ПЕРМСКИЙ
ГОСУДАРСТВЕННЫЙ
НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

И. Е. Полосков

СИСТЕМА АНАЛИТИЧЕСКИХ ВЫЧИСЛЕНИЙ MAXIMA

ОПИСАНИЕ И ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

Пермь 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»

И.Е.Полосков

СИСТЕМА АНАЛИТИЧЕСКИХ ВЫЧИСЛЕНИЙ MAXIMA

ОПИСАНИЕ И ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

Допущено методическим советом Пермского государственного национального исследовательского университета в качестве учебного пособия для студентов, обучающихся по направлениям подготовки бакалавров и магистров «Механика и математическое моделирование», «Прикладная математика и информатика» и другим физико-математическим направлениям

Пермь, 2020

УДК 681.3.062
ББК 32.973-0.18.1:22.21
П525

Полосков И.Е.

П525 Система аналитических вычислений Maxima. Описание и примеры использования [Электронный ресурс]: учебное пособие / И. Е. Полосков; Пермский государственный национальный исследовательский университет. – Электронные данные. – Пермь, 2020. – 5,3 Мб; 346 с. – Режим доступа: http://www.psu.ru/files/docs/science/books/uchebnie-posobiya/poloskov-sistema-analitiche skih-vychislenij-maxima.pdf. – Заглавие с экрана.

ISBN 978-5-7944-3509-2

Представлены введение в структуру и средства свободно распространяемой компьютерной системы аналитических вычислений Maxima, примеры её применения для решения различных задач, а также сведения, необходимые для ис-пользования этой системы в процессе обучения дисциплинам "Математика" студентов бакалавриата нематематических направлений подготовки и "Системы аналитических вычислений" на механико-математическом факультете.

Пособие может быть полезно для студентов бакалавриата и магистратуры других направлений и специальностей, которые желают иметь доступное вспомогательное средство для самостоятельной подготовки и её контроля по дисциплинам, преподавание которых базируется на серьезных знаниях различных разделов высшей математики.

УДК 681.3.062
ББК 32.973-0.18.1:22.21

Издается по решению ученого совета механико-математического факультета Пермского государственного национального исследовательского университета

Рецензенты: кафедра "Прикладная физика" Пермского национального исследовательского политехнического университета (зав. каф. – доктор физ.-мат. наук, доцент ***Д. А. Брацун***); главный научный сотрудник кафедры ЭВМ Вятского государственного университета, д-р физ.-мат. наук, проф. ***А. В. Шатров***

ISBN 978-5-7944-3509-2

ПРЕДИСЛОВИЕ

Maxima – это система (программный комплекс) компьютерной алгебры (СКА, ПККА, система аналитических вычислений, САВ, computer algebra system, CAS), принадлежащая к тому же классу программного обеспечения, что и более известные Maple [23] и Mathematica [31].

Maxima – это мощная система, предоставляющая возможности проведения преобразований как символьных, так и численных выражений. Среди таких преобразований отметим операции дифференцирования, интегрирования, разложения в степенные ряды; преобразование Лапласа; решение систем обыкновенных дифференциальных и линейных алгебраических уравнений; средства работы с многочленами, включая ортогональные, множествами, списками, векторами, матрицами и тензорами. Имеется ряд теоретико-вероятностных и статистических функций, процедур для анализа фракталов, построения базисов Грёбнера, ввода/вывода в файлы.

В среде пакета допустимо проведение расчетов с повышенной разрядностью при использовании рациональных дробей и целых чисел и чисел с плавающей точкой. Maxima позволяет строить различные формы 2D- и 3D-графиков математических и таблично заданных функций и визуализировать статистические данные. Сложные вычисления можно оформлять в виде отдельных процедур, которые затем могут быть использованы при решении других задач. В системе имеется ряд пакетов специального назначения, например, для работы с тензорами, решения рекуррентных уравнений, получения сумм и др.

Для выполнения расчетов, требующих значительного числа операций над числами с плавающей точкой и/или обработки векторно-матричных выражений большой размерности, пользователь системы может применить генерирование фрагментов программ на других языках (в особенности на языке программирования Fortran), на которых необходимые действия могут быть выполнены более эффективно.

Несмотря на то, что Maxima – универсальная СКА, многие специальные расчеты (например, факторизация больших целых чисел, манипулирование огромными полиномами и др.) часто выполняются в ней намного эффективнее, чем в специализированных пакетах.

В системе Maxima пользователь может определять свои собственные команды. Поэтому средства системы можно легко подстроить под собственные нужды. Существует и значительное число "пакетов" (специальных проблемно ориентированных библиотек программ), разработанных

создателями и сообществом пользователей системы и доступных всем заинтересованным лицам, которые имеют дело с этой САВ.

В отличие от других широко или не очень известных САВ Maxima имеет довольно долгую историю развития (см.приложение П.2) и является программным комплексом с открытым исходным кодом. Очевидное преимущество программного обеспечения с открытым исходным кодом заключается в том, что его можно скачать и использовать бесплатно. Тем не менее есть и другие преимущества для пользователей: доступные исходные коды алгоритмов позволяют увидеть, как другие люди решали подобную проблему, причем можно свободно копировать и изменять их разработки. В случае непредвиденных результатов можно обратиться к источникам, а следовательно, есть некий шанс обнаружить возможную ошибку.

Важным преимуществом системы является наличие интерфейса пользователя с сообщениями на русском языке, а также справки и инструкции по работе с программным комплексом. Несмотря на то, что русскоязычной версии справки нет, в интернете можно найти значительное количество материалов учебного характера и примеров использования системы Maxima.

1. ВВЕДЕНИЕ

1.1. Символьные вычисления на компьютере

Исторически слово "вычисление" по отношению к электронно-вычислительным машинам (ЭВМ, персональным компьютерам (ПК)) стало синонимом словосочетания "вычисление с числами". Обычно под такими расчетами понимается выполнение арифметических операций (+, -, /, *, возведение в степень), получение значений математических функций (тригонометрических, логарифмических, гиперболических, специальных и др.), решение линейных и нелинейных уравнений и т.д.

Существенным здесь является то, что исходя из некоторых чисел мы получаем другие числа. Причем известно, что проводить такие расчеты точно обычно невозможно.

Несомненно, что более 70 лет развития вычислительной техники привели к громадному увеличению мощности расчетных инструментов: от карандаша и бумаги через калькулятор к числовым "молотилкам" – суперкомпьютерам (см. таблицу и рис. 1.1, 1.2, 1.3).

Год	ЭВМ	Потребление электроэнергии (Вт)	Скорость (слож/сек)	ОП (Кб)	Цена ($)
1951	UNIVAC I	124 500	1 900	48	1 000 000
1964	IBM S360	10 000	500 000	64	1 000 000
1965	PDP-8	500	330 000	4	16 000
1976	Cray-1	60 000	166 000 000	32 768	4 000 000
1981	IBM PC	150	240 000	256	3 000
1991	HP 9000	500	50 000 000	16 384	7 400
2005	IBM notebook	20	1 000 000 000	512 000	1 900

Но есть другой важный компонент, который будем называть *символьными* или *алгебраическими вычислениями* и понимать под этим термином различные манипуляции с символами, представляющими математические объекты. Конечно, среди обрабатываемых символов могут присутствовать и числа (целые, рациональные, действительные, комплексные, алгебраические), которые могут использоваться при работе с полиномами, рациональными функциями, системами различных уравнений и даже более абстрактными математическими объектами, такими как группы, кольца, алгебры и их элементы. Более того, прилагательное "символический"

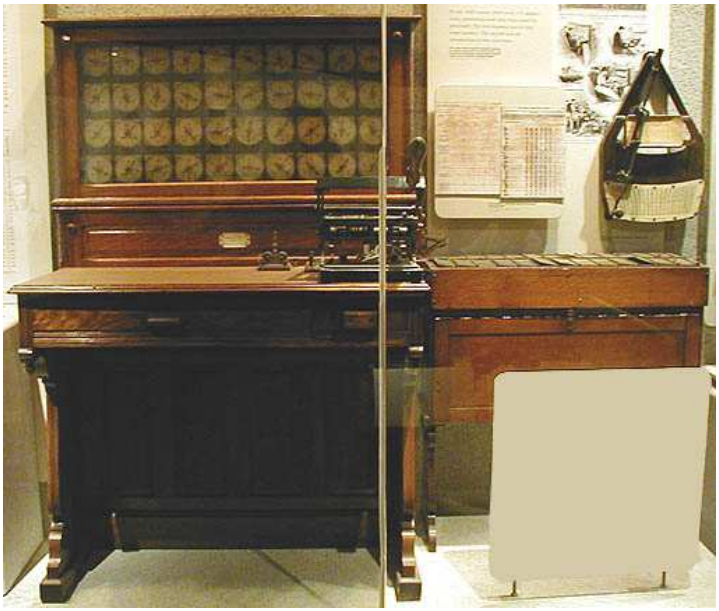


Рис. 1.1. Электрическая перфорационная система Г.Холлерита (счетная машина, кард-ридер, перфоратор и сортировочная машина), 1890, Национальный музей американской истории, Смитсоновский институт, Вашингтон, США

означает во многих случаях, что явной целью решения математической проблемы является получение ответа в виде формулы или нахождение символьной аппроксимации. Под термином "алгебраический" понимается, что вычисления проводятся точно, согласно правилам алгебры без применения арифметики действительных чисел с плавающей точкой. За примерами задач, где требуется работа с символами, далеко ходить не нужно – это дифференцирование, интегрирование, разложение функций в ряды, факторизация полиномов одной и нескольких переменных, аналитическое решение дифференциальных уравнений, упрощение математических выражений и т.д.

Суммируя вышесказанное, можно констатировать, что

– компьютерная алгебра концентрируется на преобразованиях объектов на алгебраических структурах: базовых числовых множествах, алгебраических расширениях последних, кольцах полиномов и полях функций, дифференциальных или разностных полях, группах и др. Часто более эф-

фективно при обработке сначала алгебраически упростить выражение, а затем использовать его для численных расчетов. Кроме того, это уменьшает погрешность вычислений (см. таблицу);

Численные расчеты	Символьные выкладки
$6/14 \to 0.428571$	$6/14 \to 3/7$
$10 - 3 \to 7$	$10x - 3x \to 7x$
$\cos(3.14159) \to -0.999999$	$\cos(\pi) \to -1$
	$\cos(2x) \to \cos^2 x - \sin^2 x$
$\int\limits_0^1 e^x\,dx \to 1.71828$	$\int \frac{(x-2)\,dx}{x^3+1} \to -\frac{1}{\sqrt{3}} \operatorname{arctg}\left(\frac{2x-1}{\sqrt{3}}\right) -$ $- \ln(x+1) + \frac{1}{2}\ln\left(x^2 - x + 1\right)$
Работа с числами	Алгебраические упрощения

– результаты работы САВ точны и не приводят к ошибкам аппроксимации. Например, при решении системы уравнений:

$$x^4 + 2x^2y^2 + 3x^2y + y^4 - y^3 = 0,$$
$$x^2 + y^2 - 1 = 0,$$

желательно получить представление типа $\left(\sqrt{3}/2, -1/2\right)$, а не приближение (0.86602..., –0.5);

– в общем случае ввод для алгоритмов состоит из формул и выражений. Таким же является и результат. Именно это и позволяют СКА (системы компьютерной алгебры).

Необходимо отметить, что еще в 1844 г. Авгу́ста Ада Кинг (урожденная Байрон), графиня Лавлейс, которая перевела с французского языка на английский статью об аналитической машине Чарльза Бэббиджа и снабдила ее комментариями, позволившими потомкам называть ее первым программистом в мире, высказала мысль о том, что любая аналитическая точная процедура может быть выполнена счетной машиной [28].

За последние 60 лет произошел большой скачок в создании теоретической базы символьных и алгебраических расчетов на ЭВМ, были разработаны инструменты для обработки символов на компьютерах. "Символьные решатели" революционизировали направление мыслей людей относительно вычислений в математических проблемах. Компьютеры сейчас способны обрабатывать формулы так же хорошо, как числовые данные, обеспечивая аналитическую способность проникновения в суть дела

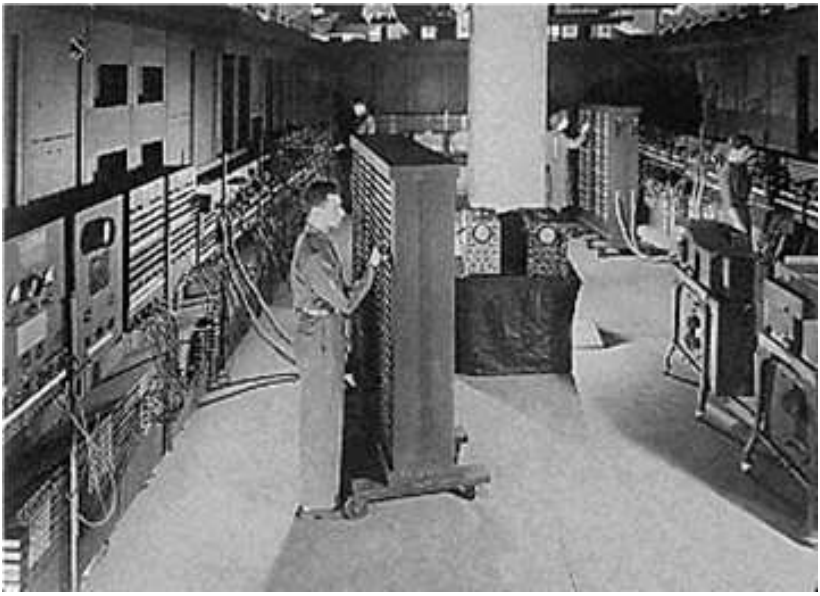


Рис. 1.2. Одна из первых ЭВМ ENIAC (начало разработки – 1937, готовность – 1943/45, списана – 1955, стоимость – $750000, скорость – 500 оп/сек)

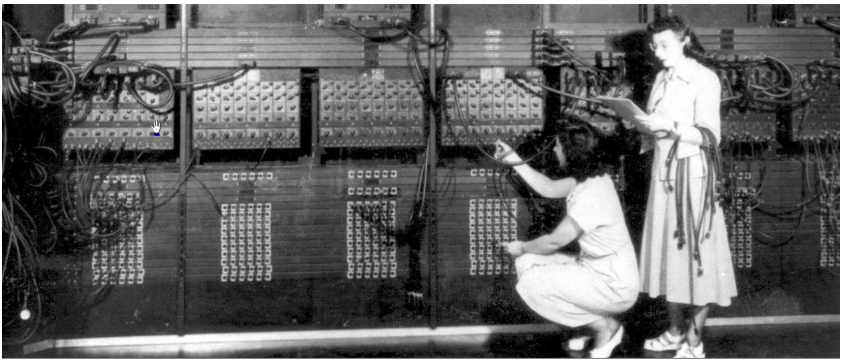


Рис. 1.3. Программирование первых ЭВМ

и точность результатов, которые раньше были недоступны. "Символические решатели" позволяют машине и пользователю связываться в тер-

минах алгебраических формул на языке ученых, причем и вход, и выход могут быть чисто алгебраическим или в численной форме специальной точности.

Все это привело к формированию новой дисциплины [1, 3, 7, 11, 19, 20, 29, 33]. Для ее именования используют различные слова: *символьные* и *алгебраические вычисления*, *символьные выкладки*, *символические* или *символьные преобразования*, *преобразования формул*, *компьютерная алгебра*[1] (КА) и т.д. К инструментам относят программы символьных преобразований, системы компьютерной алгебры и др. Вследствие того, что словосочетание "символьные вычисления" используется в различных сферах, например в работах по искусственному интеллекту, в настоящее время общепринятыми терминами являются "компьютерная алгебра" (computer algebra, CA) и "системы компьютерной алгебры" (computer algebra systems, CAS). Кроме того, в русском языке для последнего словосочетания существует синоним – *системы аналитических вычислений* (САВ).

С точки зрения оснований современная компьютерная алгебра – это раздел математики, в котором разрабатываются конструктивные методы решения математических задач, проводится анализ сложности этих методов и оценивается эффективность их реализации в компьютерных системах. Компьютерная алгебра использует методы общей алгебры, математического анализа, теории алгоритмов, математической теории сложности, теории графов и др.

С прикладной же точки зрения компьютерная алгебра есть та часть информатики (computer science), которая занимается разработкой, анализом, реализацией и применением алгебраических алгоритмов [7]. Известные западные специалисты A.M. Cohen, J.H. Davenport и A.J.P. Heck определяют САВ так: КА – это наука и технология, целью которой является автоматизация широкого спектра точных аналитических вычислений, используемых для решения математических проблем.

Необходимость создания и дальнейшего развития систем компьютерной алгебры не вызывает сомнений, так как многие математические методы исследования задач (прикладной) математики, механики, физики и других наук нередко требуют проведения значительных объемов аналитических выкладок, что практически всегда сопряжено с большой за-

[1] В последнее время в России отдельные авторы стали применять новый термин "компьютерная математика". Хотя он более точно отражает мощность и характеристики современных САВ, его использование ограничено и находится в противоречии с образовательными стандартами, где под "компьютерной математикой" понимается весь комплекс математических алгоритмов, ориентированных на компьютер и включающих, среди других, и компьютерную алгебру.

тратой сил и времени, а следовательно, и с появлением ошибок. Отметим также особую роль подобных систем в естественно-научном и математическом образовании. Они позволяют проверить результаты громоздких математических расчетов и наглядно представить сложные математические объекты. Конечно, даже современные САВ не могут гарантировать верный результат на сто процентов, но при их использовании поиск ошибок происходит на уровне верификации алгоритмов. Несомненно, что СКА должны, наконец, стать естественной средой обучения различным дисциплинам, а базовым курсам КА необходимо студентов учить с первого курса.

1.2. Первичные сведения о системе Maxima

Maxima – наиболее популярная из свободно распространяемых (лицензия GNU GPL[2]) система компьютерной алгебры, написанная на языке Lisp (на момент написания данного пособия последней доступной версией была 5.43.0, представленная в июне 2019 г.).

Рис. 1.4

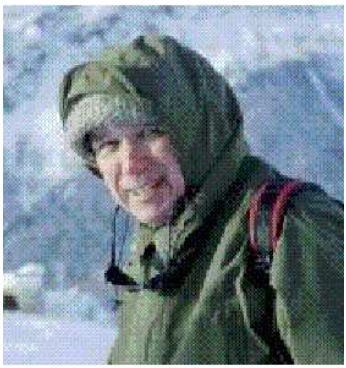


Рис. 1.5. У.Ф.Шелтер

Maxima базируется на версии 1982 г. системы Macsyma, разрабатывавшейся в МТИ с 1968 по 1982 г. в рамках проекта Project MAC, который финансировался Департаментом энергии (Department of Energy –

[2] *GNU General Public License* (Универсальная общественная лицензия GNU, Универсальная общедоступная лицензия GNU или Открытое лицензионное соглашение GNU) – лицензия на свободное программное обеспечение, созданная в рамках проекта GNU в 1988 г., по которой автор передает программное обеспечение в общественную собственность. Ее также сокращенно называют GNU GPL или даже просто GPL, если из контекста понятно, что речь идет именно о данной лицензии.

DOE) и другими государственными агентствами США. Вариант пакета, известный как DOE Macsyma, был создан и сопровождался профессором У.Ф. Шелтером[3] с 1982 г. до его смерти в 2001 г. В 1998 г. У.Ф. Шелтер получил от Департамента энергии разрешение опубликовать исходный код DOE-Macsyma под лицензией GNU GPL , а в 2000 г. он создал проект Maxima на сайте SourceForge.net (maxima.sourceforge.net) для поддержания и развития пакета DOE Macsyma, переименованного в Maxima, который в настоящее время продвигается независимой группой пользователей и разработчиков.

Заметим, что это единственная основанная на CAB Macsyma система, все еще публично доступная и имеющая активное сообщество пользователей благодаря своей открытости.

В системе Maxima отсутствуют все изменения и улучшения, произведенные в коммерческой версии пакета Macsyma с 1982 по 1999 г. (эти отличия оцениваются в 50 человеколет работы). Хотя основная функциональность подобна, программный код, зависящий от изменений, может не работать в среде Maxima. Кроме того, ошибки, исправленные в пакете Macsyma, скорее всего остаются в системе Maxima и наоборот.

СКА Maxima имеет в своем составе ядро системы, интерпретатор макроязыка (написанный на языке Lisp), библиотеки программных модулей и множество пакетов расширений (packages), написанных на макроязыке системы; включает широкий набор средств для проведения аналитических и численных вычислений и построения графиков. По спектру возможностей система входит в ведущую группу CAB и неплохо смотрится в сравнении даже с такими коммерческими системами, как Mathematica и Maple. Кроме того, она обладает высочайшей степенью переносимости. Это единственная из существующих систем аналитических вычислений, которая может работать под управлением всех основных современных операционных систем (MS Windows, Unix, BSD, Linux, MacOS X и др.) на компьютерах от самых мощных до PDA (КПК – карманных персональ-

[3] Уильям (Билл) Фредерик Шелтер (William Frederick Schelter, 1947-–2001) был профессором математики Университета Техаса в Остине, разработчиком языка Lisp и программистом. У.Ф. Шелтеру приписывается разработка и реализация GNU Common Lisp (gcl) языка Common Lisp and GPL-версии CAB Macsyma, называемой GNU Maxima. Шелтер был автором Austin Kyoto Common Lisp (AKCL) по контракту с IBM. AKCL сформировал основу другой CAB Axiom и в итоге стал языком GNU Common Lisp. Шелтеру также приписывается первый перенос GNU C на архитектуру INTEL 386, использовавший оригинальную реализацию ядра Linux. У.Ф. Шелтер получил степень PhD от Университета МасГилла в 1972 г. В научном плане он специализировался в некоммутативной теории колец и в вычислительной алгебре, включая автоматическое доказательство теорем геометрии. Летом 2001 г. в возрасте 54 лет он внезапно умер от сердечного приступа во время путешествия по нашей стране.

ных компьютеров, или "наладонников") и смартфонов.

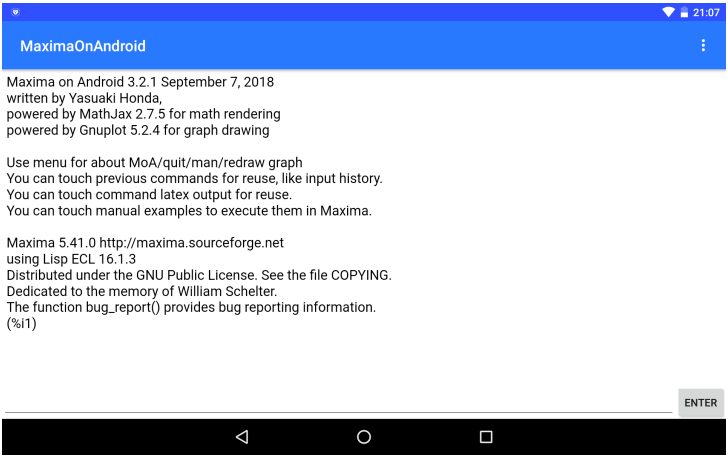


Рис. 1.6. Вид экрана планшета при работе с пакетом Maxima

Начиная с версии 5.34.0 система доступна (через Google Play) на смартфонах и планшетах, работающих под управлением ОС Android. Maxima on Android – это перенос системы Maxima в среду операционной системы Android, что стало возможным после импортирования С. Агено (S. Ageneau) языка Embeddable Common Lisp (ECL) на основе написания Я. Хонда (Yasuaki Honda) около тысячи строк кода Java и 50 строк HTML, включая код JavaScript, в ту же среду. В настоящее время САВ Maxima on Android (с очень небольшими изменениями в исходном коде стандартной системы) устойчиво работает на базе ECL, являясь сочетанием многих программ с открытым исходным кодом: ECL на Android, MathJax (приложение, позволяющее включать математические формулы на web-страницы, используя разметку LaTeX, MathML или AsciiMath, после чего формулы будут обработаны JavaScript-библиотекой и преобразованы в HTML, SVG[4] или MathML для отображения в любом современном браузере) и самой Maxima.

Можно использовать и онлайн-версию системы (проект Yamwi) в интернете (рис. 1.8), которая находится по адресу `maxima.cesga.es` (на этом же сайте можно найти краткое описание проекта и ссылку на расположе-

[4]SVG – язык разметки масштабируемой векторной графики, созданный Консорциумом Всемирной паутины и входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML.

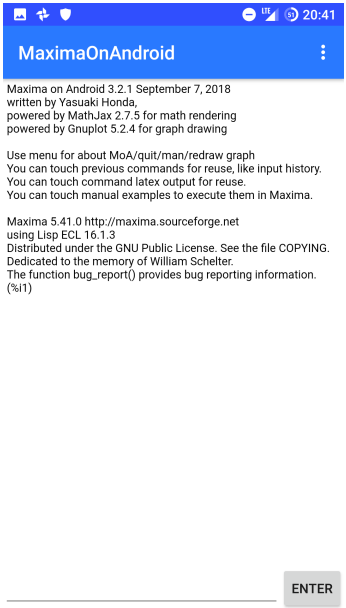


Рис. 1.7. Вид экрана смартфона при работе с пакетом Maxima

ние дистрибутива системы для установки на персональный смартфон).

Maxima в своей основе имеет интерфейс командной строки (рис. 1.9) и сама по себе не способна отображать форматированные результаты математических выкладок, ограничиваясь обычным текстовым уровнем. Для большинства пользователей, которые привыкли к продвинутым графическим интерфейсам пользователя (GUI), такая ситуация неудобна и может препятствовать освоению системы. К счастью, в настоящее время доступны более развитые графические интерфейсы. Но все равно надо помнить, что Maxima – некоммерческая система, а поэтому ее GUI не могут конкурировать с фронт-эндами коммерческих систем.

Система Maxima включает полнофункциональный язык программирования с алголоподобным синтаксисом и лиспоподобной семантикой. Поэтому эта система легко может быть использована для обучения программированию и компьютерной алгебре. Система достаточно надежна, без утечек памяти, имеет хорошую сборку "мусора", отладчик, а для проверки его работы с ним предоставляются сотни тестов.

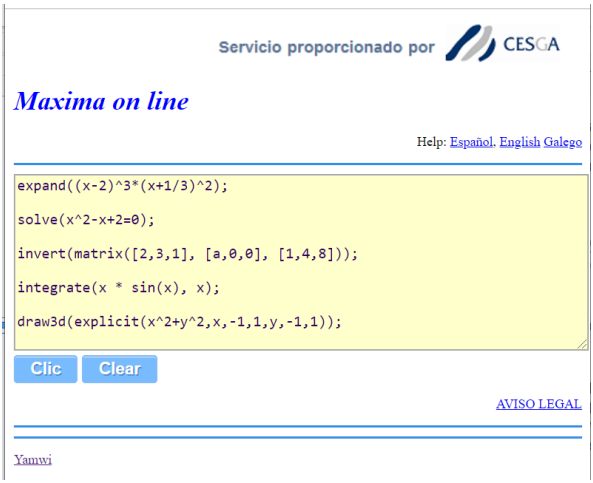


Рис. 1.8. Экран (сжатый) при работе с пакетом Maxima на ноутбуке через интернет

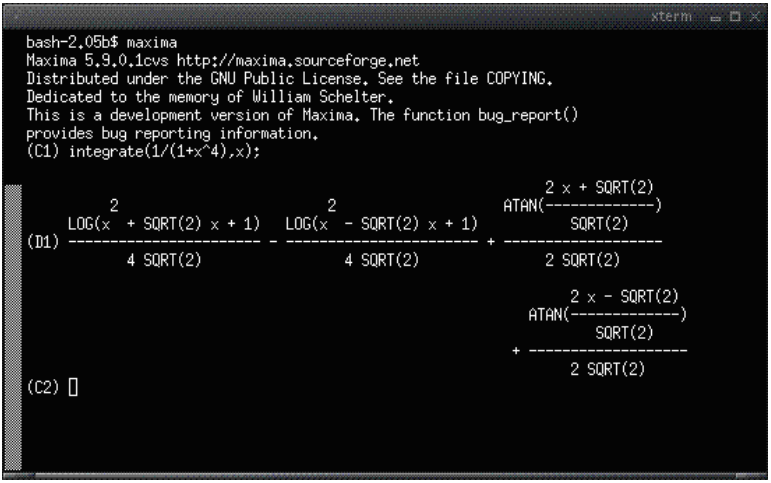


Рис. 1.9. Вид экрана при работе с пакетом Maxima через командную строку

Ядро CAB Maxima написано на языке Lisp[5]. При этом принцип об-

[5]Lisp (от List processing) – очень старый язык программирования, разработанный

работки пользовательских инструкций состоит в том, что когда вводится выражение, оно анализируется и вычисляется базовым интерпретатором Lisp. Но для работы с системой Maxima никаких знаний этого языка не требуется. Однако есть два случая, когда пользователь системы должен вступить в контакт с интерпретатором. Первый случай возникает при работе со справкой, а второй – в случае синтаксических ошибок. Обычно интерпретатор языка Lisp выражений Maxima пытается объяснить ошибку. Однако иногда он нормально не срабатывает и предоставляет весьма неясное сообщение об ошибке. В этом случае необходимо просто исправить вводимое выражение, проигнорировав содержание полученного сообщения.

Наряду со встроенным справочником существует значительное число электронных учебников по системе разного уровня, в т.ч. ряд документов на русском языке (см., например, на интернет-странице `maxima.sourceforge.net/ru/`). Вследствие того, что Maxima написана на языке Common Lisp, то ее особенности доступны для изучения, а коды легко расширяемы. К тому же для выполнения фрагмента Lisp-кода базовое ядро Lisp может быть вызвано из программы на входном языке системы Maxima.

Версии пакета

Maxima 5.10.0	сентябрь 2006	Maxima 5.28.0	август 2012
Maxima 5.11.0	декабрь 2006	Maxima 5.29.0	декабрь 2012
Maxima 5.12.0	май 2007	Maxima 5.30.0	июль 2013
Maxima 5.13.0	август 2007	Maxima 5.31.0	сентябрь 2013
Maxima 5.14.0	декабрь 2007	Maxima 5.32.0	декабрь 2013
Maxima 5.15.0	апрель 2008	Maxima 5.33.0	апрель 2014
Maxima 5.16.3	август 2008	Maxima 5.34.0	август 2014
Maxima 5.17.1	декабрь 2008	Maxima 5.35.0	декабрь 2014
Maxima 5.18.1	апрель 2009	Maxima 5.36.0	апрель 2015
Maxima 5.19.2	август 2009	Maxima 5.37.0	август 2015
Maxima 5.20.1	декабрь 2009	Maxima 5.38.0	апрель 2016
Maxima 5.21.1	май 2010	Maxima 5.39.0	декабрь 2016
Maxima 5.22.1	август 2010	Maxima 5.40.0	май 2017
Maxima 5.24.0	апрель 2011	Maxima 5.41.0	октябрь 2017
Maxima 5.26.0	декабрь 2011	Maxima 5.42.0	сентябрь 2018
Maxima 5.27.0	май 2012	Maxima 5.43.0	май 2019
		Maxima 5.44.0	июнь 2020

еще в 1958 г. в Массачусетском технологическом институте. Он до сих пор очень популярен в исследовательском сообществе по проблемам искусственного интеллекта.

2. ТЕХНИЧЕСКИЕ ВОПРОСЫ РАБОТЫ С СИСТЕМОЙ

2.1. Интерфейсы

Пакет имеет несколько графических интерфейсов пользователя и графических надстроек: простая консольная оболочка Terminal, wxMaximas (кроссплатформенная оболочка на основе wxWidgets, для нее имеется русифицированный вариант, см. следующий подраздел), XMaxima (включена в поставку ряда ОС), iMaxima, TEXmacs, Symaxx, Sage и др., но может работать и в режиме командной строки (с использованием псевдографики), как было сказано выше.

Terminal

Интерфейс Terminal – это оригинальный интерфейс для работы с Maxima. Хотя в некотором смысле все интерфейсы для Maxima можно было бы назвать терминальными интерфейсами, здесь имеется в виду именно интерфейс командной строки, не предоставляющий никаких излишеств. Он обладает наименьшими возможностями, но при этом наименее требователен к ресурсам.

Насколько удобен этот интерфейс, во многом зависит от того, какой Lisp использован для первоначальной компиляции системы Maxima. Если вы использовали значение по умолчанию, то не будет поддержки `readline` на терминале. Это означает, что нельзя будет использовать клавишу со стрелкой назад, чтобы перейти к середине выражения и изменить его, т.е. необходимо стирать все для перемещения назад. Это серьезное ограничение. В этом случае рекомендуется перекомпилировать выполнимый модуль с другой реализацией языка Lisp, такой как CLisp, который поддерживает `readline`, либо использовать один из других интерфейсов. Если вы решите использовать этот интерфейс, вы активируете его, просто набрав `maxima` в командной строке.

XMaxima

Альтернативной основному для данного пособия интерфейсу wxMaxima графической средой является XMaxima (рис. 2.1), которая обычно поставляется в дистрибутиве Maxima. Эта оболочка отличается простотой, менее удобна, но считается [27], что она более стабильна, чем wxMaxima, которая в настоящее время активно развивается.

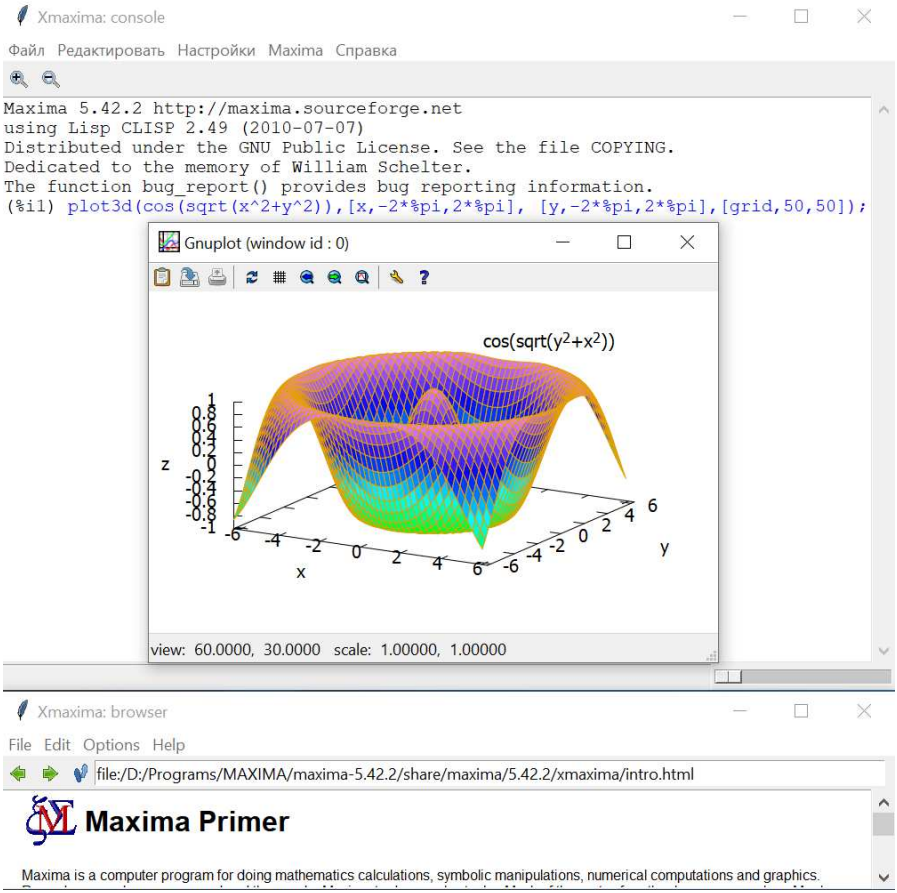


Рис. 2.1. Оболочка XMaxima в Windows

При старте открываются одновременно окно справочника и консоль команд. Пункты меню `File`, `Edit`, `Options` позволяют управлять сессией работы с системой Maxima, сохранять и запускать batch-файлы. В рабочий блокнот XMaxima можно встраивать графики в формате openmath (в зависимости от установки опции plot window). График в блокноте можно вращать, редактировать, сохранять в файл на внешний носитель.

Emacs

Emacs – семейство многофункциональных расширяемых текстовых

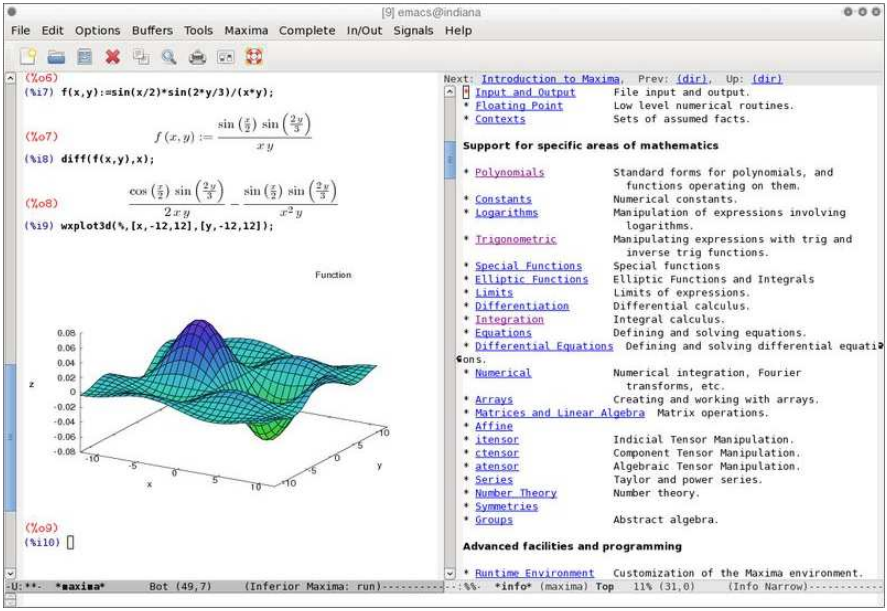


Рис. 2.2. Среда Emacs с сессией Maxima

редакторов. Первая версия Emacs была написана в 1976 г. Этот редактор также может использоваться в качестве интерфейса для среды Maxima (рис. 2.2), который включает четыре режима:

1) maxima – интерактивный режим, аналогичен консольной версии системы Maxima или интерфейсу XMaxima. Кроме того, через него осуществляется взаимодействие с выполнением Maxima в интерфейсе maxima-mode;

2) maxima-mode – пакетный режим, аналогичный обработке файлов Maxima. Удобство режима заключается в том, что на выполнение можно отправлять как весь файл (или буфер), так и отдельную его часть или одну строку;

3) iMaxima – интерактивный режим, аналогичен режиму maxima за исключением того, что вывод осуществляется не в текстовом, а в графическом виде, подобном интерфейсу wxMaxima;

4) EMaxima – интерфейс, реализуемый внутри документа LaTeX.

Emacs также может быть использован для написания программ на языке системы Maxima. Предусмотрен специальный режим, который позаботится о выделении синтаксиса, отступе и т.д. Отдельные части программы также могут автоматически отправляться в программу Maxima.

Наконец, текстовые команды и команды Maxima могут быть смешаны в режиме EMaxima ноутбука Maxima.

Режим maxima является основным режимом для написания кода Maxima. Находясь в этом режиме, Emacs предоставляет команды для перемещения по коду, выделения синтаксиса, выравнивания строк на соответствующем уровне, отправки частей кода процессу Maxima и предоставления справки по командам системы Maxima.

Интерфейс EMaxima – скорее не самостоятельный режим, а надстройка над средой LaTeX, которая интересна тем, кто использует Emacs для редактирования LaTeX-документов. В отличие от режима maxima, который предназначен для обычного изолированного запуска полноценной Maxima-сессии, здесь речь идет о возможности вставлять отдельные команды Maxima и, естественно, результаты их вычислений прямо в редактируемый LaTeX-документ [17].

Использовать интерфейс EMaxima удобно при создании объемных документов в среде LaTeX математического характера, которые предполагают включение результатов символьных вычислений.

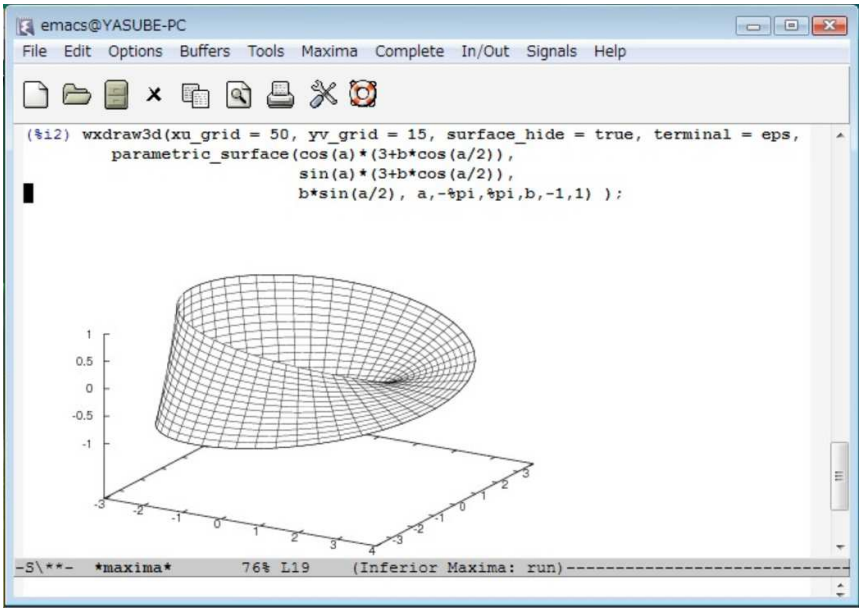


Рис. 2.3. Оболочка iMaxima

iMaxima

iMaxima представляет собой еще один графический интерфейс пользователя для системы компьютерной алгебры Maxima в среде Emacs. Для того чтобы красиво отобразить математическую формулу, iMaxima обрабатывает вывод Maxima с помощью графического редактора LaTeX. Он вставляет визуализированное изображение математической формулы в буфер. Также вставляются графики, сгенерированные программой Gnuplot. На рис. 2.3 скриншот Emacs с запуском iMaxima.

TeXMacs

GNU TeXMacs – бесплатная и свободная кроссплатформенная система WYSIWYG для подготовки и редактирования документов со специальными возможностями для ученых. Целью системы является создание унифицированной платформы для редактирования структурированных документов с содержанием различного типа (текст, иллюстрации, математические формулы, интерактивное содержимое и т.д.). Ядро для отображения использует высококачественные алгоритмы верстки для того, чтобы пользователь получал профессионально оформленные документы, пригодные для демонстрации с компьютера и с возможностью как печати, так и экспорта "выходных" документов в файлы ряда форматов, включая TeX/LaTeX и HTML/MathML. В состав системы входит текстовый редактор с поддержкой средств для редактирования формул, простых технических иллюстраций, программой для создания презентаций. TeXMacs может использоваться в качестве интерфейса для многих CAB, в частности Maxima (рис. 2.4).

TeXMacs хорошо локализован и полностью поддерживает русский язык, а также все возможности стандартного текстового процессора. Важной особенностью TeXMacs является возможность встраивать в текст документа сессии работы с различными математическими пакетами (в т.ч. и Maxima). В современных версиях TeXMacs при запуске сессии Maxima в главном меню появляется пункт `Maxima`, в котором предусмотрено выпадающее меню с перечнем основных команд Maxima.

Недостатками TeXMacs являются отсутствие русификации при работе в Maxima-режиме, а также проблемы на некоторых дистрибутивах с запуском сессии Maxima [17].

Sage

Sage (от англ. "мудрец") – система компьютерной алгебры, охватывающая много областей математики, включая алгебру, комбинаторику, вычислительную математику, математический анализ и др. [12, 17, 40].

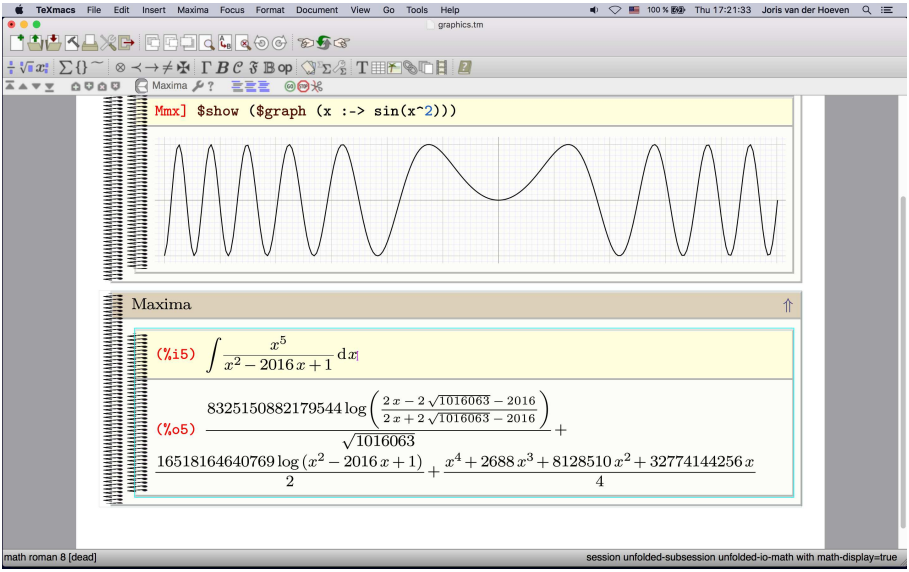


Рис. 2.4. Среда TEXMacs с сессией Maxima [41]

Первая версия Sage была выпущена в феврале 2005 г. в виде свободного программного обеспечения с лицензией GNU GPL. Первоначальной целью проекта было создание открытого программного обеспечения, альтернативного системам Magma, Maple, Mathematica и Matlab на основе большого количества готового математического программного обеспечения с открытым исходным кодом, но написанного на различных языках программирования, из которых наиболее встречаемыми являются C, C++, Fortran и Python. Разработчиком Sage является У. Стейн – математик Вашингтонского университета.

Структура Sage позволяет использовать средства следующих пакетов и систем: GAP, Singular, FLINT (алгебра); Singular (алгебраическая геометрия); MPIR, MPFR, MPFI, NTL, mpmath, Arb (арифметика произвольной точности); PARI/GP, NTL, mwrank, ECM (арифметическая геометрия); Maxima, SymPy, GiNaC, Giac, FriCAS (математический анализ); Symmetrica, Sage-Combinat (комбинаторика); ATLAS, BLAS, LAPACK, NumPy, LinBox, IML, GSL (линейная алгебра); NetworkX (теория графов); GAP (теория групп); PARI/GP, FLINT, NTL (теория чисел); GSL, SciPy, NumPy, ATLAS (численные расчеты); R, SciPy (статистика); IPython (интерфейс командной строки); ZODB, SQLite (базы данных); Sage Notebook, jsMath (графи-

ческий интерфейс); matplotlib, Tachyon, GD, Jmol (графика); ℙython (интерактивный язык программирования); Twisted (сетевые возможности); Sage Manifold (дифференциальная геометрия и тензорный анализ).

Наряду с функцией интегрирования, Sage включает достаточно развитые собственные возможности – многочисленные функции и структуры данных. Преобразование результатов, полученных, например, в системе Maxima, к структурам Sage может оказаться достаточно сложной задачей.

Symaxx

Symaxx является графическим интерфейсом для системы компьютерной алгебры Maxima в стиле CAB MathCAD. У него нет своего математического "интеллекта", всю работу выполняет система Maxima.

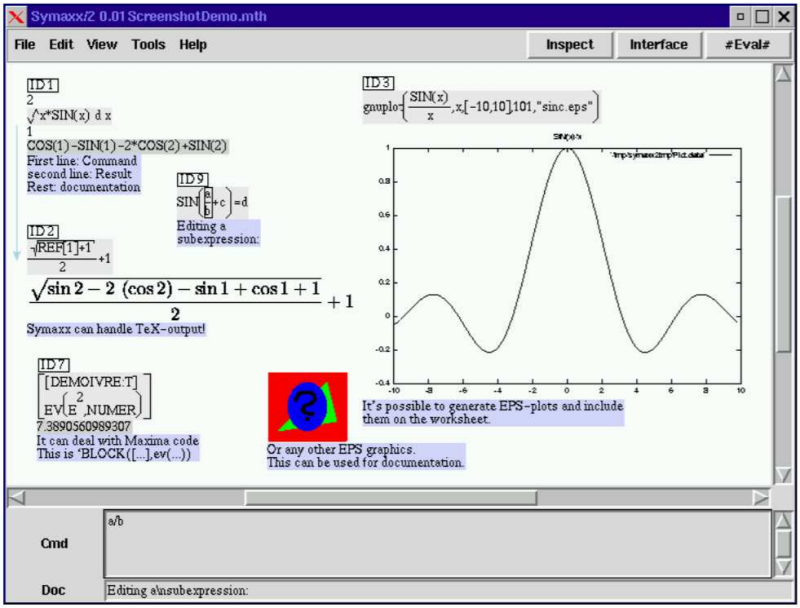


Рис. 2.5. Среда Symaxx с сессией Maxima [42]

На рис. 2.5 показан пример работы с Symaxx.

Чтобы создать новый объект, необходимо щелкнуть левой кнопкой мыши на свободном месте экрана (чтобы отменить выбор текущего выделения) и нажать среднюю кнопку мыши, чтобы создать новый объект в текущей позиции мыши.

Можно выбирать большинство подвыражений и редактировать их. Symaxx получает входные данные во внутреннем формате системы Maxima (списки в формате языка Lisp). Редактор распознает множество команд и отображает их удобным способом. Но пока Symaxx не включает графическое представление для всех команд Maxima. Если пользователь вводит неизвестную команду, Symaxx показывает ее так, как Maxima ее хранит (как функцию с одним или несколькими аргументами). Даже если графическое представление отличается от стандартной формы ASCII, все равно можно редактировать ее, используя обычный синтаксис Maxima.

Symaxx работает управлением Unix, Linux, SunOS и требует предустановленных сред Perl, `Perl Tk extension`.

2.2. wxMaxima. Система меню, панели символов

Данное пособие ориентировано на использование наиболее популярной оболочки wxMaxima, которая для настольных компьютеров, ноутбуков и нетбуков доступна по адресу

`http://wxmaxima.sourceforge.net/`

(обычно в стандартной установке системы Maxima оболочка wxMaxima присутствует).

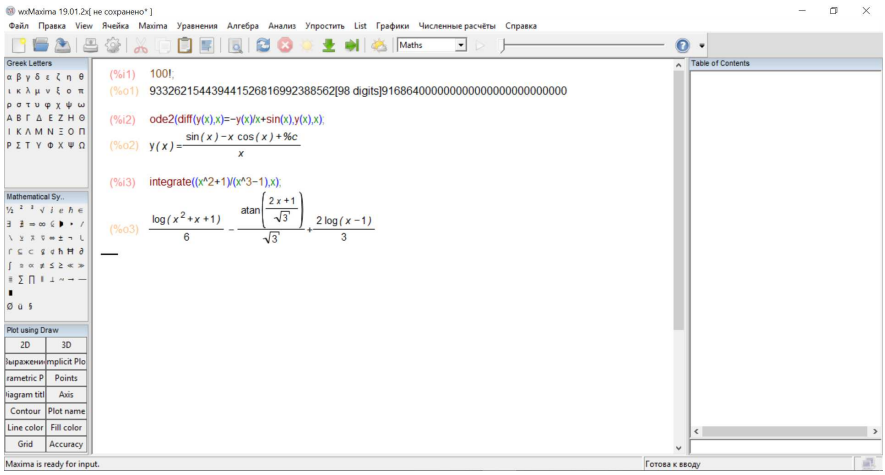


Рис. 2.6. Оболочка wxMaxima в MS Windows

Интерфейс wxMaxima состоит из блокнота (записная книжка, notebook) с меню и панели инструментов, которая позволяет неопытному пользователю выбрать соответствующую команду. Эти команды, а также их результаты печатаются в центральном окне блокнота (рис. 2.6). Естественно, такие команды также могут быть непосредственно введены с клавиатуры, а при необходимости отредактированы, что обеспечивает удобное рабочее пространство для опытного пользователя. Блокнот с результатами может быть сохранен в конце сеанса работы с системой, а затем загружен для последующего использования. Таким образом, wxMaxima позволяет пользователю создавать документ, который состоит из текста, расчетов и графиков.

В этом пособии при выводе результатов используется представление, похожее на вывод, формируемый в рабочем окне блокнота wxMaxima. Кроме того, будет дано объяснение некоторых особенностей этого интерфейса. Тем не менее далее в основном описывается интерфейс командной строки Maxima. Изучение работы с панелью инструментов ниже не затрагивается.

Далее примеры команд системы Maxima, которые набираются прямым жирным шрифтом, приводятся на желтом фоне в рамке:

```
(%i1)   <input>;
(%o1)       <output>
```

Символами (%i...) и (%o...) wxMaxima обозначает ввод и соответствующий ему вывод, причем счетчик вводов и выводов стартует с единицы в начале сеанса и увеличивается при каждом вводе (или исправлении) на 1.

wxMaxima обеспечивает удобный интерфейс. Ее блокнот организован с использованием так называемых *ячеек*. Несложно заметить, что открывающиеся квадратные скобки с треугольниками слева (если они отсутствуют, надо очистить подпараметр *Intelligently hide brackets* параметра *WorkSheet* в настройках системы) указывают, какие входные и выходные ячейки соответствуют друг другу. Всякий раз, когда ячейка в настоящее время редактируется или вычисляется, цвет этой полосы изменяется. Второй вариант будет в случае трудоемких расчетов.

wxMaxima предоставляет панель инструментов, где можно выбирать команды из меню. Это очень удобно для новичка, поскольку избавляет от необходимости запоминать синтаксис команд. С помощью интерфейса блокнота можно вставлять свои команды практически везде на странице, причем даже предыдущие команды могут быть отредактированы и выполнены снова. Конечно, при этом обновляется номер ячейки. Таким

образом, физический порядок ячеек может не соответствовать хронологическому порядку работы с системой.

Кроме того, можно вставлять заголовки разделов и текст, чтобы прокомментировать свои расчеты. Для этого необходимо воспользоваться соответствующим пунктом в меню Cell.

Когда сеанс работы с системой Maxima закончен, то блок можно сохранить, используя пункты меню File->Save. В начале нового сеанса содержание старого можно загрузить и снова использовать, если набрать File->Open.

После запуска системы с графическим интерфейсом wxMaxima появляется главное окно. Оно является стандартным для большинства приложений операционной системы Windows. Окно условно можно разделить на шесть областей. Структура окна, как правило, имеет следующий вид:

– строка заголовка, в которой представлена версия оболочки wxMaxima и располагается название программы и информация о том, сохранен ли рабочий документ (если документ сохранен, то прописывается его имя);

– панель меню программы, обеспечивающая доступ к основным функциям и настройкам. В ней находятся функции для решения большого количества типовых математических задач, разделенные по группам: уравнения, алгебра, анализ, упростить, графики, численные вычисления;

– панель инструментов, включающая кнопки для создания нового документа, быстрого сохранения документа, вызова окна справки, создания ячеек ввода, прерывания вычислений, кнопки для работы с буфером обмена и др. При задержании курсора мыши на инструментах появляется подсказка о предназначении инструмента;

– рабочая область, содержащая непосредственно сам документ, программу и комментарии к ней. В этой области формируются ячейки ввода и выводятся результаты выполненных команд;

– полосы прокрутки;

– панель с кнопками, включающая набор кнопок для быстрого вызова некоторых команд: упростить, решить уравнение или систему, построить график и др.;

– строка состояния, в которой отображается информация о текущем состоянии программы.

Ниже строки заголовка расположена строка главного меню. В строке этого меню сгруппированы все те действия, которые выполняются в окне Maxima. При нажатии на любой пункт главного меню появляется список выпадающего меню первого уровня. Некоторые пункты таких меню содержат выпадающие подменю второго уровня. Такие пункты помечены

справа треугольником. Рассмотрим некоторые пункты главного меню.

В меню Файл сгруппированы действия, связанные с открытием, сохранением, загрузкой пакетов, печатью документов, выходом из программы. Справа от названия команд показаны "быстрые клавиши" для вызова данных команд. Команды, выделенные серым цветом, в момент просмотра недоступны.

В меню Правка сгруппированы действия, связанные с редактированием текста, копированием и поиском в документе. Название команд в данном меню достаточно простые и обозначают результат действия, которые выполняются при нажатии на соответствующие пункты меню.

В меню Ячейка представлены средства для выполнения вычислений и редактирования документа. Например, команда Evaluatr All Visible Cells (Ctrl+R) применяется после редактирования некоторой ранее введенной строки. После такого редактирования необходимо пересчитать все строки, результат в которых зависит от исправленной строки.

После того, как система загрузилась, можно приступать к вычислениям. Для этого сначала следует добавить *ячейку ввода*, в которую вводится команда системе выполнить какое-либо действие. Для этого в подменю Ячейка нужно нажать кнопку Insert input cell (или щелкнуть курсором "мыши" на пустом месте рабочей области и что-либо ввести). В результате в рабочей области будет сформирована ячейка ввода.

Ячейки, в которых размещаются выходные данные (списки значений, выражения), называют *ячейками вывода*.

В системе Maxima предусмотрена возможность ввода сразу нескольких команд в одной строке. Для этого одна команда от другой отделяется символом ";"или "$". При этом формируется одна строка ввода и столько строк вывода, сколько результатов выполнения команд было сформировано.

Задание команды в ячейке ввода и формирование ячейки вывода при нажатии комбинации клавиш Ctrl+Enter называют *отдельной сессией* работы с системой.

Укажем некоторые основные приемы работы с отдельными сессиями работы.

1°. Для задания команды системе нужно в строке ввода задать само выражение и вычислить его, закончив ввод нажатием комбинации клавиш Ctrl+Enter. В результате образуется ячейка ввода и соответствующая ей ячейка вывода.

2°. Если навести курсор "мыши" на левый верхний угол квадратной скобки и нажать левую кнопку "мыши", то произойдет сворачивание части документа, относящейся к этой команде. Повторный щелчок "мыши"

вернет ячейку в начальное состояние.

3°. Для добавления ячеек (в т.ч. при копировании) в заданное место документа необходимо щелкнуть левой кнопкой мыши в необходимом месте (например, между существующими ячейками) так, чтобы появилась горизонтальная черта. После чего необходимо набрать на клавиатуре необходимый текст или сделать вставку из буфера (Ctrl-V).

4°. Если нужно пересчитать значение введенного выражения (или выражения после внесенных в него исправлений), то для этого нужно установить курсор в ячейке ввода и нажать комбинацию клавиш Ctrl+Enter.

5°. Для перевычисления ячеек всего документа можно воспользоваться командой Evaluate all cells пункта меню Ячейка.

6°. Для удаления ячейки необходимо ее выделить и нажать на клавиатуре клавишу Delete.

7°. В среде wxMaxima можно добавлять в документ текстовые комментарии. Для этого необходимо выбрать пункт меню Ячейка->Новый комментарий, после чего с клавиатуры набирается желаемый текст.

8°. С помощью пункта меню Правка->Настройка можно изменять конфигурацию графического интерфейса системы, применяемую по умолчанию. После нажатия указанного пункта меню появляется окно с настройками, где можно выбрать язык, вид панели инструментов и ряд дополнительных опций: проверять баланс скобок в тексте, копировать в буфер при выделении и др.; установить применяемый по умолчанию шрифт и его размер, а также стиль оформления (цвет и начертание) различных данных, используемых в документе: переменных, констант, чисел, имен функций и др.

2.3. Установка. Первый запуск

Последовательность действий при установке системы Maxima для работы в среде ОС MS Windows следующая:

1°. Загрузить последнюю актуальную версию Maxima. Во время написания данного пособия это 5.43.0. Для работы на компьютере с 32-битовым процессором нужно скачать первый файл, заканчивающийся `win32.exe`. Если в наличии 64-битовый процессор, то требуется версия дистрибутива системы с окончанием `win64.exe`.

2°. Стартовать загруженный файл, чтобы начать процесс установки. После старта программы–установщика необходимо ответить на ее вопросы в последовательно появляющихся окнах. Диалог установки достаточно четкий и понятный. По окончании процесса необходимо ознакомиться с содержанием файла README и License, в которых представлена информация, которую полезно знать.

Процесс установки может занять несколько минут, поскольку программа также настраивает дополнительные программы, требуемые для корректной работы Maxima.

3°. Как только процесс установки закончится, необходимо перейти в каталог, в котором была установлена система Maxima. В этом каталоге должны присутствовать подкаталоги с именами `bin` и `wxMaxima`, в которых находятся установленные программы.

4°. Необходимо проверить работоспособность установленного программного обеспечения. Для этого нужно запустить все три интерфейса для системы Maxima: `maxima` (интерфейс командной строки) и XMaxima, которые находятся в каталоге `bin`, wxMaxima в подкаталоге `wxMaxima`, причем wxMaxima и XMaxima можно стартовать из меню Пуск. При нормальном запуске должно появиться приглашение (`%i1`), после чего рекомендуется выполнить некоторые основные команды.

Размер дистрибутива – примерно 123 Мб. На HDD требуется порядка 670 свободных Мб.

При старте wxMaxima открывается окно для ввода команд и получения результатов, а также несколько панелей инструментов, если это указано при установке. Поверх основного появляется малое информационное окно "Tips of the day" ("Советы дня"), которое после ознакомления с информацией в нем (или сразу же) можно закрыть. Явное текстовое приглашение отсутствует, но есть мерцающий отрезок прямой у левого края окна.

Для начала ввода команды достаточно щелкнуть курсором "мыши" ниже этого отрезка. После этого уже можно вводить команды входного языка, оканчивая их символом ";" или "$". После нажатия клавиш `Shift+Enter` монитор интерфейса запустит интерпретацию (распознавание) ввода, вызовет вычислительное ядро Maxima (если это необходимо), которое выполнит некоторые расчеты и возвратит результаты и управление монитору. Монитор выведет результаты на экран ниже ввода, если вывод не подавлен символом "$". При этом ввод и вывод получат имена вида (%i...) и (%o...) с соответствующими номерами и будут охвачены открывающими скобками для указания границ так называемых ячеек (cells).

После появления нескольких ячеек можно вставлять новые между имеющимися. Для этого курсор "мыши" надо поставить на промежуток между двумя последовательными ячейками и щелкнуть левой клавишей "мыши". Ввод содержания такой ячейки производится обычным образом.

Редактор интерфейса позволяет копировать фрагменты строк ввода и результатов и вставлять их в другие позиции.

Завершить сессию wxMaxima можно одним из следующих способов: 1) нажать кнопку "×"в правом верхнем углу окна; 2) выбрать пункт меню `File->Выход`; 3) набрать и выполнить команду "quit();". В первых двух случаях основное окно закрывается. При этом если сеанс не сохранен, монитор предлагает это сделать. В третьем случае окно не закрывается, а монитор предлагает рестартовать систему Maxima.

2.4. Используемые символы. Латиница, греческий алфавит и кириллица

Как известно, алфавит языка включает совокупность символов, которые используются для записи команд. При работе с системой Maxima в среде wxMaxima можно использовать: строчные и заглавные буквы латинского алфавита; строчные и заглавные буквы греческого алфавита; строчные и заглавные буквы кириллицы; арабские цифры от 0 до 9; специальные знаки (+, -, *, /, ^, ., ,, :, ;, <, =, >, [,], (,), {, },!, |, #, $, %, ?, @, ’, ‘, ", _,) и др. Эти символы используются для записи команд языка, которые состоят из имен системных и пользовательских переменных, имен операторов, встроенных и пользовательских функций; разделителей и т.д.

Компьютеры традиционно используют 8-битные символы: это позволяет использовать максимум 256 различных символов, среди которых буквы латинского алфавита, цифры, управляющие символы (перевод строки, конец строки и т.д.) с добавлением небольшого числа специфических для конкретного языка. При этом для большинства стран выбранная кодовая страница из 256 символов не содержит греческих букв. Чтобы преодолеть это ограничение, был изобретен Unicode: метод включения символов, которые обычно не используются в английском языке, в текст, который (при условии, что используется только базовая форма латинских символов) выглядит как простой 8-битный ASCII.

Maxima позволяет использовать символы Unicode, если она работает в среде языка Lisp, который поддерживает эти символы, а также если библиотека wxMaxima, на которой построена библиотека wxMaxima, также поддерживает Unicode-символы. Это возможно вследствие того, что среду wxMaxima можно создавать с поддержкой Unicode.

Для включения в команду языка системы Maxima буквы греческого алфавита необходимо записать ее название (строчной или заглавной) латиницей и вставить символ процента % в начало названия:

```
(%i1)   [%alpha, %beta, %omega, %Omega, %Alpha, %lambda, %Lambda];
(%o1)       [α, β, ω, Ω, A, λ, Λ]
```

Буквы греческого алфавита можно использовать в строках, комментариях и в именах переменных.

```
(%i2)   %alpha: 5$ %alpha^3;
(%o2)       125
```

В Maxima / wxMaxima возможно использование кириллицы, буквы которой корректно отображаются не только в текстовых ячейках, строках и комментариях, но и на графиках Gnuplot (рис. 2.7). Кроме того, кириллицу можно использовать и в именах переменных, а поэтому и в командах.

```
(%i3)   xy: [[10, .6], [20, .9], [30, 1.1], [40, 1.3], [50, 1.4]]$
(%i4)   plot2d([[discrete,xy], 2*%pi*sqrt(1/980)], [1,0,50], [style,
points, lines], [color, red, blue], [point_type, asterisk], [legend,
"Эксперимент", "Теория"], [xlabel, "Длина маятника (см)"], [ylabel,
"Период (с)"])$
```

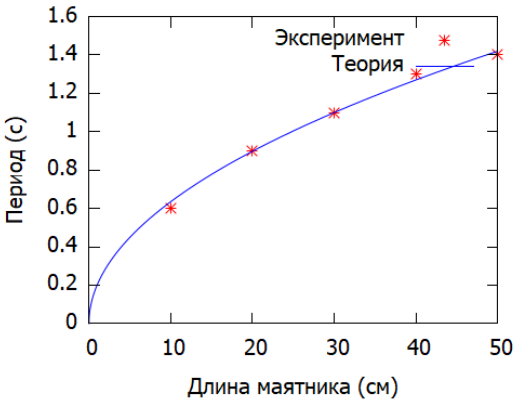


Рис. 2.7

```
(%i5)   радиус: 3$ площадьКруга: %pi*радиус^2;
(площадьКруга)       9 π
```

2.5. Представление и ввод простейших команд и числовых данных

После запуска среды wxMaxima появляется окно интерфейса, в верхней графической части окна которого указано, какая загружена версия.

Разделителями команд являются символы ";" и "$". Использование последнего приводит к подавлению вывода результатов выполнения команд на консоль. Даже если вывод результата подавлен, в памяти он сохраняется под соответствующим именем. В отличие от обычных математических текстов для указания на операцию умножения символ * нужно вставлять обязательно. При этом при выводе на экран результатов вместо такого символа формируется небольшой пробел.

Как в любом стандартном языке программирования, при работе с системой Maxima точка "." разделяет целую и дробную части в нецелых числах, а запятая "," – формальные и фактические аргументы функций (например, `max(sin(x),cos(x))`), элементы списков, множеств, массивов и др. Круглые скобки являются ограничителями параметров функций, квадратные – элементов списков, фигурные – элементов множеств. Кроме того, круглые скобки определяют последовательность выполнения операций.

Системные и пользовательские имена в среде wxMaxima чувствительны к регистру, т.е. прописные и строчные буквы в ней различаются.

Предопределенные константы типа e, π и другие указываются с помощью имени на латинице с добавлением знака процента %: `%e`, `%pi`, ... Те же имена без знака % можно использовать как обычные переменные, значение которых не связано со смыслом их имени, т.е. `g(pi)`, вообще говоря, это не то же самое, что `g(%pi)`.

Любой ввод команды и соответствующий вывод приводят к формированию ячеек ("cells") в окне интерфейса, которые выделяются графической левой квадратной скобкой, охватывающей соответствующую ячейку.

После ввода каждой команде присваивается порядковый номер `(%i1)`, `(%i2)`, `(%i3)` и т.д. Результаты вычислений имеют соответственно порядковые номера `(%o1)`, `(%o2)`, `(%o3)` и т.д. (последние, если есть результаты, но нет присваивания их некоторым переменным; иначе левая часть ячейки вывода выглядит как `(<имя переменной>)`, где "i" – сокращение от англ. "input" (ввод), а "o" – от англ. "output" (вывод)[1]. Это позволяет в дальнейшем при записи новых команд сослаться на ранее записанные (например, `(%i1)+(%i2)` означает добавление к выражению первой команды выражения второй с последующим вычислением результата). Также можно использовать и номера результатов расчетов, например, в виде `(%o1)*(%o2)`. Кроме того, символ % позволяет сослаться на результат последней выполненной команды.

[1]Заметим, что некоторые команды могут инициировать создание нескольких ячеек с результатами расчетов. В этом случае номера ячеек вывода могут формироваться различными способами.

При записи чисел используются следующие правила: 1) перед отрицательными числами ставится знак минус "-"; 2) числители и знаменатели рациональных дробей разделяются при помощи символа "/".

Для обозначения арифметических операций в системе Maxima используются следующие символы: "+" – сложение, "-" – вычитание, "*" – умножение, "/" – деление. Возведение в степень можно обозначать тремя способами: "^", "^^", "**". Извлечение корня степени n записывают в форме "^^(1/n)" (заметим, что для извлечения квадратного корня применяется функция sqrt).

Для хранения результатов промежуточных расчетов применяются переменные. Присваивание значения переменной осуществляется с использованием символа ":": a: 1 (оператор присваивания). Есть еще один оператор присваивания, обозначаемый ::. Оператор :: в отличие от : вычисляет и левую, и правую части оператора.

```
(%i1)   x: 'xx;
(%o1)       xx
```

```
(%i2)   x:: sin(y);
(%o2)       sin(y)
```

```
(%i3)   xx;
(%o3)       sin(y)
```

Для записи математической или любой другой встроенной функции, а также функции пользователя необходимо указать ее название, а затем в круглых скобках записать через запятую фактические значения параметров (аргументов). Например, tan(x).

Когда выражение вычисляется, то интерпретатор пакета заменяет все конструкции, которым присвоены значения, этими значениями, но оставляет все остальные конструкции без изменений.

```
(%i4)   fff(3*bar+2*bar) + bar^2 + log(%e);
(%o4)       fff(5 bar) + bar^2 + 1
```

Заметим, что из-за такой идеологии интерпретатор языка не может обнаружить и проинформировать пользователя о возможной опечатке. Если этот пользователь в строке ввода набирает синтаксически правильное имя переменной или функции, но на самом деле набранное имя отличается от желаемого, то неверное имя просто возвращается как оно есть, т.е. без каких-либо изменений.

Выше в этом подразделе введен оператор присваивания ":", а в подразделе 4.3 будет определен оператор функции пользователя :=. Конечно, они могут быть использованы для присваивания переменной любого выражения или определения функции. При этом если для некоторой переменной определено значение, то оно и используется в расчетах вместо этой переменной. Если же значение не определено, в вычислениях участвует имя этой переменной. В последнем случае GUI wxMaxima пытается вывести на экран такие переменные в приятной форме, например, имена, совпадающие с написанием греческих букв, представляются с помощью соответствующего символа греческого алфавита.

```
(%i5)   x: (a+beta)*(a-beta);
```
$$(\%o5) \qquad (a-\beta)\,(\beta+a)$$

```
(%i6)   y: x-a^2;
```
$$(\%o6) \qquad (a-\beta)\,(\beta+a)-a^2$$

```
(%i7)   foo(u):=log(u)-x+alpha;
```
$$(\%o7) \qquad foo(u) := \log(u)-x+\alpha$$

```
(%i8)   foo(y);
```
$$(\%o8) \qquad \log\left((a-\beta)\,(\beta+a)-a^2\right)-x+\alpha$$

Когда выражение присвоено некоторой переменной, скажем x, оно остается неизменным до тех пор, пока не будет удалено (или не будет остановлена работа пакета Maxima). Но в ситуации, когда пользователь проводит сеанс работы с пакетом на своем компьютере в течение нескольких часов и время от времени использует его, например, в качестве символического калькулятора, он может забыть о том, что x имеет некоторое значение, и из-за этого получить сомнительные результаты или же непонятые сообщения об ошибках (в худшем случае это могут быть ошибочные результаты, которые на первый взгляд выглядят правильными). Поэтому настоятельно рекомендуется уничтожать (kill) определения переменных и функций, которые устарели.

```
(%i9)   /* ... Выполнение множества вычислений ... */       /*
Теперь требуется решить уравнение относительно x и y */
eq: a*x+b*y=c;
```
$$(\%o9) \qquad b\left((a-\beta)\,(\beta+a)-a^2\right)+a\,(a-\beta)\,(\beta+a)=c$$

```
(%i10)   /* Проблема! Получено не то уравнение. Поэтому: */
         kill(all)$
```

```
(%i11)   eq: a*x+b*y=c;
```
$$(\%o11) \qquad b\,y + a\,x = c$$

Если в результате выполнения некоторой команды присутствуют рациональные числа, а хотелось бы видеть числа с плавающей точкой, то необходимо воспользоваться опцией `numer`

```
(%i12)   s: x/3+x^2/7+x^3/9$ [s, ev(s,numer)];
```
$$(\%o12) \qquad [\frac{x^3}{9}+\frac{x^2}{7}+\frac{x}{3}, 0.1111111111111111\,x^3+0.1428571428571428\,x^2+0.3333333333333333\,x]$$

Любое имя, имеющее значение, будь то имя ячейки или любое другое, которому мы назначили выражение с помощью `":"`, можно очистить с помощью функции `kill`. Ею также можно очистить всю память разом, набрав `kill(all)`. После этого нумерация ячеек вновь пойдет с единицы. В данном пособии примеры и результаты нумеруются последовательно в пределах подраздела.

2.6. Перевод сложных выражений в линейную форму записи

Одна из первых трудностей, с которой сталкивается начинающий пользователь системы Maxima, является запись сложных выражений, содержащих степени, дроби и другие конструкции, в линейной форме (в текстовой форме записи при помощи символов ASCII в одну строку). Для облегчения процесса преодоления трудностей можно дать несколько советов:

1°. В отличие от стандартной математической записи, в которой знак умножения в некоторых случаях можно опускать, при вводе команд языка системы Maxima это недопустимо, т.е. знак умножения нужно ставить обязательно. Несмотря на то, что в окне, например, интерфейса wxMaxima в качестве результата можно увидеть фрагмент $3\,f(x)$, этот фрагмент при вводе должен выглядеть как `3*f(x)`.

2°. Если не ясно, как интепретатор ввода воспримет некоторое выражение со сложной структурой, для задания необходимой последовательности (старшинства) вычислений (а иногда и для большей наглядности) рекомендуется поставить дополнительные круглые скобки. Если числитель дроби состоит из нескольких слагаемых, то его необходимо заключать в скобки. Если знаменатель выражения – это сумма или произведение нескольких подвыражений, то его также записывают в круглых

скобках. При возведении в степень необходимость использования дополнительных скобок для основания и показателя степени соответствует рекомендациям для записи знаменателя. Но, вообще говоря, в реализации системы Maxima используются стандартное для математики старшинство операций: сначала вычисляются функции, потом выполняется возведение в степень, затем операции деления и умножения и только потом сложение и вычитание. Особенности могут быть только для специализированных операций.

3°. Ни в математической записи (кроме общих рассуждений), ни при вводе выражений в Maxima нельзя функции отделять от их аргументов (если таковые имеются). При этом правила записи выражений в математике в некоторых случаях допускают, например, возводя функцию в степень, ставить показатель степени сразу после наименования функции: $\ln^2(x^4+1)$. В рамках расписывания команд в Maxima такое недопустимо в принципе. Поэтому при возведении в степень можно взять всю функцию с аргументами в скобки, а потом уже возводить полученную конструкцию в нужную степень: `(log(x^4+1))^5`.

Выражение	*Ввод*
$\left(1+\frac{3}{n}\right)^{6n}$	`(1+3/n)^(6*n)`
$\frac{\sin x-\cos^2 y}{x+y}$	`(sin(x)-(cos(y))^2)/(x+y)`

4°. Все требуемые аргументы функций записываются в скобках через запятую, например, `h(x,y,z)`.

5°. Необходимо внимательно отслеживать и контролировать результаты ввода сложных выражений (со скобками). Важно помнить, что при работе в интерфейсе wxMaxima интерпретатор ввода для открывающейся круглой скобки автоматически и без пробела дописывает соответствующую закрывающую. То же самое он делает с квадратными и фигурными скобками.

6°. Если требуется записать в формате ввода сложное выражение, то так же, как в математике, это выражение можно разбить на несколько более простых подвыражений, назвав их новыми именами, а затем, используя новые имена и соответствующие операции, собрать все части для вычисления требуемого выражения.

7°. Ячейки с неправильным или неактуальным содержимым можно удалить с экрана так: отметить курсором "мыши" квадратную скобку, охватывающую требуемую ячейку, и нажать клавишу `Del`.

2.7. Примеры работы с системой. Вывод результатов на экран

1°. Действия с числами:

– раскрытие скобок

```
(%i1)   expand((x+y)^6);
```

$$(\%o1) \qquad y^6 + 6\,x\,y^5 + 15\,x^2\,y^4 + 20\,x^3\,y^3 + 15\,x^4\,y^2 + 6\,x^5\,y + x^6$$

– разложение на множители выражения

```
(%i2)   factor(x^6-1);
```

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

– разложение на множители целого числа

```
(%i3)   factor(123412341231234);
```

$$(\%o3) \qquad 2\ 3^2\ 11\ 43\ 751\ 19301231$$

```
(%i4)   factor(2^(2^5)+1);
```

$$(\%o4) \qquad 641\ 6700417$$

– факториал

```
(%i5)   100!;
```

(%o5) 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

– перевод числа π в расширенную форму числа с плавающей точкой

```
(%i6)   bfloat(%pi);
```

(%o6) 3.1415926535897932b0

```
(%i7)   block([fpprec:1000],bfloat(%pi));
```

(%o7) 3.14159265358979323846264338327[943 digits]3001927876611195909216420199b0

– формирование непрерывной дроби

```
(%i8)   cfdisrep([1,2,3,5,2]);
```

$$(\%o8) \qquad 1 + \cfrac{1}{2 + \cfrac{1}{3 + \cfrac{1}{5 + \cfrac{1}{2}}}}$$

```
(%i9)   bfloat(%);
(%o9)      1.4320987654320991b0
```

2°. Матричная алгебра:

– задание матриц

```
(%i10)  a: matrix([1,2],[3,4]);
(a)     ⎛1  2⎞
        ⎝3  4⎠
```

```
(%i11)  b: matrix([2,3],[3,2]);
(b)     ⎛2  3⎞
        ⎝3  2⎠
```

– умножение матриц

```
(%i12)  a.b;
(%o12)     ⎛8   7 ⎞
           ⎝18  17⎠
```

– определение функции

```
(%i13)  h[i,j]:=1/(i+j);
(%o13)     h[i, j] := 1/(i + j)
```

– заполнение элементов матрицы

```
(%i14)  c: genmatrix(h,3,3);
(b)     ⎛1/2  1/3  1/4⎞
        ⎜1/3  1/4  1/5⎟
        ⎝1/4  1/5  1/6⎠
```

– вычисление определителя

```
(%i15)  determinant(a);
(%o15)     -2
```

– задание матрицы

```
(%i16)  d: matrix([2,3],[5,6])$
```

– приведение матрицы к треугольному виду

```
(%i17)  echelon(d);
(%o17)     ⎛1  3/2⎞
           ⎝0   1 ⎠
```

– обращение матрицы

```
(%i18)  invert(d);
(%o18)  matrix([-2, 1], [5/3, -2/3])
```

– вычисление собственных чисел и векторов матрицы

```
(%i19)  eigenvectors(d);
(%o19)  [[[4 - sqrt(19), sqrt(19) + 4], [1, 1]], [[[1, -(sqrt(19) - 2)/3]], [[1, (sqrt(19) + 2)/3]]]]
```

3°. Решение линейных уравнений:

```
(%i20)  linsolve([3*x+4*y=7, 2*x+4*y=13], [x,y]);
(%o20)  [x = -6, y = 25/4]
```

```
(%i21)  eq1: x^2 + 3*x*y + y^2 = 0;
(eq1)   y^2 + 3 x y + x^2 = 0
```

```
(%i22)  eq2: 3*x + y = 1;
(eq2)   y + 3 x = 1
```

```
(%i23)  solve([eq1, eq2]);
(%o23)  [[y = -(3 sqrt(5) + 7)/2, x = (sqrt(5) + 3)/2], [y = (3 sqrt(5) - 7)/2, x = -(sqrt(5) - 3)/2]]
```

4°. Программирование:

– печать в цикле

```
(%i24)  for a: 0 thru 4 step 2 do ldisplay(a);
(%t24)  a = 0
(%t25)  a = 2
(%t26)  a = 4
(%o26)  done
```

– вычисление суммы

```
(%i27)  s:0; for i:1 while i<=10 do s:s+i;
```

```
(%i28)  s;
(%o28)  55
```

– получение чисел Фибоначчи

```
(%i29)   fib[0]:0; fib[1]:1; fib[n]:=fib[n-1]+fib[n-2]; fib[20];
(fib[0])       0
(fib[1])       1
(fib[n])       fib_n := fib_{n-1} + fib_{n-2}
(%o29)       6765
```

5°. Построение графиков:
– 2D-графики (рис. 2.8–2.11);

```
(%i30)   plot2d(sin(x)/x,[x,-5,5])$
```

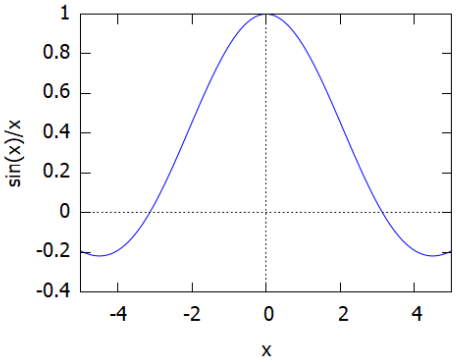


Рис. 2.8

```
(%i31)   plot2d(sec(x),[x,-2,2],[y,-20,20],[nticks,200])$
```

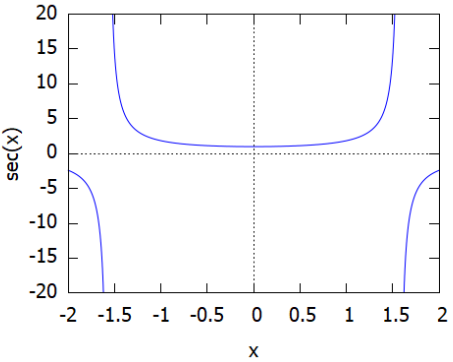


Рис. 2.9

```
(%i32)   plot2d([parametric,cos(t),sin(t),[t,-%pi*2,%pi*2]])$
```

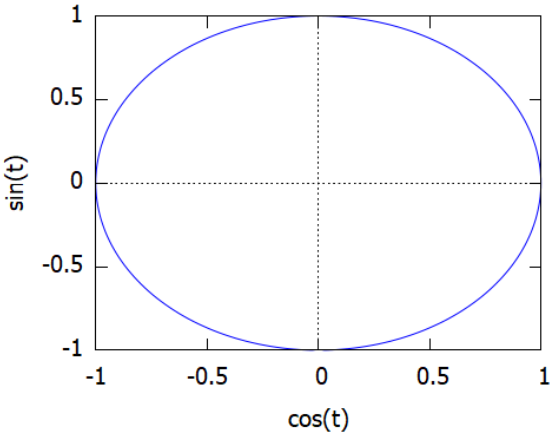


Рис. 2.10

```
(%i33)   plot2d([x^3+2,[parametric,cos(t),sin(t),[t,-5,5]]],
[x,-3,3])$
```

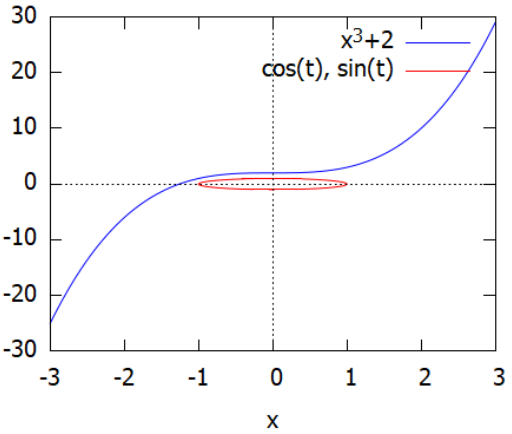


Рис. 2.11

– 3D-графики (рис. 2.12–2.15).

```
(%i34)   plot3d(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2),[x,-12,12],[y,-12,12])$
```

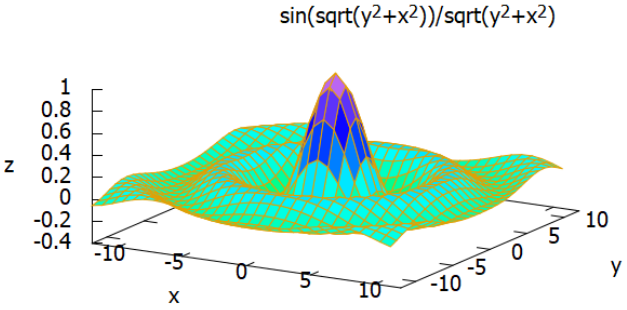


Рис. 2.12

```
(%i35)   plot3d([cos(y)*(10.0+6*cos(x)), sin(y)*(10.0+6*cos(x)),
-6*sin(x)], [x,0,2*%pi], [y,0,2*%pi], ['grid,40,40])$
```

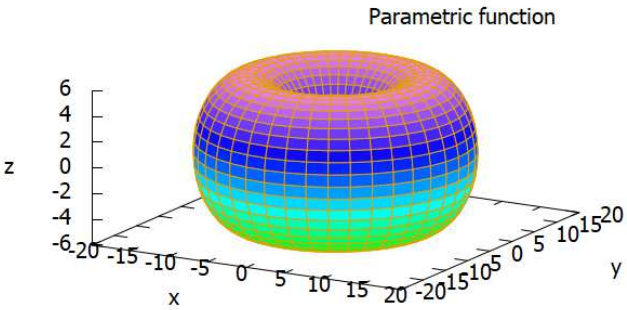


Рис. 2.13

6°. Пределы:

```
(%i37)   limit((5*x+1)/(3*x-1),x,inf);
(%o37)                    5/3
```

```
(%i36)   plot3d([5*cos(x)*(cos(x/2)*cos(y) +
sin(x/2)*sin(2*y) + 3.0)-10.0, -5*sin(x)*(cos(x/2)*cos(y) +
sin(x/2)*sin(2*y)+3.0), 5*(-sin(x/2)*cos(y) + cos(x/2)*sin(2*y))],
[x,-%pi,%pi],[y,-%pi,%pi],['grid,40,40])$
```

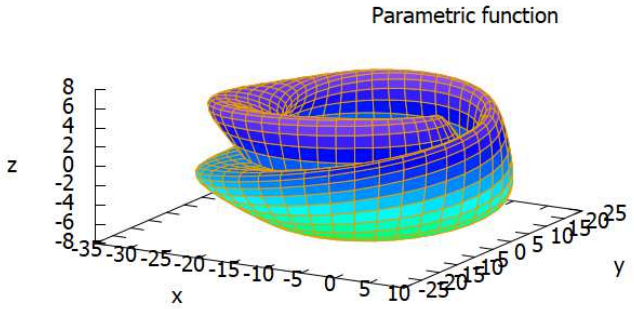


Рис. 2.14

7°. Дифференцирование и интегрирование:

```
(%i38)   diff(sin(x^2));
(%o38)       2 x cos(x^2) del(x)
```

```
(%i39)   'integrate(%e^sqrt(q*z),z,0,4);
(%o39)       integral_0^4 %e^sqrt(q z), z, 0, 4)
```

```
(%i40)   integrate(%e^sqrt(q*z),z,0,4);
(%o40)       2/q - (2 sqrt(q) %e^(2 sqrt(q)) - 4 q %e^(2 sqrt(q))) / q^(3/2)
```

```
(%i41)   integrate(sin(x),x);
(%o41)       - cos(x)
```

```
(%i42)   sum((1/2)^i,i,0,inf), simpsum;
(%o42)       2
```

8°. Обыкновенные дифференциальные уравнения:

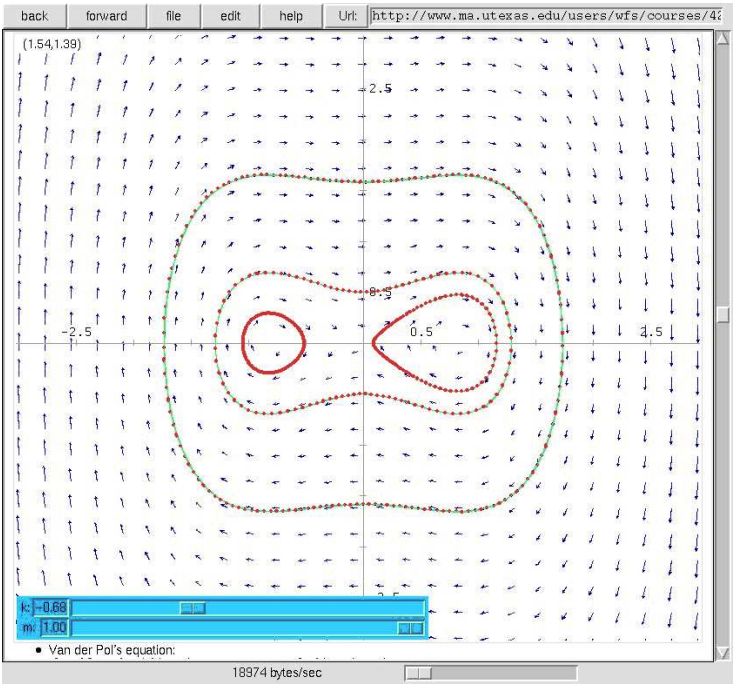


Рис. 2.15. Поле направлений для уравнения ван дер Поля

```
(%i43)  depends(y,x);
```

$$(\%o43) \qquad [y(x)]$$

```
(%i44)  diff(y,x)=(4-2*x)/(3*y^2-5);
```

$$(\%o44) \qquad \frac{d}{dx}\,y = \frac{4-2\,x}{3\,y^2-5}$$

```
(%i45)  ode2(%,y,x);
```

$$(\%o45) \qquad -\frac{y^3-5\,y}{2} = \frac{x^2-4\,x}{2} + \%c$$

9°. Работа с файлами:

```
(%i46)  load(file);
```

10°. Прекращение сеанса пакета Maxima:

```
(%i47)  quit();
```

2.8. Получение справочной информации

Как указывалось выше, Maxima имеет легкодоступную справку по системе, к которой можно обратиться как через главное меню, так и непосредственно из входной строки.

Справочник по Maxima можно просматривать в разных формах. При интерактивной работе при запросе руководство пользователя отображается как обычный текст командой `?` (то же самое, что функция `describe`). Это руководство можно просматривать в гипертекстовой форме через главное меню и через веб-страницы в сети интернет обычным браузером.

Команда	*Описание*
`? cmd`	Показать краткую справку
`?? cmd`	Поиск неточных совпадений по команде `cmd` на страницах справки
`apropos("cmd")`	Список всех имен Maxima, которые содержат подстроку `cmd`
`decribe("cmd")`	Аналог `? cmd`
`demo(filename)`	Отображение демо-файла
`example(cmd)`	Примеры работы функции

Страницы справки для всех команд и операторов доступны с использованием специальных символов `? cmd` и `?? cmd`. Двойной знак вопроса `??` может использоваться для поиска строки `cmd` в полном руководстве.

```
(%i1)   ? sqrt;
- - Function: sqrt (<x>)
     The square root of <x>. It is represented internally by
     '<x>^(1/2)'. See also 'rootscontract'.
     'radexpand' if 'true' will cause nth roots of factors of a product
     which are powers of n to be pulled outside of the radical, e.g.
     'sqrt(16*x^2)' will become '4*x' only if 'radexpand' is 'true'.
There are also some inexact matches for 'sqrt'.
Try '?? sqrt' to see them.
```

Важно не забывать вставлять пробел между `?` и `cmd`. В противном случае ввод интерпретируется как внутренняя переменная языка Lisp, которая возвращается как результат. Это также может привести к нераспознаваемому сообщению об ошибке от интерпретатора Lisp.

```
(%i2)   ?sqrt;
(%o2)       sqrt
```

Если вы помните только часть (подстроку) имени команды, то команда `apropos` будет весьма полезной. Она возвращает список всех тех имен Maxima, которые содержат эту подстроку.

```
(%i3)   apropos("sqrt");
(%o3)       [isqrt, sqrt, sqrtdispflag]
```

Для альтернативного способа доступа к справочнику в wxMaxima нужно нажать кнопку F1 или использовать кнопку Help в строке меню. Он предоставляет доступ ко всем возможностям библиотеки, включая индекс и функцию поиска.

Функция `example` отображает примеры для многих функций Maxima, например:

```
(%i4)   example(product);
        (%i4) product(x+(i*(i+1))/2,i,1,4)
(%o5)   (x + 1) (x + 3) (x + 6) (x + 10)
(%o5)   done
```

Функция `demo(filename)` выдерживает паузу после каждого выражения и продвигается далее после того, как пользователь нажимает клавишу Enter.

Для поиска необходимого демо-файла функция использует список `file_search_demo`.

2.9. Общие сведения о работе с системой

Maxima – это система компьютерной алгебры, которая получает некоторые входные данные, производит вычисления с операндами команды и возвращает результаты. Как во всех CAB (да и вообще во всех программных системах), необходимо знать синтаксические правила записи команд (операторов, выражений, фрагментов) на языке используемой системы.

Символ	*Описание*
`%i1`	Ввод (приглашение к вводу)
`%o1`	Вывод результатов (подсказка о выводе)
`;`	Маркер конца ввода
`$`	Маркер конца ввода. Вывод подавляется

После старта одной из оболочек системы (напоминаем, что далее используется ориентация на wxMaxima) можно вводить свои команды, которые должны заканчиваться точкой с запятой `;`. Заметим, что этот ввод

будет автоматически иметь префикс `%i1`. Данная строка затем вычисляется в соответствии с правилами, которые в основном основаны на сопоставлении с образцом (шаблоном). Затем результат выводится на экран, причем выводу предшествуют символы `%o1`. Их можно использовать для ссылки на результат в дальнейших расчетах.

Среда	*Запуск на вычисление*
wxMaxima	`Shift+Enter`
xMaxima	`Shift+Enter` или `Enter`
Maxima on Android	клавиша `ENTER` в правом нижнем углу экрана

Команда также может заканчиваться специальным символом `$` вместо точки с запятой. При этом вывод результата подавляется, т.е. Maxima вычисляет входное выражение, но не показывает результат его расчета. Это может быть удобно в случае, когда промежуточный результат очень длинный и занимает несколько страниц. Вот простой пример.

```
(%i1)   2+3;
(%o1)       5
```

```
(%i2)   4*5$
```

Также можно поместить более одной команды в одну строку, хотя команда может быть распределена по двум или более строкам (но должна обязательно завершаться с символом `;` или `$`).

```
(%i3)   2+3; 4*5;
(%o3)       5
(%o3)       20
```

```
(%i4)   2+3*(4+5)
        +6*7-8/4;
(%o4)       69
```

Ссылка	*Описание*
`%`	Результат последнего вычисления
`%o`i	Результат i-го расчета (содержание ячейки с результатом номер i)
`%th(i)`	Результат i-го предыдущего вычисления (содержание ячейки с результатом номер i)

Системная переменная	*По умолчанию*	*Описание*
`linenum`		Номер очередной пары ячеек ввода/вывода

Символ % отсылает к результирующему выражению, самому последнему, которое вычислила Maxima, независимо от того, отображалось оно или нет. Кроме того, результат i-го расчета доступен через ссылку %oi.

```
(%i5)   2+3;
(%o5)       5
```

```
(%i6)   % * 20;
(%o6)       100
```

```
(%i7)   % - 10;
(%o7)       90
```

```
(%i8)   %o5 - 10;
(%o8)       -5
```

Заметим, что % относится к результату последнего вычисления. Поскольку можно вводить команду практически везде в блокноте wxMaxima, то необязательно, что % ссылается на содержимое выходной ячейки, расположенной сразу выше текущей входной ячейки.

Оператор	*Описание*
`playback(...)`	Повторный вывод входных, выходных и промежуточных ячеек без их перевычисления
`print(expr_1,...,expr_n)`	Вывод выражений на экран один за одним с левого края

Функция	*Описание*
`load(pkg)`	Вычислить содержание файла `pkg` (загрузить пакет)

Как и многие другие системы, Maxima предоставляет базовый набор команд (ядро) и предлагает множество дополнительных пакетов для специальных задач. Это сводит к минимуму время запуска и потребление памяти и обеспечивает большую функциональность. Кроме того, таким образом можно расширить возможности системы. Такие пакеты либо автоматически загружаются по требованию (посредством списка команд, для которых пакеты должны быть загружены), либо должны загружаться явно. В последнем случае необходимо использовать `load` для соответствующих файлов.

```
(%i9)   load(vect);
(%o9)       D:\Programs\MAXIMA\maxima-5.42.2\share\maxima\5.42.2\share\vector\vect.mac
```

Заметим, что загруженные функции добавляются в список пользовательских функций, а затем могут отображаться и удаляться с помощью команд `functions` и `kill(all)` соответственно.

Команда	*Описание*
`/*...*/`	Комментарий (все символы между этими разделителями игнорируются)

При работе с системой Maxima может понадобиться использовать некоторое множество функций или определений сложных переменных множество раз. Код (запись на языке системы) для этих функций может быть собран в один или несколько текстовых файлов, которые можно затем подгрузить в сеансе работы Maxima с помощью команды `load`. При этом файл или файлы должны иметь расширение `.mac`.

Например, стартуем блокнот (notepad) операционной системы MS Windows и наберем строки (комментарии должны быть набраны латиницей):

```
/* Euclidean norm of vector [x,y] */
norm(x,y):=sqrt(x^2+y^2);
```

сохраним содержимое в файле с именем, например, `My_functions.mac`. Затем этот файл может быть загружен в процессе сеанса Maxima, и эта функция может использоваться как любая другая функция системы.

```
(%i10)  norm(3,4);
(%o10)      norm(3,4)
```

```
(%i11)  load("D:\\My_functions.mac");
(%o11)      D:\My_functions.mac
```

```
(%i12)  norm(3,4);
(%o12)      5
```

Строка `/*...*/` в приведенном выше примере игнорируется при загрузке файла. Обычно рекомендуется добавлять такие комментарии в свои файлы. Если файлы становятся длиннее или разрабатываемые функции усложняются, то это помогает сохранить ориентацию в коде.

Примечание. Нельзя забывать о маркерах конца ввода `;` или `$` в конце каждого фрагмента кода. Необходимо использовать строки (в примере используется `"D:\\My_functions.mac"`) в качестве аргумента для функции `load` и не забывать применять расширение файла `.mac`, если нужно загрузить свой собственный файл с кодом Maxima.

Символьные вычисления имеют тенденцию создавать много "мусора" (временные или промежуточные результаты, которые уже или в конечном итоге не используются). Эффективный сбор этого мусора может иметь решающее значение для успешного завершения некоторых программ.

2.10. Файл инициализации

`maxima-init.mac` – файл, который загружается автоматически при запуске среды Maxima. Этот файл можно использовать для настройки Maxima под конкретного пользователя. Файл `maxima-init.mac`, если он необходим, обычно помещается в каталог с именем `maxima_userdir`, хотя он может находиться в любом каталоге, который просматривается для поиска функцией `file_search`.

Ниже приведен пример файла `maxima-init.mac`:

```
(%i1)   setup_autoload("specfun.mac ultrasphericalassoc_legendre_p)$
```

В этом примере команда `setup_autoload` требует загрузить указанный файл (`specfun.mac`) в среду Maxima, если какая-либо из функций (`ultraspherical`, `assoc_legendre_p`) вызвана, но еще не определена. Таким образом, не нужно помнить о необходимости загрузки файла перед вызовом этих функций.

Файл `maxima-init.mac` может содержать любые другие назначения или операторы Maxima.

2.11. Интерпретатор, транслятор и компилятор

Как уже говорилось ранее, Maxima включает интепретатор команд, который имеет большую библиотеку предопределенных функций.

При просмотре ввода внешняя форма команд переводится во внутреннюю, а затем она распознается собственным языковым интерпретатором Maxima, который имеет несколько иную семантику, в частности символ или функция без значения дают результатом себя, а не ошибку. А после вычисления запускается упрощение. Но есть много других, более тонких, различий.

В состав системы включен также транслятор, который может переводить программы на входном языке системы Maxima в Lisp-программы и компилятор, который сначала переводит программу на Lisp и затем компилирует полученный Lisp-документ. При численных расчетах это может

привести к значительному увеличению скорости (особенно при правильных декларациях). Для символьных вычислений большая часть времени тратится внутри модуля, занимающегося упрощением выражений, и подобных ему, а поэтому ускорение будет меньше и часто даже незначительным.

Вводя новую функцию пользователя, необходимо учитывать, что можно серьезно ускорить ее выполнение, если ее оттранслировать или откомпилировать. Это следствие того, что при обычной интерактивной работе с этой функцией в среде системы, т.е. в ситуации, когда функция не оттранслирована или неоткомпилирована, при каждом очередном ее вызове Maxima каждый раз заново выполняет те действия, которые входят в определение функции, т.е. фактически делает синтаксический разбор и интерпретацию всех элементов выражения, представляющего суть функции.

Оператор	*Описание*
`compfile(filename, f_1, ..., f_n)`	Трансляция заданных функций пользователя и запись в файл
`compfile(filename, functions)`	Трансляция всех функций пользователя и запись в файл
`compfile(filename, all)`	То же самое
`comfile(f_1, ..., f_n)`	Конвертация заданных функций пользователя в Lisp и вызов Lisp-компилятора
`compfile(functions)`	Конвертация всех функций пользователя в Lisp и вызов компилятора Lisp
`compfile(all)`	То же самое
`compile_file(filename)`	Конвертация функций из файла и вызов Lisp-компилятора
`compile_file(filename, compiled_filename)`	Трансляция функций в файле, вызов Lisp-компилятора и загрузка в среду Maxima
`compile_file(filename, compiled_filename, lisp_filename)`	Трансляция функций в файле, вызов Lisp-компилятора, загрузка в среду Maxima и сохранение результата компиляции в файле

Функция `compfile` транслирует Maxima-коды функций в Lisp-коды этих функций и записывает последние в файл `filename`. Lisp-коды не вычисляются и не обрабатываются компилятором языка Lisp.

Функция `compile` транслирует Maxima-коды функций в Lisp-коды этих функций, переводит их в двоичные коды (вызывает Lisp-функцию `COMPILE` для каждой оттранслированной функции) и загружает скомпилированный код в вычислительную среду системы. Функция `compile` возвращает список имен скомпилированных функций. Компиляция функции

в Lisp-код может привести к значительному увеличению скорости и к значительному сокращению объема памяти.

Функция `compile_file` транслирует Maxima-коды функций в Lisp-коды этих функций, вызывает Lisp-компилятор и, если трансляция и компиляция были выполнены без появления серьезных ошибок, загружает скомпилированный код в вычислительную среду системы.

Рассматриваемая функция возвращает список имен четырех файлов: исходного файла системы Maxima, результат трансляции в Lisp, сообщения об ошибках трансляции и скомпилированный код. Если компиляция не удалась, то четвертым элементом будет `false`.

Некоторые объявления и определения вступают в силу, как только Lisp-код (без загрузки скомпилированного кода). К ним относятся функции, определенные с помощью оператора `:=`, макросы, заданные оператором `::=` и некоторые другие объекты (см. документацию по системе).

Присваивания и вызовы функций не выполняются, пока не будет загружен скомпилированный код. В частности, присваивания для параметров трансляции (`tr_numer` и др.) в самом файле с Maxima-кодом не влияют на трансляцию. Файл `filename` может не содержать инструкции языка Lisp.

Оператор	*Описание*
`translate(f_1, ..., f_n)`	Трансляция заданных функций пользователя и вычисление их Lisp-кода
`translate(functions)`	Трансляция всех функций пользователя и вычисление их Lisp-кода
`translate(all)`	То же самое
`translate_file (maxima_filename)`	Трансляция заданных функций и запись в файл
`translate_file (maxima_filename, lisp_filename)`	Трансляция всех функций пользователя и запись в файл

Функция `translate` транслирует Maxima-коды функций в Lisp-коды этих функций и вычисляет полученные Lisp-коды. Обычно трансляция кода функции в Lisp-код приводит к значительному увеличению скорости расчетов.

Для создания более эффективного кода, если это возможно, транслируемые функции должны в начале кода включать вызов `mode_declare`, например:

```
(%i1)   f(x_1,x_2,...):=block([v_1,v_2,...], mode_declare(v_1,
mode_1, v_2, mode_2, ...), ...)
```

где `x_1`, `x_2`, ... – параметры функции, а `v_1`, `v_2`, ... – локальные переменные.

Если переключатель `translate=true`, то код пользовательской функции транслируется в Lisp автоматически.

Необходимо иметь в виду, что оттранслированные функции могут работать не так, как до преобразования, поскольку между версиями Lisp и Maxima могут существовать определенные несовместимости. Как правило, функция `rat` с более чем одним аргументом и функция `ratvars` не должны использоваться, если какие-либо переменные являются определенными оператором `mode_declare` каноническими рациональными выражениями. Противоположная ситуация с настройкой `prederror`: если `prederror=false`, то трансляции не будет.

Результат, возвращаемый функцией `translate`, представляет собой список имен оттранслированных функций.

Функция `translate_file(maxima_filename, lisp_filename)` транслирует Maxima-коды функций из файла `maxima_filename` в файл `lisp_filename` с Lisp-кодами этих функций и возвращает список имен файлов: имя файла с Maxima-кодом, имя файла с Lisp-кодом и имя файла, содержащего дополнительную информацию о трансляции. Функция `translate_file` игнорирует расширение имени файла `lisp_filename`, если оно есть; расширение файла выходного Lisp-файла всегда LISP. Имя LISP-файла может включать в себя имя каталога или имена. В этом случае выходной LISP-файл записывается в указанный каталог.

Функция `translate_file(maxima_filename)` транслирует Maxima-коды функций из файла `maxima_filename` в файл с тем же именем, но расширением LISP с Lisp-кодами этих функций. Имя файла Maxima может включать имя или имена каталогов. В этом случае выходной Lisp-файл записывается в тот же каталог.

Опция	*По умолчанию*	*Описание*
`transcompile`	`true`	Если `true`, то функции `translate` и `translate_file` генерируют декларации для построения оттранслированного кода, более удобного для компиляции
`transrun`	`true`	Если `transrun=false`, то будут вызываться интерпретированные версия всех функций (при условии, что они все еще есть), а не оттранслированные

Функция `compfile` устанавливает `transcompile` в `true` на время выполнения.

Оператор	*Описание*
`declare_translated(f_1, ..., f_n)`	Указание имен оттранслированных функций

При переводе файла Maxima-кода в Lisp для транслятора важно знать, какие функции из тех, которые наблюдаются в файле с кодом, должны быть вызваны как оттранслированные или скомпилированные функции, а какие обычным образом как функции Maxima или вообще не определены. Помещая такое объявление в начало файла, транслятору сообщают, что, хотя символ пока и не имеет значения Lisp-функции, он будет иметь значение во время вызова.

Пусть имеется пользовательская функция, определяемая выражением

```
(%i2)   f(x) := 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$
```

Она транслируется командой

```
(%i3)   translate(f);
```

Рассмотрим пример, демонстрирующий выигрыш по времени после трансляции функции:

```
(%i4)   f(n):=block([sum, k], sum: 0, for k:1 thru n do (sum: sum +
k^2), sum)$
```

Функция `f(n)`, организованная в виде блока, позволяет вычислить сумму

$$\sum_{k=1}^{n} k^2.$$

Для выполнения тестов использовался ноутбук с ОС MS Windows, версия Maxima 5.43 (x86_64-w64-mingw32).

```
(%i5)   f(1500000);
(%o5)       1125001125000250000
```

```
(%i6)   time(%o5);
(%o6)       [5.917]
```

```
(%i7)   translate(f);
(%o7)       [f]
```

```
(%i8)   f(1500000);
(%o8)       1125001125000250000
```

```
(%i9)   time(%o8);
(%o9)       [0.797]
```

```
(%i10)   compile(f);
(%o10)       [f]
```

```
(%i11)   f(1500000);
(%o11)       1125001125000250000
```

```
(%i12)   time(%o11);
(%o12)       [0.938]
```

При непосредственном обращении к функции `f` время вычисления `f(1500000)` составило 5.917 с, после трансляции – 0.797 с, а после компиляции – 0.938 с. Увеличение времени расчета после компиляции, возможно, связано с фоновой работой операционной системы. Это объясняет и различие во временах расчета при нахождении в памяти компьютера активных программ.

Для оценки времени вычисления использована функция `time`. Функция `time(%o1,%o2,...)` возвращает список периодов времени в секундах, израсходованных для расчета результатов в ячейках `%o1`, `%o2`, ... Аргументом функции `time` могут быть только номера строк вывода, для любых других переменных функция возвращает значение `unknown`.

Следует иметь в виду, что как при трансляции, так и при компиляции Maxima старается оптимизировать функцию по скорости. Однако Maxima работает преимущественно с целыми числами произвольной длины либо текстовыми выражениями. Поэтому при работе с большими по объему функциями могут возникнуть проблемы, связанные с преобразованием типов данных.

2.12. Среда времени выполнения. Прерывания. Системные ошибки. Сообщения об ошибках

Функция	*Описание*
`room`	Вывод состояния памяти и стека
`status(feature)`	Информацию о наличии или отсутствии определенных системных свойств

Функция	*Описание*
`sstatus(keyword,item)`	Помещение свойства в список системных свойств
`system(command)`	Запуск `command` как отдельного процесса (не во всех ОС)
`time(%o1,%o2,%o3,...)`	Список времен (в секундах), необходимого для вычисления выходных строк `%o1`, `%o2`, `%o3`, ...
`timedate(T,tz_offset)`	Текущие дата и время с учетом зонального смещения
`parse_timedate(S)`	Сканирование строки, содержащей дату и время
`encode_time(year,month,day, hours,minutes,seconds, tz_offset)`	Число секунд после 00-00 01.01.1900 для текущей даты и времени
`decode_time(T,tz_offset)`	Обратная функция для `encode_time`
`absolute_real_time()`	Целое число секунд после 00-00 01.01.1900

Функция `room` в формате вызова `room()` выводит среднее описание, `room(true)` – подробное, `room(false)` – краткое.

Вызов функции `status(feature)` в форме `status(feature)` дает список характеристик системы. К ним относятся версия Lisp, тип операционной системы и т.д. Список может варьироваться от одного типа языка Lisp к другому.

Результатом вызова функции `status(feature)` в форме `status(feature,item)` будет `true`, если `item` находится в списке элементов, возвращаемых функцией `status(функция)`, и `false` в противном случае.

Переменная `features` содержит список функций, которые применяются к математическим выражениям.

Когда `keyword` является символом `feature`, то функция `sstatus(keyword,item)` помещает `item` в список системных свойств. После выполнения `sstatus(keyword,item)` функция `status(keyword,item)` возвращает значение `true`. Если `keyword` является символом `nofeature`, то `item` удаляется из списка системных свойств.

Время, предоставляемое функцией `time`, является оценкой монитора Maxima внутреннего чистого времени вычислений, а не истекшего времени. `time` может быть применено только к переменным из ячеек с результатами. Для любых других переменных функция `time` возвращает константу `unknown`.

Системная переменная	*По умолчанию*	*Описание*
`maxima_tempdir`	См. ниже	Рабочий каталог
`maxima_userdir`	См. ниже	Каталог пользователя

Переменная `maxima_userdir` указывает каталог, в котором Maxima ищет файлы в формате входных языков системы и Lisp. Maxima обращается и к некоторым другим каталогам, полные списки которых хранятся в переменных `file_search_maxima` и `file_search_lisp`.

Начальное значение переменной `maxima_tempdir` (`maxima_userdir`) – домашний каталог (подкаталог домашнего каталога) пользователя, если он есть. В противном случае система подыскивает подходящий.

Для изменения состояния переменной `maxima_tempdir` (`maxima_user dir`) может быть присвоена строка, содержащая имя каталога. Однако переназначение `maxima_userdir` не приводит к автоматическому изменению переменных `file_search_maxima` и `file_search_lisp`. Эти переменные должны быть изменены отдельно.

В среде wxMaxima прервать длинные вычисления можно, если щелкнуть курсором "мыши" по иконке ⊗. То же самое в среде xMaxima делается, если выбрать пункт меню Файл->Прервать или набрать на клавиатуре Ctrl+g, а в среде Terminal – Ctrl+c.

При интерпретации ввода и расчетах могут быть обнаружены ошибки. Краткий перечень сообщений о серьезных ошибках и их расшифровка приведены ниже.

Функция	*Описание*
`apply: no such "list" element`	Обращение к несуществующему списку
`Argument must be a non-atomic expression`	Аргумент должен быть не атомом
`Assignment: cannot assign to <function name>`	Нельзя присваивать <имени функции>
`expt: undefined: 0 to a negative exponent`	0 в отрицательной степени
`Incorrect syntax: , is not a prefix operator`	Неправильный синтаксис: "," в начале команды
`Incorrect syntax: Illegal use of delimiter)`	Неправильный синтаксис: неверный ограничитель ")"
`loadfile: failed to load <filename>`	Ошибка при загрузке файла с именем <filename>
`makelist: second argument must evaluate to a number`	Неверной вызов функции `makeliist`: второй аргумент должен быть числом
`Only symbols can be bound`	На месте символа другой объект

Функция	*Описание*
`Operators of arguments must all be the same`	Аргументами оператора должны быть однородные объекты
`Out of memory`	Нехватка памяти
`part: fell off the end`	Функция `part` обращается к несуществующему элементу списка
`Undefined variable (draw or plot)`	Неопределенная переменная при построении графиков
`VTK is not installed, which is required for Scene`	Не установлена VTK (библиотека 3D-графики)

А теперь возможные ошибки:

Функция	*Описание*
`Encountered undefined variable <x> in translation`	Обнаружена неопределенная переменная <x> в переводе
`Argument must be a non-atomic expression`	Аргумент должен быть не атомом
`Rat: replaced <x> by <y> = <z>`	Число с плавающей точкой заменяется рациональным

В систему Maxima включены функции и переменные для обнаружения ошибок и составления отчетов о них.

Функция	*Описание*
`run_testsuite([options])`	Запуск набора тестов
`bug_report()`	Номера версий Maxima и Lisp и ссылки на номера ошибок в отчете
`build_info()`	Резюме параметров сборки системы Maxima

Тесты, дающие желаемый ответ, считаются "пройден успешно", равно как и тесты, которые не дают желаемого ответа, но помечаются как известные ошибки.

Параметр функции `run_testsuite` может принимать следующие значения, определяемые ключевыми словами:

– `display_all` (показать все тесты. Обычно тесты не отображаются, если тест не пройден. По умолчанию `false`);

– `display_known_bugs` (отобразить тесты, помеченные как известные ошибки. По умолчанию `false`);

– `tests` (отдельный тест или список тестов, которые нужно запустить. Каждый тест может быть указан либо строкой, либо символом. По умолчанию все тесты запускаются. Полный набор тестов указывается с помощью `testsuite_files`);

– `time` (отображение информации о времени. Если `true`, то отображается время, необходимое для каждого тестового файла. Если `all`, то время отображается для каждого отдельного теста, если `display_all=true`. По умолчанию `false`);

– `share_tests` (загрузка дополнительных тестов для каталога `share`. Если `true`, то эти дополнительные тесты выполняются как часть набора тестов. Если `false`, то тесты из каталога `share` не выполняются. Если `only`, выполняются только тесты из каталога `share`. Конечно, фактический набор тестов, которые запускаются, может контролироваться ощией `tests`. По умолчанию `false`).

Функция `bug_report` выводит на консоль номера версий Maxima и Lisp и дает ссылку на веб-страницу отчета об ошибках проекта Maxima. Информация о версии такая же, как сообщается функцией `build_info`. Когда сообщается об ошибке, рекомендуется скопировать информацию о версиях Maxima и Lisp в отчет об ошибке.

Системная переменная	*Описание*
`testsuite_files`	Множество тестов, запускаемых функцией `run_testsuite`
`share_testsuite_files`	Множество тестов из каталога `share`, запускаемых функцией `run_testsuite`

Переменная `testsuite_files` содержит список имен файлов, содержащих тесты для запуска. Если известно, что некоторые из тестов в файле дают сбой, вместо перечисления имени файла используется список, содержащий имя файла и номера тестов, которые не пройдены.

В системе Maxima, как в любом достаточно большом программном проекте, имеются ошибки (bugs). Содружество системных программистов и пользователей системы регулярно фиксирует проблемные места и постепенно исправляет имеющиеся проблемы. Со списком замеченных ошибок можно ознакомиться на интернет-странице https://sourceforge.net/p/maxima/mailman/maxima-bugs/.

3. ДАННЫЕ И ОСНОВНЫЕ СРЕДСТВА РАБОТЫ С НИМИ

3.1. Выражения. Объекты и действия. Идентификаторы. Атомы. Комментарии

Во входном языке Maxima, как и в других системах программирования, имеется определенное число зарезервированных служебных слов, использование которых не по назначению приводит к появлению странных синтаксических ошибок при анализе входного текста интерпретатором. К таким словам относятся `and`, `do`, `elseif`, `integrate`, `product`, `then`, `unless` и многие другие, с полным списком которых можно ознакомиться в официальной документации по системе Maxima.

Большинство конструкций в Maxima – это *выражения*. Последовательность нескольких выражений можно превратить в новое выражение, записав выражения из последовательности через запятую или заключив их в скобки, что похоже на выражение с запятой.

```
(%i1)   a: 5$ (a: a-1, b: a^3);
(%o1)      64
```

```
(%i2)   (if (b > 17) then x: 2 else x: 4, 3*x);
(%o2)      6
```

Даже циклы во входном языке Maxima являются выражениями, хотя результатом их работы обычно является строка *done*.

```
(%i3)   y: (x: 1, for i from 1 thru 10 do (x: x+i^2))$ y;
(%o3)      done
```

Заметим, что если необходимо иметь результат работы последовательности нескольких выражений, то через запятую нужно включить в выражение еще один термин, который фактически и возвращает требуемое значение.

```
(%i4)   y: (x: 1, for i from 1 thru 10 do (x: x+i^2), x)$ y;
(%o4)      386
```

В пакете Maxima выражениями являются переменные, функции, математические и программные конструкции. Числа также считаются выражениями, но специального вида. Кроме того, в пакете имеются функции и команды, которые позволяют обрабатывать выражения различных типов. В частности, пакет Maxima позволяет решать алгебраические уравнения в символьной форме.

```
(%i5)  solve(a*x^2+b*x+c=0,x);
```

$$(\%o5)\qquad \left[x=-\frac{\sqrt{b^2-4\,a\,c}+b}{2\,a},\quad x=\frac{\sqrt{b^2-4\,a\,c}-b}{2\,a}\right]$$

Более того, коэффициенты и неизвестные в квадратных уравнениях могут быть выражениями.

```
(%i6)  solve((s-t)*f(y)^2+abs(s+t)*f(y)+(s+t)=0, f(y));
```

$$(\%o6)\qquad \left[f(y)=-\frac{|t+s|+\sqrt{5\,t^2+2\,s\,t-3\,t^2}}{2\,t-2\,s},\quad f(y)=-\frac{\sqrt{5\,t^2+2\,s\,t-3\,t^2}-|t+s|}{2\,t-2\,s}\right]$$

Интерпретатор входного языка Maxima различает лексемы, которые являются *объектами*, и лексемы, которые являются *действиями*. Действия и объекты – это конструкции, которые могут или не могут быть выполнены. По умолчанию имена функций являются действиями. Действие можно превратить в объект, заключив в апострофы имя функции или применив функцию `nounity`. Объект можно превратить в действие, применив функцию `verbify`. Флажок `nouns` заставляет функцию `ev` вычислить объект в выражении.

Действие распознается по лидирующему знаку доллара `$` в соответствующем символе языка Lisp, а объект – по знаку процента `%`. Некоторые объекты имеют специальные свойства отображения, такие как `'integrate` или `'derivative`, но у большинства объектов таких свойств нет. По умолчанию формы объекта и действия функции при отображении идентичны. Глобальный флажок `noundisp` заставляет систему отображать объекты с лидирующим апострофом `tquo`.

```
(%i7)  fun(t):= sin(t);
```

$$(\%o7)\qquad fun(t):=\sin(t)$$

```
(%i8)   a: fun(%pi/4)$ b: 'fun(%pi/4)$ c: 'fun(%pi/4), nouns$
[a,b,c];
(%o8)       [1/sqrt(2), fun(%pi/4), 1/sqrt(2)]
```

```
(%i9)   'integrate(x^3,x,0,2)=integrate(x^3,x,0,2);
(%o9)       integrate(x^3, x, 0, 2) = 4
```

Идентификаторы в системе Maxima могут содержать символы латиницы плюс цифры от 0 до 9 плюс любой специальный символ, которому предшествует символ \. О возможности использования кириллицы и символов греческого алфавита см. в подразделе 2.4. Цифра может быть первым символом идентификатора, если ему предшествует символ \. Этот же символ не обязателен перед последующими цифрами.

Некоторые символы могут быть объявлены алфавитными с помощью функции `declare`. Если это так, то в идентификаторе им необязательно должен предшествовать символ \. Изначально алфавитные символы – это от `A` до `Z`, от `a` до `z`, `%` и `_`. Как уже отмечалось, интерпретатор ввода Maxima чувствителен к регистру, на котором набираются лексемы. Так идентификаторы `loc`, `LOC` и `Loc` различны.

Идентификатор в языке Maxima – это также символ базового языка Lisp, который начинается со знака `$`. Любому другому символу языка Lisp предшествует знак `?`, если он появится в выражении Maxima.

```
(%i10)  declare("|",alphabetic)$ pi|squared: %pi^2$ pi|squared;
(%o10)      pi|squared
```

В системе Maxima нет разделения на объекты и данные: имена переменных и выражения могут использоваться практически в одном и том же контексте. По умолчанию, без дополнительных синтаксических ухищрений, символ, связанный с любым выражением, будет представлять это выражение; символ, не связанный ни с чем, будет представлять самого себя (трактуемого опять-таки как выражение). Из этого следует, в частности, то, что вместо любого символа в содержащее его выражение автоматически подставляется его значение только в том случае, если это значение было ему приписано до определения выражения.

Создать новое имя, ранее не использовавшееся в сеансе можно с помощью функции `gensym`.

```
(%i11)  gensym();
(%o11)      g73586
```

```
(%i12)   gensym("var");
(%o12)      gvar73587
```

```
(%i13)   gensym(12345);
(%o13)      gg12345
```

Функция `remvalue(name_1,...,name_n)` удаляет значения пользовательских переменных `name_1`, .., `name_n` (которые могут индексы) из системы, а вызов `remvalue(all)` – значения всех переменных из списка `values` и списка переменных, имена которым даны пользователем.

Атом – это самое простое неделимое выражение в языке системы Maxima. Атомами являются числа, строки, символы (), включающие идентификаторы переменных и имена функций.

Любой текст между символами `/*` и `*/` во входном потоке системы Maxima трактуется как *комментарий*.

Синтаксический анализатор программного кода обрабатывает комментарий как пробел с целью поиска лексем во входном потоке. Лексема всегда заканчивается комментарием. Вход вида `a/* foo */b`, содержит две лексемы `a` и `b` и ни одной лексемы `ab`. Комментарии всегда игнорируются: ни содержимое, ни расположение комментариев не сохраняются в проанализированных входных выражениях.

Комментарии могут быть вложены на произвольную глубину. Разделители `/*` и `*/` должны образовывать соответствующие пары.

```
(%i14)   /*Радиус*/ r: 5;
(%o14)      5
```

```
(%i15)   /*Площадь круга*/ s: %pi*r^2;
(%o15)      25 π
```

3.2. Типы числовых данных (числа целые, рациональные, действительные, комплексные). Встроенные постоянные. Характеристики точности представления данных

Maxima различает четыре типа чисел:

- целые числа: ..., –3, –2, –1, 0, 1 3, 3, ...;
- рациональные числа: отношение двух целых чисел;
- числа с плавающей точкой: состоят из мантиссы, представляющей примерно 16 десятичных цифр, и показателя степени по основанию 10, например, $1.234567890123456 \cdot 10^5$. В обычной записи они называются десятичными цифрами и обычно пишутся без показателя степени: для нашего примера это 123456.7890123456;

• числа с плавающей запятой произвольной точности, называемые длинными числами с плавающей точкой: числа с плавающей точкой, у которых размер мантиссы может быть установлен равным некоторому фиксированному, но произвольному положительному числу, используя системную переменную `fpprec`.

Команда	*Описание*
`103`	целое 103
`103.`	целое 103
`835.0`	число с плавающей точкой 835
`835e0`	число с плавающей точкой 835
`835b0`	длинное число с плавающей точкой 835

Число с плавающей точкой может быть введено либо как десятичное число, по крайней мере, с одной цифрой после десятичной точки, например `123.0`, либо с использованием научной нотации, например `123e0`.

Целые и рациональные числа хранятся без потери точности, в то время как для чисел с плавающей точкой это невозможно. Их можно рассматривать как приближение к действительным числам. Заметим, что десятичное представление действительных чисел (как элементов множества $\mathbb{R}$) может иметь бесконечное число цифр, например, как в $1/7 = 0{,}1428571428571428\ldots$ или $\sqrt{2} = 1{,}414213562373095\ldots$. При записи таких чисел в память компьютера в виде чисел с плавающей точкой может быть сохранено только ограниченное количество цифр, а следовательно, они теряют точность. Сложение и вычитание чисел с плавающей точкой могут привести к дальнейшей потере точности из-за ошибок округления. Следующий пример демонстрирует разницу между рациональными числами и числами с плавающей точкой.

```
(%i1)   2/10 * 11 - 2 - 4/20;
(%o1)      0
```

```
(%i2)   2.0/10.0 * 11.0 - 2.0 - 4.0/20.0;
(%o2)      1.665334536937735 10^-16
```

Напомним, что в отличие от представления числовых данных в десятичной форме современный компьютер использует двоичное представление чисел с помощью цифр 0 и 1, называемых битами. Как следствие, числа типа 0.2 не могут быть точно представлены из-за бесконечного двоичного представления.

Maxima старается выполнять все расчеты максимально точно. Как следствие, Maxima сокращает рациональные числа или упрощает числовые выражения, где это возможно, но не преобразует результаты в числа

с плавающей точкой, если это не требуется. В частности, Maxima может возвращать специальные константы в качестве результатов вычислений.

Примечание. Команды и операторы, используемые в следующих примерах, определены в подразделе 3.6.

```
(%i3)  19/3;
(%o3)  19/3
```

```
(%i4)  2^1000;
(%o4)  1071508607186267320948425049060[242digits]429831652624386837205668069376
```

```
(%i5)  sqrt(2);
(%o5)  sqrt(2)
```

```
(%i6)  14/6;
(%o6)  14/6
```

```
(%i7)  sqrt(12);
(%o7)  3*sqrt(2)
```

```
(%i8)  sqrt(8);
(%o8)  2^(3/2)
```

```
(%i9)  exp(3);
(%o9)  %e^3
```

```
(%i10)  atan(1);
(%o10)  %pi/4
```

```
(%i11)  tan(%pi/4);
(%o11)  1
```

Если вместо точных расчетов используются числа с плавающей запятой, результаты получаются менее точными, т.к. сохраняются в форме чисел с плавающей точкой. Как показано в первой строке следующего примера, часто достаточно иметь только одно число с плавающей точкой, чтобы получить результат в такой же форме. Но как показывает последняя строка, ответ иногда может оказаться странным.

```
(%i12)  19.0/3;
(%o12)  6.333333333333333
```

```
(%i13)  2.0^1000;
(%o13)      1.07150860718627 10^3 01
```

```
(%i14)  sqrt(2.0);
(%o14)      1.414213562373095
```

```
(%i15)  sqrt(12.0);
(%o15)      3.464101615137754
```

```
(%i16)  exp(3.0);
(%o16)      20.08553692318767
```

```
(%i17)  atan(1.0);
(%o17)      .7853981633974483
```

```
(%i18)  tan(%pi/4.0);
(%o18)      tan(0.25 * %pi)
```

Еще раз подчеркнем, что нужно помнить о разнице между целыми числами и числами с плавающей точкой. Входные данные в следующих двух строках кода различны и поэтому обрабатываются по-разному!

```
(%i19)  123;
(%o19)      123
```

```
(%i20)  123.0;
(%o20)      123.0
```

К специальным (встроенным) константам относятся следующие:

Константа	*Определение*
%e	e – основание натуральных алгоритмов ($\approx$ 2,71828...)
%i	$\boldsymbol{i}$ – мнимая единица $\left(\sqrt{-1}\right)$
%pi	π ($\approx$ 3,14159...)

Константа	*Определение*
%phi	"Золотое сечение" $((1+\sqrt{5})/2)$
inf	$+\infty$
minf	$-\infty$
infinity	Комплексная бесконечность
ind	Представление ограниченного неопределенного результата
und	Представление неопределенного результата
zeroa	Представление положительной бесконечно малой величины
zerob	Представление отрицательной бесконечно малой величины
false	Логическая постоянная "ложь"
true	Логическая постоянная "истина"

В системе Maxima эти встроенные константы используются как некие символы, причем система проводит операции с ними с учетом использования точных соотношений.

```
(%i21)   limit(sin(1/x), x, 0);
(%o21)      nind
```

```
(%i22)   limit(x*sin(x), x, 0);
(%o22)      nund
```

```
(%i23)   limit(x+zeroa);
(%o23)      x
```

CAB Maxima имеет набор системных переменных и использует их для управления поведением системы. Например, ниже переменная `fpprintprec` используется для управления печатью чисел с плавающей точкой, а переменная `numer` контролирует, идут ли расчеты с математическими функциями с плавающей запятой или нет.

Если необходимо производить вычисления только над числами с плавающей точкой, лучше установить данный режим при помощи команды `numer: true`. Результат этой команды можно отменить командой `numer: false`.

Переменная	По умолчанию	*Определение*
`fpprec`	16	Количество значащих цифр для арифметики длинных чисел с плавающей точкой
`fpprintprec`	0	Количество цифр при печати с плавающей точкой (по умолчанию: все значащие цифры)
`ratepsilon`	10^{-15}	Точность, используемая при преобразовании чисел с плавающей запятой в рациональные числа, когда параметр `bftorat` имеет значение `false`

Иногда некоторое количество цифр кажутся лишними, когда печатаются 16 цифр для чисел с плавающей точкой. Необходимое количество знаков в результате можно контролировать путем установки системной переменной `fpprintprec`.

```
(%i24)   fpprintprec: 4$
```

```
(%i25)   sqrt(2.0);
(%o25)      1.141
```

```
(%i26)   fpprintprec: 0$
```

```
(%i27)   sqrt(2.0);
(%o27)      1.414213562373095
```

CAB Maxima выполняет многие свои вычисления над полем комплексных чисел. Комплексное число задается путем добавления к действительной части выражения мнимой части, умноженной на константу `%i`. Например, корни уравнения $x^2 - 4\,x + 13 = 0$ будут записаны как `2 + 3*%i` и `2 - 3*%i`. Заметим, что упрощение произведений сложных выражений может быть достигнуто путем раскрытия скобок. Так упрощение дробей, корней и других функций от комплексных аргументов непростых конструкций обычно можно выполнить с помощью функций из подраздела 8.2.

3.3. Логические константы, переменные и функции. Предикаты. Флажки. Равенства

Логические константы	*Описание*
`true`	Логическая константа "истина"
`false`	Логическая константа "ложь"
`unknown`	Оператор с неизвестным логическим значением

Переменные, имеющие на некотором этапе сессии одно из значений – `true`, `false` или `unknown`, являются логическими или булевскими.

Логические операторы предназначены для сравнения двух величин.

Операторы `and`,`or` и `not` вызывают вычисление своих операндов, т.е. действуют аналогично вызову `is` (см. далее).

Логические операторы	*Описание*
`<`	Меньше, чем
`<=`	Меньше или равно, чем
`>`	Больше, чем
`>=`	Больше или равно, чем
`and`	Логическое И
`or`	Логическое ИЛИ
`not`	Логическое отрицание

```
(%i1)   3>4;
(%o1)      3 > 4
```

```
(%i2)   not(3>4);
(%o2)       true
```

```
(%i3)   3>4 or 7<8;
(%o3)       true
```

Дополнительные логические функции и операторы (`%and`, `%if`, `%or`, `%union`) определены в экспериментальных пакетах `to_poly` и `to_poly_solve`. Эти же пакеты включают некоторые предикаты и функции, расширяющие возможности системы Maxima и предназначенные для анализа данных, упрощений выражений и др.

Предикат – выражение, вычисляющее одну величину или более с результатом `true` или `false`. В системе Maxima имеется значительное число предикатов, проверяющих наличие различных свойств у объектов. Для удобства распознавания основная часть предикатов имеет имена, оканчивающиеся буквой `p`.

Оператор	*Описание*
`charfun(p)`	0, если предикат `p` дает `false`, 1 , если `p` дает `true`, иначе `charfun(p)`
`compare(x,y)`	Оператор сравнения, который для `x` И `y` дает `true`
`is(x)`	Оператор проверки, является ли значение предиката `x` истинным или ложным
`equal(x,y)`	Эквивалентность
`notequal(a,b)`	Неэквивалентность
`unknown(expr)`	Предикат: результат `true`, если и только если `expr` содержит оператор или функцию, не распознаваемые блоком упрощения

Оператор `is` пытается определить логическое значение предиката. Если это значение существует, то оператор `is` возвращает соответствующее значение `true` или `false`. В противном случае возвращается `unknown`.

```
(%i4)   charfun(z^2 >= 0);
(%o4)       1
```

```
(%i5)   compare(z,abs(z));
(%o5)       ⩽
```

```
(%i6)   is(3>4);
(%o6)       false
```

```
(%i7)   is(x>0);
(%o7)       unknown
```

```
(%i8)   [equal(x^2-1, (x+1)*(x-1)), is(equal(x^2-1, (x+1)*(x-1))),
notequal(x^2-1, (x+1)*(x-1)), is(notequal(x^2-1, (x+1)*(x-1)))];
(%o8)       [equal(x^2 - 1, (x - 1) (x + 1)), true, notequal(x^2 - 1, (x - 1) (x +
1)), false
```

Оператор	*Описание*
`=`	Оператор, задающий равенство
`#`	Отрицание оператора, задающего равенство
`lhs(eqn)`	Левая часть равенства
`rhs(eqn)`	Правая часть равенства

В системе Maxima широко используются отношения *равенства*, т.е. разновидности отношений эквивалентности.

Операторы = и # тестируют на синтаксическое равенство, в отличие от оператора эквивалентности `equal`. Выражения могут быть эквивалентными и не синтаксически равными.

```
(%i9)   a: (x+1)*(x-1);
(%o9)       (x - 1) (x + 1)
```

```
(%i10)   b: x^2-1;
(%o10)       x^2 - 1
```

```
(%i11)   is(a=b);
(%o11)       false
```

```
(%i12)   is(a#b);
(%o12)       true
```

```
(%i13)   is(equal(a,b));
(%o13)       true
```

Если в равенстве присутствуют объекты (как правило, атомы), требующие определения на основе этого равенства, то такое равенство трактуется как *уравнение*. В системе Maxima есть средства для решения линейных и нелинейных алгебраических, некоторых классов трансцендентных, обыкновенных дифференциальных, разностных и интегральных уравнений.

3.4. Контексты, факты и база данных (знаний)

Механизм контекста позволяет пользователю связывать воедино и именовать набор фактов, называемый контекстом. Как только это будет сделано, пользователь может заставить вычислительное ядро Maxima использовать или забывать большое количество фактов, просто активируя или деактивируя их контекст.

Функция	*Описание*
`activate(context_1,...,context_n)`	Активация контекстов
`deactivate(context_1,...,context_n)`	Деактивация контекстов
`killcontext(context_1,...,context_n)`	Удаление контекстов
`newcontext(name)`	Создание нового контекста
`supcontext(name,context)`	Создание нового подконтекста

Любой символьный атом может быть контекстом, а факты, содержащиеся в этом контексте, будут храниться в хранилище до тех пор, пока не будут уничтожены один за другим с помощью вызова функции `forget` или целиком с помощью вызова функции `kill` для уничтожения контекста, которому они принадлежат.

Контексты существуют в иерархии, причем корнем всегда является контекст `global`, который содержит информацию о системе Maxima, которая нужна некоторым функциям. Находясь в данном контексте, все факты в этом контексте являются "активными" (имеется в виду, что они используются в выводах и поисках), как и все факты в любом контексте, который является подконтекстом активного контекста.

Когда запускается новая сессия Maxima, пользователь находится в контексте, называемом `initial`, который имеет контекст `global` как подконтекст.

Функция	*Описание*
`assume(pred)`	Добавляет предикат `pred` в "базу фактов"
`declare(x,prop)`	Назначает свойство `prop` символу `x`
`facts()`	Предоставляет список предположений (фактов)
`forget(pred)`	Удаляет предикат `prop`
`askinteger(expr,integer)`	Запрос к базе данных `assume`
`askinteger(expr)`	
`askinteger(expr,even)`	
`askinteger(expr,odd)`	
`asksign(expr)`	Знак выражения

Система Maxima позволяет работать с символами, что весьма полезно в случае, когда нас интересуют общие решения задач. В некоторых

случаях у нас есть некоторые знания о переменных и параметрах используемых моделей. Например, в экономике аргументы функции полезности предполагаются неотрицательными действительными числами, а параметр производственной функции Кобба–Дугласа для обеспечения вогнутости должен удовлетворять некоторым неравенствам.

Обычно при выполнении символьных вычислений Maxima всегда рассматривает общий случай, а следовательно, нередко решения оказываются неопределенными (например, если их бесконечно много в области комплексных чисел). Более того, для некоторых расчетов (например, когда необходимо вычислить интеграл $\int x^a\,dx$) Maxima спрашивает о дополнительных свойствах данных символов (например, является ли $a-1$ нулем или нет в этом интеграле), т.к. без такой информации Maxima не может дать замкнутую форму решения. В таких случаях пользователь может сделать предположения о конкретных символах и системе не понадобится запрашивать дополнительные соотношения.

Допущения могут быть добавлены в форме логических выражений с использованием `assume`. Когда Maxima вычисляет логические выражения, то она учитывает предоставленные нами предположения. Команда `facts` предоставляет весь список наших предположений. Мы можем использовать команду `forget`, чтобы удалить ненужные далее предположения.

```
(%i1)  is(x>=0);
(%o1)      unknown
```

```
(%i2)  is(U(x)>=0);
(%o2)      unknown
```

```
(%i3)  assume(x>0,U(x)>0);
(%o3)      [x > 0, U(x) > 0]
```

```
(%i4)  is(x>=0 and U(x)>=0);
(%o4)      true
```

```
(%i5)  facts();
(%o5)      [x > 0, U(x) > 0]
```

Можно добавлять дополнительные предположения, а также удалять некоторые из них, например, утратившие актуальность.

```
(%i6)  assume(y>0);
(%o6)      [y > 0]
```

```
(%i7)  facts();
```
(%o7) $[x > 0,\ U(x) > 0,\ y > 0]$

```
(%i8)  forget(U(x)>0);
```
(%o8) $[U(x) > 0]$

```
(%i9)  facts();
```
(%o9) $[x > 0, y > 0]$

Если нужно зафиксировать равенство между величинами, то необходимо использовать оператор `equal`, а не оператор равенства `=`.

```
(%i10)  assume(a=1);
assume: argument cannot be an '=' expression; found a=1
assume: maybe you want 'equal'.
-- an error. To debug this try: debugmode(true);
```

```
(%i11)  assume(equal(a,1));
```
(%o11) $[equal(a, 1)]$

К сожалению, Maxima не умеет в полной мере делать выводы о логической истинности высказывания.

```
(%i12)  assume(x+y<1);
```
(%o12) $[-y - x + 1 > 0]$

```
(%i13)  facts();
```
(%o13) $[x > 0, y > 0, -y - x + 1 > 0]$

```
(%i14)  is(x*y>0);
```
(%o14) $true$

```
(%i15)  is(x<1);
```
(%o15) $unknown$

Тем не менее Maxima обнаруживает избыточные и противоречивые предикаты (при условии, что вычисление `is(pred)` дает `false`).

```
(%i16)  facts();
```
(%o16) $[x > 0, y > 0, y - x + 1 > 0]$

```
(%i17)  assume(x>0);
```
(%o17) $[redundant]$

```
(%i18)   assume(x<0);
(%o18)       [inconsistent]
```

```
(%i19)   facts();
(%o19)       [x > 0, y > 0, -y - x + 1 > 0]
```

В дополнение к возможности предположения в форме `facts` Maxima позволяет назначать свойства символам. Например, мы можем объявить символ n целым числом.

```
(%i20)   declare(n,integer);
(%o20)       done
```

```
(%i21)   facts();
(%o21)       [x > 0, y > 0, -y - x + 1 > 0, kind(n, integer)]
```

3.5. Строки, списки, множества, массивы и структуры

Строки

Строки (последовательности символов в кавычках) заключаются в двойные кавычки " и отображаются с кавычками или без них, в зависимости от глобальной переменной `stringdisp`. Строки могут содержать любые символы, включая символы табуляции, перехода на новую строку и возврата каретки.

```
(%i1)   s1: "This is a String.";
(%o1)       This is a String
```

Последовательность \" внутри строки распознается как символ двойной кавычки, а \\ – как обратная косая черта (). В системе Maxima нет типа данных "символ"; один символ представляется односимвольной строкой.

Функция	*Определение*
`"a string"`	Символьная строка
`concat(arg_1,arg_2,...)`	Сцепление аргументов-атомов
`sconcat(arg_1,arg_2,...)`	Сцепление строк

Функция `concat` сцепляет свои аргументы. Аргументы должны вычисляться до атомов. Возвращаемое значение является символом, если первый аргумент является символом, и строкой в противном случае.

```
(%i2)   a: 5$ b: 678$ concat(a,b/2);
(%o2)       5339
```

```
(%i3)   concat('a,b/2);
(%o3)       a339
```

Функция `sconcat` сцепляет свои аргументы, получая строку. Аргументы могут быть не атомами.

```
(%i4)   sconcat("xx[",3,"]:", expand((x+y)^3));
(%o4)       "xx[3]:y^3+3*x*y^2+3*x^2*y+x^3"
```

Пакет `stringproc` содержит дополнительные функции для работы со строками и символами. Он может быть загружен с помощью функции `load` или автоматически при вызове какой-либо функции из пакета.

Функция	*Определение*
`alphacharp(char)`	Предикат: если `true`, то `char` – алфавитный символ
`alphanumericp(char)`	Предикат: если `true`, то `char` – алфавитно-цифровой символ
`ascii(int)`	Получение символа кодом `int`
`cequal(char_1,char_2)`	Предикат: если `true`, то символы одинаковые
`cgreaterp(char_1,char_2)`	Предикат: если `true`, то код символа `char_1` > кода символа `char_2`
`clessp(char_1,char_2)`	Предикат: если `true`, то код символа `char_1` < кода символа `char_2`
`charp(obj)`	Предикат: если `true`, то `obj` – символ Maxima
`cint(char)`	Код Unicode-символа
`constituent(char)`	Предикат: если `true`, то `char` – печатный символ
`digitcharp(char)`	Предикат: если `true`, то `char` – число
`lowercasep(char)`	Предикат: если `true`, то `char` на нижнем регистре
`uppercasep(char)`	Предикат: если `true`, то `char` на нижнем регистре
`space`	Пробел
`tab`	Символ табуляции
`unicode(arg)`	Unicode-символ по коду
`charat(str,n)`	`n`-й символ в строке
`charlist(str)`	Список символов строки
`eval_string(str)`	Сканирование и вычисление лексем
`parse_string(str)`	Сканирование без вычисления
`scopy(str)`	Копия строки

Функция	*Определение*
`sdowncase(str)`	Перевод символов на нижний регистр
`sdowncase(str,start)`	
`sdowncase(str,start,end)`	
`sequal(str_1,str_2)`	Предикат: если `true`, то строки одинаковы
`simplode(list)`	Объединение символов в строку
`sinsert(seq,str,pos)`	Вставка подстроки в строку
`slength(str)`	Длина строки
`smake(num,char)`	Строка из одинаковых символов
`smismatch(str_1,str_2)`	Результат – номер символа, с которого строки не равны
`split(str)`	Разбиение на лексемы
`sposition(char,str)`	Номер позиции символа строки
`sremove(seq,str)`	Удаление заданной подстроки из строки
`ssubst(new,old,str`	Замена подстроки
`ssubstfirst(new,old,str)`	Замена первого вхождения
`stringp(obj)`	Предикат: если `true`, то `char` – строка
`substring(str,start)`	Выделение подстроки
`substring(str,start,end)`	Выделение подстроки
`supcase(str)`	Перевод на верхний регистр
`tokens(str)`	Список лексем из строки

Списки

Списки являются основными "строительными" блоками для языков Maxima и Lisp. Все типы данных, кроме массивов и чисел, представляются в виде списков Lisp. Lisp предоставляет эффективные средства копирования и обработки списков.

Списки предоставляют эффективные способы добавления и удаления элементов. На момент создания списка необязательно знать его финальный размер. Вложенные списки могут быть непрямоугольными. Недостатком использования списков по сравнению с объявленными массивами является то, что время, необходимое для доступа к случайному элементу в списке, приблизительно пропорционально расстоянию между элементом и началом списка.

Функция	*Определение*
`[...]`	Признак списка или ограничители индексов элемента списка
`emptyp(L)`	Предикат: результат `true`, если `s` – пустой список
`length(L)`	Длина списка `L`
`append(...)`	Сцепление заданных списков
`cons(expr,list)`	Вставка `expr` в начало списка
`endcons(expr,list)`	Вставка `expr` в конец списка
`delete(x,L)`	Удаление элемента `x` из списка `L`

Функция	*Определение*
`sublist(L,P)`	Выделение подсписка согласно предикату
`sublist_indices(L,P)`	Получение индексов элементов выделенного согласно предикату подсписка
`flatten(list)`	Объединение всех подсписков списка `list`

Списки используются для объединения нескольких элементов в один объект. Они отображаются заключением элементов списка в квадратные скобки. Простой способ создать список – просто перечислить элементы списка, разделяя элементы запятыми, и заключить их в квадратные скобки. Элементами списков могут быть другие списки. Список может содержать один элемент и даже ни одного. В последнем случае это пустой список.

```
(%i5)   L: [1,4,9,16,25,[36,49]];
(%o5)       [1, 4, 9, 16, 25, [36, 49]]
```

```
(%i6)   emptylist: [];
(%o6)       [ ]
```

Доступ к элементу списка можно получить по его индексу, заключенному в квадратные скобки. Первый элемент имеет индекс 1, а самый большой индекс не должен превышать длину списка.

```
(%i7)   L[4];
(%o7)       16
```

```
(%i8)   length(L);
(%o8)       6
```

```
(%i9)   L[3]: -99;
(%o9)       −99
```

```
(%i10)   L;
(%o10)       [1, 4, −99, 16, 25, [36, 49]]
```

```
(%i11)   map(sin,[0,%pi/4,%pi/2,3*%pi/4]);
(%o11)       0, 1/√2, 1, 1/√2
```

```
(%i12)   map("=",[c,d],[5,3]);
(%o12)       [c = 5, d = 3]
```

```
(%i13)  sublist([1,2,3,4,5,6], evenp);
(%o13)      [2, 4, 6]
```

Существует несколько команд для управления списками. Сцепление списков (добавление элементов одного списка в конец другого) и удаление элементов из списка являются наиболее важными функциями.

```
(%i14)  append(L,[1,2,3]);
(%o14)      [1, 4, −99, 16, 25, [36, 49], 1, 2, 3]
```

```
(%i15)  delete(1,L);
(%o15)      [4, −99, 16, 25, [36, 49]]
```

```
(%i16)  kill(a,b)$ flatten([a,b,[c,[d,e],f],[[g,h]],i,j]);
(%o16)      [a, b, c, d, e, f, g, h, i, j]
```

```
(%i17)  cons(a,f(b,c,d));
(%o17)      f(a, b, c, d)
```

```
(%i18)  sublist_indices('[a,b,b,c,1,2,b,3,b], symbolp);
(%o18)      [1, 2, 3, 4, 7, 9]
```

Заметим, что эти команды возвращают новый список и не изменяют существующий. Для последней цели нужно присвоить новый список первоначальному.

```
(%i19)  L: delete(4,L);
(%o19)      [1, −99, 16, 25, [36, 49]]
```

```
(%i20)  L;
(%o20)      [1, −99, 16, 25, [36, 49]]
```

Функция	*Определение*
`makelist(expr,i,i1,i2)`	Создание списка в цикле
`makelist(expr,x,list)`	Создание списка по элементам другого списка
`map(fct,list)`	Применение функции `fct` одного аргумента к элементам списка
`map(fct,list1,list2)`	Применение функции `fct` двух аргументов к элементам списков
`create_list(form,x_1, list_1,...,x_n,list_n)`	Создание списка перебором элементов из двух списков

Списки можно формировать программно:

```
(%i21)  makelist(x^3,x,0,2*%pi,2);
(%o21)      [0, 8, 64, 216]
```

```
(%i22)  makelist(concat(x,i),i,6);
(%o22)      [x1, x2, x3, x4, x5, x6]
```

```
(%i23)  makelist(z^j, j, [1,3,5,7,11,13,17]);
(%o23)      [z, z^3, z^5, z^7, z^11, z^13, z^17]
```

```
(%i24)  create_list([i,j], i, [a,b], j, [e,f,h]);
(%o24)      [[a, e], [a, f], [a, h], [b, e], [b, f], [b, h]]
```

Функция	*Описание*
`first(L)`	Первый элемент списка `L`
`second(L)`	Второй элемент списка `L`
`third(L)`	Третий элемент списка `L`
`fourth(L)`	Четвертый элемент списка `L`
`fifth(L)`	Пятый элемент списка `L`
`sixth(L)`	Шестой элемент списка `L`
`seventh(L)`	Седьмой элемент списка `L`
`eighth(L)`	Восьмой элемент списка `L`
`ninth(L)`	Девятый элемент списка `L`
`tenth(L)`	Десятый элемент списка `L`
`rest(L,n)`	Список `L`, в котором удалены первые n элементов
`last(L)`	Последний элемент списка `L`

```
(%i25)  first(f(x,y,z));
(%o25)      x
```

```
(%i26)  rest(f(x,y,z));
(%o26)      f(y, z)
```

```
(%i27)  last(f(x,y,z));
(%o27)      z
```

Функция	*Описание*
`unique(L)`	Выделение уникальных элементов списка
`firstn(L,count)`	Первые `count` элементов списка `L`
`lastn(L,count)`	Последние `count` элементов списка `L`
`join(L1,L2)`	Объединение соответствующих пар элементов списков `L1` и `L2`

Функция	*Описание*
`lmax(L)`	Максимум из элементов списка
`lmix(L)`	Минимум из элементов списка
`listp(expr)`	Предикат: результат `true`, если `expr` – список, иначе `false`
`member(L1,L2)`	Предикат: результат `true`, если `L1` – элемент списка `L2`
`emptyp(L)`	Предикат: результат `true`, если `L` – пустой список
`every(P,L1,L2,...)`	Предикат: результат `true`, если истинен результат применения предиката `P` ко всем элементам списков `L1`, `L2`, ...
`some(P,L1,L2,...)`	Предикат: результат `true`, если истинен результат применения предиката `P` хотя бы к одному из элементов списков `L1`, `L2`, ...
`assoc(key,list)`	Поиск в списке
`copylist(list)`	Копия списка
`sort(L,P)`	Сортировка списка `L`согласно предикату `P`

```
(%i28)   every("=", [a, b, c], [a, b, c]);
(%o28)      true
```

```
(%i29)   some("#", [a, b, c], [a, b, c]);
(%o29)      false
```

```
(%i30)   assoc(y, [[x,1], [y,2],[z,3]]);
(%o30)      2
```

```
(%i31)   copylist([1,2,3]);
(%o31)      [1, 2, 3]
```

Множества

Система Maxima включает операции над конечными множествами, такие как пересечение и объединение, которые определены явным перечислением. Maxima рассматривает списки и множества как различные объекты. Эта особенность позволяет работать с множествами, элементами которых являются списки или множества.

Функция	*Описание*
`set(a_1,...,a_n)`	Формирование множества из элементов `a_1`, ..., `a_n`
`{a_1,...,a_n}`	То же самое
`makeset(expr,x,s)`	Формирование множества из элементов на основе применения выражения `expr` к элементам множества `s`, причем аргументы выражения перечислены в списке `x`
`set()`	Пустое множество
`{}`	То же самое

Функция	*Описание*
`setp(s)`	Предикат: результат `true`, если `s` – множество
`emptyp(s)`	Предикат: результат `true`, если `s` – пустое множество
`setify(list)`	Формирование множества из элементов списка
`listify(s)`	Формирование списка из элементов множества
`union(s_1,s_2,...)`	Объединение множеств
`intersection(s_1,s_2,...)`	Пересечение множеств (можно `intersect`)
`setdifference(s_1,s_2)`	Разность множеств
`symmdifference(s_1,...,s_n)`	Симметричная разность множеств
`setequalp(s_1,s_2)`	Предикат: результат `true`, если число элементов в множествах одинаково и `is(x = y)` дает `true`, где `x` – i-й элемент множества `s_1`, а `y` – i-й элемент множества `s_2`, для всех i
`subset(s,P)`	Извлечение элементов из множества `s` по предикату `P`
`subsetp(s_1,s_2)`	Предикат: результат `true`, если `s_1` – подмножество множества `s_2`

На экране компьютера множества всегда отображаются с ограничителями в виде фигурных скобок.

Если в перечне элементов множества имеются одинаковые элементы, то при обращении к множеству все из таких элементов, кроме одного, удаляются. В процессе процедуры удаления один из двух потенциальных кандидатов `x` и `y` является лишним, т.е. `x` и `y` считаются одинаковыми тогда и только тогда, когда предикат `is(x=y)` дает значение `true`. Заметим, что предикат `is(equal(x,y))` может давать результат `true`, тогда как предикат `is(x=y)` возвращает `false`; в этом случае элементы `x` и `y` считаются различными. Например, вследствие того, что результат обращения к предикату `is((x - 1)*(x + 1) = x^2 - 1)` равен `false`, то выражения `(x - 1)*(x + 1)` и `x^2 - 1` несовпадающие. Для отбрасывания повторяющихся элементов множеств в таких случаях необходимо использовать функции типа `rat`, `trigsimp` и подобные.

Некоторые действия, например подстановки, запускают автоматическое удаление повторяющихся элементов.

Как уже отмечалось, система Maxima рассматривает списки и множества как отдельные объекты. Поэтому такие функции, как `union` и `intersection`, при вызове возвращают сообщение об ошибке, если их какой-либо аргумент не является множеством. Если необходимо применить операцию с множествами к списку, нужно сначала воспользоваться

функцией `setify`, чтобы преобразовать список в множество.

```
(%i32)   makeset(i^2, [i], [[a], [b], [c]]);
(%o32)       {a^2,b^2,c^2}
```

```
(%i33)   makeset(i^2, [i], {[a], [b], [c]});
(%o33)       {a^2,b^2,c^2}
```

```
(%i34)   S1: {1, b, c}$ S2: {2, b, c}$ symmdifference(S1,S2);
(%o34)       {1,2}
```

```
(%i35)   subset({1, 2, 7, 8, 9, 14, 2.0}, evenp);
(%o35)       {2,8,14}
```

Перебрать все элементы множества можно двумя путями:

```
(%i36)   map(f, {a, b, c});
(%o36)       {f(a),f(b),f(c)}
```

```
(%i37)   s: {a,b,c,d}$ summa:0$ for si in s do summa: summa + si^2$
summa;
(%o37)       d^2+c^2+b^2+a^2
```

Функция	*Описание*
`adjoin(x,s)`	Включение элемента `x` в множество `s`
`cardinality(s)`	Число различных элементов множества `s`
`disjoin(x,s)`	Исключение элемента `x` из множества `s`
`disjointp(s_1,s_2)`	Предикат: результат `true`, если множества `s_1` и `s_2` непересекающиеся
`elementp(x,s)`	Предикат: результат `true`, если `x` – элемент множества `s`

Функция	*Описание*
`every(P,s)`	Предикат: результат `true`, если истинен результат применения предиката `P` ко всем элементам множества `s`
`some(P,s)`	Предикат: результат `true`, если истинен результат применения предиката `P` хотя бы к одному элементу множества `s`
`full_listify(s)`	Превращение множества в список
`fullsetify(L)`	Превращение сложного списка в множество

```
(%i38)   every(integerp, 1,2,3,4,5,6);
(%o38)       true
```

```
(%i39)  some(integerp, 1,2,3,4,5,6);
(%o39)      true
```

```
(%i40)  full_listify({1, 2, {3}, {{d,e,f}, g}});
(%o40)      [1, 2, [3], [[d, e, f], g]]
```

```
(%i41)  fullsetify([1, 2, [1], [2]]);
(%o41)      {1, 2, {1}, {2}}
```

Функция	*Описание*
`belln(n)`	Число всех неупорядоченных разбиений `n`-элементного множества
`cartesian_product(s_1, ...,s_n)`	Декартово произведение элементов списков `s_i`
`divisors(n)`	Множество делителей целого числа `n`
`equiv_classes(s,F)`	Множество классов эквивалентности множества `s` согласно отношению эквивалентности `F`
`flatten(s)`	Формирование множества из атомов, входящих в различные подмножества множества `s`
`integer_partitions(n)`	Разбиения натурального числа `n` на слагаемые
`integer_partitions(n,m)`	Разбиение натурального числа `n` на `m` слагаемых
`partition_set(a,P)`	Разбиение множества на подмножества согласно истинности предиката `P`
`set_partitions(s)`	Разбиение множества `s` на подмножества
`set_partitions(s,n)`	Подмножество разбиения множества `s` на подмножества

```
(%i42)  makelist(belln(i),i,0,7);
(%o42)      [1, 1, 2, 5, 15, 52, 203, 877]
```

```
(%i43)  cartesian_product({x,y,z}, {1,2,3});
(%o43)      {[x, 1], [x, 2], [x, 3], [y, 1], [y, 2], [y, 3], [z, 1], [z, 2], [z, 3]}
```

```
(%i44)  divisors(1792);
(%o44)      {1, 2, 4, 7, 8, 14, 16, 28, 32, 56, 64, 112, 128, 224, 256, 448, 896, 1792}
```

```
(%i45)  equiv_classes({1, 1.0, 2, 2.0, 3, 3.0, 4, 5.0}, equal);
(%o45)      {{1, 1.0}, {2, 2.0}, {3, 3.0}, {4}, {5.0}}
```

```
(%i46)  flatten({a, {b}, {a,b}, {{c}}, {a,{b,c}} });
(%o46)      {a, b, c}
```

```
(%i47)  integer_partitions(5);
(%o47)      {[1, 1, 1, 1, 1], [2, 1, 1, 1], [2, 2, 1], [3, 1, 1], [3, 2], [4, 1], [5]}
```

```
(%i48)  integer_partitions(5,3);
(%o48)      {[2, 2, 1], [3, 1, 1], [3, 2, 0], [4, 1, 0], [5, 0, 0]}
```

```
(%i49)  partition_set({2, 7, 1, 8, 2, 8}, evenp);
(%o49)      [{1, 7}, {2, 8}]
```

```
(%i50)  set_partitions(1,2,3);
(%o50)    {{{1}, {2}, {3}}, {{1}, {2, 3}}, {{1, 2}, {3}}, {{1, 2, 3}}, {{1, 3}, {2}}}
```

```
(%i51)  set_partitions(1,2,3,2);
(%o51)      {{{1}, {2, 3}}, {{1, 2}, {3}}, {{1, 3}, {2}}}
```

Функция	*Описание*
`kron_delta(x1,x2,...,xp)`	Символ Кронекера

```
(%i52)  kron_delta(a,b,a,b);
(%o52)      kron_delta(a, b)
```

```
(%i53)  kron_delta(a,a,b,a+1);
(%o53)      0
```

Массивы

В системе Maxima определены три конструкции типа массив. Списки уже были рассмотрены выше. Конструкция, напоминающая массивы в других языках программирования, описывается чуть ниже. Здесь же остановимся на использовании переменных с индексами, которые не требуют инициализации до начала использования и называются `hashed array`. Такие "массивы"[1] увеличиваются в размерах динамически, а для обращения к "элементам" этих "массивов" в качестве индексов можно использовать числа, символы и строки.

```
(%i54)  f["arg"]: 1$ f[4]: 5$ f[o]: p^2$
```

```
(%i55)  print(f["arg"], " ", f[4], " ", f[o]);
(%o55)      1  5  p^2
```

[1] Проще всего назвать такие объекты *неопределенными функциями*.

Функция	*Описание*
`array(name,dim_1,...,dim_n)`	Создание массива с размерностями `dim_1`, ..., `dim_n`
`array(name,type,dim_1,..., dim_n)`	Создание массива с элементами `type` и размерностями `dim_1`, ..., `dim_n`
`array([name_1,...,name_m], dim_1,...,dim_n)`	Создание массивов с размерностями `dim_1`, ..., `dim_n`
`arrayapply(A,[i_1,...,i_n])`	Получение значение элемента `A[i_1,..., i_n]` массива `A`
`arrayinfo(A)`	Получение информации по массиву
`arraymake(A,[i_1,...,i_n])`	Генерирование выражения `A[i_1,...,i_n]`
`arrays`	Список задекларированных массивов
`arraysetapply(A,[i_1,..., i_n],x)`	Назначение `x` элементу `A[i_1,...,i_n]` массива `A`
`fillarray(A,B)`	Заполнение массива `A` элементами списка или массива `B`
`listarray(A)`	Вывод на консоль элементов массива
`make_array(type,dim_1,..., dim_n)`	Создание Lisp-массива
`rearray(A,dim_1,...,dim_n)`	Изменение размерностей массива
`subvar(x,i)`	Вычисление индексированного выражения `x[i]`
`subvarp(expr)`	Предикат: результат `true`, если `expr` – индексированная переменная

`n` не может превышать 5. Индексы могут изменяться от 0 до `dim_i`. `type` может быть `fixnum` для целых чисел ограниченного размера или `flonum` для чисел с плавающей точкой.

Функция `arrayapply` напоминает функцию `apply`: различие в том, что вместо имени функции в `apply` для `arrayapply` используется имя массива.

```
(%i56)   arraymake(A,[i,j,3]);
(%o56)      A_{i,j,3}
```

```
(%i57)   array(arr,fixnum,8)$ listarray(arr);
(%o57)      [0,0,0,0,0,0,0,0,0]
```

```
(%i58)   fillarray(arr,makelist(i,i,0,8))$ listarray(arr);
(%o58)      [0,1,2,3,4,5,6,7,8]
```

Функция `make_array` создает и возвращает Lisp-массив. `type` может быть `any`, `flonum`, `fixnum`, `ashed` или `functional`. `i`-й индекс изменяется от 0 до `dim_i`–1.

Преимущество `make_array` перед `array` состоит в том, что возвращаемое значение не имеет имени. При этом как только указатель на него будет удален, такой массив исчезнет. Например, пусть `y: make_array(...)`. Тогда `y` указывает на объект, который занимает место, но после `y: false`, `y` больше не указывает на этот объект, поэтому объект доступен для процедуры сборки мусора.

```
(%i59)   Arr: make_array(fixnum,10);
(%o59)       #{Lisp array[10]}
```

Структуры

Система Maxima предоставляет простой агрегат данных, называемый структурой. Структура – это выражение, в котором поля идентифицируются по имени, а выражение в целом определяется его оператором (именем структуры). Значением поля может быть любое выражение.

Структура определяется функцией `defstruct`. Глобальная переменная `structures` содержит список пользовательских структур. Функция `new` создает экземпляры структур. Оператор `@` относится к полям. Вызов `kill(S)` удаляет определение структуры `S`, а `kill(x @ a)` отменяет привязку поля `a` экземпляра структуры `x`.

При двумерном выводе экземпляры структуры отображаются в виде, в котором значение каждого поля, представленного в виде равенства, с именем поля в левой части и значением в правой части. В одномерном виде экземпляры структуры отображаются без имен полей.

Нельзя использовать имя поля в качестве имени функции. Также значения полей не могут быть ограничены определенными типами: любому полю можно присвоить любое выражение. Все поля всегда доступны.

Функция	*Описание*
`defstruct(S(a_1,...,a_n))`	Определение структуры

Функция	*Описание*
`defstruct(S(a_1=v_1,..., a_n=v_n))`	Определение структуры с заданием полей
`new`	Создание нового экземпляра структуры
`@`	Оператор доступа к полям структуры

Функция `defstruct(S(a_1, ..., a_n))` определяет структуру, которая представляет собой список именованных полей `a_1`, ..., `a_n`, связанных с символом `S`. Экземпляр структуры – это просто выражение, которое

имеет оператор S и ровно n аргументов. Функция new(S) создает новый экземпляр структуры S.

Аргумент, который является просто символом a, определяет имя поля. Аргумент, который является равенством a = v, задает имя поля a и его значение v по умолчанию. Значением по умолчанию может быть любое выражение.

Функция defstruct помещает S в список пользовательских структур. Функция kill(S) удаляет S из списка пользовательских структур и стирает определение структуры.

```
(%i60)   defstruct(coord(x,y,z));
(%o60)       [coord(x, y, z)];
```

```
(%i61)   coordA: new(coord(2,8,5));
(coordA)        coord(x = 2, y = 8, z = 5)
```

```
(%i62)   coordA@y;
(%o62)       8
```

3.6. Операции и операторы. Арифметические операции

Операторы – это элементы языка, с помощью которых создаются математические выражения. К ним, например, принадлежат знаки арифметических операций, вычисления сумм, произведений, производных, интегралов. Операторы используются вместе с операндами. Например, в выражении "5–1" знак "–" является оператором (операцией) вычитания, а числа "5" и "1" – *операндами*.

Операции	*Описание*
+	Сложение
-	Вычитание
/	Деление
*	Умножение
^	Возведение в степень

Примеры использования:

```
(%i1)   5+3;
(%o1)      8
```

```
(%i2)   22-7;
(%o2)      15
```

```
(%i3)   121/11;
(%o3)       11
```

```
(%i4)   25*4;
(%o4)       100
```

```
(%i5)   2^8;
(%o5)       256
```

```
(%i6)   (2+3)^4-(5+6+7)/(2+4);
(%o6)       622
```

Maxima позволяет работать с числами произвольной разрядности:

```
(%i7)   2^100;
(%o7)       1267650600228229401496703205376
```

Оператор	*Описание*
`:`	Оператор присваивания
`:=`	Оператор определения функции
`kill(x)`	Удаление все связываний (значение, описание функции) символа x
`kill(all)`	Удаление связывания (значения, описания функций) у всех символов

В среде XMaxima удалить все связывания можно с помощью главного меню: в русифицированной версии см. подменю меню Maxima. То же, но в среде wxMaxima: пункт главного меню Maxima -> Перезапустить Maxima.

Переменная	По умолчанию	*Описание*
`values`	`[]`	Список переменных, определенных пользователем
`functions`	`[]`	Список функций, определенных пользователем

Оператор присваивания `:` можно использовать для глобального изменения значения системных переменных. Команда `reset()` позволяет сбросить глобальные системные переменные и некоторые другие переменные в их значения по умолчанию. Альтернативным подходом к изменению и сбросу системных переменных является использование команды `ev`, которая позволяет вычислить выражение с локально измененными системными переменными.

Выражения, подобные числам, могут храниться в переменных, которые именуются последовательностями алфавитно-цифровых символов, начинающихся с буквы. Алфавитно-цифровые символы – это символы от `A` до `Z`, от `a` до `z`, от `0` до `9`, `%` и `n_`. Примеры допустимых имен переменных (идентификаторов): `a`, `nbeta` или `ny_1`. Таким образом, можно сохранить результат любого расчета с помощью оператора присваивания `n:`. Этот оператор вычисляет свою правую часть и связывает это значение с левой частью – переменной. Когда переменная используется в дальнейших расчетах, она заменяется ее значением. Например:

```
(%i8)   x;
(%o8)     x
```

```
(%i9)   x: 2+3;
(%o9)     5
```

```
(%i10)   x
(%o10)     5
```

```
(%i11)   y: 2*x^2;
(%o11)     50
```

Отметим, что Maxima чувствительна к используемому регистру клавиатуры, т.е. идентификаторы `foo`, `FOO` и `Foo` различны.

Функция	*Определение*
`ev(x,envir)`	Вычислить выражение `x` в указанной среде `envir`
`reset()`	Сбрасывает глобальные системные переменные

В системе Maxima любое выражение можно вычислить принудительно: такое принуждение означает, что получат значения не только само выражение, но и все входящие в него символы. Такое действие осуществляется с помощью функции `ev`. При этом каждое применение `ev` вычисляет выражение только на один уровень в глубину, но эту функцию можно применять к любому выражению сколько угодно раз.

При вызове расчетов в ячейке ввода с помощью двух апострофов эта ячейка будет автоматически вычислена. Изюминка расчета через `ev` в том, что будут вычислены все символы, входящие в выражение, независимо от того, блокировался ли их расчет при определении выражения, или сами эти символы на тот момент еще ничего не определяли, т.е. были атомами.

Вообще говоря, `ev` – едва ли не самая мощная функция в системе Maxima. С помощью дополнительных параметров, которые могут быть переданы этой функции после вычисляемого выражения в списке любой длины, можно достаточно гибко управлять всеми процессами изменения объектов.

Функция	*Определение*
`float(x)`	Преобразовать `x` в число с плавающей точкой
`bfloat(x)`	Преобразовать `x` в длинное число с плавающей точкой
`rationalize(expr)`	Преобразовать все числа с плавающей точкой и все длинные числа с плавающей точкой в выражении `expr` в рациональные числа
`evenp(expr)`	Предикат: результат `true`, если `expr` – четное число, иначе `false`
`oddp(expr)`	Предикат: результат `true`, если `expr` – нечетное число, иначе `false`
`integerp(expr)`	Предикат: результат `true`, если `expr` – целое число, иначе `false`
`nonnegintegerp(n)`	Предикат: результат `true`, если `expr` – неотрицательное целое число, иначе `false`
`floatnump(expr)`	Предикат: результат `true`, если `expr` – число с плавающей точкой, иначе `false`
`bfloatp(expr)`	Предикат: результат `true`, если `expr` – длинное число с плавающей точкой, иначе `false`
`numberp(expr)`	Предикат: результат `true`, если `expr` – целое, рациональное число, число с плавающей точкой или длинное число с плавающей точкой в цифровой записи, иначе `false`
`ratnump(x)`	Предикат: результат `true`, если `x` – целое или рациональное число в цифровой записи, иначе `false`
`set_display(ascii)`	Установить режим дисплея в текстовую форму
`set_display(xml)`	Установить режим дисплея в форму XML (по умолчанию)

Системная переменная	*По умолчанию*	*Определение*
`numer`	`false`	Если `true`, то математические функции вычисляются в арифметике чисел с плавающей точкой
`algebraic`	`false`	Если `true`, то алгебраические числа упрощаются

Когда необходим результат в форме чисел с плавающей точкой, то системную переменную `numer` нужно установить в значение `true`. Тогда далее все расчеты будут выполняться в числах с плавающей точкой. Но это можно сделать и локально, используя команду `ev`.

```
(%i12)   ev(19/3, numer:true);
(%o12)      6.333333333333333
```

Во многих ситуациях можно упростить обозначения. `numer` интерпретируется как `numer: true`. Таким образом, `ev` тоже может быть опущено.

```
(%i13)   19/3, numer;
(%o13)      6.333333333333333
```

```
(%i14)   sin(4), numer;
(%o14)      −0.75680249530793
```

```
(%i15)   %pi, numer;
(%o15)      3.141592653589793
```

Альтернативный подход – использовать команду `float`.

```
(%i16)   float(19/3);
(%o16)      6.333333333333333
```

```
(%i17)   float(sin(4));
(%o17)      −0.75680249530793
```

```
(%i18)   float(%pi);
(%o18)      3.141592653589793
```

Оболочка wxMaxima пытается печатать вывод результатов, полученных ядром системы Maxima в приятном виде. Числа обычно печатаются в одну строку. Предположим, нужно вывести все цифры `100!` или первые 500 цифр числа π. При этом отображаются не все цифры, которые вы хотите видеть[2].

```
(%i19)   100!;
(%o19)      933262154439441526816992388562[98 digits]916864000...000
```

```
(%i20)   fpprec: 500$
```

```
(%i21)   bfloat(%pi);
(%o21)      3.14159265358979323846264338327[443 digits]8857...19491b0
```

[2] В следующих двух примерах вывода на экран из-за меньшей ширины листа данного пособия по сравнению с окном wxMaxima ряд цифр заменен многоточием. Далее в соответствующих примерах разбиение искусственно.

```
(%i22)    reset()$
```

Но можно вернуться к базовому формату интерфейса Maxima с помощью команды `set_display(ascii)`. При этом все цифры напечатаются, но не в очень хорошей форме. Знак обратной косой черты \ в конце каждой строки указывает, что вывод числа продолжается в следующей строке. Но нужно не забыть снова перебросить флажок формата отображения с помощью `set_display(xml)`.

```
(%i23)    set_display(ascii)$
```

```
(%i24)    100!;
(%o24)         933262154439441526816992388562667004907159682643816214685\
929638952175999932299156089414639761565182862536979208272237582511852\
109168640000000000000000000000000
```

```
(%i25)    fpprec: 500$
```

```
(%i26)    bfloat(%pi);
(%o26)         3.14159265358979323846264338327950288419716939937510582097\
494459230781640628620899862803482534211706798214808651328230664709384\
460955058223172535940812848111745028410270193852110555964462294895493\
038196442881097566593344612847564823378678316527120190914564856692346\
034861045432664821339360726024914127372458700660631558817488152092096\
282925409171536436789259036001133053054882046652138414695194151160943\
305727036575959195309218611738193261179310511854807446237996274956735\
188575272489122793818301194910b0
```

```
(%i27)    reset()$
```

```
(%i28)    set_display(xml)$
```

3.7. Встроенные математические функции

В системе Maxima имеется достаточно большой набор встроенных математических функций.

Функция	*Смысл*	*Функция*	*Смысл*
`abs(x)`	$\|x\|$	`ceiling(x)`	Округление вверх
`sqrt(x)`	$\sqrt{x}$	`floor(x)`	Округление вниз
`log(x)`	$\ln x$	`round(x)`	Округление к ближайшему целому
`exp(x)`	e^x	`truncate(x)`	Округление к ближайшему целому$^{(*)}$

Функция	*Смысл*	*Функция*	*Смысл*
`mod(x,y)`	Остаток от деления	`entier(x)`	Округление вниз(*)
`min(x1,...,xn)`	Минимум	`fix(x)`	Округление вниз(*)
`max(x1,...,xn)`	Максимум	`signum(x)`	Знак числа

Примечание. (*) См. примеры.

```
(%i1)   [floor(-1.1), floor(-1.9), floor(1.1), floor(1.9),
ceiling(-1.1), ceiling(-1.9), ceiling(1.1), ceiling (1.9)];
(%o1)       [−2, −2, 1, 1, −1, −1, 2, 2]
```

```
(%i2)   [entier(-1.1), entier(-1.9), entier(1.1), entier(1.9),
fix(-1.1), fix(-1.9), fix (1.1), fix(1.9)];
(%o2)       [−2, −2, 1, 1, −2, −2, 1, 1]
```

```
(%i3)   [round(-1.1), round(-1.9), round(1.1), round(1.9),
truncate(-1.1), truncate(-1.9), truncate(1.1), truncate(1.9)];
(%o3)       [−1, −2, 1, 2, −1, −1, 1, 1]
```

```
(%i4)   [floor(%e), floor(%pi), ceiling(%e), ceiling(%pi),
entier(%e), entier(%pi), fix(%e), fix(%pi), round(%e), round(%pi),
truncate(%e), truncate(%pi)];
(%o4)       [2, 3, 3, 4, 2, 3, 2, 3, 3, 3, 2, 3]
```

Функция	*Смысл*	*Функция*	*Смысл*
`sin(x)`	$\sin x$	`sinh(x)`	$\operatorname{sh} x$
`cos(x)`	$\cos x$	`cosh(x)`	$\operatorname{ch} x$
`tan(x)`	$\operatorname{tg} x$	`tanh(x)`	$\operatorname{th} x$
`cot(x)`	$\operatorname{ctg} x$	`coth(x)`	$\operatorname{cth} x$
`sec(x)`	$\sec x$	`sech(x)`	$\operatorname{sch} x$
`csc(x)`	$\operatorname{cosec} x$	`csch(x)`	$\operatorname{csch} x$
`asin(x)`	$\arcsin x$	`asinh(x)`	$\operatorname{arsh} x$
`acos(x)`	$\arccos x$	`acosh(x)`	$\operatorname{arch} x$
`atan(x)`	$\operatorname{arctg} x$	`atanh(x)`	$\operatorname{arth} x$
`acot(x)`	$\operatorname{arcctg} x$	`acoth(x)`	$\operatorname{arcth} x$
`asec(x)`	$\operatorname{arcsec} x$	`asech(x)`	$\operatorname{arsch} x$
`acsc(x)`	$\operatorname{arccosec} x$	`acsch(x)`	$\operatorname{arcsch} x$
`atan2(y,x)`	$\operatorname{arctg} \frac{y}{x} \in (-\pi/2, \pi/2)$		

Примечание. Необходимо абсолютно точно писать обозначения функций и операторов системы Maxima и не пытаться использовать обозначения, принятые в русскоязычной математической литературе!

```
(%i5)   cos(%pi);
(%o5)      -1
```

```
(%i6)   exp(2);
(%o6)      %e^2
```

```
(%i7)   abs(-5);
(%o7)      5
```

Заметим, что аргументы `sin` и `cos` должны быть углами, выраженными в радианах (т.е. прямой угол $90^\circ = \pi/2$). Если нужно выполнить вычисления с использованием градусов, то нужно преобразовать градусы d в радианы x по формуле $x = d\,\pi/180$.

Экспоненциальная функция для произвольного основания a ($a > 0$, $a \neq 1$) может быть вычислена с помощью оператора `^`. Аналогично корень n-й может быть определен, если воспользоваться выражением `^(1/n)`. Отметим, что `%e^x` является альтернативой для `exp(x)`.

```
(%i8)   10^4;
(%o8)      1000
```

```
(%i9)   10000^(1/3);
(%o9)      10
```

```
(%i10)   8^(2/3);
(%o10)     4
```

Maxima имеет только функцию натурального логарифма (по основанию e). Логарифм по произвольному основанию a ($a > 0$, $a \neq 1$) может быть вычислен с использованием формулы:

$$\log_a x = \frac{\ln x}{\ln a}.$$

```
(%i11)   log(%e);
(%o11)     1
```

```
(%i12)   ln(%e);
(%o12)     1
```

3.8. Алгебраические (символьные) преобразования и подстановки

Ядро Maxima выполняет некоторый цикл действий в ответ на каждую новую введенную пользователем команду. Он состоит из четырех этапов: чтение или просмотр ввода, вычисление, упрощение и вывод. Анализатор преобразует синтаксически допустимую последовательность типизированных символов в структуру данных, которая будет использоваться для остальных операций. *Вычисление* включает замену имен на присвоенные им значения. *Упрощение* означает переписывание выражения для облегчения понимания пользователем или другими программами. *Вывод* включает в себя отображение результатов расчетов в различных форматах и обозначениях.

Вычисление и упрощение иногда имеют схожую функциональность, поскольку их цель – устранить "сложность", а разработчики системы иногда разделяют задачу так, что она выполняется частично в каждой. Например, `integrate(x,x)` получает ответ как `x*x/2`, который затем упрощается до $x^2/2$.

Вычисление всегда присутствует: это является следствием наличия системы программирования с функциями, подпрограммами, переменными, значениями, циклами, присваиваниями и т.д. На этапе расчетов имена встроенных или пользовательских функций заменяются их определениями, переменные заменяются их значениями. Это во многом аналогично действиям вычислителя обычного языка программирования, но распространяется на работу с символьными математическими данными. Из-за возможности применения различных математических процедур существуют различные возможные модели расчетов, и поэтому системы имеют дополнительные *флажки* (*переключатели*), которые могут управлять процессом вычисления.

С другой стороны, цель упрощения состоит в том, чтобы сохранить значение выражения, переформулировав его представление, чтобы оно было меньше, проще для понимания или соответствовало конкретным спецификациям (таким как "факторизован", "скобки раскрыты"). Например, `sin(0)` заменяется на нуль или `x + x` – на `2*x`. Существует несколько мощных инструментов для изменения результатов упрощения, так как именно в этой части системы пользователь может включить в систему Maxima знание недавно введенных функций или символьную нотацию.

Упрощение обычно делается на четырех разных уровнях:

– внутренний, встроенный автоматизированный блок упрощения;

– встроенные процедуры упрощения, которые могут быть явно вы-

званы пользователем в выбранных местах в программе или последовательности команд;

– пользовательские подпрограммы упрощения, связанные с блоком упрощения с помощью "tellimp" или "tellsimpafter", которые вызываются автоматически;

– пользовательские подпрограммы, которые могут быть явно вызваны пользователем в выбранных местах в программе или последовательности команд.

Функция	*Описание*
`expand(expr)`	Раскрыть скобки в выражении `expr`
`expandwrt(expr,x_1,...,x_n)`	Раскрытие скобок в выражении `expr` относительно переменных `x_1`, ..., `x_n`
`expandwrt_factored`	Раскрытие скобок в сомножителях выражения `expr` относительно переменных `x_1`, ..., `x_n`
`(expr,x_1,...,x_n)`	
`ratexpand(expr)`	Раскрыть скобки в рациональном выражении `expr` (более эффективная команда, чем `expand`)
`factor(expr)`	Факторизация выражения `expr`
`combine(expr)`	Объединение слагаемых с идентичным знаменателем
`rncombine(expr)`	Объединение слагаемых с идентичным знаменателем или знаменателями, отличающимися только числовыми множителями
`xthru(expr)`	Приведение слагаемых к общему знаменателю без раскрытия скобок и факторизации
`multthru(mult,sum)`	Умножение каждого слагаемого в сумме на множитель
`ratsimp(expr)`	Упрощение (рационального) выражения `expr`
`fullratsimp(expr)`	Применять `ratsimp` и подобные команды до тех пор, пока происходят изменения
`radcan(expr)`	Упрощение выражения, которое содержит показательные функции, логарифмы и радикалы
`rootscontract(expr)`	Преобразование произведений корней в корни от произведений
`num(expr)`	Числитель выражения `expr`
`denom(expr)`	Знаменатель выражения `expr`
`trigsimp(expr)`	Упрощает выражение, которое содержит тригонометрические функции

Функция	*Описание*
`trigreduce(expr)`	Упрощает выражение, которое содержит тригонометрические функции, используя переход к кратным аргументам
`exponentialize(expr)`	Замена тригонометрических и гиперболических функций через экспоненты

Когда Maxima оценивает выражения, она обычно не выполняет все "очевидные" расширения или упрощения. Она также не пытается преобразовать свои результаты в "приятную" форму. Однако Maxima предоставляет набор команд, которые пытаются сделать определенные упрощения или преобразования выражения. Каждая из них применяет несколько правил, которые воплощают обычные понятия простоты. Упрощение выражения является довольно сложной задачей для системы компьютерной алгебры, даже несмотря на то, что существуют стандартные подпрограммы для стандартных процедур упрощения.

Системная переменная	*По умолчанию*	*Описание*
`expandwrt_denom`	`false`	Если `true`, то скобки в выражении `expr` функцией `expandwrt` относительно переменных `x_1`, ..., `x_n` раскрываются и в числителе, и в знаменателе; иначе только в числителе
`expon`	`0`	Модуль максимальной отрицательной степени, для которой скобки раскрываются автоматически
`expop`	`0`	Максимальная положительная степень, для которой скобки раскрываются автоматически
`exponentialize`	`false`	Если `true`, то автоматически выполняется функция `exponentialize`
`radexpand`	`true`	Контроль упрощения радикалов
`rootsconmode`	`true`	Контроль работы функции `rootscontract`

Уточним, как действуют установки флажка `radexpand` на выражение `sqrt(x^2)`:

– если `radexpand=all` или выполнено `assume(x>0)`, то `sqrt(x^2)` упрощается до `x`;

– если `radexpand=true`, а область `domain=real` (по умолчанию), то `sqrt(x^2)` упрощается до `abs(x)`;

– если `radexpand=false` или `radexpand=true`, а область переменных `domain=complex`, то `sqrt(x^2)` не упрощается.

Для использования некоторых функций нужно предварительно загрузить пакет `lrats` (`fullratsubst`, `lratsubst`).

Произведения сумм можно раскрывать с помощью функций `expand` и `ratexpand`. Обратной является команда `factor`, применяемая для представления выражения в виде произведения неприводимых термов (которые называются *простыми*, например, простых чисел).

```
(%i1)  (x+y)*(x-y);
```
$$(x-y)\,(x+y)$$

```
(%i2)  expand(%);
```
$$x^2-y^2$$

```
(%i3)  factor(%);
```
$$-(y-x)\,(y+x)$$

```
(%i4)  (x+y)^4;
```
$$(y+x)^4$$

```
(%i5)  expand(%);
```
$$y^4+4\,x\,y^3+6\,x^2\,y^2+4\,x^3\,y+x^4$$

```
(%i6)  factor(%);
```
$$(y+x)^4$$

```
(%i7)  factor(-8*y-4*x+z^2*(2*y+x));
```
$$(2\,y+x)\,(z-2)\,(z+2)$$

```
(%i8)  factor(20!);
```
$$2^{18}\,3^8\,5^4\,7^2\,11\,13\,17\,19$$

```
(%i9)  r: %pi*(a*b+c+d)*(a+b)$ [expandwrt(r,a,b),
expandwrt_factored(r,a,b)];
```
$$[\pi\,b\,(d+c)+\pi\,a\,(d+c)+\pi\,a\,b^2+\pi\,a^2\,b,\pi\,(b\,(d+c)+a\,(d+c)+a\,b^2+a^2\,b)]$$

```
(%i10)  combine(x/(1+x^2)+y/(1+x^2));
```
$$\frac{y+x}{x^2+1}$$

```
(%i11)  xthru(1/(x+y)^10+1/(x+y)^12 );
```
$$\frac{(y+x)^2+1}{(y+x)^12}$$

```
(%i12)  ((x+2)^20-2*y)/(x+y)^20 + (x+y)^(-19) - x/(x+y)^20;
```

$$(\%o12)\qquad \frac{1}{(y+x)^{19}}+\frac{(x+2)^2 0-2\,y}{(y+x)^2 0}-\frac{x}{(y+x)^2 0}$$

```
(%i13)  xthru(%);
```

$$(\%o13)\qquad \frac{(x+2)^2 0-y}{(y+x)^2 0}$$

Функция `multthru` умножает каждое слагаемое в сумме на множитель, причём при умножении скобки в выражении не раскрываются.

```
(%i14)  x/(x-y)^2 - 1/(x-y) - f(x,y)/(x-y)^3;
```

$$(\%o14)\qquad -\frac{1}{x-y}+\frac{x}{(x-y)^2}-\frac{f(x,y)}{(x-y)^3}$$

```
(%i15)  multthru((x-y)^3,%);
```

$$(\%o15)\qquad x\,(x-y)-(x-y)^2-f(x,y)$$

Числитель рациональных терминов может быть разбит на соответствующие термы. Команда `ratexpand` устраняет общие делители в рациональных выражениях.

```
(%i16)  e: (x+y)^2/(x^2-y^2);
```

$$(\%o16)\qquad \frac{(y+x)^2}{x^2-y^2}$$

```
(%i17)  expand(e);
```

$$(\%o17)\qquad \frac{y^2}{x^2-y^2}+\frac{2\,x\,y}{x^2-y^2}+\frac{x^2}{x^2-y^2}$$

```
(%i18)  ratexpand(e);
```

$$(\%o18)\qquad -\frac{y}{y-x}-\frac{x}{y-x}$$

Система Maxima предоставляет команды для поиска эквивалентного, но более простого выражения для более сложного. Для этого применяется несколько правил, которые воплощают обычные понятия простоты. Упрощение выражения является довольно сложной задачей для системы компьютерной алгебры, даже несмотря на то, что существуют стандартные программы для стандартных процедур упрощения.

```
(%i19)  (x^2-y^2)/(x+y)^2;
```
(%o19)
$$\frac{x^2-y^2}{(y+x)^2}$$

```
(%i20)  ratsimp(%);
```
(%o20)
$$-\frac{y-x}{y+x}$$

```
(%i21)  (%e^x-1)/(1 + %e^(x/2));
```
(%o21)
$$\frac{\%e^x-1}{\%e^{x/2}+1}$$

```
(%i22)  radcan(%);
```
(%o22) $\%e^{x/2}-1$

Команды num и denom позволяют выделить числитель и знаменатель дроби.

```
(%i23)  num(a/b);
```
(%o23) a

```
(%i24)  num(a/b);
```
(%o24) b

```
(%i25)  (3+cos(x)^4+8*sin(x)+4*(cos(x)^2-sin(x)^2)-6*cos(x)^2
*sin(x)^2+sin(x)^4)/(8*cos(x)^3);
```
(%o25)
$$\frac{3+\cos(x)^4+8\,\sin(x)+4\,(\cos(x)^2-\sin(x)^2)-6\,\cos(x)^2\,\sin(x)^2+\sin(x)^4}{8\,\cos(x)^3}$$

```
(%i26)  trigsimp(%);
```
(%o26)
$$\frac{\sin(x)+\cos(x)^4}{\cos(x)^3}$$

```
(%i27)  rootsconmode: false$ rootscontract(x^(1/2)*y^(3/2));
```
(%o27) $\sqrt{x\,y^3}$

На самом деле система Maxima предлагает гораздо больше инструментов для упрощения, разложения, факторизации и сокращения символьных выражений. Системные переменные позволяют детально настраивать преобразования. Некоторые из этих преобразований могут быть применены с помощью меню среды wxMaxima.

Функция	*Описание*
`sum(expr,i,i_0,i_1)`	Сумма значений выражения `expr` по индексу `i`
`lsum(expr,x,L)`	Сумма значений выражения `expr` для аргумента `x`, выбранных из списка `L`
`intosum(expr)`	Внесение множителя под знак суммы
`product(expr,i,i_0,i_1)`	Произведение значений выражения `expr` по индексу `i`
`sumcontract(expr)`	Объединение всех сумм с одинаковыми пределами

Системная переменная	*По умолчанию*	*Описание*
`simpsum`	`false`	Если `true`, то применяются правила упрощения для сумм
`simpproduct`	`false`	Если `true`, то применяются правила упрощения для произведений
`distribute_over`	`true`	Если `true`, то отображение применяется к элементам списков
`domain`	`real`	Если `complex`, то `sqrt(x^2)` не изменяется

Команда `sum` вычисляет сумму слагаемых, равных первому аргументу `expr` при различных значениях `i`, где индексная переменная `i` пробегает все целые значения от `i_0` до `i_1`.

```
(%i28)   sum(z(i),i,1,5);
(%o28)       z(5)+z(4)+z(3)+z(2)+z(1)
```

```
(%i29)   sum(z[i],i,1,5);
(%o29)       z5+z4+z3+z2+z1
```

```
(%i30)   sum(i^3,i,1,10);
(%o30)       3025
```

Отметим, что должно выполняться неравенство `i_1` $\geqslant$ `i_0`. В противном случае возвращается 0. Верхняя и нижняя границы могут также быть выражениями, включающими `minf` и `inf`.

```
(%i31)  sum(z[i],i,1,n);
```
$$\sum_{i=1}^{n} z_i$$

```
(%i32)  sum(z[i],i,1,inf);
```
$$\sum_{i=1}^{\infty} z_i$$

Система постарается упростить сумму, когда системную переменную `simpsum` установить в `true`.

```
(%i33)  sum(1/4^i,i,1,inf);
```
$$\sum_{i=1}^{\infty} \frac{1}{4^i}$$

```
(%i34)  sum(1/4^i,i,1,inf), simpsum;
```
$$\frac{1}{3}$$

```
(%i35)  sum(1/i^2,i,1,inf);
```
$$\frac{1}{i^2}$$

```
(%i36)  sum(1/i^2,i,1,inf), simpsum;
```
$$\frac{\pi^2}{6}$$

```
(%i37)  lsum(x^i, i, [a,b,c]);
```
$$x^c + x^b + x^a$$

По аналогии с расчетом сумм с помощью команды `product` можно вычислять произведения конечного или бесконечного числа сомножителей.

```
(%i38)  product(i^3,i,1,10);
```
$$47784725839872000000$$

```
(%i39)  sum(z(i),i,1,inf);
```
$$\prod_{i=1}^{\infty} z(i)$$

```
(%i40)  product(x+i*(i+1)/2,i,1,4);
```
$$(x+1)\,(x+3)\,(x+6)\,(x+10)$$

Оператор	*Описание*
`'expr`	Запрещение вычисления выражения `expr`
`''expr`	Инициирование расчета выражения `expr`

Пусть необходимо не вызвать на выполнение некоторую функцию, а использовать ее форму в формате ввода. Добиться этого можно, поставив перед именем функции знак апострофа. Оператор одинарного апострофа ' запрещает расчет выражения. Применительно к вызову функции этот оператор возвращает невычисленную форму вызова функции.

```
(%i41)   y: 2;
(%o41)      2
```

```
(%i42)   'y^2 = y^2;
(%o42)      y^2 = 4
```

```
(%i43)   f(x):= x^2;
(%o43)      f(x) := x^2
```

```
(%i44)   f(a);
(%o44)      a^2
```

```
(%i45)   'f(a);
(%o45)      f(a)
```

```
(%i46)   'limit(n*(sqrt(3*n^2+1)-sqrt(3*n^2+2)),n,inf) =
limit(n*(sqrt(3*n^2+1)-sqrt(3*n^2+2)),n,inf);
(%o46)      lim_{n→∞} n (sqrt(3 n^2 + 1) - sqrt(3 n^2 + 2)) = -1/(2 sqrt(3))
```

Оператор двойного апострофа '' (два одинарных апострофа) вызывает расчет выражения *expr* в случае, когда вычисление подавлено, например, в определениях функций. Затем значение *expr* подставляется вместо `expr` во входном выражении.

```
(%i47)   f(x):= x^2+y^2;
(%o47)      f(x) := x^2 + y^2
```

```
(%i48)   f(x):= x^2+''y^2;
(%o48)      f(x) := x^2 + 2^2
```

Это очень удобно, когда результат предыдущего расчета должен использоваться как тело функции новой функции. В следующем примере мы определяем функцию f и ее производную g, которую вычисляем с помощью команды `diff`.

```
(%i49)   f(x):= exp(-x^2);
(%o49)       f(x) := exp(-x^2)
```

```
(%i50)   g(x):= ''(diff(f(x),x));
(%o50)       g(x) := 3 x^2
```

Заметим, что оператор одинарного апострофа ' и оператор двойного апострофа '' действуют только на следующую переменную или функцию. В частности, `'f(x)` препятствует вычислению только символа `f`, но не его аргумента `x`. Когда соответствующий оператор должен предотвратить или вызвать расчет для более сложного выражения, например для `f(x)`, то он должен быть заключен в скобки, `'(f(x))`.

Следующие примеры демонстрируют это. Здесь вычисляется только символ `diff` (со значением `diff`), а не все выражение `diff(f(x), x)`, как в примере выше.

```
(%i51)   kill(f,g)$ f(x):= exp(-x^2);
(%o51)       f(x) := exp(-x^2)
```

```
(%i52)   g(x):= ''diff(f(x),x);
(%o52)       g(x) := f(x)_x
```

```
(%i53)   g(t);
(%o53)       3 t^2
```

Вот простой пример для оператора апострофа:

```
(%i54)   y: 2$
```

```
(%i55)   kill(f); f(x):= exp(-x^2)$
```

```
(%i56)   f(y);
(%o56)       %e^-4
```

```
(%i57)   'f(y);
(%o57)       f(2)
```

```
(%i58)   f('y);
(%o58)      %e^-y^2
```

```
(%i59)   '(f(y));
(%o59)      f(y)
```

Применяемый к выражению в скобках оператор апострофа предотвращает вычисление всех символов и вызовов функций в выражении, но не препятствует упрощению.

Оператор	*Описание*
`ev(x,envir)`	Вычисляет выражение `xpred` при установках среды `envir`
`apply(f,L)`	Вызов функции `f`, аргументы которой находятся в списке `L`
`map(f,expr1,...,exprn)`	Применение функции `f` к частям выражений `expr1`, ..., `exprn`
`maplist(f,expr1,...,exprn)`	Применение функции `f` ко всем частям выражений `expr1`, ..., `exprn`
`scanmap(f,expr)`	Рекурсивное применение функции `f`
`fullmap(f,expr_1,...)`	То же самое, что и `map`, но на всех уровнях
`fullmapl(f,list_1,...)`	То же самое, что и `map`, но на всех уровнях вложенности списков
`funmake(F,[arg_1,...,arg_n])`	Формирование `F(arg_1,...,arg_n)`

Аргументами функции `ev` являются переключатели (логические флаги), присваивания, уравнения и функции. Результат работы `ev` получается поэтапно:

1°. Сначала среда настраивается путем сканирования аргументов функции, которые могут быть любыми или всеми из следующих:

– `simp`: упрощение выражения `expr` независимо от значения переключателя `simp=false`, который запрещает упрощение;

– `noeval`: подавление вычисления. Это полезно в сочетании с другими переключателями и для упрощения выражения `expr` без получения ее значения;

– `nouns`: вычисление объектов форм существительных (таких как `'integrate` или `'diff`);

– `expand`: раскрытие скобок;

– `expand(m,n)`: раскрытие скобок с ограничениями по порядку степеней;

– `detout`: вычисление обратных матриц без деления каждого элемента на определитель;

– `diff`: выполнение всех дифференцирований в выражении;

– `derivlist(x,y,z,...)`: выполнение дифференцирований в выражении только по указанным переменным;

– `risch`: вычисление интегралов в выражении с использованием алгоритма Риша;

– `float`: преобразование нецелых рациональных чисел в числа с плавающей точкой;

– `numer`: вычисление некоторых математических функций (включая возведение в степень) с числовыми аргументами с помощью арифметики чисел с плавающей точкой;

– `pred`: получение значений предикатов;

– `eval`: функция, вызывающая дополнительное перевычисление `expr` и др.

Аргументы `ev` могут быть заданы в любом порядке, за исключением равенств с подстановками, которые обрабатываются последовательно, слева направо, и вычислительных функций, представляющих из себя композиции.

Переключатели `simp`, `numer` и `float` могут быть установлены локально в блоке или глобально в Maxima, чтобы они оставались в силе до сбрасывания.

2°. Если какие-либо замены указаны аргументами, то они выполняются.

Система Maxima использует несколько системных переменных для детального управления вычислениями. Например, ранее было указано, что переменная `fpprec` контролирует количество значащих цифр в числах с большим числом знаков. Например, предположим, что необходимо вычислить первые 50 цифр числа e, но изменять `fpprec` глобально нежелательно. Расчет может быть выполнен путем локального изменения значения `fpprec` в функции `ev`.

```
(%i60)   ev(bfloat(%e), fpprec:100);
(%o60)      2.7182818284590452353602874713526624977572470937b0
```

Если функция `ev` не используется как часть другого выражения, то при ее использовании достаточно просто привести аргументы `ev`.

```
(%i61)   bfloat(%e), fpprec:100;
(%o61)      2.7182818284590452353602874713526624977572470937b0
```

```
(%i62)   sin(x)+cos(y)+(w+1)^2+'diff(sin(w),w);
(%o62)      cos(y) + sin(x) + d/dw sin(w) + (w + 1)^2
```

```
(%i63)  ev(%,numer,expand,diff,x=2,y=1);
```
$$\cos(w) + w^2 + 2\,w + 2.449599732693821 \quad \text{(\%o63)}$$

```
(%i64)  [a:b,b:c,c:d,d:e];
```
$$[b, c, d, e] \quad \text{(\%o64)}$$

```
(%i65)  a;
```
$$b \quad \text{(\%o65)}$$

```
(%i66)  ev(a);
```
$$c \quad \text{(\%o66)}$$

```
(%i67)  ev(a),eval;
```
$$e \quad \text{(\%o67)}$$

```
(%i68)  a,eval,eval;
```
$$e \quad \text{(\%o68)}$$

Функция `apply(f,L)` вычисляет выражение `f(a1,a2,...,an)`, где `L =[a1,a2,...,an]`.

```
(%i69)  L: [1,5,10,4,3]$ apply("+",L);
```
$$23 \quad \text{(\%o69)}$$

```
(%i70)  apply(h,[x,y,z]);
```
$$h(x, y, z) \quad \text{(\%o70)}$$

Функция `map` при использовании со списками применяет `f` к каждому элементу списка, при обработке выражений – к каждому подвыражению. Следует обратить внимание, что `f` – именно имя функции (без указания переменных, от которых она зависит).

```
(%i71)  map(ratsimp, x/(x^2+x)+(y^2+y)/y);
```
$$y + \frac{1}{x+1} + 1 \quad \text{(\%o71)}$$

```
(%i72)  map(exp,[0,1,2,3,4,5]);
```
$$[1, \%e, \%e^2, \%e^3, \%e^4, \%e^5] \quad \text{(\%o72)}$$

```
(%i73)  map(f,x+a*y+b*z,t+n+x);
```
$$f(b\,z, x) + f(2\,y, t) + f(x, n) \quad \text{(\%o73)}$$

```
(%i74)  maplist(f,x+a*y+b*z,t+n+x);
```
(%o74) $[f(b\,z,x), f(2\,y,t), f(x,n)]$

```
(%i75)  scanmap(factor,(a^2+2*a+1)*y + x^2);
```
(%o75) $(a+1)^2\,y + x^2$

Функция	*Описание*
`logarc(expr)`	Замена в выражении `expr` обратных круговых и гиперболических функций эквивалентными логарифмическими функциями без установки системной переменной `logarc`
`logcontract(expr)`	Рекурсивное сканирование выражения `expr` с преобразованием подвыражения вида `a1*log(b1) + a2*log(b2) + c` в `log(ratsimp(b1^a1*b2^a2)) + c`
`trigexpand(expr)`	Заменяет тригонометрические и гиперболические функции сумм углов и кратных углов, встречающихся в `expr`
`trigreduce(expr,x)`	Исключение целых положительных степеней тригонометрических и гиперболических функций по отношению к аргументу `x`, в т.ч. и в знаменателе
`trigreduce(expr)`	Исключение целых положительных степеней тригонометрических и гиперболических функций по отношению ко всем аргументам
`trigsimp(expr)`	Применение основных тождеств для тригонометрических и гиперболических функций с целью упрощения выражений, содержащих тангенсы, секансы и т.д.
`trigrat(expr)`	Получение упрощенной квазилинейной формы тригонометрического дробно-рационального выражения относительно синусов, косинусов, тангенсов

Системная переменная	*По умолчанию*	*Описание*
`%e_to_numlog`	`false`	Если `true` и `r` – рациональное число, то `%e^(r*log(x))` заменяется на `x^r`
`%emode`	`true`	Если `true`, то упрощается выражение `%e^(%pi*%i*x)`
`%enumer`	`false`	Если `true`, то `%e` заменяется числом 2.718... при `numer = true`
`logarc`	`false`	Если `true`, то обратные тригонометрические и гиперболические функции заменяются эквивалентными логарифмическими
`logconcoeffp`	`false`	Контроль коэффициентов, связанных с функцией `logcontract`

Системная переменная	*По умолчанию*	*Описание*
`logexpand`	`true`	Управление преобразованием логарифмов
`lognegint`	`true`	Если `true`, то `log(-n)` заменяется на `log(n) + %i*%pi` для положительных целых `n`
`logsimp`	`true`	Если `false`, то никакие преобразования выражений с `%e` содержащих логарифмы, не выполняются
`%piargs`	`true`	Если `true`, то тригонометрические функции упрощаются до констант, когда их аргументы кратны `%pi`, `%pi/2`, `%pi/3`, `%pi/4` или `%pi/6`
`%iargs`	`true`	Если `true`, то тригонометрические функции превращаются в гиперболические, когда их аргументы кратны `%i`
`halfangles`	`false`	Если `true`, то тригонометрические функции от аргументов `expr/2` заменяются такими же функциями от `expr`
`trigexpandplus`	`true`	Если `true`, то при применении функции `trigexpand` упрощаются тригонометрические функции от сумм углов
`trigexpandtimes`	`true`	Если `true`, то при применении функции `trigexpand` упрощаются тригонометрические функции от кратных углов
`triginverses`	`true`	Контроль упрощения композиций тригонометрических функций и их обратных. То же для гиперболических функций
`trigsign`	`true`	Если `true`, то разрешено упрощение отрицательных аргументов тригонометрических функций

Действия, подобные (и более сложные) инициируемым переменной `%e_to_numlog`, выполняются функций `radcan`. Обратные манипуляции с выражениями, содержащими натуральные логарифмы, проводит функция `logcontract`.

Упрощение выражения `%e^(%pi*%i*x)` заключается в следующем: 1) если `x` – число с плавающей точкой, целое или кратное 1/2, 1/3, 1/4 или 1/6, то `%e^(%pi*%i*x)` заменяется на `cos(%pi*x) + %i*sin(%pi*x)` с дальнейшим упрощением; 2) если `x` – другое число, то `%e^(%pi*%i*x)` заменяется на `%e^(%pi*%i*y)`, где `y = x - 2k`, `k` – целое, причем `abs(y) < 1`. Если `%emode = false`, то упрощение `%e^(%pi*%i*x)` не производится.

```
(%i76)   [%e^(%pi*%i*1), %e^(%pi*%i*5/6), %e^(%pi*%i*180/144)];
(%o76)        [-1, %i/2 - sqrt(3)/2, -%i/sqrt(2) - 1/sqrt(2)]
```

```
(%i77)   a: 2*%e$ %enumer: not %enumer$ b: 2*%e$ numer: not numer$
c: 2*%e$ [a, b, c];
(%o77)       [2 %e, 2 %e, 5.43656365691809]
```

```
(%i78)   kill(a,x,y)$ logcontract(2*(a*log(x) + 2*a*log(y)));
(%o78)       a log(x^2 y^4)
```

```
(%i79)   numer: not numer$ logconfun(m):=featurep(m,integer) or
ratnump(m)$ logconcoeffp: 'logconfun$ logcontract(1/2*log(x));
(%o79)       log(sqrt(x))
```

Если:

1) `logexpand = true`, то `log(a^b)` заменяется на `b*log(a)`;

2) `logexpand = all`, то `log(a*b)` заменяется на `log(a) + log(b)`;

3) `logexpand = super`, то `log(a/b)` заменяется на `log(a) - log(b)` для рациональных чисел `a/b`;

4) `logexpand = false`, то все эти упрощения будут отключены;

5) `logexpand = all` или `super`, то логарифм произведения заменяется суммой логарифмов.

При применении функции `trigexpand(expr)` для улучшения результата желательно, чтобы в выражении `expr` были раскрыты скобки. Чтобы облегчить пользовательский контроль над упрощением, эта функция раскладывает только на одном уровне за раз, например, суммы углов или кратные углы. Для полного упрощения синусов и косинусов нужно установить переключатель `trigexpand` в `true`.

Если:

1) `triginverses = true`, то упрощение `arcfun(fun(x))` выключено;

2) `triginverses = all`, то `arcfun(fun(x))` и `fun(arcfun(x))` заменяется на `x`;

3) `triginverses = false`, то упрощения `arcfun(fun(x))` и `fun(arcfun(x))` отключены.

```
(%i80)   log(10*x), logexpand=all;
(%o80)       log(x) + log(10)
```

```
(%i81)   %piargs: false$ [sin(%pi), sin(%pi/2), sin(%pi/3)];
(%o81)       [sin(%pi), sin(%pi/2), sin(%pi/3)]
```

```
(%i82)   %piargs: true$ [sin(%pi), sin(%pi/2), sin(%pi/3)];
(%o82)       [0, 1, sqrt(3)/2]
```

```
(%i83)  %iargs: false$ [sin(%i*x), cos(%i*x), tan(%i*x)];
```
$$(\%o83)\quad [\sin(\%i\,x), \cos(\%i\,x), \tan(\%i\,x)]$$

```
(%i84)  %iargs: true$ [sin(%i*x), cos(%i*x), tan(%i*x)];
```
$$(\%o84)\quad [\%i\,\sinh(x), \cosh(x), \%i\,\tanh(x)]$$

```
(%i85)  halfangles: true$ assume(x>0, x<2*%pi)$ sin(x/2);
```
$$(\%o85)\quad \frac{\sqrt{1-\cos(x)}}{\sqrt{2}}$$

```
(%i86)  x+sin(3*x)/sin(x),trigexpand=true,expand;
```
$$(\%o86)\quad -\sin(x)^2 + 3\cos(x)^2 + x$$

```
(%i87)  expand(8*trigreduce(-sin(y)^2+3*cos(x)^4+5));
```
$$(\%o87)\quad 4\cos(2\,y) + 3\cos(4\,x) + 12\cos(2\,x) + 45$$

```
(%i88)  trigrat(sin(3*a)/sin(a+%pi/3));
```
$$(\%o88)\quad \sqrt{3}\sin(2\,a) + \cos(2\,a) - 1$$

Дополнительные правила упрощений для тригонометрических функций определены в пакетах `atrig1` и `ntrig`.

Ниже приведены характеристики функций, учитываемые при преобразованиях.

Характеристика	*Описание*
`collectterms(expr,arg_1,...,arg_n)`	Приведение подобных для `arg_1`, ..., `arg_n`
`facsum(expr,arg_1,...,arg_n)`	Форма выражения `expr` как сумма термов с множителями, состоящими из функций `arg_1`, ..., `arg_n`
`factorfacsum(expr,arg_1,...,arg_n)`	Форма выражения `expr`, полученная в результате применения функции `facsum(expr,arg_1,...,arg_n)` к каждому сомножителю `expr`
`rempart(expr,n)`	Удаление части `n` из выражения `expr`
`linear(expr,x)`	Представление `expr` в виде линейной формы от `x` или `false`

Пакет `absimp` содержит шаблоны сопоставления с образцом, которые расширяют встроенные правила упрощения для функций `abs` и `signum`. Этот пакет не нарушает отношения, установленные с помощью встроенной функции `assume` и с помощью описаний, таких как `mode_declare(m, even,n,odd)`, для четных или нечетных целых чисел.

```
(%i89)   load("absimp")$ [(abs(x))^2, cosh(abs(x))];
```
(%o89) $[x^2, \cosh(x)]$

Пакет `facexp` содержит несколько связанных функций, которые предоставляют пользователю возможность структурировать выражения путем контролируемого раскрытия скобок.

Функция `rempart` требует загрузки пакета `functs`, функция `linear` – пакета `antid`.

```
(%i90)   load("facexp");
```

```
(%i91)   expand((exp(x)+2)*x+exp(x));
```
(%o91) $x\,\%e^x + \%e^x + 2\,x$

```
(%i92)   collectterms(%,exp(x));
```
(%o92) $(x+1)\,\%e^x + 2\,x$

```
(%i93)   facsum(d*e*f^2+c*e*f^2-d*e+c*e+b*c*d+a*c*d,e,f);
```
(%o93) $(d+c)\,e\,f^2 - (d-c)\,e + (b+a)\,c\,d$

```
(%i94)   r: c*d*(x*y*z + w*y*z + u*v^2*z - u*z + u*v^2*y + u*y) +
d*e*f^2 + c*e*f^2 - d*e + c*e + b*c*d$ facsum(r,c,d,[u,v]);
```
(%o94) $c\,d\,(u\,v^2\,(z+y) - u\,(z-y) + x\,y\,z + w\,y\,z + b) + c\,e\,(f^2+1) + d\,e\,(f-1)\,(f+1)$

```
(%i95)   facsum(subst(log(m+n),c,r),log(m+n),[e,f]);
```
(%o95) $log(n+m)\,(d\,(x\,y\,z + w\,y\,z + u\,v^2\,z - u\,z + u\,v^2\,y + u\,y + b) + e\,f^2 + e) + d\,e\,f^2 - d\,e$

```
(%i96)   r: x+(a.b)*(c+d)+(b.a)*(d+c)+log(a*b)*2*(c+d)+log(a/b)*2*(d
+c)$ q: expand(r*2*(e+(h+f)*(e.f)))$ factorfacsum(q,operator(".",log
));
```
(%o96) $2\,((e.f)\,(h+f)+e)\left(x + 2\,\log(a\,b)\,(d+c) + (b.a)\,(d+c) + 2\,\log\left(\frac{a}{b}\right)(d+c) + (a.b)\,(d+c)\right)$

```
(%i97)   load("antid")$ linear((1-w)*(1-x)*z,z);
```
(%o97) $[bargumentb = 0, aargumenta = (w-1)*x-w+1, xargumentx = z]$

Функция	*Описание*
`distrib(expr)`	Применение дистрибутивного закона
`properties(a)`	Список свойств символа `a`
`rem(atom,indicator)`	Удаление свойства `indicator`
`remove(a_1,p_1,...,a_n,p_n])`	Удаление свойств у символов
`remove(a_1,...,a_m],[p_1,...,p_n],...)`	
`remove("a",operator)`	
`remove(a,transfun)`	
`remove(all,p)`	
`constantp(expr)`	Предикат: результат `true`, если `expr` – константа
`featurep(a,f)`	Предикат: результат `true`, если `a` имеет свойство
`nonscalarp(expr)`	Предикат: результат `true`, если `expr` – не константа
`scalarp(expr)`	Предикат: результат `true`, если `expr` – число, постоянная или переменная, имеющая свойство `scalar`

Опциями для функции `declare(f,<опция>)`, применяемыми при преобразовании выражений, являются:

Опция	*Описание*
`additive`	Аддитивная функция (оператор)
`antisymmetric`	Антисимметрическая функция (оператор)
`symmetric`	Симметрическая функция (оператор)
`lassociative`	Левоассоциативная функция (оператор)
`rassociative`	Правоассоциативная функция (оператор)
`commutative`	Коммутативная функция (оператор)
`multiplicative`	Мультипликативная функция (оператор)
`linear`	Линейная функция (оператор)
`evenfun`	Четная функция (оператор)
`oddfun`	Нечетная функция (оператор)
`decreasing`	Убывающая функция (оператор)
`increasing`	Возрастающая функция (оператор)
`integervalued`	Функция (оператор), принимающая целые значения
`posfun`	Функция (оператор), принимающая положительные значения
`outative`	Возможность вынесения множителя за знак функции (оператора)
`constant`	Указание на постоянность символа
`even`	Указание на четность символа
`odd`	Указание на нечетность символа

Опция	*Описание*
`integer`	Указание на целое значение символа
`noninteger`	Указание на нецелое значение символа
`rational`	Указание на рациональное значение символа
`irrational`	Указание на иррациональное значение символа
`real`	Указание на действительное значение символа
`imaginary`	Указание на мнимое значение символа
`complex`	Указание на комплексное значение символа
`scalar`	Указание на скалярное значение символа

Более тонкие средства пользователя для упрощений и подстановок описаны в разделе 34 справочника по системе.

3.9. Выделение частей и анализ структуры выражений

Функция	*Описание*
`ordergreat(v_1,...,v_n)`	Изменение канонического порядка для переменных на следующий: `v_1` после `v_2`, ..., `v_(n-1)` после `v_n`
`orderless(v_1,...,v_n)`	Изменение канонического порядка для переменных на следующий: `v_1` ранее `v_2`, ..., `v_(n-1)` ранее `v_n`
`ordergreatp(expr_1,expr_2)`	Предикат: если `true`, то `expr_1` следует за `expr_2`
`orderlessp(expr_1,expr_2)`	Предикат: если `true`, то `expr_1` предшествует `expr_2`
`part(expr,n_1,...,n_k)`	Часть отображаемой формы выражения `expr`
`inpart(expr,n_1,...,n_k)`	Часть внутренней формы выражения `expr`
`allbut`	Модификация работы функции `part`
`args(expr)`	Список аргументов выражения `expr`
`atom(expr)`	Предикат: если `true`, то `expr` – атом
`symbolp(expr)`	Предикат: если `true`, то `expr` – символ
`copy(expr)`	Копия выражения `expr`
`isolate(expr,x)`	Выделение частей выражения `expr`, не содержащих `x`
`freeof(x_1,...,x_n,expr)`	Предикат: если `true`, то `expr` не включает `x_1`, ..., `x_n`
`listofvars(expr)`	Список переменных выражения `expr`
`lfreeof(L,expr)`	Предикат: если `true`, то `expr` не включает переменные из списка `L`
`nterms(expr)`	Число термов в `expr`
`op(expr)`	Ведущая операция в `expr`
`operatorp(expr,op)`	Предикат: если `true`, то `op` – ведущая операция в `expr`

Функция	*Описание*
`optimize(expr)`	Оптимизация выражения по отношению к эффективности вычислений
`partition(expr,x)`	Разбиение выражения `expr` на две части, одна из которых зависит от `x`, а другая нет

Все атомы и выражения в языке системы Macsyma сравнимы с помощью предикатов `ordergreatp` и `orderlessp`, хотя есть отдельные примеры выражений, для которых эти предикаты дают ошибочный результат

Канонический порядок атомов (символов, буквенных чисел и строк) следующий: (целые числа и числа с плавающей точкой) $\prec$ (числа расширенной арифметики) $\prec$ (объявленные константы) $\prec$ (строки) $\prec$ (объявленные скаляры) $\prec$ (первый аргумент `orderless`) $\prec$ ... $\prec$ (последний аргумент `orderless`) $\prec$ (другие символы) $\prec$ (последний аргумент `ordergreat`) $\prec$ ... $\prec$ (первый аргумент `ordergreat`) $\prec$ (объявленные основные переменные).

Системная переменная	*По умолчанию*	*Описание*
`exptsubst`	`false`	Если `true`, то возможны подстановки в экспонентах
`inflag`	`false`	Если `true`, то при извлечении частей используется внутренняя форма выражения
`listconstvars`	`false`	Если `true`, то список, возвращаемый функцией `listofvars`, состоит из констант
`opsubst`	`true`	Если `false`, то функция `subst` не делает замены в операторах выражений
`piece`		Хранилище для последнего выражения, полученного функцией `part`

Функция `part(expr,n_1,...,n_k)` выделяет часть выражения `expr`, указанную индексами `n_1`, ..., `n_k`

```
(%i1)   z: sin(x)/y^2$ [part(z,0), part(z,1), part(z,2),
part(z,1,0), part(z,1,1), part(z,2,0), part(z,2,1), part(z,2,2)];
(%o1)       [/, sin(x), y^2, sin, x, ^, y, 2]
```

Функция `part` может быть использована для получения элемента списка, строки матрицы и т.д.

Если последний аргумент функции `part` является списком индексов, то выбирается несколько подвыражений, каждое из которых соответствует индексу списка.

```
(%i2)   part(2*a-3*b+4*c-5*d+6*e,[2,4]);
(%o2)       −5 d − 3 b
```

```
(%i3)   part(2*a-3*b+4*c-5*d+6*e,allbut(2,4));
(%o3)       6 e + 4 c + 2 a
```

```
(%i4)   args(2*a-3*b+4*c-5*d+6*e);
(%o4)       [6 e, −5 d, 4 c, −3 b, 2 a]
```

```
(%i5)   [atom(25), atom(12/7), atom(v), atom(v[1]), atom(cos(x)),
atom("--")];
(%o5)       [true,false,true,false,false,true]
```

```
(%i6)   isolate_wrt_times: true$ isolate(expand((sin(x) + sin(y) +
sin(z))^2), x);
(%t6)       2 sin(y)
(%t7)       2 sin(z)
(%t8)       sin(z)^2 + 2 sin(y) sin(z) + sin(y)^2
(%o8)       sin(x)^2+%t6 sin(x)+%t7 sin(x)+ %t8
```

```
(%i9)   subst(y,%e^x,%e^(a*x));
(%o9)       %e^(a x)
```

```
(%i10)   exptsubst: not exptsubst$ [exptsubst, subst(y, %e^x,
%e^(a*x))];
(%o10)       [true,y^a]
```

```
(%i11)   expr: z^3*cos(a[1])*b^(c+d);
(%o11)       cos(a_1) b^(d+c) z^3
```

```
(%i12)   [freeof(a[1], expr), freeof(sin, expr), freeof(cos, expr),
freeof(a[2], expr)];
(%o12)       [false,true,false,true]
```

```
(%i13)   inflag: false$ a: first(x + y)$ inflag: true$ b: first(x +
y)$ [a,b];
(%o13)       [y, x]
```

```
(%i14)   listofvars(r(x[1]+y)/z^(2+a));
(%o14)       [x_1, y, z]
```

С помощью `declare(x,mainvar)` можно установить переменную `x` главной. Опция `noun` оператора `declare` определяет действие как объект.

```
(%i15)   [op(a*b+c),nterms(a*b+c)];
(%o15)       [+,2]
```

Вызов функции `operatorp` в форме `operatorp(expr, [op_1,op_2, ...,op_n])` возвращает `true`, если хотя бы для одного элемента списка `[op_1, op_2, ..., op_n]` операция `op_k` – ведущая операция в `expr`.

```
(%i16)   r: a*b+c$ [operatorp(r,"+"), operatorp(r,"-"),
operatorp(r,["+","-","/"])];
(%o16)       [true,false,true]
```

Результат для функции `partition(expr,x)` возвращается в форме списка: 1) если `exp` – произведение, то первый элемент списка – произведение множителей в `expr`, не зависящих от `x`, второй – зависящих; 2) если `expr`– сумма, то первый элемент списка – сумма слагаемых в `expr`, не зависящих от `x`, второй – зависящих; 3) если `expr` – список, то первый элемент списка – подсписок из элементов `expr`, не зависящих от `x`, второй – зависящих.

Функция	*Описание*
`psubst(list,expr)`	Нециклические подстановки, представленные списком, в `expr`
`psubst(a,b,expr)`	Подстановка `a` вместо `b` в `expr`
`subst(list,expr)`	Циклические подстановки, представленные списком, в `expr`
`subst(a,b,expr)`	Подстановка `a` вместо `b` в `expr`
`substpart(x,expr,n_1,...,n_k)`	Подстановка `x` вместо подвыражения в `expr`, указанного индексами
`verbify(f)`	Определение объекта как действия
`opsubst(f,g,e)`	Аналог `subst` но для замены имен операторов
`opsubst(g=f,e)`	
`opsubst(list,e)`	

```
(%i17)   psubst(5,x,x^2);
(%o17)       25
```

```
(%i18)   psubst([a^2=b,b=a], sin(a^2) + sin(b));
(%o18)       sin(b) + sin(a)
```

```
(%i19)   subst([a^2=b,b=a], sin(a^2) + sin(b));
(%o19)       2 sin(a)
```

```
(%i20)   substpart("+", a*x+f(b,y), 1, 0);
(%o20)       x + f(b, y) + a
```

```
(%i21)   load("opsubst")$ opsubst(sin,log,log(a)^2);
(%o21)       sin(a)^2
```

4. СРЕДСТВА ПРОГРАММИРОВАНИЯ И ИХ ИСПОЛЬЗОВАНИЕ

4.1. Элементы программирования. Вывод на экран

Как известно, программы на различных языках программирования состоят из разнообразных конструкций, принимающих форму операторов и команд, т.е. описаний, установок, обращений к функциям, присваиваний, специальных структур типа ветвления и цикла и т.д. Формы программ различны, но, как правило, в них присутствуют следующие части: получение входных данных и параметров, расчетная часть и вывод результатов. Большая часть фрагментов этих частей записываются в виде тех или иных функций, которые оформляются в виде блоков.

Линейной называется *программа*, в которой выполнение операторов и команд, а также вызовы функций всегда происходят последовательно в том порядке, в котором они записаны. При этом в линейной программе могут быть операторы присваивания, операторы ввода/вывода данных, обращения к встроенным и пользовательским функциям, но не должно быть структур ветвления, структур цикла и иных средств, нарушающих естественный порядок выполнения действий. Как правило, к линейным программам приводят задачи, в которых необходимо выполнить обработку данных по тем или иным формулам.

Функция	*Описание*
`print(expr1,...,exprn)`	Вывод выражений `expr1`, ..., `exprn` в строку
`disp(expr1,...,exprn)`	Вывод выражений `expr1`, ..., `exprn` в одну колонку
`display(expr1,...,exprn)`	Вывод выражений `expr1`, ..., `exprn` в одну колонку с именами

Функция `print(expr1,...,exprn)` является основной и самой удобной для вывода на печать, печатает значения всех своих аргументов `expr1`, ..., `exprn` в одну строку.

```
(%i1)   a: sin(x)$ b: log(y)$
```

```
(%i2)   print(a,b)$
        sin(x) log(y)
```

Функция `disp(expr1,...,exprn)` печатает значения своих аргументов, причем каждое значение печатается в отдельной строке.

```
(%i3)   disp(a,b);
                          sin(x)
                          log(y)
(%o3)   done
```

Функция `display(expr1,...,exprn)` выводит значения своих аргументов вместе с их именем, каждое в отдельной строке.

```
(%i4)   disp(a,b);
                          a = sin(x)
                          b = log(y)
(%o4)   done
```

4.2. Составные операции. Блоки

Составной оператор – это конструкция языка программирования, состоящая из нескольких команд (операторов) языка программирования, но участвующая в программе в качестве единого оператора.

Как в условных выражениях, так и в циклах (см. 4.4, 4.5) вместо простых операторов можно писать составные операторы, т.е. блоки. Стандартный блок имеет вид:

```
(%i1)   block([r,s,t], r:1, s:r+1, t:s+1, x:t, t*t);
(%o1)                          9
```

Сначала идет список *локальных переменных* блока (*глобальные переменные* с теми же именами никак не связаны с этими локальными переменными). Список локальных переменных может быть пустым. Далее идет набор операторов. Локальные переменные можно описывать с помощью оператора `local(v_1,...,v_n)`.

Упрощенный блок имеет вид:

```
(%i2)   (x:1, x: x+2, a: x);
(%o2)                          3
```

Обычно в циклах и в условных выражениях применяют именно эту форму блока.

Значением блока является значение последнего из его операторов. Внутри данного блока допускаются оператор перехода на метку (см. подраздел 4.3) и оператор `return`. Этот оператор прекращает выполнение текущего блока и возвращает свой аргумент в качестве значения блока:

```
(%i3)   block([], x:2, x: x*x, return(x), x: x*x);
(%o3)       4
```

В отсутствие оператора перехода на метку и ветвлений (см. подраздел 4.4) операторы в блоке выполняются последовательно. Оператор go выполняет переход на метку, расположенную в этом же блоке:

```
(%i4)   block([a,b,s], a: 0, b: 1, s: 1, metka, a: a+1, b: b/a, s:
s+b, if a=40 then return(s), go(metka) )$ float(%);
(%o4)       2.718281828459045
```

В этом блоке реализован цикл, который завершается по достижении переменной цикла значения 40.

4.3. Функции пользователя. Возврат результатов и передача управления

Функция	*Описание*
:=	Оператор определении функции

Функциональность системы Maxima можно расширять своими функциями с помощью оператора определения функции :=. Имена функций формируются способом, похожим на выбор имен переменных, но сопровождаются круглыми скобками (...), которые содержат разделенный запятыми список ее аргументов (которые, в свою очередь, сами являются переменными). Правая часть оператора назначения функции := (тело функции, обычно формируемое в виде блока) при определении функции не вычисляется.

Сначала указывается название функции, в круглых скобках перечисляются названия аргументов, а после знаков := (двоеточие и равно) следует описание функции. После задания пользовательская функция вызывается точно так, как и встроенные функции Maxima. Необходимо помнить, что имена этих функций не рекомендуется использовать для функций пользователя.

Возможны следующие варианты:

– f(x_1,...,x_n):=expr определяет функцию с именем f с параметрами x_1, ..., x_n и телом функции expr. Оператор := никогда не вычисляет тело функции (если явно не получено значение с помощью ''). Тело функции рассчитывается каждый раз, когда вызывается функция;

– `f[x_1,...,x_n]:=expr` вводит так называемую функцию с памятью. Ее тело функции вычисляется только один раз для каждого отдельного значения ее параметров. Это значение возвращается без расчета тела функции всякий раз, если параметры имеют те же значения снова. Функция такого типа также называется "функцией массива";

– `f[x_1,...,x_n](y_1,...,y_m):=expr` – это особый случай функции с памятью. `f[x_1,...,x_n]` – функция с памятью, которая возвращает лямбда-выражение с аргументами `y_1`, ..., `y_m`. Тело функции вычисляется один раз для каждого отдельного значения `x_1`, ..., `x_n`, и тело лямбда-выражения является этим значением.

Когда последний или единственный параметр функции `x_n` является списком из одного элемента, функция, определенная с помощью `:=`, допускает переменное число параметров. Фактические параметры присваиваются один за другим формальным параметрам `x_1`, ..., `x_(n-1)`, а любые последующие фактические параметры, если они присутствуют, присваиваются `x_n` в виде списка.

Все определения функций отображаются в одном и том же пространстве имен; определение функции `f` внутри другой функции `g` не ограничивает автоматически область действия `f` только `g`. Однако задание `local(f)` делает определение функции `f` видимым только в пределах блока или другого составного выражения, в котором появляется `local`.

Если какой-то формальный параметр `x_k` появляется с символом ', то функция, определенная с помощью `:=`, не находит значение соответствующего фактического параметра. В противном случае все фактические параметры вычисляются.

```
(%i1)   r: cos(y)-sin(x)$ F1(x, y):=r$ F2(x, y):=''r$
[F1(u,v),F2(u,v)];
(%o1)        [cos(y) − sin(x), cos(v) − sin(u)]
```

```
(%i2)   Summa([L]):=apply("+",L)$ Summa(a,b,c,1,2,3);
(%o2)        c + b + a + 6
```

```
(%i3)   myfunction(x,y):=if (x>y) then x^2 else y^3;
(%o3)        myfunction(x, y) := if x > y then x^2 else y^3
```

```
(%i4)   [myfunction(2,1), myfunction(1.0,2.0)];
(%o4)        [4, 8.0]
```

В следующем примере определяются функции `sqr` для расчета квадрата заданного выражения и `sindeg` для определения синуса, где угол

задается в градусах. Отметим, что тело функции содержит выражение `x`, которое не вычисляется. В случае обычного выражения `x` было бы заменено значением 5, которое было определено в предыдущем примере.

```
(%i5)   sqr(x):= x^2;
(%o5)       sqr(x) := x^2
```

```
(%i6)   sqr(4);
(%o6)       16
```

```
(%i7)   sindeg(x):= sin(x*%pi/180);
(%o7)       sindeg(x) := sin(x %pi/180)
```

```
(%i8)   sindeg(45);
(%o8)       1/sqrt(2)
```

Конечно, можно определять функции двух или более переменных.

```
(%i9)   norm(x,y):= sqrt(x^2+y^2);
(%o9)       norm(x, y) := sqrt(x^2 + y^2)
```

```
(%i10)   norm(3,4);
(%o10)      5
```

Системные переменные `values` и `functions` содержат список пользовательских переменных и функций соответственно.

```
(%i11)   values;
(%o11)      [x, y]
```

```
(%i12)   functions;
(%o12)      [sindeg(x), sqr(x), norm(x, y)]
```

Заметим, что и переменные, и функции остаются неизменными до тех, пока сеанс работы с CAB Maxima не будет закрыт. Иногда бывает удобно удалить их. Это можно сделать с помощью функции `kill`.

```
(%i13)   kill(x);
(%o13)      done
```

```
(%i14)   values;
(%o14)      [y]
```

```
(%i15)  functions;
```
(%o15) $[sindeg(x), sqr(x), norm(x, y)]$

```
(%i16)  kill(all);
```
(%o16) *done*

```
(%i17)  values;
```
(%o17) []

```
(%i18)  functions;
```
(%o18) []

Заметим, что нельзя использовать символ = для определения переменных и функций. Этот символ используется только для определения уравнений или проверки на равенство выражений.

```
(%i19)  x=3;
```
(%o19) $x = 3$

```
(%i20)  x;
```
(%o20) x

Не следует использовать для функций названия, зарезервированные для встроенных функций системы Maxima. Для создания функций используется также встроенная функция `define`, которая позволяет преобразовать выражение в функцию.

Функция	*Описание*
`define(f(x1,...,xn),expr)`	Неявное определение ординарной функции
`define(f[x1,...,xn],expr)`	Неявное определение массива
`define(funmake(f,[x1,... xn]),expr)`	Неявное определение ординарной функции и ее вызов
`define(arraymake(f,[x1,...,xn]),expr)`	Неявное определение массива и его вызов
`define(ev(expr1),expr2)`	Неявное определение функции
`remfunction(f_1,...,f_n)`	Стирание описаний функций
`remfunction(all)`	

```
(%i21)  dfun: cos(y)-sin(x)$ define(g(x,y), dfun)$ g(%pi/2,%pi/4);
```
(%o21) $\frac{1}{\sqrt{2}} - 1$

```
(%i22)   define(ev(fmy(x,y)), sin(x)-cos(y))$ fmy(%pi/2,%pi/4);
(%o22)       1 - 1/sqrt(2)
```

Возврат из функции пользователя может осуществляться с помощью команды `return(expr)`. Эта команда передает управление на уровень выше, т.е. произойдет прерывание выполнения соответствующего блока, цикла или внутреннего цикла. При этом возвращается выражение `expr`.

4.4. Условные операторы. Ветвление. Использование меток

Нарушать линейную последовательность выполнения операторов и команд, а также вызова функций и программы можно с помощью специальных структур ветвления и цикла. Ниже представлены некоторые возможные разновидности структуры ветвления, которые реализуемы на входном языке системы Maxima:

Форма I. `if` <условие 1> `then` <выражение 1>

Форма II. `if` <условие 1> `then` <выражение 1> `else` <выражение 2>

Форма III. `if` <условие 1> `then` <выражение 1> `elseif` <условие 2> `then` <выражение 2> ... `elseif` <условие N> `then` <выражение N>

Форма IV. `if` <условие 1> `then` <выражение 1> `elseif` <условие 2> `then` <выражение 2> ... `elseif` <условие N> `then` <выражение N> `else` <выражение N+1>

Оператор	*Описание*
`if ... then ... else ...`	Условный оператор

II – основная форма условного оператора. Если логическое значение выражения <условие1> – `true`, то вычисляется выражение <выражение 1>, иначе – выражение <выражение 2>.

Система Maxima позволяет использовать различные формы оператора `if`, например:

`if` <условие 1> `then` <выражение 1> `elseif` <условие 2> `then` <выражение 2> `else` <выражение 3>

Здесь если логическое значение выражения <условие 1> – `true`, то выполняется <выражение 1>, иначе проверяется <условие 2>. Если его логическое значение – `true`, то рассчитывается <выражение 2>, иначе <выражение 3>.

Вычисляемые <выражение 1>, <выражение 2>, <выражение 3> – произвольные выражения, допускаемые в языке системы Maxima (в т.ч.

вложенные операторы `if`). <условие 1> и <условие 2> – явные или неявные логические выражения, значениями которых являются константы `true` или `false`.

Существует много функций, таких как абсолютное значение (действительного) x, которые определяются различными выражениями на непересекающихся подынтервалах их области определения, например:

$$\mathrm{abs}(x) = \begin{cases} x, & \text{если } x \geqslant 0, \\ -x, & \text{иначе.} \end{cases}$$

Такие функции можно реализовать в пакете Maxima, если воспользоваться условным оператором:

`if <условие> then <значение`$_1$`> else <значение`$_2$`>`.

Здесь `<условие>` – это логическое утверждение, сформированное с помощью логических операторов (см. выше в этом подразделе). Если это утверждение истинно, то результатом выполнения условного оператора будет `<значение`$_1$`>`, а если ложно, то `<значение`$_2$`>`. Если же `is(<условие>)` дает `unknown`, то результат вычисления не определен.

```
(%i1)   fabs(x):= if (x>=0) then x else -x;
(%o1)       if x >= 0 then x else - x
```

Внутри блоков можно использовать переходы внутри блока. Для этого используется функция `go(lab)`, где `lab` – метка перехода. В данном случае слово "метка" означает отнюдь не метку типа `%i5` или `%o7`. Меткой может быть произвольный идентификатор. Переход осуществляется на то выражение, которое записано после запятой, следующей метки. Метка должна стоять ранее команды `go`. Переходы внутри цикла, извне цикла в цикл и из одного блока в другой невозможны.

Заметим, что цикл сам по себе является блоком. Так что прервать выполнение циклов (особенно вложенных циклов) с помощью оператора `go` невозможно, т.к. оператор `go` и метка окажутся в разных блоках. То же самое относится к оператору `return`.

Если цикл, расположенный внутри блока, содержит команду `return` то при исполнении этой команды произойдет выход из цикла, но не выход из блока. Если необходимо выйти из нескольких вложенных блоков сразу (или нескольких блоков и циклов сразу) и при этом возвратить некоторое значение, то следует применять блок `catch` в сочетании с командой `throw`:

```
(%i2)   catch(block([], a:1, a:a+1, throw(a), a:a+7),a:a+9);
(%o2)       2
```

```
(%i3)   a;
(%o3)       2
```

```
(%i4)   catch(block([], for i:1 thru 15 do if i=2 then throw(555)),
777);
(%o4)       555
```

В данном блоке выполнение цикла завершается, как только значение `i` достигает 2. Возвращаемое блоком `catch` значение равно 555.

```
(%i5)   catch(block([],for i:1 thru 15 do if i=52 then
throw(555)),777);
(%o5)       777
```

В данном блоке цикл выполняется полностью и возвращаемое блоком `catch` значение равно 777 (условия выхода из цикла при помощи `throw` не достигаются).

Оператор `throw` – аналог оператора `return`, но он обрывает не текущий блок, а все вложенные блоки вплоть до первого встретившегося блока `catch`, т.е. обеспечивает нелокальную передачу данных и управления.

4.5. Операторы цикла

Если структуры ветвления дают возможность нарушать естественную последовательность выполнения команд, то структуры цикла позволяют организовывать многократное выполнение отдельных групп команд. Ниже приведены некоторые возможные виды циклических структур цикла в языке Maxima:

Форма I. Цикл типа пересчета

`for` <переменная цикла>: <начальное значение> `thru` <конечное значение> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `step` <шаг> `thru` <конечное значение> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `next` <функция переменной цикла> `thru` <конечное значение> `do` <тело цикла>

Форма II. Цикл типа "пока"

`while` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `while` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `step` <шаг> `while` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> tpmthru <конечное значение> `while` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `step` <шаг> `thru` <конечное значение> `while` <условие> `do` <тело цикла>

Форма III. Цикл типа "пока не"

`unless` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `unless` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `thru` <конечное значение> `unless` <условие> `do` <тело цикла>

`for` <переменная цикла>: <начальное значение> `step` <шаг> `thru` <конечное значение> `while` <условие> `do` <тело цикла>

Форма IV. Цикл по списку

`for` <переменная цикла> `in` <список> `do` <тело цикла>

`for` <переменная цикла> `in` <список> `while` <условие> `do` <тело цикла>

`for` <переменная цикла> `in` <список> `unless` <условие> `do` <тело цикла>

Все эти циклы имеют общее название – цикл типа `do`. <шаг> по умолчанию равен 1. Ключевые слова `thru`, `while`, `unless` указывают на способ завершения цикла: по достижении конечного значения переменной цикла <конечное значение>; пока выполняется условие <условие>; пока не будет достигнуто условие <условие>. В циклах форм I, II и III разделитель `:` может быть заменен зарезервированным словом `from`. <начальное значение>, <конечное значение> и <тело цикла> могут быть произвольными выражениями. Переменная цикла по завершении цикла предполагается положительной (при этом начальное значение может быть и отрицательным). Все выражения и условия завершения вычисляются на каждом шаге цикла. Поэтому их сложность влияет на время выполнения цикла.

При нормальном завершении цикла возвращаемая величина – атом `done`. Принудительный выход из цикла осуществляется при помощи оператора `return`, который может возвращать произвольное значение.

Переменная цикла – локальная переменная внутри цикла. Поэтому ее изменение в цикле не влияет на контекст (даже при наличии вне цикла переменной с тем же именем).

Циклы с ключевыми словами `for`, `unless`, `while` и `in` могут вкладываться друг в друга. Такие циклы называются вложенными.

Существуют также неявные циклы, например, связанные с функциями суммирования `sum` и умножения `prod`.

```
(%i1)   s: 0$ for k in [a,b,c,d] while k#d do (s: s+f[k])$ s;
(%o1)       f_c + f_b + f_a
```

4.6. Рекурсия

В прикладной математике рекурсия используется: 1) в качестве одного из специальных способов решений определенного класса задач; 2) для формализации понятия алгоритма [4]. В программировании рекурсия применяется для компактной реализации сложных численных и символьных процедур с эффектами самовызова.

Язык системы Maxima позволяет строить рекурсивные функции пользователя, а вычислительное ядро имеет возможность обрабатывать вызовы таких функций.

Следующий пример показывает расчет центральных моментов различных порядков случайной величины, имеющей нормальное распределение с дисперсией σ:

```
(%i1)   gausscentmoms(s,n):=if oddp(n) then 0 elseif n=2 then s^2
else (n-1)*s^2*gausscentmoms(s,n-2)$
```

```
(%i2)   [gausscentmoms(sigma,5), gausscentmoms(sigma,2),
gausscentmoms(sigma,6), gausscentmoms(sigma,12)];
(%o2)       [0, σ^2, 15 σ^6, 10395 σ^12]
```

4.7. Maxima и Lisp

Вычислительная часть системы Maxima написана на языке Lisp и имеет легкий доступ к Lisp-функциям и переменным из среды системы Maxima и наоборот.

Объекты Lisp и Maxima можно отличить, если учесть соглашение о формировании имен. Имя Lisp-объекта начинается со знака `$`. Этот объект соответствует Maxima-объекту с тем же именем, но без знака `$`. Maxima-объект, имя которого начинается со знака `?`, соответствует Lisp-объекту с тем же именем, но без знака `?`.

Знаки "-", "*" и другие специальные символы в именах Lisp-объектов нужно предварять символами "\" там, где они появляются в операторах языка Maxima.

Программный Lisp-код может быть выполнен внутри сессии Maxima. Отдельный фрагмент Lisp-кода, содержащий одну команду или более, может быть выполнен специальной командой `:lisp`. Например:

```
(%i1)   :lisp(list 'a 'b 'c)
         (A B C)
```

Конструкция `:lisp` может появиться в командной строке или в файле, которые обрабатываются командами `batch` или `demo`, но не в файле, который является параметром команд `load`, `batchload`, `translate_file` или `compile_file`.

Функция `to_lisp()` открывает интерактивную Lisp-сессию. Команда `(to-maxima)` закрывает эту сессию и возвращает управление в среду Maxima.

Lisp-функции и переменные, которые должны быть видимы в среде Maxima как функции и переменные с обычными именами (без специальных расширений), должны иметь в начале Lisp-имени знак доллара `$`.

```
(%i1)   :lisp(msetq $d #$[a, b]$)
         ((MLIST SIMP) $A $B)
```

```
(%i1)   d;
(%o1)       [a, b]
```

Заметим, что после команды `:list(...)` ни один из знаков `;` или `$` не требуется. Кроме того, эта команда не меняет счетчик ячеек ввода.

4.8. Средства отладки

Отладкой программы называется проверка правильности работы всей программы и всех ее частей. Обычно отладка производится на специально подготовленных наборах данных, для которых известно, корректно ли программа их обрабатывает. На этапе отладки исправляются синтаксические, семантические и логические ошибки, а также ошибки времени исполнения.

Символ	*Описание*
`:h`	Список команд отладчика
`:help`	То же самое
`:br`	Точка останова
`:n`	Переход к следующей строке при пошаговой отладке
`:next`	То же самое
`:bt`	Список кадров стека
`:backtrace`	То же самое
`:r`	Прекращение отладки
`:resume`	То же самое
`:continue`	Продолжение вычисления
`:delete`	Удаление точек останова
`:disable`	Отключение точек останова
`:enable`	Включение точек останова
`:frame`	Печать заданного уровня стека
`:quit`	Окончание отладки на заданном уровне
`timer(f_1,...,f_n)`	Запись функций в список тех, по которым собирается статистика затрат времени
`untimer(f_1,...,f_n)`	Исключение функций из списка тех, по которым собирается статистика затрат времени
`timer_inf(f_1,...,f_n)`	Статистика затрат времени по функциям
`trace(f_1,...,f_n)`	Вывод отладочной информации по функциям
`trace_options(f,opt1,...,optn)`	Задание отладочных опций по функции `f`
`untrace(f_1,...,f_n)`	Прекращение вывода отладочной информации по функциям
`backtrace()` `backtrace(n)`	Трассировка вызовов фцнкций
`break(ex1,...,exn)`	Вывод на экран выражений `ex1`, ..., `exn`, остановка для просмотра и исправления значений переменных и выражений
`exit`	Продолжение расчетов после функции `break`
`error(ex1,...,exn)`	Вывод на экран выражений `ex1`, ..., `exn`, инициирование сообщения об ошибке и прекращение расчетов
`errcatch(ex1,...,exn)`	Вычисление выражений `ex1`, ..., `exn`„ при отсутствии ошибок возвращение `exn`. Иначе пустой список
`errormsg()`	Повторный вывод на экран последнего сообщения об ошибке

Символ	*Описание*
`warning(ex1,...,exn)`	Вывод на экран выражений `ex1`, ..., `exn`, инициирование сообщения о возможной ошибке

Maxima имеет встроенный отладчик на уровне исходного текста. Пользователь может установить точку останова на функцию и шагать далее строка за строкой. Может быть проанализирован стек вызовов функций вместе с переменными, связанными с соответствующими уровнями.

Команда `:help` (или `:h`) показывает список команд отладчика. В общем случае для команд можно использовать сокращения, если сокращение уникально. Если оно не уникально, то будут перечислены альтернативные варианты. В отладчике пользователь также может использовать любые обычные функции Maxima для проверки, определения и управления переменными и выражениями.

Точка останова устанавливается командой `:br` в строке приглашения Maxima. В отладчике пользователь может перемещаться по одной строке за раз, используя команду `:n` (`next`). Команда `:bt` (`backtrace`) показывает список кадров стека. Команда `:r` (`resume`) прекращает работу отладчика и передает управление монитору системы.

Системная переменная	*По умолчанию*	*Описание*
`debugmode`	`false`	Если `true`, то при появлении ошибки запускается отладчик
`refcheck`	`false`	Если `true`, то вывод сообщения при первом появлении заданной переменной
`setcheck`	`false`	Если установлена, то вывод сообщения при появлении переменной в операторе присваивания
`setcheckbreak`	`false`	Если `true`, то вывод приглашения точки останова, как только функции из списка `setcheck` присваивается новое значение
`setval`		Значение, которое присваивается переменной, когда возникает состояние останова `setcheckbreak`
`timer_devalue`	`false`	Если `true`, то вычитание времени, затраченного другими функциями

4.9. Операторы ввода/вывода. Файлы. Средства работы с файловой системой

Файлы – это области на произвольных устройствах хранения, которые содержат данные в различных форматах.

Файлы на HDD сгруппированы в "каталоги" древовидной структуры. Команды, которые работают с файлами:

```
appendfile        batch               batchload
closefile         file_output_append  filename_merge
file_search       file_search_maxima  file_search_lisp
file_search_demo  file_search_usage   file_search_tests
file_type         file_type_lisp      file_type_maxima
load              load_pathname       loadfile
loadprint         pathname_directory  pathname_name
pathname_type     printfile           save
stringout         with_stdout         writefile
```

Когда имя файла передается таким функциям типа `plot2d`, `save` или `writefile`, а имя файла не содержит пути, Maxima сохраняет файл в текущем рабочем каталоге. Текущий рабочий каталог зависит от операционной системы и от установки.

Функция	*Описание*
`appendfile(filename)`	Пополнение текстового файла
`writefile(filename)`	Запись в текстовый файл
`closefile()`	Закрывает текстовый файл

Функция `writefile` записывает содержание сеанса Maxima в файл `filename`. Все результаты интерактивного взаимодействия между пользователем и системой записываются в этот файл, а также отображаются на консоли.

Поскольку содержание сеанса записывается в формате вывода на консоль, оно не может быть вновь загружено в системную среду. Чтобы создать файл, содержащий выражения, которые могут быть введены, нужно использовать функции `save` и `stringout`. Функция `save` сохраняет выражения в форме Lisp, а `stringout` – в форме Maxima.

Эффект выполнения функции `writefile`, когда файл с именем `filename` уже существует, зависит от базовой реализации Lisp, а именно файл может быть очищен перед записью или пополнен.

Функция `appendfile` добавляет текстовые данные в файл с именем `filename`. Функция `appendfile` делает то же самое, что и `writefile`, за исключением того, что если файл с именем `filename` существует, то он всегда пополняется.

Функция `closefile` закрывает текстовый файл, открытый для записи функциями `appendfile` и `writefile`.

Функция	*Описание*
`save(filename,name_1, name_2,name_3,...)`	Сохранение содержимого переменных
`save(filename,values, functions,labels,...)`	Сохранение объектов
`save(filename,[m,n])`	Сохранение меток
`save(filename,all)`	Сохранения состояния среды
`save(filename,name_1=expr_1, name_2=expr_2,...)`	Сохранение выражений с именами

Функция `save` сохраняет текущие значения переменных `name_1`, `name_2`, `name_3`, ... в файле с именем `filename`. Аргументами являются имена переменных, функций или других объектов. Если имя не имеет значения или функции, связанной с ним, то оно игнорируется. Функция `save` возвращает `filename`.

Функция `save` хранит данные в виде Lisp-выражений. Если имя файла `filename` заканчивается на `.lisp`, то данные, сохраненные с помощью `save`, могут быть восстановлены с помощью функции `load(filename)`.

Глобальный флаг `file_output_append` определяет, пополняет ли `save` выходной файл или усекает его. Когда `file_output_append` имеет значение `true`, `save` пополняет выходной файл. В противном случае `save` перед записью усекает выходной файл. Перед записью команда `save` создает файл, если он еще не существует.

Специальная форма `save(filename, values, functions, labels, ...)` сохраняет `values`, `functions`, `labels` и т.д. Имена могут быть любыми, заданными переменной `infolists`. `values` включают в себя все пользовательские переменные.

Специальная форма `save(filename,[m, n])` сохраняет значения меток ввода и вывода с `m` до `n`. Заметим, что `m` и `n` должны быть целыми числами. Метки ввода и вывода также могут быть сохранены одна за другой, например, `save("foo.1", %i42, %o42)`. `save(filename,labels)` сохраняет все метки ввода и вывода. Когда сохраненные метки восстанавливаются, они перезаписывают существующие метки.

Еще одна специальная форма функции `save(filename, name_1 = expr_1, name_2 = expr_2, ...)` сохраняет значения выражений `expr_1`, `expr_2`, ..., с именами `name_1`, `name_2`, ... Полезно применять эту форму к меткам ввода и вывода, например, `save("foo.1aa = %o88)`. Правой частью равенства в этой форме может быть любое выражение, которое может быть вычислено. Эта форма не добавляет новые имена в текущую среду Maxima, а только сохраняет их в файле с именем `filename`.

Эти специальные формы и общая форма `save` могут быть перемешаны. Например, `save(filename, aa, bb, cc=42, functions, [11, 17])`.

Специальная форма `save(filename, all)` сохраняет текущее состояние системы Maxima. Это включает в себя все пользовательские переменные, функции, массивы и т.д., а также некоторые автоматически определяемые элементы. Эти элементы включают системные переменные, такие как `file_search_maxima` или `showtime`, если им пользователем были назначены новые значения.

Функция	*Описание*
`load(filename)`	Загрузка и вычисление выражений в файле
`loadfile(filename)`	То же самое
`batch(filename)`	Считывание выражений языка Maxima из файла и их расчет
`batch(filename,option)`	То же самое
`batchload(filename)`	То же самое

Функция `load(filename)` загружает файл с выражениями языка системы Maxima, вычисляет их (переменные, функции и др.) и передает результаты в среду системы. Значение любого существующего объекта перекрывается значением, введенным из файла. Чтобы найти файл, функция `load` вызывает функцию `file_search` с опциями `file_search_maxima` и `file_search_lisp` в качестве каталогов для поиска. Если загрузка прошла успешно, то возвращается имя файла. В противном случае функция `load` выдает сообщение об ошибке.

Функция `load` работает одинаково хорошо для кода на языке Lisp и для кода на языке Maxima. Файлы, созданные с помощью функций `save`, `translate_file` и `compile_file`, которые создают код на языке Lisp, и функция `stringout`, генерирующая код на языке Maxima, могут обрабатываться функцией `load`, которая вызывает функцию `loadfile` для загрузки файлов Lisp и `batchload` для загрузки файлов Maxima.

Функция `load` не распознает конструкции `:lisp` в Maxima-файлах. Также следует отметить, что структуры будут считываться как структуры, только если они были определены с помощью оператора `defstruct` перед вызовом команды `load`.

Функция `loadfile(filename)` загружает файл с выражениями языка системы Maxima, вычисляет их (переменные, функции и др.) и передает результаты в среду системы. `loadfile` не вызывает `file_search`, поэтому `filename` должно включать расширение файла и настолько длинный путь, насколько это необходимо для поиска файла.

Функция `loadfile` может обрабатывать файлы, созданные с помощью `save`, `translate_file` и `compile_file`. Может оказаться, что более удобно использовать `load` вместо `loadfile`.

Функция batch(filename) считывает выражения языка Maxima из файла с именем filename и вычисляет их. batch ищет файл с именем filename в каталоге file_search_maxima.

batch(filename, demo) – то же самое, что и demo(filename). В этом случае batch ищет файл с именем filename в каталоге file_search_demo.

Функция batch(filename,test) похожа на run_testsuite с параметром display_all = true. Для этого случая batch ищет файл с именем filename в каталоге file_search_maxima, а не в списке file_search_tests, как run_testsuite. Кроме того, run_testsuite запускает тесты, которые находятся в списке testsuite_files. С помощью batch можно запустить любой файл в тестовом режиме, который можно найти в списке file_search_maxima, что полезно при написании тестовых файлов.

Файл с именем filename содержит последовательность выражений Maxima, каждое из которых заканчивается на ; или $. Специальная переменная % и функция %th ссылаются на предыдущие результаты в файле. Файл может содержать конструкции :lisp. Пробелы, знаки табуляции и переводы на новые строки в файле игнорируются. Необходимый входной файл может быть создан текстовым редактором или функцией stringout.

batch читает каждое входное выражение из файла с именем filename, отображает входные данные на консоль, вычисляет соответствующее выходное выражение и отображает его. Имена ввода назначаются выражениям ввода, а имена вывода – выражениям вывода. batch рассчитывает каждое входное выражение в файле, если нет ошибки. Если запрашивается пользовательский ввод (например, с помощью asksign или askinteger), batch делает паузу для выполнения необходимого ввода и затем продолжает выполняться.

batch возвращает путь к файлу в виде строки, когда вызывается без второго аргумента или с опцией demo. Если вызывается с опцией test, то возвращается пустой список или список с filename и номерами задач, которые не были решены.

Функция batchload(filename) считывает выражения языка Maxima из файла с именем filename и вычисляет их, не отображая входные или выходные выражения и не присваивая переменным для выходных выражений. Однако результат вывода командой print или describe отображается. Файл не может включать конструкции :lisp. batchload возвращает путь к файлу в виде строки.

Системные переменные file_search_maxima, file_search_lisp, file_search_demo, file_search_tests и переменные задают списки каталогов, которые используются для поиска соответствующих файлов функ-

циями load, demo и некоторыми другими. Значения этих переменных по умолчанию есть имена различных каталогов в установке системы Maxima. Пользователь может изменять эти переменные либо для замены значений по умолчанию, либо для добавления дополнительных каталогов.

Системная переменная	*По умолчанию*	*Описание*
load_pathname	false	Когда файл загружается с помощью функций load, loadfile или batchload, переменной присваивается путь к файлу
file_output_append	false	Управление обрезкой или пополнением выходного файла (true)
file_search_maxima	[mac,mc,demo, dem,dm1,dm2, dm3,dmt wxm]	Содержит список директорий для поиска файлов с выражениями Maxima
file_search_lisp	[l,lsp,lisp]	Содержит список директорий для поиска файлов с выражениями Lisp
file_search_demo		Содержит список директорий для поиска файлов с демо-примерами
file_search_tests		Содержит список директорий для поиска файлов с тестами
loadprint	true	Переключатель вывода сообщений при загрузке файла

Функция	*Описание*
filename_merge (path,filename)	Построение полного пути из path и filename
file_search(filename)	Поиск пути к файлу в каталогах по умолчанию. Результат – строка, если файл найден, иначе false
file_search(filename, pathlist)	То же, но в заданном списке каталогов
file_type(filename)	Тип файла
directory(path)	Список файлов и каталогов
printfile(path)	Печать содержимого файла

Функция file_type(filename) возвращает предположение о содержимом файла с именем filename, основываясь на расширении имени файла. filename не обязательно должно ссылаться на фактический файл. Функция file_type не делает никаких попыток открыть файл и проверить его содержимое.

Возвращаемое значение будет одним из символов object, lisp или maxima. Если расширение соответствует одному из значений в каталогах,

определяемых переменной file_type_maxima, то функция file_type возвращает символ maxima. Если расширение соответствует одному из значений в каталогах, определяемых переменной file_type_lisp, то функция file_type возвращает символ lisp. Если ничего из вышеперечисленного не обнаруживается, то функция file_type возвращает символ object.

```
(%i1)   map('file_type, ["test.lisp", "test.mac", "test.dem",
"test.txt"]);
(%o1)        [lisp,maxima,maxima,object]
```

Функция directory(path) возвращает список файлов и каталогов, найденных, используя ссылку path.

Функция printfile(path) выводит содержимое файла, указанного переменой path на консоль. path может быть строкой или символом. Если это символ, он преобразуется в строку.

Если path указывает на файл, который доступен из текущего рабочего каталога, то этот файл выводится на консоль. В противном случае printfile пытается найти файл, добавляя path к каждому из элементов file_search_usage с помощью filename_merge.

Функция printfile возвращает path, если он указывает на существующий файл, иначе результат расширенного имени файла после успешного поиска.

Функция	*Описание*
stringout(filename,name_1, name_2,name_3,...) stringout(filename,[m,n]) stringout(filename,input) stringout(filename,functions) stringout(filename,values)	Запись выражения в файл в той же форме, пригодной для ввода
with_stdout(f,expr_1, expr_2, expr_3, ...) with_stdout(s,expr_1, expr_2,expr_3,...)	Вычисление и запись выражений в файл или в поток

Функция stringout записывает выражения в файл в той же форме, в которой выражения набираются для ввода. Затем файл можно использовать в качестве входных данных для команд batch или demo и его можно редактировать. stringout может запускаться при работе функции writefile.

Глобальный флаг file_output_append устанавливает, будет ли добавлять stringout данные в выходной файл или будет усекать его. string out создает файл, если он еще не существует.

Общая форма функции stringout записывает значения одного или нескольких выражений в выходной файл. Заметим, что если выражение является переменной, записывается только значение переменной, а не имя переменной. В качестве полезного особого случая выражения могут быть входными метками ввода (%i1, %i2, %i3, ...) или метками вывода (%o1, %o2, %o3, ...).

Если системная переменная grind равна true, то функция stringout форматирует вывод, используя формат функции grind. В противном случае используется формат string.

Специальная форма stringout(filename,[m, n]) сохраняет значения меток ввода и вывода с m до n; форма stringout(filename,input) – все метки ввода в файл; форма stringout(filename,functions) – все определенные пользователем функции (отмеченные в списке functions) в файл; форма stringout(filename,values) – все определенные пользователем переменные (отмеченные в списке functions) в файл. В последнем случае каждая переменная выводится как оператор присваивания с именем переменной, двоеточием и ее значением. Заметим, что общая форма функции stringout не выводит переменные в форме операторов присваивания.

Функция with_stdout вычисляет выражения expr_1, expr_2, expr_3, ... и записывает любой сгенерированный таким образом вывод в файл f или в выходной поток s. Результаты расчетов не выводятся. Вывод может быть сформирован с помощью функций print, display, grind и других функций.

Глобальный флаг file_output_append определяет, будет ли добавлять данные with_stdout к файлу или будет усекать выходной файл f. Когда file_output_append имеет значение true, вывод функции with_stdout добавляется к выходному файлу. В противном случае with_stdout усекает выходной файл. В любом случае with_stdout создает файл, если он еще не существует. Функция with_stdout возвращает значение своего последнего аргумента.

Пакет numericalio содержит ряд функций для чтения и записи файлов и stream-данных. Эти функции, предназначенные для простого текстового ввода, дают возможность считывать и записывать элементы данных, которые являются атомами: целые числа, обычные и длинные числа с плавающей точкой, символы и строки, т.е. рациональные или комплексные числа исключены. Кроме того, в составе пакета имеются функции ввода/вывода бинарных данных (assume_external_byte_order, openw_binary, openr_binary, opena_binary, read_binary_array, read_binary_matrix, read_binary_list, write_binary_data). Ниже они не рассматрива-

ются, но если есть необходимость ими воспользоваться, то см. полную документацию по системе Maxima.

Функции пакета включают средства ввода (записи) данных в существующие структурные единицы типа списка, матрицы или массива. В противном случае соответствующая функция пакета может попытаться угадать структуру объекта для хранения данных и использовать для записи ввода этот объект.

Формат входных и выходных должен быть тем же самым, что и в пакетных файлах системы Maxima или при интерактивном вводе. Никакие символы продолжения строк не распознаются.

Функции для простого текстового ввода и вывода простого текста имеют необязательный аргумент `separator_flag`, который указывает символ, разделяющий данные. Возможные значения этого параметра: `","`, `"|"`, `";"` и `" "`. Если имя файла заканчивается на `.csv` и разделитель не указан, то предполагается запятая,иначе пробел.

При вводе простого текста несколько последовательных пробелов и символов табуляции считаются одним разделителем. С другой стороны, каждый из нескольких подряд идущих символов `","`, `"|"` или `";"` рассматривается отдельно. Для простого текстового вывода к уже указанным разделителям добавляется символ `"tab"`. Например, для текстового вывода список `[1234, false, Foo]` записывается как `1234,false,Foo`.

Функция	*Описание*
`read_matrix(S)`	Чтение данных из источника `S` и запись в виде матрицы
`read_matrix(S,M)`	
`read_matrix(S,separator_flag)`	
`read_matrix(S,M,separator_flag)`	
`read_array(S,A)`	Чтение данных из источника `S` и запись в виде массива
`read_array(S,A,separator_flag)`	
`read_list(S)`	Чтение данных из источника `S` и запись в виде списка
`read_list(S,L)`	
`read_list(S,separator_flag)`	
`read_list(S,L,separator_flag)`	
`read_nested_list(S)`	Чтение данных из источника `S` и запись в виде списка подсписков
`read_nested_list(S,separator_flag)`	
`write_data(X,D)`	Запись данных по адресу `D`
`write_data(X,D,separator_flag)`	

Вызов функции `read_matrix` в форме `read_matrix(S)` инициирует чтение данных из источника `S` и их запись в виде матрицы. Размер матрицы определяется из входных данных: каждая строка файла становится одной строкой матрицы. Если строки имеют разную длину, то выдается сообщение об ошибке. Источником `S` может быть имя файла или поток.

Вызов функции `read_matrix` в форме `read_matrix(S,M)` дает возможность читать данные из источника `S` и писать их в матрицу `M`, пока матрица `M` не заполнится или не будут исчерпаны входные данные. Эти данные заполняют матрицу в порядке следования строк и не обязаны иметь то же число строк и столбцов, что и `M`.

Параметры функций `read_array`, `read_list`, `read_nested_list`, `write_data` и их смысл подобны параметрам функции `read_matrix`.

```
(%i2)   s: openr("d:/datafile.txt")$
```

```
(%i3)   M: read_matrix(s, 'comma);
(M)
```

$$\begin{pmatrix} 3.5 & 4.0 & 3.6 & 3.8 & 3.6 & 3.7 & 3.7 & 3.8 \\ 3.5 & 3.6 & 3.7 & 3.6 & 3.7 & 3.8 & 3.6 & 3.7 \\ 3.6 & 3.8 & 3.7 & 3.8 & 3.9 & 3.8 & 3.6 & 3.7 \\ 3.7 & 3.5 & 3.9 & 3.8 & 3.9 & 3.7 & 3.6 & 3.9 \end{pmatrix}$$

```
(%i4)   close(s)$
```

```
(%i5)   s: openr("d:/datafile.txt")$
```

```
(%i6)   L: read_list(s, 'comma);
(L)     [3.5, 4.0, 3.6, 3.8, 3.6, 3.7, 3.7, 3.8, 3.5, 3.6, 3.7, 3.6, 3.7, 3.8, 3.6, 3.7,
3.6, 3.8, 3.7, 3.8, 3.9, 3.8, 3.6, 3.7, 3.7, 3.5, 3.9, 3.8, 3.9, 3.7, 3.6, 3.9]
```

```
(%i7)   close(s)$
```

Функция `write_data` записывает матрицу и массив по порядку строк, по одной строке на запись. Блоки многомерных массивов разделяются дополнительными пустыми строками. Список подсписков записывается способом один подсписок на строку. Функция `write_data` записывает простой список в одну строку.

Адрес `D` может быть именем файла или потоком. Если `D` – имя файла, то глобальная переменная `file_output_append` определяет, будет ли выходной файл пополняться или усекаться. Когда `D` – поток, то функция `write_data` не предпринимает никаких специальных действий после записи всех данных, в частности поток остается открытым.

Функция	*Описание*
`close(stream)`	Закрыть поток
`flength(stream)`	Число байтов в файле или потоке
`flush_output(stream)`	Освобождение буфера вывода
`fposition(stream)`	Текущая позиция в потоке
`fposition(stream,pos)`	
`newline()`	Вывод пустой строки в поток
`newline(stream)`	

Функция	*Описание*
`opena(file)`	Открытие файла для пополнения
`openr(file)`	Открытие файла для чтения
`openw(file)`	Открытие файла для вывода
`printf(dest,string)`	Форматированный вывод
`printf(dest,string, expr_1,...,expr_n)`	
`readbyte(stream)`	Чтение байта
`readchar(stream)`	Чтение символа
`readline(stream)`	Чтение строки
`sprint(expr_1,...,expr_n)`	Печать в строку
`writebyte(byte,stream)`	Вывод байта

Maxima включает средства, позволяющие работать с файловой системой компьютера. Такие средства предоставляет пакет `operatingsystem`.

Оператор	*Описание*
`chdir(dir)`	Изменить рабочий каталог
`mkdir(dir)`	Создать каталог
`rmdir(dir)`	Удалить каталог
`getcurrentdirectory()`	Получить текущий каталог
`copy_file(file1,file2)`	Копировать файл `file1` в файл `file2`
`rename_file(file1, file2)`	Переименовать файл `file1` в файл `file2`
`delete_file(file1)`	Удалить файл `file1`
`getenv(env)`	Получить значение переменной среды `env`

```
(%i8)   load("operatingsystem")$
```

```
(%i9)   chdir("d:/tmp");
(%o9)      #P"d:/tmp/"
```

```
(%i10)   getcurrentdirectory();
(%o10)      "d:/tmp/"
```

4.10. Связь с научной редакторской системой TEX и языком программирования Fortran

Начнем с рассмотрения функций и переменных для вывода информации в формате научной редакторской системы TEX.

Заметим, что встроенные функции вывода в формат TEX в wxMaxima не используют функции, описанные ниже, а применяют собственную реализацию.

Функция	*Описание*
`tex(expr)`	Вывод представления `expr` на консоль
`tex(expr,destination)`	Отсылка представления `expr` по адресу `destination`
`tex(label)`	Вывод на консоль TEX-представления выражения, обозначенного меткой `label`
`tex(label,destination)`	Отсылка представления `expr`, обозначенного меткой, по адресу `destination`
`tex1(expr)`	Вывод представления `expr` на консоль без ограничителей
`texput(a,<опции>)`	Назначение атому `a` строки с TEX-выражением результата выполнения функции или оператора

Функция `tex` выводит представление выражения, подходящего для системы TEX. Результатом является фрагмент документа, который может быть скопирован в документ большего размера, но не обработан сам по себе.

Вызов `tex(label)` выводит на консоль TEX-представление выражения, обозначенного меткой `label`, и назначает ее номеру уравнения (для отображения слева от выражения). Номер уравнения TEX совпадает с номером ячейки системы Maxima.

Параметр `destination` может быть выходным потоком или именем файла. Когда адрес отсылки – имя файла, функция `tex` добавляет результат своей работы в файл. Функции `openw` и `opena` создают выходные потоки.

```
(%i1)   z: sin(x);
(%o1)       sin(x)
```

```
(%i2)   tex(z);
(%o2)      $$\sin x$$
```

```
(%i3)   taylor(z,x,0,5);
                                 3    5
(%o3)       x - x/6 + x/120 + ...
```

```
(%i4)   tex(%o3);
     $$x-{{x^3}\over{6}}+{{x^5}\over{120}}+ ... \leqno{\tt (%o3)}$$
(%o4)       false
```

```
(%i5)   tex1(diff(f(x,t),t));
(%o5)      {{d}\over{d\,t}} f\left(x , t\right)
```

```
(%i6)   texput(h,"\\lambda\^5");
(%o6)       "\lambda^5"
```

```
(%i7)   tex(h);
     "$$\lambda^5$$"
(%o7)       false
```

Дополнительные возможности вывода в TEX-формы см. в документации по системе.

Функция	*Описание*
`fortran(expr)`	Вывод представления `expr` на консоль в классическом формате программного текста на языке Fortaran

Выходная строка имеет отступы. Если строка слишком длинная, то функция `fortran` выводит строки продолжения. Команда `fortran` выводит оператор возведения в степень `^` как `**`, а комплексное число `a + b %i` в форме `(a, b)`.

Параметр `expr` может быть равенством. Если это именно так, функция `fortran` выводит оператор присваивания правой части равенства левой. В частности, если правая часть выражения `expr` является именем матрицы, то функция `fortran` выводит оператор присваивания для каждого элемента матрицы.

Если параметр `expr` не распознается командой `fortran`, то выражение выводится в формате функции `grind` без дополнительных сообщений. Функция `fortran` не обрабатывает списки, массивы и функции.

```
(%i8)   expr: (sin(x) -cos(2*x))^6;
(expr)      (sin(x) - cos(2 x))^6
```

```
(%i9)   fortran(expr);
      (sin(x)-cos(2*x))**6
(%o9)       false
```

```
(%i10)   fortran('x=expand(expr));
     x = cos(2*x)**6-6*sin(x)*cos(2*x)**5+15*sin(x)**2*cos(2*x)**4-
    1 20*sin(x)**3*cos(2*x)**3+15*sin(x)**4*cos(2*x)**2-6*sin(x)**5*
    2 cos(2*x)+sin(x)**6
(%o10)      done
```

Системная переменная	*По умолчанию*	*Описание*
`fortindent`	0	Контроль смещения правого края выражений, выводимых функцией `fortran`. Если 0, то смещение на 6 колонок, если больше 0, то еще больше вправо
`fortspaces`	`false`	Если `true`, то функция `fortran` заполняет пробелами каждую строку до 80 колонки

Функция	*Описание*
`f90(expr_1,...,expr_n)`	Вывод выражений `expr_1`, ..., `expr_n` на консоль в формате программного текста на языке Fortaran 90

Функция `f90` осуществляет вывод в свободном формате ввода для языка Fortran 90. Особое внимание не уделяется позициям столбцов. Длинные строки разделяются на строки фиксированной длины с помощью символов продолжения `&`. Функция `f90` выводит один символ `&` в конце разделенной строки, а другой – в начале следующей строки. Для использования этой функции необходимо загрузить пакет `f90`.

Системная переменная	*По умолчанию*	*Описание*
`f90_output_line_length_max`		Максимальное количество выводимых символов в строке, не считая амперсантов

Параметр `f90_output_line_length_max` должен быть целым положительным числом.

```
(%i11)   load("f90")$
```

```
(%i12)   f90('x=expand(expr));
     x = cos(2*x)**6-6*sin(x)*cos(2*x)**5+15*sin(x)**2*cos(2*x)**4-&
     &20*sin(x)**3*cos(2*x)**3+15*sin(x)**4*cos(2*x)**2-6*sin(x)**5*&
     &cos(2*x)+sin(x)**6
(%o12)       false
```

5. ГРАФИКИ. ОСНОВНЫЕ И ДОПОЛНИТЕЛЬНЫЕ ЭЛЕМЕНТЫ

Построение графиков функций одной и двух переменных является важным инструментом для исследования свойств таких функций. Для этой задачи Maxima использует внешние графические пакеты. Таким образом, можно выбрать один из двух мощных графических инструментов, которые присутствуют в стандартной установке Maxima:

1°. gnuplot (по умолчанию). Чтобы получить сведения по этому приложению, см. сайт `http://www.gnuplot.info/`, а также предыдущий подраздел.

2°. xmaxima (ранее openmath).

Функции построения графиков системы Maxima рассчитывают набор точек и передают их в программу печати вместе с набором команд. Все эти точки и команды построения сохраняются в файле `maxout.gnuplot`, который находится по умолчанию в локальной папке.

В среде wxMaxima можно легко переключаться между этими инструментами печати, если воспользоваться меню графиков. Заметим, что оба инструмента gnuplot и xmaxima открывают окна, в которых отображаются запрошенные графики. К сожалению, третий инструмент mgnuplot, присутствующий в списке для выбора wxMaxima версии 19.01.2x, использовавшейся при написании данного пособия, находится в нерабочем состоянии.

Далее описывается использование только первого инструмента (gnuplot).

Стандартными командами для построения графиков в Maxima являются `plot2d` и `plot3d`.

В целом существуют три типа графиков: явные, неявные и параметрические. Но при этом синтаксис функций построения графиков, описываемый в пунктах ниже, практически идентичен.

1°. Явные: функции задаются в терминах входных параметров (аргументов). Например, функция $y = f(x) = x^2$, определенная на $\mathbb{R}$, или $z = f(x, y) = \sin(x\,y)$, определенная в прямоугольнике $[-2, 2] \times [-3, 2]$.

2°. Неявные: такие функции задаются уравнениями типа $F(x, y) = 0$. Например, это функция $x^2 + y^2 = 1$ в единичном квадрате $[0, 1] \times [0, 1]$.

3°. Параметрические: кривая для построения задается как траектория, т.е. как отображение $(0, 1) \mapsto \mathbb{R}^2$. Например, параболу $y = x^2$ можно параметризовать, используя свободный параметр с именем t и установив, что $x = t$ и $y = t^2$.

5.1. 2D-графики

5.1.1. Построение графиков функций, заданных в явном виде

Команда	*Описание*
`plot2d(funct,x_range,opts)`	Построить график функции одной переменной

Опция	*По умолчанию*	*Описание*
`[x,x_min,x_max]`	Задавать обязательно	Область `x_range` переменной `x`
`[y,y_min,y_max]`	Автоматический выбор	Промежуток изменения ординаты (ось OY)

```
(%i1)   plot2d(sin(x), [x,-%pi,%pi])$
```

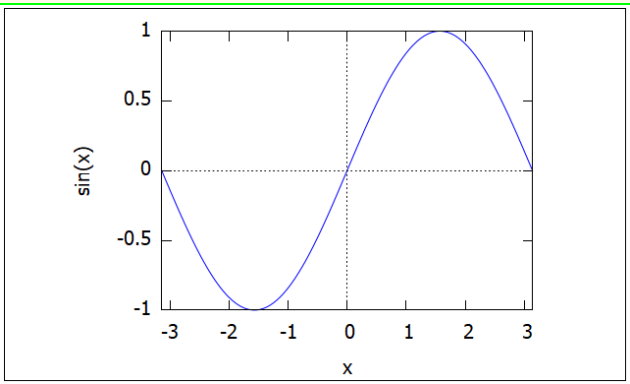


Рис. 5.1

Наиболее распространенное использование `plot2d` – это построение графика функции $y = f(x)$ (рис. 5.1). Первый аргумент `funct` – это выражение, которое описывает функцию $f(x)$. Второй аргумент `x_range` является обязательным и представляет собой список `[x,x_min,x_max]`, который содержит имя переменной (обычно x, но возможны и другие имена переменных), а также минимальные и максимальные значения `x_min` и `x_max` аргумента x соответственно. Кроме того, дополнительные опции (т.е. функция `plot2d` может иметь переменное число аргументов!) могут быть указаны в качестве третьего, четвертого, ... аргументов, каждый из которых может быть списком с заключенными в квадратные скобки параметрами (см. далее).

Диапазон для оси ординат может быть уточнен с помощью необязательного аргумента в форме `[y,y_min,y_max]` (ключевое слово `y` всегда используется для вертикальной оси). Если используется эта опция (рис. 5.2), то график будет построен именно в этих пределах независимо от значений, достигаемых функцией. Если вертикальный диапазон не указан, то он будет выбран в соответствии с минимальным и максимальным значениями второй координаты точек графика.

```
(%i2)   plot2d(sin(x), [x,-%pi,%pi], [y,-2,2])$
```

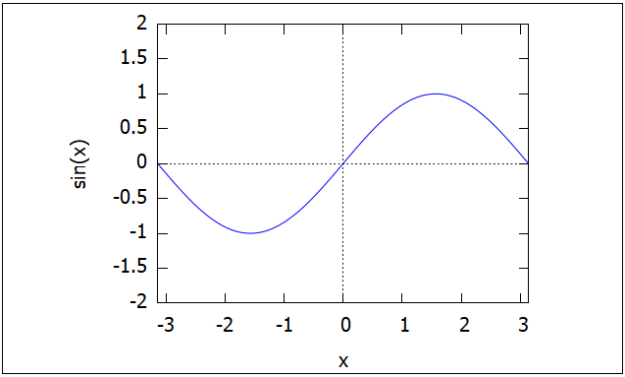


Рис. 5.2

5.1.2. Построение графиков функций, заданных параметрически

Команда	*Описание*
`plot2d([parameter,x_coord, y_coord,t_range],opts)`	График функции, заданной параметрически

Опция	*По умолчанию*	*Описание*
`[t,t_min,t_max]`	Задается обязательно	Область `t_range` параметра `t`
`nticks`	29	Число точек, используемых для построения графика

График параметрической функции определяется списком, начинающимся с ключевого слова `parametric`, за которым следуют два выражения, которые являются функциями параметра для каждой координаты, а также диапазон изменения параметра. Данный диапазон должен быть

списком с именем параметра (обычно `t`, но возможны и другие имена), за которым следуют его минимальное и максимальное значения. График показывает траекторию, пройденную точкой с координатами, заданными двумя выражениями или функциями, при увеличении параметра t от t_{min} до t_{max}.

Опция `nticks` (рис. 5.3) устанавливает количество точек, используемых для построения графика. Эта опция контролирует гладкость параметрической кривой.

```
(%i3)   plot2d([parametric,t*sin(2*%pi*t),t*cos(2*%pi*t)],[t,0,2]],
[nticks,1000])$
```

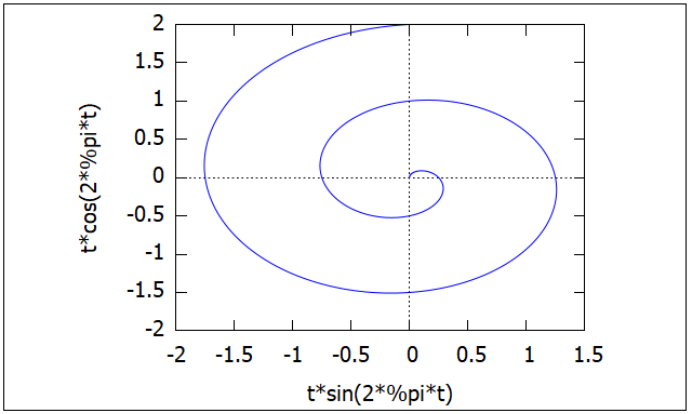


Рис. 5.3

5.1.3. Построение графиков функций, заданных таблично

Команда	*Описание*
`plot2d([discrete,[x_coords,...],[y_coords,...]],opts)`	Ломаная, построенная по заданным точкам
`plot2d([discrete,[[x_coord,y_coord],...],opts)`	То же самое

Опция	*По умолчанию*	*Описание*
`style`	`lines`	Стиль графика

Данные для графика также могут быть представлены в дискретной форме, т.е. в наборе точек с заданными координатами. Дискретный график определяется списком, начинающимся с ключевого слова `discrete`,

за которым следуют один или два списка значений. Если даны два списка, они должны иметь одинаковую длину; первый список будет интерпретирован как абсциссы точек, которые будут построены, а второй список – как ординаты. Если после дискретного ключевого слова указан только один список, то каждый элемент в таком списке должен быть подсписком с двумя элементами, которые соответствуют абсциссам и ординатам точки.

В первом примере на графике (рис. 5.4) точки соединены линиями.

```
(%i4)   plot2d([discrete, [10,20,30,40,50], [.6,.9,1.1,1.3,1.4]])$
```

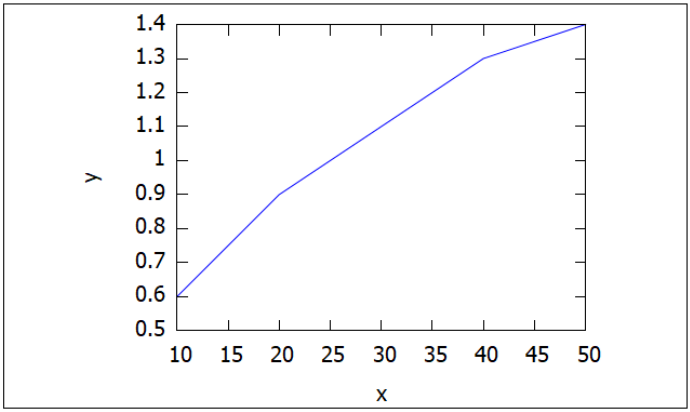


Рис. 5.4

Второй пример (рис. 5.5) просто показывает отображение точек. В этом примере точки заданы в одном списке, где каждая из них представляет собой список из двух элементов.

5.1.4. Графики в полярной системе координат

Чтобы построить график функции, заданной в полярных координатах, необходимо задать зависимость значений полярного радиуса от полярного угла. Пусть $r = g(\theta)$ $(\theta_1 \leqslant \theta \leqslant \theta_2)$ – такая зависимость. Тогда график этой функции можно построить, задав при вызове функции `plot2d` опцию `[gnuplot_preamble, "set polar"]`. Данная опция будет действовать при условии, что выбран формат графика `gnuplot`.

Есть еще один способ построения графиков в полярной системе координат в рамках функции `plot2d` в системе Maxima. Зная зависимость

```
(%i5)   plot2d([discrete, [[10,.6],[20,.9],[30,1.1],[40,1.3],
[50,1.4]]],[style,points])$
```

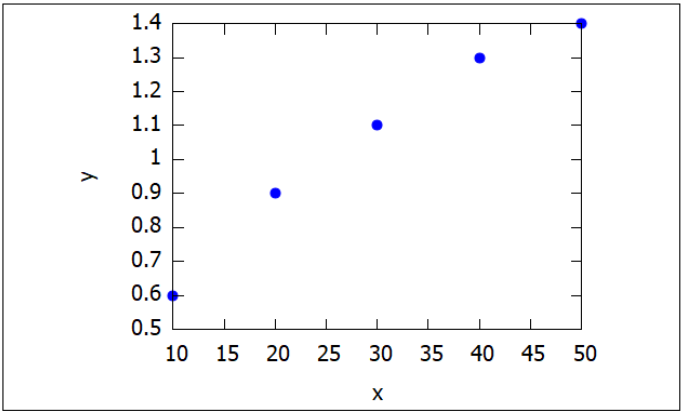


Рис. 5.5

```
(%i6)   plot2d([sin(4*theta)^2,cos(3*theta)^2], [theta,0,2*%pi],
[gnuplot_preamble, "set polar"], [x,-1.5,1.5], [y,-2,2]);)$
```

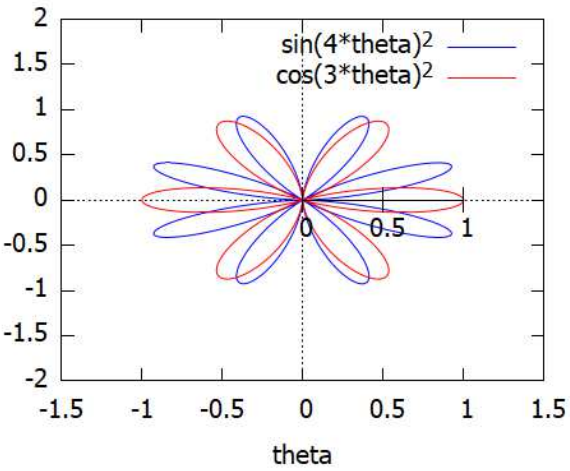


Рис. 5.6

полярного радиуса r от полярного угла, для построения таких графиков достаточно установить связь угла θ с декартовыми координатами x и y:

$$x = g(\theta)\cos\theta, \qquad y = g(\theta)\sin\theta,$$

затабулировать значения функций $x(\theta)$, $y(\theta)$ при выбранных возрастающих значениях θ, а затем обратиться к функции `plot2d` с параметром `discrete`.

```
(%i7)   k: 1200$ lst0: makelist(float(j*8*%pi/k),j,0,k)$
```

```
(%i8)   f1(t1):=1+cos(8*t1)*sin(3*t1);
```

```
(%i9)   lst1: makelist([lst0[i+1],f1(lst0[i+1])],i,0,k)$
```

```
(%i10)   lst2: makelist([lst1[i+1][2]*cos(lst1[i+1][1]),lst1[i+1][2]*
sin(lst1[i+1][1])],i,0,k)$
```

```
(%i11)   plot2d([discrete,lst2])$
```

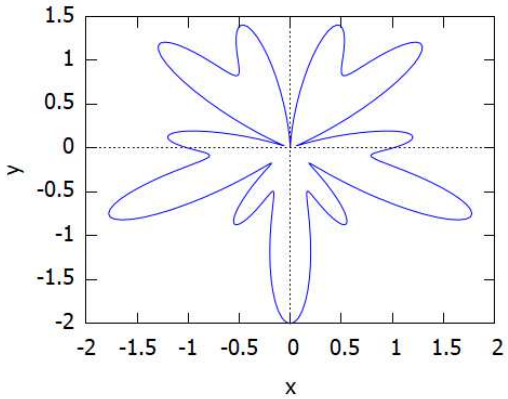


Рис. 5.7

```
(%i12)   f2(t2):=exp(cos(t2))-2*cos(4*t2)-sin(t2/12)^5$
```

```
(%i13)   lst3: makelist([lst0[i+1],f2(lst0[i+1])],i,0,k)$
```

```
(%i14)   lst4: makelist([lst3[i+1][2]*cos(lst3[i+1][1]),lst3[i+1][2]*
sin(lst3[i+1][1])],i,0,k)$
```

```
(%i15)   plot2d([discrete,lst4])$
```

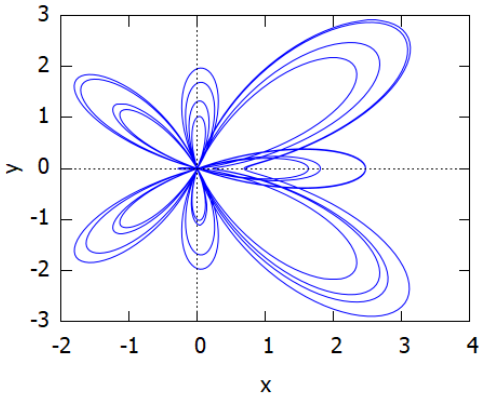


Рис. 5.8

Заметим, что в системе Maxima есть пакет расширения `draw`, 2D-сценарии которого позволяют строить графики в полярной системе координат.

5.1.5. Графики функций, заданных неявно

Команда	*Описание*
`implicit_plot(funct,x_range,y_range,opts)`	График неявной функции

Для построения неявных функций типа $F(x, y) = 0$ необходимо загрузить пакет `implicit_plot`. Синтаксис команды для построения графика неявной функции отличается от синтаксиса команды `plot2d`. Здесь требуется указать интервалы для обеих переменных, присутствующих в записи неявной функции (рис. 5.9).

5.1.6. Несколько кривых на одном графике

Команда	*Описание*
`plot2d([plot_1,...,plot_n],opts)`	Отображение нескольких графиков

Можно объединять два графика и более в одну картинку. Графики должны быть заданы в виде списка. Ниже пример, который объединяет график явно заданной функции $f(x) = x^3$ и график окружности, заданной параметрически (рис. 5.10).

```
(%i16)   load(implicit_plot)$
(%i17)   implicit_plot(x^5+y^5=x*y^2, [x,-1,1], [y,-0.9,0.9]);
```

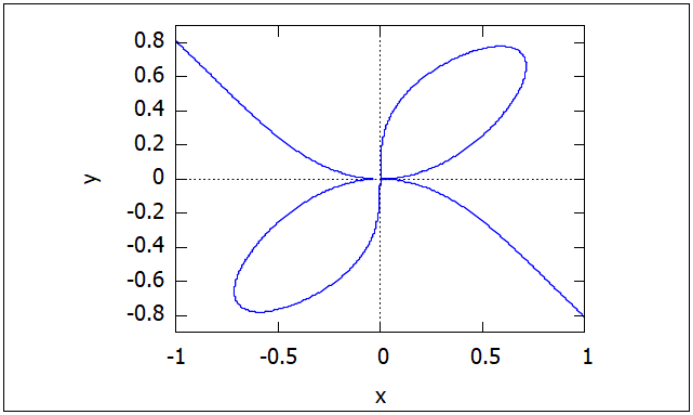


Рис. 5.9

```
(%i18)   plot2d([x^3, [parametric,cos(t),sin(t),[t,-%pi,%pi]]],
[x,-1,1])$
```

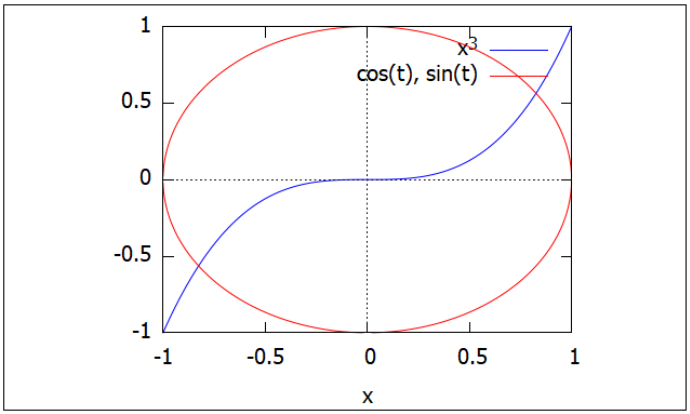


Рис. 5.10

5.2. 3D-графики

В основном функции для построения графиков в трех измерениях

аналогичны командам для двух измерений.

Функция	*Описание*
`plot3d(funct,x_range,y_range,opts)`	Построить график функции двух переменных

Обратимся к формату команды для построения графика функции двух переменных. Единственная необходимая модификация – это добавление второй переменной и границ ее изменения (рис. 5.11). Заметим, что клавишный манипулятор "мышь" можно использовать для вращения графика с целью наиболее выгодного ракурса обзора поверхности. Для этого достаточно просто щелкнуть "мышью" по изображению и удерживать кнопку нажатой во время перемещения "мыши" (какую кнопку "мыши" использовать, зависит от настроек операционной системы, установленной на компьютере).

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

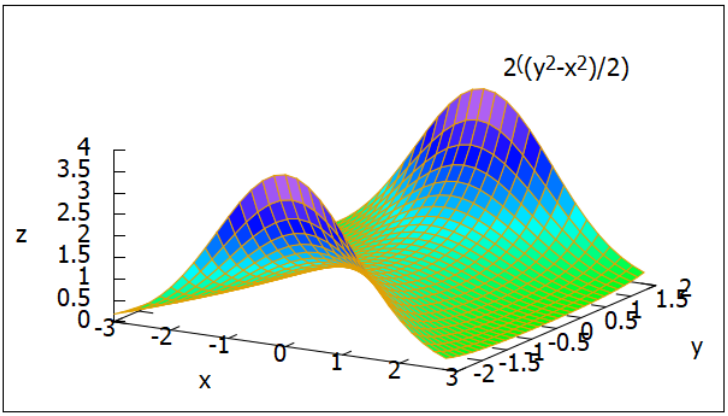


Рис. 5.11

Функция	*Описание*
`plot3d([x,y,z],s_range,t_range,opts)`	График параметрической 3D-функции

Следующий пример демонстрирует пример построения параметрического 3D-графика. Поверхность задается тремя выражениями –

$$x = \varphi(t,s), \qquad y = \psi(t,s), \qquad z = \chi(t,s)$$

```
(%i2)   moeb_x: cos(s)*(3+t*cos(s/2))$
(%i3)   moeb_y: sin(s)*(3+t*cos(s/2))$
(%i4)   moeb_z: t*sin(s/2)$
(%i5)   plot3d([moeb_x,moeb_y,moeb_z], [s,-4,4], [t,-4,4])$
```

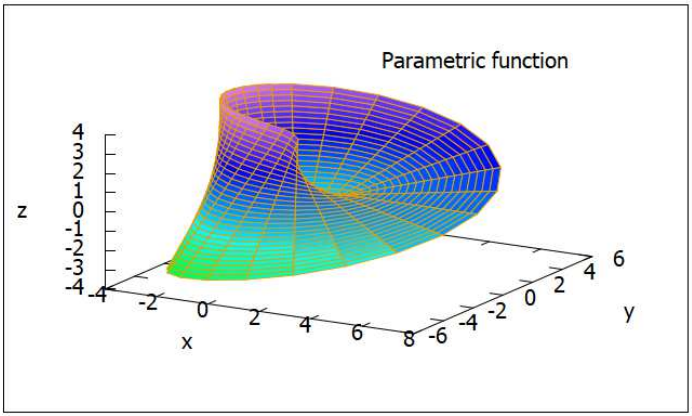


Рис. 5.12

– для трех координат в виде функций двух параметров, обычно называемых s и t. Этот график (рис. 5.12) изображает известный лист Мёбиуса.

Функция	*Описание*
`contour_plot(funct,x_range,y_range,opts)`	Построение линий уровня

График линий уровня $f(x, y) = C$ функции двух переменных $z = f(x, y)$ состоит из кривых, на которых функция z имеет одинаковые значения. Для построения необходимо передать опции в gnuplot, чтобы установить количество желаемых линий уровня. График линий уровня в следующем примере (рис. 5.13) включает 30 кривых.

5.3. Опции графиков

Maxima предлагает широкий набор опций (вариантов) создания графиков. Ниже описываются только самые важные из них. Для просмотра полного набора необходимо обратиться к разделу Plotting в руководстве по системе Maxima. Опции всегда указываются в виде списка с именем опции в качестве первого элемента: `[opt, value, ...]`.

```
(%i6)   contour_plot(u^3+v^2, [u,-4,4], [v,-4,4], [legend,false],
[gnuplot_preamble,"set cntrparam levels 12"]);
```

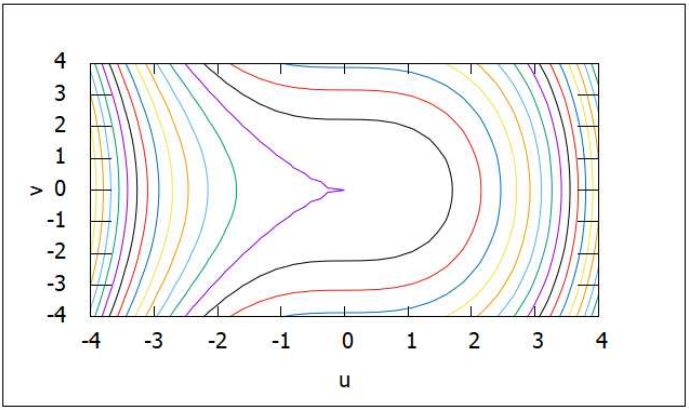


Рис. 5.13

Опция	*По умолчанию*	*Описание*
`[xlabel, "text"]`	`"x"`	Метка для оси абсцисс
`[ylabel, "text"]`	`"y"`	Метка для оси ординат
`[zlabel, "text"]`	`"z"`	Метка для оси аппликат

С помощью этих параметров можно указать любой текст для обозначений двух (`plot2d`) и трех осей (`plot3d`) соответственно.

Опция	*По умолчанию*	*Описание*
`[legend, "text1", "text2", ...]`	Автоматически	Метки для осей добавляются
`[legend, false]`		Подавление подписей

Когда на графике отображается более одной кривой / поверхности (два или более графика непрерывных или таблично заданных кривых / поверхностей строятся с помощью `plot2d` или `contour_plot`), то эти кривые / поверхности рисуются разными линиями (обычно разными цветами). Легенда в правом верхнем углу обозначает эти линии. По умолчанию используются имена выражений/функций или слова `discrete1`, `discrete2`, ... для дискретных множеств точек. Отображение легенды может быть подавлено с помощью опции `[legend, false]`.

Опция	*По умолчанию*	*Описание*
`[x,x_min,x_max]`	Задается	Интервал для координаты x
`[y,y_min,y_max]`		Интервал для координаты y
`[z,z_min,z_max]`	Дополнительно	Интервал для координаты z

Когда график функции строится с использованием `plot2d`, диапазон для переменной x является обязательным. Диапазон для вертикальной оси вычисляется автоматически: он будет настроен в соответствии с минимальным и максимальным значениями второй координаты точек графика. Это может иногда не соответствовать желаемому. При необходимости вертикальный диапазон можно настроить с помощью опции для координаты y.

Аналогично при использовании `plot3d` должны быть заданы диапазоны изменения двух переменных, а диапазон третьей координаты (z) вычисляется автоматически или может быть настроен с использованием параметра для z.

Заметим, что аргументы функций, заданных для `plot2d` и `plot3d`, необязательно называть `x` и `y`. Параметры для x и y можно использовать для настройки диапазона изменения первых координат. В частности, область построения может быть больше, чем могла бы быть, чтобы уместить график функции.

Опция	*По умолчанию*	*Описание*
`[nticks, integer]`	`29`	Начальное число точек для построения графиков командой `plot2d`

При построении функций с помощью команды `plot2d` мы можем контролировать гладкость нарисованной кривой, устанавливая количество точек с помощью опции `nticks`. Для явных графиков это просто начальное число точек, которое будет автоматически увеличено, если данная функция имеет большие вариации. Для параметрических графиков это фактическое количество точек, которые будут показаны на графике. Это может привести к построению кривой, которая выглядит не очень гладко. В этом случае рекомендуется увеличить `nticks`.

Опция	*По умолчанию*	*Описание*
`[grid, integer, integer]`	`[30,30]`	Число точек $[n_1, n_2]$ для построения поверхностей командой `plot3d`

Команда `plot3d` отрисовывает данную функцию на заданной сетке. Эту сетку $n_1 \times n_2$ можно задать, задав количество точек, которые будут

использоваться в направлениях вдоль оси OX и оси OY для построения трехмерного изображения. Необходимо предупредить, что увеличение этих чисел может привести к длительному времени вычислений и построения поверхности.

Опция	*По умолчанию*	*Описание*
`[style,type_1, type_2, ...]`	Автоматически	Стили построения графика
`[style,[type_1], [type_2],...]`	Автоматически	Стили построения графика

gnuplot выбирает некоторый стиль для каждой функции или набора данных, чтобы их можно было различить на одном графике. Тем не менее не всегда получается то, что требуется. Например, необходимо объединить точечные графики с непрерывными кривыми. Опция `style` позволяет указать такие стили явным образом. Для каждой функции или набора данных должен быть задан стиль, который может быть одним из нижеперечисленных:

- `lines` для непрерывных сегментов линий,
- `points` для отдельных точек,
- `linespoints` для непрерывных сегментов и точек,
- `dots` для малых изолированных точек.

Первые три стиля могут иметь необязательный параметр (целое число)[1], который определяет ширину линии и размер точки соответственно. Например, у каждого из этих стилей должен быть список: `[lines, 2]` задает стиль непрерывной линии толщиной 2.

Опция	*По умолчанию*	*Описание*
`[color,color_1,color_2,...]`	Автоматически	Цвет линии или точки

В `plot2d` и `implicit_plot` мы можем определить цвет (или цвета) для различных кривых, используя опцию `color`. Цвета для каждой кривой взяты из заданного списка по порядку.

В `plot3d` этот параметр определяет цвета, используемые для линий сетки одной (`color_1`) или нескольких (`color_1`, `color_2` ...) поверхностей, когда палитра не используется.

Доступные цвета: `blue`, `red`, `green`, `magenta`, `black`, `cyan`.

Опция	*По умолчанию*	*Описание*
`[point_type,type_1,type_2,...]`	Автоматически	Тип линии

[1] На самом деле параметров может быть и больше. См. полное описание системы.

При построении точек можно выбирать из списка их возможных типов. Типы точек для каждого набора данных берутся из заданного списка в последовательном порядке.

Доступные типы: `bullet`, `circle`, `plus`, `times`, `asterisk`, `box`, `square`, `triangle`, `delta`, `wedge`, `nabla`, `diamond`, `lozenge`.

Опция	*По умолчанию*	*Описание*
`[mesh_lines_color, color]`	`black`	Цвет, используемый для рисования линий сетки, представляющих поверхность
`[mesh_lines_color, false]`	Автоматически	Отключение рисования линий сетки
`[palette,[palette_1], ...]`	Автоматически	Цвета для поверхностной тени
`[palette, false]`	Автоматически	Отключение рисования поверхностной тени

Команда `plot3D` использует два метода для рисования поверхности. Она рисует сетку в соответствии с ее заданными границами. Затем квадраты этой сетки заполняются с использованием заданной цветовой палитры. Таким образом, цвета указывают значения функции (как на любой географической карте с горами) или могут имитировать затенение.

Цвета поверхности задаются опцией `palette`. Каждая палитра представляет собой список с ключевым словом, за которым следуют четыре цифры. Ключевое слово указывает, какой из трех атрибутов (`hue`, `saturation`, `value` – оттенок, насыщенность, значение) будет увеличен в соответствии со значениями координаты z. Первые три числа, которые должны быть между 0 и 1, определяют основной цвет, который должен быть назначен минимальному значению z. Последнее число указывает на увеличение, соответствующее максимальному значению z.

По умолчанию для первого графика: `[hue, 0.25, 0.7, 0.8, 0.5]`.

Если палитре присвоено значение `false`, то поверхности не будут затенены, а представлены только сеткой кривых. В этом случае цвета линий будут определяться опцией `color`.

В противном случае цвет линий сетки задается параметром `mesh_lines_color`. Он принимает те же цвета, что и для опции `color`. Также может быть задано значение `false`, чтобы полностью исключить линии сетки.

Рассмотрим примеры использования этих опций.

В первом примере (рис. 5.14) отображаются две функции, обозначенные `first function` и `second function`. Метки для координат x и y – `abscissa` и `ordinate` соответственно. При вызове команды используем

```
(%i1)   plot2d([exp(-x^2/2), sqrt(%e)*exp(-x)], [x,0,2],
[y,0,1.5], [nticks, 100], [legend, "first function", "second
function"], [xlabel, "abscissa"], [ylabel, "ordinate"], [style,
[lines,3],[lines,1]], [color, magenta,blue])$
```

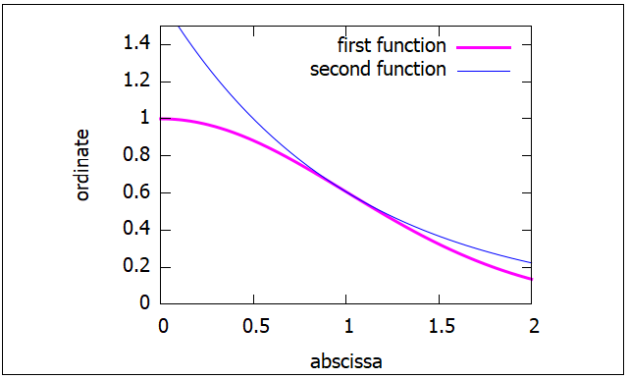


Рис. 5.14

```
(%i2)   data_list: [[0,-4], [0.5,-1], [1,0.25], [1.2,0.4],
[1.5,0.1], [1.7,0.3], [2,1], [2.2,0.75], [3,1.1]]$
(%i3)   plot2d([log(x), [discrete, data_list]], [x,0,5], [legend,
"theory", "experiment"], [style, [lines,2], [points]], [color, blue,
red], [point_type, wedge])$
```

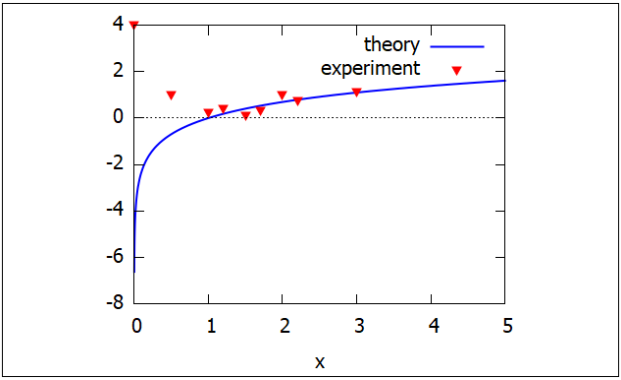


Рис. 5.15

100 точек для каждого графика и устанавливаем отрезок [0, 1.5] как область рисования вдоль вертикальной оси. Первая функция изображена толстой пурпурной линией, а вторая – тонкой синей.

В следующем примере (рис. 5.15) показаны точки данных (например экспериментальных) вместе с аппроксимирующей функцией.

В примере для построения двух поверхностей (рис. 5.16) используется только сетка, а затенение не используется. Первая поверхность раскрашивается в синий цвет, а вторая – в красный.

```
(%i4)   plot3d([sin(x+y), cos(x-y), [x,-%pi,%pi], [y,-%pi,%pi]],
[palette,false], [color, blue, red], [grid,50,50])$
```

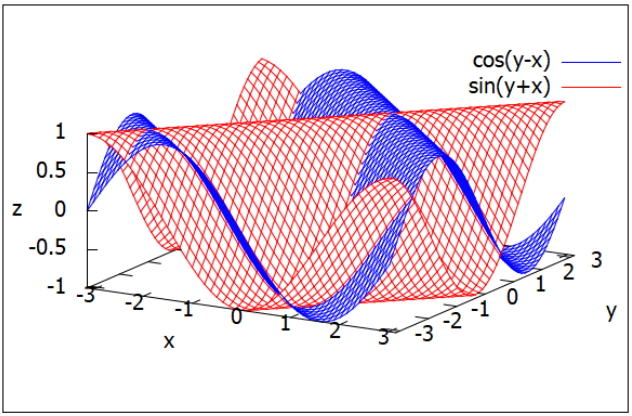


Рис. 5.16

Примечание. На рис. 5.16 поверхности отображаются заданными цветами, но цвета в легенде не соответствуют заданным.

В следующем примере (рис. 5.17) показан график без сетки. Кроме того, затенение отличается от настройки по умолчанию, и используется более тонкая сетка.

Последний пример (рис. 5.18) показывает влияние опционной сетки. Здесь используется грубая сетка.

5.4. Пакет Draw

Возможности команды `plot` системы Maxima ограничены и могут не удовлетворять всех пользователей. Более того, дополнительные параметры для базовой подпрограммы имеют синтаксис, отличный от синтаксиса Maxima.

```
(%i5)   plot3d(sin(x*y), [x,-2,2], [y,-2,2], [grid, 50,50],
[mesh_lines_color,false], [palette, [hue, 0.5, 0.7, 0.6, 0.2]])$
```

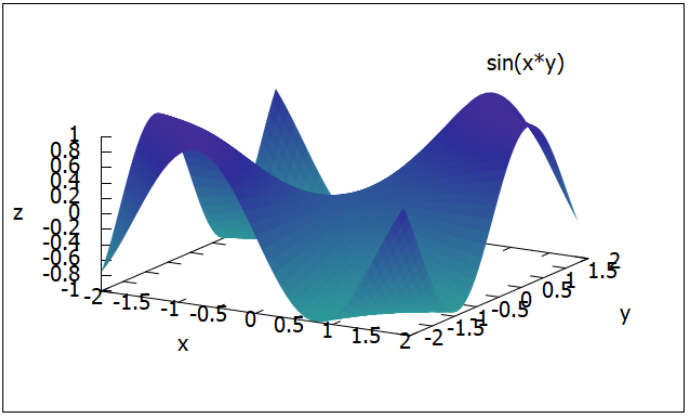


Рис. 5.17

```
(%i6)   plot3d(sin(x*y), [x, -%pi, %pi], [y, -%pi, %pi], [grid,
10, 10])$
```

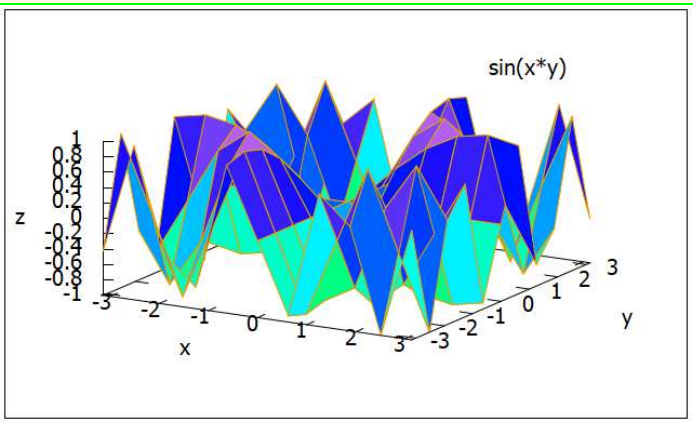


Рис. 5.18

Пакет `draw` представляет собой более мощный и гибкий инструмент для построения графиков. Более того, более сложные картинки могут

быть созданы с помощью векторной графики[2]. Однако этот пакет поддерживает только процедуру gnuplot. Но прежде чем можно будет использовать пакет, он должен быть загружен.

```
(%i1)   load(draw)$
```

В настоящем пособии описываются лишь некоторые свойства пакета draw. Более полный перечень возможностей пакета можно найти в справочнике по системе.

Функция	*Описание*
gr2d(opts,graphic_objects)	Построение 2D-объекта
gr3d(opts,graphic_objects)	Построение 3D-объекта
draw(scenes,opts)	Соединение нескольких объектов в одной картинке
draw2d(opts,graphic_object)	Короткая форма для draw(gr2d(opts,graphic_object))
draw3d(opts,graphic_object)	Короткая форма для draw(gr3d(opts,graphic_object))

Пакет draw работает по следующей схеме. Картинки собираются из объектов, формируемых gr2d или gr3d, которые затем уже передаются в функцию draw. Если описано более одного объекта то будет создана мультикартинка. Таким способом можно построить довольно сложные картинки. Если строится только один объект, то можно использовать draw2d(...) и draw3d(...). Это следствие того, что они эквивалентны draw(gr2d(...)) и draw(gr3d(...)) соответственно.

Для того чтобы получить список доступных графических объектов в двух и трех измерениях, нужно ввести запросы ? gr2d и ? gr3d.

Опции для этих функций (*графические опции*) задаются в виде пар key=value (ключевое слово = значение).

Объекты в конструкторе gr2d и gr3d интерпретируются последовательно: графические параметры влияют на те графические объекты, которые расположены справа. Некоторые графические параметры влияют на общий вид картинки, могут располагаться в любом месте или передаваться команде draw (it глобальные опции).

Как уже отмечено, 2D-объекты могут быть построены с помощью конструктора картинок gr2d, в котором есть несколько типов доступных графических объектов. Так для построения графиков функций имеются следующие формы:

[2]В векторной графике фигура состоит из элементарных геометрических объектов, таких как линии, многоугольники, круги, дуги и т.д., которые хранятся в терминах координат их характерных точек и характерных параметров, таких как радиус круга.

- `explicit`: явная функция;
- `implicit`: неявная функция;
- `points`: таблично заданная функция. Построение точек, которые соединены линиями, если для параметра `points_joined` было установлено значение `true` в предыдущей строке текущей сцены;
- `parametric`: параметрическая функция.

Помимо этого есть некоторые низкоуровневые графические объекты (*графические примитивы*) для добавления линий или аннотирования графиков. Следующие объекты имеют понятные имена. Для получения более подробной информации используйте справку по системе Maxima, например, в виде `? points`.

- `bars`
- `ellipse`: эллипс.
- `image`: построение bmp-графика.
- `label`: помещает текст на поле графика.
- `polygon`: многоугольник.
- `quadrilateral`: четырехугольник.
- `rectangle`: прямоугольник.
- `triangle`: треугольник.
- `vector`: стрелка.
- `region`: построение области, определенной неравенствами.

Есть еще графические базовые структуры, уже не совсем простые, как перечислено, в других графических пакетах. Например, в пакете построения географических карт `worldmap` есть единственный "примитив" `geomap`.

Команда	*Описание*
`explicit(func,x,xmin,xmax)`	Построить график явной функции одной переменной

Опция	*По умолчанию*	*Описание*
`xrange=[xmin,xmax]`	Automatic	Диапазон `xrange` переменной `x`
`yrange=[ymin,ymax]`	Automatic	Диапазон `yrange` переменной `y`

Графики явных функций могут быть созданы с помощью графического объекта `explicit`. Заметим, что в отличие от команды `plot2d` параметры области задается не списком, а перечислением в параметре `explicit`. Конечно, аргумент, соответствующий абсциссе, не обязательно называть `x`.

Диапазоны для горизонтальной и вертикальной осей обычно вычисляется автоматически исходя из заданной области определения функции

и диапазона значений этой функции. Эти диапазоны можно настроить с помощью соответствующих опций nxrange и yrange.Заметим, что это глобальные параметры и могут быть указаны в любом месте.

Цвет кривой можно указать с помощью параметра color. Это локальная опция и, следовательно, она влияет только на графические объекты, которые расположены справа от списка аргументов grd2d.

Команда	*Описание*
`implicit(feq,x,xmin,xmax, y,ymin,ymax)`	Построить 2D-график неявной функции одной переменной с аргументами x и y

Неявные функции двух переменных определяются соотношениями вида $F(x, y) = 0$. Такое соотношение дается аналогично явным функциям, но в отличие explicit, implicit требует, чтобы область для обеих задействованных переменных была указана. Первая переменная, указанная в implicit, будет соответствовать оси абсцисс, а вторая – оси ординат. Конечно, эти аргументы не обязаны называться x и y.

Команда	*Описание*
`parametric(xfun,yfun, t,tmin,tmax)`	Построить график параметрической функции с параметром t

Параметрический график определяется парой выражений для координат x и y соответственно и переменной (называемой *параметром*) для каждой из этих двух функций (параметр обычно называют t, но возможны и другие имена). Такой график показывает траекторию точки с координатами, заданными двумя выражениями, при увеличении параметра t от tmin до tmax.

Команда	*Описание*
`points([[x_coord,y_coord],...])`	Точки на плоскости
`points([x_coords,...],[y_coords,...]`	Точки на плоскости

Опция	*По умолчанию*	*Описание*
`point_type`	`plus` (1)	Тип точки
`point_size`	1	Размер точки
`points_joined`	`false`	Если `true`, то точки соединяются ломаной

График также может быть определен в дискретной форме, т.е. через набор точек с заданными координатами. Они могут быть заданы либо как список значений [x, y], либо двумя списками одинаковой длины координат x и y каждой точки соответственно.

```
(%i2)   draw2d(color = red, point_type = filled_circle, point_size
= 1.5, points([[10,.6], [20,.9], [30,1.1], [40,1.3], [50,1.4]]),
color = blue, point_type = filled_diamant, point_size = 2,
points([[10,.2], [20,.5], [25,0.8], [35,1.0], [43,1.3], [50,1.6]]))$
```

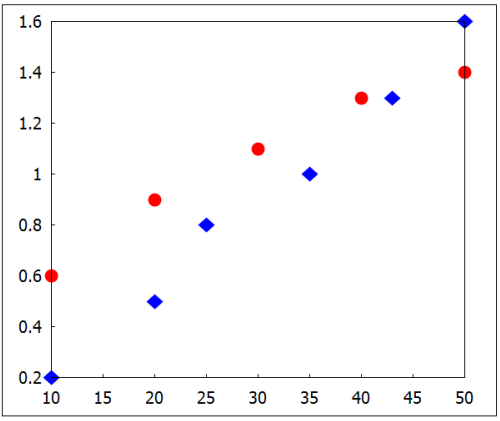


Рис. 5.19

По умолчанию Maxima создает точечный график. Стиль и размер нанесенных точек могут быть изменены с помощью соответствующих параметров `point_type` и `point_size`. Типы точек могут быть указаны либо числом, либо именем стиля графика: `none` (–1), `dot` (0), `plus` (1), `multiply` (2), `asterisk` (3), `square` (4), `filled_square` (5), `circle` (6), `filled_circle` (7), `up_triangle` (8), `filled_up_triangle` (9), `down_triangle` (10), `filled_down_triangle` (11), `diamant` (12), `filled_diamant` (13).

Опция	*По умолчанию*	*Описание*
`key="текст"`	"" (пустая строка)	Имя функции в легенде

Возможно объединение двух графических объектов и более в одной графической области. Они просто передаются в качестве аргументов для вызова `gr2d` или `draw2d`. Локальные графические опции могут быть заданы между графическими объектами. Напомним, что эти локальные параметры влияют на все те графические объекты, которые расположены справа от него. Опция `key` может использоваться для добавления имени функции или других графических объектов в легенду картинки.

Большая часть того, что говорилось о 2D-графиках, также применяется и для 3D-графиков. Трехмерные картинки могут быть построены

```
(%i3)   draw2d(title = "Two plots", xlabel = "x", ylabel = "y",
grid=true, color = red, key = "A sinus", explicit(sin(x), x, 0,
4*%pi), line_width = 3, color = blue, line_type = dots, key = "A
cosinus", explicit(cos(2*x), x, 0, 4*%pi), line_width = 5)$
```

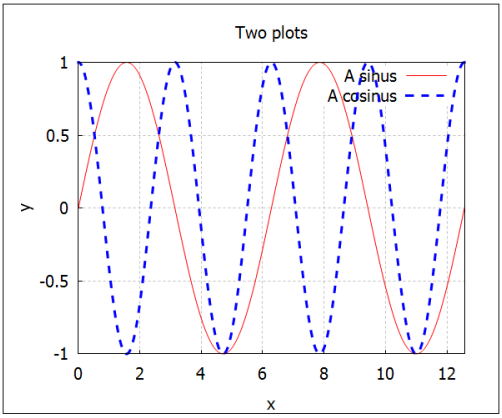


Рис. 5.20

с помощью конструктора сцены `gr3d`. Его синтаксис похож на синтаксис `gr2d`. Основным отличием этих конструкторов естественно является наличие дополнительной третьей переменной. Здесь есть пара доступных графических объектов. В частности, для построения 3D-графиков функций имеем следующие типы графиков :

- `explicit`: явная функция, заданная формулой, в которую входят входные параметры, например, `x^2+y^2`.
- `implicit`: неявная функция, определяемая как решения уравнения в форме $F(x, y, z) = 0$, например, `x^2 + y^2 + z^2 - 1=0`.
- `points`: рисование поверхности по точкам.

Кривые и поверхности в трехмерном пространстве могут быть представлены следующими графическими объектами:

- `parametric`: параметрическая кривая, например, `[sin(t*%pi), cos(t*%pi),t]`.
- `parametric_surface`: параметрическая поверхность.

Помимо этого есть некоторые низкоуровневые графические объекты для добавления линий или аннотирования графиков (см. ниже). Для получения подробной информации используйте онлайн-справку системы Maxima, например, `? points`.

```
(%i4)   draw(gr2d(title = "Sinus", color = red, key = "sin(x)",
grid=[3,3], explicit(sin(x), x, 0, 4*%pi), line_width =
3), gr2d(title = "Cosinus", color = blue, line_type = dots,
key="cos(4*x)", grid=[2,2], explicit(cos(4*x), x, 0, 4*%pi),
line_width = 5) )$
```

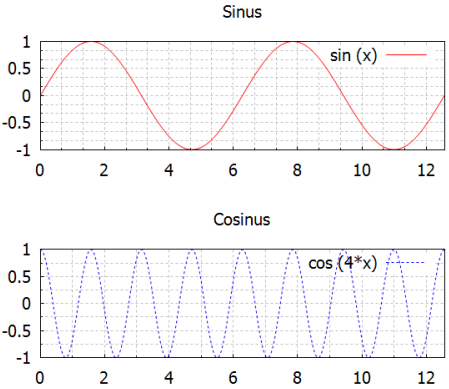


Рис. 5.21

```
(%i5)   draw3d(explicit(2^(-x^2 + 0.2*y^2), x, -3, 3, y, -2, 2),
surface_hide=true)$
```

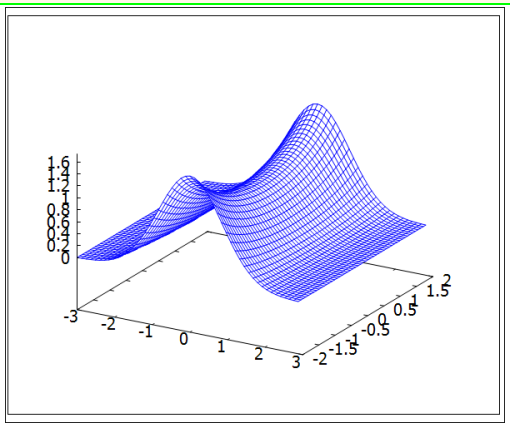


Рис. 5.22

- label: помещает текст на поле графика.
- mesh: рисует прямоугольную сетку в 3D.
- quadrilateral: четырехугольник.
- triangle: треугольник.
- tube: цилиндр.
- vector: стрелка.

Существует множество вариантов управления внешним видом трехмерной графики, но в рамках данного пособия приводится только часть из них. Остальные см. в более полном руководстве по системе Maxima.

Напомним, что при построении 3D-графиков "мышь" можно использовать для их поворота.

Команда	*Описание*
explicit(func,x,xmin,xmax, y,ymin,ymax)	График функции двух переменных

Опция	*По умолчанию*	*Описание*
surface_hide	false	Если true, то дальний план не отображается
enhanced3d	none	Получить цветную поверхность
colorbox	true	Если true, то рисование цветовой гаммой

```
(%i6)   draw3d(enhanced3d=true, palette = [red, orange, yellow,
blue], explicit(2^(-x^2 + 0.2*y^2), x, -3, 3, y, -2, 2))$
```

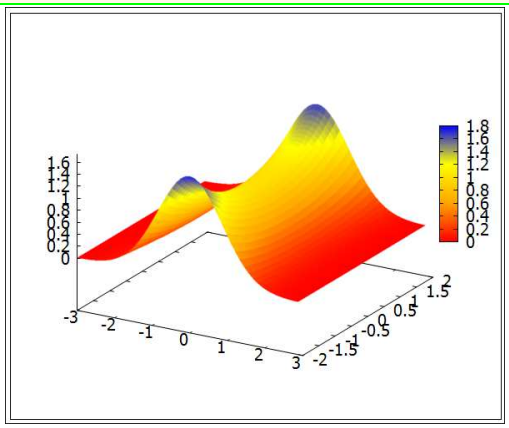


Рис. 5.23

Аналогично графикам одномерных функций можно получать графики двумерных функций с использованием графического объекта explicit.

Заметим, что в отличие от `plot3d` рисуется только сетка ("проволочная графика").

Команда	*Описание*
`implicit(func,x,xmin,xmax, y,ymin,ymax,z,zmin,zmax)`	График неявной функции трех переменных

Неявные функции определяются равенством в форме $F(x, y, z) = 0$. Они строятся аналогично явным функциям, но в отличие от `explicit`, `implicit` требуют, чтобы область для всех трех задействованных переменных была указана. В примере на рис. 5.24 показана поверхность неявной функции. Значение параметра `extended3d` установлено равным `true`, поэтому цвета подобластей поверхности зависят от значений z.

```
(%i7)   draw3d(view=[60,45], enhanced3d=true, palette = [red,
orange, yellow, blue], nticks=40, implicit(x^2/2 + sin(y) = z^2/2,
x, -3, 3, y, -2*%pi, 2*%pi, z, -2, 2))$
```

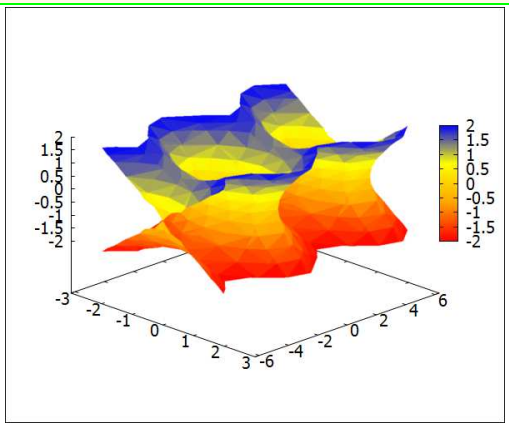


Рис. 5.24

Команда	*Описание*
`parametric(xfun,yfun, zfun,t,tmin,tmax)`	Рисование параметрических 2D- и 3D-объектов
`parametric_surface (xfun,yfun,zfun,s,smin, smax,t,tmin,tmax)`	Рисование параметрической 3D-поверхности

Графический объект `parametric` так же, как `plot3D`, позволяет рисовать кривые в 3D. Однако в отличие от двумерного графического объекта

теперь требуется дополнительный аргумент **zfun** для третьего измерения (рис. 5.25).

```
(%i8)   draw3d(nticks = 500, line_width = 3, parametric(t*sin(12*t),
t*cos(12*t),t,  t,0,%pi))$
```

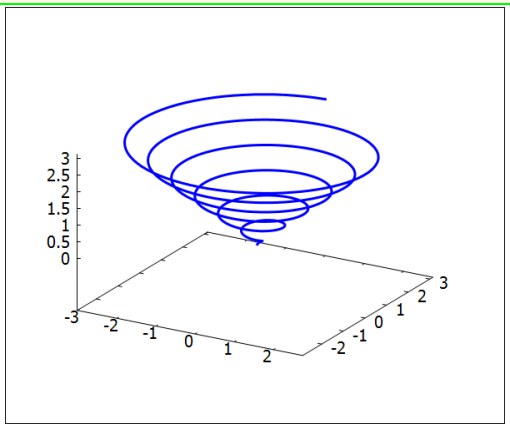


Рис. 5.25

Параметрическая 3D-поверхность определяется функциями для координат x, y и z соответственно. Каждая из них является функцией двух переменных (называемых *параметрами*). Такие поверхности представлены графическим объектом **parametric_surface** (рис. 5.26).

Опция	*По умолчанию*	*Описание*
contour	**none**	Получение линий уровня
contour_levels	5	Число или положение линий уровня

Линии уровня функции $z = f(x, y)$ – это кривые, уравнения которых есть $f(x, y) = C$ для некоторых констант C. В пакете **draw** могут быть созданы линии уровня для функций **explicit** с использованием графической опции **contour**. Возможные значения:

- **none**: линии уровня отсутствуют.
- **base**: линии уровня проецируются на плоскость OXY.
- **surface**: линии уровня проецируются на поверхность.
- **both**: рисуются два типа линий уровня: проекции на плоскость OXY и на поверхность (рис. 5.27).
- **map**: линии уровня проецируются на плоскость OXY, а точка обзора устанавливается только сверху.

```
(%i9)   draw3d(enhanced3d = true, palette = [red, orange, yellow,
blue], parametric_surface((3 + sin(s) + cos(t)) * cos(2*s), (3 +
sin(s) + cos(t)) * sin(2*s), sin(t) + 2*cos(s), s, 0, 2*%pi, t, 0,
2*%pi))$
```

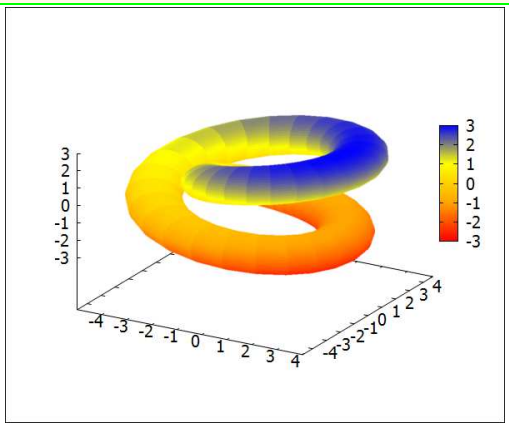


Рис. 5.26

```
(%i10)   draw3d(explicit(x^2 + y^2, x, -4, 4, y, -4, 4),
contour=both, contour_levels=20)$
```

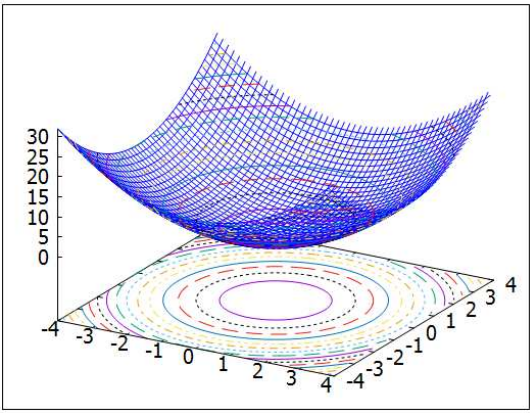


Рис. 5.27

Количество линий уровня можно контролировать с помощью графической опции `contour_levels` (рис. 5.28). Это может быть либо одно число для количества уровней, либо список или множество чисел (подробности см. в справке).

```
(%i11)   draw3d(explicit(x^4 - 96*x^2 - y^4 + 100*y^2, x, -30, 30,
y, -30, 30), contour=map, contour_levels=25)$
```

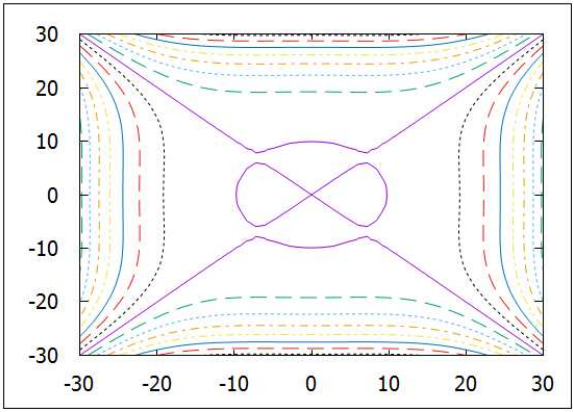


Рис. 5.28

Графический объект	*Описание*
`points([[x_coord,y_coord, z_coord],...])`	Точка в пространстве
`points([x_coords,...], [y_coords,...],[z_coords,...])`	Точки в пространстве

Конструктор `draw3d` позволяет строить графики дискретных функций в трех измерениях. Команды и опции те же, что и для `draw2d`. В следующем примере (рис. 5.29) построены точки с координатами (1, 5, 2), (2, 4, 4), (3, 2, 3), (4, 7, 1), (5, 4, 6) (красные круги) и 10 случайно выбранных точек.

Пакет `draw` предлагает широкий набор опций создания и управления графиками. Ниже описываются только самые важные из них. Для знакомства с полным списком опций см. раздел "Draw" в руководстве по системе.

Графические параметры всегда указываются в виде пар "ключевое слово = значение" (`key=value`).

```
(%i12)   xc: [1, 2, 3, 4, 5]$ yc: [5, 4, 2, 7, 4]$ zc: [2, 4,
3, 1, 6]$ r1: makelist(1 + random(7), k, 1, 10)$ r2: makelist(1
+ random(7), k, 1, 10)$ r3: makelist(1 + random(7), k, 1, 10)$
draw3d(color = red, point_type = filled_circle, point_size = 1.5,
points(xc, yc, zc), color = blue, point_type = filled_diamant,
point_size = 2, points(r1, r2, r3))$
```

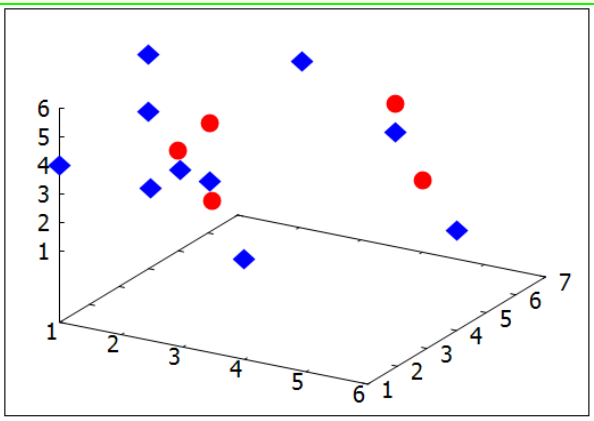


Рис. 5.29

Команда	*Описание*
`set_draw_defaults(opts,...)`	Установить графические опции пользователя

Функция `set_draw_defaults` позволяет установить глобальные значения по умолчанию для сеанса Maxima. Вызов этой функции без аргументов удаляет пользовательские настройки по умолчанию.

Следующий фрагмент кода меняет цвет по умолчанию для графиков на красный и включает построение сетки на плоскости OXY.

```
(%i13)   set_draw_defaults(color=red, grid=true)$
```

Графический объект	*Описание*
`label(["Text",x_coord, y_coord],...])`	Вписывание метки (или меток) в 2D
`label(["Text",x_coord, y_coord,z_coord],...])`	Вписывание метки (или меток) в 3D

Можно использовать графический объект `label`, чтобы написать любой текст в заданном месте графика.

Опция	*По умолчанию*	*Описание*
`title="Text"`	`""`	Главный заголовок картинки
`xlabel="Text"`	`""`	Метка для оси абсцисс
`ylabel="Text"`	`""`	Метка для оси ординат
`zlabel="Text"`	`""`	Метка для оси аппликат
`xtics`	`"auto"`	Оцифровка оси абсцисс
`ytics`	`"auto"`	Оцифровка оси ординат
`ztics`	`"auto"`	Оцифровка оси аппликат

Опции `xlabel`, `ylabel` и `zlabel` вставляют обозначения для соответствующих осей. Параметр `title` добавляет заголовок для картинки. Опции `xtics`, `ytics` и `ztics` определяют способ рисования отметок на осях. Правила их задания и возможные значения можно увидеть в справке.

Опция	*По умолчанию*	*Описание*
`key="Text"`	`""`	Имя функции в легенде

Опция `key` вставляет `Text` для аннотирования данной функции в легенде. Для каждого графика функции опция `key` должна быть задана перед графическим объектом. Запись в легенду для следующих объектов может быть подавлена установкой `key = ""`.

Опция	*По умолчанию*	*Описание*
`xaxis`	`"false"`	Если `true`, то ось x будет нарисована
`yaxis`	`"false"`	Если `true`, то ось y будет нарисована
`zaxis`	`"false"`	Если `true`, то ось z будет нарисована (только для 3D)
`grid`	`"false"`	Если `true`, то сетка будет нарисована на плоскости "OXY"

По умолчанию на картинках нет координатных осей. Их можно нарисовать, добавив параметры `xaxis = true`, `yaxis = true` и/или `zaxis = true`. Существуют также дополнительные параметры, которые управляют стилем, цветом соответствующей оси и т.д. Подробности см. в справочнике.

Опция	*По умолчанию*	*Описание*
`x=[x_min,x_max]`	`"auto"`	Диапазон для оси x
`y=[y_min,y_max]`	`"auto"`	Диапазон для оси y
`z=[z_min,z_max]`	`"auto"`	Диапазон для оси z

Эти параметры устанавливают диапазон области построения для каждого направления.

```
(%i14)   draw3d(colorbox = false, palette = [20, 5, 30], title =
"Ogo-o!", enhanced3d = true, xu_grid = 300, yv_grid = 300, zrange
= [-1, 4], xlabel = "x", ylabel = "y", xtics = none, ytics = none,
explicit(x^2 - y^2, x, -2, 2, y, -2, 2))$
```

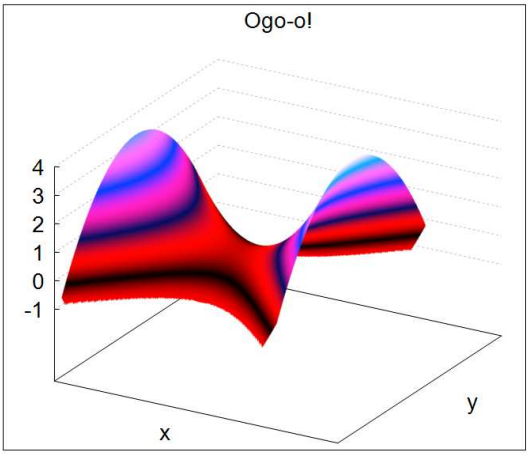


Рис. 5.30

Заметим, что в случае (явных или неявных) функций имя этих графических опций не зависит от конкретного имени переменных для функции, т.е. `x`, `y`, `z` относятся именно к первой, второй и третьей координате (направлению).

Опция	*По умолчанию*	*Описание*
`nticks=integer`	`"29"`	Исходное число точек для построения 2D-графиков

При построении явных или параметрических функций мы можем управлять гладкостью нарисованной кривой, устанавливая количество точек с помощью опции `nticks`. Для явных графиков это просто начальное количество точек, которое будет автоматически увеличено, когда данная функция имеет большие вариации. Для параметрических графиков это фактическое количество точек, которые будут показаны для графика. Это может привести к кривой, которая выглядит не так гладко, как хотелось бы. В этом случае рекомендуется увеличить `nticks`.

```
(%i15)  draw2d(grid = true, color = light_red, line_width = 2,
explicit(x^2, x, -2, 2), color = blue, line_width = 2, implicit(x^2
+ y^2 = 4, x, -2, 2, y, -2, 2), color = forest_green, point_size =
1, point_type = filled_circle, points([ [-1, -1], [2, 2] ]), color
= dark_red, line_width = 2, font = "Arial-Bold", font_size = 16,
label(["A",0,1]), nticks=100, parametric(t, t^(1/3), t, -2, 2) )$
```

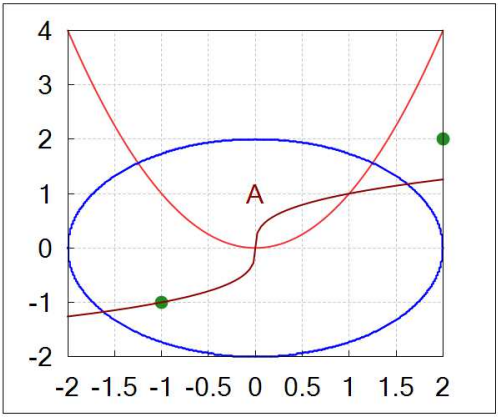


Рис. 5.31

Опция	*По умолчанию*	*Описание*
`xu_grid=integer`	`"30"`	Число точек сетки по первой координате для построения 3D-графиков
`yv_grid=integer`	`"30"`	Число точек сетки по второй координате для построения 3D-графиков

Для явных функций и параметрической поверхности в 3D вычисляется сетка по точкам. Количество точек по каждой координате можно указать с помощью двух опций `xu_grid` и `yv_grid`.

Опция	*По умолчанию*	*Описание*
`ip_grid=[integer, integer]`	`"[50,50]"`	Число точек сетки при первой попытке построения неявных графиков
`ip_grid_in=[integer, integer]`	`"[5,5]"`	Число точек сетки при второй попытке построения неявных графиков

Графики для неявных графических объектов создаются в два этапа. На первом этапе вычисляется грубая структура неявного графика. На

```
(%i16)   draw3d(user_preamble = "unset key;", xu_grid = 100, yv_grid
= 100, xaxis = true, xaxis_type = solid, xaxis_width = 2, xtics =
0.5, yaxis = true, yaxis_type = solid, yaxis_width = 2, ytics = 0.5,
zaxis = true, zaxis_type = solid, zaxis_width = 2, contour = map,
contour_levels = [-6,0. 5,8], explicit(y^3*(y - 2)*(y + 2) - x*(x +
1)*(x + 3) + x^2*y, x, -3.5, 4.5, y, -3.5, 3.5) )$
```

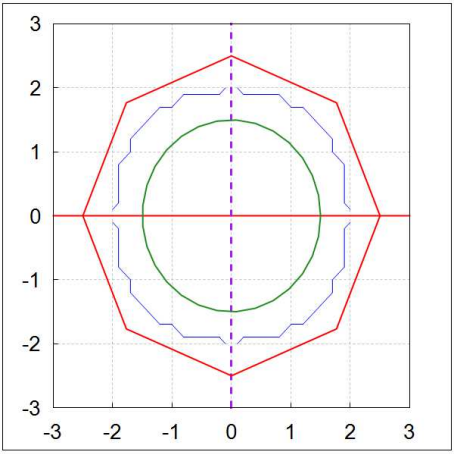


Рис. 5.32

втором этапе эта структура уточняется с помощью локальных сеток. Таким образом, гладкость неявных графических объектов контролируется с помощью `ip_grid` и `ip_grid_in`. В некоторых случаях может быть необходимо увеличить значения из `ip_grid` для быстро меняющихся функций.

Опция	*По умолчанию*	*Описание*
`terminal`	`'screen`	Выбор типа вывода
`file_name="file"`	`"maxima_out"`	Файл для графического вывода

Опция `terminal` позволяет настроить устройство вывода для графики. По умолчанию вся графика рисуется на экране. Однако рисунки также можно экспортировать во внешние графические файлы некоторых форматов, установив один из них:

- `'animated_gif`
- `'aquaterm`
- `'eps`
- `'eps_color`

```
(%i17)   Z: y^3*(y - 2)*(y + 2) - x*(x + 1)*(x - 3) + x^2*y$ Fx:
diff(Z,x)$ Fy: diff(Z,y)$ set_draw_defaults(dimensions = [800, 704],
xrange = [-1.8, 3.2], yrange = [-2.2, 2.2], zrange = [-7, 9], grid
= true)$ CP: solve([Fx, Fy], [x, y])$ draw2d(xtics = 0.5, ytics =
0.5, ip_grid = [100, 100], color = blue, makelist(implicit(Z=k,
x, -1.8, 3.2, y, -2.2, 2.2), k, -6, 8, 1), ip_grid = [50, 50],
color = dark_green, makelist(implicit(Z = k, x, 1.5, 3.2, y, 0.3,
2.0), k, [7.44, 7.6, 7.8, 7.83]), makelist(implicit(Z = k, x,
-1.5, 1.25, y, -0.75, 2), k, [-0.25, -0.5, -0.75, -0.85, -0.88,
-0.91]), color = dark_blue, point_type = filled_circle, point_size =
0.5, points(makelist([rhs(cp[1]), rhs(cp[2])], cp, CP)), makelist(
label([ascii(64 + k), rhs(CP[k][1]) + 0.15, rhs(CP[k][2])]), k, 1,
length(CP)) )$
```

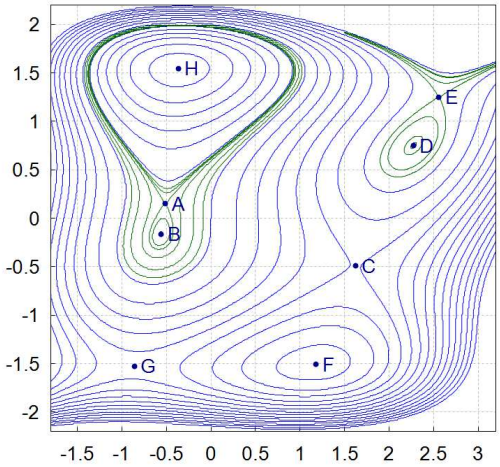


Рис. 5.33

- 'gif
- 'jpg
- 'pdf
- 'pdfcairo
- 'png
- 'pngcairo
- 'svg
- 'wxt

Выходные данные затем сохраняются в файле, заданном параметром `file_name`. Соответствующее расширение автоматически добавляется к имени файла.

Пример ниже показывает, как сохраняем график функции косинус в формате `eps_color` в файле `d:\cosine.eps`. Заметим, что добавлять расширение `.eps` не требуется.

```
(%i18)   draw2d(explicit(cos(x), x, -2*%pi,2*%pi), terminal =
'eps_color, file_name = "d:\cosine")$
```

```
(%i19)   draw3d( dimensions = [900, 900], view = [60, 110],
enhanced3d = false, proportional_axes = xy, axis_3d=false, xaxis
= true, yaxis = true, zaxis = true, xaxis_type = solid, zaxis_type =
solid, yaxis_type = solid, xaxis_color = black, zaxis_color = black,
yaxis_color = black, xaxis_width = 2, yaxis_width = 2, zaxis_width
= 2, ylabel = "x", xlabel = "y", xyplane = 0, surface_hide = true,
color = red, key = "Hyperbolic Paraboloid", explicit(x^2 - y^2, x,
-4, 4, y, -4, 4), color=blue, key = "Paraboloid", explicit(x^2 +
y^2-2, x, -4, 4, y, -4, 4) )$
```

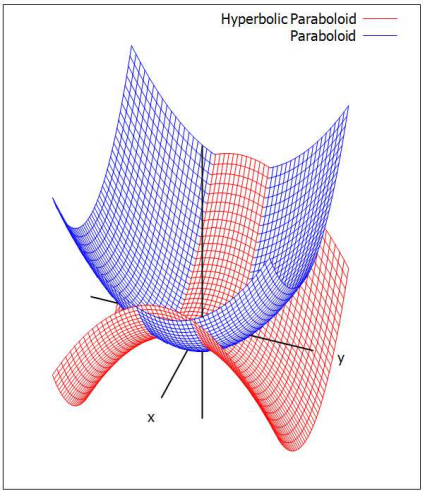


Рис. 5.34

Опция	*По умолчанию*	*Описание*
`point_type`	`plus` (1)	Тип точки
`point_size`	1	Размер точки

Опция	*По умолчанию*	*Описание*
line_type	solid	Тип линии (solid или dots)
line_width	1	Толщина линии
color=<имя>	blue	Цвет линий, точек и меток
surface_hide	false	Если true, то задний фон не просвечивает
enhanced3d	none	Цветная поверхность
palette	color	Показывает, как отобразить уровни серого в цветные компоненты

Построение точек и линий может контролироваться различными опциями. Цвета могут быть установлены с помощью опции color и заданы как именами, так и числами в шестнадцатиричном RGB-коде.

Системные названия цветов

white	black	gray0	grey0
gray10	grey10	gray20	grey20
gray30	grey30	gray40	grey40
gray50	grey50	gray60	grey60
gray70	grey70	gray80	grey80
gray90	grey90	gray100	grey100
gray	grey	light_gray	light_grey
dark_gray	dark_grey	red	light_red
dark_red	yellow	light_yellow	dark_yellow
green	light_green	dark_green	spring_green
forest_green	sea_green	blue	light_blue
dark_blue	midnight_blue	navy	medium_blue
royalblue	skyblue	cyan	light_cyan
dark_cyan	magenta	light_magenta	dark_magenta
turquoise	light_turquoise	dark_turquoise	pink
light_pink	dark_pink	coral	light_coral
orange_red	salmon	light_salmon	dark_salmon
aquamarine	khaki	dark_khaki	goldenrod
light_goldenrod	dark_goldenrod	gold	beige
brown	orange	dark_orange	violet
dark_violet	plum	purple	

По умолчанию трехмерные графические объекты отображаются в виде прозрачных сеток. Чтобы поверхность была непрозрачной, следует воспользоваться параметром surface_hide = true. Для цветной поверхности необходимо использовать опцию extended3d, чтобы включить цветные поверхности, и опцию palette, чтобы установить конкретный желаемый набор цветов (см. справочник).

6. ЭЛЕМЕНТАРНАЯ МАТЕМАТИКА

6.1. Упрощения и преобразования арифметических, алгебраических, тригонометрических, показательных выражений

Рассмотрим простые средства системы Maxima по упрощению и прочим преобразованиям выражений. В частности, рассмотрим способы автоматического раскрытии скобок и вынесения за скобки; упрощения как арифметических операций с некоторыми операндами, так и выражений с участием степенных, показательных и логарифмических функций; обработки тригонометрических выражений и др. Все эти средства призваны облегчить читаемость математических формул и упростить их восприятие.

Для преобразования рационального выражения к канонической форме (см. 8.4.1) можно использовать функцию `rat`, которая раскрывает все скобки, а затем приводит все дроби к общему знаменателю, приводит подобные и сокращает числители и знаменатели на общие множители. Кроме того, эта функция переводит все числа из конечной десятичной записи в рациональную форму.

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

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

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

$$(\%o2)\quad /R/\quad \frac{x^2-3\,x+4}{x^2+2\,x}$$

Если требуется вычислить числовое значение полученного выражения, то можно применить функцию `at`, указав в скобках выражение или его имя и значение переменной.

```
(%i3)   at(%,x=3);
```

$$(\%o3)\qquad \frac{4}{15}$$

Более компактный вид выражению можно придать с помощью функции `factor`.

```
(%i4)  factor(a*x^2-2*a*x+a);
```
$$(\%o4)\qquad a\,(x-1)^2$$

Чтобы раскрыть скобки в алгебраическом выражении, нужно применить функцию `expand`.

```
(%i5)  expand((sqrt(x)-1)^2+(sqrt(x)+1)^2);
```
$$(\%o5)\qquad 2\,x+2$$

Несложное упрощение выражения можно выполнить с помощью функции `ratsimp`.

```
(%i6)  ratsimp(3/(x-1)+x/(x-1)^2-x^2/(x-1)^3);
```
$$(\%o6)\qquad \frac{3\,x^2-7\,x+3}{x^3-3\,x^2+3\,x-1}$$

Функция `fullratsimp` используется для более серьезных упрощений. Она последовательно применяет к аргументу функцию `ratsimp`, а также некоторые другие преобразования, причем до тех пор, пока аргумент не перестанет упрощаться.

```
(%i7)  r: (x^(n/2)-1)^2*(x^(n/2)+1)^2/(x^n-1);
```
$$(r)\qquad \frac{(x^{n/2}-1)^2\,(x^{n/2}+1)^2}{x^n-1}$$

```
(%i8)  ratsimp(r);
```
$$(\%o8)\qquad \frac{x^{2n}-2\,x^n+1}{x^n-1}$$

```
(%i9)  fullratsimp(r);
```
$$(\%o9)\qquad x^n-1$$

Для разложения рациональных дробей на элементарные дроби можно воспользоваться функцией `partfrac`.

```
(%i10)  partfrac((x^3-x^2+1)/(x^5+2*x^4+2*x^3+x^2),x);
```
$$(\%o10)\qquad \frac{3\,x+2}{x^2+x+1}-\frac{1}{x+1}-\frac{2}{x}+\frac{1}{x^2}$$

Для преобразования выражений, содержащих логарифмические, показательные и степенные функции, применяют функцию `radcan`.

```
(%i11)  ((y+a)^(5/2)-sqrt((y+a)^3*(y-b)))/((y+a)*sqrt((y+a)*(y-b)));
```
$$(\%o11)\qquad \frac{(y+a)^{5/2}-\sqrt{(y+a)^3\,(y-b)}}{(y+a)\,\sqrt{(y+a)\,(y-b)}}$$

```
(%i12)  radcan(%);
```

$$(\%o12) \qquad -\frac{\sqrt{y-b}-y-a}{\sqrt{y-b}}$$

Иногда эту функцию необходимо применять последовательно несколько раз.

Для упрощения тригонометрических выражений к некоторым именам уже использовавшихся функций добавляется приставка `trig` и применяется функция с полученным именем. Для получения желаемого результата ее можно комбинировать с функциями `ratsimp`, `fullratsimp`, `radcan` и другими.

```
(%i13)  csc(x)^2*cot(x)+((1-cos(x)^2)*sin(x))/sin(x)^2;
```

$$(\%o13) \qquad \frac{1-\cos(x)^2}{\sin(x)}+\cot(x)\,\csc(x)^2$$

```
(%i14)  trigsimp(%);
```

$$(\%o14) \qquad \frac{\sin(x)^4+\cos(x)}{\sin(x)^3}$$

Для того чтобы раскрыть скобки, ограничивающие аргументы типа сумм в тригонометрических выражениях, рекомендуется применять функцию `trigexpand`, которая использует, в частности, формулы преобразования сумм углов для представления введенного выражения в как можно более простом виде.

```
(%i15)  trigexpand(8*sin(x+y)*cos(x)^3);
```

$$(\%o15) \qquad \cos(x)^3 * (\cos(x)\,\sin(y)+\sin(x)\,\cos(y))$$

Функция `trigreduce` – в некотором смысле обратная операция.

```
(%i16)  expand(trigreduce(%));
```

$$(\%o16) \qquad \sin(y+4\,x)+3\,\sin(y+2\,x)+\sin(y-2\,x)+3\,\sin(y)$$

6.2. Вычисление конечных сумм и произведений

Для вычисления

$$\sum_{k=k_1}^{k_2} r_k = r_{k_1} + r_{k_1+1} + \ldots + r_{k_2}$$

применяют функцию `sum(r_k, k, k_1, k_2)`. Невычисляемая форма функции `'sum` выводится на экран в математической нотации. Слагаемое r_k должно вычисляться для каждого значения индекса k. Результатом является явная форма суммы.

```
(%i1)  sum(k^2, k, 1, 10);
(%o1)      385
```

```
(%i2)  sum(a[k], k, 1, 10);
(%o2)      a10 + a9 + a8 + a7 + a6 + a5 + a4 + a3 + a2 + a1
```

```
(%i3)  sum(a[k], k, 1, n);
                 n
(%o3)            ====
                 \     a
                 /      k
                 ====
                 k = 1
```

```
(%i4)  'sum(k^2, k, 1, n)=ev(sum(k^2, k, 1, n),simpsum);
                 n
                 ====
                 \      2
(%o4)            /     k  = (2 n^3 + 3 n^2 + n)/6
                 ====
                 k = 1
```

Функция `product(r_k, k, k_1, k_2)` вычисляет конечное произведение

$$\prod_{k=k_1}^{k_2} r_k = r_{k_1} \cdot r_{k_1+1} \cdot \ldots \cdot r_{k_2}$$

и сходна по интерпретации параметров с функцией `sum`.

```
(%i5)  product(k^2, k, 1, 10);
(%o5)      13168189440000
```

```
(%i6)  product(a[k], k, 1, 10);
(%o6)      a1 a2 a3 a4 a5 a6 a7 a8 a9 a10
```

```
(%i7)  product(a[k], k, 1, n);
                 n
(%o7)            ====
                 \     a
                 /      k
                 ====
                 k = 1
```

```
(%i8)  'product(k, k, 1, n)=ev(product(k, k, 1, n),simpprod);
                 n
                 ====
                 \
(%o8)            /     k = n!
                 ====
                 k = 1
```

```
(%i1)   wxplot2d([x,abs(x),x^2,x^3], [x,-1,1], [y,-1,1]);
(%o1)
```

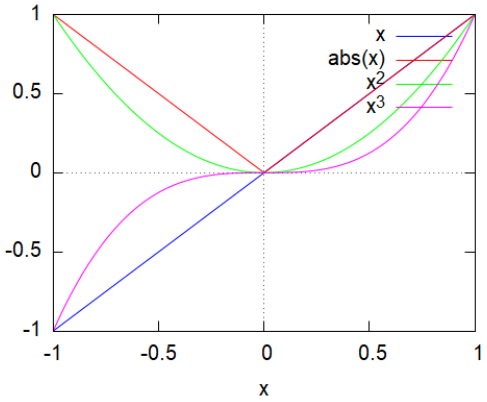


Рис. 6.1

Функция `arithsum(a,d,n)` находит сумму арифметической прогрессии, функция `geosum(a,r,n)` – геометрической (`n` может выть равно `inf`). Для использования требуется загрузка пакета `functs`.

6.3. Построение графиков простейших функций

Для данной процедуры пригодна функция `plot2d`. Используя ее, можно на одном поле можно построить графики сразу нескольких функций.

Заметим, что для формирования рис. 6.1 использована реализация функции `plot2d` для среды wxMaxima.

Эту же функцию применим для построения графиков других элементарных функций: тригонометрических, обратных тригонометрических, показательных, логарифмических (рис. 6.2, 6.3).

6.4. Нахождение решений алгебраических уравнений и систем

Простые алгебраические уравнение и системы таких уравнений могут быть решены при помощи функции `solve`. Например, вызовы функции `solve` для решения одного уравнения с одним неизвестным и числовыми коэффициентами и результат выглядят так:

```
(%i2)   wxplot2d([x*sin(x)/8,atan(x),x/8,-x/8], [x,-4*%pi,4*%pi],
[y,-%pi/2,%pi/2]);
(%o2)
```

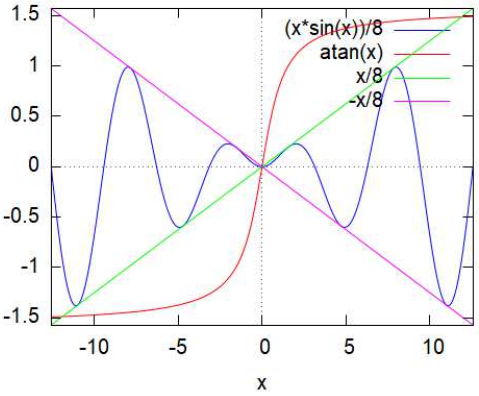


Рис. 6.2

```
(%i3)   wxplot2d([log(x+1/2),3*exp(-x)], [x,0,5], [y,-1,3]);
(%o3)
```

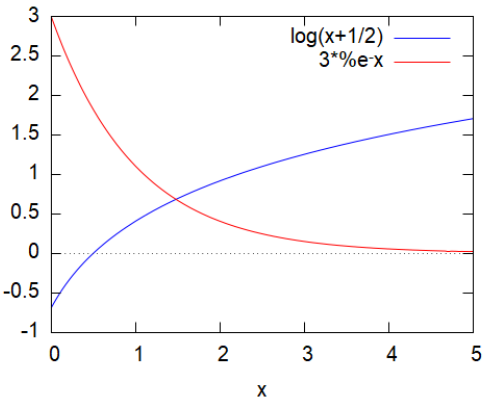


Рис. 6.3

```
(%i1)   solve(5*x+1);
(%o1)        [x = -1/5]
```

```
(%i2)   solve(x^2-5*x+4);
```
(%o2) $[x = 1, x = 4]$

Если же коэффициенты уравнения включают какие-либо символы, структура вызова несколько меняется:

```
(%i3)   solve(a*x+b=c,x);
```
(%o3) $[x = \frac{c-}{a}]$

```
(%i4)   solve(a*x+b/x=c,x);
```
(%o4) $[x = -\frac{\sqrt{c^2-4\,a\,b}-c}{2\,a}, x = \frac{\sqrt{c^2-4\,a\,b}+c}{2\,a}]$

Использование функции `solve` для решения системы уравнений с несколькими неизвестными возможно так:

```
(%i5)   solve([x+3*y-z=-3, x+y+z=1, 2*x-y-z=2],[x,y,z]);
```
(%o5) $[[x = 1, y = -1, z = 1]]$

Здесь одно решение, но в общем случае решений у системы алгебраических уравнений может быть больше.

```
(%i6)   solve([x^2/a^2+y^2/b^2=1, 4*x^2/a^2-4*y^2/b^2=1], [x,y]);
```
(%o6) $[[x = -\frac{\sqrt{5}\,a}{2^{3/2}}, y = -\frac{\sqrt{3}\,b}{2^{3/2}}], [x = -\frac{\sqrt{5}\,a}{2^{3/2}}, y = \frac{\sqrt{3}\,b}{2^{3/2}}],$
$[x = \frac{\sqrt{5}\,a}{2^{3/2}}, y = -\frac{\sqrt{3}\,b}{2^{3/2}}], [x = \frac{\sqrt{5}\,a}{2^{3/2}}, y = \frac{\sqrt{3}\,b}{2^{3/2}}]]$

В последнем примере решений несколько, и Maxima выдает результат в виде списка подсписков.

Функция `solve` применима и для решения тригонометрических уравнений. При этом в случае бесконечного множества решений таких уравнений выдается только одно порождающее решение из семейства и соответствующее сообщение.

```
(%i7)   solve(4*tan(x)^2-(4*sqrt(3)+4)*tan(x)+4*sqrt(3)=0,x);
        solve: using arc-trig functions to get a solution.
        Some solutions will be lost.
```
(%o7) $[x = \frac{\pi}{3}, x = \frac{\pi}{4}]$

Наряду с действительными решениями в численной или символьной форме возможно получение комплексных решений в обоих формах:

```
(%i8)  solve(x^2+4=0);
(%o8)      [x = -2 %i, x = 2 %i]
```

```
(%i9)  solve(x^2+4*a^2=0,x);
(%o9)      [x = -2 %i a, x = 2 %i a]
```

6.5. Решение алгебраических неравенств

Несмотря на некоторые публикации, система Maxima все-таки включает инструмент – функцию `solve_rat_ineq`, позволяющую решать несложные неравенства относительно одной переменной школьного уровня. Эти неравенства могут включать дробно-рациональные функции, коэффициенты которых – целые и рациональные числа, а также числа с плавающей точкой. Результат работы функции `solve_rat_ineq` возвращается в виде списка простых неравенств. При этом решение задачи – это объединение точечных множеств, соответствующих неравествам из списка. Алгоритм работы `solve_rat_ineq` использует функцию `algsys`. При этом в описании функции `solve_rat_ineq` рекомендуется, что к решениям, содержащим приближенные числа с плавающей точкой, следует относиться с осторожностью.

```
(%i1)  load("solve_rat_ineq")$
```

```
(%i2)  solve_rat_ineq((x-1)^2*(x+1)^2>=0)
(%o2)     all
```

```
(%i3)  solve_rat_ineq((x-1)^2*(x+1)^2<0)
(%o3)     []
```

```
(%i4)  solve_rat_ineq( (x-1)/(x+1) > 0 );
(%o4)      [[x < -1], [x > 1]]
```

```
(%i5)  solve_rat_ineq( (x-1)^2*(x+1) > 1+x );
(%o5)      [[x > -1, x < 0], [x > 2]]
```

```
(%i6)  solve_rat_ineq( (x^2+4*x+3)/(x^2-4*x+3) > 0);
(%o6)      [[x < -3], [x > -1, x < 1], [x > 3]]
```

```
(%i7)  solve_rat_ineq( x^7+x^3 > 1/x );
(%o7)      [[x > -0.8866517524235645, x < 0], [x > 0.8866517524235645]]
```

Упрощение неравенств возможно с помощью пакета `ineq`.

7. ДИСКРЕТНАЯ МАТЕМАТИКА

7.1. Теория чисел. Цепные дроби. Факторизация целых чисел

Оператор	*Описание*
`cf(expr)`	Приближение для цепной дроби
`cfdisrep(list)`	Формирование арифметического выражения из цепной дроби
`fib(n)`	`n`-е число Фибоначчи
`ifactors(n)`	Разложение целого числа на простые множители
`igcdex(n,k)`	Построение списка `[a,b,u]`, где `u` – НОД целых `n` и `k`, причем `u=a*n+b*k`
`inrt(x,n)`	Вычисление целого числа `k` такого, что `k^n <= |x| < (k+1)^n`
`isqrt(x)`	Вычисление целого числа `k` такого, что `k^2 <= |x| < (k+1)^2`
`lcm(expr_1,...,expr_n)`	Вычисление НОК выражений, в т.ч. и чисел
`next_prime(n)`	Вычисление первого простого числа после `n`
`prev_prime(n)`	Вычисление наибольшего простого числа, не превышающего `n`
`primep(n)`	Предикат: `true`, если `n` – простое число
`primes(n1,n2)`	Вычисление всех простых чисел из промежутка `[n1,n2]`
`zeta(n)`	Значение дзета-функции Римана при целом `n`

Функция `cf(expr)` вычисляет приближение для цепной дроби. Параметр `expr` – это выражение, содержащее цепные дроби, квадратные корни из целых чисел и элементы множества действительных чисел (целые и рациональные числа, обычные и длинные числа с плавающей точкой). `cf` вычисляет точные разложения для рациональных чисел, но приближенные усекаются с точностью до `ratepsilon` для обычных и до `10^(-fpprec)` для длинных чисел. Операнды в `expr` могут комбинироваться с помощью арифметических операторов. Maxima не выполняет операций с цепными дробями вне пределов функции `cf`.

Функция `cf` возвращает цепную дробь, представленную в виде списка: дробь `a + 1/(b + 1/(c + ...))` записывается в виде `[a, b, c, ...]`. Элементы списка `a`, `b`, `c`, ... должны быть такими, чтобы их величины были целыми числами. `expr` может содержать квадратные корни типа `sqrt(n)`, где `n` – целое число. Число термов непрерывной дроби, полученной с помощью функции `cf`, совпадает со значением переменной `cflength`, умноженной на период дроби [16].

```
(%i1)   a0: (1 + sqrt(7))/2$ cflength: 1$ a1: cf(a0)$ b1:
rat(cfdisrep(a1))$ [a1,b1];
```

(%o1) $[[1, 1, 4, 1, 1, 2], \frac{51}{28}]$

```
(%i2)   cflength: 2$ a2: cf(a0)$ b2: rat(cfdisrep(a2))$ [a2,b2];
```

(%o2) $[[1, 1, 4, 1, 1, 1, 4, 1, 1, 2], \frac{813}{446}]$

```
(%i3)   cflength: 3$ a3: cf(a0)$ b3: rat(cfdisrep(a3))$ [a3,b3];
```

(%o3) $[[1, 1, 4, 1, 1, 1, 4, 1, 1, 1, 4, 1, 1, 2], \frac{12957}{7108}]$

```
(%i4)   [bfloat(a0) - bfloat(b1), bfloat(a0) - bfloat(b2),
bfloat(a0) - bfloat(b3)];
(%o4)   [1.447084103723878b-3, 5.700375344641628b-6, 2.24428186590408
5b-8]
```

Цепная дробь может быть представлена числом путем вычисления выражения, возвращаемого функцией `cfdisrep`. См. также описание функции `cfexpand` для другого способа преобразования непрерывной дроби.

```
(%i5)   cfdisrep([1,2,3,4]);
```

(%o5) $1 + \cfrac{1}{2 + \cfrac{1}{3 + \cfrac{1}{4}}}$

```
(%i6)   ev(%);
```

(%o6) $\frac{43}{30}$

Системная переменная	*По умолчанию*	*Описание*
`cflength`	`1`	Контроль числа членов цепной дроби, которая вычисляется функцией `cf`
`factors_only`	`false`	Контроль результата работы функции `ifactors`

Если `factors_only=false`, то функция `ifactors` возвращает информацию о простых множителях и их кратностях. В противном случае на выходе только список множителей.

```
(%i7)   ifactors(50!);
```

(%o7) [[2,47], [3,22], [5,12], [7,8], [11,4], [13,3], [17,2], [19,2], [23,2], [29,1], [31,1], [37,1], [41,1], [43,1], [47,1]]

```
(%i8)   apply("*", map(lambda([u],u[1]^u[2]), %))-50!;
(%o8)      0
```

```
(%i9)   ifactors(50!), factors_only : true;
(%o9)      [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
```

```
(%i10)   load("gcdex")$
```

```
(%i11)   igcdex(2^20-1, 2^30-1);
(%o11)      [−1024, 1, 1023]
```

```
(%i12)   L: [1,2,3,4,5,6,7,8,9]$ map(lambda([p],inrt(10^p,3)),L);
(%o12)      [2, 4, 10, 21, 46, 100, 215, 464, 1000]
```

```
(%i13)   %^3;
(%o13)      [8, 64, 1000, 9261, 97336, 1000000, 9938375, 99897344,
1000000000]
```

```
(%i14)   [isqrt(11),isqrt(13),isqrt(15),isqrt(16),isqrt(17)];
(%o14)      [3,3,3,4,4]
```

```
(%i15)   lcm(12,20,30);
(%o15)      60
```

```
(%i16)   next_prime(10^20);
(%o16)      100000000000000000039
```

```
(%i17)   prev_prime(10^20);
(%o17)      99999999999999999989
```

```
(%i18)   primep(36893488147419103363);
(%o18)      true
```

```
(%i19)   primes(2^15,2^15+100);
(%o19)      [32771, 32779, 32783, 32789, 32797, 32801, 32803, 32831, 32833,
32839, 32843]
```

```
(%i20)   zeta(2);
(%o20)      π²/6
```

7.2. Комбинаторика

Оператор	*Описание*
`!`	Факториал (можно `factorial`)
`!!`	Двойной факториал
`binomial(x,y)`	Биномиальный коэффициент
`combination(n,r)`	То же самое
`multinomial_coeff(a_1, ...,a_n)`	Мультиномиальный коэффициент
`factcomb(expr)`	Комбинирование факториалов в выражении `expr`
`genfact(x,y,z)`	Обобщенный факториал `x (x - z)(x - 2z) ... (x - (y-1) z)`
`minfactorial(expr)`	Проверка на наличие отношения двух факториалов со сходным основанием
`permutations(a)`	Перестановки из элементов списка или множества `a`
`permutation(n,r)`	Перестановки из `n` по `r`

Функция `multinomial_coeff(a_1, ..., a_n)` для неотрицательных целых `a_1`, ..., `a_n` вычисляет величину `(a_1 + ... + a_n)!/(a_1!·...·a_n!)`.

Применение функций `combination` и `permutation` требует загрузки пакета `functs`.

Системная переменная	*По умолчанию*	*Описание*
`factlim`	100000	Контроль за вычислением выражений типа $(n+1)!$, где n – целое число
`factorial_expand`	`false`	Контроль упрощения выражений с факториалами
`sumsplitfact`	`true`	Если `sumsplitfact=false`, то после функции `factcomb` вызывается функция `minfactorial`

```
(%i1)   10!!;
(%o1)       3840
```

```
(%i2)   binomial(x,7);
(%o2)       dfrac(x − 6) (x − 5) (x − 4) (x − 3) (x − 2) (x − 1) x5040
```

```
(%i3)   sumsplitfact;
(%o3)       true
```

```
(%i4)   factcomb((n+2)*(n+1)*n!);
(%o4)       (n + 2)!
```

```
(%i5)  factlim: 10$ 20!;
(%o5)      20!
```

```
(%i6)  (n+1)!/n!,factorial_expand:true;
(%o6)      n + 1
```

```
(%i7)  minfactorial((n!*(k+3)!)/((n+2)!*(k+1)!));
(%o7)      (k + 2) (k + 3) / (n + 1) (n + 2)
```

```
(%i8)  multinomial_coeff(a,b,c);
(%o8)      (c + b + a)! / a! b! c!
```

```
(%i9)  permutations([a,b,3]);
(%o9)      {[3, a, b], [3, b, a], [a, 3, b], [a, b, 3], [b, 3, a], [b, a, 3]}
```

7.3. Графы

Пакет graphs обеспечивает среду системы Maxima возможностью работать с объектами типа графов простой структуры (без кратных ребер и петель), хотя орграфы могут иметь направленные ребра.

Во внутреннем представлении графы фиксируются списками смежности и реализованы в виде структур языка Lisp. Вершины графов идентифицируются по их указателям, которые является целыми числами. Ребра/дуги представлены списками длиной 2. Метки могут быть назначены вершинам графов, а веса – ребрам/дугам графов.

Краткий перечень функций работы с графами приведен в следующей таблице.

Функция	*Описание*
`create_graph`	Создание структуры графа
`copy_graph`	Копирование структуры графа
`draw_graph`	Рисование графа
`empty_graph(n)`	Пустой граф
`make_graph(vrt,f)`	Создание графа на основе предиката `f`

Функция `draw_graph` базируется на пакете `draw` системы Maxima, а также может использовать программу `graphviz`, доступную по адресу `http://www.graphviz.org`.

На рис. 7.1, 7.2 демонстрируется использование пакета `graphs`. Более полное описание пакета см. в документации по системе.

```
(%i1)   load("graphs");
(%o1)       C:/maxima-5.43.0/share/maxima/5.43.0/share/graphs/graphs.mac
```

```
(%i2)   g: petersen_graph(20,2);
(%o2)       GRAPH(40 vertices, 60 edges)
```

```
(%i3)   draw_graph(g, redraw=true, program=planar_embedding)$
```

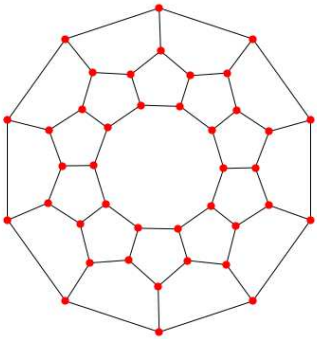


Рис. 7.1

```
(%i4)   t: tutte_graph();
(%o4)       GRAPH(46 vertices, 69 edges)
```

```
(%i5)   draw_graph(g, redraw=true, program=planar_embedding)$
```

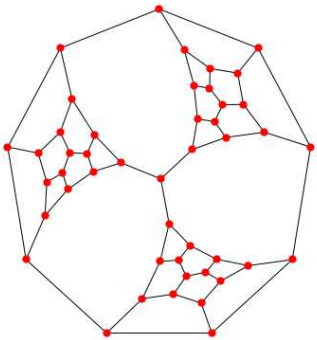


Рис. 7.2

8. ОБЩАЯ ВЫСШАЯ МАТЕМАТИКА

8.1. Высшая и линейная алгебра

8.1.1. Векторы и матрицы. Создание и операции. Определители

Работа с векторами в пакете Maxima реализована на основе списков, т.е. в векторно-матричных операциях список `[v_1, v_2, ..., v_k]` может использоваться как вектор с элементами `v_1`, `v_2`, ..., `v_k`. Алгебраическая сумма векторов и умножение вектора на скаляр вычисляются с помощью операторов `+`, `-` и `*`. Более подробную информацию о использовании векторов см. в подразделе 8.3.

Функция	*Описание*
`matrix(row_1,...,row_n)`	Создать прямоугольную матрицу со строками `row_1`, ..., `row_n`
`ematrix(m,n,x,i,j)`	Построение матрицы размером `m x n`, все элементы которой равны 0, за исключением элемента в позиции (`i`, `j`), который равен `x`
`entermatrix(m,n)`	Создание матрицы размером `m x n` в интерактивном режиме
`genmatrix(A,m,n)`	Создание матрицы `A` размера `m x n`, каждый элемент которой определяется выражением, зависящим от индексов
`columnvector(L)`	Матрица-столбец, состоящая из элементов списка `L`
`covect(L)`	То же самое
`matrixp(expr)`	Предикат: результат `true`, если `expr` – список, иначе `false`
`A+B`	Сумма матриц `A` и `B`
`A-B`	Разность матриц `A` и `B`
`c*A`	Умножить матрицы `A` на скаляр `c`
`A.B`	Некоммутативное умножение матриц `A` и `B`
`A^^n`	Возведение матрицы `A` в степень `n`
`A^^(-1)`	Обратная матрица для матрицы `A`
`transpose(A)`	Транспонирование матрицы `A`
`invert(A)`	Обратная для матрицы `A` (то же, что и `A^^(-1)`)
`invert_by_adjoint(M)`	Обратная для матрицы `A`, вычисленная через присоединенную матрицу
`determinant(A)`	Определитель матрицы `A`
`copymatrix(M)`	Копия матрицы `M`
`newdet(M)`	Определитель матрицы `A`, вычисленный методом Джонсона–Джентельмена

Матрицы создаются с помощью команды `matrix`, которая получает в качестве аргументов строки матрицы в формате записи векторов. По определению строки должны иметь одинаковую длину.

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

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

```
(%i2)  B: matrix([0,1],[2,3]);
```

$$(\%o2)\quad \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

```
(%i3)  A+B;
```

$$(\%o3)\quad \begin{pmatrix} 1 & 3 \\ 5 & 7 \end{pmatrix}$$

```
(%i4)  7*A;
```

$$(\%o4)\quad \begin{pmatrix} 7 & 14 \\ 21 & 28 \end{pmatrix}$$

```
(%i5)  A.B;
```

$$(\%o5)\quad \begin{pmatrix} 5 & 10 \\ 15 & 20 \end{pmatrix}$$

```
(%i6)  A^^2;
```

$$(\%o6)\quad \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

Обратная A^{-1} для матрицы `A` вычисляется так:

```
(%i7)  A^^(-1);
```

$$(\%o7)\quad \begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

Заметим, что операции, обозначаемые `*` и `^`, выполняют поэлементное умножение матриц и возведение в степень соответственно.

```
(%i8)  A*B;
```

$$(\%o8)\quad \begin{pmatrix} 0 & 2 \\ 6 & 12 \end{pmatrix}$$

```
(%i9)   A^2;
```
$$\begin{pmatrix} 1 & 4 \\ 9 & 16 \end{pmatrix}$$

Содержимое i-й строки матрицы A можно получить с помощью конструкции `A[i]`. Элемент a_{ij} может быть извлечен или заменен с помощью ссылки `A[i,j]`.

```
(%i10)  A: matrix([3,1],[2,5]);
```
$$\begin{pmatrix} 3 & 1 \\ 2 & 5 \end{pmatrix}$$

```
(%i11)  A[2];
(%o11)      [2, 5]
```

```
(%i12)  A[2,1];
(%o12)      2
```

```
(%i13)  A[2,1]: 10;
(%o13)      10
```

```
(%i14)  A;
```
$$\begin{pmatrix} 3 & 1 \\ 10 & 5 \end{pmatrix}$$

Обычно векторы отображаются на экране компьютера как векторы-строки. Тем не менее ядро пакета Maxima интерпретирует вектор сообразно ситуации: в зависимости от используемой операции вектор трактуется или как вектор-строка, или как вектор-столбец.

```
(%i15)  x: [2,1];
(%o15)      [2,1]
```

```
(%i16)  A.x;
```
$$\begin{pmatrix} 7 \\ 9 \end{pmatrix}$$

```
(%i17)  x.A;
(%o17)      [8,7]
```

Вычисление обратной матрицы проводится при помощи метода LU-декомпозиции. В случае, если матрица является вырожденной, выдается сообщение об ошибке вида

determinant: matrix must be square; found 0 ***rows,*** 0 ***columns.***
– an error. To debug this try: debugmode(true);

```
(%i18)  A: matrix([-3,1],[2,3]);
(%o18)      ⎛ -3  1 ⎞
            ⎝  2  3 ⎠
```

```
(%i19)  transpose(A);
(%o19)      ⎛ -3  2 ⎞
            ⎝  1  3 ⎠
```

```
(%i20)  invert(A);
(%o20)      ⎛ -3/11  1/11 ⎞
            ⎝  2/11  3/11 ⎠
```

```
(%i21)  determinant(A);
(%o21)      -11
```

Функция `copymatrix(M)` дает единственную возможность неполэлементного копирования матрицы, причем полученная копия является отдельной и независимой от источника.

8.1.2. Вычисление миноров и алгебраических дополнений элементов матрицы. Ранг матрицы

Функция	*Описание*
`minor(A,i,j)`	Матрица, получающаяся вычеркиванием `i`-й строки и `j`-го столбца квадратной матрицы `A`
`rank(A)`	Ранг матрицы `A`, т.е. размерность пространства столбцов матрицы `A`
`nullity(A)`	Коранг матрицы `A`, т.е. размерность нуль-пространства матрицы `A`
`columnspace(A)`	Базис для пространства столбцов матрицы `A`
`nullspace(A)`	Базис для нуль-пространства матрицы `A`
`orthogonal_complement (v_1,...,v_n)`	Линейная оболочка

Пакет Maxima предоставляет команды для вычисления изображения и ядра матрицы.

```
(%i22)  A: matrix([1,2,3],[4,5,6],[7,8,9]);
```

$$(\%o22)\quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i23)  rank(A);
(%o23)  2
```

```
(%i24)  nullity(A);
(%o24)  1
```

```
(%i25)  columnspace(A);
```

$$(\%o25)\quad span\left(\begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}, \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix}\right)$$

```
(%i26)  nspc: nullspace(A);
```

$$(\%o26)\quad span\left(\begin{pmatrix} -3 \\ 6 \\ -3 \end{pmatrix}\right)$$

```
(%i27)  [v1]: nspc, 'span="[";
```

$$(\%o27)\quad [\begin{pmatrix} -3 \\ 6 \\ -3 \end{pmatrix}\ ]$$

```
(%i28)  v1;
```

$$(\%o28)\quad \begin{pmatrix} -3 \\ 6 \\ -3 \end{pmatrix}$$

```
(%i29)  A.v1;
```

$$(\%o29)\quad \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Команды `nullspace` и `columnspace` возвращают базисы соответствующих пространств. Элементы этих базисов могут быть извлечены с помощью операций обработки со списками.

```
(%i30)  A: matrix([1,2,3],[4,5,6],[7,8,9]);
```

$$(\%o30)\quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i31)  cs: columnspace(A);
```

$$(\%o31)\qquad span\left(\begin{bmatrix}1\\4\\7\end{bmatrix},\begin{bmatrix}2\\5\\8\end{bmatrix}\right)$$

```
(%i32)  first(cs);
```

$$(\%o32)\qquad \begin{pmatrix}1\\4\\7\end{pmatrix}$$

```
(%i33)  second(cs);
```

$$(\%o33)\qquad \begin{pmatrix}2\\5\\8\end{pmatrix}$$

```
(%i34)  minor(matrix([1,2,3],[2,3,4],[3,4,5]),2,2);
```

$$(\%o34)\qquad \begin{pmatrix}1 & 3\\3 & 5\end{pmatrix}$$

Функция `orthogonal_complement(v_1,...,v_n)` строит линейную оболочку в виде `span(u_1,...,u_m)`, где система вектор-столбцов `u_1`, ..., `u_m` есть базис ортогонального дополнения к подпространству, порожденному вектор-столбцами `v_1`, ..., `v_n`.

8.1.3. Преобразование матрицы к треугольной или ступенчатой форме

Функция	*Описание*
`triangularize(A)`	Приведение матрицы `A` к треугольному виду методом Гаусса
`echelon(A)`	Приведение матрицы `A` к ступенчатому виду, используя элементарные преобразования

Различие между результатами работы функций `triangularize(A)` и `echelon(A)` состоит в том, что во втором случае на главной диагонали ступенчатой матрицы будут находиться единицы.

```
(%i35)  A: matrix([1,2,3],[-2,4,6],[-3,6,19])$ triangularize(A);
```

$$(\%o35)\qquad \begin{pmatrix}1 & 2 & 3\\0 & 8 & 12\\0 & 0 & 80\end{pmatrix}$$

```
(%i36)   echelon(A);
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & \frac{3}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

8.1.4. Системы линейных алгебраических уравнений и методы их решения

Определение 8.1. Конечная совокупность линейных алгебраических уравнений, включающих один и тот же набор неизвестных, называется *системой линейных алгебраических уравнений* (СЛАУ).

Общая система из m уравнений с n неизвестными может быть записана так:

$$\begin{gathered} a_{11}\,x_1 + a_{12}\,x_2 + \ldots + a_{1n}\,x_n = b_1, \\ a_{21}\,x_1 + a_{22}\,x_2 + \ldots + a_{2n}\,x_n = b_2, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{m1}\,x_1 + a_{m2}\,x_2 + \ldots + a_{mn}\,x_n = b_m, \end{gathered}$$

где x_1, x_2, ..., x_n – неизвестные; a_{11}, a_{12}, ..., a_{mn} – коэффициенты при неизвестных; b_1, b_2, ..., b_m – правые части уравнений. Эти обозначения эквивалентны матричному уравнению вида

$$\boldsymbol{\mathcal{A}}\,\boldsymbol{x} = \boldsymbol{b},$$

где $\boldsymbol{\mathcal{A}}$ – матрица системы линейных алгебраических уравнений; $\boldsymbol{x}$ – вектор неизвестных; $\boldsymbol{b}$ – вектор правых частей уравнений:

$$\boldsymbol{\mathcal{A}} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ x_m \end{bmatrix}.$$

Поиск решений таких СЛАУ является важной частью многих алгоритмов. Если $\boldsymbol{\mathcal{A}}$ – квадратная (т.е. имеет размерность $n \times n$) и невырожденная, то для решения СЛАУ можно использовать ее обратную $\boldsymbol{\mathcal{A}}^{-1}$: $\boldsymbol{x} = \boldsymbol{\mathcal{A}}^{-1}\,\boldsymbol{b}$. Для этого можно применить конструкцию `A^^(-1)` или `invert(A)`:

```
(%i37)   A: matrix([1,2,2],[3,3,2],[1,2,4]);
```

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

```
(%i38)  b: [-1,-4, 2];
```
$$(\%o38) \quad [-1,-4,2]$$

```
(%i39)  A^^(-1).b;
```
$$(\%o39) \quad \begin{pmatrix} -5 \\ -11 \\ -1 \end{pmatrix}$$

Отметим, что существуют другие более эффективные методы решения таких линейных уравнений (см. руководство по системе Maxima [36] и раздел 10 для просмотра таких функций).

Команда `solve` предлагает общий метод, который работает для любого числа уравнений неизвестных.

```
(%i40)  eq1: x[1]+2*x[2]+3*x[3]=5;
```
$$(\%o40) \quad 2\,x_3 + 2\,x_2 + x_1 = 5$$

```
(%i41)  eq2: 3*x[1]+3*x[2]+8*x[3]=16;
```
$$(\%o41) \quad 8\,x_3 + 3\,x_2 + 3\,x_1 = 16$$

```
(%i42)  eq3: 2*x[1]+4*x[2]+2*x[3]=2;
```
$$(\%o42) \quad 2\,x_3 + 4\,x_2 + 2\,x_1 = 2$$

```
(%i43)  solve([eq1,eq2,eq3], [x[1],x[2],x[3]]);
```
$$(\%o43) \quad [[x_1 = 1, x_2 = -1, x_3 = 2]]$$

`solve` также может решать неопределенные системы уравнений, как показано в следующем примере. Символ `%r1` вводится для обозначения произвольной постоянной в общем решении. Частное решение можно найти, заменив `%r1` на некоторое число.

```
(%i44)  solve([eq1,eq2], [x[1],x[2],x[3]]);
```
$$(\%o44) \quad [[x_1 = -\frac{7\,\%r1-17}{3}, x_2 = -\frac{\%r1+1}{3}, x_3 = \%r1]]$$

```
(%i45)  %, %r1=2;
```
$$(\%o45) \quad [[x_1 = 1, x_2 = -1, x_3 = 2]]$$

Функция	*Описание*
`solve([eqn_1,...,eqn_n], [x_1,...,x_n])`	Решение совместной СЛАУ относительно переменных `x_1`, ..., `x_n`
`adjoint(A)`	Присоединенная матрица к квадратной матрице `A`
`augcoefmatrix([eqn_1,..., eqn_n],[x_1,...,x_n])`	Расширенная матрица СЛАУ
`coefmatrix([eqn_1,..., eqn_n],[x_1,...,x_n])`	Матрица СЛАУ
`linsolve([expr_1,..., expr_m],[x_1,...,x_n])`	Решение СЛАУ, определяемой списком `expr_1`, ..., `expr_m`

Системная переменная	*По умолчанию*	*Описание*
`globalsolve`	`false`	Управление связыванием неизвестных с найденным функцией `linsolve` решением СЛАУ
`linsolve_params`	`true`	Если `true`, то функция `linsolve` генерирует символы вида `%r`, использующиеся для подстановки в свободные и базисные переменные в случае неопределенности СЛАУ
`linsolvewarn`	`true`	Если `true`, то в случае зависимых уравнений в СЛАУ вывод сообщения "Dependent equations eliminated" (зависимые уравнения исключены) с номерами исключенных уравнений

Если значение переменной `linsolve_params` равно `false`, то решение системы представляется в виде выражений базисных переменных через свободные. Если значение этой переменной – `true`, то, зная общее решение системы, можно найти частное решениеы.

```
(%i46)   kill(x,y,z); e1: x+z=y$ e2: 2*a*x-y = 2*a^2$ e3: y-2 z = 2$
```

```
(%i47)   globalsolve: false$ linsolve([e1,e2,e3], [x,y,z]);
(%o47)       [x = a + 1, y = 2 a, z = a - 1]
```

```
(%i48)   x;
(%o48)       x
```

```
(%i49)   globalsolve: true$ linsolve([e1,e2,e3], [x,y,z]);
(%o49)       [x : a + 1, y : 2 a, z : a - 1]
```

```
(%i50)   x
(%o50)       a + 1
```

```
(%i51)   kill(x,y,z)$ linsolvewarn: false$ e1: x+z = 3-y$ e2: x-y =
1-z$ e3: -x+2*y+z = 1-y+2*z$
```

```
(%i52)   globalsolve: false$ linsolve_params: true$ linsolve([e1,
e2,e3], [x,y,z]);
```

(%o52) $[x = 2 - \%r3,\ y = 1,\ z = \%r3]$

```
(%i53)   globalsolve: false$ linsolve_params: false$ linsolve([e1,
e2,e3], [x,y,z]);
```

(%o53) $[x = 2 - z,\ y = 1]$

Заметим, что множество всех решений неопределенных систем уравнений образует аффинное пространство, которое может быть задано частным решением СЛАУ $\boldsymbol{\mathcal{A}}\boldsymbol{x} = \boldsymbol{b}$ и нуль-пространством матрицы $\boldsymbol{\mathcal{A}}$ (которое является множеством решений однородной системы уравнений $\boldsymbol{\mathcal{A}}\boldsymbol{x} = \boldsymbol{0}$). Его базис можно получить с помощью команды `nullspace`.

```
(%i54)   nullspace(matrix([1,2,3], [3,3,8]));
```

(%o54) $$span\left(\begin{bmatrix} 7 \\ 1 \\ -3 \end{bmatrix}\right)$$

Для полноты картины отметим, что можно получить ступенчатую форму данной матрицы:

```
(%i55)   echelon(matrix([1,2,3], [3,3,8], [2,4,2]));
```

(%o55) $$span\left(\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & \frac{1}{3} \\ 0 & 0 & 1 \end{bmatrix}\right)$$

8.1.5. Решение матричных уравнений

Функция	*Описание*
`dgesv(A,B)`	Решение матричного уравнения вида `A X = B`

Матрица `A` должна быть квадратной, `B` – матрица, имеющая то же число строк, что и `A`, и любое число столбцов. Размерность результата совпадает с размерностью `B`. Команда `dgesv` вычисляет решение СЛАУ через LU-разложение матрицы `A`.

Элементы матриц `A` и `B` должны преобразовываться в действительные числа с плавающей точкой посредством команды `float`. Таким образом,

элементы этих матриц могут иметь любой числовой тип, быть символьными числовыми константами или выражениями, которые вычисляются до формы числа с плавающей точкой. Элементами матрицы X всегда являются числа с плавающей точкой как результат выполнения операций с плавающей точкой.

Для применения функции `dgesv` требуется подключить пакет `lapack` (см. подраздел 10.3).

```
(%i56)   load("lapack")$
```

```
(%i57)   A: matrix([1,-2.5], [0.375,5])$ b: matrix([1.75],
[-0.625])$
```

```
(%i58)   X : dgesv(A,B);
(%o58)
```

$$\begin{pmatrix} 1.210526315789474 \\ -0.2157894736842105 \end{pmatrix}$$

8.1.6. Характеристический многочлен, собственные числа и собственные векторы матрицы

Функция	*Описание*
`charpoly(A,x)`	Характеристический многочлен матрицы `A`
`ncharpoly(M,x)`	Аналог `charpoly(A,x)`
`eigenvalues(A)`	Собственные числа матрицы `A`
`eigenvectors(A)`	Собственные векторы матрицы `A`
`uniteigenvectors(M)`	Нормированные собственные векторы матрицы `M`
`ueivects(M)`	То же самое

Результатом работы команды `charpoly(A,lambda)` является характеристический полином матрицы `A` относительно переменной `lambda`. При использовании команды `ratsimp(charpoly(A,lambda))` будет получен упрощенный вид характеристического многочлена.

```
(%i59)   A: matrix([1,2,1],[1,4,4],[7,3,5]);
```

$$\text{(\%o59)} \quad \begin{bmatrix} 1 & 2 & 1 \\ 1 & 4 & 4 \\ 7 & 3 & 5 \end{bmatrix}$$

```
(%i60)   charpoly(A,x);
```

$$\text{(\%o60)} \quad -7\,(4-x)+((4-x)(5-x)-12)(1-x)-2\,(-x-23)+3$$

```
(%i61)  expand(%);
```
$$(\%o61)\quad -x^3+10\,x^2-8\,x+29$$

```
(%i62)  ratsimp(charpoly(A,x));
```
$$(\%o62)\quad -x^3+10\,x^2-8\,x+29$$

Собственные значения могут быть вычислены с помощью команды `eigenvalues`. Результатом ее работы является список из двух элементов. Первый – это список собственных значений, а второй элемент – список из соответствующих кратностей.

```
(%i63)  B: matrix([1,0],[6,6]);
```
$$(\%o63)\quad \begin{bmatrix}1 & 0\\ 6 & 6\end{bmatrix}$$

```
(%i64)  eigenvalues(B);
```
$$(\%o64)\quad [[1,\ 6],\ [1,\ 1]]$$

```
(%i65)  eigvals: eigenvalues(B)[1]
```
$$(\%o65)\quad [1,\ 6]$$

```
(%i66)  multiplicities: eigenvalues(B)[2];
```
$$(\%o66)\quad [1,1]$$

Команда `eigenvectors` вычисляет как собственные значения, так и собственные векторы. Она возвращает список из двух элементов: результат `eigenvalues` и список соответствующих собственных пространств. Каждое собственное пространство представлено списком, который содержит его базисные векторы (т.е. собственные векторы).

```
(%i67)  es: eigenvectors(B);
```
$$(\%o67)\quad \left[[[1,6],[1,1]],\left[[1,-\tfrac{6}{5}]\right],[[0,1]]\right]$$

```
(%i68)  eigvals: es[1][1];
```
$$(\%o68)\quad [1,6]$$

```
(%i69)  multiplicities: es[1][2];
```
$$(\%o69)\quad [1,1]$$

```
(%i70)  vectors: es[2];
```
$$(\%o70)\quad \left[[[1,6],[1,1]],\left[[1,-\tfrac{6}{5}]\right],[[0,1]]\right]$$

Заметим, что собственные значения с кратностью больше единицы указываются только один раз, как показано в следующем примере, в котором кратность единственного собственного значения 3 равна 2.

```
(%i71)  C: matrix([3,1],[0,3]);
(%o71)                  [3  1]
                        [0  3]
```

```
(%i72)  eigenvalues(C);
(%o72)                  [[3], [2]]
```

```
(%i73)  eigenvectors(C);
(%o73)                  [[[3], [2]],[[[1,0]]]]
```

```
(%i74)  D: matrix([3,0],[0,3]);
(%o74)                  [3  0]
                        [0  3]
```

```
(%i75)  eigenvectors(D);
(%o75)                  [[[3], [2]],[[[1,0],[0,1]]]]
```

Функция `eigenvalues` напрямую вычисляет корни характеристического полинома. При этом известно, что в случае симметричных матриц все собственные значения действительны. Тем не менее результат может быть получен в виде комплексных чисел. В подразделе 3.6 на примере функции `solve` показано, как получить представление результата работы функции `eigenvalue` в действительных числах.

Возвращаемое функциями `uniteigenvectors` и `ueivects` значение представляет собой список подсписков, первый подсписок из которых является результатом работы `eigenvalue`, а другие подсписки – единичные собственные векторы матрицы `M`, соответствующие этим собственным значениям. Флаги, упомянутые в описании функции `eigenvectors`, дают те же эффекты и здесь.

Если флаг `knowneigvects=true`, то пакет `eigen` предполагает, что собственные векторы матрицы пользователю известны и хранятся в системной переменной с глобальным именем `listeigvects`. Для переменной `listeigvects` содержание должно быть подготовлено в виде списка, аналогичного результату работы функции `eigenvectors`.

Для использования функций `uniteigenvectors` и `ueivects` требуется загрузить пакет `eigen`.

8.1.7. Специальные формы, степени и функции матриц

Функция	*Описание*
`addcol(A,c1,...,cn)`	Добавление к матрице `A` справа столбцов c1, ..., `cn` (в виде списков или матриц) согласованной размерности
`addrow(A,r1,...,rn)`	Добавление к матрице `A` снизу строки `r1,...,rn` (в виде списков или матриц) согласованной размерности
`col(A,i)`	Выделение `i`-го столбца матрицы `A`
`row(A,i)`	Выделение `i`-й строки матрицы `A`
`submatrix`	Построение подматрицы определенного размера из матрицы `A` (см. ниже)
`setelmx(x,i,j,A)`	Замена в матрице `A` элемента `a(i,j)` на выражение `x`
`list_matrix_entries(A)`	Список элементов матрицы `A`
`columnswap(A,i,j)`	Матрица, полученная в результате обмена местами столбцов с номерами `i`, `j` матрицы `A`
`rowswap(A,i,j)`	Матрица, полученная в результате обмена местами строк с номерами `i`, `j` матрицы `A`
`columnop(A,i,j,lambda)`	Матрица, полученная в результате умножения `j`-го столбца матрицы `A` на число `lambda` и сложения с `i`-м столбцом матрицы `A`
`rowop(A,i,j,lambda)`	Матрица, полученная в результате умножения `j`-й строки матрицы `A` на число `lambda` и сложения с `i`-й строкой матрицы `A`

Функция `submatrix` имеет следующие форматы записи:

1) `submatrix(i_1,...,i_m,A,j_1,...,j_n)` – выделение элементов из указанных строк и столбцов;

2) `submatrix(i_1,...,i_m,A)` – выделение элементов из указанных строк;

3) `submatrix(A,j_1,...,j_n)` – выделение элементов из указанных столбцов.

Функция	*Описание*
`diagmatrix(n,x)`	Диагональная матрица размерности $n \times n$ с диагональными элементами `x`
`ident(n)`	Единичная матрица порядка n
`identfor(A)`	Единичная матрица размера, соответствующего размеру матрицы `A`
`zeromatrix(m,n)`	Нулевая матрица размером `m` $\times$ `n`
`zerofor(A)`	Нулевая матрица размера, соответствующего размеру матрицы `A`
`hilbert_matrix(n)`	Матрица Гильберта порядка `n`

Функция	*Описание*
`vandermonde_matrix ([s1,s2,...,sn])`	Матрица Вандермонда, соответствующая списку чисел `s1`, `s2`, ..., `sn`
`mattrace(A)`	След (сумма элементов на главной диагонали квадратной) матрицы `A`
`mat_trace(M)`	След матрицы `M`, которая может быть блочной
`diag_matrix(s1,s2,...,sn)`	Блочная диагональная матрица из списка `s1`, `s2`, ..., `sn` матриц
`kronecker_product(A,B)`	Произведение Кронекера двух матриц `A`, `B`
`mat_unblocker(A)`	Преобразование блочной матрицы `A` в обычную
`mat_fullunblocker(M)`	Преобразование блочной матрицы `A` в обычную на всех уровнях
`mat_norm(A,p)`	`p`-норма матрицы `A` (`p` может быть 1, `inf`, `frobenius`)
`matrix_size(A)`	Размерность матрицы `A`
`mat_function(f,A)`	Функция `f(A)`, где `f` – аналитическое выражение, `A` – матрица
`addmatrices(f,M1,...,Mn)`	Сумма матриц `M1`, ..., `Mn` одинаковых размеров с использованием специальной аддитивной функции `f`
`matrixexp(M)`	Матричная экспонента
`matrixmap(f,M)`	Матрица, элементы которой равны `f(M(i,j))`
`similaritytransform(M)`	Преобразование подобия матрицы `M`
`simtran(M)`	То же самое
`blockmatrixp(M)`	Предикат: если `true`, то `M` – блочная матрица

Для использования функции `mattrace` нужно загрузить пакет `nchrpl`.

Множество единичных матриц и их произведений на числа может быть построено с помощью команд `diagmatrix` and `ident`.

```
(%i76)  ident(2);
```

$$(\%o76)\quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

```
(%i77)  diagmatrix(2,3);
```

$$(\%o77)\quad \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$$

```
(%i78)  vandermonde_matrix([1,2,3,4]);
```

$$(\%o78)\quad \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{pmatrix}$$

Функция `diag_matrix` формирует матрицу, на главной диагонали которой располагаются матрицы `s1`, `s2`, ..., `sn`, а вне диагонали – нулевые матрицы соответствующего размера. Если в качестве параметров функции используются числа, то результатом будет диагональная квадратная матрица.

```
(%i79)   M:matrix([4,2],[3,1])$ M1: matrixexp(M)$ M2:
matrixexp(-M)$
```

```
(%i80)   expand(M1.M2);
(%o80)        ( 1  0 )
              ( 0  1 )
```

Функция `similaritytransform` вычисляет преобразование подобия матрицы `M` и возвращает список, который является результатом работы функции `uniteigenvectors`. Кроме того, если флаг `nondiagonalizable` равен `false`, вычисляются две глобальные матрицы `leftmatrix` и `rightmatrix`. Эти матрицы имеют свойство, что `leftmatrix.M.rightmatrix` является диагональной матрицей с собственными значениями M на диагонали. Если `nondiagonalizable=true`, левая и правая матрицы не вычисляются.

Если флаг `hermitianmatrix=true`, тогда `leftmatrix` является комплексным сопряжением для транспонированной `rightmatrix`. В противном случае `leftmatrix` является обратной для `rightmatrix`. `rightmatrix` – это матрица, столбцы которой являются единичными собственными векторами матрицы `M`.

Для использования функции `similaritytransform` требуется загрузить пакет `eigen`

Системная переменная	*По умолчанию*	*Описание*
`matrix_element_add`	`+`	Операция, вызываемая вместо сложения в матричном умножении
`matrix_element_mult`	`.`	Операция, вызываемая вместо умножения при матричном умножении
`matrix_element_transpose`	`false`	Операция, применяемая к каждому элементу матрицы, когда она транспонируется
`scalarmatrixp`	`true`	Преобразование матрицы–скаляра

`matrix_element_add`, `matrix_element_mult`, `matrix_element_transpose` – переменные, позволяющие переопределять стандартные бинарные операции над матрицами (сложение, умножение, транспонирование) новыми операциями.

```
(%i81)   matrix_element_add: "*"$ matrix_element_mult: "^"$
```

```
(%i82)   aa: matrix([2,3,4], [5,6,7])$ bb: matrix([x,y,z],
[z,y,x])$
```

```
(%i83)   aa . transpose(bb);
```

$$(\%o83)\qquad \begin{pmatrix} 2^x\,3^y\,4^z & 4^x\,3^y\,2^z \\ 5^x\,6^y\,7^z & 7^x\,6^y\,5^z \end{pmatrix}$$

Преобразовать список в матрицу и наоборот можно так:

```
(%i84)   L: [[4,2],[3,1]]$ M: apply('matrix,L);
```

$$(M)\qquad \begin{pmatrix} 4 & 2 \\ 3 & 1 \end{pmatrix}$$

```
(%i85)   args(M.transpose(M));
(%o85)       [[20, 14], [14, 10]]
```

Если `scalarmatrixp=true`, то матрица–скаляр (1x1-матрица) превращается в скаляр. Если `scalarmatrixp=all`, то все матрицы-скаляры превращаются в скаляры. Если `scalarmatrixp=false`, то превращения в скаляр нет.

8.1.8. Решение полиномиальных уравнений

Функция	*Описание*
`=`	Оператор, определяющий уравнение
`solve(eqn,x)`	Решение уравнения `eqn` относительно `x`
`solve([eqn_1,...,eqn_n],[x_1,...,x_n])`	Решение совместной системы полиномиальных уравнений относительно переменных `x_1`, ..., `x_n`
`lhs(eqn)`	Левая часть уравнения
`rhs(eqn)`	Правая часть уравнения
`allroots(polyn)`	Численные аппроксимации корней полинома `polyn` одной переменной
`algsys([expr1,...,exprm],[x1,...,xn])`	Решение произвольной системы полиномиальных уравнений относительно переменных `x1`, ..., `xn`
`bfallroots(expr)`	Решение алгебраического уравнения с действительными и комплексными корнями
`bfallroots(eqn)`	
`funcsolve(eqn,g(t))`	Решение линейных функциональных уравнений

Системная переменная	*По умолчанию*	*Описание*
`globalsolve`	`false`	Если `true`, то связывание неизвестных с найденным функцией `solve` решением системы полиномиальных уравнений
`algepsilon`	10^{-8}	Точность расчетов для функции `algsys`
`algexact`	`false`	Контроль работы функции `algsys`
`solveexplicit`	`false`	Если `true`, то функция `solve` может искать неявные решения
`solvefactors`	`false`	Если `false`, то функция `solve` не пытается факторизовать уравнения

Команда `solve` пытается решить алгебраические уравнения и возвращает список решений уравнений. Необходимо использовать функцию `float`, чтобы получить в форме чисел с плавающей точкой.

```
(%i86)   solve(x^2+3*x-1=0, x);
```

$$(\%o86) \qquad [x = -\frac{\sqrt{13}+3}{2},\ x = \frac{\sqrt{13}-3}{2}]$$

```
(%i87)   float(%);
```

$$(\%o87) \qquad [x = -3.302775637731995,\ x = .3027756377319946]$$

Нахождение корней и нулей выражения является очень важным специальным приложением решения уравнений. Таким образом, если выражение `expr`, данное в качестве аргумента функции `solve`, не является уравнением (знак `=` отсутствует), то вместо него предполагается выражение `expr = 0`. Таким образом, можно получить корни указанного полинома следующей командой:

```
(%i88)   sol: solve(x^2+3*x-1, x);
```

$$(\%o88) \qquad [x = -\frac{\sqrt{13}+3}{2},\ x = \frac{\sqrt{13}-3}{2}]$$

Решения могут быть преобразованы в "неуравнения" или использованы в дальнейших вычислениях с помощью команды `ev`. Альтернативный способ – использовать команду `rhs` для извлечения выражения из правой части уравнения.

```
(%i89)   ev(x, sol[1]);
```

$$(\%o89) \qquad -\frac{\sqrt{13}+3}{2}$$

```
(%i90)  rhs(sol[2]);
(%o90)  (sqrt(13)-3)/2
```

Неизвестная в уравнении может быть более сложным выражением, чем простая переменная, например, неизвестная может быть функцией.

```
(%i91)  solve(exp(f(x))=10^y, f(x));
(%o91)  [f(x) = log(10) y]
```

Системы из двух уравнений и более, а также их неизвестные должны быть объединены в списки. Каждое решение также затем возвращается в виде списка. Таким образом, мы получаем список списков.

```
(%i92)  eq1: 3*x^2-y^2=2;
(%o92)  3 x^2 - y^2 = 2
```

```
(%i93)  eq2: x^2+y^2=2;
(%o93)  x^2 + y^2 = 2
```

```
(%i94)  solve([eq1,eq2], [x,y]);
(%o94)  [[x = -1, y = -1], [x = -1, y = 1], [x = 1, y = -1], [x = 1, y = 1]]
```

`solve` возвращает решение в виде уравнений. Эти решения могут быть использованы в дальнейших вычислениях с помощью `ev`.

```
(%i95)  s: solve(x^3+2*x^2-5*x-6=0, x);
(%o95)  [x = -3, x = -1, x = 2]
```

```
(%i96)  z: ev(%pi^x, s[2]);
(%o96)  1/%pi
```

`solve` не всегда может найти (все) решения системы уравнений. В таком случае возвращается пустой список. В следующем примере "очевидное" решение `[x = 1, y = 1]` не находится.

```
(%i97)  solve([x^(-1/2)*y^(1/2)=1, x^(1/2)*y^(-1/2)=1], [x,y]);
(%o97)  []
```

```
(%i98)  ev([x^(-1/2)*y^(1/2)=1, x^(1/2)*y^(-1/2)=1], [x=1,y=1]);
(%o98)  [0, 0]
```

В общем случае не существует замкнутых формул для корней многочленов степени 5 и выше. Тогда команда `allroots` может быть использована для получения хотя бы числового приближения корней.

```
(%i99)   solve(x^5-x^4+2*x^3+x^2-x+5, x);
```

$$(\%o99) \qquad [0 = x^5 - x^4 + 2\,x^3 + x^2 - x + 5]$$

```
(%i100)  allroots(x^5-x^4+2*x^3+x^2-x+5);
```

$$(\%o100) \qquad [x = 1.07479\,\%i + .970613,\ x = .970613 - 1.07479\,\%i,\ x = -1.1828,\ x = 1.41457\,\%i + .120788,\ x = .120788 - 1.41457\,\%i]$$

Решения уравнений могут быть комплексными числами.

```
(%i101)  solve(x^2-4*x+13=0,x);
```

$$(\%o101) \qquad [x = 2 - 3\,\%i,\ x = 3\,\%i + 2]$$

Несмотря на то, что комплексные выражения довольно удобны во многих приложениях, результат может быть неприемлем, если требуются чисто действительные решения. Ситуация может быть еще хуже. Это показывает следующий пример, где решение алгебраического уравнения третьей степени вычисляется по формуле Кардано: на первый взгляд, кажется, что оно состоит из комплексных чисел. Но это не так.

```
(%i102)  s: solve(x^3-x+1/3=0,x)
```

$$(\%o102) \qquad \left[x = \frac{\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}}{3\left(\frac{\%i}{23^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}} + \left(\frac{\%i}{23^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}\left(-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right),\right.$$

$$x = \left(\frac{\%i}{23^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}\left(\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right) + \frac{-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}}{3\left(\frac{\%i}{23^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}},$$

$$\left.x = \left(\frac{\%i}{23^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}} + \frac{1}{3\left(\frac{\%i}{23^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}}\right]$$

Тем не менее все три решения являются действительными числами. Можно попытаться преобразовать эти "псевдокомплексные" числа в алгебраическую форму, т.е. в представление $a+b\,\boldsymbol{i}$, где $a \in \mathbb{R}$ и $b \in \mathbb{R}$ называются действительной и мнимой частью комплексного числа соответственно. Затем можно попытаться превратить мнимую часть в 0 с помощью команды `trigsimp`.

```
(%i103)  trigsimp(rectform(s));
```

$$(\%o103)\quad \left[x=\frac{\sqrt{3}\,\sin\left(\frac{5\,\pi}{18}\right)-\cos\left(\frac{5\,\pi}{18}\right)}{\sqrt{3}},\ x=-\frac{\sqrt{3}\,\sin\left(\frac{5\,\pi}{18}\right)+\cos\left(\frac{5\,\pi}{18}\right)}{\sqrt{3}},\ x=\frac{2\,\cos\left(\frac{5\,\pi}{18}\right)}{\sqrt{3}}\right]$$

```
(%i104)  float(%);
```

$$(\%o104)\quad [x=.3949308436346985,\ x=-1.137158042603258,\ x=.7422271989685593\,]$$

Если не пользоваться `trigsimp` или если невозможно уменьшить мнимую часть до 0, то небольшая мнимая часть может остаться при преобразовании решений в числа с плавающей запятой из-за ошибок округления. Решения могут быть преобразованы в чисто действительные при с помощью выделения их действительных частей. Тем не менее нужно быть уверенным, что решение не является комплексным числом.

```
(%i105)  float(rectform(s));
```

$$(\%o105)\quad [x=.3949308436346984-1.110223024625157\,10^{-16}\,\%i,\ x=-4.163336342344337\,10^{-17}\,\%i-1.137158042603257,\ x=.7422271989685593\,]$$

```
(%i106)  realpart(%);
```

$$(\%o106)\quad [x=.3949308436346984,\ x=-1.137158042603257,\ x=.7422271989685593\,]$$

Функция `algsys([expr1,...,exprm],[x1,...,xn])` решает систему полиномиальных уравнений `[expr1=0, expr2=0, ..., exprm= 0]` относительно списка переменных `[x1, ..., xn]`. Выражения `[expr1, ..., exprm]` могут быть представлены и в виде уравнений. Количество уравнений может быть больше, меньше и равно количеству неизвестных. Фактически функция `algsys` является надстройкой над `solve`, т.к. использует последнюю для поиска точного решения.

```
(%i107)  eq1: 2*x*(1-u) - 2*(x-1)*v$ eq2: v-u$ eq3: u*(-y-x^2+1)$
eq4: v*(y-(x-1)^2)$
```

```
(%i108)  algsys([eq1,eq2,eq3,eq4], [x,y,u,v]);
```

$$(\%o108)\quad [[x=0,y=\%r1,u=0,v=0],\ [x=1,y=0,u=1,v=1]]$$

```
(%i109)   eq5: x^2-y^2$ eq6: -1-y+2*y^2-x+x^2$ algsys([eq5,eq6],
[x,y]);
```

(%o109) $[[x = -\frac{1}{\sqrt{3}}, y = \frac{1}{\sqrt{3}}], [x = \frac{1}{\sqrt{3}}, y = \frac{1}{\sqrt{3}}], [x = -\frac{1}{3}, y = -\frac{1}{3}], [x = 1, y = 1]]$

8.1.9. Исключение неизвестных

Функция	*Описание*
`eliminate([eqn_1,...,eqn_n],[x_1,...,x_k])`	Исключение переменных из уравнений (или выражений, принимаемых равными нулю)

Функция `eliminate` возвращает список из `n-k` выражений с удаленными `k` переменными `x_1`, ..., `x_k`. Сначала удаляется `x_1` с получением `n-1` выражения, затем удаляется `x_2` и т.д. Если `k=n`, то возвращается единственное выражение в списке без переменных `x_1`, ..., `x_k`. В этом случае функция вызывается для поиска последнего результата для последней переменной.

```
(%i110)   expr1:2*x^2+y*x+z$ expr2:3*x+5*y-z-1$ expr3:z^2+x-y^2+5$
```

```
(%i111)   eliminate([expr3,expr2,expr1], [y,z]);
```

(%o111) $[7425\,x^8 - 1170\,x^7 + 1299\,x^6 + 12076\,x^5 + 22887\,x^4 - 5154\,x^3 - 1291\,x^2 + 7688\,x + 15376]$

8.2. Комплексные числа

Функция	*Определение*
`rectform(z)`	Алгебраическая форма $a + b\,i$ комплексного числа `z`
`polarform(z)`	Показательная форма $r\,e^{i\theta}$ комплексного числа `z`
`realpart(z)`	Действительная часть комплексного числа `z`
`imagpart(z)`	Мнимая часть комплексного числа `z`
`conjugate(z)`	Комплексно-сопряженное число для `z`
`trigsimp(z)`	Упрощение выражений, содержащие тригонометрические функции
`abs(z)`	Модуль комплексного числа `z`
`cabs(z)`	Модуль комплексного числа `z`
`carg(z)`	Аргумент θ комплексного числа `z` ($\theta \in (-\pi, \pi]$)
`ctranspose(A)`	Комплексно-сопряженная матрица для матрицы `A`
`rational(z)`	Умножение числителя и знаменателя `z` на число, комплексно сопряженное к знаменателю

Функция	*Определение*
`plog(x)`	Представление основной ветви комплексного натурального логарифма с `-%pi < carg(x) <= +%pi`
`polartorect(r,t)`	Перевод комплексного числа из тригонометрической в алгебраическую форму
`recttopolar(a,b)`	Перевод комплексного числа из алгебраической в тригонометрическую форму
`residue(expr,z,z_0)`	Вычисление вычета в заданной точке

```
(%i1)  cabs(exp(%pi*%i));
(%o1)  1
```

```
(%i2)  cabs(sin(x+%i*y));
(%o2)  sqrt(cos(x)^2 sinh(y)^2 + sin(x)^2 * cosh(y)^2)
```

```
(%i3)  carg(exp(%pi*%i));
(%o3)  %pi
```

```
(%i4)  conjugate(5-4*%i);
(%o4)  4 %i + 5
```

```
(%i5)  imagpart(a+b*%i);
(%o5)  b
```

```
(%i6)  realpart(a+b*%i);
(%o6)  a
```

```
(%i7)  ctranspose(matrix([1,1-2*%i],[1+3*%i,1]));
(%o7)  matrix([1, 1 - 3 %i], [1 + 2 %i, 1])
```

```
(%i8)  load(functs)$ rational((1-2*%i)/(1+3*%i));
(%o8)  - (%i + 1)/2
```

```
(%i9)  polarform(1+%i);
(%o9)  sqrt(2) %e^(%i %pi/4)
```

```
(%i10)  rectform(sqrt(2)*%e^(%i*%pi/4));
(%o10)  %i + 1
```

Функция `rational` требует загрузки пакета `functs`, а функции `polartorect` и `rectpolar` – `fft`.

8.3. Векторная алгебра

8.3.1. Линейные операции

Как уже было сказано выше, работа с векторами в пакете Maxima реализована на основе списков, т.е. в векторно-матричных операциях список `[v_1, v_2, ..., v_k]` может использоваться как вектор с элементами `v_1`, `v_2`, ..., `v_k`. Алгебраическая сумма векторов и умножение вектора на скаляр вычисляются с помощью операторов `+`, `-` и `*`.

Функция	*Описание*
`u+v`	Сумма векторов `u` и `v`
`u-v`	Разность векторов `u` и `v`
`s*v`	Умножить вектор `v` на скаляр `s`
`columnvector(L)`	Вектор-столбец из элементов списка `L`
`unitvector(x)`	Нормированный вектор для заданного вектора `x`
`uvect(x)`	То же самое

```
(%i1)   [1,0,1] + [1,2,3];
(%o1)       [2, 2, 4]
```

```
(%i2)   [1,0,1] - [1,2,3];
(%o2)       [0, -2, -2]
```

```
(%i3)   5*[1,2,3];
(%o3)       [5, 10, 15]
```

Обратиться к k-му элементу вектора можно с помощью конструкции `<имя>[k]`, где `<имя>` – идентификатор вектора:

```
(%i4)   x: [1,2,3,4];
(%o4)       [1, 2, 3, 4]
```

```
(%i5)   x[3];
(%o5)       3
```

Символ `[k]` также можно использовать в качестве индекса для любого выражения, в частности, в случае необходимости создать вектор с символическими компонентами.

```
(%i6)   y[3];
(%o6)      y3
```

```
(%i7)   z: [z[1],z[2]]
(%o7)      [z1, z2]
```

Для использования функций `unitvector` и `uvect` требуется предварительная загрузка пакета `eigen`.

8.3.2. Скалярное, векторное и смешанное произведения

Функция	*Описание*
`innerproduct(x,y)`	Скалярное произведение `x`, `y`
`u.v`	Скалярное произведение векторов `u` и `v`
`dotproduct(v1,v2)`	Скалярное произведение векторов `v1` и `v2`
`v1~v2`	Векторное произведение векторов `v1` и `v2`

`innerproduct(x,y)` (или `inprod(x,y)`) вычисляет произведение `x` и `y`, которые могут быть списками одинаковой длины, векторами-столбцами или матрицами-строками, по формуле `conjugate(x).y`.

`dotproduct(v1,v2)` вычисляет произведение двух вектор-столбцов `v1`, `v2` по формуле `conjugate(transpose(v1)).v2`.

```
(%i8)   [1,0,1].[1,2,3];
(%o8)      4
```

Заметим, что символ `*` указывает на выполнение поэлементного умножения:

```
(%i9)   [1,2,3]*[1,2,3];
(%o9)      [1, 4, 9]
```

Или использования функции нужно загрузить пакет `vect`.

```
(%i10)   load("vect")$ a: [1,2,3]$ b: [3,2,1]$ a~b;
(%o10)      [1, 2, 3] ~ [3, 2, 1]
```

```
(%i11)   express(%);
(%o11)      [−4, 8, −4]
```

Используя скалярное и векторное произведения можно найти:
– смешанное произведение

```
(%i12)   c: [1,1,1]$ (a~b).c;
(%o12)       −[1, 2, 3].[1, 1, 1] ∼ [3, 2, 1]
```

```
(%i13)   express(%);
(%o13)       0
```

– проекцию вектора a на вектор b

```
(%i14)   express((a.b)/(b.b)*b);
(%o14)       [15/7, 10/7, 5/7]
```

8.3.3. Ортогонализация

Функция	*Описание*
`gramschmidt(x,F)`	Ортогонализации Грама–Шмидта системы векторов `x` относительно скалярного произведения, определенного функцией `F`

Функция `gramschmidt(x)` осуществляет процесс ортогонализации относительно стандартного скалярного произведения. Для ее использования необходимо загрузить пакет `eigen`.

```
(%i15)   load("eigen")$
```

```
(%i16)   x: matrix([1,1,-1], [-1,0,1], [-1,-1,-1])$
```

```
(%i17)   y: gramschmidt(x);
(y)      [[1, 1, −1], [−1/3, 2/3, 1/3], [−1, 0, −1]]
```

```
(%i18)   map(innerproduct, [y[1],y[2],y[3]], [y[2],y[3],y[1]]);
(%o18)       [0, 0, 0]
```

8.4. Математический анализ

8.4.1. Многочлены и дробно-рациональные функции. Корни и делимость многочленов

Полиномы и дробно-рациональные функции хранятся в системе Maxima либо в общей форме (General Form, GF), либо в форме канонических рациональных выражений (Canonical Rational Expressions, CRE). Последняя является стандартной формой и используется в таких функциях, как `factor`, `ratsimp` и т.д.

CRE – это вид представления, который особенно удобен для операций с многочленами, у которых раскрыты скобки, и рациональными функциями (а также для частично факторизованных многочленов и рациональных функций, когда для флажка `ratfac` установлено значение `true`).

В CRE-форме предполагается, что установлен порядок старшинства для переменных в каждом выражении. Полиномы представляются рекурсивно списками, состоящими из основной переменной, за которой следуют последовательности пар выражений, по одному для каждого члена полинома: первый член каждой пары – степень переменной в этом терме, второй член – коэффициент, который может быть числом или полиномом от другой переменной. Таким образом, CRE-форма для полинома `3*x^2-1` есть `(x 2 3 0 -1)`, а для `2*x*y+x-3` – `(y 1 (x 1 2) 0 (x 1 1 0 -3))` при условии, что `y` является старшей переменной, и `(x 1 (y 1 2 0 1) 0 -3)` при условии, что `x` – старшая. Старшинство обычно определяется в обратном алфавитном порядке.

"Переменные" в CRE-форме необязательно должны быть атомарными. Фактически любое подвыражение, основной операцией которого не является `"+"`, `"-"`, `"*"`, `"/"` или `"^"` с целочисленной степенью, будет считаться "переменной" выражения (в CRE-форме), в котором оно встречается. Например, переменные CRE-выражения `x+sin(x+1)+2*sqrt(x)+1` – это `x`, `sqrt(x)` и `sin(x+1)`. Если пользователь не указывает порядок переменных с помощью функции `ratvars`, то Maxima выберет алфавитный.

В общем случае CRE-формы представляют рациональные выражения, т.е. отношения полиномов, где числитель и знаменатель не имеют общих множителей, а знаменатель положителен. Внутренняя форма – это, по сути, пара полиномов (числитель и знаменатель), которым предшествует список со старшинством переменных. Если отображаемое выражение имеет CRE-форму или содержит какие-либо подвыражения в CRE-форме, то за номером ячейки будет отображен символ `/R/` (см. функцию `rat` для преобразования выражения в CRE-форму).

Расширенная CRE-форма используется для представления рядов Тейлора. Понятие рационального выражения расширяется, так что степени переменных могут быть положительными или отрицательными рациональными числами, а не просто положительными целыми числами, а сами коэффициенты могут быть рациональными выражениями, как описано выше, а не просто полиномами. Во внутреннем представлении они имеют рекурсивную полиномиальную форму, которая похожа и является обобщением CRE-формы, но несет дополнительную информацию, такую как степень усечения. Как и в случае CRE-формы, символ `/T/` следует за номером ячейки с таким выражением.

Функция	*Описание*
`coeff(expr,x,n)`	Коэффициент при `x^n` в полиноме `expr`
`coeff(expr,x)`	Коэффициент при `x` в полиноме `expr`
`content(p,x)`	Список, первый элемент которого – НОД коэффициентов членов полинома `p` по переменной `x`, а вторым элементом является результат деления `p` на `x`
`divide(p1,p2,x)`	Результат деления многочлена `p1` на полином `p2` по отношению к переменной `x`
`eliminate([eqn1,...,eqnn],[x_1,...,x_k])`	Исключение переменных из уравнений (или выражений, принимаемых равными нулю)
`ezgcd(p1,p2,p3,...)`	НОД полиномов `p1`, `p2`, `p3`, ...
`gcd(p1,p2,x)`	НОД полиномов `p1` и `p2` по переменной `x`
`gcdivide(p,q)`	Результат сокращения числителя и знаменателя на НОД, если `takegcd=true`
`poly_gcd(p1,p2,L)`	НОД полиномов `p1` и `p2` по переменной списку переменных `L`
`factor(expr)`	Факторизация многочлена
`factorout(expr,x_1,x_2,...)`	Упорядочивание `expr` и разложение в сумму слагаемых вида `f(x_1,x_2,...)*g`, где `g` – произведение выражений, не содержащих `x_i`, и `f` факторизованы
`factorsum(expr)`	Факторизация многочлена с группировкой термов в сомножителях
`sqfr(expr)`	Факторизация до множителей первого порядка
`remainder(p1,p2)`	Остаток от деления полинома `p1` на `p2`
`quotient(p1,p2)`	Целая часть от деления полинома `p1` на `p2`
`totaldisrep(expr)`	Конвертация каждого подвыражения выражения `expr` из CRE-формы в общую форму
`gcfactor(n)`	Факторизация целого числа над рациональными комплексными числами
`gfactor(expr)`	Факторизация над целыми комплексными числами
`hipow(expr,x)`	Наибольшая степень `x` в `expr`
`lopow(expr,x)`	Наименьшая степень `x` в `expr`
`polynomialp(p,L)`	Предикат: если `true`, то `p` является полиномом от переменных из списка `L`
`denom(expr)`	Знаменатель рациональной дроби
`num(expr)`	Числитель рациональной дроби
`partfrac(expr,var)`	Разложение рациональной дроби на простейшие дроби
`lratsubst(L,expr)`	Подстановка
`rat(expr)`	Перевод в CRE-форму с упрощениями `expr`
`rat(expr,x_1,...,x_n)`	То же, но по переменным `x_1`, ..., `x_n`
`ratcoef(expr,x,n)`	Выделение коэффициента при `x^n`, где `x` – выражение

Функция	*Описание*
`ratcoef(expr,x)`	Выделение коэффициента при `x`
`ratdiff(expr,x)`	Дифференцирование рациональных дробей
`ratexpand(expr)`	Раскрытие скобок в числителях и знаменателях
`ratsimp(expr)`	Упрощение всего выражения и его подвыражений
`fullratsimp(expr)`	Циклическое упрощение всего выражения и его подвыражений
`ratsubst(a,b,c)`	Подстановка `a` вместо `b` в `c`
`fullratsubst(a,b,c)`	Циклическая подстановка `a` вместо `b` в `c`
`ratp(expr)`	Предикат: если `true`, то `extr` имеет CRE-форму

В отличие от функции `ratcoef`, `coeff` – чисто синтаксическая функция, которая ищет буквенное вхождение `x^n` во внутреннее представление полинома `expr`. Вызов `coeff(expr,x^n)` эквивалентен `coeff(expr,x,n)`, а `coeff(expr,x,0)` дает часть `expr`, в которой нет `x`.

`x` может быть простой переменной, переменной с индексом или подвыражением выражения `expr`, которое включает некоторый оператор и аргументы. Требуется, чтобы можно было вычислить коэффициенты выражений, которые эквивалентны `expr` после применения функций `expand` или `coeff`. Сама функция `coeff` не использует функции `expand`, `coeff` или любые другие.

```
(%i1)   coeff(y^3*x^3 + y^2*x^2 + y*x + 1, x^3);
(%o1)       y^3
```

```
(%i2)   coeff(sin(1+x)*sin(x) + sin(1+x)^3*sin(x)^3, sin(1+x)^3);
(%o2)       sin(x)^3
```

```
(%i3)   coeff([4*a-1, 2-3*a, 2*a+4*a^2], a);
(%o3)       [4, -3, 2]
```

```
(%i4)   content(2*a*y + 4*a^2*y^2, y);
(%o4)       [2 a, 2 a y^2 + y]
```

Функция `divide(p_1,p_2,x)` вычисляет частное и остаток от деления многочлена `p_1` на многочлен `p_2`, которые являются функциями `x`. Результатом является список, первый элемент которого – частное, а второй – остаток от деления.

```
(%i5)   divide(x^2+y*x-1, x-y, x);
(%o5)       [2 y + x, 2 y^2 - 1]
```

Функция `eliminate` возвращает список из `n-k` выражений с удаленными `k` переменными `x_1`, ..., `x_k`. При исключении `x_1` остается `n-1` выражение, затем устраняется `x_2` и т.д. Если `k=n`, то результат исключения – единственное выражение в списке без переменных `x_1`, ..., `x_k`.

```
(%i6)   eliminate([z^2+x-y^2+5, 3*x+5*y-z-1, 2*x^2+y*x+z], [x,z]);
```
$$(\%o6)\qquad [2\,(540\,y^4 - 432\,y^3 + 702\,y^2 - 1123\,y + 461)]$$

Результат работы функции `ezgcd(p1,p2,p3,...)` – список, первый элемент которого – НОД полиномов `p1`, `p2`, `p3`, ..., а остальные элементы – указанные полиномы, деленные на НОД.

```
(%i7)   p1: 3*x^5-5*x^4-5*x^3+16*x^2-12*x+2$ p2:
-6*x^4+4*x^3+14*x^2-12*x+2$ p3: -12*x^5+23*x^4-15*x^3+17*x^2-12*x+2$
```

```
(%i8)   ezgcd(p1, p2, p3);
```
$$(\%o8)\qquad [3\,x^2 - 5\,x + 1, x^3 - 2\,x + 2, -2\,x^2 - 2\,x + 2, -4\,x^3 + x^2 - 2\,x + 2]$$

```
(%i9)   gcd(p1, gcd(p2, p3));
```
$$(\%o9)\qquad 3\,x^2 - 5\,x + 1$$

```
(%i10)   load(functs)$
```

```
(%i11)   gcdivide(6*x^3+19*x^2+19*x+6, 6*x^5+13*x^4+12*x^3+13*x^2+6*x);
```
$$(\%o11)\qquad \frac{x+1}{x^3+x}$$

```
(%i12)   load(grobner)$
```

```
(%i13)   poly_gcd(x^6-y^6, x^4-y^4, [x,y]);
```
$$(\%o13)\qquad x^4 - y^4$$

```
(%i14)   factor(1 + %e^(3*x));
```
$$(\%o14)\qquad (\%e^x + 1)\,(\%e^{2*x} - \%e^x + 1)$$

```
(%i15)   factorsum(a*x*z^2 + a*z^2 + 2*a*w*x*z + 2*a*w*z + a*w^2*x
+ v^2*x + 2*u*v*x + u^2*x + a*w^2 + v^2 + 2*u*v + u^2);
```
$$(\%o15)\qquad (x+1)\,(a\,(z+w)^2 + (v+u)^2)$$

```
(%i16)   sqfr(expand((x+1)*(x-1)^2*(x^3+1)));
```
$$(\%o16)\qquad (x^2-1)^2\,(x^2 - x + 1)$$

```
(%i17)  remainder((x+1)*(x-1)^2/(x^3+1),x);
```

$$\frac{1}{x^2-x+1}$$

```
(%i18)  quotient((x+1)*(x-1)^2/(x^3+1),x);
```

$$\frac{x-2}{x^2-x+1}$$

```
(%i19)  factorout(a*u^2*x^2+2*a*u*x^2+a*x^2-a*u^2-2*a*u-a,x);
```

$$a\,u^2\,(x-1)\,(x+1)+2\,a\,u\,(x-1)\,(x+1)+a\,(x-1)\,(x+1)$$

```
(%i20)  gcfactor(10);
```

$$(1+\%i)^2\,(1+2\,\%i)\,(2+\%i)$$

```
(%i21)  gfactor(x^4-1);
```

$$(x-1)\,(x+1)\,(x-\%i)\,(x+\%i)$$

```
(%i22)  lopow((x+y)^2 + (x+y)^a, x+y);
```

$$\min(2,a)$$

При применении функции `polynomialp(p,L)` предикат `coeffp` должен принимать значение `true` для каждого коэффициента, а предикат `exponp` – для всех показателей степени переменных в `L`.

```
(%i23)  polynomialp((x-1)*(x+y), [x,y]);
(%o23)  true
```

```
(%i24)  polynomialp((x-1)*(x+y)^a, [x,y]);
(%o24)  false
```

Дополнительные возможности работы с полиномами определены в рамках *базисов Грёбнера*, функции для работы с которыми включены в пакет `grobner`.

```
(%i25)  denom(sin(x)/10*cos(x)/y);
```

$$10\,y$$

```
(%i26)  partfrac((x^5-3*x^4+2*x^3-x^2+5*x-1)/(x^5*(x^3-1)),x);
```

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

```
(%i27)  load("lrats")$ lratsubst(a^2 = b, a^3);
```

(%o27) $a\,b$

```
(%i28)  ((x-2*y)^4/(x^2-4*y^2)^2 + 1)*(y + a)*(2*y + x)/(4*y^2 +
x^2);
```

(%o28)

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

```
(%i29)  rat(%,y,a,x);
```

(%o29) $\dfrac{2\,a+2\,y}{x+2\,y}$

```
(%i30)  ratcoef((a+b)^2/5 + 4*(a+b)/(c+1) - a - b, a + b);
```

(%o30) $-\dfrac{c-3}{c+1}$

```
(%i31)  ratdiff((4*x^3 + 10*x - 11)/(x^5 + 5),x);
```

(%o31) $-\dfrac{8\,x^7+40\,x^5-55\,x^4-60\,x^2-50}{x10+10\,x^5+25}$

```
(%i32)  ratdiff(f(x)^3 - f(x)^2 + 7,f(x));
```

(%o32) $3\,f(x)^2-2\,f(x)$

```
(%i33)  ratdiff((a+b)^3 + (a+b)^2, a+b);
```

(%o33) $3\,b^2+(6\,a+2)\,b+3\,a^2+2\,a$

```
(%i34)  ratexpand(((x-1)^2-5)/((x+1)^2+1)-(1+x^2)/(x-1)^2);
```

(%o34) $-\dfrac{6\,x^3}{x^4-x^2-2\,x+2}-\dfrac{2\,x^2}{x^4-x^2-2\,x+2}+\dfrac{4\,x}{x^4-x^2-2\,x+2}-\dfrac{6}{x^4-x^2-2\,x+2}$

```
(%i35)  ((x-1)^(3/2)-(x+1)*sqrt(x-1))/sqrt((x-1)*(x + 1));
```

(%o35) $\dfrac{(x-1)^{3/2}-\sqrt{x-1}\,(x+1)}{\sqrt{(x-1)\,(x+1)}}$

```
(%i36)  ratsimp(%);
```

(%o36) $-\dfrac{2\sqrt{x-1}}{\sqrt{x^2-1}}$

```
(%i37)   ratsubst(1-sin(x)^2, cos(x)^2, cos(x)^4+cos(x)^3+cos(x)^2
+ cos(x)+1);
(%o37)        sin(x)^4 - 3 sin(x)^2 + cos(x) (2 - sin(x)^2) + 3
```

Системная переменная	*По умолчанию*	*Описание*
`takegcd`	`true`	Управляющая переменная для функции `gcdivide`

8.4.2. Пределы

Функция	*Описание*
`limit(expr, x,val)`	Вычисление двустороннего предела выражения `expr` в точке `x = val`
`limit(expr, x,val,dir)`	Вычисление одностороннего предела выражения `expr` в точке `x = val`. Направление задается параметром `dir`

Константа	*Описание*
`infinity`	Комплексная бесконечность
`ind`	Ограниченная неопределенность
`und`	Неограниченная неопределенность

Параметрами команды `limit` являются выражение, имя переменной и предельная точка. Команда вычисляет предел заданного выражения `expr`, когда переменная `x` (при вычислении предела Maxima предполагает, что `x` является действительной переменной) стремится к величине `val` справа или слева, что определяется значением параметра `dir`. Этот параметр может иметь значение `plus` для предела справа и `минус` для предела слева. Если какое-либо значение `dir` отсутствует, то вычисляется двухсторонний предел.

Функция `limit` позволяет использовать следующие специальные символы, с помощью которых можно указывать предельные точки: `inf` (∞) и `minf` ($-\infty$)). Результат вычисления предела может быть следующим: ∞, $-\infty$, `infinity` (комплексная бесконечность), `ind` (неопределенность, но ограниченная) или `und` (неопределенность, но неограниченная). Заметим, что результатом будет `infinity`, если предел абсолютного значения выражения равен положительной бесконечности, но предел самого выражения не равен ни ∞, ни $-\infty$. Это включает случаи, когда предел комплексного аргумента является константой, как в `limit(log(x),x,minf)`, случаи, когда комплексный аргумент колеблется, как в `limit((-2)^x,x,inf)`, и случаи, когда комплексный аргумент различен для обеих сторон двухстороннего предела, как в `limit(1/x,x,0)` и пределе `limit(log(x),x,0)`.

Функция `limit` с одним аргументом часто вызывается для упрощения константных выражений, например, функция `limit(inf-1)`.

Системная переменная	*По умолчанию*	*Описание*
`lhospitallim`	4	Максимальное количество раз применения правила Лопиталя при нахождении предела
`limsubst`	`false`	Предотвращение попытки функции `limit` использовать замены неизвестных функций
`tlimswitch`	`true`	Возможность использования функцией `limit` разложение в ряд Тейлора, если предел входного выражения не может быть вычислен напрямую

Использование `lhospitallim` предотвращает бесконечный цикл в таких случаях, как `limit(cot(x)/csc(x), x, 0)`. Задание `limsubst` необходимо для того, чтобы избежать ситуаций, когда вычисление `limit (f(n)/f(n + 1),n,inf)` дает 1 без задания структуры функции `f(n)`. Установка `limsubst` в `true` позволяет замены неизвестных функций. Когда `tlimswitch` имеет значение true, команда `limit` будет использовать разложение в ряд Тейлора. Это позволяет находить пределы, такие как `limit(x/(x-1)-1/log(x),x,1,plus)`. Если же `tlimswitch` имеет значение `false` и предел входного выражения не может быть вычислен напрямую, функция `limit` возвратит невычисленное предельное выражение.

```
(%i38)   limit(sin(x)/x,x,0);
(%o38)      1
```

```
(%i39)   limit(atan(x),x,minf);
(%o39)     -π/2
```

```
(%i40)   limit(log(x),x,0);
(%o40)      infinity
```

```
(%i41)   limit(sin(1/x),x,0);
(%o41)      ind
```

```
(%i42)   limit(sin(1/x)/x,x,0);
(%o42)      und
```

Необходимо различать результаты `infinity` с ∞. Последний представляется в виде `inf`.

Также возможно вычислить пределы сверху и снизу, добавив четвертый аргумент. Это указывается значением `plus` для пределов сверху и `minus` для пределов снизу.

```
(%i43)  limit(log(x),x,0,plus);
(%o43)      −∞
```

```
(%i44)  limit(x/abs(x),x,0,plus);
(%o44)      1
```

```
(%i45)  limit(x/abs(x),x,0,minus);
(%o45)      −1
```

```
(%i46)  limit(x/abs(x),x,0);
(%o46)      und
```

Функция	*Описание*
`tlimit(expr,x, val,dir)`	Вычисление одностороннего предела выражения `expr`, зависящего от аргумента x, в ряд Тейлора в точке `x = val`
`tlimit(expr,x, val)`	Вычисление двустороннего предела выражения `expr` в точке `x = val`
`tlimit(expr)`	Аналогично `limit(expr)`

В функции `tlimit` направление приближения к предельной точке задается параметром `dir`.

```
(%i47)  tlimit((1-cos(x)^sin(x))/(x-tan(x)), x, 0);
(%o47)      −3/2
```

8.4.3. Дифференцирование функции одной переменной

8.4.3.1. Производная

Оператор	*Описание*
`diff(expr,x)`	Производная первого порядка
`diff(expr,x,n)`	Производная n-го порядка
`diff(expr)`	Дифференциал
`del(x)`	Представление дифференциала x

Maxima способна вычислять производные. Для этой цели есть команда `diff`: `diff(expr, x)` дифференцирует выражение `expr` по переменной `x`. Необязательный третий аргумент может быть указан для вычисления производных более высокого порядка. Например, параметры `expr, x, 2` должны быть указаны для вычисления второй производной выражения `expr` по переменной `x`.

```
(%i48) diff(x^a,x);
```
(%o48) $a\,x^{a-1}$

```
(%i49) diff(x^a,x,2);
```
(%o49) $(a-1)\,a\,x^{a-2}$

Когда `diff` вызывается только с одним аргументом (т.е. без передачи имени переменной), тогда Maxima вычисляет первый дифференциал данного выражения. Дифференциалы переменных представляются символом `del`.

```
(%i50) diff(sin(4*x));
```
(%o50) $4\,\cos(4\,x)\,del(x)$

Если вместо этого мы желаем иметь `dx`, то должны сделать замену, как показано в следующем примере.

```
(%i51) diff(sin(4*x))$ ev(%, del(x)=dx);
```
(%o51) $4\,dx\,\cos(4\,x)$

Оператор `'` может использоваться для предотвращения немедленного вычисления производной системой Maxima.

```
(%i52) 'diff(exp(-x^2),x)=diff(exp(-x^2),x);
```
(%o52) $\frac{d}{dx}\,\%e^{-x^2} = -2\,x\,\%e^{-x^2}$

Оператор	*Описание*
`depends(f,x)`	Объявление, что `f` – функция `x`

Системная переменная	*По умолчанию*	*Описание*
`dependencies`	`[]`	Список зависимостей

Можно работать с производными символов, например, производными некоторой функции f. Однако аргумент f должен быть представлен явно.

```
(%i53) diff(f(x),x);
```
(%o53) $\frac{d}{dx}\,f(x)$

```
(%i54) diff(f,x);
```
(%o54) 0

Однако Maxima позволяет объявлять зависимости между переменными. `diff` полностью это учитывает.

```
(%i55)  depends(f,x);
(%o55)      [f(x)]
```

```
(%i56)  diff(f,x);
(%o56)      d/dx f
```

Оператор	*Описание*
`derivdegree(expr,y,x)`	Порядок наивысшей производной `y` по `x`
`delta(t)`	Дельта-фунция Дирака

```
(%i57)  derivdegree('diff(y,x,2) + 'diff(y,z,3) + 'diff(y,x)*x^2,
y, x);
(%o57)      2
```

8.4.3.2. Формула Тейлора

Оператор	*Описание*
`taylor(expr,x,a,n)`	Разложение функции переменной `x` по формуле Тейлора в точке `a`

Формула Тейлора представляет дифференцируемую функцию в виде суммы конечного числа слагаемых, вычисляемых по значениям ее производных в одной и той же точке a, к которым добавляется остаточный член $R_n(x)$:

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!} (x-a)^k + R_n(x).$$

Если в формуле Тейлора отбросить остаточный член, то получим многочлен Тейлора, который состоит из степеней двучлена $x-a$ от нулевой до n-й:

$$f(x) \approx \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!} (x-a)^k =$$
$$= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... + \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

Представление функции по формуле Тейлора может быть легко получено с помощью команды `taylor`. В следующем примере вычисляется конкретная реализация формулы Тейлора до порядка 3 для функции e^x в точке 0. Точки указывают на невыписанный, но подразумеваемый остаточный член:

```
(%i58)  taylor(exp(x),x,0,3);
```

$$(\%o58) \quad /T/ \quad 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots$$

Ведущий символ $/T/$ обозначает, что далее записана формула Тейлора. Полученное выражение можно использовать в дальнейших расчетах. Для этого формула Тейлора должна быть сначала преобразована в полином, представленный рациональным выражением. Обозначается это символом $/R/$:

```
(%i59)  T(x):= ''(taylor(exp(x),x,0,3));
```

$$(\%o59) \quad /T/ \quad T(x) := 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots$$

```
(%i60)  T(3);
```

$$(\%o60) \quad /\mathrm{R}/ \quad 13$$

```
(%i61)  T(y);
```

$$(\%o61) \quad /R/ \quad \frac{y^3 + 3\,y^2 + 6\,y + 6}{6}$$

8.4.3.3. Исследование функций

Одной из задач дифференциального исчисления является предоставление инструментов поиска экстремумов функций. Как известно, локальные экстремумы функции f можно найти с помощью следующей последовательности действий:

1°. Найти критические точки f, т.е. точки x_0, где производная $f'(x_0)$ равна 0 или не существует.

2°. Вычислить вторую производную f'' и проверить ее знак в этих критических точках.

– если $f''(x_0) > 0$, то x_0 является точкой локального минимума;

– если $f''(x_0) < 0$, то x_0 является точкой локального максимума;

– если $f''(x_0) = 0$, то для решения вопроса о наличии локального экстремума нужны производные более высокого порядка в точке x_0.

Предположим, у нас есть функция $f(x) = x^4 - 3\,x^2 + 2$ и нам нужно вычислить все локальные экстремумы.

```
(%i62)  f(x):= x^4 - 3*x^2 + 2;
```
$$(\%o62) \qquad f(x) := x^4 - 3\,x^2 + 2$$

Чтобы найти все критические точки, нужно вычислить первую производную f' и найти все корни уравнения $f(x) = 0$.

```
(%i63)  fx: diff(f(x),x);
```
$$(\%o63) \qquad 4\,x^3 - 6\,x$$

```
(%i64)  crit: solve(fx=0,x);
```
$$(\%o64) \qquad \left[x = -\frac{\sqrt{3}}{\sqrt{2}},\ x = \frac{\sqrt{3}}{\sqrt{2}},\ x = 0\right]$$

Далее мы должны вычислить вторую производную f'' во всех этих критических точках и проверить знаки результатов:

```
(%i65)  fxx: diff(f(x),x,2);
```
$$(\%o65) \qquad 12\,x^2 - 6$$

```
(%i66)  fxx, crit[1];
```
$$(\%o66) \qquad 12$$

```
(%i67)  fxx, crit[2];
```
$$(\%o67) \qquad 12$$

```
(%i68)  fxx, crit[3];
```
$$(\%o68) \qquad -3$$

Отсюда следует, что функция $f(x)$ имеет локальные минимумы в точках $x_1 = -\sqrt{3/2}$ и $x_2 = \sqrt{3/2}$, а также локальный максимум в точке $x_3 = 0$.

8.4.4. Интегрирование функции одной переменной

Система Maxima предоставляет несколько подпрограмм для интегрирования функций одной переменной. Функция `integrate` использует большинство из них. Существует также пакет `antid`, который работает с неопределенными функциями и их производными. Для численного интегрирования существует набор адаптивных интеграторов в пакете `QUADPACK`: `quad_qag`, `quad_qags` и др., которые описаны в подразделе 10.7.

Необходимо отметить, что Maxima обрабатывает только такие подынтегральные функции, первообразные для которых выражаются в элементарных функциях (рациональных, простейших иррациональных, тригонометрических, логарифмических, показательных, степенных и т.д.) и некоторых их расширениях (типа функции ошибок, интегрального логарифма и др.).

8.4.4.1. Неопределенные интегралы

Функция	*Описание*
`integrate(expr,x)`	Неопределенный интеграл от выражения `expr` по переменной `x`
`risch(expr,x)`	Применение алгоритма Риша для вычисления неопределенного интеграла

Команда `integrate` пытается вычислить интеграл от заданного выражения в символьном виде. Существует значительное число методов интегрирования, которые используются для поиска первообразных в замкнутой форме. Если хотя бы один из этих методов дает ответ, то результатом выполнения команды является символьная форма первообразной. Как и при работе других САВ, к результату интегрирования произвольная постоянная обычно не добавляется.

```
(%i69)  integrate(sin(x)^3, x);
```

$$(\%o69)\quad \frac{\cos(x)^3}{3} - \cos(x)$$

```
(%i70)  integrate(x/sqrt(b^2 - x^2),x);
```

$$(\%o70)\quad -\sqrt{b^2 - x^2}$$

```
(%i71)  integrate(exp(-x^2),x);
```

$$(\%o71)\quad \frac{\sqrt{\pi}\,\mathrm{erf}(x)}{2}$$

Последний пример показывает, что первообразная может содержать редко встречающиеся имена функций. В этом случае в ответе присутствует так называемая функция ошибок, которая определяется формулой

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt.$$

Эта функция играет важную роль в теории вероятностей и математической статистике, но не является элементарной. Поэтому в действительности интеграл от e^{-x^2} является *неберущимся*.

Если команда **integrate** не может найти первообразную, то результатом работы этой команды будет вывод на дисплей интеграла от подынтегральной функции в стандартной форме.

```
(%i72)   integrate(tan(log(x)),x);
(%o72)        ∫ tan(log(x)) dx
```

Для эффективной работы по выполнению команды **integrate** вычислительному ядру CAB могут потребоваться некоторая информация о параметрах в подынтегральном выражении. В этом случае ядро сначала сверится с базой данных **assume**, но если необходимые данные отсутствуют, ядро запросит пользователя. В зависимости от вопроса подходящими ответами будут "**yes;**", "**no;**", "**pos;**", "**zero;**" или "**neg;**".

```
(%i73)   integrate(x^a,x);
a+1 zero or nonzero? zero;
(%o73)        log(x)
```

Чтобы избежать этого и подобных вопросов, необходимо добавить соответствующую информацию в текущую базу **facts**. При этом нужно использовать символ **equal**, а не символ =. Кроме того, рекомендуется давать команду **forget** или **kill** для предположения, когда оно больше не нужно.

```
(%i74)   assume(equal(a,-1));
(%o74)        [equal(a, -1)]
```

```
(%i75)   integrate(x^a,x);
(%o75)        log(x)
```

Системная переменная	*По умолчанию*	*Описание*
logabs	**false**	Если **true**, то при появлении логарифма в первообразной аргумент берется по модулю, иначе модуль не ставится

```
(%i76)   kill(x); a:integrate(1/x,x)$ logabs:true$ b: integrate
(1/x,x)$ [a, b];
(%o76)        [log(x), log(|x|)]
```

Системная переменная	*По умолчанию*	*Описание*
`integrate_use_rootsof`	`false`	Если `true` и знаменатель подынтегральной функции не факторизуется, то функция `integrate` возвращает интеграл в форме суммы интегралов от простейших дробей с учетом неизвестных корней знаменателя

```
(%i77)   integrate_use_rootsof: false$ integrate(1/(1+x+x^5),x);
```

$$(\%o77)\qquad \frac{\int \frac{x^2-4\,x+5}{x^3-x^2+1}\,dx}{7}-\frac{\log\left(x^2+x+1\right)}{14}+\frac{5\,\operatorname{atan}\left(\frac{2\,x+1}{\sqrt{3}}\right)}{7\,\sqrt{3}}$$

Функция	*Описание*
`changevar(expr,f(x,y),y,x)`	Замена переменной, заданной уравнением `f(x,y)=0`, во всех интегралах по переменной `x`, встречающихся в `expr`

```
(%i78)   ex: %e**sqrt(y)$ print('integrate(ex,y,0,4), "=",
changevar('integrate(ex,y,0,4), sqrt(y)-t, t, y), "=",
ev(changevar('integrate(ex,y,0,4), sqrt(y)-t, t, y),nouns))$
```

$$(\%o78)\qquad \int_0^4 \%e^{\sqrt{y}}\,dy = 2\int_0^2 t\,\%e^{t}\,dt = 2\left(\%e^2+1\right)$$

Функцию `changevar` можно применять и в функциях `sum` и `product`.

8.4.4.2. Определенные интегралы

Функция	*Описание*
`integrate(expr,x,a,b)`	Определенный интеграл от выражения `expr` по переменной `x` по промежутку от `a` до `b`
`defint(expr,x,a,b)`	Определенный интеграл от выражения `expr` по переменной `x` по промежутку от `a` до `b`
`ldefint(expr,x,a,b)`	Вычисление несобственного интеграла с бесконечным верхним пределом с помощью предела

Команда `integrate` также вычисляет определенные интегралы. Однако соответствующая часть ядра системы использует алгоритм, который работает независимо от алгоритма вычисления неопределенных интегралов. Заметим, что границами промежутка интегрирования могут быть `minf` и `inf`.

```
(%i79)  integrate(cos(x)^2*exp(x),x,0,%pi);
```

$$(\%o79)\qquad \frac{3\,\%e^{\pi}}{5}-\frac{3}{5}$$

```
(%i80)  integrate(x^2*exp(-x^2),x,minf,inf);
```

$$(\%o80)\qquad \frac{\sqrt{\pi}}{2}$$

Функция `defint` пытается вычислить определенный интеграл и вызывается функцией `integrate`, когда заданы пределы интегрирования, т.е. когда функция `integrate` вызывается как интегрирование `integrate (expr,x,a,b)`.

```
(%i81)  ldefint(%e^(-x^2),x,0,inf);
```

$$(\%o81)\qquad \frac{\sqrt{\pi}}{2}$$

8.4.4.3. Прямые и обратные преобразования Лапласа и Фурье

Функция	*Описание*
`laplace(expr,t,s)`	Преобразование Лапласа выражения `expr` по переменной `t` и с параметром `s`
`ilt(expr,s,t)`	Обратное преобразование Лапласа выражения `expr` по переменной `s` и с параметром `t`

Функция `laplace` распознает в выражении функции `delta`, `exp`, `log`, `sin`, `cos`, `sinh`, `cosh` и `erf`, а также функции `derivative`, `integrate`, `sum` и `ilt`. Если функции `laplace` не удается найти преобразование, вызывается функция `specint`, которая может найти преобразование Лапласа для выражений со специальными функциями, такими как функции `bessel_j`, `bessel_i`, ..., и может обрабатывать функцию `unit_step`. Если функция `specint` тоже не может получить изображение, то возвращается объект `laplace`.

Функция `laplace` может преобразовывать интегралы свертки вида `integrate(f(x)*g(t-x),x,0,t)`; другие виды конволюций не распознаются.

```
(%i82)  laplace(exp(2*t+a)*sin(t)*t,t,s);
```

$$(\%o82)\qquad \frac{\%e^{a}\,(2\,s-4)}{(s^2-4\,s+5)^2}$$

В функции `ilt` выражение `expr` должно быть отношением полиномов, знаменатель которых имеет только линейные и квадратичные множители. Используя функции `laplace` и `ilt` вместе с функциями `solve` или `linsolve`, пользователь может решить отдельное дифференциальное уравнение или интегральное уравнение типа свертки или систему таких уравнений.

```
(%i83)  'integrate(sinh(x)*f(t-x),x,0,t) + 2*f(t) = 4*t**2;
```

$$\int_0^t f(t-x)\,\sinh(x)\,dx + 2\,f(t) = 4\,t^2 \qquad \text{(\%o83)}$$

```
(%i84)  laplace(%,t,s);
```

$$\frac{\text{laplace}(f(t),t,s)}{s^2-1} + 2\,\text{laplace}(f(t),t,s) = \frac{8}{s^3} \qquad \text{(\%o84)}$$

```
(%i85)  linsolve([%], ['laplace(f(t),t,s)]);
```

$$[\text{laplace}(f(t),t,s) = \frac{8\,s^2-8}{2\,s^5-s^3}] \qquad \text{(\%o85)}$$

```
(%i86)  ilt(rhs(first(%)),s,t);
```

$$-8\,\cosh\left(\frac{t}{\sqrt{2}}\right) + 4\,t^2 + 8 \qquad \text{(\%o86)}$$

В системе Maxima имеется пакет `fourie`, который включает функции, позволяющие вычислять прямые и обратные преобразования Фурье в различных формах (функции `fourintcos(f,x)`, `fourintsin(f,x)`, `fourint(f,x)` и др.). Но, к сожалению, использовать функции из этого пакета в программах затруднительно. Значительные неудобства доставляют даже попытки интерактивной работы.

При этом для получения прямых и обратных косинус- и синус-преобразований Фурье может быть использована функция `integrate`:

```
(%i87)  f: sin(x)*exp(-x)$
```

```
(%i88)  kill(x)$ r1: ratsimp(integrate(f*cos(w*x),x,0,inf))*sqrt(2/
%pi);
```

$$-\frac{\sqrt{2}\,(w^2-2)}{\sqrt{\pi}\,(w^4+4)} \qquad \text{(r1)}$$

```
(%i89)  assume(x>0)$ integrate(r1*cos(w*x),w,0,inf)*sqrt(2/%pi);
```

$$\%e^{-x}\,\sin(x) \qquad \text{(\%o89)}$$

```
(%i90)  kill(x)$ r2: ratsimp(integrate(f*sin(w*x),x,0,inf))*sqrt(2/
%pi);
```

$$(\%o90) \quad -\frac{2^{3/2}\,w}{\sqrt{\pi}\,(w^4+4)}$$

```
(%i91)  assume(x>0)$ integrate(r2*sin(w*x),w,0,inf)*sqrt(2/%pi);
```

$$(\%o91) \quad \%e^{-x}\,\sin(x)$$

А теперь примеры прямых и обратных общих преобразований Фурье. Здесь требуется разбиение несобственных интегралов по промежутку от $-\infty$ до $+\infty$ на две части:

```
(%i92)  f1: cos(x)*exp(x)$ f2: cos(x)*exp(-x)$ kill(x)$
```

```
(%i93)  assume(w>0)$ r3: ratsimp(integrate(exp(%i*w*x)*f1,x,minf,0)
+ integrate(exp(%i*w*x)*f2,x,0,inf))/sqrt(2*%pi);
```

$$(\%o93) \quad \frac{2\,w^2+4}{\sqrt{2}\,\sqrt{\pi}\,(w^4+4)}$$

```
(%i94)  kill(w)$ assume(x>0)$ ratsimp(integrate(exp(-%i*w*x)*r3,w,
minf,inf))/sqrt(2*%pi);
```

$$(\%o94) \quad \%e^{-x}\,\cos(x)$$

```
(%i95)  f3: sin(x)*exp(x)$ f4: sin(x)*exp(-x)$ kill(x)$
```

```
(%i96)  assume(w>0)$ r4: ratsimp(integrate(exp(%i*w*x)*f3,x,minf,0)
+ integrate(exp(%i*w*x)*f4,x,0,inf))/sqrt(2*%pi);
```

$$(\%o96) \quad \frac{2^{3/2}\,\%i\,w}{\sqrt{\pi}\,(w^4+4)}$$

```
(%i97)  kill(w)$ assume(x>0)$ ratsimp(integrate(exp(-%i*w*x)*r4,w,
minf,inf))/sqrt(2*%pi);
```

$$(\%o97) \quad \%e^{-x}\,\sin(x)$$

```
(%i98)  f5: (cos(x)-3*sin(x))*exp(x)$ f6: (cos(x)-3*sin(x))*exp(-x)$
kill(x)$
```

```
(%i99)  assume(w>0)$ r5: ratsimp(integrate(exp(%i*w*x)*f5,x,minf,0)
+ integrate(exp(%i*w*x)*f6,x,0,inf))/sqrt(2*%pi);
```

$$(\%o99) \quad \frac{2\,w^6-20\,\%i\,w^5-60\,w^4+96\,\%i\,w^3+72\,w^2-16\,\%i\,w+16}{\sqrt{2}\,\sqrt{\pi}\,(w^8-4\,\%i\,w^7-8\,w^6+8\,\%i\,w^5+8\,w^4-16\,\%i\,w^3-32\,w^2+32\,\%i\,w+16)}$$

```
(%i100)   kill(w)$ assume(x>0)$ factor(expand(ratsimp(
integrate(exp(-%i*w*x)*r5,w,minf,inf))/sqrt(2*%pi)));
(%o100)          −%e^(−x) (3 sin(x) − cos(x))
```

8.4.5. Функции нескольких переменных

8.4.5.1. Частные производные и дифференциал

Функция	*Описание*
`diff(expr,x_1,n_1,x_2,n_2,...)`	Смешанная частная производная
`diff(expr)`	Полный дифференциал выражения `expr`
`implicit_derivative(f,indvars, orderlist,depvar)`	Производные неявной функции

Смешанные частные производные могут быть вычислены путем вложенных вызовов `diff` или перечисления всех переменных в качестве аргументов.

```
(%i101)   diff(diff(x^3*y^2,x),y);
(%o101)          6 x^2 y
```

```
(%i102)   diff(x^3*y^2,x,1,y,1);
(%o102)          6 x^2 y
```

```
(%i103)   diff(x^3*y^2,x,2,y,1);
(%o103)          12 x y
```

Необходимо еще раз отметить, что для смешанных частных производных каждое имя переменной должно сопровождаться порядком соответствующей производной.

Если `diff` вызывается только с одним аргументом, то для функции нескольких переменных Maxima вычисляет полный дифференциал данного выражения. Дифференциалы переменных представлены символом `del`.

```
(%i104)   diff(sin(y*x));
(%o104)          x cos(x y) del(y) + y cos(x y) del(x)
```

Если вместо этого нужны стандартные обозначения дифференциалов независимых переменных (`dx` и `dy`), то необходима замена переменных в виде, показанном в следующем примере.

```
(%i105)  diff(sin(y*x))$ ev(%, del(x) = dx, del(y) = dy);
```
$$(\%o105) \qquad dx\, y\, \cos(x\, y) + dy\, x\, \cos(x\, y)$$

Оператор	*Описание*
`depends(f,[x_1,,x_2,...])`	Объявление того, что функция `f` зависит от переменных `x_1`, `x_2`, ...
`depends([f_1,f_2,...],[x_1,x_2,...])`	Объявление того, что функции `f_1`, `f_2`, ... зависят от переменных `x_1`, `x_2`, ...

Системная переменная	*По умолчанию*	*Описание*
`dependencies`	`[]`	Список зависимостей

Так же, как в случае функции одной переменной, можно работать с производными неопределенных ранее функций, например, с производными некоторой функции `f`. Однако любые зависимости `f` должны быть представлены явно.

```
(%i106)  diff(f(x,y),x);
```
$$(\%o106) \qquad \frac{d}{dx} f(x,y)$$

Как видим, при выводе результата используется символ обычной, а не частной производной.

```
(%i107)  diff(f,x);
```
$$(\%o107) \qquad 0$$

Наверное, не тот результат, который хотелось бы иметь... Однако выше было показано, что Maxima позволяет объявлять зависимости между переменными, и если это использовать, то `diff` такие зависимости учтет.

```
(%i108)  depends(f,[x,y]);
```
$$(\%o108) \qquad [f(x,y)]$$

```
(%i109)  diff(f,x,2,y,3);
```
$$(\%o109) \qquad \frac{d^5}{dx^3\, dy^2} f$$

Теперь предположим, что само `x` зависит от некоторой другой переменной `t`. Тогда мы можем вычислить производную сложной функции $f\big(x(t)\big)$ по t с помощью цепного правила:

$$\frac{df\big(x(t)\big)}{dt} = \frac{df(x)}{dx}\,\frac{dx(t)}{dt}.$$

Чтобы продемонстрировать это правило, можно попытаться выполнить следующее:

```
(%i110)  diff(f(x(t)),t);
(%o110)      d/dt f(x(t))
```

Однако так желаемый результат не получается. Решение состоит в том, чтобы объявить зависимость `x` от `t` (заметим, что это объявление приводит к зависимости `f` от `t`).

```
(%i111)  depends(x,t);
(%o111)      [x(t)]
```

```
(%i112)  diff(f,t);
(%o112)      (d/dx f) (d/dt x)
```

Нужно удалять зависимости, когда они больше не нужны. Результат работы функции `dependencies` содержит список всех зависимостей, объявленных до сих пор. Они могут быть удалены с помощью функции `kill`.

```
(%i113)  dependencies;
(%o113)      [f(x,y),x(t)]
```

```
(%i114)  kill(f)$
```

```
(%i115)  dependencies;
(%o115)      [x(t)]
```

Вот это на самом деле более интересный аппарат. Предположим, нам нужны производные функции $f(x(t),t)$ по x и по t.

```
(%i116)  depends(f,[x,t]);
(%o116)      [f(x,t)]
```

```
(%i117)  depends(x,t);
(%o117)      [x(t)]
```

```
(%i118)  diff(f,x);
(%o118)      d/dx f
```

```
(%i119)  diff(f,t);
```
$$\left(\frac{d}{dx}\,f\right)\left(\frac{d}{dt}\,x\right)+\frac{d}{dt}\,f$$

```
(%i120)  kill(all)$
```

При вычислении полного дифференциала Maxima рассматривает каждый неизвестный символ как переменную. Тем не менее мы можем объявить некоторые из этих переменных как константы, используя команду `declare`.

```
(%i121)  diff(a*x^2+b*y^2);
```
$$2\,b\,y\,del(y)+2\,a\,x\,del(x)+y^2\,del(b)+x^2\,del(a)$$

```
(%i122)  declare([a,b], constant);
(%o122)  done
```

```
(%i123)  diff(a*x^2+b*y^2);
```
$$2\,b\,y\,del(y)+2\,a\,x\,del(x)$$

Функция `implicit_derivative(f, indvarlist,orderlist,depvar)` вычисляет производные неявно заданных функций многих переменных. Параметры: `f` – функция массива; `indvarlist` – список независимых переменных; `orderlist` – максимальные порядки производных по каждой из независимых переменных; `depvar` – имя зависимой переменной.

```
(%i124)  load("impdiff")$ f[0,0]: p^2+q^3-r^4 = 0$
implicit_derivative(f, [p,q], [1,1], r)$ display_array(f);
```
$$-r^4+q^3+p^2=0$$
$$f[0,1]=\frac{3\,q^2}{4\,r^3}$$
$$f[1,0]=\frac{p}{2\,r^3}$$
$$f[1,1]=-\frac{9\,p\,q^2}{8\,r^7}$$

```
(%o124)  done
```

8.4.5.2. Задание соотношений для частных производных

Функция	*Описание*
`gradef(f(x1,...,xn),g1,...,gn)`	Задание частных производных для функции `f` по переменным `x1`, ..., `xn`
`gradef(h,x,expr)`	
`depends(h,x1,...,xn)`	Задание зависимости функции `h` от переменных `x1`, ..., `xn`

Выражение `gradef(f(x1,...,xn),g1,...,gn)` определяет `g1`, ..., `gn` как частные производные функции `f(x1,...,xn)` по переменным `x1`, ..., `xn` соответственно.

Зависимости между переменными можно явно указать при помощи функции `depends`, которая позволяет декларировать, что некая переменная зависит от одной или нескольких других переменных, т.е. является их функцией. Например, если зависимость `f` от `x` отсутствует, то выражение `diff(f,x)` дает 0. Если предварительно установить `depends(f,x)`, то вызов `diff(f,x)` возвращает символьную производную.

При помощи функции `gradef` можно определить производные некоторой функции, даже если она сама неизвестна или ее явные производные использовать неудобно, например, из-за их громоздкости.

```
(%i125)   gradef(f(x,y), -sin(x)*cos(y), -cos(x)*sin(y));
(%o125)       f(x,y)
```

```
(%i126)   diff(f(x,y));
(%o126)       - cos(x) sin(y) del(y) - sin(x) cos(y) del(x)
```

```
(%i127)   gradef(g(x,y), -(x-a)*g(x,y), -(y-b)*g(x,y));
(%o127)       g(x,y)
```

```
(%i128)   g(x0,y0) + subst([x=x0,y=y0],diff(g(x,y),x))*(x-x0) +
subst([x=x0,y=y0],diff(g(x,y),y))*(y-y0);
(%o128)       g(x0,y0) (b - y0) (y - y0) + (a - x0) (x - x0) g(x0,y0) + g(x0,y0)
```

8.4.5.3. Векторный анализ

Функция	*Описание*
`v1~v2`	Векторное произведение
`grad(u)`	Градиент функции
`div(v)`	Дивергенция вектор-функции
`curl(v)`	Ротор вектор-функции
`laplacian(u)`	Лапласиан функции
`potential(givengradient)`	Потенциал градиента
`vectorpotential(givencurl)`	Векторный потенциал ротора
`vectorsimp(expr)`	Упрощение выражения `expr`

```
(%i129)   load("vect");
(%o129)     "C:/maxima-5.43.0/share/maxima/5.43.0/share/vector/vect.mac"
```

```
(%i130)   u: [u1(x,y,z), u2(x,y,z), u3(x,y,z)]$ v: [v1(x,y,z),
v2(x,y,z), v3(x,y,z)]$ w: [w1(x,y,z), w2(x,y,z), w3(x,y,z)]$
```

```
(%i131)   vectorsimp(express((u~v)~w+(v~w)~u+(w~u)~v)),
expandall;
(%o131)       [0,0,0]
```

```
(%i132)   express(curl(u));
(%o132)       [ d/dy u3(x,y,z) - d/dz u2(x,y,z), d/dz u1(x,y,z) - d/dx u3(x,y,z),
d/dx u2(x,y,z) - d/dy u1(x,y,z) ]
```

```
(%i133)   express(div(u));
(%o133)       d/dz u3(x,y,z) + d/dy u2(x,y,z) + d/dx u1(x,y,z)
```

```
(%i134)   express(grad(q(x,y,z)));
(%o134)       [ d/dx q(x,y,z), d/dy q(x,y,z), d/dz q(x,y,z) ]
```

```
(%i135)   express(div(grad(q(x,y,z)))-laplacian(q(x,y,z)));
(%o135)       0
```

Функция `potential` использует системную переменную `potential zeroloc[0]`.

Функция `vectorpotential` вычисляет ротор данного вектора в текущей системе координат. Переменная `potentialzeroloc` трактуется так же, как и при использовании функции `potential`, но порядок левых частей равенств должен быть циклической перестановкой координат.

Функция `vectorsimp(expr)` упрощает выражение `expr` в соответствии со следующими флажками: `expandall`, `expanddot`, `expanddotplus`, `expandcross`, `expandcrossplus`, `expandcrosscross`, `expandgrad`, `expandgradplus`, `expandgradprod`, `expanddiv`, `expanddivplus`, `expanddivprod`, `expandcurl`, `expandcurlplus`, `expandcurlcurl`, `expandlaplacian`, `expandlaplacianplus` и `expandlaplacianprod`.

По умолчанию все эти флажки имеют значение `false`. В имени суффикс `plus` относится к использованию свойств аддитивности или дистрибутивности. Суффикс `prod` относится к упрощению для операндов, которые являются произведениями любого типа.

Флажок `expandcrosscross` выражение `p~(q~r)` способствует замене на `(p.r)*q - (p.q)*r`, `expandcurlcurl` – `curl(curl(p))` на `grad(div(p)`

+ `div(grad(p))`, `expandlaplaciantodivgrad` – `laplacian(p)` на `div(grad(p))`.

8.4.5.4. Якобиан, гессиан, вронскиан

Функция	*Описание*
`jacobian(f,x)`	Матрица Якоби вектор-функции `f` нескольких переменных, заданных вектором `x`
`hessian(f,x)`	Гессиан функции `f` векторного аргумента `x`
`wronskian([f_1,...,f_n],x)`	Матрица Вронского списка функций `[f_1,...,f_n]`по переменной `x`

В системе Maxima вектор-функции, т.е. функции из пространства $\mathbb{R}^n$ в пространство $\mathbb{R}^m$, представлены векторами (однозначных) функций.

```
(%i136)  f(x,y):= [x^2 + y^2, x^2 - y^2];
```
$$(\%o136)\qquad f(x,y) := [x^2+y^2, x^2-y^2]$$

Формальной производной таких функций будут матрицы Якоби.

```
(%i137)  jacobian(f(x,y),[x,y]);
```
$$(\%o137)\qquad \begin{pmatrix} 2\,x & 2\,y \\ 2\,x & -2\,y \end{pmatrix}$$

Приведем пример получения матрицы Якоби для неопределенной вектор-функции:

```
(%i138)  f(x,y):= [f[1](x,y),f[2](x,y)];
```
$$(\%o138)\qquad f(x,y) := [f_1(x,y), f_2(x,y)]$$

```
(%i139)  jacobian(f(x,y),[x,y]);
```
$$(\%o139)\qquad \begin{pmatrix} \frac{d}{dx} f_1(x,y) & \frac{d}{dy} f_1(x,y) \\ \frac{d}{dx} f_2(x,y) & \frac{d}{dy} f_2(x,y) \end{pmatrix}$$

Градиент обычной функции нескольких переменных тоже может быть вычислен как матрица Якоби. Заметим, что функция должна быть представлена в виде списка с одним элементом. Результатом является $(1 \times n)$-матрица.

```
(%i140)   jacobian([x^3 + y^2], [x,y]);
(%o140)       [3 x^2, 2 y]
```

Гессиан функции $f(\boldsymbol{x})$ нескольких переменных вычисляются по следующей формуле:

$$\boldsymbol{\mathcal{H}}(\boldsymbol{f},\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_1\,\partial x_n} \\ \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_2\,\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_n^2} \end{bmatrix}$$

Команда `hessian` позволяет вычислить гессиан для заданной функции.

```
(%i141)   hessian(x^3+4*x*y+y^2, [x,y]);
(%o141)       matrix([6 x, 4], [4, 2])
```

Есть возможность вычислять гессианы от неопределенных функций:

```
(%i142)   depends(g,[x,y])$ hessian(g,[x,y]);
(%o142)       matrix([d^2/dx^2 g, d^2/(dx dy) g], [d^2/(dx dy) g, d^2/dy^2 g])
```

Определитель матрицы Вронского списка функций является вронскианом списка функций и вычисляется по следующей формуле:

$$W(\boldsymbol{f},x) = \begin{vmatrix} \dfrac{\partial f_1(x)}{\partial x} & \dfrac{\partial f_2(x)}{\partial x} & \cdots & \dfrac{\partial f_n(x)}{\partial x} \\ \dfrac{\partial^2 f_1(x)}{\partial x^2} & \dfrac{\partial^2 f_2(x)}{\partial x^2} & \cdots & \dfrac{\partial^2 f_n(x)}{\partial x^2} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial^{n-1} f_1(x)}{\partial x^{n-1}} & \dfrac{\partial^{n-1} f_2(x)}{\partial x^{n-1}} & \cdots & \dfrac{\partial^{n-1} f_n(x)}{\partial x^{n-1}} \end{vmatrix}$$

Перед использованием функции `wronskian` требуется загрузка пакета `functs`.

```
(%i143)   load(functs)$
```

```
(%i144)  wronskian([f(x), g(x)],x);
```

$$(\%o144)\qquad \begin{pmatrix} f(x) & g(x) \\ \frac{d}{d\,x}\,f(x) & \frac{d}{d\,x}\,g(x) \end{pmatrix}$$

8.4.5.5. Системы координат

В системе Maxima имеется пакет `vect_transform`, содержащий информацию по наиболее часто используемым *системам координат*. Пакет в основном состоит из операторов присваивания, левая часть которых – имя переменной, соответствующей системе координат, а правая – список, имеющий следующую структуру: `[<подсписок формул>, <последовательность координат>]`. Длины подсписка и последовательности совпадают и равны размерности соответствующих пространств. При этом последовательность определяет имена криволинейных координат, а формулы задают связи декартовых координат с соответствующими криволинейными. В частности, ниже приведен пример работы с 3D-декартовыми и сферическими координатами.

```
(%i145)  load(vect_transform)$ listofvars(coords);
```

$$(\%o145)\qquad [x, y, z, r, \theta, \phi, e, u, v, f, w, g]$$

```
(%i146)  cartesian3d;
```

$$(\%o146)\qquad [[x, y, z], x, y, z]$$

```
(%i147)  spherical;
```

$$(\%o147)\qquad [[\cos(\phi)\,r\,\sin(\theta), \sin(\phi)\,r\,\sin(\theta), r\,\cos(\theta)], r, \theta, \phi]$$

```
(%i148)  jacobian(spherical[1],[spherical[2],spherical[3],spherical
[4]]);
```

$$(\%o148)\qquad \begin{pmatrix} \cos(\phi)\,sin(\theta) & \cos(\phi)\,r\,\cos(\theta) & -\sin(\phi)\,r\,\sin(\theta) \\ \sin(\phi)\,\sin(\theta) & \sin(\phi)\,r\,\cos(\theta) & \cos(\phi)\,r\,\sin(\theta) \\ \cos(\theta) & -r\,\sin(\theta) & 0 \end{pmatrix}$$

```
(%i149)  trigsimp(determinant(%));
```

$$(\%o149)\qquad r^2\,\sin(\theta)$$

Среди других систем координат, определяемых в пакете, отметим следующие: 2D-декартовы, полярные, эллиптические, параболические, цилиндрические, эллиптико-цилиндрические, параболо-цилиндрические, тороидальные, конические и др.

Заметим, что аппарат, связанный с преобразованием координат, есть и в пакете `ctensor`.

8.4.5.6. Формула Тейлора

Оператор	*Описание*
`taylor(expr,[x,y],[a,b],n)`	Разложение функции переменных `x` и `y` по формуле Тейлора в точке `(a,b)`

Формула Тейлора может быть использована для представления функций нескольких переменных в заданной точке. Ниже приводится простой пример такого представления.

```
(%i150)   taylor(exp(x^2 + y),[x,y],[0,1],2);
```

$$(\%o150) \qquad /T/ \quad \%e + \%e\,(y-1) + \frac{2\,\%e\,x^2 + \%e\,(y-1)^2}{2} + \ldots$$

8.4.5.7. Экстремум функции нескольких переменных

Экстремумы функции f нескольких переменных можно найти по следующей схеме:

1°. Найти критические точки f, т.е. точки $\boldsymbol{x}^*$, где верно векторное равенство $\nabla f(\boldsymbol{x}^*) = 0$, а

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \ldots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}$$

2°. Вычислить гессиан $\boldsymbol{\mathcal{H}}(f, \boldsymbol{x})$ функции f и проверить его определенность в этих критических точках:

– если гессиан $\boldsymbol{\mathcal{H}}(f, \boldsymbol{x})$ положительно определен, то точка $\boldsymbol{x}^*$ является точкой локального минимума;

– если гессиан $\boldsymbol{\mathcal{H}}(f, \boldsymbol{x})$ отрицательно определен, то точка $\boldsymbol{x}^*$ является точкой локального максимума;

– если определитель гессиана $\left|\boldsymbol{\mathcal{H}}(f, \boldsymbol{x})\right| = 0$, то для анализа поведения функции f в точке $\boldsymbol{x}^*$ нужны производные более высокого порядка, чем вторые;

– если гессиан не является знакоопределенным (принимает как положительные, так и отрицательные значения) и невырожден ($\left|\boldsymbol{\mathcal{H}}(f, \boldsymbol{x})\right| \neq 0$), то $\boldsymbol{x}^*$ – седловая точка функции $f(\boldsymbol{x})$.

Вычисление локальных экстремумов для функций с двумя переменными и более – гораздо более сложная задача, чем для одномерных функций. Например, рассмотрим следующую функцию:

```
(%i151)  f(x,y):=1/6*x^3 - x + 1/4*x*y^2;
```
$$(\%o151)\qquad f(x,y) := \frac{1}{6}\,x^3 - x + \frac{1}{4}\,x\,y^2$$

Сначала необходимо найти критические точки f. Для этого вычисляем градиент f и решаем систему уравнений:

```
(%i152)  [fx,fy]: [diff(f(x,y),x), diff(f(x,y),y)];
```
$$(\%o152)\qquad [\frac{y^2}{4} + \frac{x^2}{2} - 1, \frac{x\,y}{2}]$$

```
(%i153)  crit: solve([fx=0,fy=0],[x,y]);
```
$$(\%o153)\qquad [[x = -\sqrt{2}, y = 0], [x = \sqrt{2}, y = 0], [x = 0, y = 2], [x = 0, y = -2]]$$

Далее вычисляем гессиан функции f:

```
(%i154)  hess: hessian(f(x,y),[x,y]);
```
$$(\%o154)\qquad \begin{bmatrix} x & \frac{y}{2} \\ \frac{y}{2} & \frac{x}{2} \end{bmatrix}$$

Существуют два метода оценки определенности симметричной матрицы $\mathcal{A}$. Первый рассматривает собственные значения матрицы:

– $\mathcal{A}$ положительно определена, если все собственные значения $\mathcal{A}$ положительны;

– $\mathcal{A}$ отрицательно определена, если все собственные значения $\mathcal{A}$ отрицательны;

– $\mathcal{A}$ неопределенна, если $\mathcal{A}$ имеет положительные и отрицательные собственные значения.

```
(%i155)  H1: ev(hess, crit[1]);
```
$$(\%o155)\qquad \begin{bmatrix} -\sqrt{2} & 0 \\ 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

```
(%i156)  eigenvalues(H1);
```
$$(\%o156)\qquad [[-\frac{1}{\sqrt{2}}, -\sqrt{2}], [1, 1]]$$

```
(%i157)  H3: ev(hess, crit[3]);
```
$$(\%o157)\qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
(%i158)  eigenvalues(H3);
```
$$(\%o158)\qquad [[-1, 1], [1, 1]]$$

Следовательно, точка $\boldsymbol{x}_1 = \left(-\sqrt{2}, 0\right)^\intercal$ – точка локального максимума, а $\boldsymbol{x}_2 = \left(0, 2\right)^\intercal$ – седловая точка.

Во втором методе используются угловые миноры M_k симметричной матрицы $\boldsymbol{\mathcal{A}}$, которые являются определителями левых верхних $k \times k$-подматриц матрицы $\boldsymbol{\mathcal{A}}$:

$$M_k = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1k} \\ a_{21} & a_{22} & \ldots & a_{2k} \\ \ldots & \ldots & \ldots & \ldots \\ a_{k1} & a_{k2} & \ldots & a_{kk} \end{pmatrix}.$$

Отсюда будем иметь:

– $\boldsymbol{\mathcal{A}}$ положительно определена, если $M_k > 0$ для всех $k = 1, 2, \ldots, n$;

– $\boldsymbol{\mathcal{A}}$ отрицательно определена, если $(-1)^k M_k > 0$ для всех $k = 1, 2, \ldots, n$;

– $\boldsymbol{\mathcal{A}}$ неопределенна, если $M_n = \left|\boldsymbol{\mathcal{A}}\right| \neq 0$, но ни один из вышеперечисленных случаев не выполняется.

Заметим, что для матрицы 2×2 это сводится к условиям:

– $\boldsymbol{\mathcal{A}}$ положительно определена, если $a_{11} > 0$ и $\left|\boldsymbol{\mathcal{A}}\right| > 0$;

– $\boldsymbol{\mathcal{A}}$ отрицательно определена, если $a_{11} < 0$ и $\left|\boldsymbol{\mathcal{A}}\right| > 0$;

– $\boldsymbol{\mathcal{A}}$ неопределенна, если $\left|\boldsymbol{\mathcal{A}}\right| < 0$.

```
(%i159)  H2: ev(hess, crit[2]);
```

$$\begin{bmatrix} \sqrt{2} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

```
(%i160)  [H2[1,1], determinant(H2)];
```

$$[\sqrt{2}, 1]$$

```
(%i161)  H4: ev(hess, crit[4]);
```

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

```
(%i162)  [H4[1,1], determinant(H4)];
```

$$[0, -1]$$

Следовательно, точка $\boldsymbol{x}_3 = \left(\sqrt{2}, 0\right)^\intercal$ – точка локального минимума, а $\boldsymbol{x}_4 = \left(0, -2\right)^\intercal$ – седловая точка. Чтобы убедиться в соответствии решения истине, нарисуем график функции f (рис. 8.1) и ее изолинии (рис. 8.2).

```
(%i163)   plot3d(f(x,y), [x,-2,2], [y,-2.5,2.5])$
```

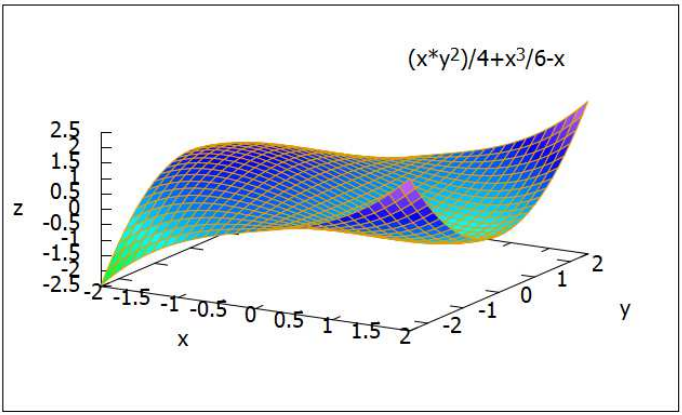


Рис. 8.1

```
(%i164)   contour_plot(f(x,y), [x,-3,3], [y,-3,3], [grid, 100, 100],
[legend,false], [gnuplot_preamble, "set cntrparam levels 50"])$
```

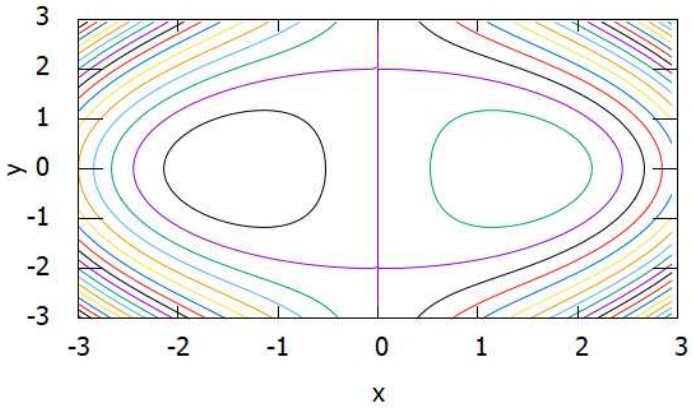


Рис. 8.2

Рассмотрим процедуру решения задачи на условный экстремум: найти минимальные и максимальные значения функции $f(\boldsymbol{x})$ при выполнении равенств $g_k(\boldsymbol{x}) = 0$ ($k = 1, 2, ..., m$, $m < n$). Для получения ответа

строится функция Лагранжа:

$$L(\boldsymbol{x},\boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{k=1}^{m} \lambda_k\, g_k(\boldsymbol{x}),$$

где λ_k – неопределенные множители Лагранжа. А далее указанным выше способом ищется решение задачи на безусловный экстремум в терминах векторных переменных $\boldsymbol{x}$ и $\boldsymbol{\lambda}$. Точки безусловных экстремумов функции $L(\boldsymbol{x},\boldsymbol{\lambda})$ (при отбрасывании $\boldsymbol{\lambda}$) дают точки условных экстремумов для функции $f(\boldsymbol{x})$.

Пример 8.1. Продемонстрируем описанную процедуру на решении следующей задачи: найти минимальное и максимальное значение функции $f = 5 - 3\,x - 4\,y$ при условии, что $x^2 + 4\,y^2 = 16$ (рис. 8.3, где при построении эллиптического цилиндра использовалось его параметрическое уравнение).

◀ Обозначим: $g = x^2 + 4\,y^2 - 16$.

```
(%i166)   f: 5 - 3*x - 4*y$ g: x^2 + 4*y^2 - 16$
```

Функция Лагранжа в данной задаче будет иметь следующий вид:

$$L(x,y,\lambda) = 5 - 3\,x - 4\,y + \lambda\left(x^2 + 4\,y^2 - 16\right),$$

```
(%i167)   L: f + lambda*g$
```

Находим частные производные функции Лагранжа:

$$\begin{aligned}
\frac{\partial L}{\partial \lambda} &= x^2 + 4\,y^2 - 16,\\
\frac{\partial L}{\partial x} &= -3 + 2\,\lambda\,x,\\
\frac{\partial L}{\partial y} &= -4 + 4\,\lambda\,y.
\end{aligned}$$

```
(%i168)   gradL: jacobian([L],[lambda,x,y])[1];
(%o168)       [4*y^2+x^2-16, 2*x*lambda-3, 8*y*lambda-4]
```

Приравниваем частные производные L к нулю:

$$x^2 + 4\,y^2 - 16 = 0, \qquad -3 + 2\,\lambda\,x = 0, \qquad -4 + 8\,\lambda\,y = 0,$$

```
(%i165)   draw3d(dimensions = [1000, 1000], view = [30, 20],
enhanced3d = false, proportional_axes = xy, axis_3d = false, xaxis =
true, yaxis = true, zaxis = true, xaxis_type = solid, zaxis_type
= solid, yaxis_type = solid, xaxis_color = black, zaxis_color
= black, yaxis_color = black, xaxis_width = 2, yaxis_width =
2, zaxis_width = 2, xlabel = "x", ylabel = "y", xyplane = 0,
surface_hide = true, color = blue, explicit(f1, x, -5, 5, y, -5,
5), color = red, parametric_surface( 4*cos(phi), 2*sin(phi), z, phi,
0, 2*%pi, z, -30, 30), point_type = filled_circle, point_size = 1.5,
points(pts))$
```

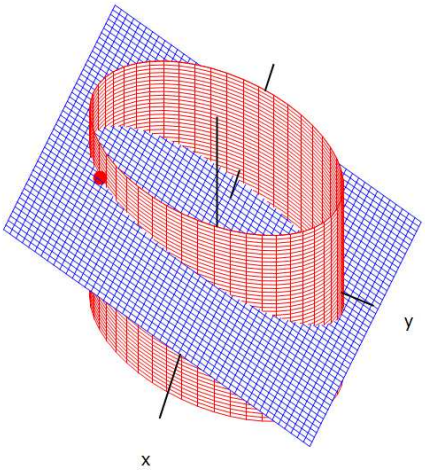


Рис. 8.3

а затем решаем полученную систему уравнений. Для этого находим x и y из второго и третьего уравнений ($\lambda \neq 0$): $x = 3/(2\,\lambda)$, $y = 1/(2\,\lambda)$, и подставляем их в первое уравнение:

$$\frac{9}{4\,\lambda^2} + \frac{4}{4\,\lambda^2} = 16,$$

откуда следует, что $\lambda_{1,2} = \pm\sqrt{13}/8$, а следовательно, $x_{1,2} = \pm 2/\sqrt{13}$, $y_{1,2} = \pm 4/\sqrt{13}$.

```
(%i169)  cp: solve(gradL);
```

$$(\%o169)\qquad [[\lambda = \frac{\sqrt{13}}{8},\ y = \frac{4}{\sqrt{13}},\ x = \frac{2}{\sqrt{13}}],[\lambda = -\frac{\sqrt{13}}{8},\ y = -\frac{4}{\sqrt{13}},\ x = -\frac{2}{\sqrt{13}}]]$$

```
(%i170)  pts: [ev([x,y,f1],cp[1]), ev([x,y,f1],cp[2])];
```

$$(\%o170)\qquad [[\frac{2}{\sqrt{13}},\frac{4}{\sqrt{13}},5-4\sqrt{13})],[-\frac{2}{\sqrt{13}},-\frac{4}{\sqrt{13}},4\sqrt{13}+5]]$$

Подставим значения $\lambda_{1,2} = \pm\sqrt{13}/8$ в функцию Лагранжа:

$$L_1 = 5 - 3\,x - 4\,y + \frac{\sqrt{13}}{8}\left(x^2 + 4\,y^2 - 16\right),$$
$$L_2 = 5 - 3\,x - 4\,y - \frac{\sqrt{13}}{8}\left(x^2 + 4\,y^2 - 16\right).$$

```
(%i171)  lambda0: [ev(lambda,cp[1]), ev(lambda,cp[2])];
```

$$(\%o171)\qquad [\frac{\sqrt{13}}{8},-\frac{\sqrt{13}}{8}]$$

```
(%i172)  L1: ev(L,lambda=lambda0[1]);
```

$$(\%o172)\qquad \frac{\sqrt{13}(4\,y^2+x^2-16)}{8} - 4\,y - 3\,x + 5$$

```
(%i173)  L2: ev(L,lambda=lambda0[2]);
```

$$(\%o173)\qquad -\frac{\sqrt{13}(4\,y^2+x^2-16)}{8} - 4\,y - 3\,x + 5$$

Находим гессианы функций L_1 и L_2:

$$\boldsymbol{\mathcal{H}}_1 = \begin{bmatrix} \frac{\sqrt{13}}{4} & 0 \\ 0 & \sqrt{13} \end{bmatrix}, \qquad \boldsymbol{\mathcal{H}}_1 = \begin{bmatrix} -\frac{\sqrt{13}}{4} & 0 \\ 0 & -\sqrt{13} \end{bmatrix}.$$

```
(%i174)  hessL1: hessian(L1,[x,y]);
```

$$(\%o174)\qquad \begin{bmatrix} \frac{\sqrt{13}}{4} & 0 \\ 0 & \sqrt{13} \end{bmatrix},$$

```
(%i175)  hessL2: hessian(L2,[x,y]);
```

$$(\%o175)\qquad \begin{bmatrix} -\frac{\sqrt{13}}{4} & 0 \\ 0 & -\sqrt{13} \end{bmatrix}$$

А теперь находим вторые дифференциалы функций L_1 и L_2:

$$d^2L_1 = \begin{bmatrix} dx & dy \end{bmatrix} \boldsymbol{\mathcal{H}}_1 \begin{bmatrix} dx \\ dy \end{bmatrix} =$$

$$= \begin{bmatrix} dx, dy \end{bmatrix} \begin{bmatrix} \frac{\sqrt{13}}{4} & 0 \\ 0 & \sqrt{13} \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} = \frac{\sqrt{13}}{4}\, dx^2 + \sqrt{13}\, dy^2,$$

$$d^2L_2 = \begin{bmatrix} dx, dy \end{bmatrix} \boldsymbol{\mathcal{H}}_2 \begin{bmatrix} dx \\ dy \end{bmatrix} =$$

$$= \begin{bmatrix} dx & dy \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{13}}{4} & 0 \\ 0 & -\sqrt{13} \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} = -\frac{\sqrt{13}}{4}\, dx^2 - \sqrt{13}\, dy^2,$$

```
(%i176)   d2L1: expand( [dx, dy].(hessL1.transpose([dx,dy])));
(%o176)       sqrt(13) dy^2 + sqrt(13) dx^2
                              ---------------
                                      4
```

```
(%i177)   d2L2: expand( [dx,dy].(hessL2.transpose([dx,dy])));
(%o177)       - sqrt(13) dy^2 - sqrt(13) dx^2
                                ---------------
                                       4
```

т.е. в точке $P_1(x_1, y_1)$ второй дифференциал L положителен, а в точке $P_2(x_2, y_2)$ второй дифференциал L отрицателен, откуда следует, что точка P_1 – точка условного минимума, а точка P_2 – точка условного максимума. ▶

8.4.5.8. Кратные интегралы

Отдельной команды для вычисления кратных интегралов в системе Maxima нет. Для решения данной проблемы необходимо воспользоваться функцией `integrate` необходимое число, предварительно вручную сведя такие интегралы к повторным и, если необходимо, сделав замену переменных. Ниже примеры вычисления двойных интегралов от функции $u = \sin(x + y)$ по области: 1) $\mathbb{D} = \{(x, y) \,|\, 0 \leqslant x, y \leqslant \pi/2\}$; 2) $\mathbb{D} = \{(x, y) \,|\, x \geqslant 0, y \geqslant 0, x + y \leqslant \pi/2\}$:

```
(%i178)   integrate(integrate(sin(x+y),x,0,%pi/2),y,0,%pi/2);
(%o178)       2
```

```
(%i179)   integrate(integrate(sin(x+y),x,0,%pi/2-y),y,0,%pi/2);
     Is 2*y-π positive, negative or zero?negative;
(%o179)       1
```

А теперь примеры вычисления тройных интегралов.

Пример 8.2. Найти объем тела [8, с. 518], ограниченного следующими поверхностями: $z = x^2 + y^2$, $z = 2\,x^2 + 2\,y^2$, $y = x$, $y = x^2$.

```
(%i180)  integrate(integrate(integrate(1, z, x^2 + y^2, 2*x^2 +
2*y^2), y, x^2, x), x, 0, 1);
```

$$(\%o180) \qquad \frac{3}{35}$$

Пример 8.3. Вычислить интеграл [8, с. 511]

$$\iiint_{\mathbb{V}} \sqrt{x^2+y^2+z^2}\,dx\,dy\,dz, \qquad \mathbb{V} = \{(x,y,z)\,|\,x^2+y^2+z^2 \leqslant z\}.$$

◀ Перейдем к сферическим координатам по формулам

$$x = r\,\cos\theta\,\sin\varphi, \quad y = r\,\sin\theta\,\sin\varphi, \quad z = r\,\cos\varphi, \quad 0 \leqslant \theta \leqslant 2\,\pi, \quad 0 \leqslant \varphi \leqslant \pi.$$

Приняв во внимание пределы изменения θ и φ, получим, что

$$\iiint_{\mathbb{V}} \sqrt{x^2+y^2+z^2}\,dx\,dy\,dz = \int_0^{\pi/2} \sin\varphi\,d\varphi \int_0^{2\,\pi} d\theta \int_0^{\cos\varphi} r^3\,dr.$$

Тогда

```
(%i181)  integrate(sin(phi)*integrate(integrate(r^3, r, 0,
cos(phi)), theta, 0, 2*%pi), phi, 0, %pi/2);
```

$$(\%o181) \qquad \frac{\pi}{10}$$

8.4.6. Ряды

Система Maxima включает средства работы с различными рядами. Некоторые функции можно использовать для работы с рядами нескольких типов.

Системная переменная	*По умолчанию*	*Описание*
`cauchysum`	`false`	Если `true` и `sumexpand=true`, то применяются правило Коши свертывания произведений рядов

```
(%i182)  sumexpand: false$ cauchysum: false$ sum(f[i],i,0,inf)*sum
(g[j],j,0,inf);
```

$$(\%o182) \qquad \left(\sum_{i=0}^{\infty} f_i\right) \sum_{j=0}^{\infty} g_j$$

```
(%i183)   sumexpand: true$ cauchysum: true$
sum(f[i[,i,0,inf)*sum(g[j],j,0,inf);
```

(%o183) $$\sum_{i1=0}^{\infty}\sum_{i2=0}^{i1} g_{i1-i2}\, f_{i2}$$

8.4.6.1. Числовые ряды

В рамках системы Maxima можно работать со числовыми рядами через функции для степенных рядов.

8.4.6.2. Степенные ряды

Функция	*Описание*
`powerseries(expr,x,a)`	Общая форма разложения функции в степенной ряд

```
(%i184)   powerseries(-4*(sin(x))^3, x,0);
```

(%o184) $$\left(\sum_{i1=0}^{\infty}\frac{(-1)^{i1}\,3^{2i1+1}\,x^{2i1+1}}{(2\,i1+1)!}\right)-3\sum_{i1=0}^{\infty}\frac{(-1)^{i1}\,x^{2i1+1}}{(2\,i1+1)!}$$

8.4.6.3. Ряды Тейлора и Маклорена

Функция	*Описание*
`powerseries(expr,x,a)`	Общая форма разложения функции в степенной ряд
`niceindices(expr)`	Переименование индексов в суммах и произведениях в ряде для `expr`
`deftaylor(f_1(x_1),expr_1,...,f_n(x_n),expr_n)`	Постановка рядов в соответствие указанным функциям
`pade(taylor_series,numer_deg_bound,denom_deg_bound)`	Построение Паде-аппроксимации для ряда Тейлора

Maxima может работать с рядами Тейлора. Для этого используется функция `powerseries`.

```
(%i185)   powerseries(sin(x),x,0);
```

(%o185) $$\sum_{i1=0}^{\infty}\frac{(-1)^{i1}\,x^{2i1+1}}{(2\,i1+1)!}$$

```
(%i186)  niceindices(%);
```

$$(\%o186)\qquad \sum_{i=0}^{\infty} \frac{(-1)^i\, x^{2i+1}}{(2\,i+1)!}$$

Maxima автоматически использует индексы i1, i2, ... Команда niceindices заменяет их на более знакомые символы i, j, ...

8.4.6.4. Ряды Фурье. Тригонометрические ряды Фурье

Для использования функций работы с тригонометрическими рядами необходима загрузка пакета fourier.

Функция	*Описание*
fourier(f,x,p)	Коэффициенты ряда Фурье из промежутка [-p,p]
foursimp(l)	Замена sin(n %pi) на 0, если sinnpiflag=0, и cos(n %pi) на $(-1)^n$, если cosnpiflag=0
fourcos(f,x,p)	Косинус-коэффициенты ряда Фурье из промежутка [0,p]
foursin(f,x,p)	Синус-коэффициенты ряда Фурье из промежутка [0,p]
totalfourier(f,x,p)	Результат работы fourexpand(foursimp(fourier(f,x,p)),x,p,'inf)
fourintcos(f,x)	Косинус-коэффициенты ряда Фурье из промежутка [0,inf]
fourintsin(f,x)	Синус-коэффициенты ряда Фурье из промежутка [0,inf]

```
(%i187)  load("fourie")$ fourier(x^2,x,%pi);
```

$$(\%t187)\qquad a_0 = \frac{\pi^2}{3}$$

$$(\%t188)\qquad a_n = \frac{2\left(\frac{\pi^2\,\sin(\pi\,n)}{n} - \frac{2\,\sin(\pi\,n)}{n^3} + \frac{2\,\pi\,\cos(\pi\,n)}{n^2}\right)}{\pi}$$

$$(\%t189)\qquad b_n = 0$$

$$(\%o189)\qquad [\%t187,\%t188,\%t189]$$

9. ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ

В данном разделе описываются функции, доступные в системе Maxima для получения аналитических решений для некоторых конкретных типов обыкновенных дифференциальных уравнений (ОДУ). Чтобы получить численное решение системы дифференциальных уравнений см. дополнительный пакет `dynamics`. О графическом представлении решения в фазовом пространстве можно узнать в описании пакета `plotdf`.

Обыкновенное дифференциальное уравнение – это равенство, связывающее неизвестную функцию, ее производные (или дифференциалы) до некоторого порядка и независимую переменную.

Дифференциальные уравнения первого порядка включают только первые производные и могут быть записаны как

$$y' = f(x, y).$$

Дифференциальные уравнения второго порядка обязательно включают вторые производные:

$$y'' = f(x, y, y').$$

9.1. Построение полей направлений и векторных полей

Для построения векторного поля (поля направлений) для ОДУ первого порядка или системы двух автономных ОДУ первого порядка используется команда `plotdf`. Поле направлений является графическим представлением решений ОДУ.

Оператор	*Описание*
`plotdf(dydx,options)`	Построение поля направлений для одного ОДУ первого порядка
`plotdf(dvdu,[u,v],options)`	Построение поля направлений для одного ОДУ первого порядка
`plotdf([dxdt,cdydt],[x,y],options)`	Построение поля направлений для системы двух автономных ОДУ первого порядка
`plotdf([dudt,cdudt],[u,cv],options)`	Построение поля направлений для системы двух автономных ОДУ первого порядка

Опции команды `plotdf` могут быть заданы в ней самой. Каждая из опций представляет собой список из двух или более элементов. Первый элемент в каждой опции – это ее имя, второй и далее – это значение или значения, назначенные этой опции.

Команда `plotdf` требует установленной среды XMaxima, даже если эта команда запускается из консольного сеанса работы с системой Maxima, поскольку график будет создаваться скриптом `Tk`[1] в XMaxima. Если среда XMaxima не установлена, команда `plotdf` выполнена не будет.

Опция	*По умолчанию*	*Описание*
`[nsteps,N]`	100	Число шагов независимой переменной для вычисления точек интегральной кривой
`[direction,dir]`	`both`	Направления изменения независимой переменной при перемещении по интегральной кривой
`[tinitial,t0]`	0	Начальное значение переменной t для вычисления точек интегральной кривой
`[versus_t,vs0]`	0	Используется для создания второго окна графика с графиком интегральной кривой в виде двух функций x, y независимой переменной t
`[trajectory_at,x,y]`	пусто	Начальная точка интегральной кривой
`[parameters,...]`	пусто	Определение списка параметров. Имена и значения параметров должны быть заданы в строке последовательностью пар `<имя> = <значение>`, разделенных запятыми
`[sliders,...]`	пусто	Определение списка параметров, которые будут изменены в интерактивном режиме с помощью ползунков, и диапазона изменения этих параметров. Имена и диапазоны параметров должны быть указаны в строке с разделенной запятыми последовательностью элементов `<имя>= min:max`
`[xfun,...]`	пусто	Определение строки с разделенными двоеточиями функциями переменной x, которая отображается на поле направлений
`[x,x_min,x_max]`	[-10,10]	Диапазон для оси OX
`[y,y_min,y_max]`	[-10,10]	Диапазон для оси OY

[1] `Tk` – кроссплатформенная библиотека базовых элементов графического интерфейса, распространяемая с открытыми исходными текстами.

Опция	*По умолчанию*	*Описание*
`[xaxislabel,...]`	Прим[2]	Метка оси абсцисс
`[yaxislabel,...]`	Прим[3]	Метка оси ординат
`[number_of_arrows]`	225	Должно быть квадратом числа, которое определяет приблизительную плотность рисования стрелок

`dydx`, `dxdt` и `dydt` являются выражениями, которые зависят от `x` и `y`. `dvdu`, `dudt` и `dvdt` являются выражениями, которые зависят от `u` и `v`. В дополнение к этим двум переменным выражения также могут зависеть от набора параметров, причем их числовые значения задаются с помощью опции `parameters` (синтаксис задания опции приведен выше) или диапазоном допустимых значений, указанных опцией `slider`.

Ряд других опций могут быть заданы в самой команде или выбраны с помощью меню. Интегральные кривые можно получить, нажав на график или с помощью опции `trajectory_at`. Направление изменения независимой переменной при интегрировании ОДУ, которая будет использоваться для вычисления точек интегральной кривой, можно контролировать с помощью параметра `direction`, который может иметь значения `forward` (увеличить независимую переменную), `backward` (уменьшить независимую переменную) или `both` (для построения интегральной кривой, которая простирается в обоих направлениях).

Количество шагов интегрирования дается параметром `nsteps`; на каждом шаге интегрирования приращение аргумента регулируется для получения смещений, намного меньших, чем размер окна графика. Для расчетов используется численный метод Рунге–Кутты 4-го порядка с автоматическим выбором шага.

Чтобы построить поле направлений одного ОДУ первого порядка, уравнение необходимо записать в следующей форме:

$$\frac{dy}{dx} = F(x, y).$$

При этом функция или выражение, обозначенное буквой F, должно быть указано в качестве первого аргумента для `plotdf`. Имена независимой и зависимой переменных задаются в виде списка в качестве второго аргумента. Если они называются `x` и `y` соответственно, то второй аргумент может быть опущен.

[2] Имя первой переменной состояния.
[3] Имя второй переменной состояния.

При вызове `plotdf` открывается новое окно, содержащее поле направлений. Длина рисуемых векторов пропорциональна абсолютному значению первой производной.

```
(%i1)   load(plotdf)$ plotdf(2*cos(x) - y, [x, y], [x, -1.5*%pi,
1.5*%pi], [y, -4, 8], [trajectory_at, -4.5, 6], [direction,
forward])$
```

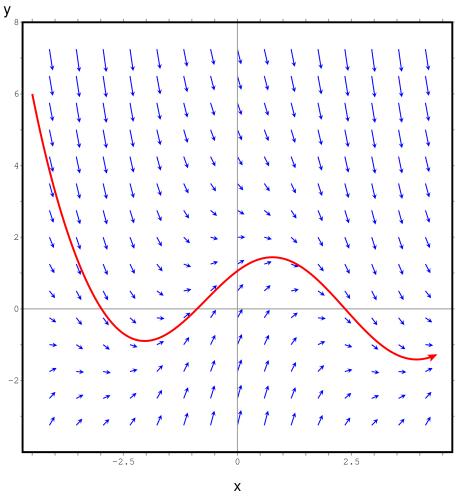


Рис. 9.1

Поле направлений и одна из траекторий для уравнения $y' = 2\cos x - y$ рисуются кодом на языке системы Maxima, показанным на рис. 9.1. Результат работы выводится в специальное окно. Напомним, что для этого сначала загружается пакет `plotdf`.

В меню окна графика есть несколько параметров (на рисунке они не показаны, так как его внешний вид сильно зависит от вычислительной среды), которые позволяют управлять графиком. Нажатие на поле направления в определенной точке добавляет траекторию, начинающуюся с этой точки. Можно нарисовать две траектории и более с разными начальными точками. С помощью опции `trajectory_at` начальные точки указываются прямо в команде `plotdf`.

Также возможно построить интегральную кривую как функцию независимой переменной `t` во втором окне графика, включив с помощью соответствующей опции в меню окна графика или с помощью опции командной строки `versus_t`.

Чтобы построить поле направления системы двух автономных ОДУ первого порядка, они должны быть записаны в следующем виде:

$$\frac{dx}{dt} = F(x,y), \qquad \frac{dy}{dt} = G(x,y).$$

При этом первый аргумент для команды `plotdf` должен быть списком с двумя функциями F и G в указанном порядке, а именно первое выражение в списке будет считаться производной по времени от переменной, изменение которой происходит вдоль горизонтальной оси, а второе выражение будет производной по времени от переменной, смещающейся вдоль вертикальной оси. Эти две переменные не обязательно должны быть x и y, но если это не так, то вторым аргументом, переданным `plotdf`, должен быть второй список с именами двух переменных: сначала для абсциссы, а затем для ординаты.

```
(%i2)   plotdf([x*(1-x-y), y*(3/4-y-x/2)], [x,y], [x,0,1.1],
[y,0,1.0], [trajectory_at,0.1,0.1], [direction,forward])$
```

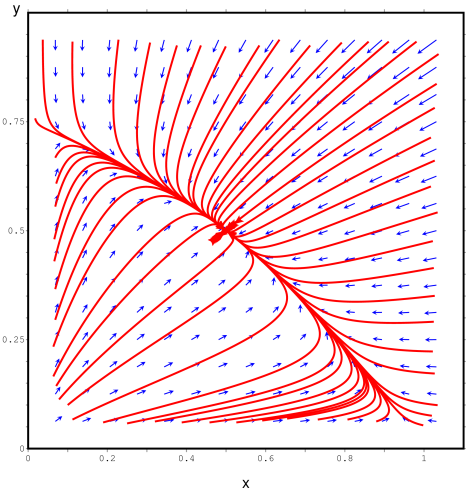


Рис. 9.2

На рис. 9.2 изображены поле направлений и траектории для системы

$$\frac{dx}{dt} = x\,(1-x-y), \qquad \frac{dy}{dt} = y\,(3/4-y-x/2).$$

```
(%i3)  load(drawdf)$colors:['red,'blue,'purple,'orange,'green,'brown]
$drawdf([y,-0.8*(x^2-1)*y-4*x],[x,-4,4],[y,-6.5,6.5],field_color=gray,
makelist([key=concat("soln",k),color=colors[k],soln_at(0,0.1+k)],
k,1,6))$
```

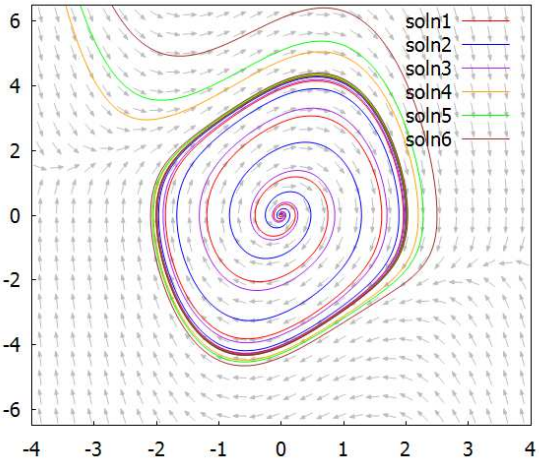


Рис. 9.3

Еще одну возможность построений полей направлений и фазовых портретов для систем ОДУ второго порядка дает функция `drawdf`, включенная в пакет `drawdf`. На рис. 9.3 изображены эти структуры для уравнения Ван дер Поля

$$\ddot{x}(t) + 0.8\left[x^2(t) - 1\right]\dot{x}(t) + 4\,x(t) = 0.$$

9.2. Аналитическое решение ОДУ при помощи встроенных функций

9.2.1. Общие решения отдельных уравнений

Функция	*Описание*
`ode2(eqn,dvar,ivar)`	Решает ОДУ первого и второго порядков

Константа	*Описание*
%c	Произвольная постоянная для ОДУ первого порядка
%k1, %k2	Произвольные постоянные для ОДУ второго порядка

Системная переменная	*По умолчанию*	*Описание*
method		Содержит имя примененного метода

Существуют два способа представления ОДУ в системе Maxima. Самый простой способ – запись уравнений первого и второго порядка через производные 'diff(y, x) и 'diff(y, x, 2) соответственно. Обыкновенное дифференциальное уравнение

$$x^2 \frac{dy}{dx} + 3\,x\,y = \frac{\sin x}{x} \tag{9.1}$$

в этом случае вводится так:

```
(%i1)   x^2*'diff(y,x) + 3*x*y = sin(x)/x;
```

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

Заметим, что перед обозначением производной diff(y,x) стоит апостроф. Он необходим, чтобы предотвратить вычисление diff(y,x) (и получение 0). Второй способ записи – указать y(x), фиксируя явно, что y – функция от x. Тогда уравнение (9.1) запишется так:

```
(%i2)   x^2*diff(y(x),x) + 3*x*y(x) = sin(x)/x;
```

$$(\%o2) \qquad x^2 \left(\frac{d}{dx} y(x)\right) + 3\,x\,y(x) = \frac{\sin(x)}{x}$$

При решении ОДУ ищется некоторая функцию $y(x)$, обращающая уравнение в верное равенство. Задача подпрограммы ode2 – решение уравнений первого и второго порядка. У подпрограммы три аргумента: ОДУ, отмеченное как eqn, имя зависимой переменной dvar и имя независимой переменная ivar. Подпрограмма пробует различные методы интегрирования, чтобы найти аналитическое решение. В случае успеха она возвращает явное или неявное решение для зависимой переменной. Символ %c используется для указания произвольной постоянной в случае уравнений первого порядка. Если подпрограмма ode2 по какой-либо причине не может получить решение, то в качестве результата она возвращает false.

```
(%i3)   ode2(x^2*'diff(y,x) + 3*x*y = sin(x)/x, y, x);
```

$$(\%o3) \qquad y = \frac{\%c - \cos(x)}{x^3}$$

Кроме того, подпрограмма `ode2` в системной переменной `method` сохраняет информацию о методе, использованном для решения соответствующего ОДУ. Указанное выше дифференциальное уравнение было признано линейным ОДУ:

```
(%i4)  method;
(%o4)      linear
```

В случае, когда указывается $y(x)$, результат будет выглядеть так:

```
(%i5)  ode2(x^2*diff(y(x),x) + 3*x*y(x) = sin(x)/x, y(x), x);
(%o5)      y(x) = (%c - cos(x))/x^3
```

Подпрограмму `ode2` можно использовать и для решения ОДУ второго порядка. В этом случае две константы интегрирования обозначаются `%k1` и `%k2`.

```
(%i6)  sol2: ode2('diff(y,x,2) + y*'diff(y,x)^3 = 0, y, x);
(%o6)      (y^3 + %k1 y)/6 = x + %k2
```

Оператор	*Описание*
`desolve(eqn,y)`	Решает линейное ОДУ второго порядка и выше
`atvalue(expr,[x_1=a_1,...],c)`	Назначает значение `c` выражению `expr` в точке `x=a`

Одно линейное ОДУ может быть решено путем использования подпрограммы `desolve`. Необходимо, чтобы функция $y(x)$ были задана в функциональной форме. Решение линейного дифференциального уравнения второго порядка

$$y''(x) + y(x) = 2\,x$$

в таком случае может быть выполнено так:

```
(%i7)  desolve(diff(y(x),x,2) + y(x) = 2*x, y(x));
(%o7)      y(x) = sin(x) (d/dx y(x) - 2) + y(0) cos(x) + 2x
```

Если подпрограмма `desolve` не может найти решение для данного(ых) ОДУ, то результатом будет `false`.

Если начальные условия при $x = 0$ известны, то они могут быть учтены с использованием оператора `atvalue`. Заметим, что начальные условия могут быть заданы только при $x = 0$ и должны быть определены до решения уравнений. Рекомендуется удалять такие (глобальные) назначения сразу после решения задачи Коши. В противном случае это может привести к неясным сообщениям об ошибках или (что еще хуже) неверным результатам далее в сеансе.

```
(%i8)   atvalue(y(x), x = 0, 1)$ atvalue(diff(y(x),x), x = 0, 2)$
desolve(diff(y(x), x, 2) + y(x) = 2*x, y(x));
(%o8)        y(x) = cos(x) + 2 x
```

```
(%i9)   kill(y)$
```

Дополнительные возможности для решения линейных и нелинейных ОДУ дают функции из пакета `contrib_ode`, который доступен, несмотря на незавершенность разработки.

9.2.2. Общие решения систем линейных уравнений

Оператор	*Описание*
`desolve([eqn_1,...,eqn_n],[y_1,..,y_n])`	Решает линейную систему ОДУ

Системы линейных ОДУ могут быть решены путем использования уже упоминавшейся функции `desolve`. Необходимо, чтобы функции $y_i(x)$ были заданы в функциональной форме.

Например, пусть уравнения выглядят так:

$$x'(t) = x(t) - 2\,y(t),$$
$$y'(t) = -2\,x(t) + y(t).$$

Тогда

```
(%i10)   eqn1: diff(x(t),t) = x(t) - 2*y(t);
(%o10)        d/dt x(t) = x(t) - 2 y(t)
```

```
(%i11)   eqn2: diff(y(t),t) = -x(t) + y(t);
(%o11)        d/dt x(t) = y(t) - x(t)
```

```
(%i12)   desolve([eqn1, eqn2], [x(t), y(t)]);
(%o12)
```

$$\left[x(t) = \%e^{t}\left(\frac{\left(2\left(-2\,y(0) - x(0)\right) + 2\,x(0)\right)\,\sinh(\sqrt{2}\,t)}{2^{3/2}} + x(0)\,\cosh(\sqrt{2}\,t)\right),\right.$$
$$\left.y(t) = \%e^{t}\left(\frac{\left(2\,y(0) + 2\left(-2\,y(0) - x(0)\right)\right)\,\sinh(\sqrt{2}\,t)}{2^{3/2}} + y(0)\,\cosh(\sqrt{2}\,t)\right)\right]$$

Для нахождения решения данной системы ОДУ функция `desolve` использовала так называемое преобразование Лапласа.

9.2.3. Решение задач Коши

Оператор	*Описание*
`ic1(sol,xval,yval)`	Решение задачи Коши для ОДУ первого порядка
`ic2(sol,xval,yval,dval)`	Решение задачи Коши для ОДУ второго порядка

Задача Коши для уравнения первого порядка может быть решена с помощью процедуры `ic1`. Она использует общее решение уравнения, полученное подпрограммой `ode2` и передаваемое на месте первого аргумента. Второй аргумент `xval` указывает начальное значение для независимой переменной в форме `x = x0`, а третий аргумент `yval` задает соответствующее начальное значение для зависимой переменной в форме `y = y0`.

Итак запрос на получение решения задачи Коши

$$x^2\,\frac{dy}{dx} + 3\,x\,y = \frac{\sin x}{x}, \qquad y(\pi) = 0$$

выглядит так:

```
(%i13)   sol1: ode2(x^2*'diff(y,x) + 3*x*y = sin(x)/x, y, x);
(%o13)
```

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

```
(%i14)   ic1(sol1, x=%pi, y=0);
(%o14)
```

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

Подпрограмму `ode2` можно использовать и для решения ОДУ второго порядка. В этом случае две константы интегрирования обозначаются `%k1` и `%k2`.

Задача Коши для ОДУ второго порядка может быть решена с помощью `ic2`. По сравнению с `ic1`, `ic2` имеет дополнительный четвертый аргумент `dval`, который задает начальное значение для первой производной зависимой переменной в виде `diff(y, x) = dy0`.

```
(%i15)  sol2: ode2('diff(y,x,2) + y*'diff(y,x)^3 = 0, y, x);
```

$$\frac{y^3 + \%k1\, y}{6} = x + \%k2 \quad \text{(\%o15)}$$

```
(%i16)  ic2(sol2, x=0, y=0, 'diff(y,x) = 2);
```

$$\frac{y^3 + 3\, y}{6} = x \quad \text{(\%o16)}$$

Задача с начальными условиями $x(0) = 1$ и $y(0) = 1$ для системы ОДУ:

$$x'(t) = x(t) - 2\, y(t),$$
$$y'(t) = -2\, x(t) + y(t),$$

решается так:

```
(%i17)  eqn1: diff(x(t),t) = x(t) - 2*y(t);
```

$$\frac{d}{dt}\, x(t) = x(t) - 2\, y(t) \quad \text{(\%o17)}$$

```
(%i18)  eqn2: diff(y(t),t) = -x(t) + y(t);
```

$$\frac{d}{dt}\, x(t) = y(t) - x(t) \quad \text{(\%o18)}$$

```
(%i19)  atvalue(x(t), t = 0, 1)$ atvalue(y(t), t = 0, 1)$
desolve([eqn1, eqn2], [x(t), y(t)]);
(%o19)
```

$$\left[x(t) = \%e^t \left(\cosh(\sqrt{2}\, t) - \sqrt{2}\, \sinh(\sqrt{2}\, t)\right),\right.$$
$$\left. y(t) = \%e^t \left(\cosh(\sqrt{2}\, t) - \frac{\sinh(\sqrt{2}\, t)}{\sqrt{2}}\right)\right]$$

9.2.4. Решение краевых задач

Оператор	*Описание*
`bc2(sol,xval1,yval1,xval2,yval2)`	Решение краевой задачи для ОДУ второго порядка

Параметры: `sol` – общее решение уравнения, найденное с помощью `ode2`; `xval1` – значение независимой переменной в первой точке в форме `x = x1`, `yval1` – значение зависимой переменной в этой точке в форме `y = y1`. Выражения `xval2` и `yval2` дают значения для этих переменных во второй точке в той же форме.

```
(%i20)   sol: ode2('diff(y,x,2) + y*'diff(y,x)^3 = 0,y,x);
(sol)      y^3 + 6 %k1 y
           ------------- = x + %k2
                 6
```

```
(%i21)   bc2(sol,x=0,y=1,x=1,y=3);
(%o21)     y^3 - 10 y
           ---------- = x - 3/2
                6
```

9.3. Разностные уравнения

Для решения линейных обыкновенных разностных уравнений (ЛОРУ) с полиномиальными коэффициентами предназначен пакет `solve_rec`.

Оператор	*Описание*
`solve_rec(eqn,var)`	Получение общего решения в терминах гипергеометрических функций
`solve_rec(eqn,var,init)`	Получение частного решения в терминах гипергеометрических функций
`simplify_sum(expr)`	Упрощение сумм, имеющихся в `expr` в замкнутой форме
`reduce_order(rec,sol,var)`	Уменьшает порядок ЛОРУ
`solve_rec_rat(eqn,var)`	Получение рационального общего решения ЛОРУ
`solve_rec_rat(eqn,var,init)`	Получение рационального частного решения ЛОРУ

Функция `solve_rec` может решать ЛОРУ с постоянными коэффициентами, находить гипергеометрические решения однородных ЛОРУ с полиномиальными коэффициентами, дробно-рациональные решения ЛОРУ с полиномиальными коэффициентами и может решать ЛОРУ типа Рикатти.

Заметим, что время работы алгоритма, используемого для поиска гипергеометрических решений, экспоненциально по отношению к степени ведущего и конечного коэффициентов.

```
(%i1)   load(solve_rec)$
```

```
(%i2)   solve_rec(a[n]=2*a[n-1]+a[n-2]+1/2^n, a[n]);
```

(%o2) $$a_n = -\frac{1}{7\,2^n} + \left(\sqrt{2}+1\right)^n \%k_2 + \left(1-\sqrt{2}\right)^n \%k_1$$

```
(%i3)   solve_rec(2*x*(x+1)*y[x] - (x^2+3*x-2)*y[x+1] +
(x-1)*y[x+2], y[x], y[1]=1, y[3]=3);
```

(%o3) $$y_x = 3\,2^{x-2} - \frac{x!}{2}$$

Для использования функции `simplify_sum` нужно загрузить пакет `simplify_sum`.

```
(%i4)   load("simplify_sum")$ sum(binomial(2*n,2*k),k,1,n);
```

(%o4) $$\sum_{k=1}^{n} \binom{2\,n}{2\,k}$$

```
(%i5)   simplify_sum(%);
```

(%o5) $$2^{2n-1} - 1$$

Для использования функции `solve_rec_rat` нужно загрузить пакет `solve_rec`.

```
(%i6)   solve_rec_rat((x+4)*a[x+3]+(x+3)*a[x+2]-x*a[x+1]+(x^2-1)*a[x]
= (x+2)/(x+1), a[x]);
```

(%o6) $$a_x = \frac{1}{(x-1)\,(x+1)}$$

Функцию `eliminate` можно применять для исключения переменных в системах ЛОРУ.

Функция	*Описание*
`eliminate([eqn_1,...,eqn_n], [x_1,...,x_k])`	Исключает переменные из уравнений (или выражений, принимаемых равными нулю)

```
(%i7)   eq1: y[t+1]=z[t]$ eq2: z[t]=c[t]+i[t]$ eq3: c[t]=c*y[t]$
eq4: i[t]=a$
```

```
(%i8)   eliminate([eq1,eq2,eq3,eq4], [z[t],c[t],i[t]]);
```

(%o8) $$[y_{t+1} - c\,y_t - a]$$

```
(%i9)   solve_rec(first(%), y[t], y[0]=y[0]);
```

(%o9) $$y_t = \frac{a\,c^t}{c-1} + y_0\,c^t - \frac{a}{c-1}$$

9.4. Интегральные уравнения

Для решения *обыкновенных интегральных уравнений* (ОИДУ) предназначен пакет `inteqn`.

Оператор	*Описание*
`ieqn(ie,unk,tech,n,guess)`	Получение решения ОИДУ

Параметры: `ie` – интегральное уравнение; `unk` – неизвестная функция; `tech` – это метод, который нужно опробовать из имеющихся (`tech = first` означает, что необходимо использовать первый метод, который находит решение; `tech = all` – применение всех возможных методов); `n` – максимальное число членов, которое нужно взять для рядов Тейлора, Неймана, Фредгольма и др. (этот параметр – также максимальная глубина рекурсии для метода дифференцирования); `guess` – начальное приближение.

Возвращаемое функцией значение представляет собой список из четырех элементов: решение, использовавшийся метод, число слагаемых, флаг. Форма вывода определяется системной переменной `ieqnprint`.

Функция `ieqn` может быть применена для решения двух форм линейных и нелинейных интегральных уравнений – первого и второго рода, которые имеют вид:

$$x(t) = F\Big(t, x(t), \int\limits_{a(t)}^{b(t)} K\big(t, s, x(t), x(s)\big)\, ds\Big),$$

$$\int\limits_{a(t)}^{b(t)} K\big(t, s, p(s)\big)\, ds = f(t),$$

где $a(t)$, $b(t)$, $f(t)$, $K(...)$, $F(...)$ – известные функции, $x(t)$ – неизвестная функция. Это общие формы ОИДУ, но большинство методов требуют частных форм функций F и K.

Используемые *методы* [35][4]:

[4] D.R. Stoutemyer первоначально написал функцию на языке Reduce, а потом она была включена в систему Macsyma, из которой перешла в Maxima.

Метод	*Признаки*	*Форма уравнения*
`flfrnk2nd`	Вырожденное ядро Конечные постоянные пределы	(9.2)
`vlfrnk`	То же самое Конечные переменные пределы	(9.3)
`vlfrnk`	То же самое Конечные переменные пределы	(9.4)
`transform` (преобразование Лапласа)	Разностное ядро Переменный верхний предел	(9.5)
`transform` (преобразование Фурье)	Разностное ядро Бесконечные пределы	(9.6)
Дифференцирование	Конечные переменные пределы	(9.7)
`fredseries` (ряды Фредгольма–Карлемана)	Конечные переменные пределы	(9.8)
`taylor` (ряды Тейлора)	Конечные переменные пределы	(9.9)
`neumann` (ряды Неймана)	Конечные переменные пределы	(9.10)
`firstkindseries` (ряды первого рода)	Конечные переменные пределы	(9.11)
`collocate` (коллокация)	Любая из форм	
`transform` (преобразование Лапласа)	Разностное ядро Переменный верхний предел Интеграл конволюции	(9.12) (сингулярное уравнение Абеля)

Формы уравнений:

$$x(t) = F\Big(t, \int_a^b \sum_{j=1}^{n} q_j(t)\, r_j\big(s, x(s)\big)\, ds\Big); \tag{9.2}$$

$$x(t) = f(t) + \int_{a(t)}^{b(t)^*} q(t)\, K(s)\, x(s)\, ds; \tag{9.3}$$

$$\int_{a(t)}^{b(t)^*} q(t)\, r_j\big(s, x(s)\big)\, ds = f(t); \tag{9.4}$$

$$\alpha\, x(t) = f(t) + \int_0^t K(t-s)\, x(s)\, ds, \quad \alpha = \text{const}; \tag{9.5}$$

$$\alpha\, x(t) = f(t) + \int_{-\infty}^{+\infty} K(t-s)\, x(s)\, ds, \quad \alpha = \text{const}; \tag{9.6}$$

$$\int_{a(t)}^{b(t)^*} K\big(t,s,x(s)\big)\,ds = f(t); \tag{9.7}$$

$$x(t) = f(t) + \int_{a(t)}^{b(t)} K(t,s)\,x(s)\,ds; \tag{9.8}$$

$$x(t) = f(t) + \int_{a(t)}^{b(t)^*} K\big(t,s,x(s)\big)\,ds; \tag{9.9}$$

$$x(t) = \Phi\Big[t, x[\beta(t)]^{**}, \int_{a(t)}^{b(t)} K\big(t,s,x[\gamma(t,s)]^{**}\big)\,ds\Big]; \tag{9.10}$$

$$\int_{a(t)}^{b(t)} K\big(t,s,x[\gamma(t,s)]^{**}\big)\,ds; \tag{9.11}$$

$$f(t) = \int_0^t \frac{x(s)}{\sqrt{t-s}}\,ds. \tag{9.12}$$

Примечание. (*) Если $a'(t) = 0$, то $b(t) = t$, и наоборот. (**) Возможно любое число таких аргументов.

Системная	*По умолчанию*	*Описание*
переменная		
`ieqnprint`	`true`	Управление формой вывода результата функцией `ieqn`

Когда переменная `ieqnprint` имеет значение `false`, список, возвращаемый функцией `ieqn`, имеет вид:

`[<решение>, <метод>, nterms, flag]`,

где `flag` отсутствует, если решение точное. В противном случае `flag = approximate` или `flag = incomplete`, что соответствует неточной или незамкнутой форме решения соответственно. Если был использован метод разложения в ряд, то `nterms` дает количество полученных членов (которое может быть меньше, чем `n`, заданное при вызове `ieqn`, если ошибка помешала расчету следующих слагаемых).

```
(%i1)   load(inteqn)$
```

```
(%i2)   ieqn(x(t)-t-'integrate(s*x(s),s,0,1),x(t),first,1,none);
(%t2)       [t + 2/3, ffrnk2nd]
(%o2)       [%t2]
```

10. ЧИСЛЕННЫЕ МЕТОДЫ

10.1. Разложение и другие операции с матрицами

Функция	*Описание*
`cholesky(A)`	Разложение Холецкого матрицы `A`
`lu_factor(A)`	LU-разложение квадратной матрицы `A`
`get_lu_factors(x)`	Получение списка в форме `[P, L, U]`

```
(%i1)   A: matrix([1,%i,1],[-%i,4,-%i],[1,%i,9]);
```

$$\text{(\%o1)} \qquad \begin{pmatrix} 1 & \%i & 1 \\ -\%i & 4 & -\%i \\ 1 & \%i & 9 \end{pmatrix}$$

```
(%i2)   B: cholesky(A);
```

$$\text{(\%o2)} \qquad \begin{pmatrix} 1 & 0 & 0 \\ -\%i & \sqrt{3} & 0 \\ 1 & 0 & 2^{3/2} \end{pmatrix}$$

```
(%i3)   B.ctranspose(B);
```

$$\text{(\%o3)} \qquad \begin{pmatrix} 1 & \%i & 1 \\ -\%i & 4 & -\%i \\ 1 & \%i & 9 \end{pmatrix}$$

Функция `get_lu_factors(L)` в случае, когда `L=lu_factor(A)`, генерирует список в форме `[P, L, U]`, где `P` – матрица перестановок, `L` – нижняя треугольная матрица с единицами на главной диагонали, `U` – верхняя треугольная матрица, причем `A = P L U`.

```
(%i4)   A: matrix([1,2,3],[-2,4,6],[-3,6,19])$ z: lu_factor(A)$ zz:
get_lu_factors(z);
```

$$\text{(zz)} \qquad \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 3/2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 0 & 8 & 12 \\ 0 & 0 & 10 \end{pmatrix} \right]$$

```
(%i5)   zz[2].zz[1].zz[3]-A;
```

$$\text{(\%o5)} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Функция	*Описание*
`invert_by_lu(A)`	Матрица, обратная к матрице `A` и полученная на основе метода LU-разложения
`eigens_by_jacobi(A)`	Вычисление собственных значений и собственных векторов симметрической матрицы `A` по методу вращения (методу Якоби)
`uniteigenvectors(A)`	Нормированные собственные векторы матрицы `A`
`jordan(A)`	Жорданова форма матрицы `A`
`JF(lambda,n)`	Жорданова клетка `n`-го порядка с собственным значением `lambda`
`dispJordan(I)`	Матрица жордановой канонической формы
`minimalPoly(I)`	Минимальный многочлен матрицы жордановой формы
`ModeMatrix(A,[jordan_info])`	Обратимая матрица `B`, такая, что `(B^(-1)).A.B` – жорданова форма матрицы `A`

Результатом выполнения функции `eigens_by_jacobi(A)` является список из двух элементов, причем первый из них – собственные значения матрицы `A`, а второй – соответствующие собственные векторы.

Результатом выполнения функции `euniteigenvectors(A)` является список из двух элементов, причем первый из них – список `[[s1, s2, ..., sn], [k1, k2, ..., kn]]` собственных значений `s1`, `s2`, ..., `sn` с указанием их алгебраических кратностей `k1`, `k2`, ..., `kn`, а второй – список соответствующих нормированных собственных векторов матрицы `A`.

Функция `jordan(A)` возвращает жорданову форму матрицы `A` в виде определенного списка. Каждый элемент списка является списком из двух элементов, причем первый из них – собственное значение матрицы `A`, второй – натуральное число, являющееся длиной жорданова блока для данного собственного значения (эти числа перечислены в порядке убывания).

Функция `dispJordan(I)` возвращает матрицу жордановой канонической формы, соответствующую списку `I`, который должен быть получен командой `jordan(A)`.

Функция `minimalPoly(I)` возвращает минимальный многочлен матрицы жордановой формы, соответствующей списку `I`.

Дополнительные функции см. в подразделе 10.3.

10.2. Решение СЛАУ

Функция	*Описание*
`lu_backsub(M,b)`	Решения СЛАУ `A x = b`, где `M=lu_factor(A)`

Матрица b размерности n x m, где n – число строк матрицы A, состоит из столбцов, представляющих правые части СЛАУ. Если есть только одна правая часть, то b должно быть матрицей n x 1.

Каждый столбец матрицы z = lu_backsub(M,b) является решением, соответствующим столбцу b с тем же номером.

10.3. Пакет lapack

Данный пакет является результатом конвертации подпрограмм из библиотеки LAPACK, написанных на языке программирования Fortran, в среду языка Common Lisp. Для использования требуется загрузка пакета с помощью команды load.

Функция	*Описание*
dgeev(A)	Вычисление собственных чисел матрицы A
dgeev(A,right_p,left_p)	Вычисление собственных чисел и собственных векторов матрицы A

Все элементы матрицы nA должны быть целыми числами или числами с плавающей точкой. Матрица A должна быть квадратной, может быть, а может и не быть симметричной.

Команда dgeev(A,right_p,left_p) вычисляет, кроме собственных значений матрицы A, правые собственные векторы, если right_p = true, и левые собственные векторы, если left_p = true. Результатом работы команды будет набор из трех элементов, а именно: 1) список собственных значений; 2) false или матрица правых собственных векторов; 3) false или матрица левых собственных векторов.

Вычисленные собственные векторы нормируются, а наибольшая компонента имеет мнимую часть, равную нулю.

```
(%i1)   load("lapack");
(%o1)     "C:/maxima-5.43.0/share/maxima/5.43.0/share/lapack/lapack.mac"
```

```
(%i2)   fpprintprec : 6;
```

```
(%i3)   M : matrix ([9.5, 1.75], [3.25, 10.45])$
```

```
(%i4)   dgeev (M);
(%o4)       [[7.543308202094681, 12.40669179790532], false, false]
```

```
(%i5)   [L, v, u] : dgeev (M, true, true);
```

$$(\%o5)\quad [[7.543308202094681,\ 12.40669179790532],\ \begin{pmatrix} -0.6666416199396576 & -0.5157922155034693 \\ 0.7453783942161386 & -0.8567137155584839 \end{pmatrix},\ \begin{pmatrix} -0.8567137155584839 & -0.7453783942161386 \\ 0.5157922155034693 & -0.6666416199396576 \end{pmatrix}]$$

Функция	*Описание*
`dgeqrf(A)`	Вычисление QR-декомпозиции матрицы `A`

Все элементы матрицы `A` должны быть целыми числами или числами с плавающей точкой. Матрица `A` может быть, а может и не быть квадратной.

Результатом работы команды будет набор из двух элементов, а именно: 1) квадратная ортонормированная матрица `Q`, которая имеет то же число столбцов, что и матрица `A`; 2) матрица `R`, имеющая ту же размерность, что и `A`, и нулевые элементы ниже главной диагонали. При этом $A = Q\,R$ с точностью по погрешности округления.

```
(%i6)   M : matrix([1, -3.2, 8], [-11, 2.7, 5.9])$
```

```
(%i7)   [q, r] : dgeqrf (M);
```

$$(\%o7)\quad \left[\begin{pmatrix} -0.09053574604251846 & 0.9958932064677037 \\ 0.9958932064677037 & 0.09053574604251868 \end{pmatrix},\ \begin{pmatrix} -11.04536101718726 & 2.978626044798859 & 5.151483949819305 \\ 0.0 & -2.942411746381851 & 8.50130655339249 \end{pmatrix}\right]$$

```
(%i8)   q.r - M;
```

$$(\%o8)\quad \begin{pmatrix} -7.7715611723761\ 10^{-16} & 1.77635683940025\ 10^{-15} & -8.88178419700125\ 10^{-16} \\ 0.0 & -1.33226762955019\ 10^{-15} & 8.881784197001252\ 10^{-16} \end{pmatrix}$$

Функция	*Описание*
`dgesvd(A)`	Сингулярное разложение матрицы `A`
`dgesvd(A,left_p,right_p)`	Сингулярное разложение и вычисление сингулярных векторов матрицы `A`

Команда вычисляет сингулярное разложение матрицы `A` (the singular value decomposition, SVD): $A = U\,S\,V^{\intercal}$, где U – $(m \times m)$–унитарная матри-

ца[1], $V^\intercal$ – $(n \times n)$–унитарная матрица, S – $(m \times m)$–диагональная матрица. Вторая форма вызова команды (см. команду `dgeev` в этом подразделе) дополнительно вычисляет левый и правый сингулярные векторы. Элементы `A` должны быть целыми числами или числами с плавающей точкой. Матрица `A` может быть квадратной или прямоугольной.

Результатом выполнения команды является набор из трех элементов: 1) список сингулярных чисел, образующих диагональ матрицы `A`; 2) `false` или матрица левых сингулярных векторов; 3) `false` или матрица правых сингулярных векторов.

```
(%i9)   M: matrix([1, 2, 3], [3.5, 0.5, 8], [-1, 2, -3], [4, 9,
7])$
```

```
(%i10)   fpprintprec : 8;
```

```
(%i11)   dgesvd(M);
(%o11)       [[14.4744434, 6.386367492, 0.4525465373], false, false]
```

```
(%i12)   [sigma,U,VT] : dgesvd(M,true,true);
(%o12)       [ [14.4744434, 6.386367492, 0.4525465373],
```

$$\begin{pmatrix} -0.2567308294 & 0.008161684591 & 0.9590291726 & -0.1195228609 \\ -0.52645588 & 0.6721158201 & -0.2062355503 & -0.4780914437 \\ 0.1079971762 & -0.5322776558 & -0.07083153984 & -0.8366600265 \\ -0.803287058 & -0.5146593129 & -0.1808669038 & 0.2390457219 \end{pmatrix},$$

$$\begin{pmatrix} -0.3744856825 & -0.5382092113 & -0.7550439182 \\ 0.1306231531 & -0.8367993003 & 0.5316996548 \\ -0.9179858743 & 0.1004876908 & 0.3836719413 \end{pmatrix}]$$

Функция	*Описание*
`dgemm(A,B)`	Произведение двух матриц `A` и `B`
`dgemm(A,B,options)`	Вычисление выражения $\alpha \cdot A \cdot B + \beta \cdot C$

При расширенном вызове A, B, C – матрицы, α, β – числовые константы. Кроме того, A и/или B могут быть транспонированы перед дальнейшими расчетами. Дополнительные параметры специфицируются ключевыми словами, которые могут записываться в любом порядке в форме

[1] *Унитарная матрица* – квадратная матрица с комплексными элементами, результат умножения которой на эрмитово-сопряженную равен единичной матрице. *Эрмитово-сопряженная матрица* – это матрица с комплексными элементами, полученная из исходной матрицы транспонированием и заменой каждого элемента комплексно-сопряженным ему.

"<ключевое слово> = <значение>". Перечень ключевых слов: "c" – добавление третьей матрицы (по умолчанию false); "alpha" – α; (по умолчанию 1); "beta" – β, если C задана ($\beta \neq 0$); "transpose_a" – умножение на транспонированную матрицу "A", если true (по умолчанию false); "transpose_b" – умножение на транспонированную матрицу "B", если true (по умолчанию false).

```
(%i13)   A : matrix([1,2,3],[4,5,6],[7,8,9])$
```

```
(%i14)   B : matrix([-1,-2,-3],[-4,-5,-6],[-7,-8,-9])$
```

```
(%i15)   C : matrix([3,2,1],[6,5,4],[9,8,7])$
```

```
(%i16)   dgemm(A,B);
(%o16)
```

$$\begin{pmatrix} -30.0 & -36.0 & -42.0 \\ -66.0 & -81.0 & -96.0 \\ -102.0 & -126.0 & -150.0 \end{pmatrix}$$

```
(%i17)   dgemm(A,B,c=C,beta=1,alpha=-1);
(%o17)
```

$$\begin{pmatrix} 33.0 & 38.0 & 43.0 \\ 72.0 & 86.0 & 100.0 \\ 111.0 & 134.0 & 157.0 \end{pmatrix}$$

```
(%i18)   -A.B + C;
(%o18)
```

$$\begin{pmatrix} 33 & 38 & 43 \\ 72 & 86 & 100 \\ 111 & 134 & 157 \end{pmatrix}$$

Функция	*Описание*
zgeev(A)	Вычисление собственных чисел комплексной матрицы A
zgeev(A,right_p,left_p)	Вычисление собственных чисел и собственных векторов комплексной матрицы A
zheev(A)	То же, что и zgeev, но для квадратной комплексной эрмитовой матрицы
zheev(A,eigvec_p)	То же, что и zheev(A), но если eigvec_p = true, то дополнительно вычисляются собственные векторы

10.4. Решение нелинейных алгебраических и трансцендентных уравнений

Функция	*Описание*
`realroots(expr,bound)`	Решение отдельных полиномиальных уравнений
`realroots(eqn,bound)`	
`realroots(expr)`	
`realroots(eqn)`	
`nroots(p,low,high)`	Количество действительных корней уравнения в интервале `(low,high)`
`horner(expr,x)`	Разложение по схеме Горнера
`horner(expr)`	
`find_root(expr,x,a,b,`	Поиск корня уравнения `lhs(expr)=rhs(expr)`
`[abserr,relerr])`	на отрезке `[a,b]`
`find_root(f,a,b,`	Поиск корня непрерывной функции `f` на отрезке `[a,b]`
`[abserr,relerr])`	
`bf_find_root(expr,x,a,b,`	Версия `find_root` для `bigfloat`
`[abserr,relerr])`	
`bf_find_root(f,a,b,`	
`[abserr,relerr])`	
`newton(expr,x,x_0,eps)`	Метод Ньютона

Функция `realroots` находит все корни выражения `expr=0` или уравнения `eqn`. Для изоляции каждого корня эта функция строит последовательность Штурма и использует алгоритм деления пополам для уточнения корня с точностью `bound` или с точностью по умолчанию, определяемой системной переменной `rootsepsilon` (по умолчанию 10^{-7}). Когда `bound` меньше 1, все целочисленные корни находятся точно.

```
(%i1)  realroots(-1-x^2-x^4+x^5, 5e-6);
(%o1)      [x = 823209/524288]
```

```
(%i2)  realroots(expand((2+x)^5*(2-x)^3*(4-x)^4), 1e-20);
(%o2)      [x = -2, x = 4, x = 2]
```

```
(%i3)  multiplicities;
(%o3)      [5,4,3]
```

```
(%i4)  fpprec:32$ bf_find_root(exp(-x)=sin(x),x,0,1);
(%o4)      5.8853274398186107743245204570296b - 1
```

При использования метода Ньютона необходимо загрузить пакет `newton1`.

```
(%i5)   load("newton1")$ newton(exp(-x)-sin(x),x,1,1/10000000000000000);
(%o5)       0.5885327439818611
```

Функция	*Описание*
`minpack_solve(flist, varlist,guess[, tolerance,jacobian])`	Решение системы n уравнений с n неизвестными

Левые части f_i уравнений системы

$$f_i(x_1, x_2, ..., x_n) = 0, \qquad i = 1, 2, ..., n,$$

представляются в списке `flist`, а имена неизвестных x_j – в списке `varlist`. Начальное приближение для решения уравнений задается в списке `guess`.

Необязательные ключевые параметры `tolerance` и `jacobian` обеспечивают некоторый контроль за работой алгоритма. `tolerance` – это оцениваемая относительная погрешность вычисления суммы квадратов функций в точке, соответствующей полученному решению. `jacobian` может быть использован для указания вычислять или нет якобиан в символьном виде. Если параметр `jacobian` не задан или имеет значение `true` (по умолчанию), то якобиан системы фуннкций $\{f_i\}$ вычисляется по символьным представлениям элементов списка `flist`. Если же `jacobian` = `false`, то используется численная аппроксимация якобиана.

Функция `minpack_solve` возвращает список, первый элемент которого – приближенное решение задачи, второй – сумма квадратов значений правых частей системы в точке решения, а третий – код окончания `info`. Возможные значения последнего:

0 – неправильные входные параметры;

1 – относительная ошибка у решения не более, чем `tolerance`;

2 – количество вычислений правых частей f_i с $iflag = 1$ достигло $100\,(n + 1)$;

3 – текущая относительная погрешность слишком мала. Дальнейшее уменьшение величины суммы квадратов невозможно;

4 – дальнейшие итерации не дадут улучшения приближенного решения.

Для вызова функции требуется предварительная загрузка пакета `minpack`, который является результатом переноса библиотеки MINPACK, написанной на языке Fortran, в среду Common Lisp.

```
(%i6)   load("minpack")$
```

```
(%i7)   minpack_solve([sin(2*x-y)-1.2*x-0.4, 0.8*x^2+1.5*y^2-1],
[x,y], [1,-1]);
(%o7)
```

$$[[0.4912379505051516, -0.7334613013135369],$$
$$2.333899001788871\ 10^{-14}, 1]$$

```
(%i8)   minpack_solve([sin(2*x-y)-1.2*x-0.4, 0.8*x^2+1.5*y^2-1],
[x,y], [-1,-1]);
(%o8)
```

$$[[-1.090593921093577, -0.1797849074477408],$$
$$9.485749680535095\ 10^{-16}, 1]$$

Здесь продемонстрировано получение двух различный решений системы двух уравнений при задании различных начальных условий.

Дополнительные возможности для решения систем и отдельных нелинейных уравнений с одной или несколькими неизвестными дает функция, включенная в пакет `mnewton` и представляющая собой реализацию метода Ньютона.

Функция	*Описание*
`mnewton(FuncList, VarList, GuessList)`	Решение системы нелинейных уравнений методом Ньютона

Параметрами функции являются: `FuncList` – список левых частей уравнений, `VarList` – список имен неизвестных, `GuessList` – вектор начальных приближений. Решение возвращается в том же формате, что и метод solve(). Если решение не найдено, возвращается пустой список `[]`. Функция `mnewton` управляется глобальными переменными `newtonepsilon` и `newtonmaxiter`.

Системная переменная	*По умолчанию*	*Описание*
`newtonepsilon`	10.0^(-fpprec/2)	Погрешность для оценки приближения алгоритма к решению
`newtonmaxiter`	50	Максимальное количество итераций до остановки работы функции `mnewton`, если вычислительный процесс не сходится или если сходится слишком медленно

Если параметр `newtonepsilon` имеет тип `bigfloat`, то и все вычисления в рамках использования функции `mnewton` будут выполняться с числами типа `bigfloat`.

```
(%i9)   load("mnewton")$
```

```
(%i10)  mnewton([x1+3*log(x1)-x2^2, 2*x1^2-x1*x2-5*x1+1],[x1, x2],
[5, 5]);
```

(%o10) $[[x1 = 3.756834008012769,\ x2 = 2.779849592817897]]$

```
(%i11)  mnewton([2*a^a-5],[a],[1]);
```

(%o11) $[[a = 1.70927556786144]]$

```
(%i12)  mnewton([2*3^u-v/u-5, u+2^v-4], [u, v], [2, 2]);
```

(%o12) $[[u = 1.066618389595407,\ v = 1.552564766841786]]$

```
(%i13)  (fpprec: 25, newtonepsilon: bfloat(10^(-fpprec+5)))$
```

```
(%i14)  mnewton([2*3^u-v/u-5, u+2^v-4], [u, v], [2, 2]);
```

(%o14) $[[u = 1.06661838959540677259117\mathit{3b}0,\ v = 1.55256476684178645010041\mathit{8b}0]]$

10.5. Аппроксимация функций

Пакет `interpol` дает возможность для приближения таблично заданных функций использовать интерполяционные многочлены Лагранжа, линейную интерполяцию, кубические сплайны и Паде-аппроксимацию.

Функция	*Описание*
`lagrange(points)`	Интерполяционный многочлен Лагранжа
`linearinterpol(points)`	Линейная интерполяция
`cspline(points)`	Кубический сплайн
`ratinterpol(points,numdeg)`	Паде-аппроксимация
`charfun2(x,a,b)`	Предикат: `true`, если `x` принадлежит полуинтервалу `[a,b)`, иначе `false`

Аргумент `points` функций `lagrange`, `linearinterpol`, `cspline` и `ratinterpol` может иметь одну из следующих форм:

– матрица из двух столбцов, например: `pts: матрица([1,4], [2,6], [3,3])`;

– список пар, например: `pts: [[2,4], [5,6], [9,3]]`;

– список чисел, например: `pts: [4,6,3]`.

В последнем случае абсциссами точек по умолчанию будут 1, 2, 3 и т.д. В первых двух случаях перед выполнением расчетов пары упорядочиваются по первой координате.

С помощью аргумента `option` при вызове функции `lagrange(points, option)` или `linearinterpol(points,option)` можно выбрать имя для независимой переменной, которое по умолчанию `x`. Чтобы определить другое имя, необходимо его задать в виде `varname=<новое имя>`.

Есть три режима вызова функции `cspline(points,option1,option2,...)`, чтобы удовлетворить конкретные запросы:

– `d1` (по умолчанию `unknown`) является первой производной по `x_1`; если имеет значение `unknown`, то вторая производная по `x_1` считается равной 0 (естественный кубический сплайн); если первая производная равна числу, то вторая производная рассчитывается на основе этого числа;

– `dn` (см. `d1`);

– `varname` (см. выше об использовании при вызове функций `lagrange (points,option)` или `linearinterpol(points,option)`).

Функция `ratinterpol(points,numdeg)` производит Паде-аппроксимацию данных, заданных параметром `points` с учетом того, что степень числителя равна `numdeg`. При этом степень знаменателя рассчитывается автоматически. С помощью аргумента `option1` при вызове функции `ratinterpol(points,numdeg,option1)` можно выбрать имя для независимой переменной, которое по умолчанию `x` (см. выше в этом подразделе).

Необходимо помнить, что при использовании полиномов высокой степени аппроксимация в терминах чисел с плавающей точкой становится неустойчивой.

```
(%i1)   load("interpol")$ load("drow")$
```

```
(%i2)   fpprintprec: 5$ pts: makelist([-3.0+k*0.6,sin(-3.0+k*0.6) *
%e^(-(-3.0+k*0.6)^2/2)],k,0,10,1);
(pts)       [[−3.0, −0.0015677], [−2.4, −0.037917], [−1.8, −0.19272],
[−1.2, −0.45367], [−0.6, −0.47163], [0.0, 0.0], [0.6, 0.47163], [1.2, 0.45367],
[1.8, 0.19272], [2.4, 0.037917], [3.0, 0.0015677]]
```

Дополнительные возможности аппроксимации функций дают функции из пакета `lsquares` (см. п. 12.2.5).

10.6. Оптимизация, линейное и нелинейное программирование

Функция	*Описание*
`minpack_lsquares (flist,varlist, guess[,tolerance, jacobian])`	Вычисление точки, в которой минимальна сумма квадратов функций

```
(%i3)   draw2d(dimensions = [1000,670], line_width = 3, line_type =
solid, color=black,explicit(sin(x)*%e^(-x^2/2),x,-3,3), line_width =
5, line_type = dots, color=orange, explicit(lagrange(pts),x,-3,3),
line_width = 3, line_type = dashes, color = red,
explicit(linearinterpol(pts),x,-3,3), line_width = 3, line_type
= dot_dash, color = dark_grey, explicit(cspline(pts),x,-3,3),
line_width = 3, line_type = short_long_dashes, color = dark_violet,
explicit(ratinterpol(pts,5),x,-3,3), point_size = 2, points(pts),
yrange=[-0.55,0.55]);
```

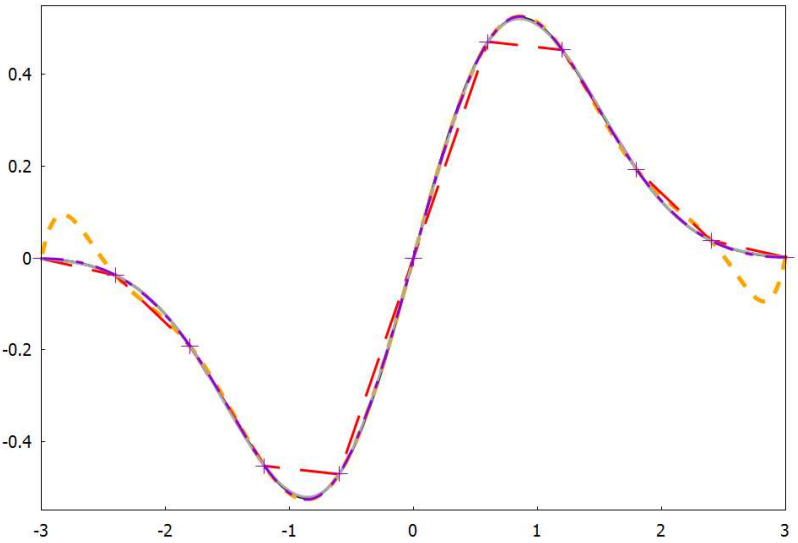


Рис. 10.1

Функции f_i, входящие в сумму

$$S^2 = \sum_{i=1}^{m} f_i^2(x_1, x_2, ..., x_n),$$

представляются в списке `flist`, а имена их аргументов x_j – в списке `varlist`. Начальное приближение минимизации S^2 задается в списке `guess`.

Необязательные ключевые параметры `tolerance` и `jacobian` обеспечивают некоторый контроль за работой алгоритма. `tolerance` – это оцениваемая относительная погрешность вычисления суммы квадратов

функций в точке, соответствующей полученному минимуму. `jacobian` может быть использован для указания вычислять или нет якобиан в символьном виде. Если параметр `jacobian` не задан или имеет значение `true` (по умолчанию), то якобиан системы фуннкций $\{f_i\}$ вычисляется по символьным представлениям элементов списка `flist`. Если же `jacobian` = `false`, то используется численная аппроксимация якобиана.

Функция `minpack_lsquares` возвращает список, первый элемент которого – приближенное решение задачи, второй – сумма квадратов S^2 в точке минимума, а третий – код окончания `info`. Возможные значения последнего:

0 – неправильные входные параметры;

1 – относительная ошибка в S^2 не более, чем `tolerance`;

2 – относительная ошибка между точным и приближенным решением не более, чем `tolerance`;

3 – условия для `info` = 1 и `info` = 2 выполнены;

4 – ортогональность вектора `fvec` столбцам якобиана с машинной точностью;

5 – количество вызовов вычислений S^2 с `iflag` = 1 достигло $100\,(n+1)$;

6 – текущая относительная погрешность слишком мала. Дальнейшее уменьшение суммы квадратов невозможно;

7 – текущая относительная погрешность слишком мала. Дальнейшее улучшение приближенного решения невозможно.

Для вызова функции требуется предварительная загрузка пакета `minpack`.

```
(%i1)   load("minpack")$
```

```
(%i2)   minpack_lsquares([cos(%pi*(x+y)/2), (x-1/4)^2+(y-3/4)^4],
[x,y], [1/2,1/2]);
(%o2)        [[0.2500000020111552, 0.7499999979888449],
6.136578379089145 10^-17, 2]
```

```
(%i3)   powellM(x1,x2,x3,x4) := [x1+10*x2, sqrt(5)*(x3-x4),
(x2-2*x3)^2, sqrt(10)*(x1-x4)^2]$
```

```
(%i4)   minpack_lsquares(powellM(x1,x2,x3,x4), [x1,x2,x3,x4],
[3,-1,0,1]);
(%o4)        [[1.16235490349985 10^-16, 4.649419613999398 10^-17,
5.867280734956379 10^-17, 5.867280734956379 10^-17], 1.161851831106349 10^-32, 4]
```

```
(%i5)   minpack_lsquares(powellM(x1,x2,x3,x4), [x1,x2,x3,x4],
[3,-1,0,1], jacobian = false);
```

(%o5) $[[6.450888984667975\ 10^{-10}, 2.58035559386719\ 10^{-10}, 3.816206728762613\ 10^{-10}, 3.816206728762613\ 10^{-10}], 3.366436557822113\ 10^{-19}, 5]$

Для использования следующей группы функций необходимо сначала загрузить пакет **simplex** – пакет программ, предназначенных для решения задач линейной оптимизации с использованием симплекс-метода [6].

Функция	*Описание*
`linear_program (A, b, c)`	Программная реализация симплекс-метода решения задач линейного программирования

Функция `linear_program(A, b, c)` вычисляет вектор `x`, для которого скалярное произведение `c.x` является минимально возможным среди векторов, для которых `A.x = b` и `x >= 0`. Аргумент `A` является матрицей, а `b` и `c` – списками.

Результат работы функции `linear_program` возвращается в списке, который содержит минимизирующий вектор `x` и минимальное значение целевой функции `c.x`. Если область решения задачи не ограничена, то пользователь получает сообщение "Problem not bounded!", а если же решение задачи отсутствует, то сообщение "Problem not feasible!".

```
(%i6)   load("simplex")$
```

```
(%i7)   A: matrix([1,2,-1,0], [3,-3,1,-1], [4,-5,-1,1])$ b:
[1,2,6]$ c: [1,-2,2,1]$
```

```
(%i8)   linear_program(A, b, c);
```

(%o8) $[[\frac{8}{7}, 0, \frac{1}{7}, \frac{11}{7}], 3]$

Более сложные задачи возникают при решении специально подобранных тестовых примеров, таких как в случае многогранников Кли–Минти[2].

[2]Куб или многогранник Кли–Минти, названный в честь В. Кли и Дж.Дж. Минти (V. Klee, G.J. Minty), – это единичный гиперкуб размерности, как правило, выше трех, переменного размера, углы которого искажены. В. Кли и Дж.Дж. Минти [26] продемонстрировали, что симплекс-метод имеет плохую производительность при неудачном выборе начальной точки, а именно если алгоритм стартует в одной из вершин их деформированного куба. Известно также, что ряд других алгоритмов линейной оптимизации демонстрируют низкую производительность при применении к кубу Кли–Минти. Поэтому куб Кли–Минти служит контрольной задачей при проверке новых алгоритмов решения задач линейного программирования.

Для того чтобы использовать данные для гиперкубов различной размерности применяется функция `klee_minty`, генерирующая входные данные для функции `linear_program`. При таких данных требуется экспоненциальное время решения задачи оптимизации без масштабирования:

$$\max \sum_{j=1}^{n} 10^{n-j} x_j \quad \text{при} \quad 2\sum_{j=1}^{i-1} 10^{i-j} x_j \leqslant 100^{i-1}, \quad 1 \leqslant i \leqslant n, \quad x_j \geqslant 0.$$

```
(%i9)   load(klee_minty);
(%o9)        C:/maxima-5.43.0/share/maxima/5.43.0/share/simplex/Tests/klee
_minty.mac
```

```
(%i10)   apply(linear_program, klee_minty(6));
(%o10)        [[0, 0, 0, 0, 0, 10000000000, 1, 100, 10000, 1000000, 100000000, 0],
−10000000000]
```

```
(%i11)   epsilon_sx: 0$ scale_sx: true$ apply(linear_program,
klee_minty(10));
(%o11)        [[0, 0, 0, 0, 0, 0, 0, 0, 0, 100000000000000000, 1, 100, 10000, 1000000,
100000000, 10000000000, 1000000000000, 100000000000000, 10000000000000000,
0], −100000000000000000]
```

Функция	*Описание*
`maximize_lp(obj,cond,[pos])`	Максимизация линейной целевой функции
`minimize_lp(obj,cond,[pos])`	Минимизация линейной целевой функции

Функция `maximize_lp` максимизирует, а `minimize_lp` минимизирует линейную целевую функцию `obj` с учетом нескольких линейных ограничений `cond`, где `cond` – список линейных уравнений или неравенств. При этом в строгих неравенствах знак `>` заменяется на `>=`, а `<` – на `<=`. Необязательный аргумент `pos` – это список переменных решения, которые предполагаются положительными.

Если максимум (минимум) существует, то функция `maximize_lp` (`minimize_lp`) возвращает список, который содержит максимальное (минимальное) значение целевой функции и список значений переменных, для которых достигается максимум (минимум). Если область решения задачи не ограничена, то пользователь получает сообщение "Problem not bounded!", а если же решение задачи отсутствует, то сообщение "Problem not feasible!".

По умолчанию считается, что переменные задачи не обязаны быть неотрицательными. Если же необходимо, чтобы эти переменные были

неотрицательными, нужно установить для параметра `nonegative_lp` значение `true`. Если только некоторые из переменных задачи являются положительными, требуется их перечислить в необязательном аргументе `pos` (из документации по системе Maxima следует, что это более эффективный путь, чем добавление ограничений).

Опция	*По умолчанию*	*Описание*
`epsilon_lp`	10^{-8}	Используется при численных расчетах в `linear_program`
`nonegative_lp`	`false`	Если параметр имеет значение `true`, то все переменные решения для функций `minimal_lp` и `maximize_lp` считаются положительными
`scale_lp`	`false`	Если `scale_lp` имеет значение true, то функция `linear_program` масштабирует свои входные данные таким образом, чтобы максимальное абсолютное значение в каждой строке или столбце равнялось 1

Результат	*Описание*
`pivot_count_sx`	Содержит число вращений в последнем вычислении в функции `linear_program`
`pivot_max_sx`	Максимальное количество вращений, разрешенное функцией `linear_program` значение в каждой строке или столбце равнялось 1

```
(%i12)  minimize_lp(1/2*x+y, [3*x+2*y>2, x+4*y>3]);
(%o12)      [4/5,[y = 7/10,x = 1/5]]
```

```
(%i13)  minimize_lp(x+y, [3*x+y=0, x+2*y>2]);
(%o13)      [4/5,[y = 6/5,x = -2/5]]
```

```
(%i14)  minimize_lp(x+y, [3*x+y>0, x+2*y>2]), nonegative_lp=true;
(%o14)      [1,[y = 1,x = 0]]
```

```
(%i15)  minimize_lp(x+y, [3*x+y=0, x+2*y>2]), nonegative_lp=true;
(%o15)      Problem not feasible!
```

```
(%i16)  minimize_lp(x+y, [3*x+y>0]);
(%o16)      Problem not bounded!
```

Еще одна функция – `lbfgs` – из одноименного пакета предназначена для решения задач безусловной минимизации с помощью квазиньютоновского алгоритма для ограниченной памяти и основана на алгоритме, который вместо всего обратного гессиана хранит аппроксимацию низкого порядка обратной матрицы для матрицы Гессе.

10.7. Вычисление определенных, несобственных, кратных и специальных интегралов

Пакет `QUADPACK` предоставляет несколько функций для приближенного вычисления определенных интегралов. Пакет загружается автоматически.

Функция	*Описание*
`quad_qag(f(x),x,a,b,key, [epsrel,epsabs,limit])`	Приближенное вычисление определенного интеграла от выражения `expr` по переменной `x` по промежутку от `a` до `b`
`quad_qag(f,x,a,b,key, [epsrel,epsabs,limit])`	
`quad_qags(f(x),x,a,b, [epsrel,epsabs,limit])`	
`quad_qags(f,x,a,b, [epsrel,epsabs,limit])`	
`quad_qagi(f(x),x,a,b, [epsrel,epsabs,limit])`	Приближенное вычисление несобственных интегралов I рода
`quad_qagi(f,x,a,b, [epsrel,epsabs,limit])`	
`quad_qawc(f(x),x,c,a,b, [epsrel,epsabs,limit])`	Вычисление главного значения интеграла по Коши от функции `f(x)/(x-c)`
`quad_qawc(f,x,c,a,b, [epsrel,epsabs,limit])`	
`quad_qawf(f(x),x,a,omega,trig, [epsabs,limit,maxp1,limlst])`	Приближенное вычисление синус- и косинус-преобразований Фурье
`quad_qawf(f,x,a,omega,trig, [epsabs,limit,maxp1,limlst])`	
`quad_qawo(f(x),x,a,b,omega,trig, [epsrel,epsabs,limit,maxp1,limlst])`	Интегрирование `f(x)cos(omega x)` или `f(x)sin(omega x)` по конечному промежутку при `omega=const`
`quad_qawo(f,x,a,b,omega,trig, [epsrel,epsabs,limit,maxp1,limlst])`	
`quad_qaws(f(x),x,a,b,alpha,beta, wfun,[epsrel,epsabs,limit])`	Интегрирование `w(x) f(x)` на конечном интервале, где `w(x)` – алгебраическая или логарифмическая функция
`quad_qaws(f,x,a,b,alpha,beta, wfun,[epsrel,epsabs,limit])`	

Функция	*Описание*
`quad_qagp(f(x),x,a,b,points, [epsrel,epsabs,limit])` `quad_qagp(f,x,a,b,points, [epsrel,epsabs,limit])`	Интегрирование функции по конечному промежутку с указанием сетки

Ниже приводится пример использования функций `quad_qag` и `quad_qags`, результатом работы которых является список из четырех элементов:

- приближенное значение интеграла;
- оценка абсолютной погрешности аппроксимации;
- число вычислений подынтегральной функции;
- код ошибки, который равный 0, если проблем не возникло.

```
(%i1)   quad_qag(sin(x)^(1/2)*log(1/x),x,0,1,3,'epsrel=5d-8);
(%o1)       [0.4376639183627556, 8.737223500044989 10^-9, 899, 0]
```

```
(%i2)   quad_qags(sin(x)^(1/2)*log(1/x),x,0,1,'epsrel=5d-8);
(%o2)       [0.4376639183497706, 6.211697822777751 10^-14, 315, 0]
```

Несложно видеть, что функция `quad_qags` на приведенном примере эффективнее `quad_qag`.

А теперь вычисление несобственного интеграла:

```
(%i3)   [quad_qagi(x^5*exp(-4*x),x,0,inf,'epsrel=1d-8),
float(integrate(x^5*exp(-4*x),x,0,inf))];
(%o3)       [[0.029296875, 1.069776093237562 10^-10, 165, 0], 0.029296875]
```

Функция	*Описание*
`romberg(expr,x,a,b)`	Приближенное вычисление определенного интеграла методом Ромберга
`romberg(F,a,b)`	
`dblint(f,r,s,a,b)`	Приближенное вычисление двойного интеграла

Если `x` – действительное число с плавающей точкой, то параметр `expr` должен быть выражением, значение которого тоже должно быть действительным числом с плавающей точкой.

Вызов функции `romberg`, включенной в пакет `romberg`, в форме `romberg(F,a,b)` возвращает величину интеграла `integrate(F(x),x,a,b)`, где `x` представляет собой безымянный единственный аргумент `F`; фактический аргумент не называется `x`. `F` должна быть функцией Maxima или

Lisp, которая возвращает действительное число с плавающей точкой, когда аргумент является значением с плавающей запятой. F может быть именем оттранслированной или откомпилированной функции Maxima.

Точность функции romberg определяется глобальными переменными rombergabs и rombergtol. Выполнение romberg будет успешно завершено, когда модуль разности между последовательными приближениями меньше, чем rombergabs, или относительная погрешность в последовательных приближениях меньше, чем rombergtol.

Во время расчетов функция romberg для достижения требуемой точности уполовинивает размер шага. Максимальное число вычислений подынтегральной функции составляет 2^rombergit (по умолчанию rombergit=11). Функция romberg всегда делает по крайней мере rombergmin итераций (по умолчанию rombergmin=0). Это эвристика, предназначенная для предотвращения ложного завершения, когда подынтегральное выражение имеет колебательный тип.

```
(%i4)   load("romberg")$ rombergtol: 1e-6$ rombergit: 15$
```

```
(%i5)   h(x):= 1/((x - 1)^2 + 4) + 1/((x + 1)^2 +4)+ 1/(x^2 +16);
(%o5)       h(x) := 1/((x-1)^2+4) + 1/((x+1)^2+4) + 1/(x^2+16)
```

```
(%i6)   float(romberg(h(x),x,-5,5)-integrate(h(x),x,-5,5));
(%o6)       2.291447032121141 10^-10
```

```
(%i7)   g(x,y):=x*y/(x^2+y^2+1);
(%o7)       g(x,y) := x y/(x^2+y^2+1)
```

```
(%i8)   assume(x>0,x<2)$ ap:romberg(romberg(g(x,y),y,0,1-x/2),x,0,2)$
ex: integrate(integrate(g(x,y),y,0,1-x/2),x,0,2)$ float(ap-ex);
(%o8)       5.968016358881556 10^-11
```

Функция dblint использует правило Симпсона в направлениях x и y для расчета двойных интегралов. Для ее применения необходима загрузка пакета dblint.

```
(%i9)   load(dblint)$ dblint_x: 30$ dblint_y: 30$
```

```
(%i10)   F(X,Y):=(mode_declare([X,Y],float),sqrt(1.0-X^2-Y^2))$
R(X):=(mode_declare(X,float),0.0)$ S(X):=(mode_declare(X,float),
sqrt(1.0-X^2))$
```

```
(%i11)   translate(F,R,S)$ dblint('F,'R,'S,0.0,1.0);
(%o11)      0.5234339657130088 + 1.432155719122046 10^-11 %i
```

```
(%i12)   float(%pi/6);
(%o12)      0.5235987755982988
```

Последнее значение – точное решение задачи (восьмая часть объема шара радиуса 1) в форме действительного числа.

10.8. Приближенное интегрирование обыкновенных дифференциальных уравнений

Функция	*Описание*
`rk(ODE,var,initial,domain)`	Метод Рунге–Кутты приближенного решения одного ОДУ
`rk([ODE1,...,ODEm],[v1,...,vm], [init1,...,initm],domain)`	Метод Рунге–Кутты решения системы ОДУ

```
(%i1)   sln: rk([x*((1-x/7)-1.5*y/(1+x)), 0.25*(1-0.5*y/x)*y],
[x,y],[3,1],[t,0,40,0.002])$ r1: makelist([p[1],p[2]],p,sln)$ r2:
makelist([p[1],p[3]],p,sln)$ plot2d([[discrete,r1], [discrete,r2]],
[xlabel,"t"],[ylabel,"x,y"], [legend,"x(t)","y(t)"])$
```

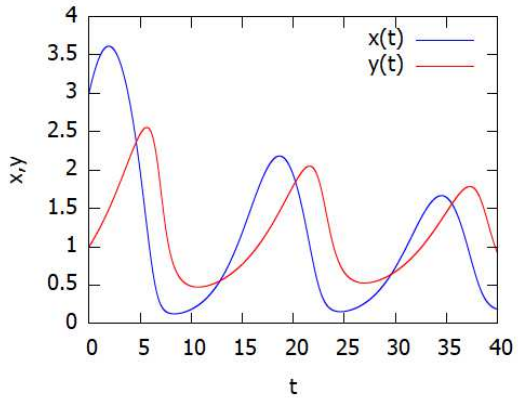


Рис. 10.2

Первая форма вызова функции `rk` решает численно одно обыкновенное дифференциальное уравнение первого порядка, а вторая форма

– систему из m этих уравнений. Используется метод Рунге–Кутты 4-го порядка. mvar представляет зависимую переменную. ОДУ должно быть выражением, которое зависит только от независимых и зависимых переменных и определяет производную зависимой переменной по отношению к независимой переменной.

Если решается m уравнений, то должно быть m зависимых переменных v1, v2, ..., vm. Начальные значения для этих переменных будут init1, init2, ..., initm. Как и в предыдущем случае, область решения определяется только одной независимой переменной. ODE1, ..., ODEm – это выражения, которые определяют производные каждой зависимой переменной по независимой переменной. Единственные переменные, которые могут появляться в этих выражениях, – это независимая переменная и любая из зависимых переменных.

Программа попытается интегрировать уравнения от начального значения независимой переменной до ее последнего значения, используя постоянный шаг. Если на каком-то шаге одна из зависимых переменных принимает слишком большое абсолютное значение, интегрирование будет прекращено в этой точке. Результатом будет список с таким же количеством элементов, каково количество выполненных итераций. Каждый элемент в списке результатов является подсписком из m+1 элементов: значением независимой переменной, за которым следуют значения зависимых переменных, соответствующих этой точке.

11. СПЕЦИАЛЬНЫЕ ФУНКЦИИ

Определение 11.1. *Специальные функции* в широком смысле – совокупность отдельных классов функций, возникающих при решении как теоретических, так и прикладных задач в самых различных разделах математики. В узком смысле под специальными функциями подразумеваются функции математической физики, которые появляются при решении дифференциальных уравнений с частными производными методом разделения переменных.

Специальные функции могут быть определены с помощью степенных рядов, производящих функций, бесконечных произведений, последовательного дифференцирования, интегральных представлений, дифференциальных, разностных, интегральных и функциональных уравнений, тригонометрических рядов, рядов по ортогональным функциям.

Научно-технический прогресс привел к резкому увеличению числа специальных функций, применяемых в приложениях.

К наиболее важным классам специальных функций относятся гамма- и бета-функции; гипергеометрическая и вырожденная гипергеометрическая функции; функции Бесселя, Лежандра, параболического цилиндра; интегральные синус, косинус и др.; неполная гамма-функция; интеграл вероятности; различные классы ортогональных многочленов одного и многих переменных; эллиптическая функция и эллиптический интеграл; функции Ламе, Матьё; дзета-функция Римана; автоморфная функция; некоторые специальные функции дискретного аргумента и т.д.

Теория специальных функций связана с теорией представлений групп, методами интегральных представлений, опирающихся на обобщение формулы Родрига для классических ортогональных многочленов, и методами теории вероятностей.

Для специальных функций имеются таблицы значений, а также таблицы интегралов и рядов.

В связи с широким использованием специальных функций при решении прикладных задач и сложностью обращения с ними во многих САВ реализованы средства, не только позволяющие вычислять значения специальных функций, но и имеющие базы знаний, содержащие основные свойства таких функций. Одной из таких САВ является система Maxima, включающая ряд функций и пакетов функций, которые предназначены для работы:

– с функциями Бесселя, Ламберта, Струве, Ханкеля, Эйри, различными формами гамма и бета-функции, функции ошибок;

– с интегральными экспонентами;
– с гипергеометрическими функциями;
– с эллиптическими функциями и интегралами и др.

Ниже рассматриваются средства для работы с двумя подклассами специальных функций – оптогональными многочленами и многочленами Бернштейна. Описание функций системы Maxima для обращения к другим специальным функциям см. в документации по системе.

11.1. Ортогональные полиномы

Функции, предназначенные для символьных выкладок и численных расчетов при использовании некоторых подклассов *ортогональных полиномов*, включающих полиномы Гегенбауэра, Лагерра, Лежандра, Чебышева, Эрмита и Якоби, представлены в пакете `orthopoly`. Вспомогательные функции обеспечивают получение трехчленных рекуррентных формул для многочленов, интервалов ортогональности и весовых функций. Кроме того, этот пакет включает поддержку сферических функций Бесселя, Ханкеля и сферических гармонических функций.

```
(%i1)  load("orthopoly")$
```

```
(%i2)  expand(hermite(4,x));
(%o2)      12 (4 x^4/3 - 4 x^2 + 1)
```

```
(%i3)  orthopoly_recur(hermite), [n,x]);
(%o3)      H[n+1](x) = 2 H[n](x) x - 2 H[n-1](x) n
```

```
(%i4)  w: orthopoly_weight(hermite, [n,x]);
(w)      [%e^-x^2, -inf, inf]
```

```
(%i5)  integrate(w[1]*hermite(3,x)*hermite(2,x),x,w[2],w[3]);
(%o5)      0
```

```
(%i6)  diff(hermite(n, x),x);
(%o6)      2 H[n-1](x) n
```

```
(%i7)  chebyshev_t(4,x);
(%o7)      -16 (1-x) + 8 (1-x)^4 - 32 (1-x)^3 + 40 (1-x)^2 + 1
```

```
(%i8)  chebyshev_u(4,x);
(%o8)      5 (-8 (1-x) + 16 (1-x)^4/5 - 64 (1-x)^3/5 + 84 (1-x)^2/5 + 1)
```

```
(%i9)  jacobi_p(4,1,2,x);
```

$$(\%o9)\qquad 5\left(-8\,(1-x)+\frac{33\,(1-x)^4}{8}-15\,(1-x)^3+18\,(1-x)^2+1\right)$$

```
(%i10)  laguerre(4,x);
```

$$(\%o10)\qquad \frac{x^4}{24}-\frac{2\,x^3}{3}+3\,x^2-4\,x+1$$

```
(%i11)  legendre_p(4,x);
```

$$(\%o11)\qquad -10\,(1-x)+\frac{35\,(1-x)^4}{8}-\frac{35\,(1-x)^3}{2}+\frac{45\,(1-x)^2}{2}+1$$

```
(%i12)  legendre_q(4,x);
```

$$(\%o12)\qquad \frac{105\log\left(-\frac{x+1}{x-1}\right)x^4-210x^3-90\log\left(-\frac{x+1}{x-1}\right)x^2+110x+9\log\left(-\frac{x+1}{x-1}\right)}{48}$$

```
(%i13)  spherical_bessel_j(4,x);
```

$$(\%o13)\qquad \frac{\left(\frac{-360}{8\,x^2}+\frac{210}{2\,x^4}+1\right)\sin(x)+\frac{10\left(\frac{-252}{24\,x^2}+1\right)\cos(x)}{x}}{x}$$

```
(%i14)  spherical_hankel1(4,x);
```

$$(\%o14)\qquad -\frac{105\,\%i\left(\frac{x^4}{105}+\frac{2\,\%i\,x^3}{21}-\frac{3\,x^2}{7}-\%i\,x+1\right)\%e^{\%i\,x}}{x^5}$$

```
(%i15)  spherical_harmonic(2,2,x,y);
```

$$(\%o15)\qquad \frac{3\sqrt{5}\,\sin(x)^2\,\%e^{2\%iy}}{4\sqrt{6}\,\sqrt{\pi}}$$

```
(%i16)  ultraspherical(2,3,x);
```

$$(\%o16)\qquad 21\left(-\frac{16\,(1-x)}{7}+\frac{8\,(1-x)^2}{7}+1\right)$$

11.2. Полиномы Бернштейна

Определение 11.2. *Многочлены Бернштейна* – это алгебраические многочлены $B_n(f;x)$, представляющие собой линейную комбинацию *базисных многочленов Бернштейна* $b_{kn}(x)$ и играющие важную роль при построении *кривых Безье*.

Базисные многочлены Бернштейна степени n строятся по формуле

$$b_{kn}(x) = \mathsf{C}_n^k\, x^k\, (1-x)^{n-k}, \qquad k = 0, 1, ..., n,$$

где C_n^k – биномиальный коэффициент, и образуют базис для линейного пространства многочленов степени n. В свою очередь многочлен в форме Бернштейна степени n имеет следующий вид:

$$B_n(f;x) = B_n(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) b_{kn}(x),$$

где $f\left(\frac{k}{n}\right)$ – *коэффициенты Бернштейна* или *коэффициенты Безье*.

Для использования функций системы Maxima, ориентированных на расчет коэффициентов Бернштейна, требуется загрузка пакета `bernstein`.

Оператор	*Описание*
`bernstein_poly(k,n,x)`	Базисный многочлен Бернштейна при неотрицательном `k`
`multibernstein_poly ([k1,k2,...,kp],[n1,n2,...,np], [x1,x2,...,xp])`	Произведение базисных многочленов Бернштейна многих переменных
`bernstein_approx(f,[x1,x2,...,xn],n)`	Аппроксимация функции многих переменных многочленами Бернштейна
`bernstein_expand(e,[x1,x2,...,xn])`	Представление полинома `e` многочленами Бернштейна многих переменных

Когда `k` или `n` не являются целыми числами, системный параметр `bernstein_explicit` управляет представлением полиномов Бернштейна в явной форме.

```
(%i1)  load("bernstein")$
```

```
(%i2)  bernstein_poly(k,n,x);
(%o2)     bernstein_poly(k,n,x)
```

```
(%i3)  bernstein_poly(k,n,x), bernstein_explicit: true;
(%o3)        binomial(n,k) (1-x)^(n-k) x^k
```

```
(%i4)  bernstein_approx(f(x),[x],2);
(%o4)     f(1) x^2 + 2 f(1/2) (1-x) x + f(0) (1-x)^2
```

12. ВЕРОЯТНОСТНО-СТАТИСТИЧЕСКИЕ РАСЧЕТЫ

Система Maxima содержит ряд встроенных функций для решения задач теории вероятностей, математической статистики и статистического анализа, которые включены в пакеты `distrib`, `descriptive`, `stats`... Стандартные процедуры статистического оценивания Maxima выполняет достаточно просто и надежно. Открытость программного кода позволяет легко создавать новые, необходимые пользователю функции.

12.1. Теория вероятностей

12.1.1. Описание и расчет характеристик случайных величин

Пакет `distrib` содержит набор функций для вычисления теоретических вероятностных характеристик как для дискретных, так и для непрерывных одномерных случайных величин. В пакете `distrib` есть соглашение об именах. Каждое имя функции состоит из двух частей.

Первая часть имени содержит ссылку на закон распределения, числовую характеристику или параметр случайной величины, который требуется вычислить.

Функция	*Описание*
`pdf_*`	Плотность распределения
`cdf_*`	Функция распределения
`quantile_*`	Квантиль
`mean_*`	Математическое ожидание
`var_*`	Дисперсия
`std_*`	Стандартное отклонение
`skewness_*`	Коэффициент асимметрии
`kurtosis_*`	Коэффициент эксцесса

Вторая часть – это явная ссылка на вероятностную модель, которая может основываться на одном из:

– непрерывных распределений:

Функция	*Описание*
`*normal`	Нормальное
`*student_t`	Стьюдента
`*chi2`	χ^2
`*noncentral_chi2`	Нецентральное χ^2
`*f`	F Фишера–Снедекора
`*exp`	Показательное
`*lognormal`	Логнормальное
`*gamma`	Гамма

Функция	*Описание*
`*beta`	бета
`*continuous_uniform`	Непрерывное равномерное
`*logistic`	Логистическое
`*pareto`	Парето
`*weibull`	Вейбулла
`*rayleigh`	Релея
`*laplace`	Лапласа
`*cauchy`	Коши
`*gumbel`	Гамбеля

– дискретных распределений:

Функция	*Описание*
`*binomial`	Биномиальное
`*poisson`	Пуассона
`*bernoulli`	Бернулли
`*geometric`	Геометрическое
`*discrete_uniform`	Дискретное равномерное
`*hypergeometric`	Гипергеометрическое
`*negative_binomial`	Отрицательное биномиальное
`*general_finite_discrete`	Конечное дискретное

Например, `pdf_student_t(x,n)` – это плотность распределения Стьюдента с n степенями свободы, `std_pareto(a, b)` – стандартное отклонение распределения Парето с параметрами `a` и `b`, а `kurtosis_poisson(m)` – коэффициент эксцесса распределения Пуассона с параметром `m`.

Чтобы использовать пакет `distrib`, сначала нужно загрузить его, набрав

```
(%i1)   load("distrib")$
```

А потом можно использовать функции, включенные в этот пакет.

```
(%i2)   cdf_normal(x,a,sigma);
(%o2)        erf (x-a)/(sqrt(2) sigma) / 2 + 1/2
```

```
(%i3)   pdf_general_finite_discrete(2, [1/7, 4/7, 2/7]);
(%o3)        4/7
```

Здесь указаны вероятности того, что случайная величина примет значения 1, 2, 3.

```
(%i4)   pdf_general_finite_discrete(2, [1, 4, 2]);
(%o4)      4/7
```

А здесь задаются вероятности того, что случайная величина примет значения 1, 2, 3, специальным образом, причем эти вероятности равны также $1/7$, $4/7$, $2/7$.

Кроме того, в пакете `functs` имеется функция `gaussprob(x)`, вычисляющая значения плотности вероятности стандартного нормального распределения.

12.1.2. Задачи с использованием случайных величин

Пример 12.1. В контейнере принесли 70 расставленных случайным образом бутылок напитков четырех различных наименований, причем 10 бутылок первого наименования, 15 – второго, 20 – третьего и 25 – четвертого. На 5 праздничных столов нужно поставить по одинаковому количеству бутылок. С какой вероятностью на каждом столе окажется по 2 бутылки первого наименования, 3 – второго, 4 – третьего и 5 – четвертого, если бутылки вынимаются случайным образом.

◀ Пусть событие $A = \{$на каждом столе по 2 бутылки первого наименования, 3 – второго, 4 – третьего и 5 – четвертого$\}$. Для решения этой задачи на классическую вероятность необходимо посчитать общее число исходов n и число благоприятных исходов m_A для наступления события A.

Тогда n можно подсчитать так: <число возможностей выбора произвольных 14 бутылок из 70> x <число возможностей выбора 14 произвольных бутылок из 56> x <число возможностей выбора 14 произвольных бутылок из 42> x <число возможностей выбора 14 произвольных бутылок из 28> x <число возможностей выбора 14 произвольных бутылок из 14> $= C_{70}^{14} \times C_{56}^{14} \times C_{42}^{14} \times C_{28}^{14} \times C_{14}^{14}$.

Соответственно m_A вычислим так: <число возможностей выбора 2 произвольных бутылок из 10 бутылок первого наименования, 3 из 15 – второго, 4 из 20 – третьего и 5 из 25 – четвертого> x <число возможностей выбора 2 произвольных бутылок из 8 бутылок первого наименования, 3 из 12 – второго, 4 из 16 – третьего и 5 из 20 – четвертого> x <число возможностей выбора 2 произвольных бутылок из 6 бутылок первого наименования, 3 из 9 – второго, 4 из 12 – третьего и 5 из 15 – четвертого> x <число возможностей выбора 2 произвольных бутылок из 4 бутылок первого наименования, 3 из 6 – второго, 4 из 8 – третьего и 5 из 10 – четвертого> x <число возможностей выбора 2 произвольных бутылок из

2 бутылок первого наименования, 3 из 3 – второго, 4 из 4 – третьего и 5 из 5 – четвертого> $= \left(\mathsf{C}_{10}^2 \cdot \mathsf{C}_{15}^3 \cdot \mathsf{C}_{20}^4 \cdot \mathsf{C}_{25}^5\right) \times \left(\mathsf{C}_8^2 \cdot \mathsf{C}_{12}^3 \cdot \mathsf{C}_{14}^4 \cdot \mathsf{C}_{20}^5\right) \times \left(\mathsf{C}_6^2 \cdot \mathsf{C}_9^3 \cdot \mathsf{C}_{12}^4 \cdot \mathsf{C}_{15}^5\right) \times \left(\mathsf{C}_4^2 \cdot \mathsf{C}_6^3 \cdot \mathsf{C}_8^4 \cdot \mathsf{C}_{10}^5\right) \times \left(\mathsf{C}_2^2 \cdot \mathsf{C}_3^3 \cdot \mathsf{C}_4^4 \cdot \mathsf{C}_5^5\right)$.

Тогда вероятность события A будет равна:

$$\mathcal{P}(A) = \frac{m_A}{n},$$

$$m_A = \left(\mathsf{C}_{10}^2 \cdot \mathsf{C}_{15}^3 \cdot \mathsf{C}_{20}^4 \cdot \mathsf{C}_{25}^5\right) \times \left(\mathsf{C}_8^2 \cdot \mathsf{C}_{12}^3 \cdot \mathsf{C}_{14}^4 \cdot \mathsf{C}_{20}^5\right) \times \left(\mathsf{C}_6^2 \cdot \mathsf{C}_9^3 \cdot \mathsf{C}_{12}^4 \cdot \mathsf{C}_{15}^5\right) \times$$
$$\times \left(\mathsf{C}_4^2 \cdot \mathsf{C}_6^3 \cdot \mathsf{C}_8^4 \cdot \mathsf{C}_{10}^5\right) \times \left(\mathsf{C}_2^2 \cdot \mathsf{C}_3^3 \cdot \mathsf{C}_4^4 \cdot \mathsf{C}_5^5\right),$$
$$n = \mathsf{C}_{70}^{14} \times \mathsf{C}_{56}^{14} \times \mathsf{C}_{42}^{14} \times \mathsf{C}_{28}^{14} \times \mathsf{C}_{14}^{14}.$$

Довести до конечного результата такие вычисления не удастся ни вручную, ни на обычном калькуляторе. Воспользуется "калькулятором" 𝕄axima:

```
(%i5)   n: binomial(70,14)*binomial(56,14)*binomial(42,14)*
binomial(28,14)*binomial(14,14);
(n)      2378829279642818668557063558238537401024000000
```

```
(%i6)   mA: binomial(10,2)*binomial(15,3)*binomial(20,4)*binomial
(25,5)*binomial(8,2)*binomial(12,3)*binomial(16,4)*binomial(20,5)
*binomial(6,2)*binomial(9,3)*binomial(12,4)*binomial(15,5)
*binomial(4,2)*binomial(6,3)*binomial(8,4)*binomial(10,5)
*binomial(2,2)*binomial(3,3)*binomial(4,4)*binomial(5,5);
(mA)      3632154140097479439823383843840000000000
```

```
(%i7)   print("P(A)=", mA/n, "=", float(mA/n))$
```

$$\mathsf{P(A)} = \frac{140446819178986320000}{919837079619937926255412 27} = 1.526866249369038\ 10^{-6}$$

Пример 12.2. На сборочный конвейер поступает поток одинаковых микросхем (МС) разных партий. Известно, что в таком потоке может быть с равной вероятностью от 0 до 5 бракованных на 1000 МС. В каждый прибор монтируется 10 МС, причем такой прибор будет работать, если в него попадет не более одной бракованной МС. С какой вероятностью случайно отобранный после сборки прибор будет работоспособным?

◀ Пусть событие $A = \{$случайно отобранный после сборки прибор работоспособен$\}$. Для решения задачи нужно воспользоваться формулой полной вероятности, причем гипотезами будут $\mathcal{H}_k = \{$МС из потока, в котором k бракованных на 1000 МС$\}$, $k = 0, 1, 2, 3, 4, 5$, а все вероятности гипотез равны $1/6$. Тогда

$$\mathcal{P}(A|\mathcal{H}_0) = 1, \qquad \mathcal{P}(A|\mathcal{H}_1) = 1,$$

$$\mathcal{P}(A|\mathcal{H}_2) = \frac{\mathsf{C}_{9998}^{10}}{\mathsf{C}_{1000}^{10}} + \frac{\mathsf{C}_{9998}^{9}\,\mathsf{C}_{2}^{1}}{\mathsf{C}_{1000}^{10}}, \qquad \mathcal{P}(A|\mathcal{H}_3) = \frac{\mathsf{C}_{9997}^{10}}{\mathsf{C}_{1000}^{10}} + \frac{\mathsf{C}_{9997}^{9}\,\mathsf{C}_{3}^{1}}{\mathsf{C}_{1000}^{10}},$$
$$\mathcal{P}(A|\mathcal{H}_4) = \frac{\mathsf{C}_{9996}^{10}}{\mathsf{C}_{1000}^{10}} + \frac{\mathsf{C}_{9996}^{9}\,\mathsf{C}_{4}^{1}}{\mathsf{C}_{1000}^{10}}, \qquad \mathcal{P}(A|\mathcal{H}_5) = \frac{\mathsf{C}_{9995}^{10}}{\mathsf{C}_{1000}^{10}} + \frac{\mathsf{C}_{9995}^{9}\,\mathsf{C}_{5}^{1}}{\mathsf{C}_{1000}^{10}},$$

а вероятность события A вычислится по формуле:

$$\mathcal{P}(A) = \mathcal{P}(A|\mathcal{H}_0)\,\mathcal{P}(\mathcal{H}_0) + \mathcal{P}(A|\mathcal{H}_1)\,\mathcal{P}(\mathcal{H}_1) + \mathcal{P}(A|\mathcal{H}_2)\,\mathcal{P}(\mathcal{H}_2) + \\ + \mathcal{P}(A|\mathcal{H}_3)\,\mathcal{P}(\mathcal{H}_3) + \mathcal{P}(A|\mathcal{H}_4)\,\mathcal{P}(\mathcal{H}_4) + \mathcal{P}(A|\mathcal{H}_5)\,\mathcal{P}(\mathcal{H}_5)\,.$$

Несложно видеть, что опять требуется расширенная арифметика, в результате применения которой получим:

```
(%i8)   s0: binomial(1000,10)$ s2: binomial(998,10) +
binomial(998,9) * binomial(2,1)$ s3: binomial(997,10) +
binomial(997,9) * binomial(3,1)$ s4: binomial(996,10) +
binomial(996,9) * binomial(4,1)$ s5: binomial(995,10) +
binomial(995,9) * binomial(5,1)$ r: 1/6*(1+1+(s2+s3+s4+s5)/s0)$
```

```
(%i9)   print("P(A)=", r, "=", float(r))$
```

$$\mathsf{P(A)} = \frac{687320279191}{687524270850} = 0.9997032953342173$$

Пример 12.3. Химический завод изготовляет серную кислоту номинальной плотности 1,84 г/см^3. В результате статистических испытаний обнаружено, что практически 99,9% всех выпускаемых реактивов имеют плотность в интервале (1,82; 1,86). Найти вероятность того, что кислота удовлетворяет стандарту, если для этого достаточно, чтобы ее плотность не отклонялась от номинала более, чем на 0,01 г/см^3. Считать, что плотность кислоты распределена по нормальному закону.

◀ Пусть X – случайная плотность серной кислоты (г/см^3). Из условий задачи следует, что $m_X = 1.84$, $\mathcal{P}(X \in (1.82, 1.86)) = 0.999$. При этом нужно найти вероятность $\mathcal{P}(|X - m_X| < \epsilon)$, где $\epsilon = 0.01$.

Известно, что

$$\mathcal{P}(a < X < b) = F_X(b) - F_X(a),$$

где в рассматриваемом случае $F_X(x) = F_{N(m_X,\sigma_X)}(x)$ – функция распределения нормально распределенной случайной величины X. Плотность распределения $f_X(x)$ (и функция распределения) X зависят от двух параметров m_X и среднего квадратичного отклонения σ_X, которые необходимы и для вызова функции `cdf_normal`, вычисляющей значения

$F_{N(m_X,\sigma_X)}(x)$. m_X известно, а σ_X нет. Найти этот параметр можно из заданного соотношения ($\varepsilon = 0.02$):

$$0.999 = \mathcal{P}(X \in (1.82, 1.86)) = \mathcal{P}(X \in (m_X - \varepsilon, m_X + \varepsilon)) =$$
$$= F_{N(m_X,\sigma_X)}(a + \varepsilon) - F_{N(m_X,\sigma_X)}(a - \varepsilon) = 2\,F_{N(0,1)}(\varepsilon/\sigma_X),$$

т.е. $2F_{N(0,1)}(\varepsilon\sigma_X) = 0.999$, или $F_{N(0,1)}(\varepsilon/\sigma) = 0.4995$. Отсюда можно найти значение аргумента функции $F_{N(0,1)}$, затем σ_X и, наконец, искомое решение задачи:

```
(%i10)   load("distrib")$ fpprintprec: 8$ mX: 1.84$ eps: 0.02$
epss:0.01$
```

```
(%i11)   arg:float(quantile_normal(0.995,0.0,1.0))$ sigmaX:eps/arg;
(sigmaX)       0.0077644897
```

```
(%i12)   print("P(|X-",mX,"| <",epss,")=", float(cdf_normal(mX+epss,
mX,sigmaX) - cdf_normal(mX-epss,mX,sigmaX)))$
     P(|X -1.84| < 0.01) = 0.80222433
```

12.2. Математическая статистика

12.2.1. Описательная статистика

В системе Maxima средства описательной статистики объединены в пакет `descriptive`, который включает следующие функции:

Функция	*Описание*
`mean`	Среднее арифметическое вектора и матрицы
`geometric_mean`	Среднее геометрическое
`harmonic_mean`	Среднее гармоническое
`cor`	Выборочная корреляционная матрица
`cov`, `cov1`	Выборочная ковариационная матрица
`median`	Выборочная медиана
`var`, `var1`	Выборочная и исправленная выборочная дисперсия случайной величины
`std`, `std1`	Выборочное и исправленное выборочное среднее квадратичное отклонение
`noncentral_moment`	Выборочный начальный момент заданного порядка
`central_moment`	Выборочный центральный момент заданного порядка
`cv`	Выборочный коэффициент вариации
`median`	Выборочная медиана
`skewness`	Выборочная асимметрия

Функция	*Описание*
kurtosis	Выборочный эксцесс
quantile	Выборочный квантиль
maxi, maxi	Наибольшее и наименьшее значения
range	Размах выборки
smin/smax	Минимальная/максимальная варианта выборки
qrange	Выборочная интерквартильная широта
mean_deviation / median_deviation	Сумма абсолютных отклонений от выборочного среднего / медианы
pearson_skewness	Выборочный коэффициент асимметрии Пирсона
quartile_skewness	Выборочная квартильная асимметрия
km	Оценка Каплана–Мейера функции надежности
cdf_empirical	Эмпирическая функция распределения
list_correlations	Список, включающий две матрицы – матрицу, обратную ковариационной, и матрицу частных коэффициентов корреляции
global_variances	Список, содержащий различные формы характеристик рассеивания
principal_components	Список, содержащий главные компоненты многомерной выборки

В рамках пакета возможны следующие действия с выборками:

Функция	*Описание*
build_sample	Построение выборки
continuous_freq	Построение интервального статистического ряда для непрерывного признака
disc rete_freq	Построение статистического ряда для дискретного признака
standardize	Стандартизация элементов выборки
subsample	Выделение части выборки
transform_sample	Преобразование многомерной выборки согласно заданным функциям

Пример вычисления наибольшего и наименьшего значений, минимальной и максимальной вариант, размаха выборки; среднего арифметического, выборочной и исправленной выборочной дисперсии, выборочной медианы; выборочных асимметрии, эксцеса, квантиля 0.5 (медианы), интерквартильной широты.

```
(%i1)   load("descriptive")$ s: openr("d:/datafile.txt")$ L:
read_list(s,'comma)$
```

```
(%i2)   print(maxi(L)," ",mini(L)," ",smin(L)," ",smax(L),"
",range(L))$
     4.0     3.5     3.5     4.0     0.5
```

```
(%i3)   print(mean(L)," ",var(L)," ",var1(L)," ",cv(L),"
",median(L))$
     3.7125     0.01609375     0.016612903     0.034171352     3.7
```

```
(%i4)   print(skewness(L)," ",kurtosis(L)," ",quantile(L,0.5),"
",qrange(L))$
     0.22385151     -0.63700632     3.7     0.2
```

12.2.2. Статистическая графика

Система Maxima предоставляет функции для построения графических иллюстраций в рамках описательной статистики, которые объединены в пакет `descriptive`:

Функция	*Описание*
`scatterplot`	Непосредственная визуализация данных
`histogram`	Построение гистограммы
`barsplot`	Построение столбиковых диаграмм
`boxplot`	Построение графика Бокса–Уискера ("ящик с усами")
`piechart`	Построение круговых диаграмм
`boxplot_description`	Объединение различных графиков
`starplot`	Звездные диаграммы для дискретных признаков как для одной, так и для нескольких выборок
`stemplot`	Построение стволовых и листовых диаграмм

Функция `barsplot(data1,data2,...,option_1,option_2,...)` строит столбиковые диаграммы для дискретных случайных величин как для одной, так и для нескольких выборок. Параметр `data` может быть списком результатов, представляющих одну выборку, или матрицей из m строк и n столбцов, представляющих n выборок размером m каждая.

Возможные опции: `box_width` (относительная ширина столбцов, значения: от 0 до 1), `grouping` (гуппировка, clustered / stacked), `groups_gap` (зазор между группами столбцов),`bars_colors` (список цветов),`frequency` (частота, absolute / relative / percent),`ordering` (упорядочевание,orderlessp / ordergreatp), `sample_keys` (строки легенды), start_at (начало графика), все глобальные опции `draw`, за исключением некоторых.

Также есть функция `wxbarsplot` для создания встроенных гистограмм в интерфейсах wxMaxima и iMaxima.

Далее приведены команды для построения и график Бокса–Уискера ("ящик с усами") с аннотациями по осям (рис. 12.2).

```
(%i5)   s: openr("d:/datafile.txt")$ L: read_list(s,'comma)$
(%i6)   barsplot(L, box_width=1, fill_density=1, groups_gap=1,
bars_colors=[orange])$
```

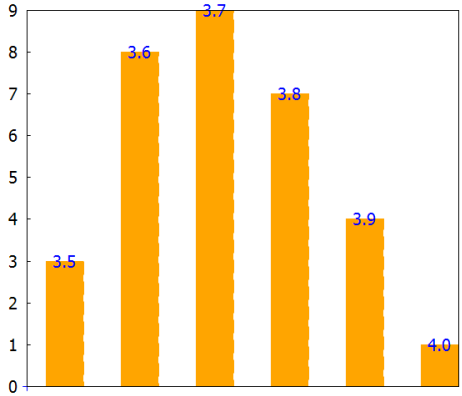


Рис. 12.1

```
(%i7)   s0: read_matrix(file_search("wind.data"))$
(%i8)   boxplot(s0, xlabel="Сезоны", terminal=png,
file_name="d:/CASM_12_02_02_020")$
```

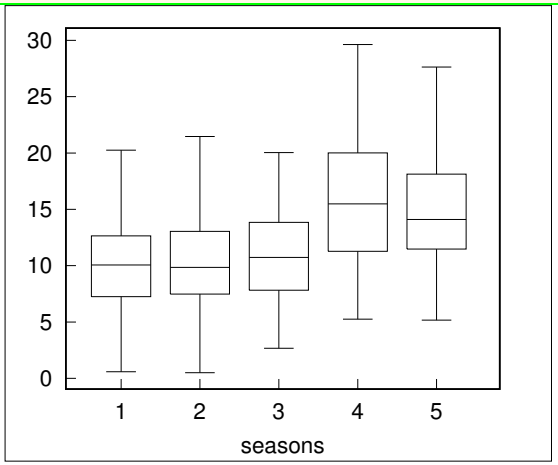


Рис. 12.2

Наиболее популярный инструмент дескриптивного анализа непрерывных признаков – гистограммы. Ниже (рис. 12.3) приведен пример построения такого представления данных.

```
(%i9)   histogram(L, nclasses=6, fill_color=grey, fill_density=0.3,
terminal=png, file_name="d:/CASM_12_02_02_030")$
```

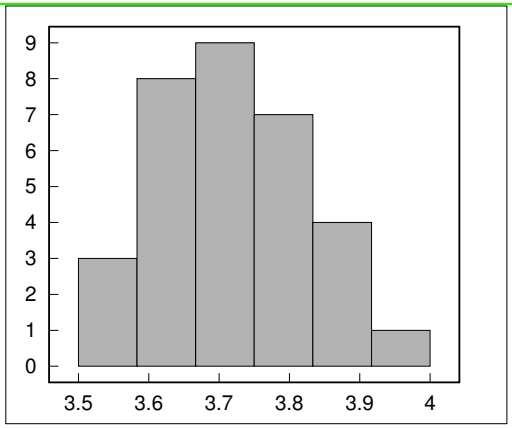


Рис. 12.3

Следующим демонстрируем результат построение корреляционного поля (рис. 12.4).

И последний пример статистической графики – круговая диаграмма (рис. 12.5).

12.2.3. Проверка статистических гипотез

Пакет `stats` содержит набор классических процедур статистического вывода и проверки гипотез. Все эти функции возвращают объект Maxima типа `inference_result`, который содержит необходимые результаты для выводов по генеральной совокупности и принятия решений. Глобальная переменная `stats_numer` контролирует, будут ли результаты представлены в формате с плавающей запятой, в символьном или рациональном формате.

При проверке статистических гипотез пакет `stats` позволяет, в частности, проводить сопоставление средних или дисперсий двух выборок. Предусмотрена проверка нормальности распределения, а также ряд других стандартных тестов, хотя ряд функций перегружен дополнениями, что затрудняет их использование.

```
(%i10)   load("stats")$ x: random_normal(2,0.5,50)$ y:
random_normal(3,0.25,50)$
(%i11)   LL: makelist([x[k],y[k]],k,1,50)$ pts: apply(matrix,LL)$
(%i12)   scatterplot(pts,terminal=png,file_name="d:/CASM_12_02_02_04")$
```

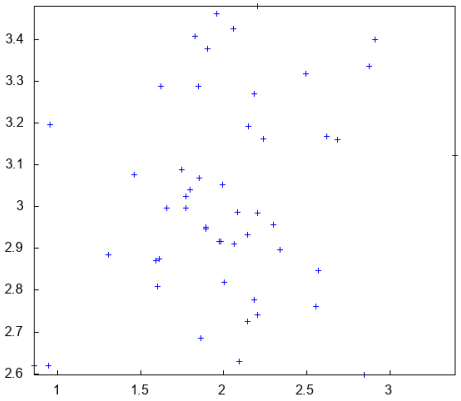


Рис. 12.4

```
(%i13)   piechart(L, xrange=[-1.1, 1.1], yrange=[-1.1, 1.1])$
```

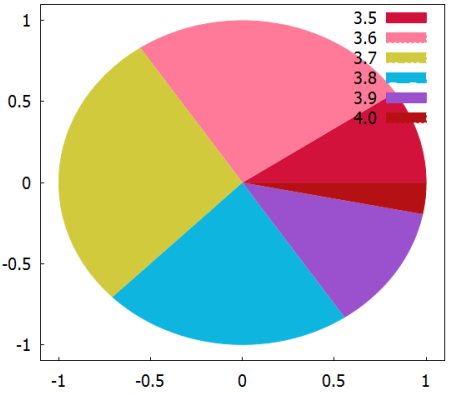


Рис. 12.5

Данный пакет содержит некоторые утилиты для манипулирования структурами данных (списков и матриц), например, для извлечения под-

выборок.

Для использования `stats` пакет необходимо загрузить командой

```
(%i14)   load("stats")$
```

В свою очередь `stats` подгружает обязательные пакеты `descriptive`, `distrib` и `inference_result`.

Функции для построения законов распределения статистик:

Функция	*Описание*
`pdf_signed_rank`	Плотность вероятности точного распределения знаковой ранговой статистики
`cdf_signed_rank`	Функция распределения точного закона распределения знаковой ранговой статистики
`pdf_rank_sum`	Плотность вероятности точного распределения статистики ранговых сумм
`cdf_rank_sum`	Функция распределения точного закона распределения статистики ранговых сумм

Функции для проверки статистических гипотез:

Функция	*Описание*
`test_mean`	t-тест для математического ожидания, включая точечное и интервальное оценивание
`test_means_difference`	t-тест разности средних для двух выборок, включая доверительные интервалы
`test_variance`	χ^2-тест для дисперсии, включая точечное и интервальное оценивание
`test_variance_ratio`	F-критерий отношения дисперсий для двух нормальных совокупностей, включая точечное и интервальное оценивание
`test_proportion`	Выводы о долях, включая точечное и интервальное оценивание
`test_proportions_difference`	Выводы о разности двух долей, включая точечное и интервальное оценивание
`test_sign`	Непараметрический критерий знаков для медианы непрерывной совокупности
`test_signed_rank`	Знаковый ранговый тест Уилкоксона, позволяющий сделать выводы о медиане непрерывной совокупности
`test_rank_sum`	Тест Уилкоксона–Манна–Уитни для сравнения медиан двух непрерывных совокупностей
`test_normality`	Проверка нормальности распределения по критерию Шапиро–Уилка

Рассмотрим параметры одной из функций этого списка и структуру результата ее работы.

Функция `test_mean` предназначена для проведения t-теста проверки математического ожидания на совпадение с заданным значением. Имеются две формы вызова этой функции – `test_mean(x)` и `test_mean(x,opts ...)`. Аргумент `x` – это список или матрица столбцов, содержащая одномерную выборку.Эта же функция выполняет асимптотический тест на основе центральной предельной теоремы, если опция `'asymptotic` верна.

При вызове функции `test_mean` можно использовать следующие опции:

– `'mean` (по умолчанию 0), математическое ожидание для проверки;

– `'alternative` (по умолчанию `'twosided`), конкурирующая гипотеза; возможные значения: `'twosided`, `'greater` и `'less`;

– `'dev` (по умолчанию `'unknown`), значение стандартного отклонения, когда оно известно; возможные значения: tpm'unknown или положительное выражение;

– `'conflevel` (по умолчанию 95/100), доверительная вероятность для построения доверительного интервала; это должно быть выражение, которое принимает значение из интервала (0,1);

– `'asymptotic` (по умолчанию `false`), признак применения точного t-критерия или его асимптотического варианта; возможные значения: `true` и `false`.

Результаты работы функции `test_mean` представляются в виде объекта Maxima `inference_result`, содержащего следующие элементы:

– `'mean_estimate`: среднее арифметическое выборки;

– `'conf_level`: доверительная вероятность, выбранная пользователем;

– `'conf_interval`: доверительный интервал для математического ожидания;

– `'method`: наименование теста и условия применения;

– `'hypotheses`: нулевая и альтернативные гипотезы для проверки;

– `'statistic`: значение выборочной статистики, используемой для проверки нулевой гипотезы;

– `'distribution`: распределение выборочной статистики вместе с ее параметрами;

– `'p_value`: уровень значимости.

Ниже функция `test_mean` вызывается для проверки точного t-критерия с неизвестной дисперсией признака X. Нулевая гипотеза $\mathcal{H}_0 : m_X = 3.7$ против двусторонней альтернативы $\mathcal{H}_1 : m_X \neq 3.7$.

```
(%i15)   load("stats")$ s: openr("d:/datafile.txt")$ L:
read_list(s,'comma)$
```

```
(%i16)  test_mean(L,'conflevel=0.95,'alternative='twosided,'mean=37);
```

(%o16)

$$MEAN\ TEST$$
$$mean\_estimate = 3.712500000000001$$
$$conf\_level = 0.95$$
$$conf\_interval = [3.666029820354814, 3.758970179645187]$$
$$method = Exact\ t-test.\ Unknown\ variance.$$
$$hypotheses = H0:\ mean\ =\ 3.7,\ H1:\ mean\ \#\ 3.7$$
$$statistic = 0.5486081240616457$$
$$distribution = [student\_t, 31]$$
$$p\_value = 0.5872041510640704$$

Согласно полученным результатам значение 'p_value достаточно велико, чтобы отвергнуть нулевую гипотезу $\mathcal{H}_0$.

12.2.4. Линейный и нелинейный регрессионный анализ

Функция	*Описание*
`linear_regression(X)`	Линейная регрессия

Функция `linear_regression` строит уравнение множественной регрессии $Y = b_0 + b_1 X_1 + b_2 X_2 + ... + b_k X_k + \varepsilon$, где ε – случайная величина, имеющая нормальное распределение $\mathcal{N}(0, \sigma^2)$, по представленным в матрице `X` выборочным данным. Параметр `X` должен быть матрицей с более чем одним столбцом, причем в последний записываются варианты y_i.

При вызове функции `linear_regression,option)` параметр `option` задает уровень значимости γ $(0 < \gamma < 1$, по умолчанию $\gamma = 0.95)$ и записывается в виде `conflevel=<`γ`>`.

Результатом работы функции `linear_regression` является объект `inference_result` системы Maxima со следующими компонентами:

– `b_esvaluation`: оценки коэффициентов регрессии;

– `b_covariances`: ковариационная матрица оценок коэффициентов регрессии;

– `b_conf_int`: доверительные интервалы для коэффициентов регрессии;

– `b_statistics`: статистика при проверке коэффициентов;

– `b_p_values`: p-значения для тестов коэффициентов;

– `b_distribution`: распределение вероятностей для тестирования коэффициентов;

– `v_estimation`: несмещенная оценка дисперсии;
– `v_conf_int`: доверительный интервал для дисперсии;
– `v_distribution`: распределение вероятностей для тестирования дисперсии;
– `residuals`: остатки;
– `adc`: скорректированный коэффициент детерминации;
– `aic`: информационный критерий Акаике;
– `BIC`: информационный критерий Байеса.

По умолчанию отображаются только пункты 1, 4, 5, 6, 7, 8, 9 и 11 в указанном порядке. Остальные можно увидеть, если воспользоваться функциями `items_inference` и `take_inference`.

Для использования функции `linear_regression` должен быть загружен пакет `stats`.

```
(%i17)   load("stats")$ fpprintprec: 4$
```

```
(%i18)   sample: matrix( [16.3,82.4,77.4], [7.1,71.9,74.3],
[16.4,152.0,93.6], [18.7,149.6,152.8], [8.7,74.2,94.9],
[11.0,86.7,117.1], [7.1,54.8,45.7], [8.7,67.5,64.9],
[10.1,73.9,98.9], [7.0,48.1,45.6], [11.4,78.5,97.0],
[9.9,67.5,74.3], [7.1,65.8,72.8], [12.3,88.5,80.0],
[16.7,112.1,118.2], [23.6,111.8,143.0])$
```

```
(%i19)   res: linear_regression(sample);
```

(res)

$$
\begin{aligned}
&\textit{LINEAR REGRESSION MODEL}\\
&b_{estimation} = [22.38, 3.04, 0.367]\\
&b_{statistics} = [1.495, 1.829, 1.353]\\
&b\_p\_values = [0.1587, 0.09038, 0.199]\\
&b_{distribution} = [student_t, 13]\\
&v_{estimation} = 367.5\\
&v\_conf\_int = [193.1, 953.8]\\
&v_{distribution} = [chi2, 13]\\
&adc = 0.6077
\end{aligned}
$$

Функция	*Описание*
`lsquares_estimates (M,vars,eq,pars)`	Оценка параметров модели с помощью МНК
`lsquares_estimates _exact(MSE,pars)`	Точная оценка параметров `pars` для минимизации среднеквадратичной ошибки `MSE`
`lsquares_estimates _approximate(MSE,pars, initial=L,tol=t)`	Приближенная оценка параметров `pars` для минимизации среднеквадратичной ошибки `MSE`
`lsquares_mse(M,vars,eq)`	Расчет среднеквадратичной ошибки
`lsquares_residuals (M,vars,eq,pars)`	Расчет остатков для уравнения `eq` с указанными параметрами `pars` и данными `M`

Дополнительные вычислительные средства для нелинейного регрессионного анализа собраны в пакете `lsquares`, который представляет собой набор функций для расчетов в рамках метода наименьших квадратов (МНК) для оценки параметров моделей по числовым данным.

Параметрами функции `lsquares_estimates` являются: `M` – матрица с данными, `vars` – имена переменных модели, `eq` – модель, `pars` – имена параметров модели. Обычно сначала функция `lsquares_estimates` ищет точное решение, а если это не удается, то вычисляет приближенное. Результатом работы функции является список подсписков, элементами которых являются равенства типа `[a=..., b=..., c=...]`. Каждый элемент такого списка дает отдельный эквивалентный минимум среднеквадратичной ошибки.

В матрице `M` каждая строка представляет собой один элемент данных, а каждый столбец содержит значения одной переменной. Уравнение `eq` представляет собой выражение или уравнение в переменных `vars` и `pars`. Если `eq` не является равенством, то оно рассматривается так же, как `eq=0`.

Дополнительные аргументы в вызове `lsquares_estimates(M,vars, eq,pars,initial=L,tol=t)` указываются в виде равенств и напрямую передаются функции `lbfgs`, которая вызывается для поиска оценок параметров модели численным методом, когда точный результат не может быть найден.

Если какое-либо точное решение может быть найдено (с помощью `solve`), данные в матрице `M` могут содержать нечисловые значения. Однако если точное решение не найдено, каждый элемент ttM должен иметь числовое значение. Сюда входят числовые константы, такие как `%pi` и `%e`, а также алгебраические числа (целые, рациональные, обычные и длинные числа с плавающей точкой). Численные расчеты выполняются с использованием обычной арифметики с плавающей точкой. Поэтому все другие виды чисел для расчетов преобразуются в числа с плавающей точкой стандартной точности.

Если функции `lsquares_estimates` требуется слишком много времени или не хватает памяти, то можно воспользоваться функцией `lsquares_estimates_approximate`, которая пропускает этап поиска точного решения. Возможно, эта функция сможет получить решение.

Функция `lsquares_estimates_exact(MSE,pars)` (`lsquares_estimates_approximate(MSE,pars,initial=L,tol=t)`) вычисляет параметры `pars` так, чтобы минимизировать среднеквадратичную ошибку `MSE`, на основе построения системы уравнений для параметров модели и использования символьного (приближенного) решения с помощью функции `solve` (`lbfgs`). Среднеквадратичная ошибка – это выражение, зависящее от па-

раметров модели, например, возвращаемое функцией `lsquares_mse`.

Структура результата сходна с формой возвращаемого функцией `lsquares_estimates` списка, который может содержать ноль, один, два или более элементов (для функции `lsquares_estimates_approximate` один). Если возвращается два подсписка или более, то каждый из них представляет собой отдельный эквивалентный минимум среднеквадратичной ошибки.

Дополнительные аргументы функции `lsquares_estimates_approximate(MSE,pars,initial=L,tol=t)` указываются в виде равенств и напрямую передаются функции `lbfgs`.

```
(%i20)   load("lsquares")$
```

```
(%i21)   sample: matrix([0,1,1],[1,3,2],[2,2,3],[3,5/2,2],[4,3,1])$
```

```
(%i22)   rez: lsquares_estimates(sample,[z,x,y],(z+z0)^2=cx*x+cy*y+
c0,[cx, cy,c0,z0]);
(rez)        [[cx = -5/9, cy = -25/12, c0 = 36481/5184, z0 = -151/72]]
```

```
(%i23)   lsquares_residuals(sample,[z,x,y],(z+z0)^2=cx*x+cy*y+c0,
first(rez));
(%o23)       [0, 0, 1/3, -2/3, 1/3]
```

12.2.5. Генерирование псевдослучайных чисел

Оператор	*Описание*
`random(x)`	Вычисление ПСЧ
`make_random_state(n)`	Новый объект случайного состояния, создаваемый из целочисленного начального значения, равного `n` по модулю 2^{32}
`make_random_state(s)`	Копия случайного состояния `s`
`make_random_state(true)`	Новый объект случайного состояния, использующий текущее состояние компьютерных часов в качестве начального значения
`make_random_state(false)`	Копия состояния генератора ПСЧ
`set_random_state(s)`	Копирование `s` в структуру состояния генератора ПСЧ

Функция `random(x)` возвращает псевдослучайное число (ПСЧ). Если `x` является целым числом, то результат – целое число в промежутке от 0 до `x - 1` включительно. Если `x` – число с плавающей точкой, то результат – неотрицательное число с плавающей запятой, меньшее, чем `x`. Функция `random(x)` завершается с ошибкой, если `x` не является ни целым числом, ни числом с плавающей точкой, а также если $x < 0$.

Функция `make_random_state` представляет структуру случайного состояния генератора ПСЧ. Эта структура состоит из 627 32-битных слов.

Генератор ПСЧ системы Maxima является реализацией вихря Мерсенна MT 19937[1].

Функция	*Описание*
`random_normal(m,s)`	Вычисление ПСЧ в соответствии с нормальным распределением $\mathcal{N}(m, s^2)$
`random_normal(m,s,k)`	Вычисление k ПСЧ в соответствии с нормальным распределением $\mathcal{N}(m, s^2)$
`random_exp(lambda)`	Вычисление ПСЧ в соответствии с показательным распределением с параметром $lambda > 0$
`random_exp(lambda,k)`	Вычисление k ПСЧ в соответствии с показательным распределением с параметром $lambda > 0$
`random_beta(a,b)`	Вычисление ПСЧ в соответствии с бета-распределением с параметрами $a > 0$, $b > 0$
`random_beta(a,b,k)`	Вычисление k ПСЧ в соответствии с бета-распределением с параметрами $a > 0$, $b > 0$
`random_continuous_uniform(a,b)`	Вычисление ПСЧ в соответствии с непрерывным равномерным распределением на интервале (a, b)
`random_continuous_uniform(a,b,k)`	Вычисление k ПСЧ в соответствии с непрерывным равномерным распределением на интервале (a, b)
`random_pareto(a,b)`	Вычисление ПСЧ в соответствии с распределением Парето с параметрами $a > 0$, $b > 0$
`random_pareto(a,b,k)`	Вычисление k ПСЧ в соответствии с распределением Парето с параметрами $a > 0$, $b > 0$
`random_rayleigh(b)`	Вычисление ПСЧ в соответствии с распределением Релея с параметром $b > 0$

[1] Вихрь Мерсенна (англ. Mersenne twister, MT) – генератор ПСЧ – разработан в 1997 г., основан на свойствах простых чисел Мерсенна (отсюда название), обеспечивает быструю генерацию высококачественных по критерию случайности ПСЧ и лишен многих недостатков, присущих другим генераторам, таких как малый период, предсказуемость, легко выявляемые статистические закономерности. Существуют, по меньшей мере, два общих варианта алгоритма, различающихся только величиной используемого простого числа Мерсенна, наиболее распространенным из которых является алгоритм MT 19937, период (количество чисел до повторения последовательности) которого составляет $2^{19937} - 1 \approx 4.3 \cdot 10^{6001}$.

Функция	*Описание*
`random_rayleigh(b,k)`	Вычисление k ПСЧ в соответствии с распределением Релея с параметром $b > 0$
`random_laplace(a,b)`	Вычисление ПСЧ в соответствии с распределением Лапласа с параметрами a, $b > 0$
`random_laplace(a,b,k)`	Вычисление k ПСЧ в соответствии с распределением Лапласа с параметрами a, $b > 0$
`random_gumbel(a,b)`	Вычисление ПСЧ в соответствии с распределением Гумбеля с параметрами a, $b > 0$
`random_gumbel(a,b,k)`	Вычисление k ПСЧ в соответствии с распределением Гумбеля с параметрами a, $b > 0$
`random_student_t(n)`	Вычисление ПСЧ в соответствии с распределением Стьюдента с n степенями свободы
`random_student_t(n,k)`	Вычисление k ПСЧ в соответствии с распределением Стьюдента с n степенями свободы
`random_chi2(n)`	Вычисление ПСЧ в соответствии с распределением χ^2 с n степенями свободы
`random_chi2(n,k)`	Вычисление k ПСЧ в соответствии с распределением χ^2 с n степенями свободы
`random_noncentral_chi2(n,ncp)`	Вычисление ПСЧ в соответствии с нецентральным распределением χ^2 с n степенями свободы и параметром нецентральности `ncp` $\geqslant 0$
`random_noncentral_chi2(n,ncp,k)`	Вычисление k ПСЧ в соответствии с нецентральным распределением χ^2 с n степенями свободы и параметром нецентральности `ncp` $\geqslant 0$
`random_f(m,n)`	Вычисление ПСЧ в соответствии с F-распределением с (n, m) степенями свободы
`random_f(m,n,k)`	Вычисление k ПСЧ в соответствии с F-распределением с (n, m) степенями свободы
`random_general_finite_discrete(v)`	Вычисление ПСЧ в соответствии с общим дискретным равномерным распределением с вектором вероятностей
`random_general_finite_discrete(v,k)`	Вычисление k ПСЧ в соответствии с общим дискретным равномерным распределением с вектором вероятностей
`random_binomial(n,p)`	Вычисление ПСЧ в соответствии с биномиальным распределением $Bi(n, p)$ с параметрами $n > 0$ и $p > 0$
`random_binomial(n,p,k)`	Вычисление k ПСЧ в соответствии с биномиальным распределением $Bi(n, p)$ с параметрами $n > 0$ и $p > 0$
`random_poisson(lambda)`	Вычисление ПСЧ в соответствии с распределением Пуассона с параметрами $lambda > 0$

Функция	*Описание*
`random_poisson(lambda,k)`	Вычисление k ПСЧ в соответствии с распределением Пуассона с параметрами $lambda > 0$
`random_bernoulli(p)`	Вычисление ПСЧ в соответствии с распределением Бернулли с параметром $p > 0$
`random_bernoulli(p,k)`	Вычисление k ПСЧ в соответствии с распределением Бернулли с параметром $p > 0$
`random_discrete_uniform(n)`	Вычисление ПСЧ в соответствии с дискретным распределением с параметром $n > 1$
`random_discrete_uniform(n,k)`	Вычисление k ПСЧ в соответствии с дискретным распределением с параметром $n > 1$
`random_hypergeometric(n1,n2,n)`	Вычисление ПСЧ в соответствии с гипергеометрическим распределением с параметрами $n_1 \geqslant 0$, $n_2 \geqslant 0$, $n \geqslant 0$, $0 \leqslant n \leqslant n_1 + n_2$
`random_hypergeometric(n1,n2,n,k)`	Вычисление k ПСЧ в соответствии с гипер геометрическим распределением с параметрами $n_1 \geqslant 0$, $n_2 \geqslant 0$, $n \geqslant 0$, $0 \leqslant n \leqslant n_1 + n_2$
`random_negative_binomial(n,p)`	Вычисление ПСЧ в соответствии с отрицательным биномиальным распределением с параметрами $n > 0$ и $p > 0$
`random_negative_binomial(n,p,k)`	Вычисление k ПСЧ в соответствии с отрицательным биномиальным распределением с параметрами $n > 0$ и $p > 0$

```
(%i24)  load("distrib")$
```

```
(%i25)  random_general_finite_discrete([1,3,1,5], 10);
(%o25)      [4,2,2,4,4,1,4,4,2,4]
```

12.2.6. Случайные перестановки

Оператор	*Описание*
`random_permutation(a)`	Случайная перестановка из элементов списка или множества `a`
`random_perm(n)`	Случайная перестановка из `n` чисел

```
(%i26)  random_permutation ([a, b, c, 1, 2, 3]);
(%o26)      [c, 1, 2, 3, a, b]
```

```
(%i27)  random_permutation ([a, b, c, 1, 2, 3]);
(%o27)      [b, 3, 1, c, a, 2]
```

```
(%i28)  load("combinatorics")$ random_perm(7);
(%o28)      [1,7,2,5,4,6,3]
```

ПРИЛОЖЕНИЯ

П.1. Список сокращений и обозначений

ДУ	дифференциальное уравнение
КА	компьютерная алгебра
НОД	наибольший общий делитель
ОДУ	обыкновенное дифференциальное уравнение
ОС	операционная система
ПКА	пакет компьютерной алгебры
ПСЧ	псеадослучайное число
САВ	система аналитических вычислений
СВ	случайная величина
СЛАУ	система линейных алгебраических уравнений
GUI	графический интерфейс пользователя

П.2. Система Macsyma

Macsyma (Project MAC's SYmbolic MAnipulator) [3, 32] – одна из старейших систем компьютерной алгебры, первая версия которой разрабатывалась начиная с 1968 г. в Массачусетском технологическом институте (МТИ, США) в лаборатории Project MAC. С 1982 г. система распространялась на коммерческой основе, в 1999 г. ее развитие было прекращено.

Macsyma, целиком написанная на языке программирования Lisp (диалект MacLisp), была первой универсальной системой символьных вычислений и одной из первых CAB, основанных на знаниях. Многие из идей, реализованных в Macsyma, впоследствии были заимствованы разработчиками таких систем, как Mathematica, Maple и др. Для своего времени она была одной из самых больших написанных на Lisp программ. Macsyma является предком свободной системы компьютерной алгебры Maxima.

Проект был инициирован в июле 1968 г. К. Энгельманом (C. Engelman), Б.А. Мартином (W.A. Martin, интерфейс пользователя, отображение выражений, арифметика полиномов) и Дж. Мозесом (J. Moses, механизм упрощения выражений, неопределенные интегралы: эвристики / алгоритм Риша). Б. Мартин был руководителем проекта до 1971 г., а Дж. Мозес – следующие десять лет. К. Энгельман и его команда покинули проект в 1969 г. и вернулись в MITRE Corporation. Впоследствии основными участниками разработки математического ядра программы были

Я. Августис (Ya. Avgoustis) – специальные функции; Д. Бартон (D. Barton) – решение нелинейных алгебраических уравнений; Р. Боген (R. Bogen) – специальные функции; Б. Дубик (B. Dubuque) – пределы, базис Гребнера, TriangSys, неопределенные интегралы, степенные ряды, теория чисел, специальные функции, функциональные уравнения, сопоставление с образцом; Р. Фейтман (R. Fateman) – дроби, сопоставление с образцом, длинная арифметика; М. Генезерет (M. Genesereth) – сравнение, база знаний; Дж. Голден (J. Golden) – упрощение выражений, язык, системное программирование; Б. Госпер (B. Gosper) – конечные суммы, специальные функции); Ч. Карни (Ch. Karney) – графики; Дж. Кульп (J. Kulp), Э. Лафферти (E. Lafferty) – обыкновенные дифференциальные уравнения, специальные функции; Ст. Макракис (St. Macrakis) – комплексные числа, системное программирование; Б. Трагер (B. Trager) – алгебраическое интегрирование, факторизация, базисы Гребнера; П. Ван (P. Wang) – факторизация многочленов, пределы, определенные интегралы; Д. Юнь (D. Yun), Г. Захариас (G. Zacharias) – базисы Гребнера; Р. Зиппель (R. Zippel) – степенные ряды, факторизация многочленов, теория чисел, комбинаторика.

Macsyma была написана на языке программирования MacLisp, разработанном в рамках проекта MAC диалекте Lisp, и являлась, в некоторых случаях, ключевым мотиватором для улучшения этого диалекта языка Lisp в области численных расчетов, эффективной компиляции и дизайна самого языка. Программы на языке MacLisp выполнялись в основном на компьютерах PDP-6 и PDP-10 под управлением операционных систем ITS и TOPS 10/20, позже – на компьютере GE-600 под управлением ОС Multics на мейнфреймах фирмы Honeywell и на Lisp-машинах (компьютерах, выполнявших команды языка Lisp на аппаратном уровне). В то время Macsyma была одной из самых больших (если не самой большой) из программ на языке программирования (ЯП) Lisp. Потребности переноса Macsyma на другие аппаратные платформы привели к появлению нескольких новых диалектов Lisp, в частности Franz Lisp.

Долгие годы существовали версии этой системы только для мощных вычислительных машин коллективного пользования (в том числе и для работы в глобальных сетях), таких как IBM, VAX[2] и др., но рост возможностей персональных компьютеров (ПК) привел к созданию версий этой системы и для ПК. Длительность эксплуатации и развития этой системы, а также многочисленные проекты, которые основаны на ней, позволяют

[2]VAX (Virtual Address eXtension) – 32-битная компьютерная архитектура, разработанная в середине 1970-х фирмой DEC как развитие линии миниЭВМ PDP-11 в рамках проекта Star (http://ru.wikipedia.org/wiki/VAX).

считать ее лидером среди CAB. Система включает обширную коллекцию программ, созданных пользователями. Эта коллекция называется библиотекой SHARE.

В настоящее время различные версии системы работают под управлением Linux, MS Windows, Mac OS X и других операционных систем. Основными преимуществами системы являются:

– правильное решение бóльшего числа математических задач, чем иными CAB;

– максимальная пригодность численных и аналитических интеграторов для систем ОДУ, включая методы теории возмущений и численные методы для жестких систем;

– бóльшее, чем в других системах, число операций линейной алгебры (числовых и символических);

– генерация ДУвЧП в произвольных координатах, поиск символьных решений и симметрий Ли в сочетании с возможностью использования системой подсистемы PDEase (см. ниже), предназначенной для конечно-элементного анализа (не)линейных статических, динамических проблем и задач на собственные функции;

– легкость использования "шпаргалки", математического браузера; большое число (более 800) примеров постановки и решения для общих операций; более 2500 гипертекстовых описаний с перекрестными ссылками; более 1000 выполняемых примеров и демонстраций, доступных из браузера, меню, гипертекста и клавиатуры; 600 заготовок команд, которые позволяют создавать ваши собственные команды;

– автозагрузка команд из внешних и пользовательских библиотек;

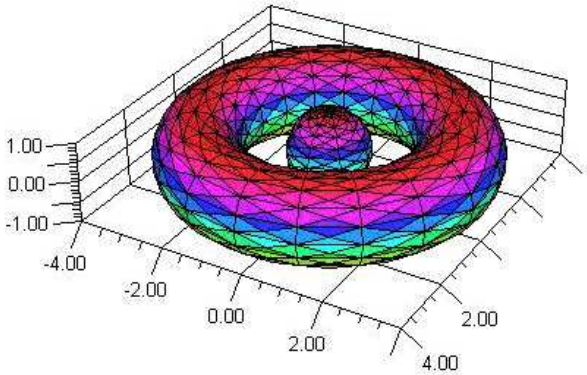
– контекстно-ориентированные меню, выбор средств и памятки;

– прекрасные возможности создания и выполнения документов с формулами, текстом, графиками в виде записных книжек (notebooks);

– доступность редактирования 2D- и 3D-мерных графиков в записных книжках; сотни атрибутов и пять привычных диалогов для контроля точки наблюдения, размера, вырезания, представления, раскраски, анимации; запрос координат с помощью "мыши";

– кроме отображения математических символов и греческих букв форматирование больших выражений для облегчения и убыстрения зрительной интерпретации; дополнительное форматированное эхо математического ввода;

– обеспечение полной текстовой обработки, включая выбор шрифтов для каждого символа, разделение на параграфы, форматы страниц с ма-



тематическими символами и греческими буквами, включенными в текст;

– возможность организации пользователем гипертекстовых связей между записными книжками системы; диалог "Navigate" для показа таблицы ссылок или коротких аннотаций каждой секции в записной книжке для перехода к требуемой секции;

– возможность импортирования, экспортирования, просмотра, редактирования, анализа и графического отображения числовых массивов информации.

К настоящему время разработаны реализации системы Macsyma для IBM-совместимых персональных компьютеров (80386, 80486, Pentium) и

рабочих станций (HP9000 RISC series, DECstation, IBM RS/6000, SGI IRIS, в среде SPARC Solaris, SPARC Sun OS).

В системе доступны следующие операции:

– над множествами, целыми и рациональными числами, числами с плавающей точкой, комплексными числами и выражениями, системными и пользовательскими, символьными и числовыми специальными функциями;

– алгебры и тригонометрии (факторизация и раскрытие скобок, упрощение рациональных выражений, упрощение логарифмов и радикалов, разложение в ряды гармоник, точное и приближенное решение алгебраических уравнений и систем таких уравнений);

– анализа (дифференцирование, интегрирование, обратное преобразование Лапласа, специальные функции, интегральные уравнения и преобразования, формула Тейлора, решение ОДУ первого и второго порядка, первые интегралы ОДУ, теория возмущений для ОДУ, симметрии ДУ,численный анализ для ОДУ, ДУвЧП и др.);

– векторного и тензорного анализа;

– построения 2D- и 3D-графиков (обычные, параметрические, неявные, цветные, векторные поля);

– связи с другими языками (Fortran, C, RatFor[3], TEX) и др.

Система Macsyma имеет программный инструмент–спутник PDEase [30], который считывает символьное представление дифференциального уравнения в частных производных (ДУвЧП), построенное Macsyma, и решает его численно методом конечных элементов.

Клоны системы Macsyma. В настоящее время существуют, по крайней мере, 8 версий Macsyma-подобных систем:

1°. Macsyma (1983) фирмы Symbolics, Inc. для Windows 95, 98, 2000 и XP. Сайт: www.symbolics-dks.com.

2°. Macsyma (1992) фирмы Macsyma, Inc. Сайты: www.macsyma.com/macsyma.html – Macsyma, www.macsyma.com/pdease.html – PDEase.

3°. DOE-Macsyma (∼1984) фирмы Paradigm Associates для ЭВМ VAX. Сайт: ftp.gwdg.de/pub/math/symbolic_soft/DOE-Macsyma.

4°. Maxima (1998) профессора У. Шелтера [5, 22] (версия DOE-Macsyma).

5°. Vaxima (1980) профессора Р. Фейтмена [24].

6°. Aljabr (1991) фирмы Fort Pond Research для ЭВМ типа Macintosh.

[3] RatFor (Rational Fortran) – язык программирования, реализованный как препроцессор для языка Fortran 66. RatFor предоставлял современные структуры управления, недоступные в Fortran 66.

7°. Punimax (∼1994) Б. Хайбле (B. Haible).

8°. ParaMax (или ParaMacs, 1991) фирмы Paradigm Associates Inc.

Основные и многие дополнительные характеристики идентичны для всех версий. Системы 1, 2 написаны на Zeta Lisp, 3, 4 – на Common Lisp, 5, 6 – на Franz Lisp, 7 – на CLisp.

CAB Aljabr построена из кода системы Macsyma, выполняется на платформе Mac и является универсальной системой аналитических вычислений с широкими возможностями символьной (факторизация полиномов, дифференцирование, интегрирование, пределы, ряды Тейлора, Лорана и Пуассона, решение линейных алгебраических и полиномиальных алгебраических и дифференциальных уравнений, преобразования) и численной математики. Интерфейс системы подобен другим математическим системам МТИ. Aljabr обладает языком программирования высокого уровня и возможностями построения графиков.

Punimax – это версия системы Maxima, являющаяся надстройкой над CLisp[4] – одной из самых маленьких Lisp-систем, которые могут виртуально выполняться на всех платформах.

Ряд разработок-последователей системы Macsyma после 1982 г., когда произошла коммерциализация этой CAB, развивался на доступных для пользователей исходных кодах системы, созданных в МТИ.

DOE-Macsyma – это наиболее мощная CAB. Самая последняя версия, работающая на Lisp-машине фирмы GigaMos, отличается от всех других версий системы Macsyma практически во всем и имеет продвинутые возможности, введенные для поддержки объемного Macsyma-программирования и генерирования Fortran-кода. Имеются версии системы для различных реализаций языка Lisp. Версия Zeta Lisp работает на платформах Lisp-машин GigaMos, Symbolics и TI Explorer; Nil – на ЭВМ VAX под управлением операционной системы VMS; Franz Lisp – на Unix-машинах, включая компьютеры VAX и Sun. Из названия ясно, что DOE-Macsyma доступна лишь для правительственных органов США и их контрагентов. DOE-Macsyma поставляется с собственным компилятором Lisp-программ, называемым NIL (New Implementation of Lisp).

Система ParaMax (или ParaMacs, 1991) основана на Common Lisp, который был перенесен с оригинального источника MacLisp. ParaMax доступен для платформах VAX/VMS на основе DEC VAXLisp 2.2, на Sun-3 и Sun-4 на основе Allegro Common Lisp фирмы Franz Inc. Доступна версия для SGI, разрабатывались релизы для Mac/PC. ParaMax характеризуется продвинутыми разделяемой библиотекой функций, функционалом для

[4] CLisp – это реализация Common Lisp, авторы B. Haible и M. Stoll, Univ. of Karlsruhe.

решения ОДУ и анимированной графикой

Другой системой, аналогичной ParaMax, является Vaxima. Vaxima – реализация системы Macsyma, использующая VAX /UNIX Franz Lisp и работающая на некоторых станциях с установленным языком Franz Lisp.

Если изначально система Macsyma развивалась в условиях университетских исследований, то после 1982 г. разработки начали интенсивно коммерциализироваться, появился ряд фирм (Lisp Machines Inc., Symbolics, Inc., Macsyma Inc. и др.), создающих и продающих как различные варианты DOE-Macsyma, так и свои разработки, включая технические. К концу 1980-х гг. система Macsyma была портирована на ПК, и в тот момент аналитические выкладки в этой системе выполнялись эффективнее, чем на конкурирующих CAB SMP, Maple и Mathematica. Но недостаточная проработка поддержки численных расчетов и пользовательского интерфейса, в т.ч., вследствие недостаточного количества программистов, например, в фирме Macsyma Inc., и нехватка финансовых средств на новые разработки привели к тому, что рассматриваемая система к середине 1990-х гг. окончательно проиграла борьбу за рынок.

Список литературы

1. *Акритас А.* Основы компьютерной алгебры с приложениями. – М.: Мир, 1994. – 544 с.

2. *Берков Н.А.* Применение пакета Maxima: математический практикум / Моск. гос. ин-т управл. – М., 2008. – 89 с.

3. *Дэвенпорт Дж., Сирэ И., Турнье Э.* Компьютерная алгебра. – М.: Мир, 1991. – 352 с.

4. *Есаян А.Р., Чубариков В.Н., Добровольский Н.М., Якушин А.В.* Программирование в Maxima: учеб. пособие. – Тула: Изд-во ТГПУ им. Л.Н. Толстого, 2012. – 351 с.

5. *Ильина В.А., Силаев П.К.* Система аналитических вычислений Maxima для физиков-теоретиков. – М.: Изд-во МГУ, 2007. – 113 с.

6. *Карманов В.Г.* Математическое программирование: учеб. пособие. – 5-е изд., стереотип. – М.: Физматлит, 2004. – 264 с.

7. Компьютерная алгебра: Символьные и алгебраические вычисления: пер. с англ. / под ред. Б. Бухбергера, Дж. Коллинза, Р. Лооса. – М.: Мир, 1986. – 392 с.

8. *Ляшко И.И., Боярчук А.К., Гай Я.Г., Головач Г.П.* Математический анализ в примерах и задачах. Ч. 2. Ряды, функции нескольких переменных, кратные и криволинейные интегралы. – Киев: Вища школа, 1977. – 672 с.

9. *Маевский Е.В., Ягодовский П.В.* Компьютерная математика. Высшая математика в СКМ Maxima.- - Ч. 1. Введение / Моск. финанс. ун-т. – М., 2013. – 150 с.

10. *Малакаев М.С., Секаева Л.Р., Тюленева О.Н.* Основы работы с системой компьютерной алгебры Maxima: учеб.-метод. пособие. – Казань: Из-во Казанского унив-та, 2012. – 57 с.

11. *Панкратьев Е.В.* Элементы компьютерной алгебры: учеб. пособие. – М.: ИНТУИТ, 2007. – 247 с.

12. *Полосков И.Е.* Системы аналитических вычислений. Общие сведения, структура и приложения [Электронный ресурс]: учеб. пособие / Перм. гос. нац. исслед. ун-т. – Пермь, 2013. – 660 с.

13. *Роганов Е. А., Тихомиров Н. Б., Шелехов А. М.* Математика и информатика для юристов: учебник / Моск. гос. ин-т управл. – М., 2005. – vi, 364 с.

14. Современные технологии в процессе обучения: монография / под ред. проф. В.В. Миронова. – Рязань: Book Jet, 2017. – 160 с.

15. *Стахин Н.А.* Основы работы с системой аналитических (символьных) вычислений Maxima (ПО для решения задач аналитических (символьных) вычислений): учеб. пособие. – М.: Федеральное агенство по образованию, 2008. – 86 с.

16. *Хинчин А.Я.* Цепные дроби. – М.: ГИФМЛ, 1960. – 112 с.

17. *Чичкарёв Е.А.* Компьютерная математика с Maxima: руководство для школьников и студентов. – М.: ALT Linux, 2012. – 384 с.

18. *Юдин С.В.* Математика в экономике: учеб. пособие. – М.: Изд-во РГТЭУ, 2009. – 228 с.

19. *Cohen J.S.* Computer algebra and symbolic computation. Elementary Algorithms. – Natick, MA: A K Patrick, 2002. – 340 p.

20. *Cohen J.S.* Computer algebra and symbolic computation. Mathematical methods. – Natick, MA: A K Patrick, 2003. – 468 p.

21. *Connan G., Grognet S.* Guide du Calcul avec les logiciels libres: XCAS, Scilab, Bc, Gp, GnuPlot, Maxima, MuPAD. – Paris: Dunod, 2008. – xiii, 303 p.

22. *de Souza P.N., Fateman R.J., Moses J., Yapp C.* The Maxima Book. – 155 p. – URL: https://www.researchgate.net/profile/Richard_Fateman/publication/242402691_

The_Maxima_Book/links/02e7e53c7ed51d8a1b000000/The-Maxima-Book.pdf (дата просмотра: 23.09.2019)

23. *Heck A.* Introduction to Maple. – 3rd ed. – New York: Springer, 2003. – xvi, 828 p.

24. *Hohti A.* An introduction to Vaxima. – Abo akademi, 1987. – 118 p.

25. *Hammock M.R., Mixon J.W.* Microeconomic theory and computation. – New York: Springer, 2013. – xix, 385 p.

26. *Klee V., Minty G.J.* How good is the simplex algorithm? // Inequalities III: Proc. of the 3rd Symp. on Inequalities (Los Angeles, Sept. 1–9, 1969) / O. Shisha (ed.). – New York, London: Academic Press, 1972. – P. 159–175.

27. *Leydold J., Petry M.* Introduction to Maxima for Economics. – WU Wien: Institute for Statistics and Mathematics, 2011. – iv, 115 p.

28. *Liska R., Drska L.* FIDE: A REDUCE package for automation of finite difference method for solving PDE // Proc. of the Intern. Symposium on Symbolic and Algebraic Computation (ISSAC'90) / S. Watanabe, M. Nagata (eds.). – New York: Addison Wesley, ACM Press, 1990. – P. 169–176.

29. *Liska R., Drska L., Limpouch J. et al.* Computer algebra: Algorithms, systems and applications. – 1999. – 169 p. – URL: www-troja.fjfi.cvut.cz/∼liska/ca/ca.ps.gz

30. Macsyma and PDEase scientific notebook interface reference. – Arlington, MA: Macsyma Inc., 1998. – 129 p.

31. *Mangano S.* Mathematica cookbook. – Cambridge: O'Reilly, 2010. – xxiv, 800 p.

32. The MACSYMA Manual. – Cambridge: MIT, 1977. – 328 p.

33. *Mignotte M.* Mathematics for computer algebra. – New York, Berlin: Springer, 1992. – 360 p.

34. *Mixon W.* Introduction to mathematical economics. – 2018. – 362 p. – URL: `https://ww w.researchgate.net/publication/327076435_Introduction_to_Mathemati cal_Economics` (дата просмотра: 11.06.2019)

35. *Stoutemyer D.R.* Analytically solving integral equations by using computer algebra // ACM Transactions on Mathematical Software. – 1977. – Vol. 3, № 2. – P. 128–146.

36. http://maxima.sourceforge.net/docs/manual/maxima.html

37. https://people.richland.edu/james/spring15/m122/projects/draw.html

38. https://people.richland.edu/james/fall15/m221/projects/project11.html

39. http://www.scotchildress.com/wxmaxima/3DPlotting/3dplotting.html

40. https://ru.wikipedia.org/wiki/Sage

41. https://en.wikipedia.org/wiki/GNU_TeXmacs

42. http://symaxx.sourceforge.net/doc/Manual.pdf

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Учебное издание

Полосков Игорь Егорович

Система аналитических вычислений Maxima. Описание и примеры использования

Учебное пособие

Редактор *Л. Г. Подорова*
Корректор *Л. И. Иванова*
Компьютерная верстка: *И. Е. Полосков*

Объем данных 5,3 Мб
Подписано к использованию 26.08.2020.

Размещено в открытом доступе
на сайте *www.psu.ru*
в разделе НАУКА / Электронные публикации
и в электронной мультимедийной библиотеке ELiS

Издательский центр
Пермского государственного
национального исследовательского университета
Пермь, 614990, ул. Букирева, 15