

Если требуется вывод с увеличенной точностью только для результатов нескольких команд, то следует использовать функцию **block**. Первым аргументом этой функции является список локальных переменных, т. е. переменных, значения которых будут восстановлены после завершения выполнения команд блока. При указании таких переменных возможно присваивание им временных значений. После списка локальных переменных через запятую указывается последовательность команд.

```
(%i16) FPPREC:16;
(%o16)                                  16

(%i17) bfloat(%pi);
(%o17)                    3.141592653589793B0

(%i18) block([FPPREC:100], bfloat(%pi));
(%o18)      3.14159265358979323846264338327950#
288419716939937510582097494459230781640628#
286208998628034825342117068B0

(%i19) ''(%i17);
(%o19)                      3.141592653589793B0
```

Функции в Maxima: log(x) – натуральный логарифм; abs(x) – модуль; sqrt(x) – корень квадратный; mod(x) – остаток от деления, min(x1, x2,…, xn) - минимальный элемент из списка; max(x1, x2,…, xn) - максимальный элемент из списка.

В Maxima доступны прямые и обратные тригонометрические функции: sin (синус), cos (косинус), tan (тангенс), cot (котангенс), asin (арксинус), acos (арккосинус), atan (арктангенс), acot (арккотангенс). Кроме них имеются две менее известные функции - секонс (sec x = 1/cos x) и косеконс (csc x = 1/sin x).

```
(%i20)      block([FPPREC:100],sin(bfloat(%pi)));
(%o20)      - 2.57057919802972371168956906471378l#
273847841138560124733757044937800020983036 8#
3173940359193381364537 7B-101

(%i21)      block([FPPREC:100],sin(%pi)) ;
(%o21)      0

(%i22)      sin(%pi /6)^2 +cos(%pi /6)^2;
(%i22)      1

(%i23)      sec(%pi /3);
(%o23)                                  2
```

```
(%i24)      asin(1/2);
                                    % pi
(%o24)                               ---
                                     6
```

К сожалению, Maxima не может в символьном виде вычислить значения обратных тригонометрических функций в некоторых точках:

```
(%i25)      asin(sqrt(3)/2);
                           SQRT(3)
(%o25)                        ASIN(-------)
                              2
(%i26)      %, expand;
                           SQRT(3)
(%o26)                        ASIN(-------)
                              2
```

Для вычисления натурального логарифма используется функция `log`:

```
(%i27)      log(%e^2);
(%o27)                                2
(%i28)      5*log(a)+6*log(b);
(%o28)                        6*log(b)+5*log(a)
(%i29)      logcontract(%);

(%o29)      log(a^5*b^6)
```

Алгебраические преобразования

Особо важную роль играет способность системы Maxima производить разнообразные символьные преобразования алгебраических выражений. Следующий пример демонстрирует раскрытие скобок в них:

```
(%i1)      p1:x^2-1;

(%o1)                           x^2-1

(%i2)      p2:x-1;
(%o2)                                x - 1

(%i3)      expand(p1*p2);

(%o3)                            x^3-x^2-x+1
(%i4)      expand((p1+p2)^2);

(%o4)                       x^4+2*x^3-3*x^2-4*x+4
```

Функцию **divide** можно использовать для нахождения частного и остатка от деления одного многочлена на другой:

```
(%i5)      divide(p1*p2, p1);
(%o5)                                [x - 1, 0]
```

Функция **gcd** определяет наибольший общий делитель многочленов, а **factor** осуществляет разложение многочлена на множители:

```
(%i6)      gcd(x^3-1, x^2-1, x-1);
(%o6)         x - 1

(%i7)      factor(x^8-1);

(%o7)           (x-1)*(x+1)*(x^2+1)*(x^4+1)
```

Подстановка какого-либо выражения вместо переменной осуществляется при помощи операции =. Например, заменим все вхождения x в выражении на 5/z:

```
(%i8)      x^4+3*x^3-2*x, x=5/z;

(%o8)                        -10/z+375/z^3+625/z^4
```

Функция **ratsimp** выносит за скобки наибольший общий делитель:

```
(%i9)      ratsimp(%);
(%o9) -(10*z^3-375*z-625)/z^4
```

Используя функцию **assume** (to assume - допускать), можно при вычислениях учитывать дополнительные условия, задаваемые неравенствами:

```
(%i10) sqrt(x^2);
(%o10)                              ABS(x)
(%i11) assume(x<0);
(%o11)                              [x < 0]
(%i12) sqrt(x^2);
(%o12)                               - x
```

Функция **forget** (to forget - забывать) снимает все ограничения, наложенные при помощи `assume`:

```
(%i13) forget(x<0);
(%o13)                              [x < 0]
(%i14) sqrt(x^2);
(%o14)                              ABS(x)
```

Maxima легко оперирует тригонометрическими выражениями. Так, функция **trigexpand** использует формулы преобразования сумм двух углов для представления введенного выражения в как можно более простом виде:

```
(%i15) sin(u+v)*cos(u)^3;


(%o15)              cos(u)^3*sin(v+u)

(%o14) (sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8trigexpand(%);

(%o16)       cos(u)^3*(cos(u)*sin(v)+sin(u)*cos(v))
```

Функция **trigreduce** преобразует тригонометрическое выражение к сумме элементов, каждый из которых содержит только единственный sin или cos:

```
(%i17) (sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8trigreduce(%);

(%o17) (sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8
```

Функции **realpart** и **imagpart** возвращают действительную и мнимую часть комплексного выражения:

```
(%i18) z1:-3+%i*4;
(%o18)                        4 %i - 3

(%i19) z2:4-2*%i;
(%o19)                        4 - 2 %i

(%i20) z1*z2;
(%o20)          (4-2*%i)*(4*%i-3)
```

```
(%i21) expand(%);
(%o21)                              22 %i - 4

(%i22) realpart(%);
(%o22)                                    - 4

(%i23) imagpart(%);
(%o23)                                     22
```

Следующие примеры иллюстрируют присвоения в выражениях:

```
(%i1) p(x):= x^2+3*x-8;
                                    2
(%o1)                   p(x) := x  + 3 x - 8

(%i2) p(2*x);
                                    2
(%o2)                            4 x  + 6 x - 8
(%i3) factor(p(2*x-4));
                                  2
(%o3)                       2 (2 x  - 5 x - 2)
(%i4) f(x):=sin(x);
(%o4)                                  f(x) := SIN(x)
(%i5) g(x):=log(x);
(%o5)                                  g(x) := LOG(x)
(%i6) f(g(x));
(%o6)                               SIN(LOG(x))
(%i7) g(f(x));
(%o7)                                     LOG(SIN(x))
(%i8) dist(x,y):=sqrt(x^2+y^2);
                                         2    2
(%o8)             dist(x, y) := SQRT(x  + y )
(%i9) dist(3,4);
(%o9)                                                5
```

Operator: **!**

Оператор факториала. Переменные factlim, minfactorial, и factcomb управляют упрощением выражений, содержащих факториалы.

```
(%i1) factlim : 10;
(%o1)                               10
(%i2) [0!, (7/2)!, 4.77!, 8!, 20!];
             105 sqrt(%pi)
(%o2)   [1, -------------, 81.44668037931199, 40320, 20!]
```

В случае комплексного числа упрощают факториал после оценки операнда.

```
(%i1) [(%i + 1)!, %pi!, %e!, (cos(1) + sin(1))!];
(%o1)      [(%i + 1)!, %pi!, %e!, (sin(1) + cos(1))!]
(%i2) ev (%, numer, %enumer);
(%o2) [(%i + 1)!, 7.188082728976037, 4.260820476357,
                                                1.227580202486819]
```

```
(%i1) kill (foo);
(%o1)                         done
(%i2) foo!;
(%o2)                         foo!
```

Восклицательный знак, стоящий после своего аргумента (т. е. постфиксный оператор), традиционно обозначает факториал. Двумя восклицательными знаками обозначен полуфакториал. Функции abs(x) и signum(x) возвращают модуль и знак числа. А функции max(x1,...,xn) и min(x1,...,xn) — соответственно максимальное и минимальное из заданных чисел. При этом факториал от натурального числа (и нуля) автоматически упрощается до натурального же числа:

```
(%i1)  0!;6!;28!;
(%o1)  1
(%o2)  720
(%o3)  304888344611713860501504000000
```

Точно так же и модуль определен для всех комплексных чисел. Минимум, максимум и знак определены только для действительных чисел, так как комплексные числа общего вида, как известно, между собой несравнимы.

```
(%i4)  abs(-x);
(%o4)  |x|

(%i5)  max(x,x+1,x-abs(a));
(%o5)  x+1
```

Справка о той или иной функции выводится по команде ***describe*** *(имя функции). При работе в графической оболочке wxMaxima, можно воспользоваться пунктом меню* ***help.*** *Процедура* ***example****(имя функции) демонстрирует примеры использования функции.*

Использование логических операций

В Maxima, в отличие от большинства «традиционных» процедурных и объектных языков программирования, где существует так называемый условный оператор, привычная связка if-then-else является не синтаксической конструкцией, а самым настоящим оператором. По своему действию он больше всего похож на оператор языка C, только с синтаксисом: if условие then выражение1 else выражение2. При выполнении «условия» из двух «выражений» вычисляется только первое и возвращается как результат оператора; в противном случае выполняется только второе и оно же является значением всего выражения if-then-else. Часть конструкции else выражение2, как и в большинстве языков

программирования, опциональна. Если ее нет, а условие все-таки не выполнилось, результат оператора if будет равен false.

Operator: <
Operator: <=
Operator: >=
Operator: >

```
(%i1) [x, y, z] : [123, 456, 789];
(%o1)                     [123, 456, 789]
(%i2) is (x < y);
(%o2)                          true
(%i3) maybe (y > z);
(%o3)                          false
(%i4) if x >= z then 1 else 0;
(%o4)                            0
(%i5) block ([S], S : 0, for i:1 while i <= 100 do S : S + i, return
(S));
(%o5)                          5050
```

```
(%o1)                     [123, 456, 789]
(%i2) [x < y, y <= z, z >= y, y > z];
(%o2)    [123 < 456, 456 <= 789, 789 >= 456, 456 > 789]
(%i3) map (is, %);
(%o3)                [true, true, true
```

```
(%i1) eta(x,y):= if x = y then x
              else (if x > y then x-y else x+y);
(%o2)          eta(x, y) := IF x = y THEN x ELSE
              (IF x > y THEN x - y ELSE x + y)
(%i2) eta(5,6);
(%o2)                          11
(%i3) eta(eta(7,7), eta(1,2));
(%o3)
```

При этом, конечно же, никто вам не мешает использовать этот оператор как обычную условную конструкцию, а возвращаемое значение просто игнорировать. С другой стороны, оператор if можно применять, например, для задания рекурсивных последовательностей:

(%i1) $a(n) := \text{if } n \leqslant 1 \text{ then } n \text{ else } \frac{a(n-1)+a(n-2)}{2}$

(%i2) $a(20)$

(%o2) $\frac{349525}{524288}$

(%i3) $\%, \text{numer}$

(%o3) 0.66666603088379

Немного о самих условиях, которые могут проверяться оператором if. Условия >, <, >=, <= записываются и расшифровываются традиционно, так же как и логические операторы and, or, not. А вот о равенствах-неравенствах нужно сказать пару слов. Равенство в Maxima есть двух видов: синтаксическое и логическое. Знаком = обозначается как раз первое, а второе вычисляется с помощью функции equal(). Чтобы не быть многословными, отличие синтаксического равенства от логического продемонстрируем на примере; здесь дополнительно используется предикат по имени is, которые проверяет на истинность свой аргумент.

```
(%i1) is((x+1)^2=x^2+2*x+1)
(%o1) false
(%i2) is(equal((x+1)^2,x^2+2*x+1))
(%o2) true
```

Ну и неравенств, тоже существует два, с тем же смыслом. Синтаксическое неравенство обозначается — через #. Логическое неравенство обозначено через notequal().

Конечно, кроме упомянутых сравнений в условном операторе можно использовать любые предикаты, то есть функции, возвращающие логические значения true/false. Цикл в Maxima будто бы тоже один. Но он имеет много различных вариантов. Вот как выглядят основные разновидности:

- for переменная:начало step шаг thru конец do выражение
- for переменная:начало step шаг while условие do выражение
- for переменная:начало step шаг unless условие do выражение

Первый прокручивает цикл, изменяя переменную с заданным шагом от начала до конца; второй — от начала и пока выполняется условие; третий — наоборот, пока условие не выполняется. К примеру, мы можем получить список из первых десяти членов последовательности из позапрошлого примера:

```
(%i1) a(n):=if n<=1 then n else (a(n-1)+a(n-2))/2$
(%i2) A:[]$for n:0 thru 10 do A:append(A,[a(n)])
(%o3) done
(%i4) A
(%o4) [0,1,1/2,3/4,5/8,11/16,21/32,43/64,85/128,171/256,341/512]
(%i5) A,numer
(%o5) [0,1,0.5,0.75,0.625,0.6875,0.65625,
0.671875,0.6640625,0.66796875,0.666015625]
```

Как видите, в качестве оператора цикл в простейшем его виде, в отличие от условия, использовать смысла нет, так как его возвращаемое значение всегда равно done. В этом примере один из элементов циклического оператора не указан; шаг может быть опущен и по умолчанию равен единице. Самое интересное в этом операторе то, что опустить

позволяется любую его часть, кроме do; и в том числе в любых комбинациях. К примеру, опустив кроме step еще и for, мы получаем из этого же оператора традиционные циклы while и unless (второй и третий варианты). А проделав то же самое с первым вариантом записи, получим цикл без счетчика вида thru число do выражение, который просто повторится заданное число раз. Можно, наоборот, опустить условие окончания и получить цикл с индексной переменной, но бесконечный. А оставив только do, получим самый простой вариант бесконечного цикла. Из таких бесконечных циклов можно выйти с помощью оператора return(выражение) (точнее, конструкции из двух операторов вида if условие then return(выражение)), который прервет выполнение цикла и вместо done вернет заданное выражение. Естественно, оператор return() можно применять во всех видах циклов, а не только в бесконечных.

Но и это еще не все. Кроме всех уже рассмотренных вариаций, цикл может принимать еще две возможности. Во-первых, вместо step может использоваться конструкция next выражение, смысл которой виден на примере

```
(%i1) for i: 162304 next i/2 while integerp(i) do display(i)$
```

```
i = 162304
i = 81152
i = 40576
i = 20288
i = 10144
i = 5072
i = 2536
i = 1268
i = 634
i = 317
```

После next может стоять любое вычислимое выражение относительно индекса цикла, и применяться эта конструкция может во всех трех вариантах цикла (thru/while/unless).

А «во-вторых» — это еще один отдельный вариант цикла: for переменная in список do выражение; либо расширенная форма: for переменная in список условие do выражение. Здесь цикл будет прокручен с переменной, изменяющейся по всем элементам списка; плюс можно задать еще и дополнительное условие на прерывание цикла. Группировка, или, как ее принято называть, составной оператор, в Maxima — это опять-таки самый настоящий оператор, который тоже, как и положено оператору, возвращает некоторое значение. Обозначается он скобками, самыми что ни на есть круглыми и обыкновенными; а разделяются сгруппированные операторы/выражения внутри этих скобок не менее обыкновенными запятыми. Возвращаемым значением составного оператора является последнее вычисленное выражение.

С условным оператором, столь разнообразными циклами и составным оператором мы уже можем, комбинируя их между собой и с любыми другими функциями и выражениями Maxima, писать полноценные программы с использованием богатого символьного математического аппарата. Естественно, теперь захочется сохранять эти программы в виде внешних файлов, чтобы не набирать их каждый раз вручную, а подгружать одной короткой командой.

Операции математического анализа

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

Maxima может вычислять производные и интегралы, раскладывать функции в ряды Тейлора, вычислять пределы и находить точные решения обыкновенных дифференциальных уравнений.

Функция нахождения предела может принимать три различных варианта списка аргументов. Эта функция вполне соответственно ее действию: limit; и ее вызов выглядит как limit(выражение, переменная, точка), то есть то, что в математической записи выглядит как $\lim_{x \to a} f(x)$, в контексте Maxima запишется как limit(f(x), x, a):

(%i1) $\text{limit}\left(\frac{x^2-1}{2x^2-x-1}, x, 1\right)$

(%o1) $\frac{2}{3}$

(%i2) $\text{limit}\left(\frac{(1+mx)^n-(1+nx)^m}{x^2}, x, 0\right)$

(%o2) $-\frac{m(m-n)n}{2}$

(%i3) $\text{limit}\left(\frac{(x^n-a^n)-na^{n-1}(x-a)}{(x-a)^2}, x, a\right)$

(%o3) $\frac{n(a^n n-a^n)}{2a^2}$

Maxima может искать пределы не только в конечных точках, но и на бесконечности. Среди стандартных обозначений программы существуют универсальные названия для разных бесконечностей: плюс-бесконечность записывается через inf (от слова *infinity*, как нетрудно догадаться), минус-бесконечность — через minf (от *minus infinity*); для комплексных чисел бесконечность, как известно, одна, и она (комплексная бесконечность) обозначается полным словом infinity. При работе с пределами все три обозначения могут как использоваться при вводе, так и возникать в виде найденного значения предела; отдельно здесь надо отметить один момент касательно работы с интерфейсом к Maxima в редакторе TeXmacs: символы inf и minf при выводе здесь отображаются в своей традиционной математической нотации, то есть как ∞ и $-\infty$; символ вместо inf можно, кроме того, использовать еще и при вводе.

(%i1) $\lim\left(\frac{\sqrt{x}+\sqrt[3]{x}+\sqrt[4]{x}}{\sqrt{2x+1}}, x, \infty\right)$

(%o1) $\frac{1}{\sqrt{2}}$

(%i2) $\lim\left(\sqrt{x+\sqrt{x+\sqrt{x}}}-x, x, \infty\right)$

(%o2) $-\infty$

Второй вариант вызова функции limit() — это расширенная версия первого: limit(выражение, переменная, точка, направление), для поиска односторонних пределов. Для предела справа в качестве «направления» указывается plus, для предела слева — minus:

(%i1) $\lim\left(\frac{\mathrm{abs}(\sin(x))}{x}, x, 0, \mathrm{plus}\right)$

(%o1) 1

(%i2) $\lim\left(\frac{\mathrm{abs}(\sin(x))}{x}, x, 0, \mathrm{minus}\right)$

(%o2) -1

Кроме упомянутых выше бесконечностей, на выходе возможно появление и еще двух обозначений, на случай, если заданный предел не существует: **ind** (от слова *indefinite* — *неопределенный*) и **und** (от слова *undefined* — опять же *неопределенный*). В документации первое из этих обозначений описано как *indefinite but bounded* (*не определен, но ограничен*), что дает предположить, что функция, не имеющая предела, при этом ограничена либо в окрестности предельной точки, либо на всей прямой.

(%i1) $\lim\left(\sin\left(\frac{1}{x}\right), x, 0\right); \lim\left(\tan\left(\frac{1}{x}\right), x, 0\right)$

(%o1) ind
(%o2) und

Здесь все правильно, tan(1/x) не ограничена в окрестности нуля. Рассмотрим следующие примеры:

```
(%i3) limit(a: 1/(1-e^(1/x)), x, 0)
(%o3) und
(%i4) limit(a, x, 0, plus); limit(a, x, 0, minus); limit(a, x, inf); limit(a, x, minf)
(%o4) 0
(%o5) 1
(%o6) -inf
(%o7) inf
(%i8) limit(a: atan(e^(1/x)), x, 0)
(%o8) und
(%i9) limit(a, x, 0, plus); limit(a, x, 0, minus); limit(a, x, inf); limit(a, x, minf)
(%o9) %pi/2
(%o10) 0
(%o11) %pi/4
(%o12) %pi/4
```

Первая функция имеет конечные односторонние пределы в нуле, а вторая ограничена вообще на всей оси. Но это, думаю, не столь критично: главное, что наличие любого из этих символов в качестве вывода дает нам понять, что искомого предела не существует.

Функция **limit**() в третьем варианте — limit(выражение) — предназначена уже не для поиска собственно пределов, а для упрощения выражений, содержащих символы inf и minf:

```
(%i1) limit(1/inf)
(%o1) 0
```

Выражения такого рода могут возникать, к примеру, при подстановках в формулы результатов вычисления каких-то других пределов или интегралов.

Такая способность — принимать различные списки аргументов — не является в Maxima чем-то особенным; она свойственна очень многим встроенным функциям, как и различное действие в зависимости от значений разнообразных переключателей. Имена наиболее часто используемых функций запомнить несложно, а о дополнительных ключах или флагах, в случае чего, можно прочитать во встроенной справке, набрав имя-функции.

Об этих самых ключах к функции limit и осталось рассказать. Первый ключ называется lhospitallim и задает максимальное количество применений правила Лопиталя; Правило это гласит, что в случае неопределенности вида 0/0 или можно продифференцировать числитель и знаменатель — и предел от этого не изменится. Ограничитель количества применений этого правила нужен для того, чтобы избежать зацикливаний, которые могут случиться для бесконечно дифференцируемых функций, у которых в данной точке равны нулю либо бесконечности все производные. По умолчанию значение lhospitallim равно четырем. Второй ключ к функции limit — это флаг limsubst,

который, будучи выставлен в true, позволяет этой функции производить подстановки внутрь неизвестных выражений. По умолчанию этот флаг равен false, что исключает ошибки вроде такой:

(%i1) $\text{limit}\left(\frac{f(x)}{f(x+a)}, x, \infty\right), \text{limsubst: true}$

(%o1) 1

(%i2) $f(x) := \sin(x)\$"\%i1$

(%o3) und

И, наконец, последний дополнительный параметр — еще один флаг, по имени **tlimswitch.** По умолчанию он тоже выключен, а если его включить, функция **limit** будет, при невозможности найти предел другими способами, пытаться его найти путем разложения функции в ряд Тейлора в окрестности заданной точки:

(%i1) $\text{limit}\left(\frac{\sqrt{x+\sqrt{x+\sqrt{x}}}}{\sqrt{x+1}}, x, \text{inf}\right)$

```
Quotient by a polynomial of higher degree
 -- an error.  Quitting.  To debug this try debugmode(true);
```

(%i2) $\text{limit}\left(\frac{\sqrt{x+\sqrt{x+\sqrt{x}}}}{\sqrt{x+1}}, x, \text{inf}\right), \text{tlimswitch: true}$

(%o2) 1

Но в случае поиска односторонних пределов, в тех точках, где они не равны между собой, то есть полного предела не существует, этим флагом нужно пользоваться с осторожностью: при его включении функция **limit** может вернуть в качестве полного предела один из односторонних:

(%i1) $\text{limit}\left(\sqrt{\frac{1}{x}+\sqrt{\frac{1}{x}+\sqrt{\frac{1}{x}}}}-\sqrt{\frac{1}{x}-\sqrt{\frac{1}{x}+\sqrt{\frac{1}{x}}}}, x, 0\right), \text{tlimswitch: true}$

(%o1) 1

У этой функции в точке ноль - только предел справа равен единице; а предел слева — нулю.

Для вычисления пределов используется функция limit:

```
(%i1) limit(1/x,x,inf);
(%o1)                                0
```

Для вычисления односторонних пределов используется дополнительный параметр, принимающий значение plus для вычисления предела справа и minus - слева.

Дифференцирование и интегрирование

Функция diff, diff(выражение, переменная)- возвращает производную от «выражения» по заданной переменной; с одним, diff(выражение) — полный дифференциал заданного выражения. Другими словами, запись diff(f, x) равнозначна математическому обозначению d*f*/d*x*, а diff(f) — d*f*.

Но это еще не все. Кроме одного либо двух, эта функция может также принимать любое нечетное число агрументов вида diff(выражение, переменная, порядок, переменная, порядок, …) и возвращает при этом производную либо смешанную частную производную от выражения заданных порядков по заданным переменным. К примеру, diff(f, x, 3) означает d^3f/dx^3, а diff(f, x, 1, y, 2, z, 1) — $d^4f/dxdy^2dz$. Единственный флаг, имеющий прямое отношение к самой функции diff — это флаг derivabbrev, который влияет на отображение производных в ячейках вывода Maxima. По умолчанию он равен false, и производные обозначаются в виде дробей с буквой d; если же его выставить в true, производные будут отображаться в сокращенном виде, с переменными дифференцирования записанными в виде индексов:

```
(%i1) diff(f(x)^2,x,4)
```

$$(\%o1)\quad 2f(x)\left(\frac{d^4}{dx^4}f(x)\right)+8\left(\frac{d}{dx}f(x)\right)\left(\frac{d^3}{dx^3}f(x)\right)+6\left(\frac{d^2}{dx^2}f(x)\right)^2$$

```
(%i2) derivabbrev: true$ '%i1
```

$$(\%o3)\quad 2f(x)\left(f(x)\right)_{xxxx}+8\left(f(x)\right)_x\left(f(x)\right)_{xxx}+6\left(\left(f(x)\right)_{xx}\right)^2$$

Кроме того, функция **diff** используется еще и для обозначения производных в дифференциальных уравнениях. Но об этом чуть позже, а сейчас перейдем к интегрированию.

Основная функция интегрирования называется **integrate** и имеет два варианта вызова: для нахождения неопределенного и определенного интегралов. Первый выглядит как integrate(выражение, переменная), второй — как integrate(выражение, переменная, нижний-предел, верхний-предел):

```
(%i1) integrate(sqrt((e^x-1)/(e^x+1)),x)
```

$$(\%o1)\quad \log\left(2\sqrt{e^{2x}-1}+2e^x\right)+\arcsin\left(e^{-x}\right)$$

```
(%i2) integrate(cos(x)*cos(2*x)*cos(3*x),x)
```

$$(\%o2)\quad \frac{\sin(6x)}{24}+\frac{\sin(4x)}{16}+\frac{\sin(2x)}{8}+\frac{x}{4}$$

```
(%i3) integrate((x^(2*n-1))/(x^n+1),x)
```

$$(\%o3)\quad \frac{e^{n\log(x)}}{n}-\frac{\log\left(e^{n\log(x)}+1\right)}{n}$$

```
(%i4) integrate(x^2*sqrt(a^2-x^2),x,0,a)
Is a positive, negative, or zero?p
```

$$(\%o4)\quad \frac{\pi a^4}{16}$$

Обратите внимание еще на один момент в ячейках %i4–%o4. Когда в выражении используется какой-либо независимый символ, результат, вообще говоря, может зависеть от значения этого символа. Если при этом о возможных значениях символа ничего не известно, то Maxima задаст вам один или несколько вопросов об этом значении, и решение будет искать в зависимости от ваших ответов на них. Так, в этом примере значение определенного интеграла напрямую зависит от знака параметра a:

```
(%i5) integrate(x^2*sqrt(a^2-x^2),x,0,a)
Is a positive, negative, or zero?n
```

$$(\%o5)\quad -\frac{\pi\, a^4}{16}$$

Кроме обычных определенных интегралов Maxima умеет искать также и несобственные интегралы, то есть такие, у которых неограничена либо область интегрирования, либо подынтегральная функция; и делается это все той же функцией integrate:

```
(%i1) integrate(1/(1+exp(x)),x,1,inf)
(%o1) log(exp(1)+1)-1
(%i2) integrate(1/sqrt(x),x,0,1)
(%o2) 2
```

В случае, если искомый интеграл не сходится, будет выдано сообщение об ошибке, говорящее о том, что интеграл расходящийся:

```
(%i3) integrate(1/x^2,x,0,1)

Integral is divergent
 -- an error.  Quitting.  To debug this try debugmode(true);
```

В случае, если интеграл не может быть найден, он либо целиком возвращается в несовершенном виде, либо упрощается частично и на выходе получается некоторая формула, включающая в несовершенном виде интеграл той части подынтегрального выражения, которую проинтегрировать не удалось:

```
(%i1) integrate(1/(x^4-4*x^3+2*x^2-7*x-4),x)
```

$$(\%o1)\quad \frac{\log(x-4)}{73}-\frac{\int \frac{x^2+4x+18}{x^3+2x+1}\,\mathrm{d}x}{73}$$

Кроме функций diff и integrate, в Maxima есть еще много разнообразных возможностей, связанных с производными и интегралами, в частности, функции для численного расчета значений определенных интегралов, а также инструменты,

применимые при работе с дифференциальными и интегральными уравнениями. А более тезисно, хотя и на английском языке, они описаны в документации.

Для нахождения производной используется функция ***diff****, первым аргументом которой является функция, вторым - переменная, по которой производится дифференцирование, и третьим (необязательным) - порядок производной:*

```
(%i1)  f:(x-2*sqrt(x))/x^2;
```

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

```
(%i2)  diff(f, x);
```

$$(\%o2)\quad \frac{1-\frac{1}{\sqrt{x}}}{x^2}-\frac{2\left(x-2\sqrt{x}\right)}{x^3}$$

```
(%i3)  expand(''%i2);
```

$$(\%o3)\quad \frac{3}{x^{5/2}}-\frac{1}{x^2}$$

```
(%i4)  g:x^6;
```

$$(\%o4)\quad x^6$$

```
(%i5)  diff(g, x, 1);
```

$$(\%o5)\quad 6\,x^5$$

```
(%i6)  diff(g, x, 4);
```

$$(\%o6)\quad 360\,x^2$$

При вычислении кратных производных по нескольким переменным после указания функции перечисляются переменные дифференцирования с указанием соответствующих кратностей, например,

```
(%i7)  diff(x^6*y^3, x, 4, y, 2);
```

$$(\%o7)\quad 2160\,x^2\,y$$

Функция **integrate** позволяет вычислять интегралы. Для нахождения неопределенного интеграла после функции указывается единственный аргумент - переменная интегрирования:

```
(%i8)  f:x^2/(4*x^6+1);
                    2
                   x
(%o8)  ---------------
                6
               4 x  + 1
```

```
(%i9)  integrate(f, x);
                 3
            atan(2 x )
(%o9)    ------------
                 6
```

Maxima в случае неоднозначного ответа может задавать дополнительные вопросы, как в следующем примере:

```
(%i10)integrate(x^n,x);
Is  n + 1  zero or nonzero?
nonzero;
                                n + 1
                               x
(%o10)                        ------
                               n + 1
(%i11)integrate(x^n,x);
Is  n + 1  zero or nonzero?
zero;
(%o11)                          LOG(x)
```

Можно использовать функцию **assume** для задания дополнительных условий (не забывайте затем удалить наложенные ограничения):

```
(%i12)assume(notequal(n,-1));
(%o11)                        [NOT EQUAL(n, - 1)]
(%i13)integrate(x^n,x);
                               n + 1
                              x
(%o13)                       ------
                               n + 1
(%i14)forget(notequal(n,-1));
(%o14)                        [NOT EQUAL(n, - 1)]
(%i15)integrate(x^n,x);
Is  n + 1  zero or nonzero?

zero;
(%o15)                                LOG(x)
```

Для нахождения определенного интеграла следует указать дополнительные агрументы - пределы интегрирования:

```
(%i16)integrate(x^2, x, 0, 6);
(%o16)                                      72

(%i17)integrate(sin(x), x, 0, %pi);
(%o17)                                      2

(%i18)integrate(integrate(x*y, x, 1, 3), y, 0, 4);
(%o18)                                     32
```

Maxima допускает задание и бесконечных пределов интегрирования. Для обозначения бесконечности используется переменная `INF` (`inf`):

```
(%i19)integrate(1/x^2, x, 1, inf);
(%o19)                           1

(%i20)integrate(1/(1+x^2), x, -inf, inf);
(%o20)                                    %pi

(%i21)integrate(1/x, x, 0, inf);

Integral is divergent  -- an error.
Quitting. To debug this try DEBUGMODE(TRUE);)
```

В последнем примере система сообщила о невозможности вычисления интеграла, т. к. он расходится (is divergent).

При вычислении достаточно сложных интегралов ответ не всегда будет представлен в наиболее простом виде. В следующем примере Maxima не может в символьном виде получить ответ, равный PI/4:

```
(%i22)g:1/sqrt(2-x^2);
                                 1
(%o22)                      ------------
                                      2
                             SQRT(2 - x )

(%i23)integrate(g,x, 0,1);
                                  SQRT(2)
(%o23)                           ASIN(-------)
                                  2
```

Перечисленные выше операции математического анализа могут быть вычислены с помощью меню **Анализ** и далее соответствующие команды меню (Integrate), (Differentiate), (Find Limit).

Пример.

Исследуем на непрерывность функцию arctg(1/(x-4)). Эта функция не определена в точке x = 4. Вычислим пределы справа и слева:

```
(%i5)  limit(atan(1/(x-4)), x, 4, plus);
(%o5)  %pi
       ---
        2

(%i6)  limit(atan(1/(x-4)), x, 4, minus);
(%o6)  - %pi
         ---
          2
```

Как видим, точка x = 4 является точкой разрыва I рода для данной функции, так как существуют пределы слева и справа, равные -PI/2 и PI/2 соответственно.

Задания

1. Вычислите первую производную функции $tg^2(x^4 - 2)$.

2. Найдите предел при x -> 0 функции (3x - sin x)/tg 2x.

3. Найдите одну из первообразных функции $\cos^2 x$.

Матричные вычисления

Maxima позволяет легко манипулировать матрицами. В следующем примере задаются две матрицы, которые затем складываются (+) и перемножаются (.):

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

```
(%i1)
(%o1)  [ 1  2 ]
       [      ]
       [ 3  4 ]
```

```
(%i2)  B:matrix([1,1],[1,1]);
(%o2)  [ 1  1 ]
       [      ]
       [ 1  1 ]
```

```
(%i3)  A + B;
(%o3)  [ 2  3 ]
       [      ]
       [ 4  5 ]
```

```
(%i4)  A . B;
(%o4)  [ 3  3 ]
       [      ]
       [ 7  7 ]
```

Используя меню Maxima, переходим к кнопке Алгебра →Enter Matrix, и заполняем необходимые поля. С помощью меню можно вычислить определитель ранее введенной матрицы, вычислить обратную матрицу (Invert Matrix), транспонированную матрицу (Transpose Matrix), найти собственные числа (eigenvalues) и собственные значения (eigenvectors).

Функция **determinant** вычисляет определитель матрицы.

```
(%i7)  determinant(A);
(%o7)  -2
```

```
(%i8)  determinant(matrix([a,b],[c,d]));
(%o8)  a d - b c
```

Вычисление ранга матрицы

```
(%i6)  rank(%);
(%o6)  3
```

Функция **minor** вычисляет минор матрицы, первый элемент – матрица, далее индекс строки и столбца соответственно.

```
(%i27) minor(%,1,1);
(%o27)
```

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

submatrix – вычисляет матрицу, полученную путем удаления соответствующих строк и столбцов, где в качестве параметров выписывают номера удаляемых строк, имя матрицы, номера удаляемых столбцов.

Транспонирование матрицы осуществляется функцией **transpose**.

```
(%i9)  transpose(A);
(%o9)
```

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

Для получению обратной матрицы используется операция ^^-1 или функция **invert**.

```
(%i10)  A^^-1;
(%o10)
```

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

```
(%i11)  invert(A);
(%o11)
```

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

Как известно, каждый элемент b^{ij} обратной матрицы $B = A^{-1}$ получается делением алгебраического дополнения A^{ij} соответствующего элемента исходной матрицы на ее определитель |A|. Для того чтобы вынести 1/|A| в качестве сомножителя применяется функция **detout**.

```
(%i12)  invert(A), detout;
(%o12)
```

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

Убедимся в правильности полученного результата, умножив A на обратную к ней матрицу:

```
(%i13) A.A^^-1
```

Будьте внимательны: в результате выполнения операции ^-1 получится матрица, каждый элемент которой обратен элементу исходной, а не обратная матрица.

```
(%i14) A^-1;
```

$$(\%o14)\quad \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{4} \end{bmatrix}$$

Использование матриц позволяет легко решать системы линейных уравнений с несколькими переменными. Пусть A - матрица коэффициентов системы, X - матрица неизвестных, B - матрица свободных членов системы. Тогда матрица X находится по формуле $X = A^{-1} . B$, где операция . означает матричное умножение.

Пример.
Решим следующую систему уравнений матричным способом.

$$\begin{cases} x + 2y + z &= 0, \\ 2x + y + z &= 1, \\ x + 3y + z &= 0. \end{cases}$$

Сначала заполним соответствующие матрицы, а затем получим матрицу результатов:

```
(%i15) A:matrix([1, 2, 1], [2, 1, 1], [1, 3, 1]);
```

$$(\%o15)\quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 1 & 3 & 1 \end{bmatrix}$$

```
(%i16) B:matrix([0],[1],[0]);
```

$$(\%o16)\quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

```
(%i17)  (A^^-1).B;
                  [ 1 ]
                  [   ]
(%o17)            [ 0 ]
                  [   ]
                  [-1 ]
```

Задания.

1. Вычислите произведение матриц A.B и B.A, где

$$A = \begin{pmatrix} 2 & 4 & 0 \\ 2 & 0 & 4 \\ 1 & 2 & 3 \end{pmatrix}; \qquad B = \begin{pmatrix} 2 & 1 & 0 \\ 1 & -1 & 2 \\ 3 & 2 & 1 \end{pmatrix}.$$

2. Найдите определители матриц C и D.

$$C = \begin{pmatrix} 4 & 2 & 1 & 4 \\ 1 & -2 & 0 & 3 \\ -2 & -3 & 2 & 1 \\ 3 & 2 & 0 & 1 \end{pmatrix}; \qquad D = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}.$$

3. Для матрицы D найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.
4. Решите следующую систему уравнений матричным способом

$$\begin{cases} 3x + 2y + 2z = 5, \\ x + 3y + z = 0, \\ 5x + 3y + 4z = 10. \end{cases}$$

Задания для самостоятельной работы

Вариант №1

4. Вычислите первую производную функции $y = \frac{1}{6}x^6 - \frac{2}{5}x^5 + \frac{5}{3}x^3 + 2x + 7$

5. Найдите $\lim\limits_{x \to \pi} \frac{\sin^2 x}{1 + \cos^3 x}$

6. Найдите $\int \frac{x dx}{(x-1)(x+1)^2}$.

7. Вычислите произведение матриц A.B и B.A, где

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 1 & -4 \\ 7 & -2 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 3 & 1 \\ 2 & 8 & -2 \\ 6 & -7 & 4 \end{pmatrix}$$

8. Найдите определители матриц A и B

9. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

10. Решите следующую систему уравнений матричным способом

$$\begin{cases} x-2y+4z=2 \\ x-y+3z=1 \\ x+3y-2z=0 \end{cases}$$

11. Построить транспонированную матрицу к матрице B.

12. Найти спектр и собственные векторы матрицы B

13. Найти ранг и минор M_{13} матрицы A.

Вариант №2

1. Вычислите первую производную функции $y = ctg^3\sqrt{x^4+5}$

2. Найдите $\lim\limits_{x\to 0}\dfrac{\ln(1+4x)}{x}$

3. Найдите $\int 9x\sin x dx$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix} 3 & 2 & 1 \\ 2 & 1 & -5 \\ 7 & -8 & 4 \end{pmatrix},\ B=\begin{pmatrix} 7 & 7 & 2 \\ 4 & 4 & -2 \\ 2 & -1 & 4 \end{pmatrix}$$

5. Найдите определители матриц A и B

6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

7. Решите следующую систему уравнений матричным способом

$$\begin{cases} x-y+2z=4 \\ x-y+3z=1 \\ x+3y-2z=0 \end{cases}$$

8. Построить транспонированную матрицу к матрице B.

9. Найти спектр и собственные векторы матрицы B

10. Найти ранг и минор M_{13} матрицы A.

Вариант №3

1. Вычислите первую производную функции $y = x^3\cos^2 x^5$

2. Найдите $\lim\limits_{x\to 0}\dfrac{arctgx}{x}$

3. Найдите $\int x^{10}\cos^3 x dx$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix}2 & 2 & 6\\ 3 & 0 & -4\\ 5 & -2 & 4\end{pmatrix},\quad B=\begin{pmatrix}5 & 3 & 1\\ 0 & 2 & -2\\ 7 & -7 & 1\end{pmatrix}$$

5. Найдите определители матриц A и B
6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.
7. Решите следующую систему уравнений матричным способом
$$\begin{cases}x-y+3z=2\\ x-2y+3z=6\\ x+2y-2z=3\end{cases}$$
8. Построить транспонированную матрицу к матрице B.
9. Найти спектр и собственные векторы матрицы B
10. Найти ранг и минор M_{13} матрицы A.

Вариант №4

1. Вычислите первую производную функции $y=\dfrac{\sin^3 x}{e^x}$
2. Найдите $\lim\limits_{x\to 0}\dfrac{arctgx}{x}$
3. Найдите $\int\dfrac{dx}{\sin^3 x}$.
4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix}5 & 2 & -3\\ 2 & 9 & -1\\ 3 & -2 & 4\end{pmatrix},\quad B=\begin{pmatrix}2 & 3 & 11\\ 6 & 4 & -23\\ 0 & -4 & 4\end{pmatrix}$$

5. Найдите определители матриц A и B
6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.
7. Решите следующую систему уравнений матричным способом
$$\begin{cases}x-y+4z=2\\ x-2y+3z=1\\ x+3y-2z=0\end{cases}$$
8. Построить транспонированную матрицу к матрице B.
9. Найти спектр и собственные векторы матрицы B
10. Найти ранг и минор M_{13} матрицы A.

Вариант №5

1. Вычислите первую производную функции $y=\arccos x+e^{x^2}$
2. Найдите $\lim\limits_{x\to 0}\dfrac{\sin^3\frac{x}{2}}{x^3}$

3. Найдите одну $\int \frac{dx}{\cos^4 x}$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix} 1 & 3 & 0 \\ 2 & 1 & -4 \\ 7 & -2 & 4 \end{pmatrix},\ B=\begin{pmatrix} 2 & 3 & 1 \\ 2 & 8 & -2 \\ 6 & -7 & 4 \end{pmatrix}$$

5. Найдите определители матриц A и B

6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

7. Решите следующую систему уравнений матричным способом

$$\begin{cases} x-2y+4z=2 \\ x-y+3z=1 \\ x+3y-2z=0 \end{cases}$$

8. Построить транспонированную матрицу к матрице B.

9. Найти спектр и собственные векторы матрицы B

10. Найти ранг и минор M_{13} матрицы A.

Вариант №6

1. Вычислите первую производную функции $y=\sqrt{1+e^{2x}}\cdot \arccos x$

2. Найдите $\lim\limits_{x\to 0}\dfrac{\sin(x-1)}{x^3-1}$

3. Найдите $\int \dfrac{dx}{3+\sin x+\cos x}$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix} 3 & 3 & 7 \\ 2 & 11 & -4 \\ 5 & -2 & 4 \end{pmatrix},\ B=\begin{pmatrix} 2 & 3 & 10 \\ 1 & 5 & -2 \\ 6 & -3 & 4 \end{pmatrix}$$

5. Найдите определители матриц A и B

6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

7. Решите следующую систему уравнений матричным способом

$$\begin{cases} x-2y-z=-1 \\ x-y+3z=1 \\ x+4y-2z=8 \end{cases}$$

8. Построить транспонированную матрицу к матрице B.

9. Найти спектр и собственные векторы матрицы B

10. Найти ранг и минор M_{13} матрицы A.

Вариант №7

1. Вычислите первую производную функции $y = arctg\sqrt{x^2+1}$

2. Найдите $\lim\limits_{x\to\pi}\dfrac{\sin^2 x}{1+\cos^3 x}$

3. Найдите $\int\limits_0^{\frac{\pi}{4}}\dfrac{dx}{5\cos^2 x+9\sin^2 x}$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix}6 & 3 & 12\\ 2 & 2 & -8\\ 2 & -2 & 4\end{pmatrix},\ B=\begin{pmatrix}2 & 3 & 15\\ 2 & 10 & -2\\ 1 & -7 & 4\end{pmatrix}$$

5. Найдите определители матриц A и B

6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

7. Решите следующую систему уравнений матричным способом

$$\begin{cases}2x-2y+z=2\\ x-y+z=1\\ x+2y-2z=7\end{cases}$$

8. Построить транспонированную матрицу к матрице B.

9. Найти спектр и собственные векторы матрицы B

10. Найти ранг и минор M_{13} матрицы A.

Вариант №8

1. Вычислите первую производную функции $y=\sqrt{\dfrac{3\sin x-2\cos x}{5}}$

2. Найдите $\lim\limits_{x\to 0}\dfrac{1-\cos x-tg^2 x}{x\sin x}$

3. Найдите $\pi\int\limits_0^a\left(a^2-x^2\right)dx$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix}1 & 3 & 0\\ 2 & 1 & -4\\ 7 & -2 & 4\end{pmatrix},\ B=\begin{pmatrix}2 & 3 & 4\\ 5 & 3 & -2\\ 6 & -7 & 4\end{pmatrix}$$

5. Найдите определители матриц A и B
6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.
7. Решите следующую систему уравнений матричным способом

$$\begin{cases}x-y+4z=-2\\ x-y+3z=-11\\ x+3y-2z=7\end{cases}$$

8. Построить транспонированную матрицу к матрице B.
9. Найти спектр и собственные векторы матрицы B
10. Найти ранг и минор M_{13} матрицы A.

Вариант №9

1. Вычислите первую производную функции $y = \sqrt{\sin^2 x + 5}$

2. Найдите $\lim\limits_{x\to 0}\left(\dfrac{\sin\frac{x}{2}}{x}\right)^{x+3}$

3. Найдите $\int\limits_0^1 xe^{-x}dx$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix}1 & 3 & 7\\ 2 & 3 & -4\\ 1 & -2 & 4\end{pmatrix},\ B=\begin{pmatrix}3 & 3 & 1\\ 4 & 9 & -2\\ 6 & -7 & 1\end{pmatrix}$$

5. Найдите определители матриц A и B

6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

7. Решите следующую систему уравнений матричным способом

$$\begin{cases}x-2y+4z=-2\\ 2x-y+3z=3\\ x+3y-2z=5\end{cases}$$

8. Построить транспонированную матрицу к матрице B.

9. Найти спектр и собственные векторы матрицы B

10. Найти ранг и минор M_{13} матрицы A.

Вариант №10

1. Вычислите первую производную функции $y = \sqrt{1 + \arcsin x}$

2. Найдите $\lim\limits_{x\to 0}\dfrac{\sin x}{\sqrt{x+9}-3}$

3. Найдите $\int\limits_0^{\frac{\pi}{2}} \sin^3 x dx$.

4. Вычислите произведение матриц A.B и B.A, где

$$A=\begin{pmatrix}7 & 3 & 1\\ 12 & 22 & -6\\ 2 & -2 & 3\end{pmatrix},\ B=\begin{pmatrix}12 & 3 & 1\\ 2 & 17 & -12\\ 1 & -7 & 4\end{pmatrix}$$

5. Найдите определители матриц A и B

6. Для матрицы B найдите обратную, после чего проверьте, что в результате их произведения получается единичная матрица.

7. Решите следующую систему уравнений матричным способом

$$\begin{cases} x-3y+z=2 \\ x-2y+z=8 \\ x+2y-z=4 \end{cases}$$

8. Построить транспонированную матрицу к матрице B.

9. Найти спектр и собственные векторы матрицы B

10. Найти ранг и минор M_{13} матрицы A.

Нахождение суммы ряда. Разложение в ряд Тейлора

Для вычисления конечных и бесконечных сумм следует записать сумму в символьном виде, после чего упростить полученное выражение:

```
(%i1)  sum(1/n^2,n,1,inf);
```

$$(\%o1)\quad \sum_{n=1}^{\infty} \frac{1}{n^2}$$

```
(%i2)  %,simpsum;
```

$$(\%o2)\quad \frac{\%pi^2}{6}$$

Maxima может находить разложение функций в ряд Тейлора.

Пример. Разложить в ряд Тейлора функцию g(x)=ln x в точке x=1. Получим многочлен Тейлора четвертого порядка в точке x=1:

```
(%i3)  g:log(x);
(%o3)  log(x)
```

```
(%i4)  taylor(g,x,1,4);
```

$$(\%o4)\quad x-1-\frac{(x-1)^2}{2}+\frac{(x-1)^3}{3}-\frac{(x-1)^4}{4}+\ldots$$

Формат задания: **taylor** *(expr, [x_1, x_2, ...], a, n)* (**taylor**(функция, переменные, точка *x=a, до какой степени параметра разложения вычисляется ряд*).
taylor *(expr, [x_1, x_2, ...], [a_1, a_2, ...], [n_1, n_2, ...])* ,
taylor *(expr, [x_1, a_1, n_1], [x_2, a_2, n_2], ...)* – разложение в ряд Тейлора функции нескольких переменных в окрестностях точек *x1=a1, x2=a2,.., до порядка n1,n2,...*

Примеры.

```
(%i5) taylor (sqrt (x + 1), x, 0, 5);
                       2    3      4       5
                   x   x    x    5 x    7 x
(%o5)/T/      1 + - - -- + -- - ---- + ---- + . . .
                   2   8    16   128    256
```

```
(%i6) taylor (sin (y + x), x, 0, 3, y, 0, 3);
                3                   2
               y                   y
(%o6)/T/ y - -- + . . . + (1 - -- + . . .) x
               6                   2

                     3                          2
                y   y              2       1   y              3
          + (- - + -- + . . .) x  + (- - + -- + . . .) x  + . . .
                2   12                     6   12
(%i7) taylor (sin (y + x), [x, y], 0, 3);
                     3        2      2       3
                    x  + 3 y x  + 3 y  x + y
(%o8)/T/   y + x - ------------------------ + . . .
                                6
```

Решение уравнений

Уравнения и системы уравнений решаются в Maxima функцией solve. Но прежде чем рассмотреть ее подробнее, рассмотрим списки, или векторы в Maxima; поскольку именно в виде списков solve возвращает корни, да и принимает параметры в случае решения системы уравнений, а не одного уравнения.

Синтаксис списков в Maxima весьма прост; это перечисление элементов в квадратных скобках: [элемент1, элемент2, …, элементN]. Особенность — не в синтаксисе. Основное достоинство списков в том, что их элементами могут быть совершенно любые выражения: символы, арифметические выражения, вызовы функций, присвоения, уравнения, другие списки. Функция solve в своем простейшем варианте, для решения одиночного уравнения, в качестве аргументов никаких списков не принимает (а принимает либо уравнение и символ, относительно которого его надо решать, либо только уравнение, если символ в нем всего один). А вот в качестве результата она уже и в таком варианте возвращает список, состоящий из всех корней заданного уравнения:

```
(%i1)  eq: (x+1)/(x^2+1)=x^2/(x+2)$solve(eq);
```

$$(\%o2)\quad \left[x=-\frac{\sqrt{5}-1}{2},\ x=\frac{\sqrt{5}+1}{2},\ x=-\frac{\sqrt{7}\,\%i+1}{2},\ x=\frac{\sqrt{7}\,\%i-1}{2}\right]$$

Как видите, функция solve находит все комплексные корни уравнения, а не только действительные.

К элементу списка можно обратиться с помощью тех же квадратных скобок, указав в них номер элемента после имени списка. Напомню, что равенство, переданное в качестве

дополнительного параметра функции ev, означает подстановку переменной в вычисляемое выражение. Вот так мы можем осуществить проверку решения, подставив корень из выданного списка в исходное уравнение:

```
(%i3) eq,%[1]
```

$$(\%o3)\quad \frac{1-\frac{\sqrt{5}-1}{2}}{\frac{\left(\sqrt{5}-1\right)^2}{4}+1}=\frac{\left(\sqrt{5}-1\right)^2}{4\left(2-\frac{\sqrt{5}-1}{2}\right)}$$

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

$$(\%o4)\quad \frac{\sqrt{5}-3}{\sqrt{5}-5}=\frac{\sqrt{5}-3}{\sqrt{5}-5}$$

Точно таким же образом можно обратиться и к любому другому элементу списка:

```
(%i5) eq,%o2[2],ratsimp;eq,%o2[3],ratsimp;eq,%o2[4],ratsimp
```

$$(\%o5)\quad \frac{\sqrt{5}+3}{\sqrt{5}+5}=\frac{\sqrt{5}+3}{\sqrt{5}+5}$$

$$(\%o6)\quad -1=-1$$

$$(\%o7)\quad -1=-1$$

Вообще говоря, в качестве первого аргумента функции solve можно задавать не только уравнение, а вообще любое выражение. При этом «корни выражения» (не являющегося уравнением) ищутся в том самом смысле, в каком эта фраза понимается в математике: корни выражения — это те значения переменной, на которых выражение обращается в ноль. Возможность такой записи позволяет, к примеру, легко найти критические точки любой непрерывной функции (а заодно и вычислить значения функции в этих точках):

```
(%i1) f:(1+x-x^2)/(1+x+x^2)$
(%i2) solve(diff(f,x))
```

$$(\%o2)\quad [x=-2,x=0]$$

```
(%i3) f,%[1];f,%th(2)[2]
```

$$(\%o3)\quad -\frac{5}{3}$$

$$(\%o4)\quad 1$$

В этом примере есть еще два важных момента. Первый — функция %th(). Она, как видно из контекста, вызывается как %th(n) и возвращает *n*-ю с конца ячейку вывода. Это, так же как и обозначения % и _, удобно, чтобы не обращать внимания на номера ячеек, и кроме того, применимо в командных файлах Maxima, которые могут загружаться в том числе и прямо из интерактивной сессии (с помощью функции load) — и тогда просто заранее неизвестно, начиная с какой ячейки данный файл загружен. И второй момент: здесь проиллюстрировано, что в Maxima операция индексирования списка доступна не

только по отношению к именам переменных, но и к вызовам функций; другими словами, если функция возвращает список значений, мы можем выбрать одно конкретное из них, написав его номер в квадратных скобках прямо после вызова функции.

Вернемся к функции solve. А именно, перейдем теперь к решению систем уравнений. Для этого существует такой вариант записи: solve([уравнение1, уравнение2, …], [переменная1, переменная2, …]); либо сокращенный, аналогично варианту для одиночного уравнения: если количество уравнений и количество неизвестных равны, список неизвестных можно не писать: solve([уравнение1, уравнение2 …]) (не забудьте квадратные скобки, иначе Maxima примет его за вариант с одним уравнением).

(%i1) $\mathrm{solve}\left(\left[x^2+y^2=2, x+y=1\right]\right)$

(%o1) $\left[\left[y=-\frac{\sqrt{3}-1}{2}, x=\frac{\sqrt{3}+1}{2}\right], \left[y=\frac{\sqrt{3}+1}{2}, x=-\frac{\sqrt{3}-1}{2}\right]\right]$

Здесь возвращается список из нескольких списков, каждый из которых соответствует одному решению системы. В качестве подстановок можно использовать как такие списки целиком (например, в данном контексте, %o1[1]), так и отдельные их элементы (например, %o1[1][1]).

В случае, когда уравнений меньше, чем неизвестных, solve поступит точно так же, как и в случае одного уравнения с несколькими символами: все неуказанные будет воспринимать как параметры:

(%i1) $\mathrm{solve}\left(\left[x^2+y^2=a^2, x+y=2a+1\right], [x, y]\right)$

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

Если solve не находит точных решений, она может, как и integrate, вернуть уравнение или систему уравнений в некотором упрощенном виде, а может и самостоятельно попытаться решить систему численно:

```
(%i1) eqs: [4 x^2 - y^2 = 12, x y - x = 2]$
(%i2) solve(eqs)
(%o2) [[y = 2, x = 2], [y = - 0.15356757100197, x = - 1.733751846381093], [y = 3.60800322187\
0287 i + 0.076783785237878, x = - 0.5202594388652 i - 0.13312403573587], [y = 0.07678378523\
7878 - 3.608003221870287 i, x = 0.5202594388652 i - 0.13312403573587]]
```

В таком случае, если вам все же нужны точные значения корней (в аналитической записи), либо если они не найдены даже в числах, можно попробовать решить уравнения по очереди, выражая одно неизвестное через другое:

```
(%i3) solve(eqs[2], y)
```

$$(\%o3)\quad \left[y = \frac{x+2}{x}\right]$$

```
(%i4) eqs[1], %
```

$$(\%o4)\quad 4\,x^2 - \frac{(x+2)^2}{x^2} = 12$$

```
(%i5) solve(%)
```

И подставляя в оставшиеся уравнения:

```
(%i5) solve(%)
```

$$(\%o5)\quad \left[x = \frac{7\left(\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right)}{36\left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}} + \left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}\left(-\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right) - \frac{2}{3}, x = \left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}\left(\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right) + \frac{7\left(-\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right)}{36\left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}} - \frac{2}{3}, x = \left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}} + \frac{7}{36\left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}} - \frac{2}{3}, x = 2\right]$$

Теперь можем подставить обратно — и найти значения второй неизвестной, например, для первого и последнего корней из последнего списка:

```
(%i6) %o3[1], %[1]; %o3[1], %th(2)[4]
```

$$(\%o6)\quad y = \frac{\frac{7\left(\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right)}{36\left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}} + \left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}\left(-\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right) + \frac{4}{3}}{\frac{7\left(\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right)}{36\left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}} + \left(\frac{\sqrt{139}}{24\sqrt{3}} - \frac{8}{27}\right)^{\frac{1}{3}}\left(-\frac{\sqrt{3}\,\mathrm{i}}{2} - \frac{1}{2}\right) - \frac{2}{3}}$$

$$(\%o7)\quad y = 2$$

Функция solve имеет довольно большое количество различных переключателей, из которых может пригодиться в своем не-умолчательном значении в первую очередь один: solveradcan. Умолчание здесь равно false, а выставив этот флаг в true, мы заставим solve, помимо его умолчательного поведения, применять radcan — функцию по упрощению показательных, логарифмических и степенных (с рациональными степенями) функций. Это делает работу функции solve более медленной (потому по умолчанию этот режим и выключен), но в некоторых случаях может помочь разрешить проблемы, которые без этого ключа приведут к невозможности найти точное решение.

Maxima может решать уравнения и системы алгебраических уравнений с помощью функции ***solve****. Равная нулю правая часть уравнения может быть опущена*:

```
(%i1) solve (x^2=1, x);
(%o1)                                [x = - 1, x = 1]

(%i2) solve(x^2-1,x);
(%o2)                                [x = - 1, x = 1]
```

```
(%i3) solve(log(x+3)=1, x);
(%o3)                                [x=%e-3]
```

При решении тригонометрических уравнений выдается только одно из бесконечного множества возможных решений:

```
(%i4) solve(sin(x)-1, x);
SOLVE is using arc-trig functions to get
a solution. Some solutions will be lost.
(%o4)                       [x=%pi/2]
```

В следующем примере функция solve используется для решения системы из трех уравнений с тремя неизвестными:

```
(%i5) s:[x+y+z=3, x+2*y-z=2, x+y*z+z*x=3];
(%o5)             [z + y + x = 3, - z + 2 y + x = 2,
              y z + x z + x = 3]
(%i6) solve(s, [x,y,z]);
(%o6)               [[x = 1, y = 1, z = 1],
                [x = 7, y = - 3, z = - 1]]
```

Если уравнение не имеет решений на множестве действительных чисел, то Maxima ищет решения среди комплексных чисел:

```
(%i7) solve(x^2+1,x);
(%o7)                              [x = - %i, x = %i]
```

Задание.

Решите уравнение *ln(tgx)=0.*

Замечание. В Maxima можно находить приближенные значения корней алгебраических уравнений используя команды меню Maxima или встроенные функции **allroots** *(expr),* **realroots** *(expr, bound),* **find_root** *(expr, x, a, b)*

Пример 1. Найти все корни уравнения: *(1 + 2*x)^3 = 13.5*(1 + x^5);*

```
(%i1) eqn: (1 + 2*x)^3 = 13.5*(1 + x^5);
                              3          5
(%o1)                (2 x + 1)  = 13.5 (x  + 1)
(%i2) soln: allroots (eqn);
(%o2) [x = .8296749902129361, x = - 1.015755543828121,

x = .9659625152196369 %i - .4069597231924075,

x = - .9659625152196369 %i - .4069597231924075, x = 1.0]
(%i3) for e in soln
        do (e2: subst (e, eqn), disp (expand (lhs(e2) - rhs(e2))));
                      - 3.5527136788005E-15

                     - 5.32907051820075E-15

          4.44089209850063E-15 %i - 4.88498130835069E-15

         - 4.44089209850063E-15 %i - 4.88498130835069E-15

                       3.5527136788005E-15
```

```
(%o3)                          done
(%i4) polyfactor: true$
(%i5) allroots (eqn);
(%o5) - 13.5 (x - 1.0) (x - .8296749902129361)

                              2
 (x + 1.015755543828121) (x  + .8139194463848151 x

 + 1.098699797110288)
```

Пример 2. Находим только вещественные корни уравнения: *-1 - x + x^5=0*

```
(%i1) realroots (-1 - x + x^5, 5e-6);
                               612003
(%o1)                     [x = ------]
                               524288
(%i2) ev (%[1], float);
(%o2)                  x = 1.167303085327148
(%i3) ev (-1 - x + x^5, %);
(%o3)                - 7.396496210176905E-6
(%i1) realroots (expand ((1 - x)^5 * (2 - x)^3 * (3 - x)), 1e-20);
(%o1)                  [x = 1, x = 2, x = 3]
(%i2) multiplicities;
(%o2)                       [5, 3, 1]
```

Пример 3. Найти значение всех корней уравнения: $\sin x - x = 0$ на отрезке $[0.1, \pi]$

```
(%i1) f(x) := sin(x) - x/2;
                                        x
(%o1)                   f(x) := sin(x) - -
                                        2
(%i2) find_root (sin(x) - x/2, x, 0.1, %pi);
(%o2)                    1.895494267033981
(%i3) find_root (sin(x) = x/2, x, 0.1, %pi);
(%o3)                    1.895494267033981
(%i4) find_root (f(x), x, 0.1, %pi);
(%o4)                    1.895494267033981
(%i5) find_root (f, 0.1, %pi);
(%o5)                    1.895494267033981
(%i6) find_root (exp(x) = y, x, 0, 100);
                             x
(%o6)           find_root(%e  = y, x, 0.0, 100.0)
(%i7) find_root (exp(x) = y, x, 0, 100), y = 10;
(%o7)                    2.302585092994046
(%i8) log (10.0);
(%o8)                    2.302585092994046
```

Решение дифференциальных уравнений

Помимо «просто» уравнений, Maxima позволяет также решать и обыкновенные дифференциальные уравнения первого и второго порядка. Функций, непосредственно занимающихся решением таких уравнений, существует две. Первая из них занимается поиском частных решений линейных дифференциальных уравнений и систем таких

уравнений; зовут ее desolve, от слов *differential equation solve*. Эта функция принимает два аргумента, первый из которых — уравнение либо список уравнений, а второй — соответственно одна переменная или список переменных. Если не заданы значения функций и/или их производных в нуле, то в найденном решении они просто отображаются в виде f(0) или

$$\left.\frac{d}{dx} f(x)\right|_{x=0}$$

задать эти значения позволяет функция *atvalue (выражение, переменная = точка, значение)*; то есть, в данном случае *atvalue(f(x), x=0, значение)* или *atvalue('diff(f(x)), x=0, значение)*. Производные в уравнениях и системах, решаемых с помощью этой функции, должны быть записаны непременно в виде *'diff(f(x), x)*, а не просто *'diff(f, x)*, а сами функции, соответственно, тоже в виде *f(x)*, а не *f* — нужно продемонстрировать зависимость функции от ее аргумента.

```
(%i1) ['diff(f(x),x) = 'diff(g(x),x) + sin(x),
       'diff(g(x),x,2) = 'diff(f(x),x) - cos(x)]
```

$$\left[\frac{d}{dx} f(x) = \frac{d}{dx} g(x) + \sin x, \frac{d^2}{dx^2} g(x) = \frac{d}{dx} f(x) - \cos x\right]$$

```
(%i2) atvalue('diff(g(x),x), x = 0, a)$ atvalue(f(x), x = 0, 1)$
(%i4) desolve(%o1, [f(x), g(x)])
```

$$[f(x) = a\,e^x - a + 1, g(x) = \cos x + a\,e^x - a + g(0) - 1]$$

И конечно же, точно так же как для обычных уравнений и систем, здесь мы тоже можем проверить решение с помощью подстановки, но только надо еще дополнительно задать принудительное вычисление производных, так как в уравнениях они фигурируют в несовершенной форме:

```
(%i5) %o1, %, diff
```

$$[a\,e^x = a\,e^x, a\,e^x - \cos x = a\,e^x - \cos x]$$

Вторая функция из этой группы называется ode2 и предназначена она для решения обыкновенных дифференциальных уравнений первого и второго порядка; ее название происходит от фразы *ordinary differential equations of 1st or 2nd order*. Пишется она так: ode2(уравнение, зависимая-переменная, независимая-переменная). Здесь уже независимая переменная указывается в списке параметров функции явно, и потому обозначения вида y(x) не нужны: и функция, и переменная обозначаются просто одиночными буквами. Также в отличие от предыдущей функции, ode2 ищет не частное, а общее решение. Произвольная константа в решении уравнения первого порядка обозначена через %c; в решении уравнения второго порядка таких констант, естественно, две, и обозначаются они как %k1 и %k2.

(%i1) $x^2\,'\mathrm{diff}(y,x)+3\,y\,x=\frac{\sin(x)}{x}$

(%o1) $x^2\left(\frac{d}{d\,x}\,y\right)+3\,x\,y=\frac{\sin x}{x}$

(%i2) $\mathrm{ode2}(\%,y,x)$

(%o2) $y=\frac{\%\mathrm{c}-\cos x}{x^3}$

(%i3) $'\mathrm{diff}(y,x,2)+y\,'\mathrm{diff}(y,x)^3=0$

(%o3) $\frac{d^2}{d\,x^2}\,y+y\left(\frac{d}{d\,x}\,y\right)^3=0$

(%i4) $\mathrm{ode2}(\%,y,x)$

(%o4) $\frac{y^3+6\,\%\mathrm{k1}\,y}{6}=x+\%\mathrm{k2}$

В дополнение к функции ode2 существуют три функции для поиска частных решений на основе полученных общих. Иначе говоря, эти функции, получая конкретные условия относительно значения функции-решения в заданной точке, находят исходя из этих значений соответствующие им величины интегральных констант. Одна из этих функций предназначена для обработки решения дифференциального уравнения первого порядка. Она называется ic1 (*i* от *initial value* — *начальное значение*; *c* от *constant* — *константа*; *1* от *1st order* — *первого порядка*) и принимает три аргумента: первый — само решение, в том виде, в котором его находит функция ode2; второй — значение независимой переменной (*x*-координаты), третий — значение функции (зависимой переменной, *y*) при этом значении *x* и возвращает частное решение, проходящее через точку с заданными координатами (*x*, *y*):

(%i5) $\mathrm{ic1}(\%o2,x=\pi,y=0)$

(%o5) $y=-\frac{\cos x+1}{x^3}$

Две функции работают с решениями уравнений второго порядка. Так как в общем решении уравнения второго порядка фигурируют две независимые константы, то эти функции задают уже по два условия для поиска частного решения. Первая функция выглядит как ic2 (общее решение, x, функция в точке x, производная в точке x). Расшифровка названия аналогична предыдущей функции. Действует тоже аналогично ей, а в качестве второго условия задает значение производной в той же заданной точке:

(%i8) $\mathrm{bc2}(\%o4,x=0,y=1,x=1,y=3)$

(%o8) $\frac{y^3-10\,y}{6}=x-\frac{3}{2}$

Нахождение общих решений дифференциальных уравнений первого и второго порядка с помощью меню Maxima выполняется посредством задания последовательности кнопок панели Maxima: Уравнения→Solve ODE, частные решения находим далее Initial Value Problem (1), Initial Value Problem (2) Boundary Value Problem.

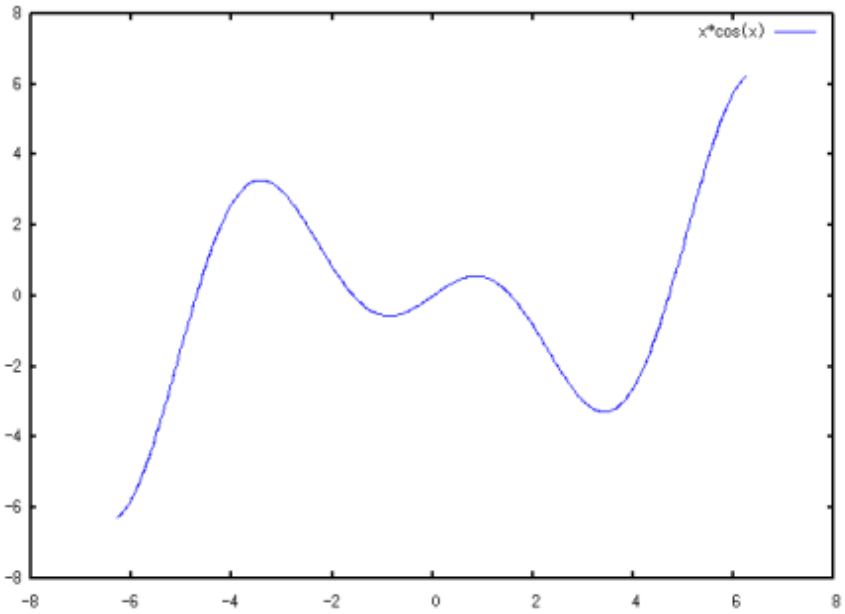
Построение графиков

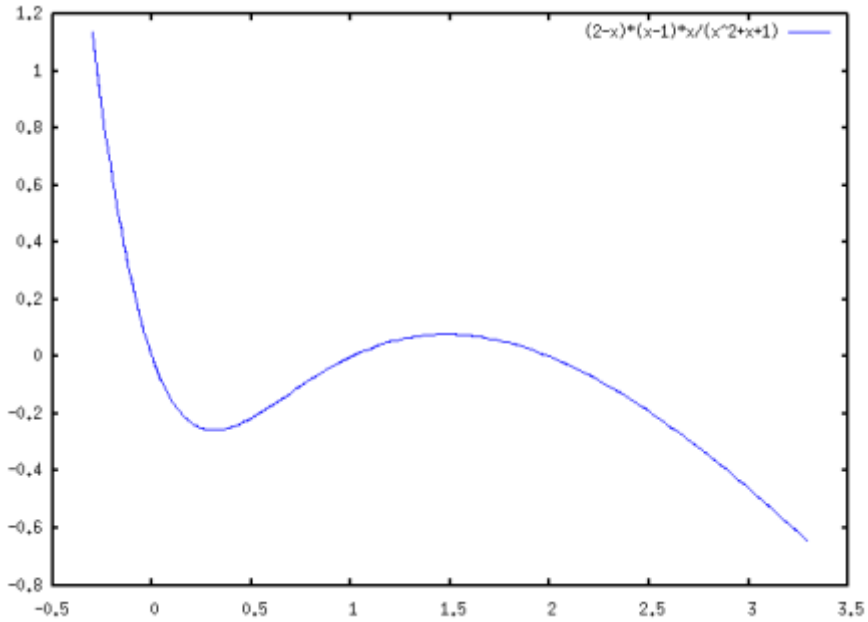
В функциях построения графиков основных таких функций всего две, с названиями — plot2d и plot3d (одно из значений слова *plot — график*, а аббревиатуры *2d* и *3d* переводятся как *двумерный* и *трехмерный*). По умолчанию, построением графиков занимается gnuplot, но кроме него есть разрабатываемый вместе с Maxima и идущий в ее же пакете openmath. Gnuplot необходимо установить (вручную либо автоматически — как зависимость Maxima) из пакета gnuplot-nox, либо просто gnuplot, а для работы openmath нужен командный интерпретатор wish, входящий обычно в пакет tk; и, начиная с версии 5.10.0, еще и xMaxima. Двумерная графика - plot2d. Вызов функции: plot2d(выражение, [символ, начало, конец]), где выражение задает функцию, график которой нужно построить, символ — неизвестное (он, понятное дело, должен быть единственным неопределенным символом, входящим в выражение), а начало и конец задают отрезок оси X для построения графика; участок по оси Y в таком варианте записи выбирается автоматически, исходя из минимума и максимума функции на заданном промежутке. Обратите внимание, что неизвестное и концы промежутка нужно задавать не тремя отдельными параметрами, как, скажем, в integrate, а в виде списка. Это связано с тем, что plot2d может принимать еще и дополнительные аргументы — в таком случае они перечисляются следом за таким списком.

После вызова функции plot2d в таком варианте откроется окно gnuplot, в котором будет отображен затребованный график. Никакой интерактивной работы с полученным изображением gnuplot не предусматривает, кроме автоматического его масштабирования при изменении размеров окна. Можно закрыть окно с графиком клавишей Q, либо, в случае работы с Maxima в редакторе TeXmacs или wxMaxima, просто переключиться обратно в интерфейс, оставив окно gnuplot открытым, и продолжить работу:

(%i1) $\mathrm{plot2d}(x\cos(x),[x,-2\pi,2\pi])$

(%i2) $\mathrm{plot2d}\left(\frac{x(x-1)(2-x)}{x^2+x+1},[x,-0.3,3.3]\right)$

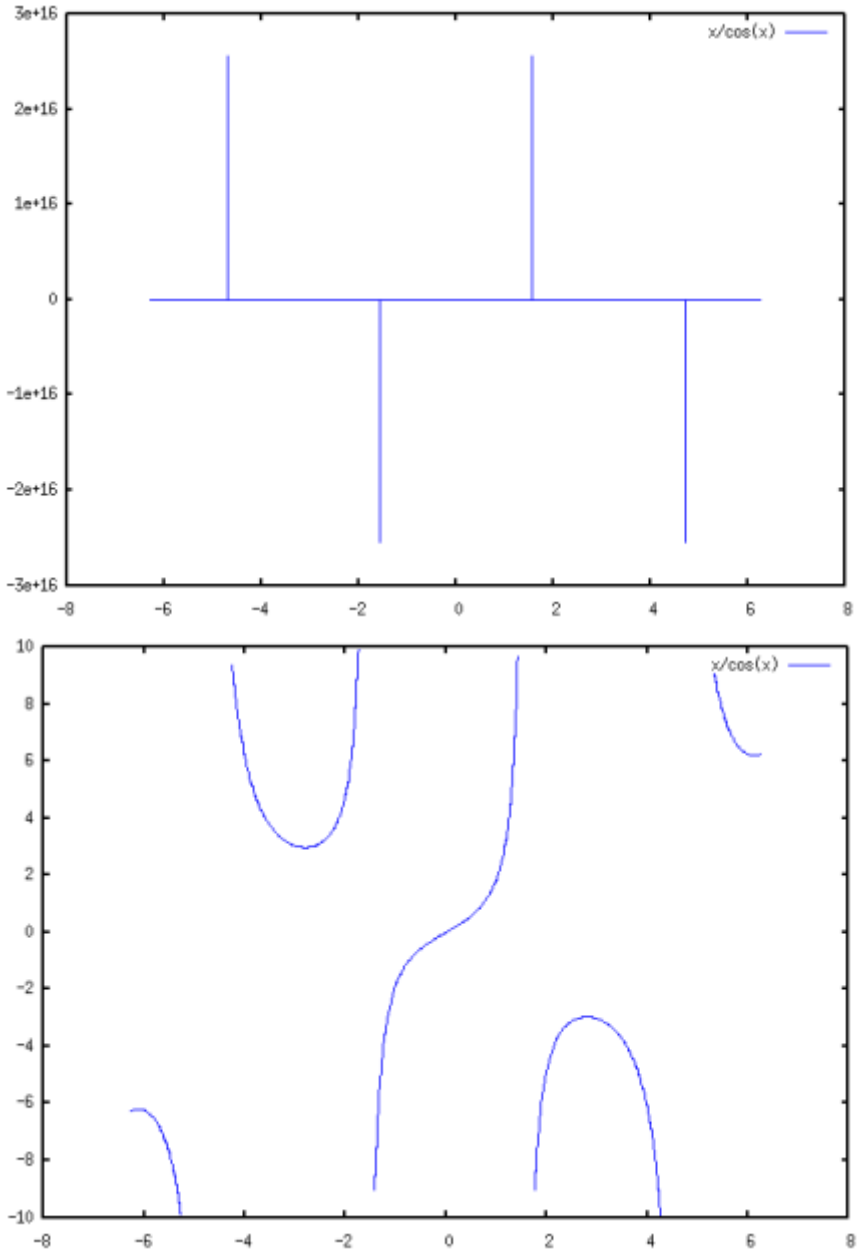




В некоторых случаях автоматический подбор отображаемого участка вертикальной оси может нас не устроить. Например, он работает не очень хорошо, если функция имеет на заданном промежутке точку разрыва, хотя бы один из односторонних пределов в которой равен бесконечности: тогда промежуток по оси Y будет выбран слишком большим. Да и в других случаях может понадобиться изменить поведение «по умолчанию». Для этого предусмотрен такой вариант вызова функции: plot2d(выражение, [символ, начало, конец], [y, начало, конец]). Здесь буква y используется в качестве обозначения вертикальной оси, а остальные два параметра имеют тот же смысл, что и выше.

(%i1) $\mathrm{plot2d}\left(\frac{x}{\cos(x)}, [x, -2\pi, 2\pi]\right)$$

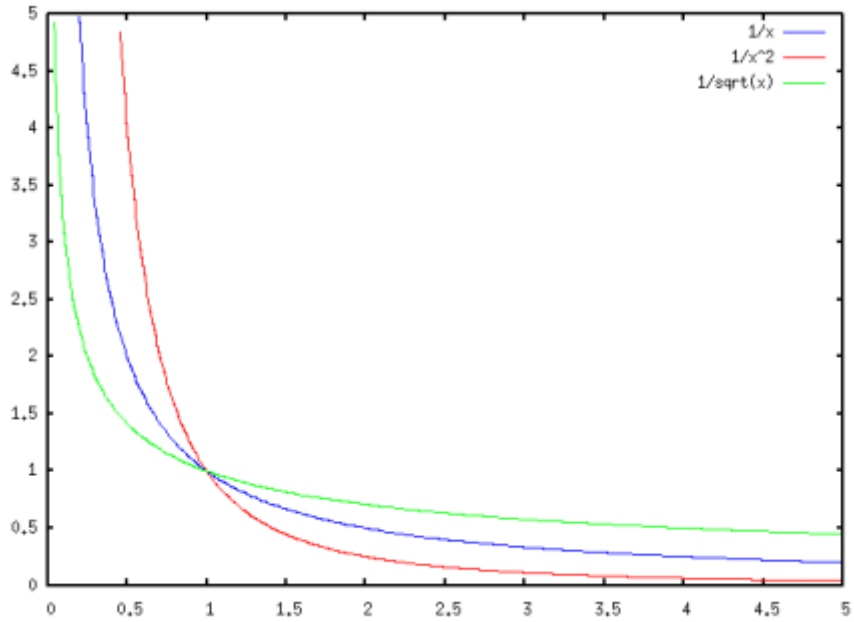
(%i2) $\mathrm{plot2d}\left(\frac{x}{\cos(x)}, [x, -2\pi, 2\pi], [y, -10, 10]\right)$$



Как видите, умолчательный вид графиков в gnuplot достаточно прост, но здесь можно очень и очень многое менять с помощью дополнительных опций.

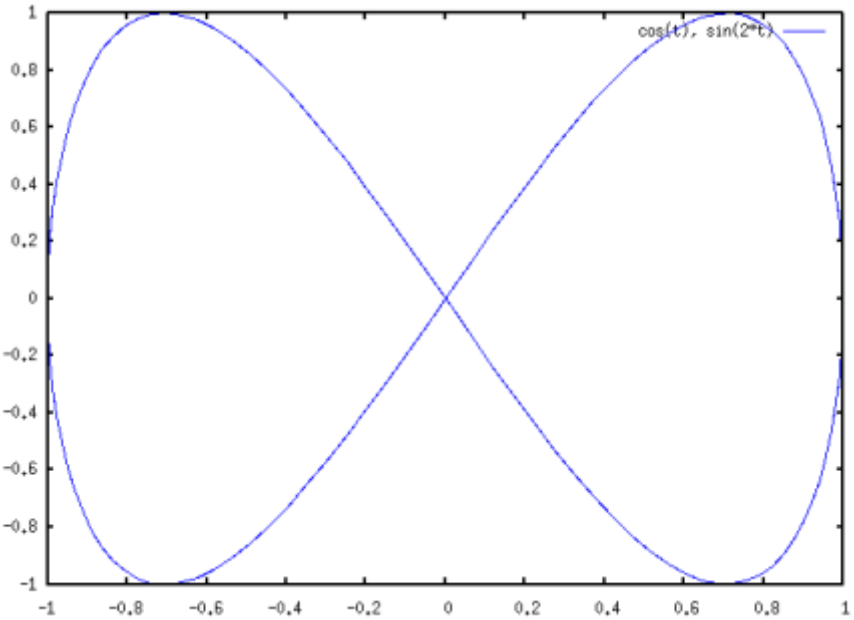
Чтобы построить на одной и той же картинке одновременно два графика (или больше), просто передайте функции plot2d вместо отдельного выражения их список:

(%i1) $\mathrm{plot2d}\left(\left[\frac{1}{x},\frac{1}{x^2},\frac{1}{\sqrt{x}}\right],[x,0.01,5],[y,0,5]\right)$



Может plot2d строить и графики параметрически заданных функций. Для этого используется список с ключевым словом parametric: plot2d([parametric, x-выражение, y-выражение, [переменная, начало, конец], [nticks, количество]]). Здесь x-выражение и y-выражение задают зависимость координат от параметра, то есть, по сути, это две функции вида $x(t)$, $y(t)$, где t — переменная параметризации. Эта же переменная должна фигурировать в следующем аргументе-списке, а параметры начало, конец, как и в двух других рассмотренных случаях, задают отрезок, в пределах которого этот параметр будет изменяться. Последний аргумент-список, с ключевым словом nticks, задает количество кусочков, на которые будет разбит интервал изменения параметра при построении графика. Этот аргумент опционален, но на практике он нужен почти всегда: умолчательное значение nticks равно 10. Вот пример построения графика параметрической функции:

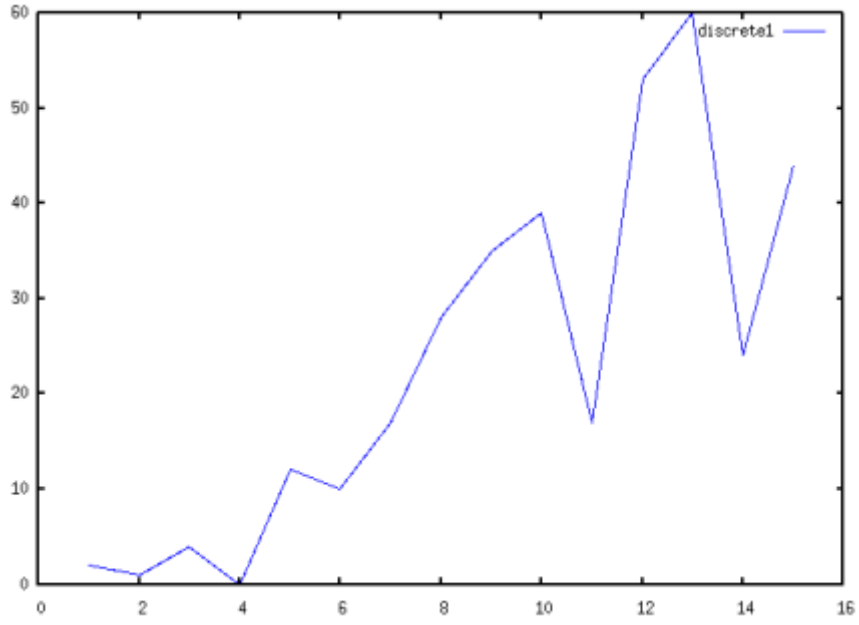
```
(%i1) plot2d([parametric, cos(t), sin(2*t), [t, -%pi, %pi], [nticks, 120]])$
```



Кроме parametric, функция plot2d понимает еще одно ключевое слово: discrete. Предназначено оно, как нетрудно догадаться, для отображения на плоскости дискретных множеств; точнее говоря, конечных наборов точек. По записи аргументов такой вариант распадается еще на два: plot2d([discrete, x-список, y-список]) и plot2d([discrete, [x, y]-список]). В первом варианте координаты задаются как два отдельных списка [x1, x2, …, xn], [y1, y2, ,…, yn], а во втором — как список пар координат отдельных точек [[x1, y1], [x2, y2], …, [xn, yn]].

Если мы, к примеру, имеем набор статистических значений, зависящих от номера, мы можем отобразить его, задав в качестве x-координат сами эти номера, то есть натуральные числа:

```
(%i1) data: makelist(random(5*x), x, 1, 15)
(%o1) [2, 1, 4, 0, 12, 10, 17, 28, 35, 39, 17, 53, 60, 24, 44]
(%i2) plot2d([discrete, makelist(x, x, 1, 15), data])$
```

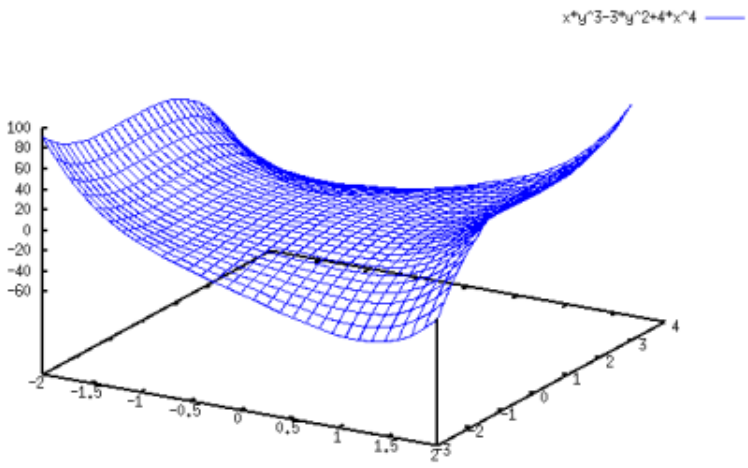


По умолчанию множество отображается в виде ломаной с вершинами в заданных точках; такое поведение можно изменить и получить вывод, к примеру, в виде отдельных точек. Это достигается использованием специальных опций, применимых как к plot2d, так и к plot3d.

Построение 3-х мерных изображений

Функция plot3d имеет два варианта вызова: один для явного задания функции и один для параметрического. В обоих случаях функция принимает три аргумента. Для явно заданной функции: plot3d (выражение, [переменная1, начало, конец], [переменная2, начало, конец]); аргументы аналогичны plot2d, с той разницей, что здесь независимых переменных две.

(%i1) $plot3d(4\,x^4+x\,y^3-3\,y^2,[x,-2,2],[y,-3,3])$

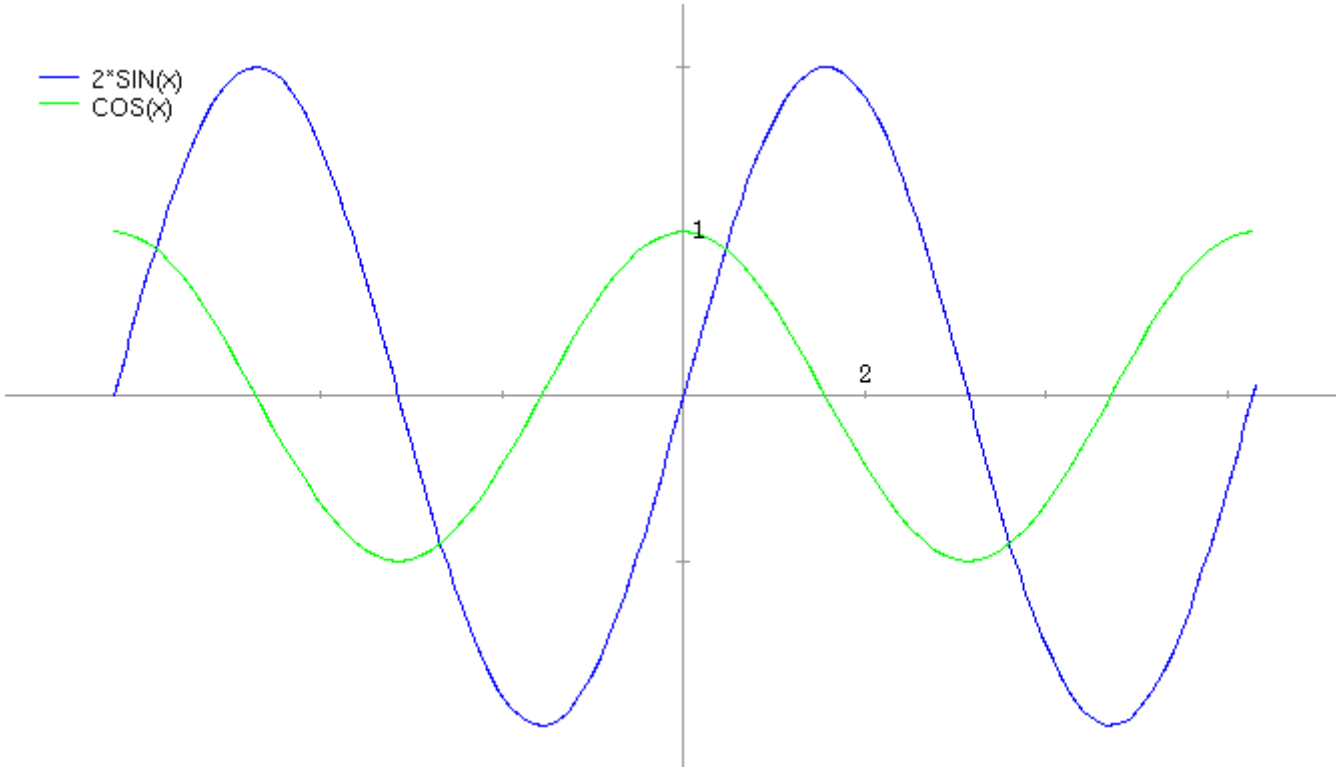


Построение нескольких поверхностей на одном графике не поддерживается — потому, вероятно, что на таком рисунке проблематично было бы что-либо разглядеть. Для параметрически заданной функции ключевое слово parametric не требуется. График параметрически заданной функции строится так: plot3d([выражение1, выражение2, выражение3], [переменная1, начало, конец], [переменная2, начало, конец]), где выражения отвечают, по порядку, $x(u, v)$, $y(u, v)$, $z(u, v)$.

Построение через меню: Графики→Plot 2d или (Plot 3d)

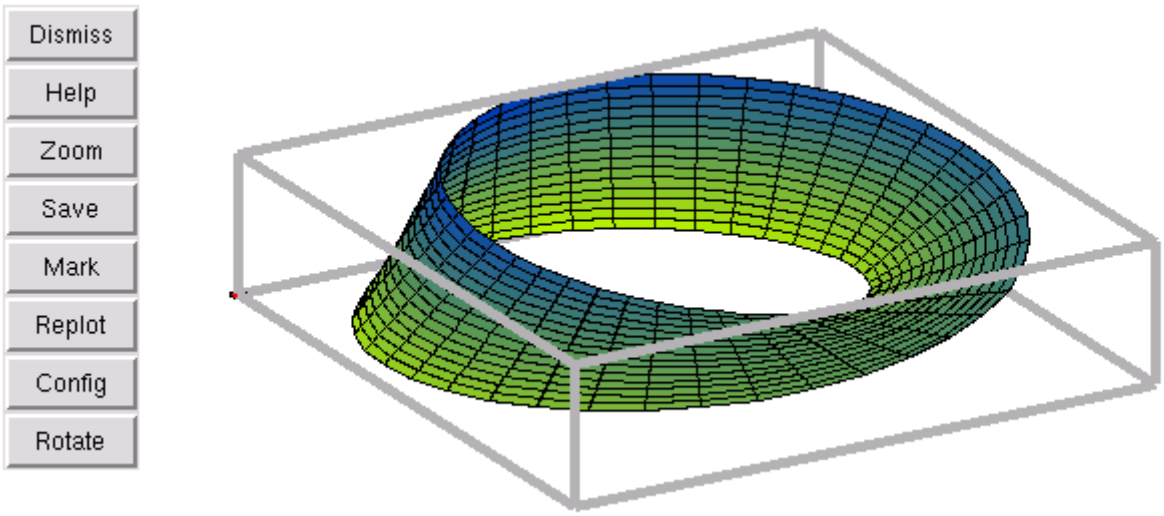
*Графики строятся при помощи функции **plot2d**, например,*

```
(%i1) plot2d([2*sin(x), cos(x)], [x, -2*%PI, 2*%PI]);
```



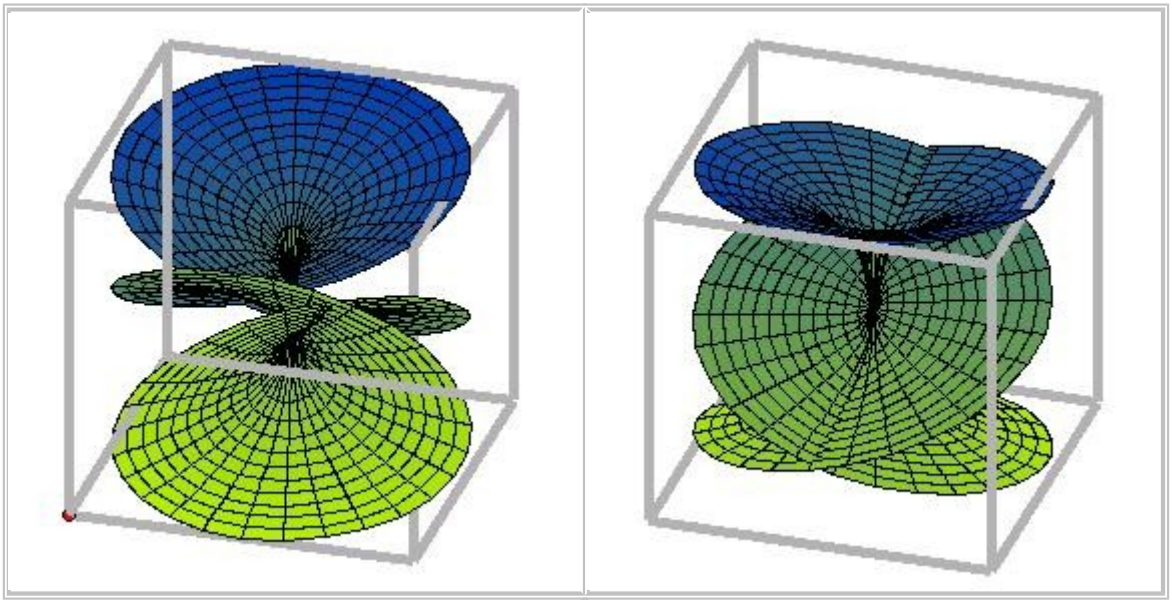
Для построения изображений трехмерных объектов используется функция ***plot3d****. Вот как, например, запишется функция для рисования листа Мебиуса:*

```
(%i2) plot3d([cos(x)*(3+y*cos(x/2)),
  sin(x)*(3+y*cos(x/2)), y*sin(x/2)],
  [x,-%pi,%pi],[y,-1,1],['grid,40,15]);
```



В левом верхнем углу рисунка находится меню (появляющееся при подведении курсора). Оно содержит ряд опций для сохранения рисунка и его преобразований. Ниже приводятся два рисунка, соответствующие рассматриваемой с разных точек римановой поверхности.

```
(%i3) plot3d(r^.33*cos(th/3),[r,0,1],[th,0,6*%pi],
     ['grid,12,80],
     ['transform_xy,polar_to_xy],
     ['view_direction,1,1,1.4],
     ['colour_z,true]);
```



Опции графики

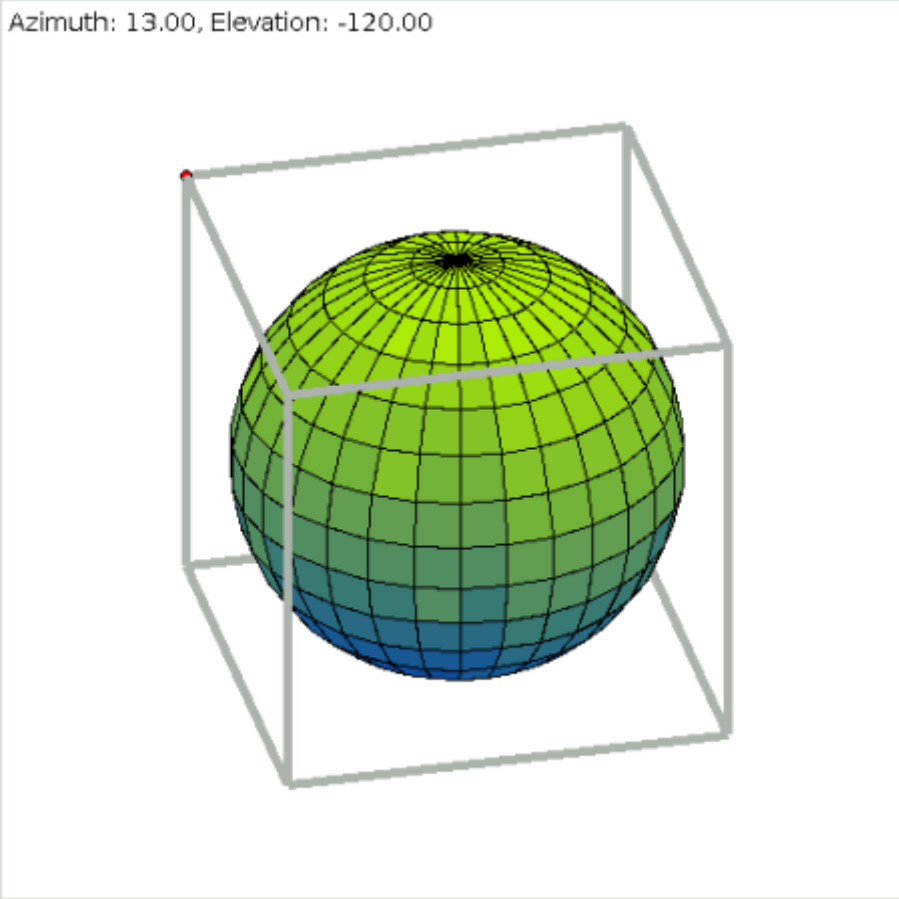
Первая опция, которую мы рассмотрим, задает формат вывода результата; так она и называется: plot_format. Формат может принимать одно из четырех значений, первое из которых действует по умолчанию: gnuplot, mgnuplot, openmath и ps. Gnuplot имеет перед следующими двумя интерфейсами только один недостаток: она генерирует статичное изображение, тогда как mgnuplot и openmath позволяют в реальном времени масштабировать и передвигать картинку, а plot3d — еще и вращать линию или поверхность в разные стороны в пространстве.

Следующий вариант — mgnuplot — является дополнительным интерфейсом к gnuplot, написанным на Tcl/Tk, но динамика у него настолько «задумчивая», а остальные возможности настолько бедны, что я не вижу смысла останавливаться на нем подробнее.

Openmath - предоставляет хорошую интерактивность, особенно ценную в трехмерном варианте: после того, как объект сгенерирован, его можно масштабировать и очень динамично вращать, разглядывая со всех сторон. Ggnuplot позволяет задавать точку обзора трехмерного объекта в качестве одного из многочисленных параметров, то есть хотя картинка и статична, но с какой стороны на нее смотреть, мы можем указать произвольно. Последнее значение опции plot_format подталкивает Maxima к непосредственной генерации PostScript-документа с изображением. Но и здесь надо сказать: генерировать PostScript-вывод умеет и все тот же gnuplot.

Большинство остальных опций относятся только к формату вывода gnuplot. Рассмотрим еще одну универсальную, пригодную для всех форматов и преобразующую не результирующее изображение, а сам процесс построения графика; точнее, систему координат. Называется эта опция transform_xy, по умолчанию она равна false. Передавать ей нужно выражение, сгенерированное функцией make_transform([x, y, z], f1(x, y, z), f2(x, y, z), f3(x, y, z)). Кроме того, существует одно встроенное преобразование, известное как polar_xy и соответствующее make_transform([r, th, z], r*cos(th), r*sin(th), z), то есть переходу к полярной цилиндрической системе координат. В качестве примера использования transform_xy приведу преобразование к полярным сферическим координатам, раз уж во встроенном виде его нет:

```
(%i1) plot3d(1, [f, 0, 2*%pi], [p, 0, 2*%pi],
        [transform_xy, make_transform([t, f, r],
         r*sin(f)*sin(t), r*cos(f)*sin(t), r*cos(t))],
        [plot_format, openmath])$
```



Обратите внимание: в первом аргументе-списке к make_transform последним должен идти зависимый символ, то есть тот, который будет выступать функцией от двух других.

Если вам нужно постоянно работать со сферическими координатами, можете задать, скажем, spherical_xy:make_transform([t, f, r], r*sin(f)*sin(t), r*cos(f)*sin(t), r*cos(t)), и затем при построении графиков писать [transform_xy, spherical_xy].

Элементы программирование в системе Maxima

До сих пор мы использовали систему Maxima в интерактивном режиме, подобно калькулятору. Если часто приходится выполнять определенную последовательность вычислений, то лучше оформить ее в виде программы, которая затем вызывается в случае надобности. Ниже приводится небольшая программа для нахождения критических точек функции f(x). Пользователю предлагается ввести функцию f, после чего вычисляется производная введенной функции и при помощи функции solve решается уравнение $f_x = 0$. Программа записывается в текстовый файл и затем загружается в систему Maxima при помощи функции **batch**.

```
critpts():=(
   print("Программа нахождения критических точек"),

/* Запрос на ввод функции     */
```

```
   print("Введите функцию f(x):"),
   f:read(),

/* Печать введенной функции (для контроля) */
   print("f = ",f),

/* В переменную eq помещаем значение производной */
   eq:diff(f,x),

/* Решаем уравнение */
   solve(eq, x)
)$
```

Программа состоит из единственной функции (без аргументов), которая называется critpts. Команды отделяются друг от друга запятыми. Вот пример выполнения программы:

```
(C1) batch("critpoints.max");

batching #p/home/test/critpoints.max
(C2) critpts() := (PRINT("Программа #
нахождения критических точек"),
PRINT("Введите функцию f(x):"),
f : READ(),
PRINT("f = ", f),
eq : DIFF(f, x),
SOLVE(eq, x))

(C3) critpts() ;
Программа нахождения критических точек
Введите функцию f(x):
(x+2)/(x^2+1);

                    x + 2
              f =  ------
                    2
                   x  + 1

(D3) [x = - SQRT(5) - 2, x = SQRT(5) - 2]
```

Среди средств для операций с файлами функции с наиболее очевидными именами — save и load — имеют, вопреки привычной для Maxima логичности всех названий, различный контекст. Первая предназначена для выгрузки Maxima-выражений в виде исходных кодов на Lisp, load — для обработки так называемых пакетных (batch) файлов, хранящих выражения уже в синтаксисе самой Maxima. А поскольку в виде таких файлов поставляется немалое количество функционала Maxima, то начнем с загрузки.

Примеры. 1. Нахождение приближенного решения уравнения $f(u)=0$ в окрестности точки u=a, с указанной точностью, методом Ньютона, здесь $f(u)=\cos u$, $u=1, \varepsilon=0,01$

```
(%i1) load (newton1);
(%o1) /usr/share/maxima/5.10.0cvs/share/numeric/newton1.mac
(%i2) newton (cos (u), u, 1, 1/100);
(%o2)                  1.570675277161251
(%i3) ev (cos (u), u = %);
(%o3)                1.2104963335033528E-4
```

2. Нахождение приближенного решения уравнения $f(x)=0$ в окрестности точки x=a, с указанной точностью, методом Ньютона, здесь $f(x)=x^2-a^2$, $x=a/2$, $\varepsilon=\frac{a^2}{100}$

```
(%i4) assume (a > 0);
(%o4)                          [a > 0]
(%i5) newton (x^2 - a^2, x, a/2, a^2/100);
(%o5)                    1.00030487804878 a
(%i6) ev (x^2 - a^2, x = %);
                                                2
(%o6)                 6.098490481853958E-4 a
```

Функции чтения файлов с выражениями Maxima существует три: demo(*имя-файла*), batch(*имя-файла*) и batchload(*имя-файла*). Первая предназначена для загрузки так называемых демо-файлов, задуманных, как и явствует из названия, для демонстрационных примеров. Она загружает демо-файл и выполняет его в пошаговом режиме, ожидая нажатия Enter после выполнения каждой строки. В составе Maxima поставляется значительное количество демо-файлов; упоминания о них можно найти в документации, а сами файлы несложно обнаружить среди содержимого пакета maxima-share (либо, в случае отсутствия такового в вашем дистрибутиве, просто maxima) по их расширению — .dem.

Функция batch() загружает Maxima-файл с расширением .mac или .mc (от первоначального названия программы — Macsyma) и выполняет содержащиеся в нем выражения так, как если бы они вводились прямо в текущей сессии, то есть с отображением результата каждого выражения и назначением меток %iN, %oN. Функция batchload(), напротив, подгружает пакетный файл «молча»: все назначенные в нем функции и переменные становятся доступны, но результаты не видны, и весь хранимый ввод-вывод, включая значения символов % и _ и результаты, возвращаемые функцией %th(), остается тем же, что и до вызова.

Функции batch() и batchload() используют при поиске файлов для загрузки путь (точнее сказать, шаблон, потому как в нем содержатся не только имена каталогов, но и допустимые расширения файлов), который хранится в переменной file_search_maxima. По умолчанию эта переменная содержит все каталоги, в которые устанавливаются .mac-файлы из пакетов Maxima, а также ~/.maxima, предназначенный для пользовательских файлов. Для других функций загрузки существуют отдельные переменные: file_search_lisp и file_search_demo, смысл которых понятен из их названий.

Нужно вспомнить о вышеназванной функции load. Она, фактически, является оберткой над двумя функциями: уже описанной выше batchload() и loadfile(), вторая, совершенно аналогично первой, загружает файл, но уже не с выражениями Maxima, а с исходным кодом Lisp, то есть является парной к функции save(). Функцию load() можно, в принципе, использовать вместо batchload(): путь file_search_maxima задан в ней раньше, чем file_search_lisp, так что в случае неоднозначности она будет загружать файлы Maxima; а кроме того, так короче.

Некоторый функционал Maxima содержится в неподгружаемых автоматически внешних файлах, которые, соответственно, нужно принудительно загрузить перед использованием:

```
(%i1) A: matrix([a11, a12, a13], [a21, a22, a23])
(%o1) ( a11  a12  a13 )
      ( a21  a22  a23 )
(%i2) matrix_size(A)
(%o2) matrix_size( ( a11  a12  a13 ) )
                   ( a21  a22  a23 )
(%i3) load(linearalgebra)
(%o3) /usr/share/maxima/5.10.0/share/linearalgebra/linearalgebra.mac
(%i4) '%i2
(%o4) [2, 3]
```

Помимо ручной загрузки нужного файла, можно также настроить Maxima на автоматическую подгрузку в случае вызова заданной функции. Делается это так: setup_autoload(*имя-файла*, *имена-функций*); нужные функции здесь перечисляются через запятую прямо после имени файла. Удобнее, конечно, будет не вызывать функцию setup_autoload() вручную (так в ней и толку немного), а настроить Maxima на автоматический ее запуск при старте программы. Файл, который, при его наличии, вызывается при каждом запуске Maxima, называется maxima-init.mac и самое логичное для него местоположение — все тот же каталог ~/.maxima. Конечно, он может содержать не только вызовы функции setup_autoload(), а любые выражения Maxima, которые вы хотите выполнять при каждом ее запуске. Использование этой функции может сделать вашу работу с Maxima намного более удобной в том случае, если вы часто используете некоторые из внешних функций Maxima или функции, вами же и написанные.

Для полноценного чтения файлов всего сказанного уже вполне достаточно, теперь перейдем к записи в них. Функция stringout(), которая позволяет выгружать в файл любые выражения и функции Maxima в точно таком виде, в каком их загружают функции demo(), batch() и batchload(). С ее помощью можно писать выражения, которые вы хотите иметь во внешнем модуле, находясь непосредственно в интерфейсе Maxima, с последующей записью в этот самый модуль. Для выгрузки функций в один из стандартных каталогов Maxima (самым логичным вариантом будет, пожалуй, упомянутый выше ~/.maxima) имя файла во всех вариантах вызова функции stringout() нужно задавать с полным путем; в случае задания имени без пути файл будет создан в текущем каталоге, то есть в том, откуда производился запуск Maxima.

Варианты заданий для самостоятельной работы

Вариант 1.

1. Решить дифференциальные уравнения:

A) $xy' - y = y^3$; B) $y' + \frac{y}{x} = -xy^2$ C) $y'' - y' - 2y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(1 + e^x\right) y \cdot y' = e^x$; $y = 1$ при $x = 0$;

B) $y'' + 4y' = 12x^2 - 2x + 2$; $y = 0$, $y' = 0$
при $x = 0$, построить графики решений.

3. Найти значение всех корней уравнения:
A) $x^5 - 1 = 0$;
B) найти приближенно значение $\cos x$, в окрестности точки $x = 1$, с точностью до 0.001.

4. Разложить в ряд Тейлора следующие функции

1. $\sin(x + \frac{\pi}{4})$. 2. e^{x^2}.

Вариант 2.

1. Решить дифференциальные уравнения:
A) $y' \operatorname{tg} x = y$; B) $x\left(x + 2y\right) dx + \left(x^2 - y^2\right) dy = 0$ C) $y'' - 9y = 2 - x$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $xy' + y - e^x = 0$; $y = b$ при $x = a$;

B) $y'' + 4y = 2\cos 2x$; $y = 0$, $y' = 4$
при $x = 0$, построить графики решений.

3. Найти приближенннное значение корней уравнения:
A)(-4.5 + 6x)^3 = 0.7(9 + x^5);
B) найти приближенно значение $\sin x = 0$, в окрестности точки $x = 1$.

4. Разложить в ряд Тейлора следующие функции

1. $\sin^2 x$ 2. $\ln(2 + x)$

Вариант 3.

1. Решить дифференциальные уравнения:

A) $\left(1 + y\right) dx - \left(1 - x\right) dy = 0$; B) $y' = \frac{y}{x} - 1$ C) $y'' + y' = \frac{1}{2}$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(xy^2+x\right)dx+\left(x^2y-y\right)dy=0$; $y=1$ при $x=0$;

B) $y''-9y'=2-x$; $y=0$, $y'=1$

при $x=0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:
A) (5.6 - 2x)^3 = -7.7(-1 + x^5)
B) $2\sin x-x=0$.

4. Разложить в ряд Тейлора следующие функции

1. $\cos x^2$. 2. $\ln(1+x-2x^2)$.

Вариант 4.

1. Решить дифференциальные уравнения:

A) $\left(1+y^2\right)dx+\left(1+x^2\right)dy=0$; B) $y'=-\dfrac{x+y}{x}$ C) $y''+3y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $x\left(x-1\right)y'+y=x^2\left(2x-1\right)$; $y=4$ при $x=2$;

B) $y''-2y'+3y=0$; $y=1$, $y'=3$

при $x=0$, построить графики решений.

3. Найти приближеннное значение действительных корней уравнения:
$(1-x)^5(2-x)^3(3-x)=0$.

4. Разложить в ряд Тейлора следующие функции

1. $\cos(x+y)$. 2. $\dfrac{x}{4+x^2}$.

Вариант 5.

1. Решить дифференциальные уравнения:

A) $\left(1+e^x\right)yy'=e^x$; B) $\left(x-y\right)ydx-x^2dy=0$ C) $y''-4y'+10y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y'+y\cos x=\cos x$; $y=1$ при $x=0$;

B) $y''-2y'+2y=0$; $y=1$, $y'=3$

при $x=0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:
A) (11.4 + x)^3 = 1.9(5 + x^5)$
B) $\sin x - x/2 = 0$.

4. Разложить в ряд Тейлора следующие функции

1. $\frac{1}{\sqrt{4-x^2}}$. 2. $\ln\frac{1+x}{1-x}$.

Вариант 6.

1. Решить дифференциальные уравнения:

A) $2xyy' = x^2 + y^2$; B) $(x+y)dx + xdy = 0$; C) $y'' + 2y' - 2y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $xy' - y + x^2 = 0;\ \ y = 0$ при $x = 1$;

B) $y'' - 4y' + 3y = 0;\ \ y = 6,\ y' = 10$
при $x = 0$, построить графики решений.

3. Найти приближеннное значение действительных корней уравнения:
$x^5 - x - 1$.

4. Разложить в ряд Тейлора следующие функции

1. $\cos 2x$. 2. $\frac{x}{9+x^2}$.

Вариант 7.

1. Решить дифференциальные уравнения:

A) $x(x+2y)dx + (x^2 - y^2)dy = 0$; B) $y' = \frac{x-y}{x-2y}$ C) $y'' + 16y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y' - y\,\mathrm{tg}\,x = \frac{1}{\cos x};\ \ y = 0$ при $x = 0$;

B) $y'' + 4y' = 12x^2 - 2x + 2;\ \ y = 0,\ y' = 0$
при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:
A) (18 + 2x)^3 = 5.3(1 + x^5);
B) $\sin x - x = 0$ на отрезке [-0,π]

4. Разложить в ряд Тейлора следующие функции

1. xe^{-6x}. 2. $e^{(x+y)}$.

Вариант 8.

1. Решить дифференциальные уравнения:

A) $y' + 2y = e^{-x}$; B) $y' - \frac{y}{x} = x$ C) $y'' - 9y' - 10y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y' - \frac{y}{1-x^2} - 1 - x = 0$; $y = 0$ при $x = 0$;

B) $y'' + 4y = 2\cos 2x$; $y = 0$, $y' = 4$
при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней системы уравнений:
$24x^2 - y^2 = 10, \quad xy - x = 2$

4. Разложить в ряд Тейлора следующие функции

1. xe^{-2x}. 2. e^{-x}.

Вариант 9.

1. Решить дифференциальные уравнения:

A) $xy' - 2y = x^3 \cos x$; B) $y' + \frac{2y}{x} = x^3$; C) $y'' + 4y' - 7y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(x^2 - 3y^2\right)dx + 2xydy = 0$; $y = 1$ при $x = 2$;

B) $y'' - 9y' = 2 - x$; $y = 0$, $y' = 1$
при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:
A) $x^6 + 1 = 0$;
B) $\cos x - x = 0$ на отрезке [$-\pi, \pi$]

4. Разложить в ряд Тейлора следующие функции

1. $\frac{2x-3}{(x-1)^2}$. 2. $\frac{3x-5}{x^2-4x+3}$.

Вариант 10.

1. Решить дифференциальные уравнения:

A) $y' x\ln x - y = 3x^3\cos x$; B) $y' + \frac{y}{x} = -xy^2$ C) $y'' + 20y' + 19y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y' \sin x = y\ln y;\ y = 1$ при $x = \frac{\pi}{2}$;

B) $y'' - 2y' + 3y = 0;\quad y = 1,\ y' = 3$
при $x = 0$, построить графики решений.

3. Найти значение всех корней уравнения:
A) $x^3 - 1 = 0$;
B) $\cos(x) - x/2 = 0$ на отрезке [$-\pi, \pi$]

4. Разложить в ряд Тейлора следующие функции

1. $\cos^2 x$. 2. $\sin(x+y)$.

ЛИТЕРАТУРА

1. Вадим Житников. Компьютеры, математика и свобода. Компьютерра. №16 (636), 2006. http://old.computerra.ru/gid/266002/
2. Википедия о Maxima : https://ru.wikipedia.org/wiki/Maxima
3. Тарнавский Т. Maxima – алгебра и начала анализа // LinuxFormat №11, 2006.
4. Русскоязычный раздел официального сайта maxima.sourceforge.net
5. Чичкарев Е.А. Компьютерная математика с Maxima : Руководство для школьников и студентов / Е.А Чичкарев - М. : ALT Linux, 2012. - 384 с.

Учебное издание

Учебное пособие для вузов

Составители: Ткачева Светлана Анатольевна
Безручкина Людмила Валентиновна
Садчиков Павел Валерьевич

Редактор