В серии: Библиотека ALT Linux

Компьютерная математика с Maxima

Руководство для школьников и студентов

Е.А. Чичкарёв

Москва
ALT Linux
2009

УДК
ББК

Компьютерная математика с Maxima: Руководство для школьников и студентов / Е. А .Чичкарёв — М. : ALT Linux, 2009. — 233 с. : ил. — (Библиотека ALT Linux).

ISBN

Текст аннотации ...
Сайт книги: `http://books.altlinux.ru/altlibrary/`

Данная книга посвящена решению различных математических задач с использованием пакета Maxima

УДК
ББК

По вопросам приобретения обращаться:

Данный документ является черновиком готовящейся к публикации книги «Компьютерная математика с Maxima: Руководство для школьников и студентов». Поскольку предлагаемый вариант не является окончательным, использование документа разрешается в личных целях; цитирование и распространение текста настоятельно не рекомендуется,

За ходом работы над подготовкой окончательной версии документа можно следить на странице книги по адресу `http://www.altlinux.org/Books:Maxima`. Пожелания и предложения, а также найденные недоработки просим присылать по адресу `books@altlinux.org`.

Книга содержит следующий текст, помещаемый на первую страницу обложки: «В серии “Библиотека ALT Linux”». Название: «Компьютерная математика с Maxima: Руководство для школьников и студентов». Книга не содержит неизменяемых разделов. Авторы разделов указаны в заголовках соответствующих разделов. ALT Linux — торговая марка компании ALT Linux. Linux — торговая марка Линуса Торвальдса. Прочие встречающиеся названия могут являться торговыми марками соответствующих владельцев.

ISBN

© Чичкарёв Е.А., 2009
© ALT Linux 2009

Оглавление

Глава 1

Введение

Данная книга посвящена открытым программным средствам, позволяющим провести весь цикл разработки какой-либо математической модели: от поиска и просмотра необходимой литературы до непосредственного решения задачи (аналитического и/или численного) и подготовки отчета или статьи к печати. В ней предпринята попытка объяснить, что система аналитических вычислений Maxima и (если необходимо) вычислительная среда Octave — хороший выбор для проведения любой учебной задачи или серьезного исследования, где требуется математика: от курсовой работы до научной или инженерной разработки высокого класса. С помощью этих пакетов проще готовить и выполнять задания, устраивать демонстрации и гораздо быстрее решать исследовательские и инженерные задачи.

Под системами компьютерной математики (СКМ) понимают программное обеспечение, которое позволяет не только выполнять численные расчеты на компьютере, но и производить аналитические (символьные) преобразования различных математических и графических объектов. В настоящее время компьютерные программы этого класса (проприетарные — Maple, Matbematica, MATLAB, MathCad и др. — или с открытым кодом) находят самое широкое применение в научных исследованиях, становятся одним из обязательных компонентов компьютерных технологий, используемых в образовании.

Эти системы имеют дружественный интерфейс, реализуют множество стандартных и специальных математических операций, снабжены мощными графическими средствами и обладают собственными языками программирования. Все это предоставляет широкие возможности для эффективной работы специалистов разных профилей, о чем говорит активное применение математических пакетов в научных исследованиях и преподавании.

Для школьников СКМ являются незаменимым помощником в изучении математики, физики, информатики, освобождая их от рутинных расчетов и сосредотачивая их внимание на сущности метода решения той или иной задачи. Применение СКМ позволяет решать целый спектр новых трудоемких, но интересных задач: от упрощения громоздких алгебраических выражений, аналитического решения уравнений и систем с параметрами, графических построений до анимации графиков и пошаговой визуализации самого процесса решения. Учащимся предоставляется возможность выполнять более содержательные задания и получать наглядные результаты. Это способствует закреплению знаний и умений, приобретенных ими при изучении других школьных дисциплин, помогает в полной мере проявлять свои творческие и исследовательские способности.

Для студентов СКМ — удобное средство решения всевозможных задач, связанных с символьными преобразованиями (математический анализ, высшая математика, линейная алгебра и аналитическая геометрия и т. п.), а также средство решения задач моделирования статических (описываемых алгебраическими уравнениями) и динамических (описываемых дифференциальными уравнениями) систем. Кроме того, добротная СКМ — средство создания графических иллюстраций и документов, содержащих математичекие формулы и выкладки. В настоящее время для проведения расчетов по всевозможным техническим дисциплинам студентами-нематематиками широко используется пакет MatCad, в основе которого лежит ядро Maple. При некотором навыке и наличии документации связка Maxima+TexMacs или ядро Maxima+интерфейс wxMaxima могут стать вполне разумной заменой MatCad в Unix-среде.

Кроме того, наличие универсального интерфейса в виде TexMacs или Emacs позволяет объединять в одном документе расчеты, выполненные в Maxima, Octave, Axiom и т. п.

Для научных работников и инженеров СКМ — незаменимое средство анализа постановки всевозможных задач моделирования. Все широко известные математические пакеты: Maple, Matlab, Matematica — позволяют проводить как символьные вычисления, так и использовать численные методы. В настоящее время такие системы являются одним из основных вычислительных инструментов компьютерного моделирования в реальном времени и находят применение в различных областях науки. Они открывают также новые возможности для преподавания многих учебных дисциплин, таких как алгебра и геометрия, физика и информатика, экономика и статистика, экология. Применение СКМ существенно повышает производительность труда научного работника, преподавателя ВУЗа, учителя.

Конечным продуктом исследования выступают публикации, подготовка, распространение н использование которых в настоящее зремя требуют квалифицированного применения компьютера. Это касается редактирования текста, изготовления графических материалов, ведения библиографии, размещения электронных версий в Интернете, поиска статей и их просмотра. Де-факто стандартными системами подготовки научно-технических публикаций сейчас являются различные реализации пакета TeX и текстовый редактор Word. Кроме того, необходимы минимальные знания о стандартных форматах файлов, конверторах, программах и утилитах, используемых при подготовке публикаций.

Глава 2

Возникновение и развитие систем компьютерной математики

2.1 Определение систем компьютерной алгебры

История математики насчитывает около трех тысячелетий и условно может быть разделена на несколько периодов. Первый — становление и развитие понятия числа, решение простейших геометрических задач. Второй период связан с появлением «Начал» Евклида и утверждением хорошо знакомого нам способа доказательства математических утверждений с помощью цепочек логических умозаключений. Следующий этап берет свое начало с развития дифференциального и интегрального исчисления. Наконец, последний период сопровождается появлением и распространением понятий и методов теории множеств и математической логики, на прочном фундаменте которых возвышается все здание современной математики. Мы живем во время начала нового периода развития математики, который связан с изобретением и применением компьютеров. Прежде всего, компьютер предоставил возможность производить сложнейшие численные расчеты для решения тех задач, которые невозможно (по крайней мере, на данный момент) решить аналитически. Появилось так называемое «компьютерное моделирование» — целая отрасль прикладной математики, в которой с помощью самых современных вычислительных средств изучается поведение многих сложных экономических, социальных, экологических и других динамических систем.

Изучение математики дает в распоряжение будущего инженера, экономиста, научного работника не только определенную сумму знаний, но и развивает в нем способность ставить, исследовать и решать самые разнообразные задачи. Иными словами, математика развивает мышление будущего специалиста и закладывает прочный понятийный фундамент для освоения многих специальных дисциплин. Кроме того, именно с ее помощью лучше всего развиваются способности логического мышления, концентрации внимания, аккуратности и усидчивости.

Компьютерная алгебра — область математики, лежащая на стыке алгебры и вычислительных методов. Для нее, как и для любой области, находящейся на стыке различных наук, трудно определить четкие границы. Часто говорят, что к компьютерной алгебре относятся вопросы слишком алгебраические, чтобы содержаться в учебниках по вычислительной математике, и слишком вычислительные, чтобы содержаться в учебниках по алгебре. При этом ответ на вопрос о том, относится ли конкретная задача к компьютерной алгебре, часто зависит от склонностей специалиста.

2.1.1 Недостатки численных расчетов

Большинство первых систем компьютерной математики (Eureka, Mercury, Excel, Lotus-123, Mathcad для MS-DOS, PC MATLAB и др.) предназначались для численных расчетов. Они как бы превращали компьютер в большой программируемый калькулятор, способный быстро и автоматически (по введенной программе) выполнять арифметические и логические операции над числами или массивами чисел. Их результат всегда конкретен: это или число, или набор чисел, представляющих таблицы, матрицы или точки графиков. Разумеется, компьютер позволяет выполнять такие вы-

числения с немыслимой ранее скоростью, педантичностью и даже точностью, выводя результаты в виде хорошо оформленных таблиц или графиков.

Однако результаты вычислений редко бывают абсолютно точными в математическом смысле: как правило, при операциях с вещественными числами происходит их округление, обусловленное принципиальным ограничением разрядной сетки компьютера при хранении чисел в памяти. Реализация большинства численных методов (например, решения нелинейных или дифференциальных уравнений) также базируется на заведомо приближенных алгоритмах. Часто из-за накопления погрешностей эти методы теряют вычислительную устойчивость и расходятся, давая неверные решения или даже ведя к полному краху работы вычислительной системы — вплоть до злополучного «зависания».

Условия появления ошибок и сбоев не всегда известны — их оценка довольно сложна в теоретическом отношении и трудоемка на практике. Поэтому рядовой пользователь, сталкиваясь с такой ситуацией, зачастую становится в тупик или, что намного хуже, неверно истолковывает явно ошибочные результаты вычислений, «любезно» предоставленные ему компьютером. Трудно подсчитать, сколько «открытий» на компьютере было отвергнуто из-за того, что наблюдаемые колебания, выбросы на графиках или асимптоты ошибочно вычисленных функций неверно истолковывались как новые физические закономерности моделируемых устройств и систем, тогда как на деле были лишь грубыми погрешностями численных методов решения вычислительных задач.

Многие ученые справедливо критиковали численные математические системы и программы реализации численных методов за частный характер получаемых с их помощью результатов. Они не давали возможности получить общие формулы, описывающие решение задач. Как правило, из результатов численных вычислений невозможно было сделать какие-либо общие теоретические, а подчас и практические выводы. Поэтому прежде чем использовать такие системы в реализации серьезных научных проектов, приходилось прибегать к дорогой и недостаточно оперативной помощи математиков-аналитиков. Именно они решали нужные задачи в аналитическом виде и предлагали более или менее приемлемые методы их численного решения на компьютерах.

2.1.2 Отличия символьных вычислений от численных

Термин «компьютерная алгебра» возник как синоним терминов «символьные вычисления», «аналитические вычисления», «аналитические преобразования» и т. д. Даже в настоящее время этот термин на французском языке дословно означает «формальные вычисления».

В чем основные отличия символьных вычислений от численных и почему возник термин «компьютерная алгебра»?

Когда мы говорим о вычислительных методах, то считаем, что все вычисления выполняются в поле вещественных или комплексных чисел. В действительности же всякая программа для ЭВМ имеет дело только с конечным набором рациональных чисел, поскольку только такие числа представляются в компьютере. Для записи целого числа отводится обычно 16 или 32 двоичных символа (бита), для вещественного — 32 или 64 бита. Это множество не замкнуто относительно арифметических операций, что может выражаться в различных переполнениях (например, при умножении достаточно больших чисел или при делении на маленькое число). Еще более существенной особенностью вычислительной математики является то, что арифметические операции над этими числами, выполняемые компьютером, отличаются от арифметических операций в поле рациональных чисел. Особенностью компьютерных вычислений является неизбежное наличие погрешности или конечная точность вычислений. Каждую задачу требуется решить с использованием имеющихся ресурсов ЭВМ за обозримое время с заданной точностью, поэтому оценка погрешности — важная задача вычислительной математики.

Решение проблемы точности вычислений и конечности получаемых численных результатов в определенной степени дается развитием систем компьютерной алгебры. Системы компьютерной алгебры, осуществляющие аналитические вычисления, широко используют множество рациональных чисел. Компьютерные операции над рациональными числами совпадают с соответствующими операциями в поле рациональных чисел. Кроме того, ограничения на допустимые размеры числа (количество знаков в его записи) позволяет пользоваться практически любыми рациональными числами, операции над которыми выполняются за приемлемое время.

В компьютерной алгебре вещественные и комплексные числа практически не применяются, зато широко используется алгебраические числа. Алгебраическое число задается своим минимальным многочленом, а иногда для его задания требуется указать интервал на прямой или область в комплексной плоскости, где содержится единственный корень данного многочлена. Многочлены играют в символьных вычислениях исключительно важную роль. На использовании полиномиальной арифметики основаны теоретические методы аналитической механики, они применяются во многих областях математики, физики и других наук. Кроме того, в компьютерной алгебре рассматриваются такие объекты, как дифференциальные поля (функциональные поля), допускающие показательные, логарифмические, тригонометрические функции, матричные кольца (элементы матрицы принадлежат кольцам достаточно общего вида) и другие. Даже при арифметических операциях над такими объектами происходит разбухание информации, и для записи промежуточных результатов вычислений требуется значительный объем памяти ЭВМ.

В научных исследованиях и технических расчетах специалистам приходится гораздо больше заниматься преобразованиями формул, чем собственно численным счетом. Тем не менее, с появлением ЭВМ основное внимание уделялось автоматизации численных вычислений, хотя ЭВМ начали применяться для решения таких задач символьных преобразований, как, например, символьное дифференцирование, еще в 50-х годах прошлого века. Активная разработка систем компьютерной алгебры началась в конце 60-х годов. С тех пор создано значительное количество разнообразных систем, получивших различную степень распространения: некоторые продолжают развиваться, другие отмирают, и постоянно появляются новые.

2.2 Классификация, структура и возможности систем компьютерной математики

2.2.1 Классификация систем компьютерной математики

В настоящее время системы компьютерной математики (СКМ) можно разделить на семь основных классов:

- системы для численных расчетов;
- табличные процессоры;
- матричные системы;
- системы для статистических расчетов;
- системы для специальных расчетов;
- системы для аналитических расчетов (компьютерной алгебры);
- универсальные системы.

Каждая система компьютерной математики имеет нюансы в своей архитектуре или структуре. Тем не менее можно прийти к выводу, что у современных универсальных СКМ следующая типовая структура:

- Центральное место занимает ядро системы - коды множества заранее откомпилированных функций и процедур, обеспечивающих достаточно представительный набор встроенных функций и операторов системы.
- Интерфейс дает пользователю возможность обращаться к ядру со своими запросами и получать результат решения на экране дисплея. Интерфейс современных СКМ основан на средствах популярных операционных систем Windows 95/98/NT и обеспечивает присущие им удобства работы.
- Функции и процедуры, включенные в ядро, выполняются предельно быстро. Поэтому объем ядра ограничивают, но к нему добавляют библиотеки более редких процедур и функций.

- Кардинальное расширение возможностей систем и их адаптация к решаемым конкретными пользователями задачам достигаются за счет пакетов расширения систем. Эти пакеты (нередко и библиотеки) пишутся на собственном языке программирования той или иной СКМ, что делает возможным их подготовку обычными пользователями.

- Ядро, библиотеки, пакеты расширения и справочная система современных СКМ аккумулируют знания в области математики, накопленные за тысячелетия ее развития.

Возрастающий интерес к алгебраическим алгоритмам возник в результате осознания центральной роли алгоритмов в информатике. Их легко описать на формальном и строгом языке и с их помощью обеспечить решение задач, давно известных и изучавшихся на протяжении веков. В то время как традиционная алгебра имеет дело с конструктивными методами, компьютерная алгебра интересуется еще и эффективностью, реализацией, а также аппаратными и программными аспектами таких алгоритмов. Оказалось, что при принятии решения об эффективности и определении производительности алгебраических методов требуются многие другие средства, например теория рекурсивных функций, математическая логика, анализ и комбинаторика. В начальный период применения вычислительных машин в символьной алгебре быстро стало очевидным, что непосредственные методы из учебников часто оказывались весьма неэффективными. Вместо обращения к методам численной аппроксимации компьютерная алгебра систематически изучает источники неэффективности и ведет поиск иных алгебраических методов для улучшения или даже замены таких алгоритмов.

2.2.2 Задачи систем компьютерной алгебры

Первые ЭВМ изначально создавались для того, чтобы проводить сложные расчеты, на которые человек тратил очень много времени. Следующим шагом развития ЭВМ стали ПК. Эти машины могут проводить вычисления разной сложности (от самых простых до самых сложных), и такая особенность стала использоваться в разных областях знаний. Развитие компьютерных математических систем привело к появлению отдельного класса программ, который получил названия Системы Компьютерной Алгебры (CAS).

Главная задача CAS — обработка математических выражений в символьной форме. Символьные операции обычно включают в себя: вычисление символьных либо числовых значений для выражений, преобразование, изменение формы выражений, нахождение производной одной или нескольких переменных, решение линейных и нелинейных уравнений, решение дифференциальных уравнений, вычисление пределов, вычисление определенных и неопределенных интегралов, работу с множествами, вычисление и работу с матрицами. В дополнение к перечисленному, большинство CAS поддерживают разнообразные численные операции: расчет значений выражений при определенных значениях переменных, построение графиков на плоскости и в пространстве. Большинство CAS включают в себя высокоуровневый язык программирования, который позволяет реализовать свои собственные алгоритмы. Наука, которая изучает алгоритмы, применяемые в CAS, называется компьютерной алгеброй.

2.2.3 Место компьютерной алгебры в информатике

Компьютерная алгебра — часть информатики, которая занимается разработкой, анализом, реализацией и применением алгебраических алгоритмов. От других алгоритмов алгебраические алгоритмы отличаются наличием простых формальных описаний, существованием доказательств правильности и асимптотических границ времени выполнения, которые можно получить на основе хорошо развитой математической теории. Кроме того, алгебраические объекты можно точно представить в памяти вычислительной машины, благодаря чему алгебраические преобразования могут быть выполнены без потери точности и значимости. Обычно алгебраические алгоритмы реализуются в программных системах, допускающих ввод и вывод информации в символьных алгебраических обозначениях. Благодаря всему этому специалисты, работающие в информатике, математике и в прикладных областях, проявляют все больший интерес к компьютерной алгебре. Многие алгоритмы компьютерной алгебры можно рассматривать как получисленные (в смысле Кнута).

2.2.4 Взаимосвязь систем компьютерной алгебры и традиционных математических дисциплин

Отделить компьютерную алгебру от таких математических дисциплин, как алгебра, анализ или численный анализ, нелегко.

Системы компьютерной алгебры обычно включают алгоритмы для интегрирования, вычисления элементарных трансцендентных функций, решения дифференциальных уравнений и т. п. Особенность упомянутых алгоритмов заключается в следующем:

- они оперируют термами и формулами и вырабатывают выходную информацию в символьной форме;
- решение достигается посредством некоторого вида алгебраизации задачи (например, производную от полинома можно определить чисто комбинаторным образом);
- существуют методы точного представления величин, определяемых через пределы и имеющих бесконечное численное представление.

Часто формулы, получаемые в качестве выходной информации при выполнении алгоритмов компьютерной алгебры, используются затем как входная информация в численных процедурах. Например, при интегрировании рациональных функций от нескольких переменных первое и, возможно, второе интегрирование выполняются в символьном виде, а остальные — численно. Численные процедуры используют арифметику конечной точности и основываются на теории аппроксимации. Например, численная процедура нахождения корней не всегда может отделить все корни, так как работает с числами конечной точности; она отделяет лишь кластеры корней, диаметр которых зависит от заданной точности представления чисел и многих других параметров.

В принципе, желательно и возможно описывать численные алгоритмы с той же строгостью, как и алгебраические, однако требуемая при этом детализация гораздо выше, а сходство с математической постановкой задачи менее прозрачно. С другой стороны, при использовании некоторого алгебраического алгоритма точность оплачивается большими — в общем случае существенно — временем выполнения и необходимым объемом памяти, чем для его численного аналога.

Тем не менее можно привести много примеров таких задач, в которых аппроксимация не имеет большого смысла. Поэтому методы символьных вычислений и чисто численные алгоритмы обычно дополняют друг друга. Современные системы компьютерной алгебры обязательно включают тот или иной набор стандартных численных алгоритмов. Соременные системы, рассчитанные на использование в первую очередь численных расчётов (MatLab, его клоны и т. п.), всегда включают более или менее полный набор функций, осуществляющих символьные преобразования.

2.2.5 Возможности повышения эффективности решениия математических и вычислительных задач

Реализация на ЭВМ символьной математики открыла принципиально новые возможности использования вычислительных машин в естественнонаучных и прикладных исследованиях. Сейчас уже трудно указать область естественных наук, где методы аналитических вычислений на ЭВМ не нашли бы плодотворного применения. Характерной особенностью проблематики символьных преобразований является сочетание весьма тонких математических и алгоритмических методов с самыми современными методами программирования, эффективно реализующими нечисленную математику в рамках программных систем аналитических вычислений. К числу последних относятся, например, такие популярные системы, как MACSYMA, REDUCE, АНАЛИТИК и др.

Хорошо известно, что аналитические преобразования являются неотъемлемой частью научных исследований, и зачастую на их выполнение затрачивается больше труда, чем на остальную часть исследований, а для реализации специализированных методов (например методов современного группового анализа дифференциальных уравнений) особенное значение имеет точность аналитических выражений. Однако ручные вычисления по любому из подобных методов требуют непомерно больших затрат времени. Именно здесь и помогают методы компьютерной алгебры (КА) и соответствующие программные системы, являющиеся практически единственным средством решения

таких задач, требующих больших затрат ручных вычислений и очень чувствительных к потере точности при численном счете на ПК. Благодаря методам и алгоритмам аналитических вычислений современный компьютер становится уже не столько вычислительной, сколько общематематической машиной. ПК под силу реализовать интегрирование и дифференцирование символьных выражений, перестановки и перегруппировки членов, подстановки в выражения с последующим их преобразованием, решать дифференциальные уравнения и т. д. Аналитические вычисления (АВ) являются составной частью теоретической информатики, которая занимается разработкой, анализом, реализацией и применением алгебраических алгоритмов. Цели АВ лежат в области искусственного интеллекта, несмотря на то, что методы все более и более удаляются от нее. Кроме того, используемые алгоритмы вводят в действие все менее элементарные математические средства. Таким образом, АВ как самостоятельная дисциплина на самом деле лежит на стыке нескольких областей: информатики, искусственного интеллекта, современной математики (использующей нетрадиционные методы), что одновременно обогащает ее и делает более трудной в исследовательском плане. Наименование этой научной дисциплины длительное время колебалось и, наконец, стабилизировалось как «Calcul formel» во французском языке, «Computer algebra» — в английском языке и «аналитические вычисления» или «компьютерная алгебра» — в русском.

Наиболее интуитивная цель АВ заключается в манипуляции с формулами. Математическая формула, описанная на одном из обычных языков программирования (Фортран, Паскаль, Бейсик, ...), предназначена только для численных расчетов, когда переменным и параметрам присвоены численные значения. В языке, допускающем АВ, для этой формулы также можно получить численное значение, но, кроме того, она может стать объектом формальных преобразований: дифференцирования, разложения в ряд, различных других разложений и даже интегрирования.

Интеллектуальность разработанных на сегодняшний день систем аналитических вычислений (САВ) определяется их использованием для организации баз знаний по математическим методам в обучении и образовании. Можно выделить три вида обучения: подготовка специалистов в области АВ (студенты и аспиранты), обучение работе с САВ широкого круга пользователей (знакомство с современным инструментом исследования) и применение САВ в образовании математического и физического профиля (интенсификация образования по курсу бакалавриата).

2.3 Коммерческие и свободно распространяемые системы компьютерной математики

CAS были созданы в 70-ые годы и развивались в рамках проектов, связанных с искуственным интеллектом. Поэтому сфера их применения достаточно широкая и разнообразная. Первыми популярными системами были Reduce, Derive, Macsyma. Некоторые из них до сих пор находятся в продаже. Свободно распространяемая версия Macsyma — Maxima. На данный момент лидерами продаж являются Maple и Mathematica. Оба этих пакета активно используются в математических, инженерных и других научных исследованиях. Существует множество коммерческих систем компьютерной алгебры: Maple, Mathematica, MathCad и другие. Свободно распространяемые программы — Axiom, Eigenmath, Maxima, Yacas и др.

Успех в современном использовании САВ лежит в интеграции всех машинных возможностей (символьный и численный интерфейс, встроенная графика, мультипликация, базы и банки данных и т. д.). Все современные коммерческие системы компьютерной математики (Mathematica, Maple, MatLab и Reduce) обладают стандартным набором возможностей:

- имеется входной макроязык (для общения пользователя с системой), включающий специализированный набор функций для решения математических задач;
- имеются основные символьные (математические) объекты: полиномы, ряды, рациональные функции, выражения общего вида, векторы, матрицы;
- системы используют целые, рациональные, вещественные, комплексные числа;
- имеется несколько дополняющих друг друга режимов работы: редактирование, диагностика, диалог, протокол работы;

- присутствует связь со средствами разработки программ: возможны подстановки, вычисления значений, генерация программ, использование стандартного математического обеспечения (библиотек);

- используются интерфейсы для связи с офисными средствами, базами данных, графическими программными средствами и т. п.

Хотя между системами имеются различия, синтаксис ассоциированных языков не является проблемой, затрудняющей их использование. Синтаксис языков СКМ в значительной степени аналогичен синтаксису Паскаля. Обязательно имеются операторы присваивания, понятие вызывающей функции (команды), более или менее богатый выбор управляющих структур (if, do, while, repeat и т. д.), возможности для определения процедур и т. д. — в общем, весь арсенал классических языков программирования, необходимый для записи алгоритмов.

Системы компьютерной алгебры можно условно разделить на системы общего назначения и специализированные. К системам общего назначения относятся Macsyma, Reduce, Mathematica, Maple, Axiom и др.

В 80-е годы прошлого века широкое распространение в СССР получила система Reduce. Она первоначально предназначалась для решения физических задач, разрабатывалась на наиболее широко распространенных компьютерах, и разработка до определенного времени не носила коммерческого характера (система до конца 80-х годов распространялась бесплатно). Открытый характер системы позволил привлечь к ее разработке огромную армию пользователей, обогативших систему многочисленными пакетами для решения отдельных задач.

Macsyma так же, как и Reduce, является «старой» системой. В отличие от Reduce, Macsyma разрабатывалась с самого начала как коммерческий продукт. В ней более тщательно проработаны алгоритмические вопросы, ее эффективность существенно выше, но меньшее ее распространение можно объяснить двумя обстоятельствами: длительное время она была реализована только на малом числе «экзотических» компьютеров и распространялась только на коммерческой основе.

Система Maple, созданная в 80-х годах прошлого века в Канаде, с самого начала была задумана как система для персональных компьютеров, учитывающая их особенности. Система развивается «вширь и вглубь», даже ее ядро переписывалось с одного алгоритмического языка на другой. В настоящее время Maple широко применяется во многих странах (в частности в США и Канаде) в учебном процессе, а также в различных областях научных и технических исследований.

В конце прошлого века получила широкое распространение и сейчас быстро развивается система Mathematica. Ее успех в значительной степени объясняется ее широкими графическими возможностями, а также электронной документацией, которую можно рассматривать как электронную библиотеку, посвященную различным разделам математики и информатики.

Особое место среди СКМ занимает система Axiom. В отличие от остальных систем, представляющих собой пакеты программ, общение с которыми осуществляется на некотором алголоподобном языке, система Axiom, развившаяся из системы SCRATCHPAD-II, имеет дело с более привычными для математиков объектами. В частности, ключевым понятием в ней является понятие категории: здесь можно рассматривать, например, категории множеств, полугрупп, дифференциальных колец, левых модулей и т. д. Система имеет высокую степень универсальности, требует для своей реализации мощные компьютеры, распространяется за достаточно высокую плату, поэтому используется только в ограниченном числе мощных университетских и научных центров.

Специализированные системы отличаются более высокой эффективностью, но область их применения ограничена. К специализированным системам относятся такие системы, как CALEY и GAP — специализированные системы для вычислений в теории групп; MACAULEY, CoCoA, Singular — системы разной степени универсальности для вычислений в кольце многочленов; SCHOONSHIP — специализированная система для вычислений в физике высоких энергий; muMATH и ее правонаследница Derive — системы, широко используемые в учебном процессе (в частности, в Австрии лицензия на установку системы Derive приобретена для всех средних школ), и многие другие.

Maple — это система для аналитического и численного решения математических задач, возникающих как в математике, так и в прикладных науках. Развитая система команд, удобный интерфейс и широкие возможности позволяют эффективно применять Maple для решения проблем математического моделирования. Maple состоит из ядра, процедур, написанных на языке C и в высшей

степени оптимизированных, библиотеки, написанной на Maple-языке, и интерфейса. Ядро выполняет большинство базисных операций. Библиотека содержит множество команд и процедур, выполняемых в режиме интерпретации. Программируя собственные процедуры, пользователь может пополнять ими стандартный набор и таким образом расширять возможности Maple. Работа в Maple проходит в режиме сессии (session). Пользователь вводит предложения (команды, выражения, процедуры и др.), которые воспринимаются Maple. По умолчанию результаты сеанса сохраняются в файле с расширением `ms`. Если задан режим сохранения состояния сеанса (session), то в файле с расширением `m` будут записаны текущие назначения.

Mathematica — это широко используемая CAS, изначально разработанная Стивеном Вольфрандом, которая продается компанией Wolfram Research. Автор начал роботу над Mathematica в 1986 году и выпустил ее в 1988 году. Mathematica не только CAS, но и мощный язык программирования. Этот язык программирования реализован на основе обьектно ориентированного варианта языка C, расширяемого при помощи так называемых библиотек кода. Эти библиотеки представляют собой текстовые файлы, написанные на языке Mathematica. Архитектура Mathematica представлена ядром и пользовательским интерфейсом. Ядро программы отвечает за интерпретацию програм, написанных на языке Mathematica, и непосредственно занимается вычислениями. Пользовательские интерфейсы предназначены для выводов результатов в форме, понятной пользователю. По мнению компании-разработчика, большая часть пользователей Mathematica — это технические профессионалы. Также Mathematica широко используется в образовании. Сейчас несколько тысяч курсов на основе этого продукта читаются во многих учебных заведениях, начиная от средней школы и заканчивая аспирантурой. Mathematica используется в самых крупных университетах по всему миру и в группе компаний Fortune 500, а также во всех 15 основных министерствах правительства США.

MathCad — это CAS, очень похожая на Mathematica, которая распространяется компанией Mathsoft. MathCad ориентирован на поддержку концепций рабочего листа. Уравнения и выражения выражаются на рабочем листе так, как они выглядели бы на какой-нибуть презентации, а не так, как выглядят на языке программирования. Некоторые задачи, которые выполняет программа, — решение дифференциальных уравнений, построение графиков на плоскости и в пространстве, символьное исчисление, операции с векторами и матрицами, символьное решение систем уравнений, подбор графиков, набор статистических функций и вероятностных распределений. По мнению разработчиков MathCad, главный конкурент этого пакета — электронные таблицы. Многие пользователи используют электронные таблицы или языки программирования для выполнения вычислений. Но ни те, ни другие не справляются с задачей, когда дело доходит до обработки полученных данных. Электронные таблицы разработаны для бухгалтерских, а не для инженерных расчетов. Для последних они не слишком удобны: уравнения спрятаны в ячейках, сложно вставлять комментарии. Это делает работу довольно затруднительной, а устранять ошибки и разбираться в чьих-то вычислениях еще более сложно. Электронные таблицы трудны для понимания и повторного использования другими пользователями.

Yacas — это свободная (open source based) CAS общего назначения. Базируется на собственном языке программирования, главной целью при разработке которого была простота реализации новых алгоритмов. Этот язык очень похож на LISP, поддерживает ввод и вывод в обычном текстовом режиме как интерактивно, так и в режиме пакетного выражения.

Maxima является потомком DOE Macsyma, которая начала свое существование в конце 1960 года в MIT (англ. Massachusetts Institute of Technology — Массачусетсский технологический институт). Macsyma первая создала систему компьютерной алгебры, она проложила путь для таких програм как Maple и Mathematica. Главный вариант Maxima разрабатывался Вильямом Шелтером с 1982 по 2001 год. В 1998 году он получил разрешение на реализацию открытого кода на GPL. Благодаря его умению Maxima сумела выжить и сохранить свой оригинальный код в рабочем состоянии. Вскоре Вильям передал Maxima группе пользователей и разработчиков, которые обеспечили ее поддержку и развитие. На сегодняшний день пакет достаточно активно развивается и во многих отношениях не уступает таким развитым системам компьютерной математики, как Maple или Matematica.

Глава 3

Основы Maxima

3.1 Структура Maxima

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

3.1.1 Области математики, поддерживаемые в Maxima

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

Перечень дополнительных пакетов для Maxima, которые необходимо загружать перед использованием, существенно расширяющих её возможности и круг решаемых задач. приведен в приложении 1.

3.2 Достоинства программы

Основными преимуществами программы Maxima являются:

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

- возможность функционирования под управлением различных ОС (в частности Linux и Windows);
- небольшой размер программы (дистрибутив занимает порядка 23 мегабайт, в установленном виде со всеми расширениями потребуется около 80 мегабайт);
- широкий класс решаемых задач;
- возможность работы как в консольной версии программы, так и с использованием одного из графических интерфейсов (xMaxima, wxMaxima или как плагин (plug-in) к редактору TexMacs);
- расширение wxMaxima (входящее в комплект поставки) предоставляет пользователю удобный и понятный интерфейс, избавляет от необходимости изучать особенности ввода команд для решения типовых задач;
- интерфейс программы на русском языке;
- наличие справки и инструкций по работе с программой (русскоязычной версии справки нет, но в сети Интернет присутствует большое количество статей с примерами использования Maxima);

3.3 Установка и запуск программы

Скачать последнюю версию программы можно с ее сайта в сети Интернет: http://maxima.sourceforge.net/.
Русская локализация сайта: http://maxima.sourceforge.net/ru/.

Система компьютерной алгебры Maxima присутствует в большинстве дистрибутивов, однако зачастую в списке дополнительных программ, которые можно скачать в Интернете в версии для данного дистрибутива. Примеры и расчёты в данной книге выполнены с использованием дистрибутива AltLinux 4.1.

3.4 Интерфейс wxMaxima

Для удобства работы сразу обратимся к графическому интерфейсу wxMaxima, т. к. он является наиболее дружественным для начинающих пользователей системы.

Достоинствами wxMaxima являются:

- возможность графического вывода формул (см. иллюстрации ниже)
- упрощенный ввод наиболее часто используемых функций (через диалоговые окна), а не набор команд, как в классической Maxima.
- разделение окна ввода данных и области вывода результатов (в классической Maxima эти области объединены, и ввод команд происходит в единой рабочей области с полученными результатами).

Рассмотрим рабочее окно программы. Сверху вниз располагаются: текстовое меню программы - доступ к основным функциям и настройкам программы. В текстовом меню wxMaxima находятся функции для решения большого количества типовых математических задач, разделенные по группам: уравнения, алгебра, анализ, упростить, графики, численные вычисления. Ввод команд через диалоговые окна упрощает работу с программой для новичков.

При использовании интерфейса wxMaxima, Вы можете выделить в окне вывода результатов необходимую формулу и вызвав контекстное меню правой кнопкой мыши скопировать любую формулу в текстовом виде, в формате TEX или в виде графического изображения, для последующей вставки в какой-либо документ.

Также в контекстном меню, при выборе результата вычисления, Вам будет предложен ряд операций с выбранным выражением (например, упрощение, раскрытие скобок, интегрирование, дифференцирование и др.).

3.5 Ввод простейших команд Maxima

Все команды вводятся в поле ВВОД, разделителем команд является символ ; (точка с запятой). После ввода команды необходимо нажать клавишу Enter для ее обработки и вывода результата. В ранних версиях Maxima и некоторых ее оболочках (например, xMaxima) наличие точки с запятой после каждой команды строго обязательно. Завершение ввода символом $ (вместо точки с запятой) позволяет вычислить результат введённой команды, но не выводить его на экран. В случае, когда выражение надо отобразить, а не вычислить, перед ним необходимо поставить знак ' (одинарная кавычка). Но этот метод не работает, когда выражение имеет явное значение, например, выражение $sin(\pi)$ заменяется на значение равное нулю.

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

Пример:

```
(%i1) aa:1024;
```

(%o1) 1024

```
(%i2) bb:19;
```

(%o2) 19

```
(%i3) sqrt(aa)+bb;
```

(%o3) 51

```
(%i4) '(sqrt(aa)+bb);
```

(%o4) $bb + \sqrt{aa}$

```
(%i5) ''%;
```

(%o5) 51

3.5.1 Обозначение команд и результатов вычислений

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

Результат вычисления также имеет порядковый номер, например (%o1), (%o2) и т.д.,где **i** - сокращение от англ. input (ввод), а **o** - англ. output (вывод). Этот механизм позволяет избежать в последующих вычисления повторения полной записи уже выполненных команды, например (%i1)+(%i2) будет означать добавление к выражению первой команды - выражения второй и последующего вычисления результата. Также можно использовать и номера результатов вычислений, например (%o1)*(%o2). Для последней выполненной команды в Maxima есть специальное обозначение - %. Пример: Вычислить значение производной функции $y(x) = x^2 \cdot exp(-x)$):

```
(%i1) diff(x^2*exp(-x),x);
```

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

```
(%i2) f(x):=''%;
```

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

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

Другой пример (с очевидным содержанием):

```
(%i3) x:4;
```

(%o3) 4

```
(%i4) sqrt(x);
```

(%o4) 2

```
(%i5) %^2;
```

(%o5) 4

3.6 Числа, операторы и константы

3.6.1 Ввод числовой информации

. Правила ввода чисел в Maxima точно такие, как и для многих других подобных программ. Целая и дробная часть десятичных дробей разделяются символом точка. Перед отрицательными числами ставится знак минус. Числитель и знаменатель обыкновенных дробей разделяется при помощи символа / (прямой слэш). Обратите внимание, что если в результате выполнения операции получается некоторое символьное выражение, а необходимо получить конкретное числовое значение в виде десятичной дроби, то решить эту задачу позволит применение флага numer. В частности он позволяет перейти от обыкновенных дробей к десятичным. Преобразование к форме с плавающей точкой осуществляет также функция float.

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

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

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

(%o2) 2.095238095238095

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

(%o3) 2.095238095238095

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

(%o4) 0.71428571428571

3.6.2 Арифметические операции

Обозначение арифметических операций в Maxima ничем не отличается от классического представления: + , - , * , /. Возведение в степень можно обозначать несколькими способами: ^ ; ** . Извлечение корня степени n записываем, как степень 1/n. Операция нахождение факториала обозначается восклицательным знаком, например 5!. Для увеличения приоритета операции, как и в математике, используются круглые скобки: (). Список основных арифметических и логических операторов приведен в таблицах ниже.

Таблица 3.1. Арифметические операторы

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

Таблица 3.2. Логические операторы

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

3.6.3 Константы

В Maxima для удобства вычислений имеется ряд встроенных констант. Самые распространенные из них показаны в следующей таблице:

3.7 Типы данных, переменные и функции

Для хранения результатов промежуточных расчетов применяются переменные. Заметим, что при вводе названий переменных, функций и констант важен регистр букв, так переменные x и X - две разные переменные. Присваивание значения переменной осуществляется с использованием символа: (двоеточие), например x:5. Если необходимо удалить значение переменной (очистить ее), то применяется метод kill:

kill(x) - удалить значение переменной x;

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

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

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

Таблица 3.3. Основные константы Maxima

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

3.7.1 Списки

Списки - базовые строительные блоки для Maxima и Lisp. Все прочие типы данных (массивы, хэш-таблицы, числа) представляются как списки. Чтобы задать список, достаточно запи- сать его элементы через запятую и ограничить запись квадратными скобками. Список может быть пустым или состоять из одного элемента

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

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

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

(%o2) []

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

(%o3) [3]

Элементом списка может и другой список

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

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

Ссылка на элемент списка производится по номеру элемента списка:

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

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

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

(%o5) 1

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

(%o6) [3, 4]

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

(%o7) 4

3.7.1.1 Функции для элементарных операций со списками

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

```
(%i8) length(list4);
```

(%o8) 4

```
(%i9) length(list3);
```

(%o9) 1

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

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

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

```
(%i2) list2:copylist(list1);
```

(%o2) $[1, 2, 3, x, y + x]$

Функция makelist создаёт список, каждый элемент которого генерируется из некоторого выражения. Возможны два варианта вызова этой функции:

```
makelist (expr, i, i_0, i_1)
makelist (expr, x, list)
```

Вызов

```
makelist (expr, i, i_0, i_1)
```

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

Вызов

```
makelist (expr, x, list)
```

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

Примеры:

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

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

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

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

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

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

Во многом аналогичные действия выполняет функция $create_list(form, x_1, list_1, ..., x_n, list_n)$. Эта функция строит список путём вычисления выражения $form$, зависящего от x_1, к каждому элементу списка $list_1$ (аналогично $form$, зависящая и от x_2, применяется к $list_2$ и т.д.). Пример:

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

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

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

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

Функция append позволяет склеивать списки. При вызове

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

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

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

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

Создать новый список, компонуя элементы двух списков поочередно в порядке следования, позволяет функция join(l,m). Новый список содержит l_1, затем m_1, затем l_2, m_2 и т.д. Пример:

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

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

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

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

Длина полученного списка ограничивается минимальной длиной списков l и m.

Функция cons (expr, list) создаёт новый список, первым элементом которого будет expr, а остальные - элементы списка list. Функция endcons (expr, list) также создаёт новый список, первые элементы которого - элементы списка list, а последний - новый элемент expr. Пример:

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

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

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

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

Функция reverse меняет порядок элементов в списке на обратный

```
(%i5) list1:[1,2,3,x];
```

(%o5) $[1, 2, 3, x]$

```
(%i6) list2:reverse(list1);
```

(%o6) $[x, 3, 2, 1]$

Функция $member(expr_1, expr_2)$ возвращает "true если expr1 является элементом списка expr2, и "false"в противном случае. Пример:

```
(%i1) member (8, [8, 8.0, 8b0]);
```

(%o1) $true$

```
(%i2) member (8, [8.0, 8b0]);
```

(%o2) $false$

```
(%i3) member (b, [[a, b], [b, c]]);
```

(%o3) $false$

```
(%i4) member ([b, c], [[a, b], [b, c]]);
```

(%o4) $true$

Функция rest(expr) выделяет остаток после удаления первого элемента списка expr. Можно удалить первые n элементов, используя вызов rest(expr,n). Функция last(expr) выделяет последний элемент списка expr (аналогично first - первый элемент списка). Примеры:

```
(%i1) list1:[1,2,3,4,a,b];
```

(%o1) $[1, 2, 3, 4, a, b]$

```
(%i2) rest(list1);
```

(%o2) $[2, 3, 4, a, b]$

```
(%i3) rest(%);
```

(%o3) $[3, 4, a, b]$

```
(%i4) last(list1);
```

(%o4) b

```
(%i5) rest(list1,3);
```

(%o5) $[4, a, b]$

Сумирование и перемножение списков (как и прочих выражений) осуществляется фугкциями sum и product. Функция sum(expr,i,in,ik) суммирует значения выражения expr при изменении индекса i от in до ik. Функция product(expr,i,in,ik) перемножает значения выражения expr при изменении индекса i от in до ik. Пример:

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

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

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

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

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

(%o3) 576

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

(%o4) 30

3.7.1.2 Функции, оперирующие с элеметами списков

Функция $map(f, expr_1, ..., expr_n)$ позволяет применить функцию (оператор, символ операции) f к частям выражений $expr_1$, $expr_2$, $\ldots$, $expr_n$. При использовании со списками применяет f к каждому элементу списка. Следует обратить внимание, что f - именно имя функции (без указания переменных, от которых она зависит). Примеры:

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

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

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

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

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

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

Функция f может быть и заданной пользователем, например:

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

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

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

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

Функция apply применяет заданную функцию ко всему списку (список становится списком аргументов функции; при вызове $(F, [x_1, ..., x_n]$ вычисляется выражение $F(arg_1, ..., arg_n))$. Следует учитывать, что apply не распознаёт ординарные функции и функции от массива.

Пример:

```
(%i1) L : [1, 5, -10.2, 4, 3];
```

(%o1) $$[1, 5, -10.2, 4, 3]$$

```
(%i2) apply(max,L);
```

(%o2) $$5$$

```
(%i3) apply(min,L);
```

(%o3) $$-10.2$$

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

3.7.2 Массивы

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

Возможно создание и использование переменных с индексами до объявления соответствующего масива. Такие переменные рассматриваются как элементы массивов с неопределёнными размерностями (так называемые хэш-массивы). Размеры неопределённых массивов растут динамически по мере присваивания значений элементам. Интересн, что индексы массивов с неопределёнными границами не обязательно должны быть числами. Дляповышения эффективности вычислений рекомендуется преобразовывать массивы с неопределёнными границами в обычные массивы (для этого используется функция array).

Создание массива производится функцией array. Синтаксис обращения к функции:

```
array (name, dim_1, ..., dim_n)

array (name, type, dim_1, ..., dim_n)

array ([name_1, ..., name_m], dim_1, ..., dim_n)
```

Индексы обычного массива - целые числа, изменяющиеся от 0 до dim_i.

Пример:

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

(%o1) a

```
(%i2) a[0,0]:0; a[0,1]:1; a[1,0]:2; a[1,1]:3;
```

(%o5) 0123

```
(%i6) listarray(a);
```

(%o6) $[0, 1, 2, 3]$

Функция listarray, использованная в примере, преобразовывает массив в список. Синтаксис вызова:
listarray (A)

Аргумент A может быть определённым или неопредедённым массивом, функцией массива или функцией с индексами. Порядок включения элементов массива в список - по строкам.

Функция arrayinfo выводит информацию о массиве A. Синтаксис вызова:
arrayinfo (A) Аргумент A, как и в случае listarray, может быть определённым или неопредедённым массивом, функцией массива или функцией с индексами. Пример использования:

```
(%i1) array (aa, 2, 3);
```

(%o1) aa

```
(%i2) aa [2, 3] : %pi;
```

(%o2) π

```
(%i3) aa [1, 2] : %e;
```

(%o3) e

```
(%i4) arrayinfo (aa);
```

(%o4) $[declared, 2, [2, 3]]$

```
(%i5) bb [FOO] : (a + b)^2;
```

(%o5) $(b + a)^2$

```
(%i6) bb [BAR] : (c - d)^3;
```

(%o6) $(c - d)^3$

```
(%i7) arrayinfo (bb);
```

(%o7) $[hashed, 1, [BAR], [FOO]]$

```
(%i8) listarray (bb);
```

(%o8) $[(c - d)^3, (b + a)^2]$

Функции listarray и arrayinfo применимы и к функциям массива:

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

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

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

(%o10) 2

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

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

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

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

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

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

Ещё один пример - создание и вывод информации о функциях с индексами:

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

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

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

(%o2) 4

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

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

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

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

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

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

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

Достоинство make_array по сравнению с array - возможность динамически управлять распределением памяти для массивов. Присваивание $y : make\_array(\dots)$ создаёт ссылку на массив. Когда массив больше не нужен, ссылка уничтожается присваиванием $y : false$, память освобождается затем сборщиком мусора. Примеры:

```
(%i1) A1 : make_array (fixnum, 10);
```

(%o1) #(0000000000)

```
(%i2) A1[1]:8;
```

(%o2) 8

```
(%i3) A3 : make_array (any, 10);
```

(%o3) $\#(NILNILNILNILNILNILNILNIL)$

```
(%i4) arrayinfo(A3);
```

(%o4) $[declared, 1, [9]]$

Переменная arrays содержит список имен массивов первого и второго видов, определенных на данный момент. Пример:

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

(%o1) a

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

(%o2) b

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

(%o3) $[a, b]$

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

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

(%o1) a

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

(%o2) a

```
(%i3) a[1,1];
```

(%o3) 4

```
(%i4) a2 : make_array (fixnum, 8);
```

(%o4) #(00000000)

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

(%o5) #(12345555)

Как видно из рассмотреных примеров, длина списка может и не совпадать с размерностью массива. Если указан тип массива, он должен заполняться элементами того же типа. Удаление массивов из памяти осуществляется функцией remarray.

Кроме того, для изменения размерности массива имеется функция ;$rarray(A, dim_1, ..., dim_n)$. Новый массив заполняется элементами старого по строкам. Если размер старого масива меньше, чем нового, остатокнового заполняется нулями или false (в зависимости от типа массива).

3.7.3 Матрицы и простейшие операции с ними

В Maxime определены прямоугольные матрицы.

Основной способ создания матриц - использования функции matrix. Синтаксис вызова: $matrix(row_1, ..., row_n)$. Каждая строка - списоквыражений, все строки одинаковой длины. На множестве матриц определены операции сложения, вычитания, умножения и деления. Эти операции выполняются поэлментно, если операнды - две матрицы, скаляр и матрица или матрица и скаляр. Возведение в степень возможно, если один из операндов - скаляр. Перемножение матриц (в общем случае некоммутативная операция) обозначается символом “.”. Операция умножения матрицы самой на себя может рассматриваться как возведение в степень. Возведение в степень “-1”- как обращение (если это возможно).

Пример создания двух матриц:

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

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

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

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

Выполнение арифметических операций с матрицами:

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

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

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

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

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

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

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

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

Обратите внимание - операции выполняются поэлементно. При попытке выполнять арифметические операции, как представлено выше, над матрицами различных размеров, выдаётся ошибка. Пример операций с матрицами и скалярами:

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

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

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

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

Умножение матрицы на матрицу:

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

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

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

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

Очевидно, что для успешного перемножения матрицы должны быть согласованы по размерам. Возведение в степень -1 даёт обратную матрицу:

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

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

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

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

Стоит обратить внимание, что операции

```
x^^-1
```

дают разный результат! Пример:

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

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

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

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

Функция genmatrix возвращает матрицу заданной размерности, составленную из элементов двухиндексного массива. Синтаксис вызова:

```
genmatrix (a, i_2, j_2, i_1, j_1)
genmatrix (a, i_2, j_2, i_1)
genmatrix (a, i_2, j_2)
```

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

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

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

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

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

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

(%o3) a

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

(%o4) e

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

(%o5) π

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

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

Функция zeromatrix возвращает матрицу заданной размерности, составленную из нулей (синтаксис вызова zeromatrix (m, n)).

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

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

Функция ident возвращает единичную матрицу заданной размерности (синтаксис ident(n))

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

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

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

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

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

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

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

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

(%o3) $$10$$

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

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

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

Функции row и col позволят извлечь соответственно сироку и столбец заданной матрицы, получая список. Синтаксис вызова:
row (M, i) - возвращает i-ю строку;
col (M, i) - возвращает i-й столбец.

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

```
addcol (M, list_1, ..., list_n)
addrow (M, list_1, ..., list_n)
```

Здесь $list_1, \ldots, list_n$ - добавляемые строки или столбцы. Пример:

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

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

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

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

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

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

Функция submatrix возвращает новую матрицу, состоящую из подматрицы заданной. Синтаксис вызова:

```
submatrix (i_1, ..., i_m, M, j_1, ..., j_n)
submatrix (i_1, ..., i_m, M)
submatrix (M, j_1, ..., j_n)
```

Подматрица строится следующим образом: из матрицы M удаляются строки $i_1, ..., i_m$ и $j_1, ..., j_n$. Пример (используем последний результат из предыдущего примера, удаляем третью строку и третий столбец):

```
(%i6) submatrix(3,%,3);
```

$$(\%o6) \qquad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Для заполнения матрицы значениями некоторой функции используется функция *matrixmap* (аналог map, apply, fullmap). Синтаксис вызова: matrixmap (f, M). Функция *matrixmap* возвращает матрицу с элементами i,j , равными f(M[i,j]).

Пример:

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

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

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

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

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

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

Для работы с матрицами существует ещё много функций, но они относятся к решению различных задач линейной алгебры, поэтому обсуждаются ниже, в главе 4.

3.7.4 Математические функции

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

- тригонометрические функции: sin (синус), cos (косинус), tan (тангенс), cot (котангенс);
- обратные тригонометрические функции: asin (арксинус), acos (арккосинус), atan (арктангенс), acot (арккотангенс);
- sec (секанс, sec x = 1/cos x), csc (косеканс, csc x = 1/sin x);
- sinh (гиперболический синус), cosh (гиперболический косинус), tanh (гиперболический тангенс), coth (гиперболический котангенс), sech (гиперболический секанс), cosh (гиперболический косеканс);
- log (натуральный логарифм);
- sqrt (квадратный корень);
- mod (остаток от деления);
- abs (модуль);
- min(x1,...,xn) и max(x1,...,xn) - нахождение минимального и максимального значения в списке аргументов;
- sign (определяет знак аргумента: pos - положительный, neg -отрицательный, pnz - не определен, zero - значение равно нулю);
- Специальные функции - функции Бесселя, гамма-функция, гипергеометрическая функция и др.;
- Эллиптические функции различных типов.

3.7.5 Вычисление и преобразование аналитических выражений

Функция **ev** является основной функцией, обрабатывающей выражения. Синтаксис вызова: $ev(expr, arg_1, ..., arg_n)$ Функция **ev** вычисляет выражение **expr** в окружении, определяемом аргументами $arg_1, ..., arg_n$. Аргументы могут быть ключами (булевскими флагами, присваиваниями, уравнениями и функциями. Функция **ev** возвращает результат (другое выражение).

Во многих случаях можно опускать имя функции **ev** (т.е. применять значения переменных к некоторому выражению) $expr, flag1, flag2, ...$ $expr, x = val1, y = val2, ...$ $expr, flag1, x = val1, y = val2, flag2, ...$

На выражение **expr** по умолчанию действует функция упрощения. Необходимость выполнения упрощения регулируется флагом *simp* (если установить $simp = false$, упрощение будет отключено. Кроме того, используют флаги $float$ и $numer$, определяющие формат представления рациональных чисел (в виде дробей или с плавающей точкой) и результатов вычисления математических функций. Флаг *pred* опрежеляет необходимость вычисления примнтельно к логическим выражениям. Аргументами *ev* могут быть и встроенные функции, выполняющие упрощение или преобразование выражений (expand, factor, trigexpand, trigreduce) или функция $diff$.

Если указаны подстановки (в виде **x=val1** или **x:val2**), то они выполняются.

При этом повторный вызов функции **ev** вполне способен еще раз изменить выражение, т.е. обработка выражения не идет до конца при однократном вы- зове функции **ev**. Пример:

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

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

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

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

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

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

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

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

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

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

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

Флаг simp разрешает либо запрещает упрощение выражений. Изначально она равна **true**, если установить ее равной **false**, то упрощения производиться не будут:

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

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

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

(%o2) $$false$$

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

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

Функцию *ev* не обязательно указывать явно, например:

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

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

Оператор, принудительного вычисления, обозначенный двумя апострофами, является синонимом к функции ev(выражение). Сама функция ev предоставляет гораздо более широкие возможности, нежели простое принудительное вычисление заданного выражения: она может принимать произвольное число аргументов, первый из которых — вычисляемое выражение, а остальные — специальные опции, которые как раз и влияют на то, как именно будет производиться вычисление.

В терминологии Maxima невычисленная форма выражения называется «noun form», вычисленная — «verb form». Сохраняя лингвистические параллели, на русский это можно перевести как «несовершенная форма» и «совершённая форма». Значение вводимого выражения в Maxima закономерно сохраняется до его вычисления (т. е. в несовершенной форме), а значение выводимого выражения — после (т. е. в совершённой); другими словами, тут имеется естественный порядок «ввод — вычисление — вывод».

Функция factor факторизует (т.е. представляет в виде произведения некоторых сомножителей) заданное выражение (функция gfactor - аналогично, но на множестве комплексных чисел и выражений). Пример:

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

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

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

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

Ещё примеры факторизации различных выражений:

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

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

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

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

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

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

Пример использования функции gfactor:

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

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

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

(%o7) $$\left(x+i\,a\right)^{2}$$

Функция factorsum факторизует отдельные слагаемые в выражении.

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

(%o8) $$a\,x\,z^{2}+a\,z^{2}+2\,a\,w\,x\,z+2\,a\,w\,z+a\,w^{2}\,x+v^{2}\,x+2\,u\,v\,x+u^{2}\,x+a\,w^{2}+v^{2}+2\,u\,v+u^{2}$$

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

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

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

```
(%i10) gfactorsum( a^3+3*a^2*b+3*a*b^2+b^3+x^2+2*%i*x*y-y^2 );
```

(%o10) $$\left(b+a\right)^{3}-\left(y-i\,x\right)^{2}$$

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

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

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

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

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

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

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

.

Функция combine объединяет слагаемые с идентичным знаменателем

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

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

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

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

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

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

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

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

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

Функция multthru умножает каждое слагаемое в сумме на множитель, причем при умножении скобки в выражении не раскрываются. Она допускает два варианта синтаксиса multthru(mult,sum); multthru(expr); В последнем случае выражение *expr* включает и множитель, и сумму (см. второй пример).

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

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

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

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

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

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

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

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

Функции assume (ввод ограничений) и forget (снятие ограничений) позволяют управлять условиями выполнения (контекстом) прочих функций и операторов. Пример:

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

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

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

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

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

Функция gcd позволяет найти наибольший общий делитель многочленов

Подстановки осуществляются функцикй subst. вызов этой функции: subst (a, b, c) (подставляем a вместо b в выражении c). Пример:

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

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

3.7.6 Преобразование рациональных выражений

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

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

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

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

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

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

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

Функция rat приводит выражение к каноническому представлению. Она упрощает любое выражение, рассматривая его как дробно- рациональную функцию, т.е. работает с операциями

```
"+", "-", "*", "/"
```

и с возведением в целую степень. Синтаксис вызова: $rat(expr)$ $rat(expr, x_1, \ldots, x_n)$ Переменные упорядочиваются в соответствии со списком $x_1, \ldots, x_n$. При этом вид ответа зависит от способа упорядочивания переменных Изначально переменные упорядочены в алфавитном порядке. Пример использования rat:

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

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

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

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

После указания порядка использования переменных получаем следующее выражение:

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

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

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

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

Функция ratsimp приводит все части (в том числе аргументы функций) выражения, которое не является дробно-рациональной функцией, к каноническому представлению, производя упрощения, которые не выполняет функция "rat". Повторный вызов функции в общем случае может изменить результат, т.е. не обязательно упрощение проводится до конца. Применением упрощения к экспоненциальным выражениям управляет флаг $ratsimexpons$, по умолчанию равный $false$ (если его установить в $true$, упрощение применяется и к показателям степени или экспоненты.

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

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

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

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

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

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

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

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

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

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

Функция fullratsimp вызывает функцию "ratsimp"до тех пор, пока выражение не перестанет меняться. Пример:

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

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

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

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

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

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

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

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

Пример влияния флага *ratsimpexponds* на результат вычислений:

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

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

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

(%o2) $$true$$

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

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

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

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

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

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

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

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

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

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

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

Подстановка в рациональных выражениях осуществляется функцией ratsubst. Синтаксис вызова: ratsubst (a, b, c) Выражение a подставляется вместо выражения b в выражении c (b может быть суммой, произведением, степенью и т.п.). Пример использования ratsubst:

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

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

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

(%o2) $$cos\left(x\right)^4+cos\left(x\right)^3+cos\left(x\right)^2+cos\left(x\right)+1$$

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

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

3.7.7 Преобразование тригонометрических выражений

Функция trigexpand раскладывает все тригонометрические и гиперболические функции от сумм и произведений в комбинации соответствующих функций единичных углов и аргументов. Для усиления пользовательского контроля один вызов trigexpand выполняет упрощение на одном уровне. Для управления вычислением имеется флаг "trigexpand". Изначально флаг "trigexpand"установлен в "false". Если флаг "trigexpand"установить в "true то функция "trigexpand"будет работать до тех пор, пока выражение не перестанет меняться.

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

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

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

(%o2) $$cos\left(10\,x\right)\,sin\left(y\right)+sin\left(10\,x\right)\,cos\left(y\right)$$

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

(%o3) $$sin\left(x\right)^4-sin\left(x\right)^3-6\,cos\left(x\right)^2\,sin\left(x\right)^2+3\,cos\left(x\right)^2\,sin\left(x\right)+cos\left(x\right)^4$$

Функция trigreduce свертывает все произведения тригонометрических и гиперболических функций в комбинации соответствующих функции от сумм. Функция работает не до конца, так что повторный вызов может изменить выражение. При вызове функции в формате $trigreduce(expr, x)$ преобразования осуществляются относительно функций x. Примеры:

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

(%o8) $$\frac{cos\left(4\,x\right)+4\,cos\left(2\,x\right)+3}{8}+\frac{cos\left(3\,x\right)+3\,cos\left(x\right)}{4}+\frac{cos\left(2\,x\right)+1}{2}+cos\left(x\right)+1$$

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

(%o9) $$\frac{cos\left(2\,x\right)}{2}+3\,\left(\frac{cos\left(2\,x\right)}{2}+\frac{1}{2}\right)+x-\frac{1}{2}$$

Функция trigsimp упрощает тригонометрические и и гиперболические выражения, применяя к ним правила $sin(x)^2+cos(x)^2=1$ и $cosh(x)^2-sinh(x)^2=1$. Пример:

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

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

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

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

Функция trigrat (синтаксис вызова $trigrat(expr)$) приводит заданное тригонометрическое выражение $expr$ к канонической упрощённой квазилинейной форме. Это выражение рассматривается как рациональное, содержащее sin, cos, tan, аргкменты которых линейные формы некоторых переменных и $\frac{\pi}{n}$ (n - целое). Всегда, когда возможно, заданное выражение линеаризуется. Пример:

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

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

3.7.8 Преобразование степенных и логарифмических выражений

Функция radcan упрощает выражения, содержащие экспоненты, логарифмы и радикалы, путем преобразования к форме, которая является канонической для широкого класса выражений. Переменные в выражении упорядочиваются. Эквивалентные выражения в этом классе не обязательно одинаковы, но их разность упрощается применением radcan до нуля. Примеры:

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

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

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

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

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

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

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

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

Функция logcontract (expr) рекурсиво сканирует выражение expr, преобразуя выражения вида $a1*log(b1)+a2*log(b2)+c$ к форме $log(ratsimp(b1^a1*b2^a2))+c$.

Пример:

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

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

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

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

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

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

(%o1) $$done$$

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

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

3.7.9 Пользовательские функции

Для записи функции необходимо указать ее название, а затем, в круглых скобках записать через запятую значения аргументов. Если значением аргумента является список, то он заключается в квадратные скобки, а элементы списка также разделяются запятыми. Пример:

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

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

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

(%o44) f(x):=x^2 (%i45) f(3 + 7) ; (%o45) 100

Не следует использовать для функций названия, зарезервированные для встроенных функций Maxima. Для создания функций используется также встроенная функция $define$, которая позволяет преобразовать выражение в функцию. Синтаксис вызова $define$ довольно многообразен:

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

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

Ординарная функция:

```
(%i1) expr : cos(y) - sin(x);
```

(%o1) $$cos\,(y) - sin\,(x)$$

```
(%i2) define (F1 (x, y), expr);
```

(%o2) $$F1\,(x, y) := cos\,(y) - sin\,(x)$$

```
(%i3) factor(F1(a,b));
```

(%o3) $$cos\,(b) - sin\,(a)$$

Созздание функции-массива:

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

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

Создание массива:

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

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

Использование функции *ev* для задания пользовательской функции:

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

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

3.8 Решение задач элементарной математики

3.8.1 Нахождение корней уравнений и систем алгебраических уравнений

Решение алгебраических уравнений и их систем осуществляется при помощи функции solve, в качестве параметров в первых квадратных скобках указывается список уравнений через запятую, во-вторых - список переменных, через запятую (либо несколько упрощённые формы записи):

```
solve (expr, x)
solve (expr)
solve ([eqn_1, ..., eqn_n], [x_1, ..., x_n])
```

Примеры: Решение одного уравнения с одним неизвестным

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

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

Решение одного уравнения в символьном виде:

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

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

Решение системы уравнений в символьном виде:

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

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

В последнем примере решений несколько, и Maxima выдаёт результат в виде списка. Функция solve применима и для решения тригонометрических уравнений. При этом в случае множества решений у тригонометрических уравнений выдается соответствующее сообщение только и одно из решений. Пример:

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

(%o13) $$`solve'\,is\,using\,arc - trig\,functions\,to\,get\,a\,solution.\,Some\,solutions\,will\,be\,lost.[x = 0]$$

Также Maxima позволяет находить комплексные корни

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

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

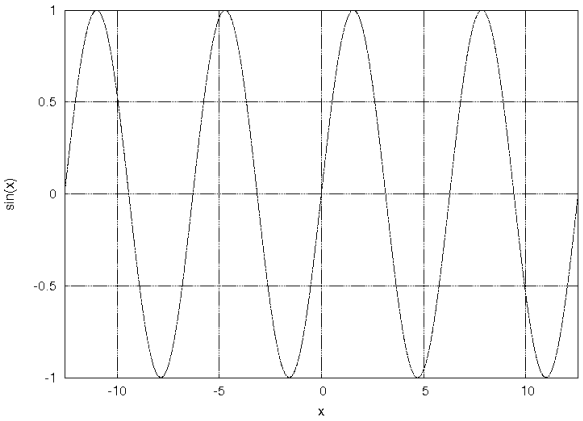


Рис. 3.1. Простейшая команда построения графика

3.9 Построение графиков и поверхностей

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

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

Все варианты интерфейса (кроме wxMaxima) для построения графиков используют две базовых функции: plot2d(построение двумерных графиков) и plot3d(построение трхмерных графиков). При использовании wxMaxima кроме них испольуются ещё две аналогичные команды: wxplot2d и wxplot3d. Все команды позволяют либо вывести график на экран, либо (в зависимости от параметров функции) в файл.

3.9.1 Построение графика явной функции y=f(x)

График функции y=f(x) на отрезке [a, b] можно построить с помощью функции **plot2d(f(x), [x,a,b], опции) или plot2d(f(x), [x,a,b], [y,c,d], опции).** Опции не обязательны, однако, для изменения свойств графика их нужно задавать. Параметр [y,c,d] можно не задавать, тогда высота графика выбирается по умолчанию. Построим график функции y=sinx на отрезке [-4π, 4π].

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

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

Результаты приведены на рис. 3.1, 3.2.

3.9.2 Построение графиков функций, заданных параметрически

Для построения графиков функций, заданных параметрически, используется опция **parametric**. Для построения графика указывается область изменения параметра. Пример графика простейшей параметрической функции представлен на 3.3. Команда построения графика: plot2d ([parametric, cos(t), sin(t), [t,-%pi,%pi], [nticks,80]],[x, -4/3, 4/3])

Опция ntics указывает число точек, по которым проводится кривая

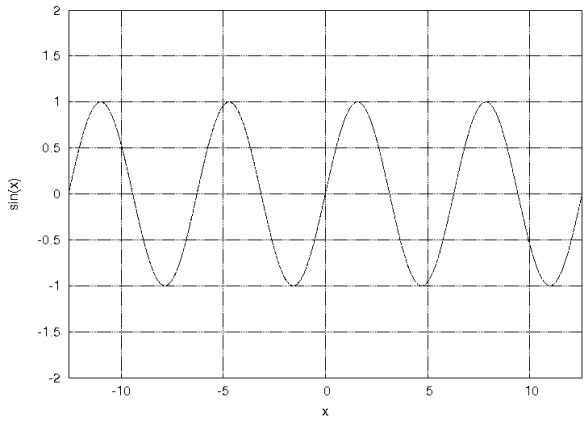


Рис. 3.2. Простейшая команда построения графика с указанием интервала по оси Oy

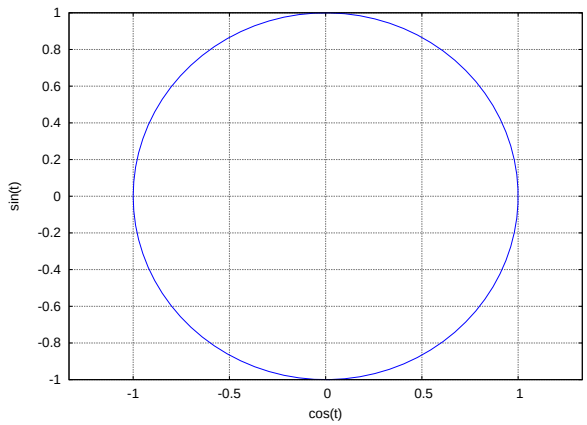


Рис. 3.3. Простейшая команда построения графика функции, заданной параметрически

Рассмотрим некоторые опции.

Опции указываются в виде аргументов функции plot2d в квадратных скобках. Возможна установка легенды, меток на осях, цвета и стиля графика. Применение нескольких опций характеризует следующий пример:

```
(%i17) plot2d([[discrete,xy], 2*\%pi*sqrt(l/980)], [l,0,50],

       [style, [points,5,2,6], [lines,1,1]],

       [legend, experiment , theory ],

       [xlabel,"pendulum s length (cm)"], [ylabel,"period (s)"])
```

В данном примере в одних осях строятся 2 графика. Первый ([discrete,xy]) строится в виде точек по массиву xy с указанием стиля points. Второй строится по уравнению функции 2*%pi*sqrt(l/980) с указанием стиля lines. Опция legend указывает подписи кривых, опции xlabel и ylabel — подписи осей. Результат приведен на рис. 3.4.

Формирование массивов для построения графыика осуществляется следующим образом:

```
(%i12) xx:[10, 20, 30, 40, 50];

(%i13) yy:[.6, .9, 1.1, 1.3, 1.4];
```

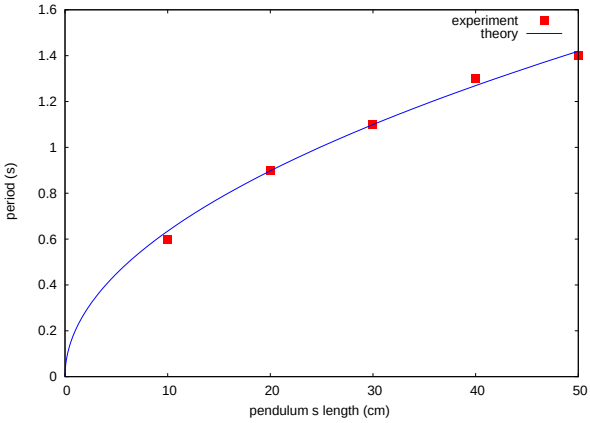


Рис. 3.4. Совмещение на одном графике действия серии опций

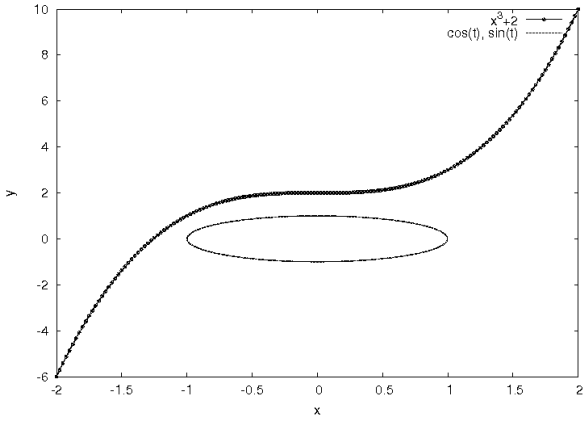


Рис. 3.5. Совмещение на одном графике параметрической и заданной явно кривых

```
(%i14) xy:[[10,.6], [20,.9], [30,1.1], [40,1.3], [50,1.4]];
```

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

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

например (см. 3.5):

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

Опции

```
[gnuplot_term, ps],[gnuplot_out_file, "test.eps"]
```

указывают, что графическая иллюстрация выводится в файл test.eps в формате postscript (бэкенд для вывода графиков — gnuplot).

Опции

```
[style, [linespoints,3,2], lines,3,1]]
```

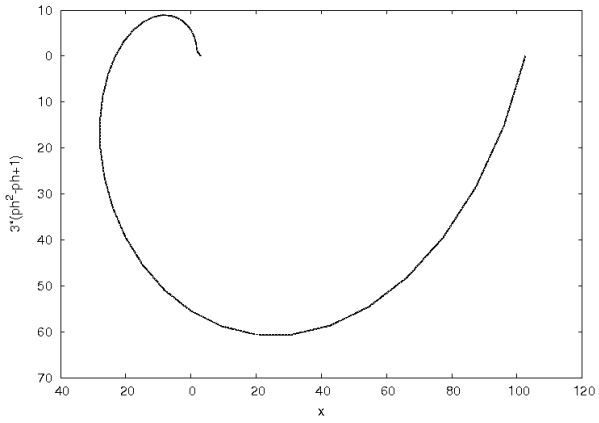


Рис. 3.6. Кривая в полярных координатах

позволяют указать стиль линий на графике (линия с точками или сплошная линия).

Для вывода результатов в формат png можно использовать опции (указание размеров 400,400 в общем случае необязательно):

```
[gnuplot_term, png size 400,400 ],[gnuplot_out_file, max.png ]
```

3.9.3 Построение кривых в полярной системе координат

Для построения графика в полярных координатах нужно задать изменение значений полярного радиуса и полярного угла. Пусть $\mathbf{r = r(f)}$ $\mathbf{(a \le f \le b)}$ – зависимость полярного радиуса **r** от полярного угла **f**. Тогда график этой функции в полярных координатах можно построить, задав у функции **plot2d** опцию **[gnuplot_preamble, set polar; set zeroaxis]**. Данная опция будет действовать лишь при условии, что выбран формат графика gnuplot. Пример: построить в полярных координатах график функции r = $3(1–\varphi + \varphi^2)$, $0 \le \varphi$ 2π.

Для создания графика используем команду:

```
plot2d([3*(1-ph+ph^2)], [ph,0,2*%pi],
[gnuplot_preamble,"set polar","set zeroaxis","set encoding koi8r"],
[xlabel, x],[gnuplot_term,ps],[gnuplot_out_file, "max.eps"], [plot_format,gnuplot]);
```

Результат приведен на рис. 3.6 . Толщину и стиль линии можно регулировать, используя опцию style (например, опция

```
[style,␣[lines,3,1]]
```

устанавливает ширину линии 3 и синий цвет)

Пример: построить в полярных координатах графики трех функций

r = $6\cos\varphi$, r=φ, r=$2\sin\varphi$, $0 \le \varphi$ 2π.

Для создания графика используем команду:

```
plot2d([6*cos(ph),ph,2*sin(ph)], [ph,0,2*%pi],
[gnuplot_preamble,"set polar","set zeroaxis","set encoding koi8r"],
[xlabel, x],[gnuplot_term,ps],[gnuplot_out_file, "max3.eps"], [plot_format,gnuplot]);
```

Результат приведен на 3.7

3.9.4 Построение трёхмерных графиков

Основная команда для построения трёхмарных графиков - plot3d.

Рассмотрим технологию построения графиков с использованием интерфейса gnuplot.

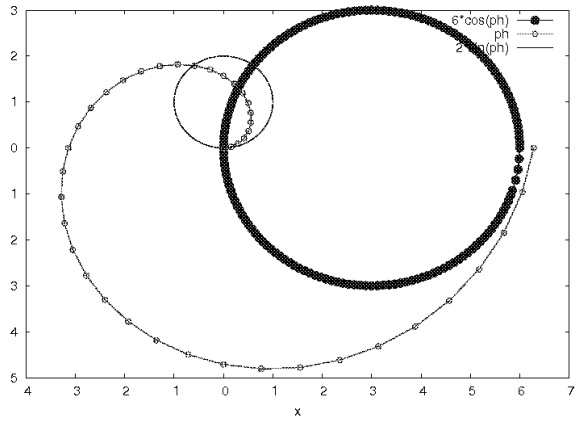


Рис. 3.7. Совмещение на одном графике нескольких параметрических кривых

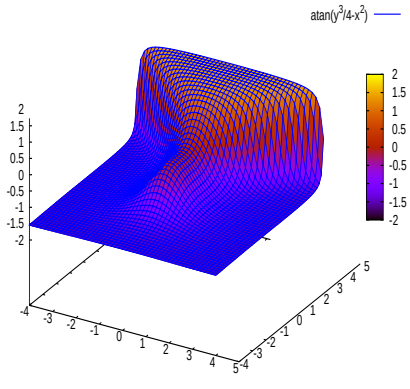


Рис. 3.8. График функции двух переменных с окраской поверхности

Поверхность функции в цветном изображении строится с использованием опции pm3d (рис. 3.8). Пример:

```
(%i2) plot3d (atan (-x^2 + y^3/4), [x, -4, 4], [y, -4, 4], [grid, 50, 50], [gnuplot_pm3d,
  true],[gnuplot_term,ps],[gnuplot_out_file,"plot31.eps"])$
```

С использованием этой опции и особенностей программы gnuplot можно построить и изображение линий уровня функции. Пример (рис.3.9):

```
(%i3) plot3d (cos (-x^2 + y^3/4), [x, -4, 4], [y, -4, 4],[gnuplot_preamble, "set view map;
  unset surface"], [gnuplot_pm3d, true], [grid, 150, 150],[gnuplot_term,ps],[gnuplot_out_file,"plot3
```

Более строгий результат можно получить, используя стандартный формат функции plot3d. Пример (рис. 3.10):

```
(%i4) plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2]);
```

Для вывода графика в файл всё равно необходимо использовать опции gnuplot (установить терминал gnuplot и имя файлп результата). Необходимая команда:

```
(%i5) plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2],[gnuplot_term,ps],[gnuplot_out_file,"plot33.eps
```

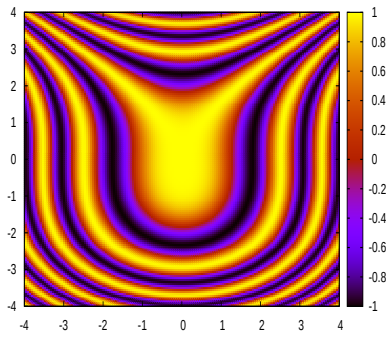


Рис. 3.9. График линий уровня функции двух переменных с окраской поверхности

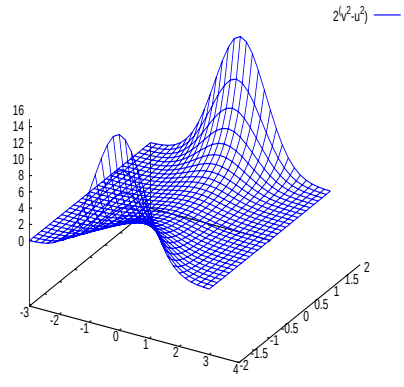


Рис. 3.10. Простой график функции двух переменных

Смена формата графики также возможна за счёт использования опций plot3d. Пример(вывод графики в формате openmath - рис. 3.11):

```
(%i6) plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2],[plot_format, openmath]);
```

Достоинством данного формата является встроенная возможность сохранения копии графического изображения в файл, редактирования и поворота построенного графика.

Функция, для которой строится трёхмерный график, может хадаваться как Maxima или Lisp-функция, лямбда-функция либо выражение Maxima общего вида. При использовании формата plot3d (f, ...) выражение f рассматривается как функция двух переменных. при использовании формата plot3d ([f_1, f_2, f_3], ...), каждая функция (f_1, f_2, f_3) рассматривается как функциея трёх переменных.

Пример использования формата plot3d ([f_1, f_2, f_3], ...) (рис. 3.12):

Функция plot3d позволяет строить графики функций, заданных в цилиндрических или сферических координатах за счёт использования преобразования координат (опция [transform_xy, polar_to_xy] или функция make_transform (vars, fx, fy, fz)).

Определённые преимущества обеспечивает формат wxplot (wxplot2d или wxplot3d). Команда построение графика в формате wxMaxima по синтаксису мало отличается от синтаксиса команд plot2d и plot3d. Качество воспроизведения графиков на экране wxMaxima относительно невысо-

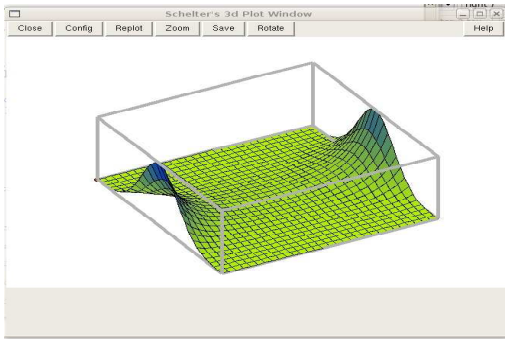


Рис. 3.11. Простой график функции двух переменных

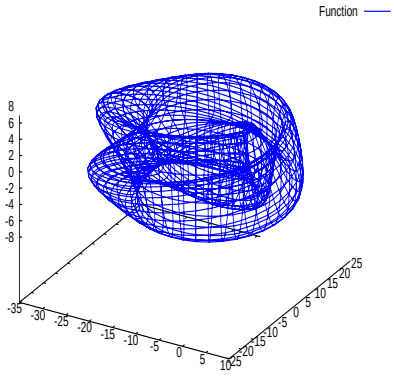


Рис. 3.12. График функции, определённой в формате [f_1, f_2, f_3]

кое, но легко, выделив график щелчком мыши, сохранить его в файл (по умолчанию maxout.png). Качество копии в файле намного лучше, чем рисунка в окне wxMaxima.

Глава 4

Задачи высшей математики с Maxima

4.1 Операции с комплексными числами

4.1.1 Представление комплексных чисел

Значение целой положительной степени комплексного аргумента, значение функции $f(z) = z^n$, проще всего вычислять в тригнометрической форме. Если $z = x + iy = r(\cos j + i \sin j)$, то для любого целого положительного числа n имеет место формула:

$$w = f(z) = z^n = r^n(\cos nj + i \sin nj)$$

. Если w = f (z) Значение целой положительной степени комплексного аргумента, значение функции $f(z) = z^n$, проще всего вычислять в тригнометрической форме. Если $z = x + iy = r(\cos j + i \sin j)$, то для любого целого положительного числа n имеет место формула: $w = f(z) = z^n = rn(\cos nj + i \sin nj)$. Если $w = f(z) = f(x + iy) = u(x, y) + iv(x, y)$, то $u(x, y) = rncosnj, u(x, y) = rnsinnj$. Корнем n-й степени из комплексного числа z называется число $w = \sqrt[n]{z}$ такое, что $w^n = z$. Для любого комплексного числа z существует n комплексных чисел w таких, что $w^n = z$. Значение корня, т.е. значение функции $f(z) = \sqrt[n]{z}$ проще всего вычислять в тригнометрической форме. Если $z = x + iy = r(\cos j + i \sin j)$, то для любого целого положительного числа n имеет место формула: $f(z) = \sqrt[n]{z} = \sqrt[n]{r\left(\cos\varphi + i\sin\varphi\right)} = \sqrt[n]{r}\left(\cos\frac{\varphi+2k\pi}{n} + \sin\frac{\varphi+2k\pi}{n}\right)$ Т.е. функция $f(z) = \sqrt[n]{z}$ является многозначной функцией - каждому значению аргумента отвечает n различных значений корня. Если $z = x + iy = r(\cos j + i \sin j)$, то значения функции f(z) = exp(z) вычисляются по формуле $f(z) = e^z = e^{x+iy} = e^x(\cos y + i \sin y)$.

Логарифмом комплексного числа z называется такое число w, что exp(w) = z. Значения логарифмической функции $f(z) = Ln(z)$ вычисляются по формуле $Ln(z) = ln(|z|) + iArgz = ln(|z|) + iargz + 2k\pi$, $k = 0, 1, 2, \ldots$ Величину $ln(|z|) + iargz$ называют главным значением логарифма. Функция $f(z) = Ln(z)$ является многозначной функцией - каждому значению аргумента отвечает бесконечное множество различных значений логарифма.

Комплексное выражение определено в Maxima посредством сложения действительной части выражения к произведению %i (мнимой едитницы) и мнимой части (т.е. в алгебраической форме). Например, корни из уравнения $x^2 - 4 * x + 13 = 0$ равны 2 + 3 * % i и 2 - 3 * %i. Решение в Maxima:

```
(%i1) eq:x^2-4*x+13=0;
```

(%o1) $$x^2 - 4\,x + 13 = 0$$

```
(%i2) solve(eq,x);
```

(%o2) $$[x = 2 - 3\,i, x = 3\,i + 2]$$

```
(%i3) x1:%o2[1]$  x2:%o2[2];
```

(%o4) $$x = 3\,i + 2$$

```
(%i5) print(x1,x2);
```

(%o5) $$x = 2 - 3\,ix = 3\,i + 2x = 3\,i + 2$$

Более сложный пример вычисления корней алгебраического уравнения n-й степени:

```
(%i1) solve(x^3=1,x);
```

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

```
(%i2) solve(x^5=1,x);
```

(%o2) $$[x = e^{\frac{2\,i\,\pi}{5}}, x = e^{\frac{4\,i\,\pi}{5}}, x = e^{-\frac{4\,i\,\pi}{5}}, x = e^{-\frac{2\,i\,\pi}{5}}, x = 1]$$

Количество корней, возвращаемое Maxima, соответствует основной теореме алгебры (уравнение третьей степени имеет три корня, пятой - пять и т.д.).

Преобразование комплексных выражений может осуществляться функциями для работы с алгебраическими выражениями (radcan, expand и др.), но предусмотрен и ряд специфических функций, рассчитанных на операции именно с комплексными числами.

4.1.2 Функции для работы с комплексными числами

Упрощение частных, корней, и других функций комплексных выражений может обычно достигаться при использовании функций realpart, imagpart, rectform, polarform, abs, carg.

Вычисление модуля комплексного числа осуществляется функцией cabs. Аргумент комплексного выражения вычисляется при помощи функции carg. Комплексный аргумент - θ в пределах $[-\pi, \pi]$ таким образом, что r exp (тета %i) = z где r - модуль комплексного числа z. Следует учитывать, что carg - вычислительная функция, не предназначенная для упрощения комплексных выражений. Пример:

```
(%i1) carg (1);
```

(%o1) $$0$$

```
(%i2) carg (1 + %i);
```

(%o2) $$\frac{\pi}{4}$$

```
(%i3) carg (exp (%i));
```

(%o3) $$1$$

```
(%i4) carg (exp (%pi * %i));
```

(%o4) π

```
(%i5) carg (exp (3/2 * %pi * %i));
```

(%o5) $-\frac{\pi}{2}$

Для преобразования комплексных выражений используют также функцию demoivre. Управление её работой осуществляется флагом demoivre.

Когда переменная demoivre установлена (demoivre = true), комплексные показательные функции преобразованы в эквивалентные выражения в терминах тригонометрических функций: exp (a + b*%i) упрощает к виду $\%e^a * (cos(b) + \%i * sin(b))$ если выражение b не содержит %i. Значение по умолчанию demoivre - false. Кроме того, преобразование различных форм комплексных чисел осуществляется функцией exponentialize, которая преобразует тригонометрические и гиперболические функции в экспоненциальную форму. Флаги demoivre и exponentialize не могут оба быть установлены в true одновременно.

Пример:

```
(%i1) demoivre:true;
```

(%o1) $true$

```
(%i2) demoivre (exp (3+3/2 * %pi * %i));
```

(%o2) $-e^3\, i$

```
(%i3) demoivre (exp (%pi+3/2 * %pi * %i));
```

(%o3) $-e^\pi\, i$

Комплексно-сопряжённые выражения вычисляются при помощи функции conjugate (x). Пример:

```
(%i1) declare ([aa, bb], real, cc, complex, ii, imaginary);
```

(%o1) $done$

```
(%i2) conjugate (aa + bb*%i);
```

(%o2) $aa - i\, bb$

```
(%i3) conjugate (ii);
```

(%o3) $-ii$

Как видно из примера, функция declare позволяет объявить тип выражений: действительные, комплексные и число мнимые (imaginary).

Функция plog (x) представляет основную ветвь комплексного логарифма, соответствующую -%pi < carg(x) <= +%pi, например:

```
(%i1) a:1+%i;
```

(%o1) $i+1$

```
(%i2) plog(a);
```

(%o2) $$\frac{log\,(2)}{2}+\frac{i\,\pi}{4}$$

Функция polarform (expr) возвращает выражение r %e(%i theta), эквивалентное expr (параметры r и theta действительны). Преобразование комплексного выражения к алгебраической форме осуществляется функцией rectform(x). Пример:

```
(%i1) a:1+%i;
```

(%o1) $i+1$

```
(%i2) polarform(a);
```

(%o2) $\sqrt{2}\,e^{\frac{i\,\pi}{4}}$

```
(%i3) rectform(%);
```

(%o3) $i+1$

Функция residue (expr, z, z_0) вычисляет остаток в комплексной плоскости для выражения expr , когда переменная z принимает значение z_0. Остаток - коэффициент при $(z-z_0)^(-1)$ ряда Лорана для expr. Пример:

```
(%i1) residue (s/(s**2+a**2), s, a*%i);
```

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

```
(%i2) residue (sin(a*x)/x**4, x, 0);
```

(%o2) $$-\frac{a^3}{6}$$

4.2 Задачи линейной алгебры

Пакет Maxima включает большое число функций для решения разнообразных задач линейной алгебры.

Рассмотрим основные функции, позволяющие оперировать матрицами и решать основные задачи линейной алгебры.

4.2.1 Простейшие операции с матрицами

В Maxima на матрицах определены обычные операции умножения на число, сложения и матричного умножения. Последнее реализуется с помощью бинарной операции "."(точка). Размерности матриц-сомножителей должны быть согласованы.

Рассмотрим несколько примеров. Создание двух прямоугольных матриц:

```
(%i1) a:matrix([1,2,3],[4,5,6]);
```

$$(\%o1)\qquad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
(%i2) b:matrix([2,2],[3,3],[4,4]);
```

$$(\%o2)\qquad \begin{pmatrix} 2 & 2 \\ 3 & 3 \\ 4 & 4 \end{pmatrix}$$

Функция transpose транспонирует матрицу:

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

$$(\%o2)\qquad \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Умножение матрицы на число:

```
(%i3) c:2*b;
```

$$(\%o3)\qquad \begin{pmatrix} 4 & 4 \\ 6 & 6 \\ 8 & 8 \end{pmatrix}$$

Сложение матриц (естественно, матрицы должны быть одинаковой формы, иначе возникает ошибка):

```
(%i4) d:b+c;
```

$$(\%o4)\qquad \begin{pmatrix} 6 & 6 \\ 9 & 9 \\ 12 & 12 \end{pmatrix}$$

```
(%i5) a+b;
```

$$`fullmap'foundargumentswithincompatiblestructure.--anerror.Todebugthistrydebugmode(true);$$

Умножение матриц (в данном случаем исходные матрицы a и b согласованы по размерам):

```
(%i6) f:a.b;
```

$$(\%o6)\qquad \begin{pmatrix} 20 & 20 \\ 47 & 47 \end{pmatrix}$$

Если матрица - левый сомножитель, то правым сомножителем может быть не только векстор-столбец, но и вектор-строка и даже список.

Maxima позволяет также возводить матрицы в степень, но фактически эта операция применяется к каждому элементу.

4.2.2 Обращение матриц и вычисление определителей

Для обращения матриц используется функция invert. Пример:

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

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

```
(%i2) invert(a);
```

$$(\%o2)\qquad \begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

```
(%i3) %.a;
```

$$(\%o3)\qquad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Определитель вычисляется соответствующей функцией (determinant):

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

$$(\%o4)\qquad -2$$

4.2.3 Характеристический полином, собственные числа и собственные векторы матрицы

Характеристический полином матрицы вычислячеься функцией charpoly(M,x) (M - матрица , x - переменная, относительно которой строится полином). Пример:

```
(%i6) charpoly(a,x);
```

$$(\%o6) \qquad (1-x)\,(4-x)-6$$

```
(%i7) ratsimp(%);
```

$$(\%o7) \qquad x^2-5\,x-2$$

Корни характеристического полинома являются собственнми числами матрицы.

Однако для вычисления собственных чисел и собственных векторов матрицы обычно используют специальные функции: eigenvalues и eigenvectors.

Функция eigenvalues вычисляет собственные значения матрицы аналитически, если это возможно. Для нахождения корней характеристического полинома используется функция solve. Результат, возвращаемый eigenvalues - список, содержащий два подсписка. Первый содержит собственные значения, а второй — их кратности. Пример:

Функция eigenvectors аналитически вычисляет собственные значения и собственные вектора матрицы, если это возможно. Она возвращает список, первый элемент которого — список собсвенных чисел (аналогично "eigenvalues"), а далее идут собственные вектора, каждый из которых представлен как список своих проекций. Пример:

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

$$(\%o1) \qquad \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$$

```
(%i2) eigenvalues(a);
```

$$(\%o2) \qquad [[0,6],[2,1]]$$

```
(%i3) eigenvectors(a);
```

$$(\%o3) \qquad [[[0,6],[2,1]],[1,0,-1],[0,1,-1],[1,2,3]]$$

Функция uniteigenvectors отличается от функции "eigenvectors"тем, что возвращает нормированные на единицу собственные векторы.

4.2.4 Ортогонализация

Maxima включает специальную функцию для вычисления ортонормированного набора векторов из заданного. Используется стандартный алгоритм Грама-Шмидта. Синтаксис вызова: gramschmidt(x) или gschmidt(x). Аргумент функции - матрица или список. В качестве компонентов системы векторов, на базе которой строится ортонормированная система, рассматриваются строки матрицы x или подсписки списка x. Для использования данной функции необходимо явно загрузить пакет "eigen". Пример:

```
(%i1) load("eigen");
```

(%o1) $/usr/share/maxima/5.13.0/share/matrix/eigen.mac$

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

(%o2)
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
(%i3) y:gramschmidt(x);
```

(%o3) $$[[1,2,3],[\frac{2^2\,3}{7},\frac{3}{7},-\frac{2\,3}{7}]]$$

```
(%i4) ratsimp(%[1].%[2]);
```

(%o4) $$0$$

4.2.5 Преобразование матрицы к треугольной форме

Преобразование матрицы к треугольной форме осуществляется методом исключения Гаусса посредством функции echelon(M) (аналогияный результат дает функция triangularize(M)):

```
(%i1) a:matrix([1,2,3],[4,5,x],[6,7,y]);
```

(%o1)
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & x \\ 6 & 7 & y \end{pmatrix}$$

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

(%o2)
$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & -\frac{x-12}{3} \\ 0 & 0 & 1 \end{pmatrix}$$

Отличия рассматриваемых функций в том, что echelon нормирует диагональный элемент на 1, а triangularize - нет. Обе функции используют алгоритм исключения Гаусса.

4.2.6 Вычисление ранга и миноров матрицы

Для расчёта ранга матрицы (порядка наибольшегоо невырожденного минора матрицы) используется функция rank. Пример:

```
(%i1) a:matrix([1,2,3,4],[2,5,6,9]);
```

Матрица a - невырожденная (две строки, ранг равен 2). Вычислим ранг вырожденной матрицы, содержащей линейно-зависимые строки.

(%o1)
$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 9 \end{pmatrix}$$

```
(%i2) rank(a);
```

(%o2) 2

```
(%i3) b:matrix([1,1],[2,2],[3,3],[4,5]);
```

(%o3)
$$\begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 5 \end{pmatrix}$$

```
(%i4) rank(b);
```

(%o4) 2

```
(%i5) echelon(b);
```

(%o5)
$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Минор матрицы вычисляется при помощи функции minor(M,i,j), где M - матрица, i,j - индексы элемента, для которого вычисляется минор.

4.2.7 Решение матричных уравнений

Пусть дано матричное уравнение AX=B, где A – квадратная матрица размерности n; B – матрица размерности n×k; X – неизвестная матрица размерности n×k. Пусть A – невырожденная матрица (т.е. det(A) $\neq 0$, тогда существует единственное решение этого уравнения. Решение можно найти по формуле $X = A^{-1}B$

Пример. Найти решение матричного уравнений AX=B, где

$$A = \begin{bmatrix} 1 & 2 & 2 \\ -1 & -1 & 3 \\ 2 & 5 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 10 & 0 \\ -2 & 5 \\ 1 & 4 \end{bmatrix}.$$

Сначала зададим матрицы A и B:

```
(%i1) A: matrix( [1, 2, 2],[ -1, -1, 3], [2, 5, 0]);
```

$$(\%o1) \qquad \begin{pmatrix} 1 & 2 & 2 \\ -1 & -1 & 3 \\ 2 & 5 & 0 \end{pmatrix}$$

```
(%i2) B:matrix([10, 0],[-2, 5], [1, 4]);
```

$$(\%o2) \qquad \begin{pmatrix} 10 & 0 \\ -2 & 5 \\ 1 & 4 \end{pmatrix}$$

Проверим существование и единственность решения:

```
(%i3) determinant(A);
```

$$(\%o3) \qquad -9$$

Матрица A невырожденная, значит, решение существует и единственно. Найдем его:

```
(%i4) A1:invert(A);
```

$$(\%o4) \qquad \begin{pmatrix} \frac{5}{3} & -\frac{10}{9} & -\frac{8}{9} \\ -\frac{2}{3} & \frac{4}{9} & \frac{5}{9} \\ \frac{1}{3} & \frac{1}{9} & -\frac{1}{9} \end{pmatrix}$$

```
(%i5) X:A1.B;
```

$$(\%o5) \qquad \begin{pmatrix} 18 & -\frac{82}{9} \\ -7 & \frac{40}{9} \\ 3 & \frac{1}{9} \end{pmatrix}$$

Выполним проверку:

```
(%i6) A.X;
```

$$(\%o6) \qquad \begin{pmatrix} 10 & 0 \\ -2 & 5 \\ 1 & 4 \end{pmatrix}$$

Аналогично решается матричное уравнение XA=B, где A – квадратная матрица размерности n, B – матрица размерности k × n, X – неизвестная матрица размерности k × n. Если A – невырожденная матрица, то существует единственное решение $X = BA^{-1}$.

Пример. Найти решение X матичного уравнений XA=C, где матрица A из предыдущей задачи, C - заданная матрица. Аналогично предыдущему примеру, вычисляем решение:

```
(%i7) C:matrix([10,0,-2],[5,1,4]);
```

(%o7)
$$\begin{pmatrix} 10 & 0 & -2 \\ 5 & 1 & 4 \end{pmatrix}$$

```
(%i8) X:C.A1;
```

(%o8)
$$\begin{pmatrix} 16 & -\frac{34}{3} & -\frac{26}{3} \\ 9 & -\frac{14}{3} & -\frac{13}{3} \end{pmatrix}$$

```
(%i9) X.A;
```

(%o9)
$$\begin{pmatrix} 10 & 0 & -2 \\ 5 & 1 & 4 \end{pmatrix}$$

В общем случае (когда A – вырожденная матрица, или A – не квадратная матрица) матричное уравнение AX=B можно решить при помощи функции solve. Синтаксис вызова: solve([eq1, eq2...,eqn], [x1,x2,...,xm]), где [eq1, eq2...,eqn] — список уравнений, [x1,x2,...,xm] —список неизвестных, относительно которых осуществляется решение.

4.2.8 Специальные функции для решения систем линейных и полиномиальных уравнений

Функция linsolve([expr_1, expr_2, ..., expr_m], [x_1, x_2,...,x_n]) решает список одновременных линейных уравнений [expr_1, expr_2, ..., expr_m] относительно списка переменных [x_1,...,x_n]. Выражения $[\mathrm{expr}_1, \ldots, \mathrm{expr}_n]$ могут быть полиномами указанных переменных и представляться в виде уравнений. Пример. Решить системы линейных уравнений $\begin{cases} x + y + z + t = 6, \\ 2x - 2y + z + 3t = 2, \\ 3x - y + 2z - t = 8. \end{cases}$

Решение в Maxima:

```
(%i1) ex1:x+y+z+t=6; ex2:2*x-2*y+z+3*t=2; ex3:3*x-y+2*z-t=8;  linsolve([ex1,ex2,ex3],[x,y,z,t]);
```

(%o4)
$$z+y+x+t=6 z-2 y+2 x+3 t=22 z-y+3 x-t=8[x=-\frac{3\,\%r1-14}{4}, y=-\frac{\%r1-10}{4}, z=\%r1, t=0]$$

Таким образом общее решение имеет вид: x=(–14+3c)/4, y= (c - 10)/4, z = c, t = 0, где c – произвольная постоянная. Ей можно задавать произвольные действительные значения. При каждом значении c получается частное решение. Например, при c = 1 получается частное решение

```
(%i5) ev(%),%r1=1;
```

(%o5)
$$[x=\frac{11}{4}, y=\frac{9}{4}, z=1, t=0]$$

Способ представления решения определяется флагом linsolve_params (по умолчанию true) Если указанный флаг установлен в true, решение недоопределённых систем включает параметры %r1, %r2 и т.д. Если флаг linsolve_params установлен в false, связанные переменные выражаются через свободные.

Во многом аналогичный результат позволяет получить функция algsys (фактически, это надстройка над solve). Функция algsys([expr_1, expr_2, ..., expr_m], [x_1, x_2,...,x_n]) решает список одновременных полиномиальных уравнений [expr_1=0, expr_2=0, ..., expr_m=0] относительно списка переменных [x_1,...,x_n]. Выражения $[\mathrm{expr}_1, \ldots, \mathrm{expr}_n]$ могут быть представлены и в виде уравнений. Количество уравнений может превышать количество неизвестных, или наоборот. Пример:

```
(%i6) e1: 2*x*(1 - a1) - 2*(x - 1)*a2;  e2: a2 - a1;
e3: a1*(-y - x^2 + 1);  e4: a2*(y - (x - 1)^2);
```

(%o9) $$2\,(1-a1)\,x-2\,a2\,(x-1)\,a2-a1a1\,\left(-y-x^2+1\right)a2\,\left(y-(x-1)^2\right)$$

```
(%i10) algsys ([e1, e2, e3, e4], [x, y, a1, a2]);
```

(%o10) $$[[x=0, y=\%r2, a1=0, a2=0], [x=1, y=0, a1=1, a2=1]]$$

Для вычисления корней единичных полиномиальных уравнений используется функция realroots. Варианты синтаксиса: realroots (expr, bound); realroots (eqn, bound); realroots (expr); realroots (eqn). Функция находит все корни выражения expr=0 или уравнения eqn. Функция строит последовательность Штурма для изоляции каждого корня и использует алгоритм деления пополам для уточнения корня с точностью bound или с точностью, заданной по умолчанию. Пример:

```
(%i11) realroots (2 - x + x^5, 5e-6);
```

(%o11) $$[x=-\frac{664361}{524288}]$$

```
(%i12) float(%);
```

(%o12) $$[x=-1.267168045043945]$$

```
(%i13) ev(2-x+x^5,%[1]);
```

(%o13) $$3.0858501665065319\,10^{-6}$$

Все корни полинома (действительные и комплексные) можно найти при помощи функции allroots. Способ представления решения определяется переменной polyfactor (по умолчанию false; если установить в true, то функция возвращает результат факторизации). Алгоритм поиска корней получисленный. Пример:

```
(%i1) eqn:x^4+1;  soln:allroots (eqn);
```

(%o2)
$x^4+1[x=0.70710678118655\,i+0.70710678118655, x=0.70710678118655-0.70710678118655\,i, x=0.70710678118655\,i-0.$

```
(%i3) polyfactor:true$ eqn:x^4+1;  soln:allroots (eqn);
```

(%o5) $$x^4 + 11.0\,\left(x^2 - 1.414213562373095\,x + 1.0\right)\,\left(x^2 + 1.414213562373095\,x + 1.0\right)$$

Количество днйствительных корней уравнения в некотором интервале возвращает функция nroots (синтаксис nroots (p, low, high)). Пример (находим число корней уравнения на отрезке [-6,9]):

```
(%i1) p: x^10 - 2*x^4 + 1/2$ nroots (p, -6, 9);
```

(%o2) 4

Для преобразования уравнений используются функции lhs и rhs, позволяющие выделить левую и правую часть уравнения соответственно. Пример:

```
(%i1) eqn:x^2+x+1=(x-1)^3;
```

(%o1) $$x^2 + x + 1 = (x - 1)^3$$

```
(%i2) lhs(eqn);
```

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

```
(%i3) rhs(eqn);
```

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

Упрощение систем уравнений достигается функцией eliminate, позволяющей исключить те или иные переменные. Вызов eliminate ([eqn_1, ..., eqn_n], [x_1, ..., x_k]) исключает переменные [x_1, ..., x_k] из указанных выражений. Пример:

```
(%i1) expr1: 2*x^2 + y*x + z; expr2: 3*x + 5*y - z - 1;  expr3: z^2 + x - y^2 + 5;
```

(%o3) $$z + x\,y + 2\,x^2 - z + 5\,y + 3\,x - 1z^2 - y^2 + x + 5$$

```
(%i4) eliminate ([expr3, expr2, expr1], [y, z]);
```

(%o4) $$[7425\,x^8 - 1170\,x^7 + 1299\,x^6 + 12076\,x^5 + 22887\,x^4 - 5154\,x^3 - 1291\,x^2 + 7688\,x + 15376]$$

4.2.9 Классификация и основные свойства функций

Функция называется *явной* (или *заданной в явном виде*), если она задана формулой, в которой правая часть не содержит зависимой переменной; например, функция $y = x^3 + 7x + 5$.

Функция y аргумента x называется *неявной* (или *заданной в неявном виде*), если она задана уравнением $F(x, y) = 0$, не разрешенным относительно зависимой переменной. Например, функция $y(y \geq 0)$, заданная уравненим $x^3+y^2-x = 0$. Отметим, что последнее уравнение задает две функции, $y = \sqrt{x - x^3}$ при $y \geq 0$, и $y = -\sqrt{x - x^3}$ при $y < 0$.

Обратная функция. Пусть $y = f(x)$ есть функция от независимой переменной x, определенной на промежутке X с областью значений Y. Поставим в соответствие каждому $y \in Y$ *единственное* значение $x \in X$, при котором $f(x) = y$. Тогда полученная функция $x = g(y)$, определенная на промежутке Y с областью значений X называется *обратной* по отношению к функции $y = f(x)$.

Например, для функции $y = a^x$ обратной будет функция $x = \log_a x$.

Сложная функция. Пусть функция $y = f(u)$ есть функция от переменной u, определенной на множестве U с областью значений Y, а переменная u в свою очередь является функцией $u = \phi(x)$ от переменной x, определенной на множестве X с областью значений U. Тогда заданная на множестве X функция $y = f[\phi(x)]$ называется *сложной* функцией.

Например, $y = \sin x^5$ – сложная функция, так как ее можно представить в виде $y = \sin u$, где $u = x^5$.

Понятие элементарной функции. *Основными элементарными функциями* являются

а) степенная функция $y = x^r$, $r \in R$;

б) показательная функция $y = a^x$ $(a > 0, a \neq 1)$;

в) логарифмическая функция $y = \log_a x$ $(a > 0, a \neq 1)$;

г) тригонометрические функции $y = \sin x$, $y = \cos x$, $y = \operatorname{tg} x$, $y = \operatorname{ctg} x$;

д) обратные тригонометрические функции $y = \arcsin x$, $y = \arccos x$, $y = \operatorname{arctg} x$, $y = \operatorname{arcctg} x$.

Из основных элементарных функций новые *элементарные* функции могут быть получены при помощи: а) алгебраических действий; б) операций образования сложных функций.

Определение. *Функции, построенные из основных элементарных функций с помощью конечного числа алгебраических действий и конечного числа операций образования сложной функции, называются* ***элементарными***.

Например, функций

$$y = \frac{\sqrt{x} + \arcsin x^5}{\ln^3 x + x^3 + x^7}$$

является элементарной.

Примером неэлементарной функции является функция $y = signx$.

4.2.9.1 Основные свойства функций

1. Четность и нечетность. Функция $y = f(x)$ называется *четной*, если $f(-x) = f(x)$ и *нечетной*, если $f(-x) = -f(x)$. В противном случае фукция называется *общего вида*.

Например, функция $y = x^2$ является четной, а функция $y = x^3$ – нечетной. Функция $y = x^2 + x^3$ является функцией общего вида.

График четной функции симметричен относительно оси ординат, а график нечетной функции симметричен относительно начала координат.

2. Монотонность. Функция $y = f(x)$ называется *монотонно возрастающей (убывающей)* на промежутке X, если для любых x_1, x_2 $(x_1, x_2 \in X)$ и $x_2 > x_1$ выполняется неравенство $f(x_2) > f(x_1)$ $(f(x_2) < f(x_1))$. А если выполняется неравенство $f(x_2) \geq f(x_1)$ $(f(x_2) \leq f(x_1))$, то функция называется неубывающей (невозрастающей).

3. Ограниченность. Функция $y = f(x)$ называется *ограниченной* на промежутке X, если существует такое положительное число $M > 0$, что $|f(x)| \leq M$ для любого $x \in X$.

Например, функция $y = \sin x$ ограничена на всей числовой оси, так как $|\sin x| \leq 1$ для любого $x \in R$.

4. Периодичность. Функция $y = f(x)$ называется *периодической* с периодом $T \neq 0$ на промежутке X, для любого $x \in X$ выполняется равенство $f(x + T) = f(x)$.

4.2.10 Предел функции и его свойства

4.2.10.1 Предел функции в бесконечности

Определение. Число A называется пределом **функции $f(x)$ при x, стремящемся к бесконечности**, если для любого сколь угодно малого положительного числа $\epsilon > 0$, найдется такое положительное число $\delta > 0$, что для всех x удовлетворяющих условию $|x| > \delta$ выполняется неравенство $|f(x) - A| < \epsilon$.

Этот предел функции обозначается следующим образом:

$$\lim_{x \to \infty} f(x) = A$$

или $f(x) \to A$ при $x \to \infty$.

4.2.10.2 Предел функции в точке

Пусть функция $y = f(x)$ определена в некоторой окрестности точки a, кроме, быть может, самой точки a.

Определение. Число A называется пределом **функции $f(x)$ при x, стремящемся к a (или в точке a),** если для любого сколь угодно малого положительного числа $\epsilon > 0$, найдется такое положительное число $\delta > 0$, что для всех x, удовлетворяющих условию $0 < |x - a| < \delta$ выполняется неравенство $|f(x) - A| < \epsilon$. Условие $0 < |x - a|$ означает, что $x \neq 0$.

Предел функции обозначается следующим образом:

$$\lim_{x \to a} f(x) = A$$

или $f(x) \to A$ при $x \to a$.

4.2.10.3 Односторонние пределы.

Если $x > a$ и $x \to a$, то употребляют запись $x \to a + 0$. Если $x < a$ и $x \to a$, то употребляют запись $x \to a - 0$.

Выражения $\lim\limits_{x \to a+0} f(x)$ и $\lim\limits_{x \to a-0} f(x)$ называются соответственно пределами функции $f(x)$ в точке a справа и слева.

Если существует предел $\lim\limits_{x \to a} f(x)$, то существуют пределы $\lim\limits_{x \to a+0} f(x)$ и $\lim\limits_{x \to a-0} f(x)$ и

$$\lim_{x \to a+0} f(x) = \lim_{x \to a-0} f(x) = \lim_{x \to a} f(x).$$

Это равенство выполняется также, если пределы слева и справа равны.

4.2.10.4 Теоремы о пределах

1. Предел суммы двух функций равен сумме пределов этих функций, если те существуют, то есть

$$\lim_{x \to x_0} [f(x) + \psi(x)] = A + B,$$

где $A = \lim\limits_{x \to x_0} f(x)$, $B = \lim\limits_{x \to x_0} \psi(x)$.

2. Предел произведения двух функций равен произведению пределов этих функций.

$$\lim_{x \to x_0} [f(x) \cdot \psi(x)] = A \cdot B.$$

3. Предел частного двух функций равен частному пределов этих функций.

$$\lim_{x \to x_0} \frac{f(x)}{\psi(x)} = \frac{A}{B},$$

причем $B \neq 0$.

4. Если,

$$\lim_{u \to u_0} f(u) = A;$$

$$\lim_{x \to x_0} \psi(x) = B,$$

то предел сложной функции

$$\lim_{x \to x_0} f[\psi(x)] = A.$$

4.2.11 Вычисление пределов различных классов функций

Предел выражения $f(x)$ при $x \to a$ вычисляется с помощью функции
$limit(f(x), x, a)$;
Рассмотрим пример: вычислить предел $\lim_{x \to 0} \frac{\sin x}{x}$.

Решение: выполним команду

```
(%i1) limit(sin(x)/x, x, 0);
```

Результат на экране:

(%o1) 1

Более сложные варианты вычисления пределов иллюстрирует следующие несколько примеров, включающие пределы слева, справа, при стремлении к бесконечности и т.п. Рассмотрим пределы:
$\lim_{x \to +\infty} e^x$,
$\lim_{x \to -\infty} e^x$,
$\lim_{x \to 0-0} \frac{1}{x}$,
$\lim_{x \to 0+0} \frac{1}{x}$.

Предел неограниченной функции на бесконечности:

```
(%i2) limit (exp(x), x, inf);
```

(%o2) ∞

```
(%i3) limit (exp(x), x, minf);
```

(%o3) 0

Пределы при $x \to 0$ слева и справа:

```
(%i3) limit(1/x, x, 0, minus);
```

(%o3) $-\infty$

```
(%i4) limit(1/x, x, 0, plus);
```

(%o4) ∞

4.2.11.1 Предел и непрерывность функции

Вычислить пределы $\lim\limits_{x\to 8}\sqrt[3]{x}$ и $\lim\limits_{x\to -8}\sqrt[3]{x}$.

```
(%i8) limit(x^(1/3), x, 8);
```

(%o8) 2

```
(%i9) limit(x^(1/3), x, -8);
```

(%o9) -2

4.2.11.2 Пределы рациональных дробей

Вычислить предел $\lim\limits_{x\to -1}\frac{x^3-3x-2}{(x^2-x-2)^2}$.

```
(%i10) y(x):=(x^3-3*x-2)/(x^2-x-2)^2; limit(y(x), x,-1);
```

(%o10) $$y\left(x\right):=\frac{x^3-3\,x-2}{\left(x^2-x-2\right)^2}$$

(%o11) $$-\frac{1}{3}$$

При операциях с рациональными дробями и выделения носителй нуля целесообразно использовать факторизацию выражений, например: вычисление предела непосредственно

```
(%i16) limit((x^2-4)/(x^2-3*x+2), x, 2);
```

(%o16) 4

Вычисление предела после факторизации рационального выражения:

```
(%i17) factor((x^2-4)/(x^2-3*x+2));
```

В числителе и знаменателе дроби сокращается носитель нуля при $x \to 2$, т.е. выражение $x - 2$.

(%o17) $$\frac{x+2}{x-1}$$

```
(%i18) limit(%,x,2);
```

(%o18) 4

4.2.11.3 Пределы, содержащие иррациональные выражения

Вычисление пределов данного класса во многом аналогично вычислению пределов рациональных дробей, т.к. сводится к сокращению носителей нуля в числителе и знаменателе анализируемых выражений, например: Вычислить предел выражения

$$\frac{\sqrt{x}-1}{x-1}$$

при $x \to 1$. При вычислении предела непосредственно имеем:

```
(%i1) limit((sqrt(x)-1)/(x-1), x, 1);
```

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

Для упрощения и сокращения носителей нуля используется функция radcan:

```
(%i2) factor((sqrt(x)-1)/(x-1));
```

$$(\%o2) \qquad \frac{\sqrt{x}-1}{x-1}$$

```
(%i3) radcan(%);
```

$$(\%o3) \qquad \frac{1}{\sqrt{x}+1}$$

```
(%i4) limit(%, x, 1);
```

$$(\%o4) \qquad \frac{1}{2}$$

4.2.11.4 Пределы тригонометрических выражений

Первым замечательным пределом называется предел

$$\lim_{x\to 0}\frac{\sin x}{x}=1.$$

Рассмотрим примеры нахождения некоторых пределов с использованием первого замечательного предела.

Пример. Найти предел

$$\lim_{x\to 0}\frac{\sin 5x}{x}$$

.

$$\lim_{x\to 0}\frac{\sin 5x}{x}=\lim_{x\to 0}5\frac{\sin 5x}{5x}=5\lim_{t\to 0}\frac{\sin t}{t}=5,$$

где $t=5x$.

Расчёт с использованием Maxima:

```
(%i1) limit(sin(5*x)/x, x, 0);
```

(%o1) 5

Пример. Найти предел $\lim\limits_{x\to 0}\dfrac{1-\cos 2x}{x^2}$.

$$\lim_{x\to 0}\frac{1-\cos 2x}{x^2}=\lim_{x\to 0}\frac{2\sin x^2}{x^2}=\lim_{t\to 0}\frac{2\sin t}{t}=2,$$

где $t=x^2$. Расчёт с использованием Maxima:

```
(%i4) limit((1-cos(2*x))/x^2, x, 0);
```

(%o4) 2

4.2.11.5 Пределы экспоненциальных выражений

Вторым замечательным пределом называется предел

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x=e=2.718281828\cdots$$

, равный

Можно показать, что функция

$$y(x)=\left(1+\frac{1}{x}\right)^x$$

при $x\to+\infty$ и при $x\to-\infty$ таже имеет предел, равный e.

$$e=\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x.$$

Заменяя x на $x=1/t$ получим еще одну запись числа e

$$e=\lim_{t\to 0}(1+t)^{1/t}.$$

Число e (число Эйлера или неперово число) играет важную роль в математическом анализе.

Функция $y=e^x$ носит название *экспоненты.* Если показатель экспоненты громоздкий, то ее принято записывать в виде: $\exp(x)$.

Логарифм по основанию e называтся натуральным. Его обозначают символом ln, т.е. $\log_e x=\ln x$.

Важную роль в математическом анализе играют также *гиперболические функции* (*гиперболический синус, гиперболический косинус, гиперболический тангенс*), определяемые формулами:

$$\operatorname{sh}(x)=\frac{\exp(x)-\exp(-x)}{2};$$

$$\operatorname{ch}(x)=\frac{\exp(x)+\exp(-x)}{2};$$

$$\operatorname{th}(x)=\frac{\operatorname{sh}(x)}{\operatorname{ch}(x)}.$$

Рассмотрим примеры нахождения некоторых пределов с использованием второго замечательного предела. **Пример.** Найти предел $\lim\limits_{x\to 0}\dfrac{\ln(1+x)}{x}$.

$$\lim_{x\to 0}\frac{\ln(1+x)}{x}=\lim_{x\to 0}\ln(1+x)^{1/x}=\lim_{x\to 0}\ln e=1.$$

Итак $\lim\limits_{x\to 0}\dfrac{\ln(1+x)}{x}=1$.

Пример. Найти предел $\lim\limits_{x\to 0}\dfrac{a^x-1}{x}$.

Пусть $a^x-1=u$. Тогда $a^x=1+u$; $x=\dfrac{\ln(1+u)}{\ln a}$.

$$\lim_{x\to 0}\frac{a^x-1}{x}=\lim_{u\to 0}\frac{u\ln a}{\ln(1+u}=\ln a\cdot\lim_{u\to 0}\frac{u}{\ln(1+u)}=\ln a\cdot 1=\ln a.$$

Итак $\lim\limits_{x\to 0}\dfrac{a^x-1}{x}=\ln a$.

Вычисление при помощи Maxima:

```
(%i5) limit(log(1+x)/x, x, 0);
```

(%o5) 1

```
(%i6) limit((a^x-1)/x, x, 0);
```

(%o6) $log\,(a)$

Найдем предел $\lim\limits_{x\to\infty}\left(\dfrac{x}{2+x}\right)^{3x}$. Аналитический расчт даёт следующий результат:

$$\lim_{x\to\infty}\left(\frac{x}{2+x}\right)^{3x}=\exp\left[\lim_{x\to\infty}\left(\frac{x}{2+x}-1\right)3x\right]=e^{-6}.$$

Используя Maxima, получаем:

```
(%i7) limit((x/(2+x))^(3*x), x, inf);
```

(%o7) e^{-6}

4.2.11.6 Бесконечно малые и бесконечно большие функции

Сравнение бесконечно малых функций Рассмотрим предел частного от деления двух бесконечно малых $\alpha(x)$ и $\beta(x)$ при $x\to a$.

Предел отношения двух бесконечно малых величин $A=\lim\limits_{x\to a}\dfrac{\alpha(x)}{\beta(x)}$ может быть равен нулю, конечному числу или ∞.

1. Если A конечно, то $\alpha(x)$ и $\beta(x)$ называют бесконечно малыми одного порядка и пишут $\alpha(x)=O[\beta(x)]$ при $x\to a$.

Если $A=1$, то $\alpha(x)$ и $\beta(x)$ называют эквивалентными и пишут $\alpha(x)\sim\beta(x)$ при $x\to a$.

2. Если $A=0$, то $\alpha(x)$ называют бесконечно малой более высокого порядка, чем $\beta(x)$ и пишут $\alpha(x)=o[\beta(x)]$ при $x\to a$.

Если существует действительное число $r>0$ такое, что

$$\lim_{x\to a}\frac{\alpha(x)}{[\beta(x)]^r}\neq 0$$

то $\alpha(x)$ называют бесконечно малой порядка r относительно $\beta(x)$ при $x\to a$.

3. Если $A \to \infty$ при $x \to a$,то в этом случае $\beta(x)$ называют бесконечно малой более высокого порядка, чем $\alpha(x)$ и пишут $\beta(x) = o[\alpha(x)]$.

Конечно, может случиться, что отношение двух бесконечно малых не стремиться ни к какому пределу; например, если взять $\alpha = x$ и $\beta = x \sin \frac{1}{x}$, то их отношение, равное $\sin \frac{1}{x}$, равное $x \to 0$ предела не имеет. В таком случае говорят, что две бесконечно малые не сравнимы между собой.

Пример вычислений с Maxima:

Рассмотрим две бесконечно малые функции при $x \to 0$

```
(%i26) f(x):=sin(3*x)*sin(5*x)$ g(x):=(x-x^3)^2$(%i28) limit(f(x)/g(x), x, 0);
```

Вычислим предел отношения $f(x)/g(x)$ при $x \to 0$

```
(%i28) limit(f(x)/g(x), x, 0);
```

(%o28) 15

Результат, равный постоянному числу, свидительствует о том, что рассматриваемые бесконечно малые одного порядка.

Эквивалентные бесконечно малые. Их применение к вычислению пределов При вычислении пределов полезно иметь в виду эквивалентность следующих бесконечно малых величин:

$$\sin x \sim x; \operatorname{tg} x \sim x; \operatorname{arcsin} x \sim x; \operatorname{arctg} x \sim x; \ln(1+x) \sim x,$$

при $x \to 0$.

Их несложно получить, используя правило Лопиталя (см. ниже).

Пример:

Сравнить бесконечно малые $\alpha(x) = x^2 \sin^2 x$ и $\beta(x) = x \operatorname{tg} x$ при $x \to 0$.

Заменим $\sin^2 x$ и $\operatorname{tg} x$ на их эквивалентные бесконечно малые $\sin^2 x \sim x^2$ и $\operatorname{tg} x \sim x$. Получим

$$\lim_{x\to 0} \frac{\alpha(x)}{\beta(x)} = \lim_{x\to 0} \frac{x^2 \sin^2 x}{x \operatorname{tg} x} = \lim_{x\to 0} \frac{x^2 \cdot x^2}{x \cdot x} = \lim_{x\to 0} \frac{x^4}{x^2} = \lim_{x\to 0} x^2 = 0.$$

Таким образом, $\alpha(t) = o[\beta(t)]$ при $t \to 0$. Кроме того, $\alpha(x)$ является бесконечно малой порядка 2 относительно $\beta(x)$.

Пример. Определить порядок малости $\alpha(x) = \sin(\sqrt{x+1}-1)$ относительно $\beta(x) = x$ при $x \to 0$.

Так как

$$\sqrt{x+1} - 1 = \frac{(\sqrt{x+1}-1)(\sqrt{x+1}+1)}{(\sqrt{x+1}+1)} = \frac{x}{(\sqrt{x+1}+1)}$$

то

$$\lim_{x\to 0} \frac{\alpha(x)}{\beta(x)} = \lim_{x\to 0} \left[\frac{1}{x} \sin\left(\frac{x}{\sqrt{x+1}+1}\right)\right] = \frac{1}{2}.$$

При вычислениях с использованием Maxima более естественно использовать при вычислении сложных пределов и сравнении бесконечно малых разложение числителя и знаменателя в ряд Тейлора (подробное обсуждение степенных рядов - см. ниже) При вычислении с использованием меню во вкладке меню Анализ → Найти предел, установить пункт "Использовать ряд Тейлора". Для вычислений используется функция tlimit, работа которой основана на замене исследуемых функций рядом Тейлора (где это возможно). По умолчанию флаг замены установлен в *false*, поэтому для использования tlimit флаг замены устанавливается в *true* :

```
(%i1) tlimswitch=true;
```

(%o1) $false = true$

пример вычисления с использованием tlimit:

```
(%i1) f(x):=(tan(x)-sin(x))/(x-sin(x));
```

$$(\%o1) \qquad f(x) := \frac{tan(x) - sin(x)}{x - sin(x)}$$

```
(%i2) tlimit(f(x),x,0);
```

$$(\%o2) \qquad 3$$

Бесконечно большие функции. Связь между бесконечно малыми и бесконечно большими величинами Функция $f(x)$ называется бесконечно большой величиной при $x \to a$, если для любого $\epsilon > 0$ найдётся такое $\delta > 0$, что для всех x, удовлетворяющих условию $0 < |x - a| < \delta$, будет выполнено неравенство $|f(x)| > \epsilon$.

Запись того, что функция $f(x)$ бесконечно большая при $x \to a$ означает следующее: $\lim\limits_{x \to a} f(x) = \infty$ или $f(x) \to \infty$ при $x \to a$.

Пример: $y = \operatorname{tg} x$ бесконечно большая при $x \to \pi/2$.

Замечание: Функция может быть неограниченной, но не бесконечно большая. Например, функция $y = x \sin x$ – не ограничена на $(-\infty, \infty)$, но не бесконечно большая при $x \to \infty$.

Если функция $\alpha(x)$ есть бесконечно малая величина при $x \to a (x \to \infty)$, то функция $f(x) = \dfrac{1}{\alpha(x)}$ является бесконечно большой при $x \to a (x \to \infty)$.

И обратно, если функция $f(x)$ бесконечно большая при $x \to a (x \to \infty)$, то функция $\alpha(x) = \dfrac{1}{f(x)}$ есть величина бесконечно малая при $x \to a (x \to \infty)$.

Например, функция $y = \cos x$ – бесконечно малая при $x \to \pi/2$, тогда функция $\dfrac{1}{\cos x}$ – бесконечно большая. Функция $y = \dfrac{1}{2x - 7}$ - бесконечно малая при $x \to \infty$, тогда функция $y = 2x - 7$ - бесконечно большая при $x \to \infty$.

Непрерывность функции в точке Понятие непрерывности функции, так же как и понятие предела, является одним из основных понятий математического анализа.

Дадим два определения понятия непрерывности функции в точке.

Определение 1. *Функция $f(x)$ называется* ***непрерывной в точке*** *a, если она удовлетворяет трем условиям: 1) $f(x)$ определена в некоторой окрестности точки $x = a$, 2) существует конечный предел $\lim\limits_{x \to a} f(x)$, 3) этот предел равен значению функции $f(x)$ в точке a, т.е.* $\lim\limits_{x \to a} f(x) = f(a)$. Очевидно, что непрерывность функции в данной точке выражается непрерывностью ее графика при прохождении данной точки.

Расмотрим второе определение непрерывности функции в точке.

Придадим аргументу a приращение $\Delta x \neq 0$. Тогда функция $y = f(x)$ получит приращение Δy, определяемое как разность наращенного и исходного значения функции: $\Delta y = f(a + \Delta x) - f(a)$ (см. рис. 1).

Определение 2. *Функция $y = f(x)$ называется* **непрерывной в точке** a, если она определена в некоторой окрестности точки $x = a$, и приращение ее Δy в этой точке, соответствующее приращению Δx, стремится к нулю при стремлении Δx к нулю:

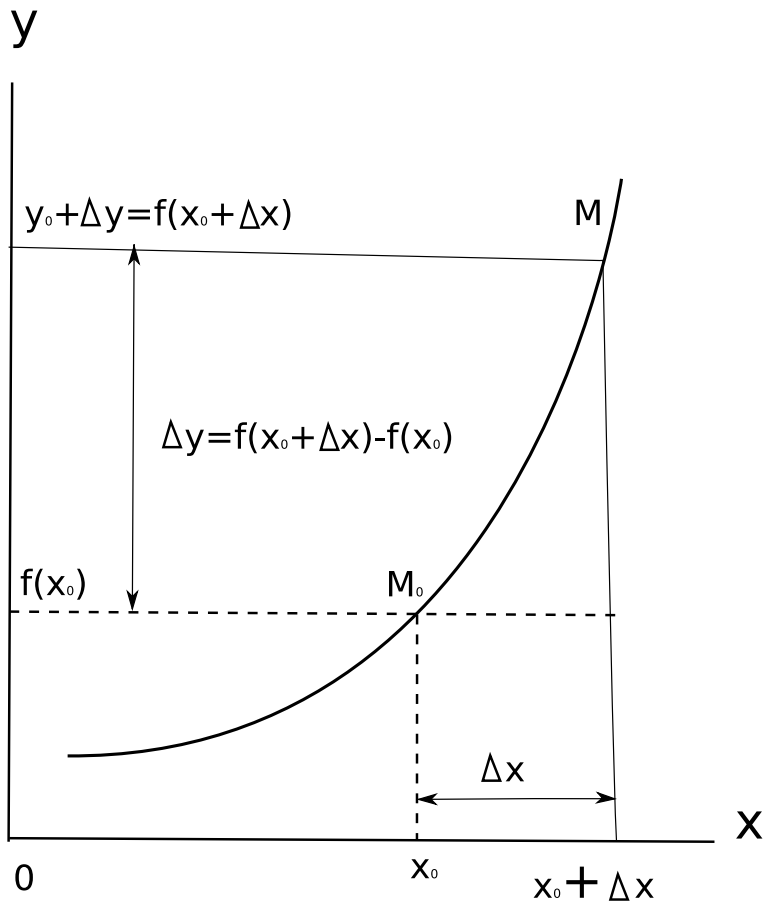
$\lim\limits_{\Delta x \to 0} \Delta y = 0$.

В руководствах по математическому анализу доказывается, что оба определения равносильны.

Пример исследования непрерывности функции с Maxima:

Функция

$$f(x) := \frac{1}{1 + exp\left(\frac{1}{1-x}\right)}$$



имеет возможную точку разрыва при $x = 1$. Сопоставим пределы данной функции при стремлении x к 1 слева и справа:

```
(%i16) f(x):=1/(1+exp(1/(1-x)));
```

(%o16) $$f(x) := \frac{1}{1 + exp\left(\frac{1}{1-x}\right)}$$

```
(%i17) limit(f(x),x,1,plus);
```

(%o17) 1

```
(%i18) limit(f(x),x,1,minus);
```

(%o18) 0

Пределы не совпадают, поэтому делаем вывод, что исследуемая функция разрывна.

Свойства непрерывных функций **1.** *Если функция* $f(x)$ *и* $g(x)$ *непрерывны в точке* a*, то их сумма*

$$f(x) + g(x),$$

произведение

$$f(x)g(x)$$

и частное

$$\frac{f(x)}{g(x)}$$

(при условии, что $g(a) \neq 0$) являются функциями, непрерывными в точке a.

2. *Если функция $y = f(x)$ непрерывна в точке a и $f(a) > 0$, то существует такая окрестность точки a, в которой $f(x) > 0$.*

3. *Если функция $y = f(u)$ непрерывна в точке u_0, а функция $u = \psi(x)$ непрерывна в точке $u_0 = \psi(x_0)$, то сложная функция $y = f[\psi(x)]$ непрерывна в точке $_0$.*

Свойство 3 может быть записано в виде:

$$\lim_{x \to x_0} f[\psi(x)] = f\left[\lim_{x \to x_0} \psi(x)\right],$$

т.е. *под знаком непрерывной функции можно переходить к пределу.*

Функция $y = f(x)$ называется *непрерывной на промежутке X*, если она непрерывна в каждой точке этого промежутка. Можно доказать, что все элементарные функции непрерывны в области их определения.

Точки разрыва функций и их классификация Точка a, принадлежащая области определения функции или являющаяся граничной для этой области, называется точкой разрыва функции $f(x)$, если в этой точке нарушается условие непрерывности функции.

Если существуют конечные пределы

$$f(a-0) = \lim_{x \to a-0} f(x) \text{ и } f(a+0) = \lim_{x \to a+0} f(x),$$

причем не все три числа $f(a)$, $f(a-0)$, $f(a+0)$ равны между собой, то точка a называется *точкой разрыва 1 рода* (существуют конечные односторонние пределы функции слева и справа при, не равные друг другу).

Точки разрыва 1 рода подразделяются, в свою очередь, на *точки устранимого разрыва* (когда $f(a-0) = f(a+0) \neq f(a)$, т.е. когда левый и правый пределы функции $f(x)$ в точке a равны между собой, но не равны значению функции $f(x)$ в этой точке) и на *точки скачка* (когда $f(a-0) \neq f(a+0)$, т.е. когда левый и правый пределы функции в точке a различны); в последнем случае разность $f(a+0) - f(a-0)$ называется *скачком* функции $f(x)$ в точке a.

Точки разрыва, не являющиеся точками разрыва 1 рода, называются *точками разрыва 2 рода.* В точках разрыва 2 рода не существует хотя бы один из односторонних пределов.

Рассмотрим предыдущий пример. Функция

$$f(x) := \frac{1}{1 + exp\left(\frac{1}{1-x}\right)}$$

имеет точку разрыва при $x = 1$.

Так как пределы

$$\lim_{x \to 0-0} f(x)$$

и

$$\lim_{x \to 0+0} f(x)$$

не совпадают, но оба конечны, делаем вывод о наличии точки разрыва первого рода при $x = 1$.

Графическую иллюстрацию получаем при помощи wxMaxima

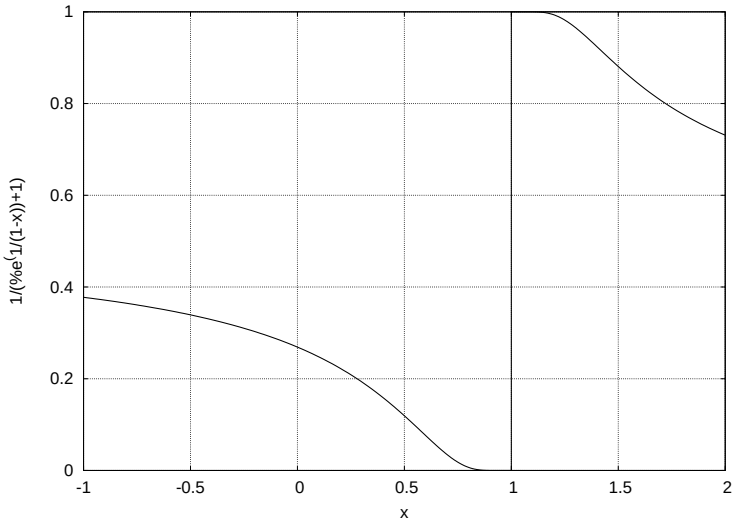


Рис. 4.1. Разрыв исследуемой функции

4.2.12 Дифференцирование с помощью пакета Maxima

Пакет Maxima предоставляет мощные средства для дифференцирования функций и вычисления дифференциалов. Для вычисления простейшей производной следует в командном окне после приглашения Maxima ввести команду следующего вида: diff(<функция>,<переменная>); где <функция> – выражение, задающее функцию (не обязательно одной переменной), например $x^2+2*x+1$; <переменная> – имя переменной, по которой будет вестись дифференцирование, например x

Примером вычисления производной может служить такая команда: $diff(x^2+2*x+1,x)$;

С помощью команды diff можно вычислять производные высших порядков. При этом команда имеет следующий формат: diff(<функция>,<переменная>,<порядок>); где <порядок> - порядок вычисляемой производной.

В решениях некоторых примеров этой главы с помощью MAXIMA будут использованы дополнительные команды MAXIMA:

- ratsim(<выражение >), radcan(<выражение >)- упрощение алгебраического вырежения.
- trigsim(<выражение >), trigexpand(<выражение >)- упрощение или подстановка тригонометрического вырежения.
- factor(<выражение>); – разложить <выражение> на множители.
- at(<выражение>,<old>=<new>); – подставить выражение <new> на место <old> в <выражении>.
- <переменная>:solve(<выр1>=<значение>,<выр2>); – присвоить <переменной> значение выражения <выр2>, полученное разрешением уравнения <выр1>(<выр2>)=<значение>.
- taylor(<f(x)>,x,<x0>,<n>); – разложить функцию f(x) по формуле Тейлора с центром в точке x0 до порядка n включительно.

4.2.12.1 Вычисление производных и дифференциалов

Для вычисления производной функции используется функция diff, для вычисления производных различного порядка удобно создать пользовательскую функцию (в примере ниже - f(x)):

```
(%i3) f(x):=sin(9*x^2);
```

(%o3) $$f(x) := sin\left(9\,x^2\right)$$

```
(%i4) d1:diff(f(x),x,1);
```

(%o4) $$18\,x\,cos\left(9\,x^2\right)$$

```
(%i5) d2:diff(f(x),x,2);
```

(%o5) $$18\,cos\left(9\,x^2\right) - 324\,x^2\,sin\left(9\,x^2\right)$$

```
(%i6) d3:diff(f(x),x,3);
```

(%o6) $$-972\,x\,sin\left(9\,x^2\right) - 5832\,x^3\,cos\left(9\,x^2\right)$$

Пример вычисления дифференциала (del(x) равноценно dx, не указана явно переменная дифференцирования):

```
(%i8) diff(log(x));
```

(%o8) $$\frac{del\,(x)}{x}$$

Аналогичный подход применим и для функции нескольких переменных. Функция diff с единственным аргументом - дифференцируемой функцией - возвращает полный дифференциал.

Пример:

```
(%i9) diff(exp(x*y));
```

(%o9) $$x\,e^{x\,y}\,del\,(y) + y\,e^{x\,y}\,del\,(x)$$

Пример:

```
(%i10) diff(exp(x*y*z));
```

(%o10) $$x\,y\,e^{x\,y\,z}\,del\,(z) + x\,z\,e^{x\,y\,z}\,del\,(y) + y\,z\,e^{x\,y\,z}\,del\,(x)$$

Если указать апостроф перед символом diff, то производная не вычисляется и упрощение, обычно предусмотренное по умолчанию, не осуществляется.

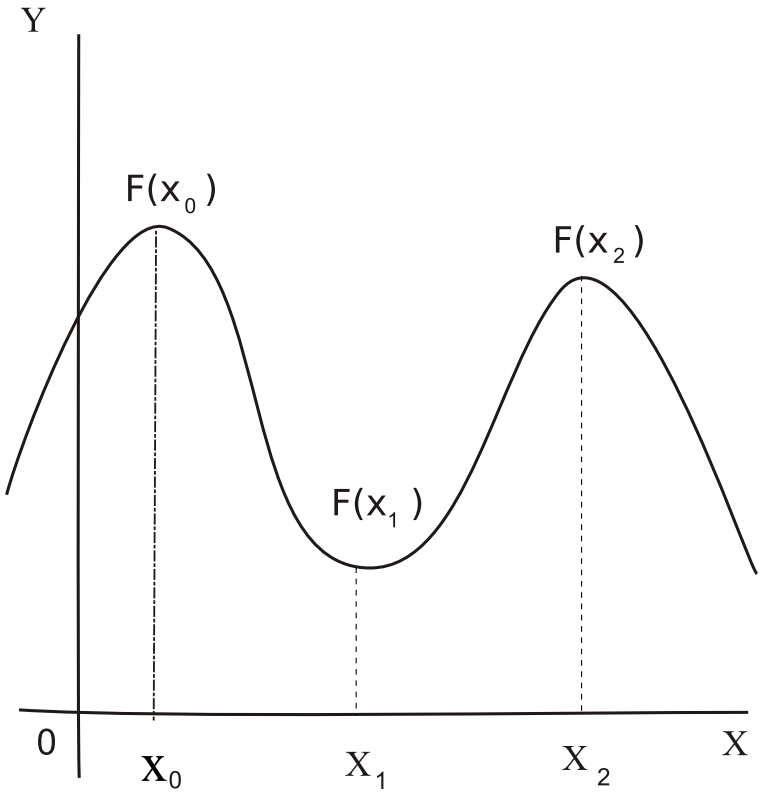
Пример:

Создаём функцию $f(x, z)$:

```
(%i18) f(x,z):=x^2*z+z^2*x;
```

(%o18) $$f(x,z) := x^2\,z + z^2\,x$$

Вычисляем дифференциалное выражение:




```
(%i19) diff (f(x,z), x, 2) + diff (f(x,z), z, 3) + diff (f(x,z), x) * x^2;
```

(%o19) $$x^2 \left(z^2 + 2\,x\,z\right) + 2\,z$$

Производим формальное диференцирование, не вычисляя непосредственно результат:

```
(%i20) 'diff (f(x,z), x, 2) + 'diff (f(x,z), z, 3) + 'diff (f(x,z), x) * x^2;
```

(%o20) $$\frac{d^3}{d\,z^3}\left(x\,z^2 + x^2\,z\right) + \frac{d^2}{d\,x^2}\left(x\,z^2 + x^2\,z\right) + x^2\left(\frac{d}{d\,x}\left(x\,z^2 + x^2\,z\right)\right)$$

4.3 Экстремумы функций

4.3.0.2 Отыскание максимумов и минимумов

Необходимое условие экстремума. Теорема Ферма Точки, где достигается наибольшее или наименьшее значение функции называются соответственно точками максимума или минимума функции.

Определение 1. *Точка x_0 называется точкой* ***максимума*** *функции $f(x)$, если в некоторой окрестности точки x_0 выполняется неравенство $f(x) \geq f(x_0)$* (см. рис. 1).

Определение 2. *Точка x_1 называется точкой* ***минимума*** *функции $f(x)$, если в некоторой окрестности точки x_1 выполняется неравенство $f(x) \leq f(x_1)$* (см. рис. 1).

Значения функции в точках x_0 и x_1 называются соответственно *максимумом и минимумом функции*. Максимум и минимум функции объединяются общим названием *экстремума функции*.

Экстремум функции часто называют *локальным* экстремумом, подчеркивая тем самым,что понятие экстремума связано лишь с достаточно малой окрестностью точки x_0. Так что на одном промежутке функция может иметь несколько экстремумов, причем может случиться так, что минимум в одной точке больше максимума в другой, например, на рис.1 $f_{min}(x_3) > f_{max}(x_0)$. Наличие

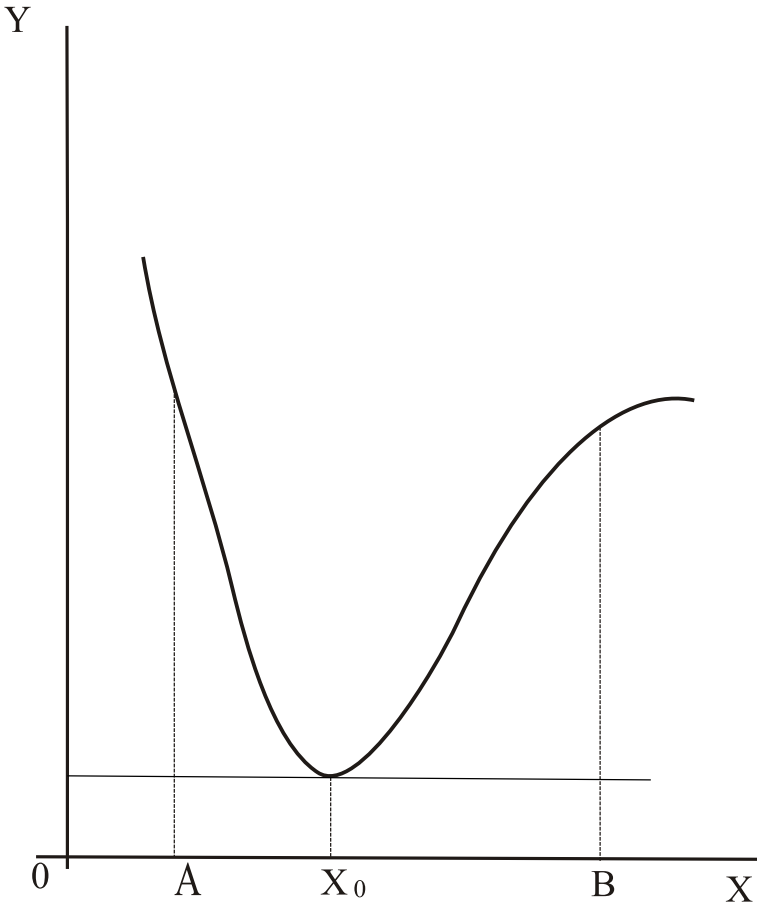


Рис. 4.2. Иллюстрация теоремы Ферма

максимума (или минимума) в отдельной точке промежутка X вовсе не означает, что в этой точке функция $f(x)$ принимает наибольшее (наименьшее) значение на этом промежутке (или, как говорят имеет глобальный максимум (минимум)).

Если дифференцируемая на промежутке X функция $y = f(x)$ достигает наибольшего или наименьшего значения в внутренней точке x_0,то тогда производная функции в этой точке равна нулю, т.е. $f'(x_0) = 0$.

Пусть функция $y = f(x)$ дифференцируема на промежутке X и в точке $x_0 \in X$ принимает наименьшее значение (см. рис. 2).

Тогда

$$f(x_0 + \Delta x) \geq f(x_0)$$

если $x_0 + \Delta x \in X$ и, следовательно

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \geq 0$$

при достаточно малых Δx и независимо от знака Δx.

Поэтому

$$\frac{\Delta y}{\Delta x} \geq 0 \text{ при } \Delta x > 0 \text{ (справа от } x_0);$$

$$\frac{\Delta y}{\Delta x} \leq 0 \text{ при } \Delta x < 0 \text{ (слева от } x_0).$$

Переходя к пределу справа и слева получим

$$\lim_{\Delta x\to 0+}\frac{\Delta y}{\Delta x}\geq 0 \text{ и } \lim_{\Delta x\to 0-}\frac{\Delta y}{\Delta x}\leq 0.$$

Так как функция дифференцируема на промежутке X, то пределы справа и слева равны

$$\lim_{\Delta x\to 0+}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0-}\frac{\Delta y}{\Delta x}.$$

Отсюда $f'(x_0)=0$.

Аналогичную последовательность рассуждений можно построить и для максимума.

Теорему Ферма часто называют необходимым условием экстремума **дифференцируемой** функции.

Геометрический смысл теоремы Ферма: *в точке экстремума, достигаемого внутри промежутка X, касательная к графику функции параллельна оси абсцисс.*

Необходимое условие экстремума Если в точке x_0 дифференцируемая функция $f(x)$ имеет экстремум, то в некоторой окрестности этой точки выполняются условия теоремы Ферма, и следовательно, производная функции в этой точке равна нулю, т.е. $f'(x_0)=0$. Но функция может иметь экстремум и в точках, в которых она не дифференцируема. Так, например, функция $y=|x|$ имеет экстремум (минимум) в точке $x=0$, но не дифференцируема в ней. Функция $y=\sqrt[3]{x^2}$ также имеет в точке $x=0$ минимум, а ее производная в этой точке бесконечна: $y'=\dfrac{2}{3\sqrt[3]{x}}$, $y'(0)=\infty$.

Поэтому необходимое условие экстремума может быть сформулировано следующим образом.

Для того чтобы функция $y=f(x)$ имела экстремум в точке x_0, необходимо, чтобы ее производная в этой точке равнялась нулю ($f'(x_0)=0$) или не существовала.

Точки, в которых выполнено необходимое условие экстремума, называются *критическими* (или *стационарными*). Но *критическая точка не обязательно является точкой экстремума.*

Пример. Найти критические точки функции и убедиться в наличии или отсутствии экстремума в этих точках:

$$1.y=x^2+1;\quad 2.y=x^3-1.$$

1.$y'=2x$. $y'(x)=0$ при $x=0$. В точке $x=0$ функция $y=x^2+1$ имеет минимум.

2. $y'=3x^2$. $y'(x)=0$ при $x=0$. В точке $x=0$ функция $y=x^3-1$ не имеет экстремума. Функция $y=x^3-1$ возрастает на всей числовой оси.

Итак, для нахождения экстремумов функции требуется дополнительное исследование критических точек.

Пример: Исследовать на наличие экстремума следующую функцию

$$y(x)=x^3-3*x^2+3*x+2$$

Задаём исследуемую функцию

```
(%i1) f(x):=x^3-3*x^2+3*x+2;
```

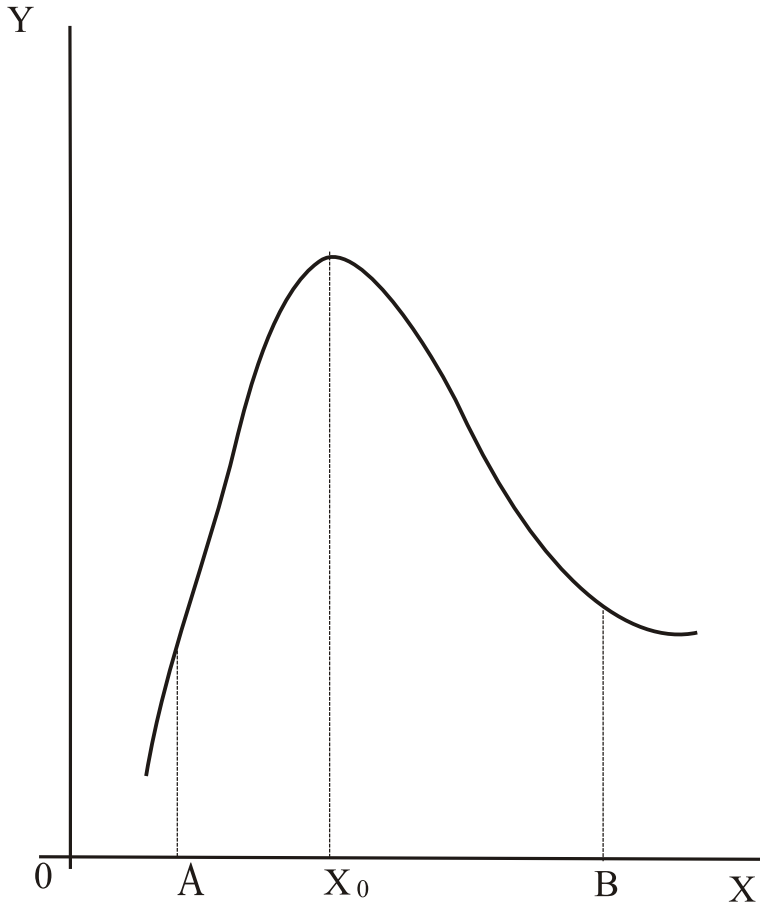
(%o1) $$f(x):=x^3-3\,x^2+3\,x+2$$

Производную в форме функции определяем **явно**, используя функцию ***define***

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

(%o2) $$df(x):=3\,x^2-6\,x+3$$

Решая уравнение $df(x)=0$ (т.е. $f'(x)=0$, находим критические точки




```
(%i3) solve(df(x)=0,x);
```

(%o3) $[x = 1]$

В данном случае критическая точка одна - $x = 1$.

Первое достаточное условие экстремума **Теорема.** *Если при переходе через точку x_0 производная дифференцируемой функции $y = f(x)$ меняет свой знак с плюса на минус, то точка x_0 есть точка максимума функции $y = f(x)$, а если с минуса на плюс, то – точка минимума.*

Пусть производная меняет знак с плюса на минус, т.е. в некотором интервале (a, x_0) производная положительна ($f'(x) > 0$), а в некотором интервале (x_0, b) – отрицательна ($f'(x) < 0$) (см. рис. 6). Тогда в соответствии с достаточным условием монотонности функция $f(x)$ возрастает на интервале (a, x_0) и убывает на интервале (x_0, b).

По определению возрастающей функции $f(x_0) \geq f(x)$ при всех $x \in (a, x_0)$, а по определению убывающей функции $f(x) \leq f(x_0)$ при всех $x \in (x_0, b)$, т.е. $f(x_0) \geq f(x)$ при всех $x \in (a, b)$, следовательно, x_0 – точка максимума функции $y = f(x)$.

Аналогично рассматривается случай, когда производная меняет знак с минуса на плюс.

Отметим, что дифференцируемость функции в самой точке x_0 не использовалась при доказательстве теоремы. На самом деле она и не требуется – достаточно, чтобы функция была непрерывна в точке x_0.

Если изменение знака производной не происходит, то экстремума нет. Однако при работе с системами компьютерной математики удобнее второе достаточное условие экстремума.

Второе достаточное условие экстремума **Теорема.** *Если первая производная $f'(x)$ дважды дифференцируемой функции $y = f(x)$ равна нулю в некоторой точке x_0, а вторая производная в этой точке $f''(x_0)$ положительна, то x_0 есть точка максимума функции $y = f(x)$; если $f''(x_0)$ отрицательна, то x_0 – точка максимума.*

Пусть $f'(x_0) = 0$, а $f''(x_0) > 0$. Это значит, что

$$f''(x) = (f'(x))' > 0$$

также и в некоторой окрестности точки x_0, т.е. $f'(x)$ возрастает на некотором интервале (a, b), содержащем точку x_0.

Но $f'(x_0) = 0$, следовательно, на интервале (a, x_0) $f'(x) < 0$, а на интервале (x_0, b) $f'(x) > 0$, т.е. $f'(x)$ при переходе через точку x_0 меняет знак с минуса на плюс, т.е. x_0 – точка минимума.

Аналогично рассматривается случай $f'(x_0) = 0$ и $f''(x_0) < 0$.

Продолжим исследование функции

$$y(x) = x^3 - 3 * x^2 + 3 * x + 2$$

Как установлено выше, имеется одна критическая точка: $x = 1$.

Задаёмся функцией $d2f(x)$

```
(%i4) define(d2f(x),diff(df(x),x));
```

(%o4) $$d2f\,(x) := 6\,x - 6$$

Вычисляем значение второй производной в критической точке:

```
(%i5) map(d2f,%o3);
```

(%o5) $$[6\,x - 6 = 0]$$

В данном примере невозможно определить, является ли точка $x = 1$ экстремумом исследуемой функции, т.к. вторая производная в ней оказалась равной 0. Следует обратить внимание на способ вычисления - функция $d2f(x)$ применяется ко всем элементам списка, полученного при решении уравненния $f'(x) = 0$ (используется встроенная функция Maxima **map**).

Воспользуемся первым достаточным признаком наличия экстремума

```
(%i6) df(0);
```

(%o6) 3

```
(%i7) df(2);
```

(%o7) 3

Как видно из приведенного результата, первая производная не изменяет знак в критической точке, что свидетельствует об отсутствии экстремума в ней.

Полученный результат иллюстрируется графиком исследуемой функции и её производных :

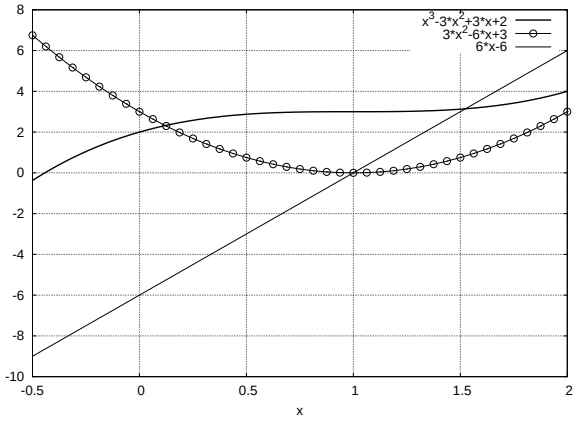


Рис. 4.3. Пример исследования функции

Схема исследования функции $y = f(x)$ на экстремум // 1. Найти производную $y' = f'(x)$.

2. Найти критические точки функции, в которых производная $f'(x) = 0$ или не существует.

3.1. Исследовать знак производной слева и справа от каждой критической точки и сделать вывод о наличии экстремумов функции.

Или

3.2. Найти вторую производную $f''(x)$ и определить ее знак в каждой критической точке.

4. Найти экстремумы (экстремальные значения) функции.

Пример. Исследовать на экстремум функцию $y = x(x-1)^3$.

1. $y' = (x-1)^3 + 3x(x-1)^2 = (x-1)^2(4x-1)$.

2. Критические точки $x_1 = 1$ и $x_2 = \dfrac{1}{4}$.

3. Изменение знака производной при переходе через точку x_1 не происходит, поэтому в этой точке нет экстремума.

$y'' = 2(x-1)(4x-1) + 4(x-1)^2 = 2[(x-1)(6x-3)]$.

$y''(x_2) > 0$, поэтому в этой точке наблюдается минимум функции $y = x(x-1)^3$.

4. $y_{min} = y\left(\dfrac{1}{4}\right) = -\dfrac{27}{256}$.

Выполним тот же расчёт при помощи Maxima

```
(%i13) f(x):=x*(x-1)^3;
```

$$(\%o13) \qquad f(x) := x\,(x-1)^3$$

```
(%i14) define(df(x),diff(f(x),x));
```

$$(\%o14) \qquad df(x) := 3\,(x-1)^2\,x + (x-1)^3$$

```
(%i15) solve(df(x)=0,x);
```

$$(\%o15) \qquad \left[x = \frac{1}{4}, x = 1\right]$$

```
(%i16) define(d2f(x),diff(df(x),x));
```

$$(\%o16) \qquad d2f(x) := 6\,(x-1)\,x + 6\,(x-1)^2$$

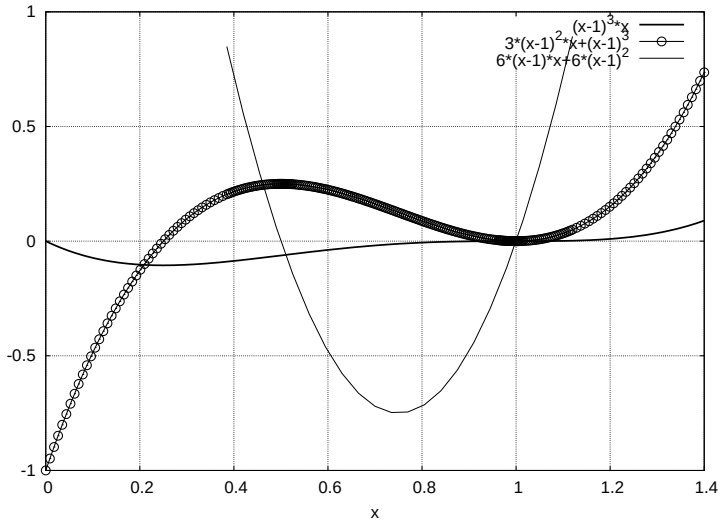


Рис. 4.4. Пример исследования функции на экстремум

```
(%i17) map(d2f,%o15);
```

$$(\%o17) \qquad [6\ (x-1)\ x+6\,(x-1)^2=\frac{9}{4}, 6\ (x-1)\ x+6\,(x-1)^2=0]$$

В точке $x=1$ вторая производная равна 0, поэтому вычисляем значения первой производной слева и справа от $x=1$:

```
(%i18) df(2);
```

$$(\%o18) \qquad 7$$

```
(%i19) df(1/3);
```

$$(\%o19) \qquad \frac{4}{27}$$

Производная в окрестности точки $x=1$ не меняет знак, поэтому экстремум у исследуемой функции один - точка $x=\frac{1}{4}$. Так как $d2f(\frac{1}{4})>0$, $x=\frac{1}{4}$ - точка минимума. Иллюстрация полученного результата - на рисунке.

Нахождение наибольших и наименьших значений функции Наибольшее или наименьшее значение функции на некотором отрезке может достигаться как в точках экстремума, так и в точках на концах отрезка.

Пусть функция $y=f(x)$ определена на некотором отрезке $[a,b]$.

Нахождение наибольших и наименьших значений функций происходит по следующей схеме.

1. Найти производную $f'(x)$.

2. Найти критические точки функции, в которых $f'(x_0)=0$ или не существует.

3. Найти значения функции в критических точках и на концах отрезка и выбрать из них наибольшее f_{MAX} и наименьшее f_{MIN} значения. Это и будут наибольшее и наименьшее значение функции на исследуемом отрезке.

Пример. Найти наибольшее и наименьшее значения функции $y=3x^2-6x$ на отрезке $[0,3]$.

Аналитический расчёт:
1. $y' = 6x - 6$;$y'' = 6$.
2. $x_0 = 1$.
3. $y(1) = -3$; $y(0) = 0$; $y(3) = 9$.
В точке $x = 1$ наименьшее значение функции, а в точке $x = 3$ – наибольшее.
Расчёт с использованием Maxima:
Находим критические точки исследуемой функции

```
(%i29) f(x):=3*x^2-6*x;
```

(%o29) $$f\left(x\right) := 3\,x^2 - 6\,x$$

```
(%i30) define(df(x),diff(f(x),x));
```

(%o30) $$df\left(x\right) := 6\,x - 6$$

```
(%i31) solve(df(x)=0,x);
```

(%o31) $$[x = 1]$$

Результат расчёта - список, включающий один элемент ($[x = 1]$).
Создаём новый список, включающий граничные значений и критические точки:

```
(%i32) L:[%o31[1],x=0,x=3];
```

(%o32) $$[x = 1, x = 0, x = 3]$$

Применяем функцию $f(x)$ к каждому элементу списка L:

```
(%i33) map(f,L);
```

(%o33) $$[3\,x^2 - 6\,x = -3, 3\,x^2 - 6\,x = 0, 3\,x^2 - 6\,x = 9]$$

Результат - наибольшие и наименьшие значения - находим в списке полученных значений.

Выпуклость функции **Определение.** *График функции $y = f(x)$ называется* ***выпуклым*** *в интервале (a, b), если он расположен ниже касательной, проведенной в любой точке этого интервала* (см. рис. 7а).

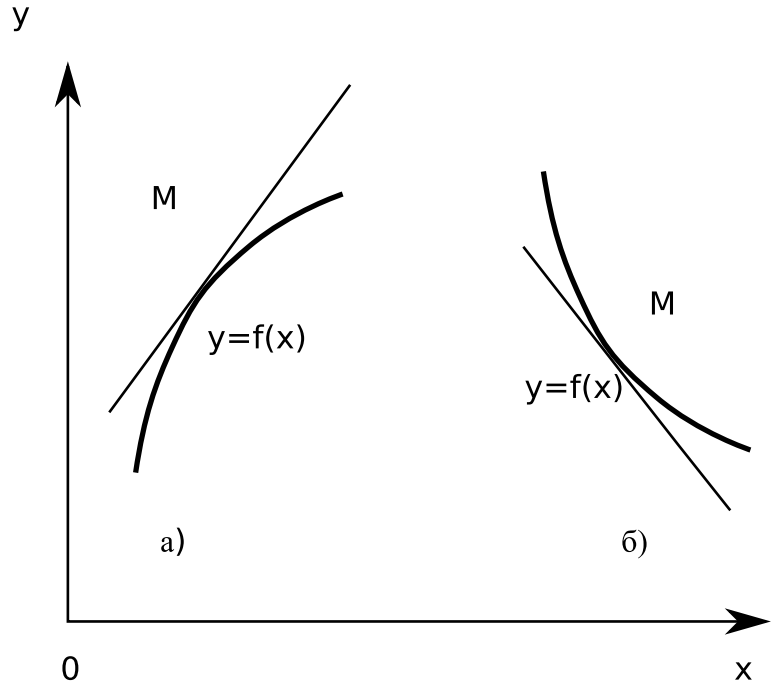
График функции $y = f(x)$ называется ***вогнутым*** *в интервале (a, b), если он расположен выше касательной, проведенной в любой точке этого интервала* (см. рис. 7б).

ßНеобходимые и достаточные условия выпуклости (вогнутости) функции Для определения выпуклости (вогнутости) функции на некотором интервале можно использовать следующие теоремы.

Теорема 1. *Пусть функция $f(x)$ определена и непрерывна на интервале X и имеет конечную производную $f'(x)$. Для того, чтобы функция $f(x)$ была выпуклой (вогнутой) в X, необходимо и достаточно, чтобы ее производная $f'(x)$ убывала (возрастала) на этом интервале.*

Теорема 2. *Пусть функция $f(x)$ определена и непрерывна вместе со своей производной $f'(x)$ на X и имеет внутри X непрерывную вторую производную $f''(x)$. Для выпуклости (вогнутости) функции $f(x)$ в X необходимо и достаточно, чтобы внутри X*

$$f''(x) \leq 0; f''(x) \geq 0.$$



Докажем теорему 2 для случая выпуклости функции $f(x)$.

Необходимость. Возмем произвольную точку $x_0 \in X$. Разложим функцию $f(x)$ около точки x_0 в ряд Тейлора

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + r_1(x),$$

$$r_1(x) = \frac{(x - x_0)^2}{2} f''(x_0 + \theta(x - x_0)) \quad (0 < \theta < 1).$$

Уравнение касательной к кривой $f(x)$ в точке, имеющей абсциссу x_0:

$$Y(x) = f(x_0) + f'(x_0)(x - x_0).$$

Тогда превышение кривой $f(x)$ над касательной к ней в точке x_0 равно

$$f(x) - Y(x) = r_1(x).$$

Таким образом, остаток $r_1(x)$ равен величине превышения кривой $f(x)$ над касательной к ней в точке x_0. В силу непрерывности $f''(x)$, если $f''(x_0) > 0$, то и $f''(x_0 + \theta(x - x_0)) > 0$ для x, принадлежащих достаточно малой окрестности точки x_0, а потому, очевидно, и $r_1(x) > 0$ для любого отличного от x_0 значения x, принадлежащего к указанной окрестности.

Значит, график функции $f(x)$ лежит выше касательной $Y(x)$ и кривая $f(x)$ выпукла в произвольной точке $x_0 \in X$.

Достаточность. Пусть кривая $f(x)$ выпукла на промежутке X. Возмем произвольную точку $x_0 \in X$.

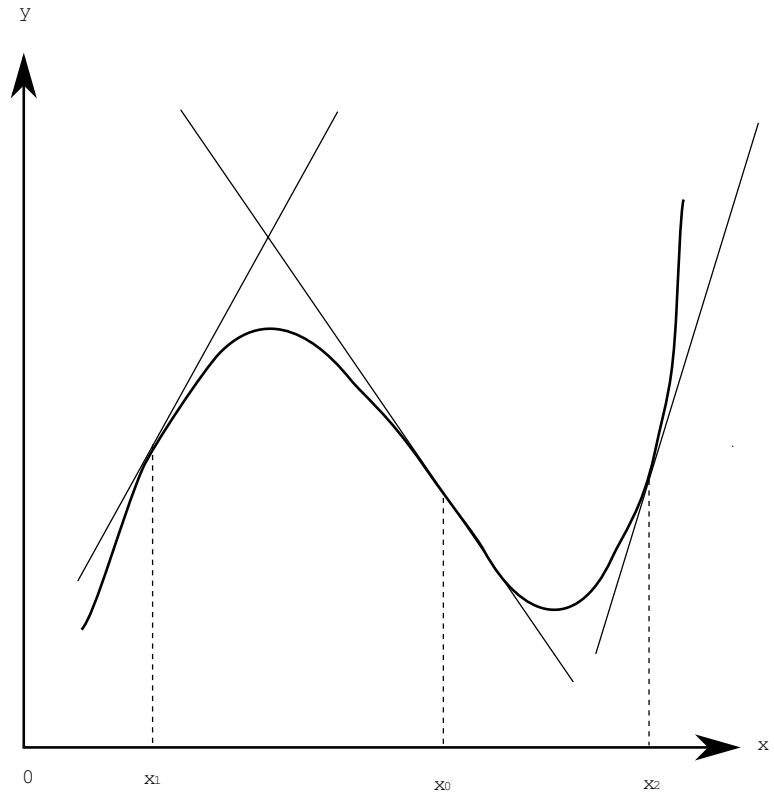
Аналогично предыдущему разложим функцию $f(x)$ около точки x_0 в ряд Тейлора

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + r_1(x),$$

$$r_1(x) = \frac{(x - x_0)^2}{2} f''(x_0 + \theta(x - x_0)) \quad (0 < \theta < 1).$$

Превышение кривой $f(x)$ над касательной к ней в точке, имеющей абсциссу x_0, определяемой выражением $Y(x) = f(x_0) + f'(x_0)(x - x_0)$, равно

$$f(x) - Y(x) = r_1(x).$$



Так как превышение положительно для достаточно малой окрестности точки x_0, то положительна и вторая производная $f''(x_0 + \theta(x - x_0))$. При стремлении $x \to x_0$ получаем, что для произвольной точки x_0 $f''(x_0) > 0$.

Пример. Исследовать на выпуклость (вогнутость) функцию $y = x^2 - 16x + 32$.

Ее производная $y' = 2x - 16$ возрастает на всей числовой оси, значит по теореме 1 функция вогнута на $(-\infty, \infty)$.

Ее вторая производная $y'' = 2 > 0$, поэтому по теореме 2 функция вогнута на $(-\infty, \infty)$.

Точки перегиба **Определение.** ***Точкой перегиба*** *графика непрерывной функции называется точка, разделяющая интервалы, в которых функция выпукла и вогнута.*

Из этого определения следует, что точки перегиба – это точки точки экстремума первой производной. Отсюда вытекают следующие утверждения для необходимого и достаточного условий перегиба.

Теорема (необходимое условие перегиба). *Для того чтобы точка x_0 являлась точкой перегиба дважды дифференцируемой функции $y = f(x)$, необходимо, чтобы ее вторая производная в этой точке равнялась нулю ($f''(x_0) = 0$) или не существовала.*

Теорема (достаточное условие перегиба). *Если вторая производная $f''(x)$ дважды дифференцируемой функции $y = f(x)$ при переходе через некоторую точку x_0 меняет знак, то x_0 есть точка перегиба.*

Отметим, что в самой точке вторая прозводная $f''(x_0)$ может не существовать.

Геометрическая интерпретация точек перегиба иллюстрируется рис. 8.

В окрестности точки x_1 функция выпукла и график ее лежит *ниже* касательной, проведенной в этой точке. В окрестности точки x_2 функция вогнута и график ее лежит *выше* касательной, проведенной в этой точке. В точке перегиба x_0 касательная разделяет график функции на области выпуклости и вогнутости.

Исследование функции на выпуклость и наличие точек перегиба 1. Найти вторую производную $f''(x)$.

2. Найти точки, в которых вторая производная $f''(x) = 0$ или не существует.

3. Исследовать знак второй производной слева и справа от найденных точек и сделать вывод об интервалах выпуклости или вогнутости и наличии точек перегиба.

Пример. Исследовать функцию $y(x) = 2x^3 - 6x^2 + 15$ на выпуклость и наличие точек перегиба.

1. $y' = 6x^2 - 12x; y'' = 12x - 12$.

2. Вторая производная равна нулю при $x_0 = 1$.

3. Вторая производная $y''(x)$ меняет знак при $x_0 = 1$, значит точка $x_0 = 1$ – точка перегиба.

На интервале $(-\infty, 1)$ $y''(x) < 0$, значит функция $y(x)$ выпукла на этом интервале.

На интервале $(1, \infty)$ $y''(x) > 0$, значит функция $y(x)$ вогнута на этом интервале.

Общая схема исследования функций и построения графика При исследовании функции и построении ее графика рекомендется использовать следующую схему:

1. Найти область определения функции.

2. Исследовать функцию на четность – нечетность. Напомним, что график четной функции симметричен относительно оси ординат, а график нечетной функции симетричен относительно начала координат.

3. Найти вертикальные асимптоты.

4. Исследовать поведение функции в бесконечности, найти горизонтальные или наклонные асимптоты.

5. Найти экстремумы и интервалы монотонности функции.

6. Найти интервалы выпуклости функции и точки перегиба.

7. Найти точки пересечения с осями координат.

Исследование функции проводится одновременно с построением ее графика.

Пример. Исследовать функцию $y(x) = f(x) = \dfrac{1 + x^2}{1 - x^2}$ и построить ее график.

1. Область определения функции – $(-\infty, -1) \bigcup (-1, 1) \bigcup (1, \infty)$.

2. Исследуемая функция – четная $y(x) = y(-x)$, поэтому ее график симметричен относительно оси ординат.

3. Знаменатель функции обращается в ноль при $x = \pm 1$, поэтому график функции имеет вертикальные асимптоты $x = -1$ и $x = 1$.

Точки $x = \pm 1$ являются точками разрыва второго рода, так как пределы слева и справа в этих точках стремятся к ∞.

$$\lim_{x \to 1-0} y(x) = \lim_{x \to -1+0} y(x) = \infty; \lim_{x \to 1+0} y(x) = \lim_{x \to -1-0} y(x) = -\infty.$$

4. Поведение функции в бесконечности.

$$\lim_{x \to \pm\infty} y(x) = -1,$$

поэтому график функции имеет горизонтальную асимптоту $y = -1$.

5. Экстремумы и интервалы монотонности. Находим первую производную

$$y'(x) = \frac{4x}{(1 - x^2)}.$$

$y'(x) < 0$ при $x \in (-\infty, -1) \bigcup (-1, 0)$, поэтому в этих интервалах функция $y(x)$ убывает.

$y'(x) > 0$ при $x \in (0, 1) \bigcup (1, \infty)$, поэтому в этих интервалах функция $y(x)$ возрастает.

$y'(x) = 0$ при $x = 0$, поэтому точка $x_0 = 0$ является критической точкой.

Находим вторую производную

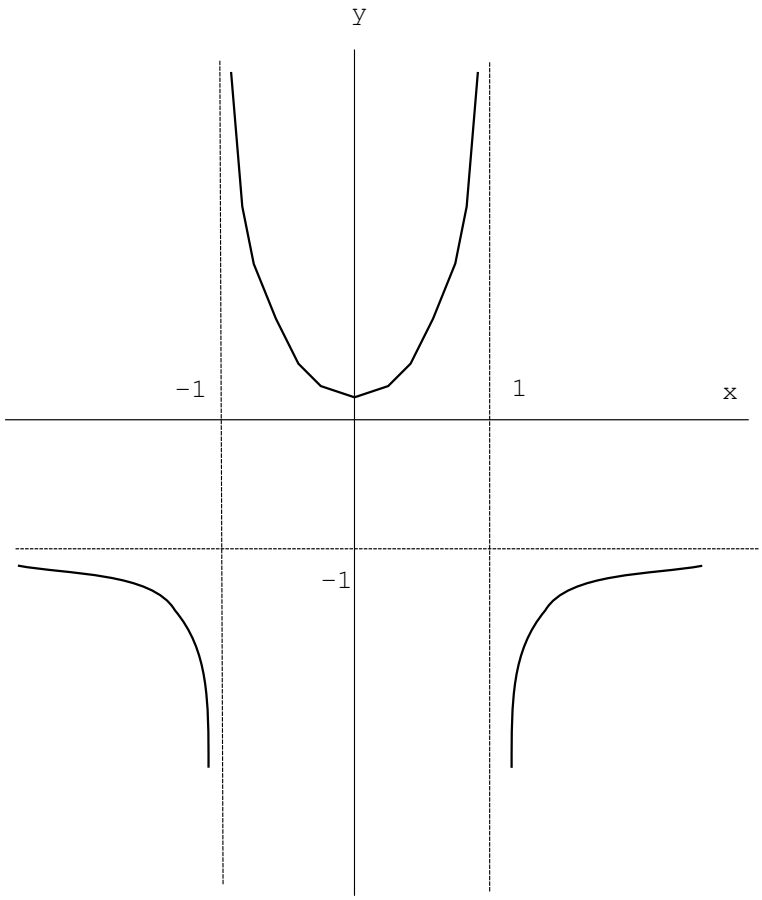
$$y''(x) = \frac{4(1 + 3x^2)}{(1 - x^2)^3}.$$

Так как $y''(0) > 0$, то точка $x_0 = 0$ является точкой минимума функции $y(x)$.

6. Интервалы выпуклости и точки перегиба.

Функция $y''(x) > 0$ при $x \in (-1, 1)$, значит на этом интервале функция $y(x)$ вогнута.

Функция $y''(x) < 0$ при $x \in (-\infty, -1) \bigcup (1, \infty)$, значит на этих интервалах функция $y(x)$ выпукла.



Функция $y''(x)$ нигде не обращается в ноль, значит точек перегиба нет.

7. Точки пересечения с осями координат.

Уравнение $f(0) = y$, имеет решение $y = 1$, значит точка пересечения графика функции $y(x)$ с осью ординат $(0, 1)$.

Уравнение $f(x) = 0$ не имеет решения, значит точек пересечения с осью абсцисс нет.

С учетом проведенного исследования можно строить график функции

$$y(x) = \frac{1 + x^2}{1 - x^2}.$$

Схематически график функции изображен на рис. 9.

Асимптоты графика функции **Определение.** ***Асимптотой*** *графика функции $y = f(x)$ называется прямая, обладающая тем свойством, что расстояние от точки $(x, f(x))$ до этой прямой стремится к 0 при неограниченном удалении точки графика от начала координат.*

Асимптоты бывают 3 видов: вертикальные (см. рис. 2а), горизонтальные (см. рис. 2б) и наклонные (см. рис. 2в).

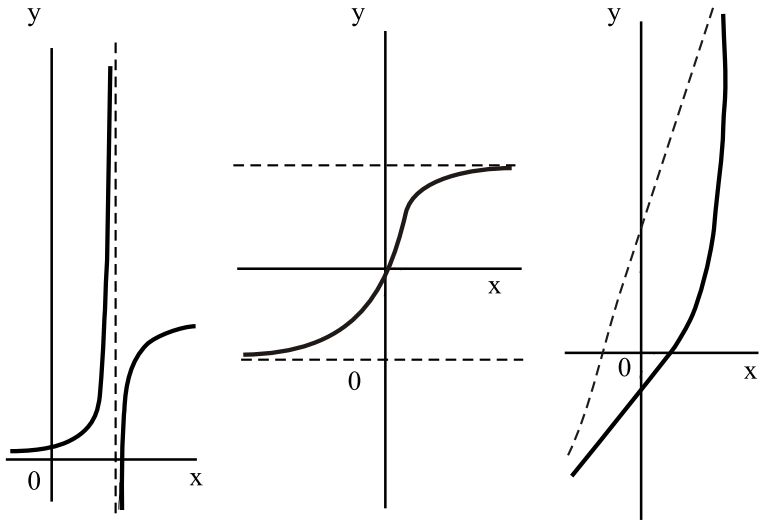
Асимптоты находят, используя следующие теоремы:

Теорема 1. *Пусть функция $y = f(x)$ определена в некоторой окрестности точки x_0 (исключая, возможно, саму эту точку) и хотя бы один из пределов функции при $x \to x_0 - 0$ (слева) или $x \to x_0 + 0$ (справа) равен бесконечности. Тогда прямая является $x = x_0$ вертикальной асимптотой графика функции $y = f(x)$.*

Вертикальные асимптоты $x = x_0$ следует искать в точках разрыва функции $y = f(x)$.

Теорема 2. *Пусть функция $y = f(x)$ определена при достаточно больших x и существует конечный предел функции*

$$\lim_{x \to \mp\infty} f(x) = b.$$



а) б) в)

Тогда прямая $y = b$ есть горизонтальная асимптота графика функции $y = f(x)$.

Теорема 3. *Пусть функция $y = f(x)$ определена при достаточно больших x и существуют конечные пределы*

$$\lim_{x\to\mp\infty}\frac{f(x)}{x} = k$$

и

$$\lim_{x\to\mp\infty}[f(x) - kx] = b.$$

Тогда прямая $y = kx + b$ является наклонной асимптотой графика функции $y = f(x)$.

Пример. Найти асимптоты графика дробно-рациональной функции

$$y(x) = \frac{ax+b}{cx+d}; c \neq 0; ad - bc \neq 0.$$

Если $c = 0$, то дробно-рациональная функция становится линейной

$$y(x) = \frac{a}{d}x + \frac{b}{d}.$$

Особая точка $x = -d/c$. Найдём предел $\lim\limits_{x\to -d/c} f(x)$.

Перепишем дробно-рациональную функцию в виде:

$$y(x) = \frac{ax+b}{c(x+d/c)}$$

Так как $ad - bc \neq 0$ то при $x \to d/c$ числитель дробно-рациональной функции не стремится к нулю. Поэтому прямая $x = -d/c$ - асимптота графика дробно-рациональной функции.

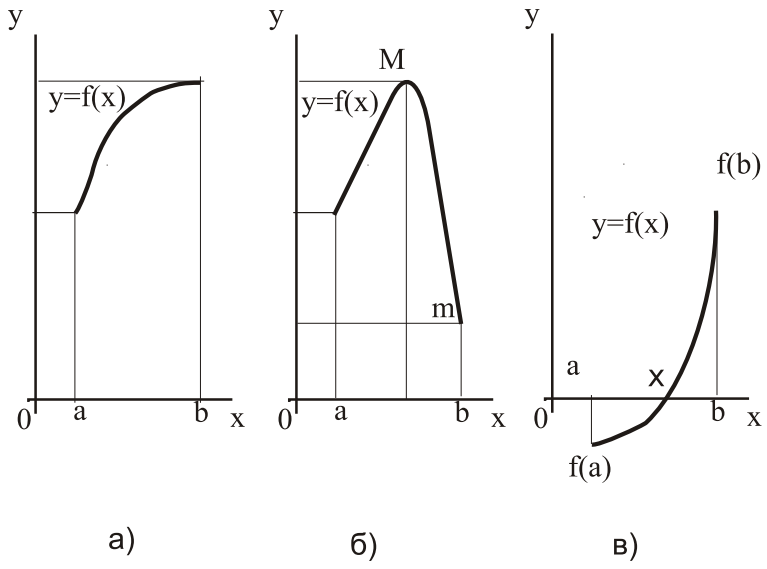
Найдём предел $\lim\limits_{x\to\pm\infty} f(x)$.

$$\lim_{x\to\pm\infty}\frac{ax+b}{cx+d} = \lim_{x\to\pm\infty}\frac{a+b/x}{c+d/c} = \frac{a}{c}$$

$y = a/c$ - является горизонтальной асимптотой дробно-рациональной функции.

Пример. Найти асимптоты кривой $y(x) = \dfrac{x^3}{x^2+1}$.

$$\lim_{x\to\pm\infty}\frac{f(x)}{x} = \lim_{x\to\pm\infty}\frac{x^2}{x^2+1} = 1.$$



Поэтому $k = 1$.

Теперь ищем b.

$$b = \lim_{x\to\pm\infty}\left[\frac{x^3}{x^2+1} - x\right] = \lim_{x\to\pm\infty}\left(\frac{-x}{x^2+1}\right)$$

Функция $y(x) = \dfrac{x^3}{x^2+1}$ имеет наклонную асимптоту $y = x$.

Свойства функций, непрерывных на отрезке. Теоремы Вейерштрасса **1.** *Если функция $y = f(x)$ непрерывна на отрезке $[a, b]$, то она ограничена на этом отрезке, т.е. существуют такие постоянные и конечные числа m и M, что*

$$m \le f(x) \le M \quad \text{при} \quad a \le x \le b$$

(см. рис. 3а).

2. *Если функция $y = f(x)$ непрерывна на отрезке $[a, b]$, то она достигает на этом отрезке наибольшего значения M и наименьшего значения m* (см. рис. 3б).

3. *Если функция $y = f(x)$ непрерывна на отрезке $[a, b]$, и значения её на концах отрезка $f(a)$ и $f(b)$ имеют противоположные знаки, то внутри отрезка найдётся точка $\xi \in (a, b)$, такая, что $f(\xi) = 0$* (см. рис. 3в).

4.3.1 Дифференцирование функций нескольких переменных

Для определения набора частных производных функции нескольких переменным (компонентов градиента) используется функция **gradef** в формате $gradef(f(x_1, ..., x_n), g_1, ..., g_m)$ или $gradef(a, x, expr)$ Выражение $gradef(f(x_1, ..., x_n), g_1, ..., g_m)$ определяет $g_1, g_2, \dots g_n$ как частные производные функции $f(x_1, c_2, \dots, x_n)$ по переменным $x_1, x_2, \dots, x_n$ соответственно.

Зависимости между переменными можно явно указать при поиощи функция depends, которая позволяет декларировать, что переменная зависит от одной или нескольких других переменных. Например, если зависимость f и x отсутствует, выражение diff(f,x) возвращает 0. Если декларировать её при помощи depends(f,x), выражение diff(f,x) возвращает символьную производную.

Пример:

```
(%i1) depends(y,x);
```

(%o1) $[y(x)]$

```
(%i2) gradef(f(x,y),x^2,g(x,y));
```

(%o2) $f(x,y)$

```
(%i3) diff(f(x,y),x);
```

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

```
(%i4) diff(f(x,y),y);
```

(%o4) $g(x,y)$

Вторая форма обращения к *gradef* фактически устанавливает зависимость a от x. При помощи *gradef* можно определить производные некоторой функции, даже если она сама неизвестна, посредством *diff* определить производные высших порядков. Для прямых вычислений, связанных с операциями векторного анализа, необходимо загрузить пакет *vect*. Кроме того, применения операторов *div*,*curl*,*grad*,, *laplasian* к некоторому выражению используется функция *express*. Пример: Вычисление градиента функции трех переменных

```
(%i2) grad (x^2 + 2*y^2 + 3*z^2);
```

(%o2) $grad\left(3\,z^2+2\,y^2+x^2\right)$

```
(%i3) express(%);
```

(%o3) $\left[\frac{d}{d\,x}\,\left(3\,z^2+2\,y^2+x^2\right),\frac{d}{d\,y}\,\left(3\,z^2+2\,y^2+x^2\right),\frac{d}{d\,z}\,\left(3\,z^2+2\,y^2+x^2\right)\right]$

```
(%i4) ev(%,diff);
```

(%o4) $[2\,x,4\,y,6\,z]$

Вычисление дивергенции

```
(%i5) div([x^2,2*y^2,3*z^2]);
```

(%o5) $div\left(\left[x^2,2\,y^2,3\,z^2\right]\right)$

```
(%i6) express(%);
```

(%o6) $\frac{d}{d\,z}\,\left(3\,z^2\right)+\frac{d}{d\,y}\,\left(2\,y^2\right)+\frac{d}{d\,x}\,x^2$

```
(%i7) ev(%,diff);
```

(%o7) $$6\,z+4\,y+2\,x$$

Вычисление вихря:

```
(%i8) curl([x^2,2*y^2,3*z^2]);
```

(%o8) $$curl\left(\left[x^2,2\,y^2,3\,z^2\right]\right)$$

```
(%i9) express(%);
```

(%o9) $$\left[\frac{d}{d\,y}\,\left(3\,z^2\right)-\frac{d}{d\,z}\,\left(2\,y^2\right),\frac{d}{d\,z}\,x^2-\frac{d}{d\,x}\,\left(3\,z^2\right),\frac{d}{d\,x}\,\left(2\,y^2\right)-\frac{d}{d\,y}\,x^2\right]$$

```
(%i10) ev(%,diff);
```

(%o10) $$[0,0,0]$$

Вычисление оператора Лапласа:

```
(%i13) laplacian(x^2+2*y^2+3*z^2);
```

(%o13) $$laplacian\left(3\,z^2+2\,y^2+x^2\right)$$

```
(%i14) express(%);
```

(%o14) $$\frac{d^2}{d\,z^2}\,\left(3\,z^2+2\,y^2+x^2\right)+\frac{d^2}{d\,y^2}\,\left(3\,z^2+2\,y^2+x^2\right)+\frac{d^2}{d\,x^2}\,\left(3\,z^2+2\,y^2+x^2\right)$$

```
(%i15) ev(%,diff);
```

(%o15) $$12$$

Рассмотрим пример исследования функции нескольких переменных: исследовать на экстремум функцию $f(x,y)=y^2-4\,y+x^3-\frac{9\,x^2}{2}+6\,x-12$

Загружаем пакет **vect**

```
(%i1) load("vect");
```

(%o1) $$/usr/share/maxima/5.13.0/share/vector/vect.mac$$

Определяем исследмое выражение и вычисляем его градиент:

```
(%i2) f:x^3-9/2*x^2+6*x+y^2-4*y-12;
```

(%o2) $$y^2 - 4\,y + x^3 - \frac{9\,x^2}{2} + 6\,x - 12$$

```
(%i3) grad(f);
```

(%o3) $$grad\left(y^2 - 4\,y + x^3 - \frac{9\,x^2}{2} + 6\,x - 12\right)$$

```
(%i4) express(%);
```

(%o4)
$$[\frac{d}{d\,x}\left(y^2 - 4\,y + x^3 - \frac{9\,x^2}{2} + 6\,x - 12\right), \frac{d}{d\,y}\left(y^2 - 4\,y + x^3 - \frac{9\,x^2}{2} + 6\,x - 12\right), \frac{d}{d\,z}\left(y^2 - 4\,y + x^3 - \frac{9\,x^2}{2} + 6\,x - 12\right)]$$

```
(%i5) ev(%,diff);
```

(%o5) $$[3\,x^2 - 9\,x + 6, 2\,y - 4, 0]$$

Выделяем из полученного списка частные производные и решаем систему $f_x(x,y) = 0; f_y(x,y) = 0$

```
(%i6) dfdx:%o5[1];
```

(%o6) $$3\,x^2 - 9\,x + 6$$

```
(%i7) dfdy:%o5[2];
```

(%o7) $$2\,y - 4$$

```
(%i8) solve([dfdx=0,dfdy=0],[x,y]);
```

(%o8) $$[[x = 1, y = 2], [x = 2, y = 2]]$$

В результате решения находим две критические точки $M_1(1,2)$ и $M_2(2,2)$ Для проверки, достигается ли в критических точках экстремум, используем достаточное условие экстремума:

```
(%i9) A:diff(dfdx,x);
```

(%o9) $$6\,x - 9$$

```
(%i10) C:diff(dfdy,y);
```

(%o10) $$2$$

```
(%i11) B:diff(dfdx,y);
```

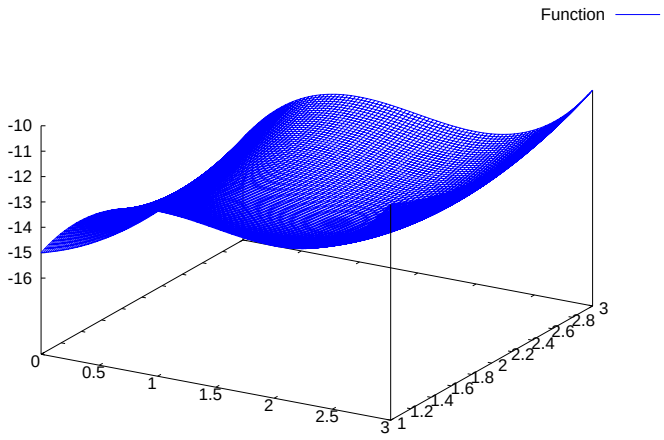


Рис. 4.5. Пример поиска экстремума функции нескольких переменных

(%o11) 0

```
(%i12) A*C-B^2;
```

(%o12) $2\,(6\,x-9)$

Так как $A*C-B^2>0$ только в точке $M_2(2,2)$, то исследуемая функция имеет единственный экстремум. Учитывая, что в точке $M_2(2,2)$ $A>0$, точка M_2 - точка минимума. Результат иллюстрируем графически (рис. 4.5).

4.4 Аналитическое и численное интегрирование

4.4.1 Основные команды

Неопределенный интеграл $\int f(x)dx$ вычисляется с помощью команды **integrate(f, x),** где **f** – подынтегральная функция, **x** – переменная интегрирования. Для вычисления определенного интеграла $\int_a^b f(x)dx$ в команде **integrate** добавляются пределы интегрирования, например,

integrate((1+cos(x))^2, x, 0, %pi);

$$\int_0^{\pi}(1+\cos(x))^2dx=\frac{3}{2}\pi$$

Несобственные интегралы с бесконечными пределами интегрирования вычисляются, если в параметрах команды **integrate** указывать, например, **x, 0, inf**.

Численное интегрирование выполняется функцией romberg (см. ниже) или при поиощи функций пакета quadpack.

4.4.2 Интегралы, зависящие от параметра. Ограничения для параметров

Если требуется вычислить интеграл, зависящий от параметра, то его значение может зависеть от знака этого параметра или каких-либо других ограничений. Рассмотрим в качестве примера интеграл $\int_0^{+\infty}e^{-ax}dx$, который, как известно из математического анализа, сходится при $a>0$ и расходится при $a<0$. Если вычислить его сразу, то получится:

```
(%i1) integrate(exp(-a*x),x,0,inf);
```

$$Is a positive, negative, or zero?$$

```
p;
```

(%o1) $$\frac{1}{a}$$

Результат аналитического интегрирования

$$\int_0^{+\infty} e^{(-ax)}dx = \lim_{x\to\infty} -\frac{e^{(-ax)}-1}{a}.$$

Для получения явного аналитического результата вычислений следует сделать какие-либо предположения о значении параметров, то есть наложить на них ограничения. Это можно сделать при помощи команды **assume(expr1)**, где **expr1** – неравенство. Описание наложенных ограничений параметра **a** можно вызвать командой **properties(a)**.

```
(%i1) assume (a > 1)$ integrate (x**a/(x+1)**(5/2), x, 0, inf);
```

$$Is \frac{2\,a+2}{5} an integer?$$

```
(%i2) no;
```

$$Is 2\,a-3 positive, negative, or zero?$$

```
(%i2) neg;
```

(%o2) $$\beta\left(a+1, \frac{3}{2}-a\right)$$

```
(%i3) properties(a);
```

(%o3) $$[database info, a > 1]$$

Вернемся к вычислению интеграла с параметром $\int_0^{+\infty} e^{-ax}dx$, которое следует производить в таком порядке:

```
(%i1) assume(a>0); integrate(exp(-a*x),x,0,inf);
```

(%o2) $$[a > 0]\frac{1}{a}$$

Отменить принятые ограничения на значения параметров можно, используя функцию forget. Пример:

```
(%i1) assume(n+1>0); integrate((a+b)*x^(n+1),x);
```

(%o2) $$[n > -1]\frac{(b+a)\ x^{n+2}}{n+2}$$

Отмена ограничения влечёт за собой вопрос о значениях параметров подинтегральной функции:

```
(%i3) forget(n+1>0); integrate((a+b)*x^(n+1),x);
```

(%o3) $$[n > -1]Isn + 2zeroornonzero?$$

```
(%i4) zero;
```

(%o4) $$(b+a)\ log\,(x)$$

Результат, который получен, совершенно другой!

4.4.3 Основные приёмы интегрирования

В *Maxima* имеется функция, предназначенных для выполнения расчетов шаг за шагом, осуществляющая замену переменной **changevar**.
Формулу интегрирования по частям:

$$\int u(x)v'(x)dx = u(x)v(x) - \int u'(x)v(x)dx$$

придётся применять вручную. В Maxima (в отличие от, например, Maple), функция интегрирования по частям не выделена явно, хотя в отдельных случаях этот способ используется integrate.

Для вычисления первообразных дифференциальных выражений используется пакет "antid"(основные функции пакета - antidiff и antid). Функция antidiff выполняет интегрирование выражений с произвольными функциями (в том числе неопределёнными), перед ее первым вызовом следует загрузить пакет (antid отличается от неё форматом выводимого результата). Пример:

```
(%i1) load("antid")$(%i2) expr: exp(z(x))*diff(z(x),x)*sin(x);
```

(%o2) $$e^{z(x)}\ sin\,(x)\ \left(\frac{d}{d\,x}\,z\,(x)\right)$$

```
(%i3) a1: antid (expr, x, z(x));
```

(%o3) $$[e^{z(x)}\ sin\,(x)\,, -e^{z(x)}\ cos\,(x)]$$

При помощи пакета "antid"можно интегрировать и полные дифференциалы, например:

Если в интеграле требуется сделать замену переменных, используется функция changevar. Синтаксис вызова этой функции: changevar (expr, f(x,y), y, x).

Функция осуществляет замену переменной в соответствии с уравнением f(x,y)=0 во всех интегралах, встречающихся в выражении expr (предполагается, что y - новая переменная, x - исходная). При использовании совместно с changevar часто используется отложенное вычи сление интеграла (одинарная кавычка перед функцией integrate).

Пример:

```
(%i5) assume(a > 0)$  'integrate (%e**sqrt(a*y), y, 0, 4);
```

(%o6) $$\int_0^4 e^{\sqrt{a}\,\sqrt{y}}\,dy$$

Данный интеграл не вычисляется аналитически непосредственно, поэтому выполняем замену:

```
(%i7) changevar (%, y-z^2/a, z, y);
```

(%o7) $$-\frac{2\int_{-2\sqrt{a}}^{0} z\,e^{|z|}dz}{a}$$

Исходный интеграл был записан с признаком отложенного вычисления, пожтому приводим результат в "завершённую"форму (выполняем ev с ключом nouns).

```
(%i8) ev(%,nouns);
```

(%o8) $$-\frac{2\left(-2\sqrt{a}\,e^{2\sqrt{a}}+e^{2\sqrt{a}}-1\right)}{a}$$

Не всегда можно вычислять интеграл (как определённый, так и неопределённый) до конца лишь за счёт использования функции integrate. В этом случае функция возвращает выражение с отложенным вычислением вложенного (возможно, более простого по форме) интеграла. Пример:

```
(%i10) expand ((x-4) * (x^3+2*x+1));
```

(%o10) $$x^4-4\,x^3+2\,x^2-7\,x-4$$

```
(%i11) integrate (1/%, x);
```

Не зная корней знаменателя, невозможно полностью вычислять интеграл от рационального выражения, поэтому один из компонентов результата - неопределённый интеграл, для окончательного вычисления которого необходимо найти корни знаменателя (например, используя allroots).

(%o11) $$\frac{log\,(x-4)}{73}-\frac{\int\frac{x^2+4\,x+18}{x^3+2\,x+1}dx}{73}$$

Возможным решением является упрощение интеграла, сопровождающееся понижением степени рационального выражения в знаменателе. При этом необходимо установить в true переменную integrate_use_rootsof. Однако при этом результат может быть довольно трудно интерпретируемым.

Рассмотрим предыдущий пример, выполнив предварительно факторизацию знаменателя:

```
(%i1) f:expand ((x-4) * (x^3+2*x+1));
```

(%o1) $$x^4-4\,x^3+2\,x^2-7\,x-4$$

```
(%i2) polyfactor:true$ ffact:allroots(f);
```

(%o3)
$$1.0\,(x-3.999999999999997)\,(x+0.4533976515164)\left(x^2-0.45339765151641\,x+2.205569430400593\right)$$

```
(%i4) float(integrate(1/ffact,x));
```

Полученный результат всё равно трудно назвать однозначно приемлемым, т.к. он включает одновременно очень большие и очень малые величины. Причина в том, что корни знаменателя представлялись рациональныит числами. Для того. чтобы получить компактный результат, желательно для коэффициентов вида $r = \frac{m}{n}$ уменьшить m и n.

Интегралы от тригонометрических и логарифмических функций Maxima вычисляет довольно успешно. Рассмотрим несколько примеров.

```
(%i1) integrate(sin(x)*sin(2*x)*sin(3*x),x);
```

$$(\%o1) \qquad \frac{cos\,(6\,x)}{24} - \frac{cos\,(4\,x)}{16} - \frac{cos\,(2\,x)}{8}$$

```
(%i2) integrate(1/cos(x)^3,x);
```

$$(\%o2) \qquad \frac{log\,(sin\,(x)+1)}{4} - \frac{log\,(sin\,(x)-1)}{4} - \frac{sin\,(x)}{2\,sin\,(x)^2-2}$$

```
(%i3) integrate(x^3*log(x),x);
```

$$(\%o3) \qquad \frac{x^4\,log\,(x)}{4} - \frac{x^4}{16}$$

4.4.4 Преобразование Лапласа

Прямое и обратное преобразование Лапласа вычисляются посредством функций laplace и ilt соответственно.

Синтаксис обращения к функции laplace: laplace (expr, t, s).

Функция вычисляет преобразование Лапласа выражения expr по отношению к переменной t. Образ выражения expr будет включать переменную s.

Функция laplace распознаёт в выражении expr функции delta, exp, log, sin, cos, sinh, cosh, и erf, а также производные, интегралы, суммы и обратное преобразование Лапласа (ilt). При наличии других функций вычисление преобразования может и не удаться.сления преобразования

Кроме того, вычисление преобразования Лапласа возможно и для дифференциальных уравнений и интегралов типа свёртки.

```
(%i1) laplace(c,t,s);
```

$$(\%o1) \qquad \frac{c}{s}$$

```
(%i2) laplace(erf(t),t,s);
```

$$(\%o2) \qquad \frac{e^{\frac{s^2}{4}}\,\left(1-erf\left(\frac{s}{2}\right)\right)}{s}$$

```
(%i3) laplace(sin(t)*exp(-a*t),t,s);
```

(%o3) $$\frac{1}{s^2+2\,a\,s+a^2+1}$$

Функция ilt (expr, t, s) вычисляет обратное преобразование Лапласа относительно переменной t с параметром s. Пример:

```
(%i1) laplace(c,t,s);
```

(%o1) $$\frac{c}{s}$$

```
(%i2) ilt(%,s,t);
```

(%o2) $$c$$

```
(%i3) laplace(sin(2*t)*exp(-4*t),t,s);
```

(%o3) $$\frac{2}{s^2+8\,s+20}$$

```
(%i4) ilt(%,s,t);
```

(%o4) $$e^{-4\,t}\,sin\,(2\,t)$$

4.5 Методы теории приближения в численном анализе

Курс высшей математики для студентов технических вузов содержит первичные основы численных методов как свою составную часть. Для специалистов инженерного профиля крайне важным представляется одновременное нахождение решения в замкнутой аналитической форме и получение численных значений результанта. Представление функции в виде степенного ряда позволяет свести изучение свойств приближаемой функции к более простой задаче изучения этих свойств у соответствующего аппроксимирующего полиномиального разложения. Этим обьясняется важность всевозможных аналитических и численных приложений полиномиальных приближений для аппроксимации и вычисления функции. Замена функций на их степенные разложения и полиномиальные приближения помогает изучению пределов, анализу сходимости и расходимости рядов и интегралов, приближенному вычислению интегралов и решению дифференциальных уравнений. Степенные ряды и разложения по многочленам Чебышева широко используются при вычислении значений функции с заданной степенью точности. Они являются эффективным вычислительным средством при решении широкого круга научно-технических задач.

4.5.1 Приближенное вычисление математических функций

Пусть функция $f(x)$ задана на интервале $(x_0 - R, x_0 + R)$ и нам требуется вычислить значение функции $f(x)$ при $x = x_1 \in (x_0 - R, x_0 + R)$ с заданной точностью $\epsilon > 0$.

Предположив, что функция $f(x)$ в интервале $x \in (x_0 - R, x_0 + R)$ раскладывается в степенной ряд

$$f(x) = \sum_{i=0}^{\infty} u_i(x) = \sum_{i=0}^{\infty} a_i(x - x_0)^i = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + ... + a_n(x - x_0)^n + ...,$$

мы получим, что точное значение $f(x_1)$ равно сумме этого ряда при $x = x_1$

$$f(x_1) = \sum_{i=0}^{\infty} a_i(x_1 - x_0)^i = a_0 + a_1(x_1 - x_0) + a_2(x_1 - x_0)^2 + ... + a_n(x_1 - x_0)^n + ...,$$

а приближенное - частичной сумме $S_n(x_1)$

$$f(x_1) \approx S_n(x_1) = \sum_{i=0}^{n} a_i(x_1 - x_0)^i = a_0 + a_1(x_1 - x_0) + a_2(x_1 - x_0)^2 + ... + a_n(x_1 - x_0)^n.$$

Для погрешности приближения мы имеем выражение в виде остатка ряда

$$f(x_1) - S_n(x_1) = r_n(x_1),$$

где

$$r_n(x_1) = \sum_{i=1}^{\infty} x_1^{n+i} = a_{n+1}x_1^{n+1} + a_{n+2}x_1^{n+2} + ...$$

Для знакопеременных рядов с последовательно убывающими членами

$$|r_n(x)| = |\sum_{i=1}^{\infty} u_{n+i}(x_1)| < |u_{n+1}(x_1)|.$$

Точность аппроксимации, как правило, возрастает с ростом степени приближающего степенного разложения и тем выше, чем точка x ближе к точке x_0. Для равномерной аппроксимации на интервале наиболее удобными оказываются разложения по многочленам Чебышева.

Для приближенного нахождения значений функции посредством степенных рядов, как правило, используются ее разложения в виде рядов Тейлора.

Ряд Тейлора для функции $f(x)$ - это степенной ряд вида

$$\sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k,$$

где числовая функция f предполагается определенной в некоторой окрестности точки x_0 и имеющей в этой точке производные всех порядков.

Многочленами Тейлора для функции $f(x)$, порядка n соответственно, называются частные суммы ряда Тейлора

$$\sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

Если мы распишем эту формулу, то получим следующее выражение

$$f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n.$$

Формула Тейлора для функции $f(x)$ - это представление функции в виде суммы ее многочлена Тейлора степени $n(n = 0, 1, 2, ...)$ и остаточного члена. Другими словами это называют разложением функции $f(x)$ по формуле Тейлора в окрестности точки x_0. Если действительная функция f одного переменного имеет n производных в точке x_0, то ее формула Тейлора имеет вид

$$f(x) = P_n(x) + r_n(x),$$

где

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

- многочлен Тейлора степени n, а остаточный член может быть записан в форме Пеано

$$r_n(x) = o((x - x_0)^n), x \to x_0.$$

Получаем, что

$$P_n(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n.$$

Если функция f дифференцируема $n+1$ раз в некоторой окрестности точки $x_0, (x_0-\delta, x_0+\delta), \delta > 0$, то остаточный член в этой окрестности может быть записан в форме Лагранжа

$$r_n(x) = \frac{f^{(n+1)}(x_0 + \theta(x - x_0))}{(n + 1)!}(x - x_0)^{(n+1)},$$

$$0 < \theta < 1, x \in (x_0 - \delta, x_0 + \delta).$$

Заметим, что при $n = 1$ выражение для $P_1(x) = f(x_0) + f'(x_0)(x - x_0)$ совпадает с формулой Лагранжа конечных приращений для функции $f(x)$.

Формула Тейлора для многочленов. Пусть имеется произвольный многочлен $f(x) = a_0x^n + a_1x^{n-1} + ... + a_n$. Тогда при любых x и h имеет место следующая формула:

$$f(x + h) = a_0(x + h)^n + a_1(x + h)^{n-1} + ... + a_n =$$
$$= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + ... + \frac{f^{(k)}(x)}{k!}h^k + ... + \frac{f^{(n)}(x)}{n!}h^n.$$

Рядом Маклорена для функции $f(x)$ называется ее ряд Тейлора в точке 0 начала координат, то есть таким образом это степенной ряд вида

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!}x^k.$$

Таким образом формула Маклорена является частным случаем формулы Тейлора. Предположим, что функция $f(x)$ имеет n производных в точке $x = 0$. Тогда в некоторой окрестности этой точки $(-\delta, \delta), \delta > 0$, функцию $f(x)$ можно представить в виде

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!}x^k + r_n(x),$$

$$x \in (-\delta, \delta),$$

где $r_n(x)$ - остаточный член $n-$ ого порядка в форме Пеано.

Приведем разложения по формуле Маклорена для основных элементарных математических функций:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} + o(x^n),$$

$$sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} + ... + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + o(x^{2n}),$$

$$cosx = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + ... + (-1)^{n}\frac{x^{2n}}{(2n)!} + o(x^{2n+1}),$$

$$(1+x)^\alpha = 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!}x^2 + ... + \frac{\alpha(\alpha-1)...(\alpha-n+1)}{n!}x^n + o(x^n),$$

$$ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + ... + (-1)^{n-1}\frac{x^n}{n} + o(x^n).$$

В Maxima существует специальная команда, позволяющая вычислять ряды и многочлены Тейлора: $taylor(expr, x, a, n)$. Здесь $expr$ -разлагаемое в ряд выражение, a - значение x, в окрестности которого выполняется разложение (по степеням $x - a$), n - параметр, указывающий на порядок разложения и представленный целым положительным числом. Если a указывается просто в виде имени переменной, то производится вычисление ряда и многочлена Маклорена.

Пример 1. Найти многочлен Тейлора 9-ой степени экспоненциальной функции e^x в начале координат.

```
(%i29) taylor(exp(x),x,0,9);
```

(%o29) $$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + \frac{x^8}{40320} + \frac{x^9}{362880} + ...$$

Многочлены Тейлора дают наиболее точную аппроксимацию приближаемой функции вблизи точки x_0. По мере удаления от точки x_0 погрешность возрастает. Для приближения приходится использовать многочлены Тейлора более высокой степени, но иногда и они не помогают в связи с накоплением вычислительной погрешности.

Интересно проследить этот процесс графически. Пакет Maxima предоставляет такую возможность с помощью команды plot.

Пример 2. Найти число e с точностью до 0.001. Положим $x = 1$. Тогда чтобы вычислить значение e, необходимо выполнить серию команд:

Строим разложение функции e^x в ряд Тейлора (до 8 порядка включительно)

```
(%i1) t:taylor(exp(x),x,0,8);
```

(%o1) $$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + \frac{x^8}{40320} + ...$$

Вычисляем частичную сумму ряда при $x = 1$:

```
(%i2) ev(t,x=1);
```

$$(\%o2) \qquad \frac{109601}{40320}$$

Значение e в форме с плавающей точкой находим, используя функцию float:

```
(%i3) float(%);
```

$$(\%o3) \qquad 2.71827876984127$$

Интересно провести вычисления и сравнить результаты, получающиеся для числа e при различных степенях используемого многочлена Тейлора. Получаются следующие результаты:
$k = 1, e_1 = 1, k = 2, e_2 = 2, k = 3, e_3 = 2.5, k = 4, e_4 = 2.666666667, k = 5, e_5 = 2.708333333, k = 6, e_6 = 2.716666667, e_7 = 2.718055556, k = 8, e_8 = 2.718253968, k = 9, e_9 = 2.718281526, e_{10} = 2.718281801.$

Отсюда видно, что значение e с точностью 0.001 вычисляется при использовании многочлена Тейлора степени не ниже 7-ой. Также следует, что число e с точностью 0.000001 или что то же самое 10^{-6} вычисляется помощи с многочлена Тейлора 9-ой или более высокой степени.

Оценку остатка ряда произведем по формуле остаточного члена ряда Маклорена

$$|f(x_1) - S_n(x_1)| = |r_n(x_1)| = |\frac{f^{n+1}(c)}{(n+1)!}|,$$

где c находится между 0 и x_1. Следует $r_n(1) = \frac{e^c}{(n+1)!}, 0 < c < 1$. Так как $e^c < e < 3$, то $r_n(1) < \frac{3}{(n+1)!}$. При $n = 7$ имеем $r_7 < \frac{3}{7!} < 0.001, e \approx 2.718$.

Наряду с командой *taylor* для разложения функций и выражений в ряды используется команда *powerseries*(выражение, x, a) (строится разложение для заданного выражения по переменной x в отрестности a). Результатом выполнения команды *powerseries* может быть построение ее ряда Тейлора в общей форме, напрмер:

```
(%i1) powerseries(sin(x),x,0);
```

$$(\%o1) \qquad \sum_{i2=0}^{\infty} \frac{(-1)^{i2}\, x^{2\,i2+1}}{(2\,i2+1)!}$$

```
(%i2) powerseries(sin(x^2),x,0);
```

$$(\%o2) \qquad \sum_{i3=0}^{\infty} \frac{(-1)^{i3}\, x^{2\,(2\,i3+1)}}{(2\,i3+1)!}$$

Для разложения в ряд Тейлора функции нескольких переменных используется функция *taylor* с указанием списка переменных в форме: $taylor(expr, [x_1, x_2, ...], [a_1, a_2, ...], [n_1, n_2, ...])$

Пример 3. Найти многочлен Тейлора 6-ой степени от функции $\frac{x}{1+x}$.

```
(%i1) f(x):=x/(1+x);
```

(%o1) $$f(x) := \frac{x}{1+x}$$

```
(%i2) powerseries(f(x),x,0);
```

(%o2) $$x \sum_{i1=0}^{\infty} (-1)^{i1} x^{i1}$$

```
(%i3) taylor(f(x),x,0,6);
```

(%o3) $$x - x^2 + x^3 - x^4 + x^5 - x^6 + ...$$

Пример 4. Найти разложение функции $\arccos(x)$ в ряд Маклорена.

```
(%i6) taylor(acos(x),x,0,12);
```

(%o6) $$\frac{\pi}{2} - x - \frac{x^3}{6} - \frac{3\,x^5}{40} - \frac{5\,x^7}{112} - \frac{35\,x^9}{1152} - \frac{63\,x^{11}}{2816} + ...$$

Пример 5. Найти разложение функции $\exp(x)+1$ по формуле Тейлора 5-ой степени в окрестности точки $x = 2$.

```
(%i7) taylor(exp(x)+1,x,2,5);
```

(%o7) $$1 + e^2 + e^2\,(x-2) + \frac{e^2\,(x-2)^2}{2} + \frac{e^2\,(x-2)^3}{6} + \frac{e^2\,(x-2)^4}{24} + \frac{e^2\,(x-2)^5}{120} + ...$$

Пример 6.Найти разложение гиперболического косинуса в ряд Маклорена 8-ой степени.

$> taylor(cosh(x), x, 10);$

Получаем
$1 + \frac{1}{2}x^2 + \frac{1}{24}x^4 + \frac{1}{720}x^6 + \frac{1}{40320}x^8 + O(x^{10})$.

Заметим, что у аналитических функций их разложения в ряд Тейлора существуют всегда. Приведем пример функции, не имеющей разложения в ряд Тейлора и для которой команда *taylor* не дает результат: $f(x) = 1/x^2 + x$.

```
(%i8) taylor(1/x^{2}+x,x,0,7);
```

(%o8) $$\frac{1}{x^2} + x + ...$$

В результате выполнения команды taylor или powerseries получаем исходное выражение $x^{-2} + x$. В то же время в окрестности других точек, например точки $x = 2$, формула Тейлора вычисляется

```
(%i13) taylor(1/x^{2}+x,x,2,2);
```

(%o13) $$\frac{2\,2^2+1}{2^2}-\frac{\left(2-2\,2^2\right)\,(x-2)}{2\,2^2}+\frac{\left(2^2+2\right)\,(x-2)^2}{8\,2^2}+\ldots$$

```
(%i14) ratsimp(%);
```

(%o14) $$2^{-2-3}\,\left(\left(2^2+2\right)\,x^2+\left(2^{2+3}-4\,2^2-8\,2\right)\,x+4\,2^2+12\,2+8\right)$$

Пакет Maxima дает возможность как нахождения разложений математических функций в ряды Тейлора, так и графической интерпретации точности этих разложений. Подобная графическая визуализация помогает пониманию сходимости многочленов Тейлора к самой приближаемой функции.

Рассмотрим примеры такой графической визуализации для функции $\cos(x)$. Сравним графики самой функции $\cos(x)$ с графиками ее разложений Тейлора различных степеней.

Пример 7. Сравним функцию $\cos(x)$ с ее разложением Маклорена 4-ой степени на интервале $[-5,5]$.

Построим разложение

```
(%i15) appr:taylor(cos(x),x,0,5);
```

(%o15) $$1-\frac{x^2}{2}+\frac{x^4}{24}+\ldots$$

Построим график (экранная форма, в формате wxMaxima)

```
(%i16) wxplot2d([appr,cos(x)], [x,-5,5], [y,-1.1,1.1],
 [nticks,100]);
```

Выведем график в файл:

```
(%i17) plot2d([appr,cos(x)], [x,-5,5], [y,-1.1,1.1],
 [gnuplot_preamble, "set grid;"], [gnuplot_term, ps],
 [gnuplot_out_file, "appr.eps"])$
```

Легко заметить, что при небольших значениях x графики самой функции и приближающего ее разложения практически совпадают, однако при возрастании x начинают отличаться.

Пример 8. Сравним функцию $\cos(x)$ с ее разложением Маклорена 8-ой степени на интервале $[-5,5]$. Сопоставим результат с предыдущим примером.

Построим разложение более высокой степени:

```
(%i18) appr1:taylor(cos(x),x,0, 9);
```

(%o18) $$1-\frac{x^2}{2}+\frac{x^4}{24}-\frac{x^6}{720}+\frac{x^8}{40320}+\ldots$$

Пример показывает, что при использовании разложения Тейлора более высокой степени точность приближения возрастает и удается достичь удовлетворительного приближения на более широком интервале. Однако заметим, что степень разложения Тейлора нельзя повышать неограниченно в связи с накапливанием вычислительной погрешности.

Разложение в ряд Тейлора может использоваться и для вычисления пределов (функция tlimit, по синтаксису аналогичная limit).

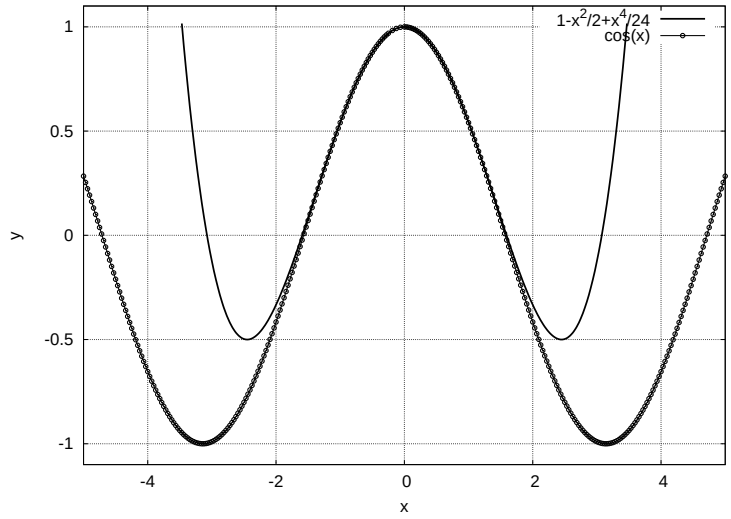


Рис. 4.6. Сопоставление разложенияв ряд Маклорена и функции y=cos(x)

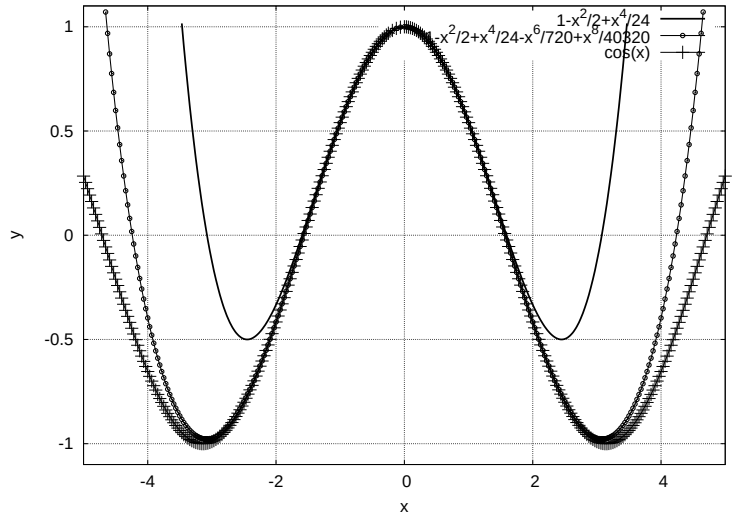


Рис. 4.7. Сопоставление двух разложений в ряд Маклорена и функции y=cos(x)

4.5.2 Приближенное вычисление определенных интегралов

Степенные ряды эффективны и удобны при приближенном вычислении определенных интегралов, не выражающихся в конечном виде через элементарные функции. Для вычисления $\int_0^x f(t)dt$ подинтегральная функция $f(t)$ раскладывается в степенной ряд. Если

$$f(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n + ...,\ |x| < R,$$

то при $|x| < R$ степенной ряд можно интегрировать почленно. Получаем метод вычисления интеграла $\int_0^x f(t)dt$ с любой наперед заданной точностью

$$\int_0^x f(t)dt = a_0x + a_1\frac{x^2}{2} + a_2\frac{x^3}{3} + ... + a_n\frac{x^{n+1}}{n+1} + ...$$

Пример 10. Приближенное вычисление интеграла вероятностей

$$\phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-x}^{x} e^{-t^2/2}dt = \frac{2}{\sqrt{2}\pi}\int_0^x e^{-t^2/2}dt.$$

Так как

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...,\ |x| < \infty,$$

то

$$e^{-x^2/2} = 1 - \frac{x^2}{2} + \frac{x^4}{2^2 2!} - \frac{x^6}{2^3 3!} + ...$$

Подставив этот ряд под знак интеграла и произведя почленное интегрирование получаем

$$\phi(x) = \frac{2}{\sqrt{2\pi}}\int_0^x e^{-t^2/2}dt = \frac{2}{\sqrt{2\pi}}\left[x - \frac{x^3}{3\cdot 2} + \frac{x^5}{5\cdot 2^2\cdot 2!} - \frac{x^7}{7\cdot 2^3\cdot 3!} + ...\right].$$

Так как это знакопеременный ряд с последовательно убывающими слагаемыми, то погрешность вычисления интеграла последовательно убывает и не превышает последнего слагаемого.

Рассмотрим пример приближенного представления интеграла в виде полинома некоторой степени в том случае, когда он не вычисляется в замкнутой аналитической форме.

Пример 14. Вычислить $\int_0^1 e^{-x^2/2}dx$, оценить достигнутую точность

Используем разложение подынтегральной функции в ряд.Подставляя в полученное выражение $x = 1$, вычисляем искомый интеграл. Так как исследуемый ряд знакопеременный, погрешность замены бесконечной суммы конечным выражением по абсолютной величине не превышает первого отброшенного члена.

```
(%i1) f(x):=exp(-x^2/2);
```

(%o1) $$f(x) := exp\left(\frac{-x^2}{2}\right)$$

```
(%i2) taylor(f(x),x,0,8);
```

(%o2) $$1 - \frac{x^2}{2} + \frac{x^4}{8} - \frac{x^6}{48} + \frac{x^8}{384} + ...$$

Интегрируя в пределах от 0 до 1, получаем числовой результат:

```
(%i3) integrate(%,x,0,1);
```

$$(\%o3) \qquad \frac{103499}{120960}$$

```
(%i4) float(%);
```

$$(\%o4) \qquad 0.85564649470899$$

Точночть расчёта оцениваем, интегрируя в пределах от 0 до a:

```
(%i5) integrate(%o2,x,0,a);
```

$$(\%o5) \qquad \frac{35\,a^9 - 360\,a^7 + 3024\,a^5 - 20160\,a^3 + 120960\,a}{120960}$$

При $a = 1$ находим:

```
(%i6) expand(%);
```

$$(\%o6) \qquad \frac{a^9}{3456} - \frac{a^7}{336} + \frac{a^5}{40} - \frac{a^3}{6} + a$$

```
(%i7) float(1/3456);
```

$$(\%o7) \qquad 2.89351851851851841\,10^{-4}$$

Таким образом, точность расчёта значения интеграла $\int_0^1 e^{-x^2/2}dx$, не хуже 0,0003. Окончательно

$$\int_0^1 e^{-x^2/2}dx = 2.8935 \pm 0.0003$$

4.6 Преобразование степенных рядов

Пакет Maxima позволяет не только строить разложение различных функций в степенные ряды, но и представления их в виде дробно-рациональной функции (аппроксимация Паде) или цепной дроби.

Аппроксимацией Паде для функции $f(x) = \sum_{k=0} \infty a_k x^k$, заданной степенным рядом, называется такая дробно-рациональная функция $R(x) = \frac{\sum_{k=0} L p_k x^k}{1+\sum_{k=1} M q_k x^k}$ чье разложение в степенной ряд совпадает со степенным рядом f(x) с точностью до коэффициента при x L+M.

Паде-аппроксиммант задается значением функции в заданной точке и M+L значениями её производных в этой же точке. Эта же информация может послужить основой для степенного ряда, так в чем же отличие? Главное отличие в том, что задав M+L+1 член степенного ряда, мы отбрасываем остальные члены ряда, приравнивая их к нулю. Паде-аппроксимант не является полиномом, поэтому задав M+L+1 членов разложения Паде-аппроксиманта в степенной ряд, мы в неявной форме задаем и остальные члены.

Чему эти дополнительные члены будут равны? Это вопрос, на который нет однозначного ответа. В одних случаях они позволят нам построить более точную аппроксимацию, в других - наоборот могут ухудшить положение. Нет способа, который позволил бы сказать, насколько точна окажется Паде-аппроксимация и в какой окрестности и с какой точностью можно получить результаты.

Ещё одним недостатком этого метода является то, что он требует информации не о значениях функции, а о её производных высших порядков, которые могут быть значительно большими по абсолютной величине, чем сами згачения функции.

Паде-аппрокимация наиболее эффективна для функций, имещих полюса на комплексной плоскости в окрестностях точки разложения. Например, функция $f(x) = \frac{1}{1+\sin^2(x)}$ непрерывна на действительной оси, но имеет полюса на комплексной плоскости. Поэтому она неэффективно аппроксимируется степенным рядом (до шестой степени включительно), но хорошо аппроксимируется по Паде со степенями числителя и знаменателя равными 4 и 2.

Функция pade, представленная в пакете Maxima, аппроксимирует отрезок ряда Тейлора, содержащий слагаемые до n-го порядка включительно, дробно-рациональной функцией. Ее аргументы — ряд Тейлора, порядок числителя n, порядок знаменателя m. Разумеется, количество известных коэффициентов ряда Тейлора должно совпадать с общим количеством коэффициентов в дробно-рациональной функции минус один (поскольку числитель и знаменатель определены с точностью до общего множителя). Синтаксис вызова функции pade:

```
pade (ряд Тейлора, степень числителя, степень знаменателя)
```

Вместо ряда Тейлора может использоваться ряд Лорана. В этом случае степени числителя и знаменателя могут быть и бесконечными (inf). В этом случае рассматриваются все рациональные функции, суммарная степень которых меньше или равна длине степенного ряда. Примеры:

```
(%i1) t:taylor(exp(x),x,0,3);
```

$$(\%o1) \qquad 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + ...$$

```
(%i2) pade(t,1,2);
```

$$(\%o2) \qquad [\frac{2\,x + 6}{x^2 - 4\,x + 6}]$$

```
(%i3) taylor(sin(x)/x,x,0,7);
```

$$(\%o3) \qquad 1 - \frac{x^2}{6} + \frac{x^4}{120} - \frac{x^6}{5040} + ...$$

```
(%i4) pade(%,2,4);
```

$$(\%o4) \qquad [-\frac{620\,x^2 - 5880}{11\,x^4 + 360\,x^2 + 5880}]$$

```
(%i5) taylor (1/(cos(x) - sec(x))^3, x, 0, 5);
```

$$(\%o5) \qquad -\frac{1}{x^6} + \frac{1}{2\,x^4} + \frac{11}{120\,x^2} - \frac{347}{15120} - \frac{6767\,x^2}{604800} - \frac{15377\,x^4}{7983360} + ...$$

```
(%i6) pade(%,3,inf);
```

$$(\%o6) \qquad [-\frac{120}{41\,x^{10} + 60\,x^8 + 120\,x^6}, \frac{8806092\,x^2 - 16847160}{1353067\,x^{10} - 382512\,x^8 + 16847160\,x^6}]$$

Более специфичной является функция cf, которая рассчитывает коэффициенты цепной дроби, аппроксимирующей заданное выражение (синтаксис вызова - cf(expr)). Выражение expr должно состоять из целых чисел, квадратных корней целых чисел и знаков арифметических операций. Функция возвращает список коэффициентов (непрерывная дробь a + 1/(b + 1/(c + ...)) представляется списком [a, b, c, ...]). Флаг cflength определяет количество периодов цепной дроби. Изначально установлено значение 1. Функция cfdisrep преобразует список (как правило выдачу функции "cf) в собственно цепную дробь вида a + 1/(b + 1/(c + ...)).

Примеры использования функций cf и cfdisrep:

```
(%i1) cf ([1, 2, -3] + [1, -2, 1]);
```

(%o1) $[1, 1, 1, 2]$

```
(%i2) cfdisrep (%);
```

(%o2) $1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}$

```
(%i3) cflength: 3;
```

(%o3) 3

```
(%i4) cf (sqrt (3));
```

(%o4) $[1, 1, 2, 1, 2, 1, 2]$

```
(%i5) cfdisrep(%);
```

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

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

(%o6) 1.731707317073171

4.7 Решение дифференциальных уравнений в Maxima

4.7.1 Основные определения

Дифференциальным уравнением называется уравнение вида $F(x, y, y', \ldots, y_{(n)}) = 0$, где $F(t_0, t_1, \ldots, t_{n+1})$ - функция, определенная в некоторой области D пространства R^{n+2}, x - независимая переменная, y - функция от x, $y', \ldots, y^n$ - ее производные. Порядком уравнения n называется наивысший из порядков производных y, входящих в уравнение. Функция $f(x)$ называется решением дифференциального уравнения на промежутке $(a; b)$, если для всех x из $(a; b)$ выполняется равенство: $F(x, f(x), f'(x), \ldots, f^n(x)) = 0$. Дифференциальному уравнению удовлетворяет бесконечное множество функций, но при некоторых условиях решение такого уравнения единственное. Интегральная

кривая – это график решения дифференциального уравнения, т.е график функции, удовлетворяющей этому уравнению.

Если дифференциальное уравнение имеет одну независимую переменную, то оно называется обыкновенным дифференциальным уравнением, если же независимых переменных две или более, то такое дифференциальное уравнение называется дифференциальным уравнением в частных производных.

Пример 1: решить уравнение $y' = 0$. Очевидно, что его решение $f(x) = const$ определено на $(-\infty, \infty)$. Отметим, что эта постоянная – произвольная и решение – не единственное, а имеется бесконечное множество решений.

Пример 2: Решить уравнение $y' = \frac{y}{x}$, или $\frac{dy}{dx} = \frac{y}{x}$. Преобразуя уравнение, получим: $\frac{dy}{y} = \frac{dx}{x}$. Интегрируя обе части уравнения, получим: $\int \frac{dy}{y} = \int \frac{dx}{x} \Rightarrow ln\, y = ln\, x + ln\, C$, или $y = Cx$. Общее решение изображается серией линейных интегральных кривых, проходящих через точку (0,0). При этом через любую точку, не принадлежащую $(0, 0)$, проходит только одна интегральная кривая (решение).

Общее решение – множество решений дифференциального уравнения $y' = f(x, y)$ есть совокупность функций $F(x, y, C) = 0$, $C = const$. Частное решение получают при подстановке конкретного значения константы в общее решение. Особые решения не входят в общие решения, и через каждую точку особого решения проходит более одной интегральной кривой. Особые решения нельзя получить из общего решения ни при каких значениях постоянной C. Если построить семейство интегральных кривых дифференциального уравнения, то особое решение будет изображаться линией, которая в каждой своей точке касается по крайней мере одной интегральной кривой.

Пример 3: Рассмотрим уравнение $y' = \frac{-x}{y}$. Преобразуя его, найдём: $\frac{dy}{dx} = -\frac{x}{y} \Rightarrow 2ydy + 2xdx = 0 \Leftrightarrow d(x^2 + y^2) = 0$. Интегрируя, получаем $x^2 + y^2 = C$.

Пример 4: Дифференциальное уравнение $y = 2\sqrt{y}$ имеет общее решение $y = (x - C)^2$ и особое решение $y = 0$. При конкретном значении C (например, $C = 1$) получаем частное решение: $y = (x - 1)^2$.

Геометрически множество решений диференциального уравнения представляется в виде поля направлений. В каждой точке области, в которой определено поле направлений, задаётся прямая с угловым коэффициентом, равным производной решения. Касательная ко всем подобным прямым и даёт интегральную кривую.

Возможность однозначного решения дифференциального уравнения определяется теоремой единственности:

Пусть $f(x, y)$ - непрерывная функция в области $D = \{(x, y; a < x < b; c < y < d)\}$, причем причём частная производная $\frac{\partial f}{\partial y}(x, y)$ также непрерывна в D. Тогда существует единственное решение $y = y(x)$ дифференциального уравнения $y' = f(x, y$ с начальным условием $y(x_0) = y_0$,$(x_0, y_0 \in D$. Следовательно, через точку $(x_0, y_0) \in D$ проходит только одна интегральная кривая.

4.7.2 Функции для решения дифференциальных уравнений в Maxima

В Maxima предусмотрены специальные средства решения задачи Коши для систем обыкновенных дифференциальных уравнений, заданных как в явной форме dx/dt= F(t,x) , так и в неявной Mdy/dt=F(t,x), где M- матрица, - т.н. решатель ОДУ (solver ODE), обеспечивающий пользователю возможность выбора метода, задания начальных условий и др. Функция ode2 позволяет решить обыкновенные дифференциальные уравнения первого и второго порядков. Синтаксис вызова ode2 (eqn, dvar, ivar)., где eqn - выражение, определяющее само дифференциальное уравнение, зависимая переменная - dvar, независимая переменная - ivar. Дифференциальное уравнение представляется в форме с "замороженной"произ- водной (т.е. с производной, вычисление которой запрещено с помощью одиноч- ной кавычки: “diff(y,x) "). Другой вариант явно указать зависимость $y = y(x)$ - использовать функцию *depends* (в этом случае можно не использовать начальный апостроф см. пример). Если ode2 не может получить решение, она возвращает значение $false$.

Посредством функции ode2 могут быть решены следующие типы ОДУ первого порядка: линейные, ОДУ с разделяющимися переменными, однородные ОДУ, уравнения в полных дифференциалах, уравнения Бернулли, обобщённые однородные уравнения.

Кроме того, при помощи функции ode2 могут быть решены следующие типы уравнений второго порядка: с постоянными коэффициентами; в полных дифференциалах; линейные однородные с переменными коэффициентами, которые могут быть сведены к уравнениям с постоянными коэффициентами; уравнения Эйлера; уравнения, разрешимые методом вариации постоянных; уравнения, свободные от независимой переменной, дорускающие понижение порядка.

Тип используемого метода сохраняется в переменной method. При использовании интегрирующего множителя он сохраняется в переменной intfactor. Частное решение неоднородного уравнения сохраняется в переменной yp.

Для отыскания частных решений задач Коши с начальными условиями используются функции $ic1$(для уравнений первого порядка) и $ic2$ (для уравнений второго порядка). Частные решения граничных задач для уравнений второго порядка используют функцию $bc2$.

Рассмотрим примеры использования функции ode2.

Вариант использования отложенного дифференцирования ($'diff$):

```
(%i1) ode2('diff(y,x)=2*y+exp(x),y,x);
```

$$(\%o1) \qquad y = \left(\%c - e^{-x}\right)\, e^{2\,x}$$

Вариант с явным указанием зависимости $y = y(x)$:

```
(%i1) depends(y,x);
```

$$(\%o1) \qquad [y\,(x)]$$

```
(%i2) ode2(diff(y,x)=2*y+exp(x),y,x);
```

$$(\%o2) \qquad y = \left(\%c - e^{-x}\right)\, e^{2\,x}$$

Параметр %c - постоянная интегрирования для уравнения первого порядка.

Решение уравнения второго порядка:

```
(%i4) ode2('diff(y,x,2)-3*'diff(y,x)+2*y=0,y,x);
```

$$(\%o4) \qquad y = \%k1\, e^{2\,x} + \%k2\, e^{x}$$

Параметры %k1 и %k2 - постоянные интегрирования для уравнений второго порядка.

Рассмотрим варианты вычисления частных решений: для уравнения первого порядка

```
(%i2) ic1(%,x=1,y=1);
```

$$(\%o2) \qquad y = e^{-2}\, \left((e+1)\, e^{2\,x} - e^{x+2}\right)$$

для уравнения второго порядка

```
(%i5) ic2(%,x=0,y=1,diff(y,x)=1);
```

$$(\%o5) \qquad y = e^{x}$$

4.7.3 Решение основных типов дифференциальных уравнений

4.7.3.1 Уравнения с разделяющимися переменными

Уравнениями с разделяющимися переменными называются уравнения вида $y' = f(x) \cdot g(y)$, где $f(x)$ - функция, непрерывна на некотором интервале (a, b), а функция $g(y)$ - функция, непрерывна на интервале (c, d), причем $g(y) \neq 0$ на (c, d)).

Преобразуя уравнение, получаем: $\frac{dy}{dx} = f(x) \cdot g(y) \Leftrightarrow \frac{dy}{g(y)} = f(x)dx$

Интегрируя обе части, получаем $\int \frac{dy}{g(y)} = \int f(x)dx$. Обозначая $G(y)$ любую первообразную для $\frac{1}{g(y)}$, а $F(x)$ - любую первообразную для $f(x)$, получаем общее решение дифференциального уравнения в виде неявно выраженной функции $G(y) = F(x) + C$.

Пример решения в Maxima:

Отыскиваем общее решение:

```
(%i1) difur1:'diff(y,x)=sqrt(1-y^2)/sqrt(1-x^2);
```

(%o1) $$\frac{d}{d\,x}\,y = \frac{\sqrt{1-y^2}}{\sqrt{1-x^2}}$$

```
(%i2) rez:ode2(difur1,y,x);
```

(%o2) $$asin\,(y) = asin\,(x) + \%c$$

Отыскиваем различные варианты частных решений:

```
(%i3) ic1(rez,x=0,y=0);
```

(%o3) $$asin\,(y) = asin\,(x)$$

```
(%i4) ic1(rez,x=0,y=1);
```

(%o4) $$asin\,(y) = \frac{2\,asin\,(x) + \pi}{2}$$

4.7.3.2 Однородные уравнения

Под однородными уравнениями понимаются уравнения вида $y' = f(\frac{y}{x}$. Для их решения используется замена вида $y = u \cdot x$, после подстановки которой получается уравнение с разделяющимися переменными: $y' = u'\,x + u \Rightarrow u'\,x + u = f(u)$. Разделяя переменные и интегрируя, получаем: $x\frac{du}{dx} = f(u) - u \Rightarrow \int \frac{du}{f(u)-u} = \int \frac{dx}{x}$

Пример решения в Maxima:

Находим общее решение:

```
бщее
(%i1) homode:'diff(y,x) = (y/x)^2 + 2*(y/x);
```

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

```
(%i2) ode2(homode,y,x);
```

(%o2) $$-\frac{x\,y+x^2}{y}=\%c$$

Находим частное решение:

```
(%i3) ic1(%,x=2,y=1);
```

(%o3) $$-\frac{x\,y+x^2}{y}=-6$$

Более общий вариант дифференциальных уравнений, сводим ых к однородным - уравнения вида: $y'=\frac{a_1x+b_1y+c_1}{a_2x+b_2y+c_2}$. Maxima не способна решать их при помощи ode2 непосредственно, а лишь после необходимого преобразования.

4.7.3.3 Линейные уравнения первого порядка

Дифференциальное уравнение называется линейным относительно неизвестной функции и ее производной, если оно может быть записано в виде:

$$y'=P(x)\cdot y=Q(x)$$

при этом, если правая часть Q(x) равна нулю, то такое уравнение называется линейным однородным дифференциальным уравнением, если правая часть Q(x) не равна нулю, то такое уравнение называется линейным неоднородным дифференциальным уравнением. При этом P(x) и Q(x)- функции непрерывные на некотором промежутке $x\in(a,b)$.

Рассмотрим решение линейного дифференциального уравнения в Maxima:

```
(%i1) lineq1:'diff(y,x)-y/x=x;
```

(%o1) $$\frac{d}{d\,x}\,y-\frac{y}{x}=x$$

```
(%i2) ode2(lineq1,y,x);
```

(%o2) $$y=x\,\left(x+\%c\right)$$

При работе с Maxima не требуется приводить дифференциальное уравнение к стандартной форме вида

$$y'=P(x)\cdot y=Q(x)$$

```
(%i3) lineq2:y^2-(2*x*y+3)*'diff(y,x)=0;
```

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

```
(%i4) ode2(lineq2,y,x);
```

(%o4) $$\frac{x\,y+1}{y^3}=\%c$$

4.7.3.4 Уравнения в полных дифференциалах

Дифференциальное уравнение первого порядка вида:

$$P(x,y)dx + Q(x,y)dy = 0$$

называется уравнением в полных дифференциалах, если левая часть этого уравнения представляет собой полный дифференциал некоторой функции $u = F(x,y)$. Данное дифференциальное уравнение является уравнением в полных дифференциалах, если выполняется условие:

$$\frac{\partial P}{\partial y} = \frac{\partial Q}{\partial x}$$

.

Общий интеграл уравнения имеет вид $U(x,y) = 0$.

Если уравнение $P(x,y)dx + Q(x,y)dy = 0$ не является уравнением в полных дифференциалах, но выполняются условия теоремы единственности, то существует функция $\mu = \mu(x,y)$ (интегрирующмй множитель) такая, что

$$\mu(Pdx + Qdy) = dU$$

.

Функция μ удовлетворяет условию:

$$\frac{\partial(\mu P)}{\partial y} = \frac{\partial(\mu Q)}{\partial x}$$

Примеры решения в Maxima:

Для решения в Maxima дифференциальное уоавнение представляется в форме

$$P(x,y) + Q(x,y)\frac{dy}{dx} = 0$$

Уравнение, приводимое к уравнению в полных дифференциалах

```
(%i1) deq:(2*x*y+x^2*y+y^3/3)+(x^2+y^2)*'diff(y,x)=0;
```

$$(\%o1) \qquad \left(y^2 + x^2\right)\left(\frac{d}{d\,x}\,y\right) + \frac{y^3}{3} + x^2\,y + 2\,x\,y = 0$$

```
(%i2) ode2(deq,y,x);
```

$$(\%o2) \qquad \frac{e^x\,y^3 + 3\,x^2\,e^x\,y}{3} = \%c$$

Указание на интегрирующий множитель

```
(%i3) intfactor;
```

$$(\%o3) \qquad e^x$$

Указание на использованный метод

```
(%i4) method;
```

$$(\%o4) \qquad exact$$

Уравнение в полных дифференциалах

```
(%i5) deq1:(3*x^2+6*x*y^2)+(6*x^2*y+4*y^3)*'diff(y,x)=0;
```

(%o5) $$\left(4\,y^3+6\,x^2\,y\right)\,\left(\frac{d}{d\,x}\,y\right)+6\,x\,y^2+3\,x^2=0$$

```
(%i6) ode2(deq1,y,x);
```

(%o6) $$y^4+3\,x^2\,y^2+x^3=\%c$$

Указание на использованный метод

```
(%i7) method;
```

(%o7) $$exact$$

4.7.3.5 Уравнения Бернулли

Уравнением Бернулли называется уравнение вида

$$y'=P(x)\cdot y=Q(x)\cdot y^{\alpha}$$

где P и Q – функции от x или постоянные числа, а α – постоянное число, не равное 1.

Для решения уравнения Бернулли применяют подстановку $z=\frac{1}{y^{\alpha-1}}$, с помощью которой, уравнение Бернулли приводится к линейному.

Пример решения уравнения Бернулли с помощью Maxima:

```
(%i1) deq:'diff(y,x)=4/x*y+x*sqrt(y);
```

(%o1) $$\frac{d}{d\,x}\,y=\frac{4\,y}{x}+x\,\sqrt{y}$$

```
(%i2) ode2(deq,y,x);
```

(%o2) $$y=x^4\,\left(\frac{log\left(x\right)}{2}+\%c\right)^2$$

```
(%i3) method;
```

(%o3) $$bernoulli$$

```
(%i4) de1:'diff(y,x)+y/x=-x*y^2;
```

(%o4) $$\frac{d}{d\,x}\,y+\frac{y}{x}=-x\,y^2$$

```
(%i5) ode2(de1,y,x);
```

(%o5) $$y=\frac{1}{x\,\left(x+\%c\right)}$$

4.7.3.6 Уравнения высших порядков

В Maxima при помощи функции ode2 возможно прямое решение лишь линейных дифференциальных уравнений второго порядка При решении выполняется проверка, является ли заданное уравнение линейным, т.е. возможно ли его преобразование к форме $y'' + p(x)y + q(x)y = r(x)$.

Первоначально отыскивается решение однородного уравнения вида $y'' + p(x)y + q(x)y = 0$ в форме $y = k_1y_1 + k_2y_2$ (k_1, k_2 - произвольные постоянные). Если $r(x) \neq 0$, отыскивается частное решение неоднородного уравнения методом вариации постоянных.. .

4.7.3.7 Уравнения с постоянными коэффициентами

Решения однородных уравнений вида $y'' + a*y' + b*y = 0$ отыскиваются по результатам решения характеристического уравнения $r^2 + ar + b = 0$. Возможны следующие варианты комбинаций его корней r_1, r_2:

- r_1, r_2 - вещественные и различные. Решение представляется в форме $y = k_1 \cdot e^{r_1 \cdot x} + k_2 \cdot e^{r_2 \cdot x}$.
- ешения од $r_1, = r_2$ - корни вещественные одинаковые. Решение представляется в форме $y = (k_1 + k_2 \cdot x)e^{r_1 \cdot x}$.
- r_1, r_2 - комплексные (сопряжённые). Если $r_1 = \alpha + \beta i$, $r_2 = \alpha - \beta i$, то решение представляется в виде $y = e^{\alpha x}(k_1 cos\,\beta x + k_2 sin\beta x)$

Общее решение неоднородного уравнения с постоянными коэффициентами представляется в виде суммы общего решения соответствующего одногодного уравнения и какого-либо частного решения неоднородного.

Примеры решения ОДУ второго порядка с постоянными коэффициентами

Неоднородное уравнение общего вида:

```
(%i1) de1:2*'diff(y,x,2)-'diff(y,x)-y=4*x*exp(2*x);
```

(%o1)
$$2\left(\frac{d^2}{d\,x^2}\,y\right) - \frac{d}{d\,x}\,y - y = 4\,x\,e^{2\,x}$$

```
(%i2) ode2(de1,y,x);
```

(%o2)
$$y = \frac{(20\,x - 28)\ e^{2\,x}}{25} + \%k1\,e^{x} + \%k2\,e^{-\frac{x}{2}}$$

Частное решение неоднородного уравнения сохраняется в переменной *yp*:

```
(%i3) yp;
```

(%o3)
$$\frac{(20\,x - 28)\ e^{2\,x}}{25}$$

Неоднородное уравнение с кратными корнями характеристического уравнения:

```
(%i1) de2:'diff(y,x,2)-2*'diff(y,x)+y=x*exp(x);
```

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

```
(%i2) ode2(de2,y,x);
```

(%o2) $$y = \frac{x^3\,e^x}{6} + (\%k2\,x + \%k1)\;e^x$$

```
(%i3) yp;
```

(%o3) $$\frac{x^3\,e^x}{6}$$

и

Неоднородное уравнение с комплексными корням:

```
(%i4) de3:'diff(y,x,2)+y=x*sin(x);
```

(%o4) $$\frac{d^2}{d\,x^2}\,y + y = x\,sin\,(x)$$

```
(%i5) ode2(de3,y,x);
```

(%o5) $$y = \frac{2\,x\,sin\,(x) + \left(1 - 2\,x^2\right)\;cos\,(x)}{8} + \%k1\,sin\,(x) + \%k2\,cos\,(x)$$

```
(%i6) yp;
```

(%o6) $$\frac{2\,x\,sin\,(x) + \left(1 - 2\,x^2\right)\;cos\,(x)}{8}$$

4.7.3.8 Уравнения с переменными коэффициентами

Аналогично уравнению с постоянными коэффициентами, общее решение однородного уравнения $y'' + p(x)y' + q(x)y = 0$ имеет вид $y = C_1y_1 + C_2y_2$, где y_1, y_2 - линейно независимые решения однородного ОДУ (фундаментальная система решений). Общее решение неоднородного уравнения $y'' + p(x)y' + q(x)y = f(x)$ с непрерывными коэффициентами и правой частью имеет вид $y = y_0 + Y$, где y_0 - общее решение соответствующего однородного уравнения, Y - частное решение неоднородного.

Если известна фундаментальная система решений однородного уравнения, общее решение неоднородного может быть представлено в форме:

$$y = C_1(x) \cdot y_1 + C_2(x) \cdot y_2,$$

где $C_1(x), C_2(x)$ определяются методом вариации произвольных постоянных.

Пример 1:

```
(%i3) difur:x^2*'diff(y,x,2)-x*'diff(y,x)=3*x^3;
```

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

```
(%i4) ode2(difur,y,x);
```

(%o4) $$y = x^3 + \%k2\, x^2 - \frac{\%k1}{2}$$

Пример 2:

```
(%i3) difur1:x*'diff(y,x,2)+'diff(y,x)=x^2;
```

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

```
(%i4) ode2(difur1,y,x);
```

(%o4) $$y = \%k1\, log\,(x) + \frac{x^3}{9} + \%k2$$

4.7.3.9 Уравнение Эйлера

Однородное уравнение $x^2y'' + axy' + by = 0$ называется уравнением Эйлера. Его общее решение имеет вид $y = k_1x^{r_1} + k_2x^{r_2}$, где r_1 и r_2 - решения уравнения $r(r-1) + ar + b = 0$. В случае, когда уравнение $r(r-1) + ar + b = 0$ имеет двукратный корень r, решение представляется в форме $y = k_1x^r + k_2ln(x)x^r$.

Неоднородное уравнение тпа Эйлерасводится к однородному с постоянными коэффициентами путём соответствующей замены.

Пример:

```
(%i1) du:x^2*'diff(y,x,2)+x*'diff(y,x)+y=1;
```

(%o1) $$x^2\left(\frac{d^2}{d\,x^2}\,y\right) + x\left(\frac{d}{d\,x}\,y\right) + y = 1$$

```
(%i2) ode2(du,y,x);
```

(%o2) $$y = sin\,(log\,(x))^2 + \%k1\, sin\,(log\,(x)) + cos\,(log\,(x))^2 + \%k2\, cos\,(log\,(x))$$

4.7.3.10 Граничные задачи

Для задания граничных условий при интегрировании ОДУ второго порядка используется функция bc2. Синтаксис вызова: bc2 (solution, xval1, yval1, xval2, yval2), где xval1 - значение x в первой граничной точке, yval1 - значение решения y в той же точке (обе величины задаются в форме x=a, y=b).

Пример использования ode2 и bc2:

```
(%i1) 'diff(y,x,2) + y*'diff(y,x)^3 = 0;
```

(%o1) $$\frac{d^2}{d\,x^2}\,y + y\left(\frac{d}{d\,x}\,y\right)^3 = 0$$

```
(%i2) ode2(%,y,x);
```

(%o2) $$\frac{y^3 + 6\,\%k1\,y}{6} = x + \%k2$$

```
(%i3) bc2(%,x=0,y=1,x=1,y=3);
```

(%o3) $$\frac{y^3 - 10\,y}{6} = x - \frac{3}{2}$$

4.7.4 Операторный метод решения

Для решения систем обыкновенных линейных дифференциальных уравнений в Maxima имеется функция desolve. Работа функции desolve основана на преобразовании Лапласа заданных дифференциальных уравнений.

Пусть задана функция действительного переменного $f(t)$, которая удовлетворяет следующим условиям:

1) однозначна и непрерывна вместе со своими производными n-го порядка для всех $t > 0$, кроме тех, где она и ее производные имеют разрывы 1-го рода. При этом в каждом конечном интервале изменения имеется конечное число точек разрыва;

2)$f(t) = 0$ для всех $t > 0$;

3) возрастает медленнее некоторой экспоненциальной функции $M \cdot e^{at}$, где M и a - некоторые положительные величины, т.е. всегда можно указать такие M и a, чтобы при любом $t > 0$ соблюдалось неравенство $|f(t)| < M \cdot e^{at}$.

Рассматриваемой функции $f(t)$ ставится в соответствие новая функция , определяемая равенством

$$F(s) = L\{f(t)\} = \int_0^\infty e^{-st} f(t)\,dt$$

где s - положительное действительное число или комплексное число с положительной действительной частью.

Функция $f(t)$ при этом называется оригиналом, а $F(s)$ - изображением функции $f(t)$ по Лапласу. Переход от оригинала к изображению называется преобразованием Лапласа. Соответственно, обратный переход от изображения к оригиналу называется обратным преобразованием Лапласа.

Для преобразования Лапласа выполняется теорема единственности: если две непрерывнные функции f(x) и g(x) имеют одно и то же изображение по Лапласу F(p), то они тождественно равны.

С помощью операционного исчисления можно сравнительно просто решать различные задачи, сводящиеся к интегрированию линейных дифференциальных уравнений. Переход от исходных функций к их изображениям позволяет заменить решение системы дифференциальных уравнений решением системы алгебраических уравнений (но при этом обратное преобразование Лапласа может быть достаточно сложной задачей)

При вычислении преобразования Лапласа производные заменяются алгебраическими выражениями следующего вида:

$$pF(p) - f(0) = f'(t)$$

$$p^2F(p) - pf(0) - f'(0) = f''(t)$$

и т.д., поэтому использование преобразования Лапласа для решения систем ОДУ требует задания начальных условий.

Использование desolve ограничивается одним из свойств преобразования Лапласа: если L f (t) = F(s), то L t f (t) = −F (s). Поэтому desolve предполагает, что решается система ОДУ с постоянными коэффициентами.

Синтаксис вызова desolve: $desolve(delist, fnlist)$, где $delist$ - список решаемых дифференциальных уравнений, $fnlist$ - список искомых функций. При использовании desolve необходимо явно задавать функциональные зависимости (вместо $'diff(y,x)$ использовать запись $diff(y(x),x)$).

Примеры использования desolve:

Система ОДУ первого порядка:

```
(%i1) de1:diff(f(x),x)=diff(g(x),x)+sin(x);
```

(%o1)
$$\frac{d}{d\,x}\,f\,(x)=\frac{d}{d\,x}\,g\,(x)+sin\,(x)$$

```
(%i2) de2:diff(g(x),x,2)=diff(f(x),x) - cos(x);
```

(%o2)
$$\frac{d^2}{d\,x^2}\,g\,(x)=\frac{d}{d\,x}\,f\,(x)-cos\,(x)$$

```
(%i3) desolve([de1,de2],[f(x),g(x)]);
```

(%o3)
$$[f\,(x)=e^x\,\left(\frac{d}{d\,x}\,g\,(x)\Big|_{x=0}\right)-\frac{d}{d\,x}\,g\,(x)\Big|_{x=0}+f\,(0)\,,g\,(x)=e^x\,\left(\frac{d}{d\,x}\,g\,(x)\Big|_{x=0}\right)-\frac{d}{d\,x}\,g\,(x)\Big|_{x=0}+cos\,(x)+g\,(0)-1]$$

Единичное дифференциальное уравнение второго порядка:

```
(%i1) de3:diff(f(x),x,2)+f(x) = 2*x;
```

(%o1)
$$\frac{d^2}{d\,x^2}\,f\,(x)+f\,(x)=2\,x$$

```
(%i2) desolve(de3,f(x));
```

(%o2)
$$f\,(x)=sin\,(x)\,\left(\frac{d}{d\,x}\,f\,(x)\Big|_{x=0}-2\right)+f\,(0)\;cos\,(x)+2\,x$$

Для указания начальных условий используется функция atvalue. Синтаксис вызова:

```
atvalue (expr, [x_1 = a_1, ..., x_m = a_m], c)
atvalue (expr, x_1 = a_1, c)
```

Функция atvalue присваивает значение c выражению $expr$ в точке $x = a$. Выражение expr - функция $f(x_1, \ldots, x_m)$ или производной

```
diff (f(x_1, ..., x_m), x_1, n_1, ..., x_n, n_m)
```

Здесь n_i - порядок дифференцирования по переменной x_i.

Пример использования desolve и atvalue:

```
(%i1) de1:diff(f(x),x)=diff(g(x),x)+sin(x);
```

(%o1)
$$\frac{d}{d\,x}\,f\,(x)=\frac{d}{d\,x}\,g\,(x)+sin\,(x)$$

```
(%i2) de2:diff(g(x),x,2)=diff(f(x),x) - cos(x);
```

(%o2) $$\frac{d^2}{d\,x^2}\,g\,(x) = \frac{d}{d\,x}\,f\,(x) - cos\,(x)$$

```
(%i3) atvalue(f(x),x=0,1);
```

(%o3) 1

```
(%i4) atvalue(g(x),x=0,2);
```

(%o4) 2

```
(%i5) atvalue(diff(g(x),x),x=0,3);
```

(%o5) 3

```
(%i6) desolve([de1,de2],[f(x),g(x)]);
```

(%o6) $$[f\,(x) = 3\,e^x - 2, g\,(x) = cos\,(x) + 3\,e^x - 2]$$

Управление начальными условиями осуществляется при помощи функций properties и prontprops. Функция properties (синтаксис вызова - properties (a)) печатает свойства переменной (атома a), а функция printprops печатает информацию о заданном свойстве переменной. Кроме того, функция at вычисляет значение выражения в заданной точке с учетом свойства "atvalue". Синтаксис вызова printprops:

```
printprops (a, i)
printprops ([a_1, ..., a_n], i)
printprops (all, i)
```

Данная функция позволяет просмотреть свойства атома a (или группы атомов Lisp, указанных в списке), определённые индикатором i.

Отмена установок, произведённых atvalue, осуществляется функцией remove (удаление свойства p у атомов $a_1, \ldots, a_n$ осуществляется вызовом remove (a_1, p_1, ..., a_n, p_n); удаление списка свойств - вызовом remove ([a_1, ..., a_m], [p_1, ..., p_n], ...)).

Пример синтаксиса и использования рассмотренных функций:

```
(%i1) eq1:'diff(f(x),x)='diff(g(x),x)+sin(x);
```

(%o1) $$\frac{d}{d\,x}\,f\,(x) = \frac{d}{d\,x}\,g\,(x) + sin\,(x)$$

```
(%i2) eq2:'diff(g(x),x,2)='diff(f(x),x)-cos(x);
```

(%o2) $$\frac{d^2}{d\,x^2}\,g\,(x) = \frac{d}{d\,x}\,f\,(x) - cos\,(x)$$

```
(%i3) atvalue('diff(g(x),x),x=0,a);
```

(%o3) a

```
(%i4) atvalue(f(x),x=0,1);
```

(%o4) 1

```
(%i5) properties(f);
```

(%o5) $[atvalue]$

```
(%i6) printprops(f,atvalue);
```

(%o6) $f\left(0\right)=1done$

```
(%i7) desolve([eq1,eq2],[f(x),g(x)]);
```

(%o7) $[f\left(x\right)=a\,e^{x}-a+1,g\left(x\right)=cos\left(x\right)+a\,e^{x}-a+g\left(0\right)-1]$

```
(%i8) at(%,[x=1]);
```

(%o8) $[f\left(1\right)=e\,a-a+1,g\left(1\right)=e\,a-a+cos\left(1\right)+g\left(0\right)-1]$

Ещё один пример анализа свойств:

```
(%i9) atvalue (f(x,y), [x = 0, y = 1], a^2);
```

(%o9) a^2

```
(%i10) atvalue ('diff (f(x,y), x), x = 0, 1 + y);
```

(%o10) $@2+1$

```
(%i11) printprops (all, atvalue);
```

(%o11) $\left.\frac{d}{d\,@1}\,g\left(@1\right)\right|_{[@1=0]}=a\,\left.\frac{d}{d\,@1}\,f\left(@1,@2\right)\right|_{[@1=0]}=@2+1f\left(0,1\right)=a^{2}f\left(0\right)=1done$

4.7.5 Дополнительные возможности решения ОДУ

4.7.5.1 Пакет contrib_ode

Как видно из описания возможностей Maxima выше, возможности основной функции для аналитического решения ОДУ - функции ode2 - весьма ограничены. Для расширения возможностей решения ОДУ первого и второго порядка в последних версиях Maxima существует пакет расширения contrib_ode. При помощи contrib_ode возможно решение уравнений Клеро, Лагранжа, Риккати и др. В общем случае результат - список решений. Для некоторых уравнений (в частности Риккати) решение представляется в форме другого ОДУ - результата замены переменных. Функция contrib_ode реализует методы факторизации (factorization), Клеро (Clairault), Лагранжа(Lagrange), Риккати(Riccati), Абеля(Abel) и метод симметрии Ли (Lie symmetry method). Для использования пакет contrib_ode необходимо загрузить:

```
(%i1) load("contrib_ode");
```

(%o1) $/usr/share/maxima/5.13.0/share/contrib/diffequations/contrib\_ode.mac$

Пример решения ОДУ с использованием функции contrib_ode:

```
(%i2) eqn:x*'diff(y,x)^2-(1+x*y)*'diff(y,x)+y=0;
```

(%o2) $$x\left(\frac{d}{d\,x}\,y\right)^2-(x\,y+1)\,\left(\frac{d}{d\,x}\,y\right)+y=0$$

```
(%i3) contrib_ode(eqn,y,x);
```

(%o3)
$$x\left(\frac{d}{d\,x}\,y\right)^2-(x\,y+1)\,\left(\frac{d}{d\,x}\,y\right)+y=0 firstorderequationnotlinearin y'[y=log\,(x)+\%c, y=\%c\,e^x]$$

```
(%i4) method;
```

(%o4) $factor$

Достоинство contrib_ode - возможность решения нелинейных ОДУ первого порядка. Т.к. они могут иметь в общем случае несколько решений, результат представляется в виде списка. Синтаксис вызова contrib_ode не отличается от синтаксиса вызова ode2.

Рассмотрим примеры решения других типов уравнений.

4.7.5.2 Уравнения Клеро и Лагранжа

Уравнение Клеро

```
(%i1) load("contrib_ode");
```

(%o1) $/usr/share/maxima/5.13.0/share/contrib/diffequations/contrib\_ode.mac$

```
(%i2) eqn:'diff(y,x)^2+x*'diff(y,x)-y=0;
```

(%o2) $$\left(\frac{d}{d\,x}\,y\right)^2+x\,\left(\frac{d}{d\,x}\,y\right)-y=0$$

```
(%i3) contrib_ode(eqn,y,x);
```

(%o3) $$\left(\frac{d}{d\,x}\,y\right)^2+x\,\left(\frac{d}{d\,x}\,y\right)-y=0 firstorderequationnotlinearin y'[y=\%c\,x+\%c^2,y=-\frac{x^2}{4}]$$

```
(%i4) method;
```

(%o4) *clairault*

Уравнение Лагранжа

```
(%i5) leq:y=(1+'diff(y,x))*x+('diff(y,x))^2;
```

(%o5) $$y=\left(\frac{d}{d\,x}\,y\right)^2+x\,\left(\frac{d}{d\,x}\,y+1\right)$$

```
(%i6) contrib_ode(leq,y,x);
```

(%o6)
$$y=\left(\frac{d}{d\,x}\,y\right)^2+x\,\left(\frac{d}{d\,x}\,y+1\right) firstorderequationnotlinearin y'[[x=e^{-\%t}\,\left(\%c-2\,\left(\%t-1\right)\,e^{\%t}\right),y=\left(\%t+1\right)\,x+\%t^2]]$$

```
(%i7) method;
```

(%o7) *lagrange*

В некоторых случаях возможно только решение в параметрической форме. Пример (%t - параметр):

```
(%i8) eqn:'diff(y,x)=(x+y)^2;
```

(%o8) $$\frac{d}{d\,x}\,y=\left(y+x\right)^2$$

```
(%i9) contrib_ode(eqn,y,x);
```

(%o9) $$[[x=\%c-atan\left(\sqrt{\%t}\right),y=-x-\sqrt{\%t}],[x=atan\left(\sqrt{\%t}\right)+\%c,y=\sqrt{\%t}-x]]$$

```
(%i10) method;
```

(%o10) *lagrange*

4.7.5.3 Другие задачи с использованием contrib_ode

Пакет contrib_ode позволяет решать дифференциальные уравнения, не разрешимые при помощи ode2 непосредственно. Пример - обобщённые однородные уравнения (см. выше). Представленные задачи используют методы Абеля и симметрии Ли.

```
(%i11) eqn:(2*x-y+4)*'diff(y,x)+(x-2*y+5)=0;
```

$$(\%o11) \qquad (-y+2\,x+4)\,\left(\frac{d}{d\,x}\,y\right)-2\,y+x+5=0$$

```
(%i12) contrib_ode(eqn,y,x);
```

$$(\%o12) \quad [\frac{log\left(3-\frac{2\,(2\,x+4)-x-5}{-y+2\,x+4}\right)-3\,log\left(1-\frac{2\,(2\,x+4)-x-5}{-y+2\,x+4}\right)+2\,log\left(-\frac{2\,(2\,x+4)-x-5}{4\,(-y+2\,x+4)}\right)}{2}=log\,(x+1)+\%c]$$

```
(%i13) method;
```

$$(\%o13) \qquad abel2$$

```
(%i14) eqn1:'diff(y,x)=(1-3*x-3*y)/(1+x+y);
```

$$(\%o14) \qquad \frac{d}{d\,x}\,y=\frac{-3\,y-3\,x+1}{y+x+1}$$

```
(%i15) contrib_ode(eqn1,y,x);
```

$$(\%o15) \qquad [\frac{2\,log\,(y+x-1)+y+3\,x}{2}=\%c]$$

```
(%i16) method;
```

$$(\%o16) \qquad lie$$

4.7.5.4 Решение однородных линейных уравнений

Другие полезные функции пакета contrib_ode: odelin и ode_check.

Функция odelin решает однородные линейные уравнения первого и второго порядка, и возвращает фундаментальное решение ОДУ. Пример:

```
(%i4) odelin(x*(x+1)*'diff(y,x,2)+(x+5)*'diff(y,x,1)+(-4)*y,y,x);
```

(%o4)

$$...trying\,factor\,method...solving\,7\,equations\,in\,4\,variables...trying\,the\,Bessel\,solver...solving\,1\,equations\,in\,2\,variables...tryin$$

Примечание: функции gauss'a и gauss'b - специальные функции, представляющие собой решения гипергеометрического уравнения.

Функция ode_check позволяет подставить в ОДУ найденное решение. Пример:

```
(%i1) load("contrib_ode");
```

(%o1) $/usr/share/maxima/5.13.0/share/contrib/diffequations/contrib\_ode.mac$

```
(%i2) eqn:(1+x^2)*'diff(y,x,2)-2*x*'diff(y,x);
```

(%o2) $$\left(x^2+1\right)\left(\frac{d^2}{d\,x^2}\,y\right)-2\,x\left(\frac{d}{d\,x}\,y\right)$$

```
(%i3) odelin(eqn,y,x);
```

(%o3) $...trying factor method...solving 7 equations in 4 variables 1, x\left(x^2+3\right)$

```
(%i4) ode_check(eqn,y=x*(x^2+3));
```

(%o4) 0

```
(%i5) ode_check(eqn,y=1);
```

(%o5) 0

4.7.6 Численные методы решения ОДУ

Однако в ряде случаев отыскать символьное решение ОДУ в достаточно компактном виде невозможно. В этом случае целесообразно использовать численные методы. Maxima включает пакет расширения dynamics, позволяющий проинтегрировать систему ОДУ методом Рунге-Кутта.

Начиная с версии 5.12, Maxima включает пакет dynamics (его необходимо загружать перед использованием). Помимо метода Рунге-Кутта, пакет dynamics включает ряд функций для построения различных фракталов.

Метод Рунге-Кутта реализует функция rk. Синтаксис вызова её вызова:

```
rk([eq], [vars],[init],[t_range]),
```

где eq - список правых частей уравнений; vars - список зависимых переменных; init - список начальных значений; t-range - список [t, t0, tend,ht], содержащий символьное обозначение независимой переменной (t), её начальное значение (t0), конечное значение (tend), шаг интегрирования (ht).

Пример:

Решить ОДУ

$$\frac{dx}{dt}=4-x^2-4*y^2; \qquad \frac{dy}{dt}=y^2-x^2+1;$$

при t = [0...4], x(0)=-1,25, y(0)=0,75.

Используем пакет “dynamics ”.

(%i1) load(“dynamics”);

(%o2) /usr/share/maxima/5.12.0/share/dynamics/dynamics.mac

Выбираем шаг интегрирования 0,02. (%i2) sol: rk([4-xˆ2-4*yˆ2,yˆ2-xˆ2+1],[x,y],[-1.25,0.75],[t,0,4,0.02]);

В результате решения получаем список значений в формате [[t,x,y]].

```
(%i1) load("dynamics");
```

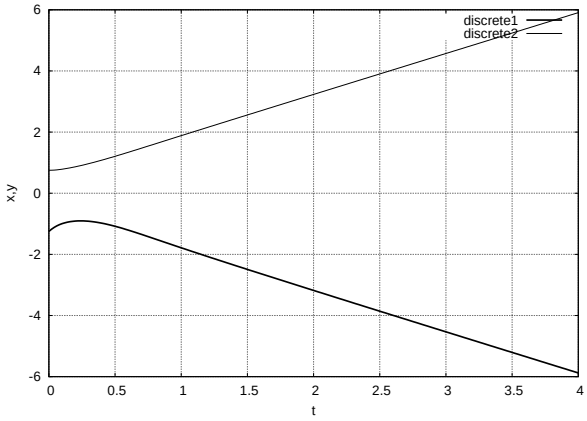


Рис. 4.8. Пример графического решения системы ОДУ численным методом

(%o1) $/usr/share/maxima/5.13.0/share/dynamics/dynamics.mac$

```
(%i2) rp1:4*x^2-4*y^2;
```

(%o2) $$4\,x^2-4\,y^2$$

```
(%i3) rp2:y^2-x^2+1;
```

(%o3) $$y^2-x^2+1$$

```
(%i4) sol:rk([rp1,rp2],[x,y],[-1.25,0.75],[t,0,4,0.02])$
```

Список sol не выводим на экран (он достаточно длинный, поэтому завершаем ввод команды символом $).

Для построения графика решения преобразуем полученный список, построив отдельно список значений t(список xg в примере), x(список yg1), y(список yg2). При построении графика используем опцию discrete.

```
(%i5) len:length(sol);
```

(%o5) 201

```
(%i6) xg:makelist(sol[k][1],k,1,len)$\\
(%i7) yg1:makelist(sol[k][2],k,1,len)$\\
(%i8) yg2:makelist(sol[k][3],k,1,len)$\\
(%i9) plot2d([[discrete,xg,yg1],[discrete,xg,yg2]]);
```

Результат решения представлен на рис. 4.8

Аналогичный, хотя и несколько более сложный пример - моделирование аттрактора Лоренца (см. главу 7).

4.8 Ряды Фурье по ортогональным системам

Пакет Maxima включает достаточно широкие возможности для работы как с классическими тригонометрическими рядами Фурье, так и с рядами Фурье по другим ортогональным системам. Рассмотрим краткое введение, необходимое для понимания приводимых примеров.

4.8.1 Понятие ряда Фурье

Пусть в бесконечномерном пространстве E со скалярным произведением дана ортогональная система (φ_k), т.е. $\varphi_k \neq 0$, k=1,2,...; (φ_k, φ_l)=0 при l ≠ k. Ряд вида $\sum_{k=1}^{\infty} \alpha_k \varphi_k$ называется рядом по ортогональной системе (φ_k). Пусть x∈E. Числа $C_k = \frac{(x,\varphi_k)}{\|\varphi_k\|^2}$ коэффициентами Фурье элемента x по ортогональной системе (φ_k), а ряд $\sum_{k=1}^{\infty} c_k \varphi_k$ называется рядом Фурье (по ортогональной системе (φ_k)), составленным для элемента x (ряд элемента x). Многочлен $\sum_{k=1}^{n} c_k \varphi_k$ - частичная сумма ряда Фурье – называется многочленом Фурье (элемента x).

Наилучшее приближение элемента x посредством элементов из L_n есть многочлен Фурье элемента x: $\sum_{k=1}^{n} c_k \varphi_k$.

Для суммы ряд Фурье элемента x выполняется неравенство Бесселя:

$$\sum_{k=1}^{\infty} |c_k|^2 \|\varphi_k\|^2 \leq \|x_k\|^2$$

Следствие:

Если $\|\varphi_k\| \geq \alpha > 0,$, $k = 1, 2, ...$, то коэффициенты Фурье c_k любого элемента $\in E$ стремятся к нулю при $k \to \infty$.

4.8.2 Равенство Парвеваля-Стеклова

Ортогональная система (φ_k) из гильбертова пространства H называется полной, если для любого $x \in H$ $\sum_{k=1}^{\infty} c_k \varphi_k = x$ (ряд Фурье, составленный для x, сходится к x).

Полная ортогональная система является ортогональным базисом гильбертова пространства H.

Для того чтобы система (φ_k) была полной, необходимо и достаточно, чтобы

$$\sum_{k=1}^{\infty} |c_k|^2 \|\varphi_k\|^2 = \|x\|^2$$

Таким образом, в случае полной системы (φ_k) (и только в этом случае) неравенство Бесселя превращается в равенство. Это равенство называется равенством Парсеваля-Стеклова. Заметим, что полнота ортогональной системы означает, что ее нельзя дополнить до более широкой ортогональной системы путем присоединения новых элементов.

Ортогональная нормированная система (φ_k) называется замкнутой, если для любого $x \in E$ справедливо равенство

$$\sum_{n=1}^{\infty} {c_k}^2 = \|x\|$$

Замкнутость системы (φ_k) равносильна тому, что для каждого $f \in H$ частичные суммы ряда Фурье $\sum_{k=1}^{\infty} c_n \varphi_n$ сходятся к f.

Понятие замкнутости ортогональной нормированной системы тесно связано с понятием полноты системы.

Пример: В гильбертовом пространстве $L_2[-\pi;\pi]$ функций суммируемых с квадратом модуля на отрезке $[-\pi;\pi]$ семейство тригонометрических функции 1, cos t, sin t, sin 2t,...образуют ортогональный базис. Однако эта система функций не является нормированной.

4.8.3 ОРТОГОНАЛЬНОСТЬ ФУНКЦИЙ

Пусть даны две функции f(x) и g(x), произведение которых интегрируемо на отрезке [a,b].

Функции f(x) и g(x), называются ортогональными на [a,b], если

$\int_a^b f(x)g(x)\rho(x)dx = 0$, где $\rho(x)$ - весовая функция.

Функциональная последовательность

$\{\varphi_n(x)\} = \{\varphi_0(x), \varphi_1(x), ..., \varphi_n(x), ...\}$ называется ортогональной на $[a, b]$, если $\int_a^b \varphi_n(x)\,\varphi_m(x)\,\rho(x)\,dx = 0$, $\forall n \neq m$.

Функциональная последовательность $\{\varphi_n(x)\}$ называется ортонормированной на $[a, b]$, если

$$\int_a^b \varphi_n(x)\,\varphi_m(x)\,\rho(x)\,dx = \begin{cases} 1, & \text{если } n = m \\ 0, & \text{если } n \neq m \end{cases}$$

Часто используемая последовательность тригонометрических функций
1, cos(x), sin(x), cos(2x), sin(2x),..., cos(nx), sin(nx), ...
ортогональна на отрезке $[-\pi, \pi]$ с весовой функцией $\rho(x)$.

Проверим свойство ортогональности, вычисляя соответствующие интегралы. При $m \neq n$ получаем:

$$\int_{-\pi}^{\pi} 1 \cdot \sin(nx)dx = -\frac{\cos(nx)}{n}|_{-\pi}^{\pi} = 0, \tag{4.1}$$

$$\int_{-\pi}^{\pi} 1 \cdot \sin(nx)dx = -\frac{\cos(nx)}{n}|_{-\pi}^{\pi} = 0, \tag{4.2}$$

$$\int_{-\pi}^{\pi} \cos(nx)dx = \frac{1}{n}\sin(nx)|_{-\pi}^{\pi} = 0, \forall n \in N; \tag{4.3}$$

$$\int_{-\pi}^{\pi} \cos(mx) \cdot \cos(nx)dx = \tag{4.4}$$

$$\frac{1}{2}\int_{-\pi}^{\pi} (\cos((m-n)x) + \cos((m+n)x))dx = \tag{4.5}$$

$$\frac{1}{2}((\frac{\sin((m-n)x)}{m-n} + \frac{\sin(m+n)x}{m+n})|_{-\pi}^{\pi} = 0 \tag{4.6}$$

Если же m = n , то

$$\int_{-\pi}^{\pi} \cos^2(mx)dx = \frac{1}{2}\int_{-\pi}^{\pi} (1 + \cos(2mx))\,dx = \frac{1}{2}\left(x + \frac{\sin(2mx)}{2m}\right)\Big|_{-\pi}^{\pi} = \pi.$$

Следовательно, $\int_{-\pi}^{\pi} \cos(mx)\cos(nx)dx = \begin{cases} 0, m \neq n; \\ \pi, m = n. \end{cases}$. Аналогичным образом устанавливаем, что $\int_{-\pi}^{\pi} \sin(mx)\sin(nx)dx = \begin{cases} 0, m \neq n; \\ \pi, m = n \end{cases}$.

Остаётся вычислить интеграл $\int_{-\pi}^{\pi} \cos(mx)\sin(nx)dx$.

Поскольку подынтегральная функция является нечётной, то

$$\int_{-\pi}^{\pi} \cos(mx)\sin(nx)dx = 0,$$

Как следует из приведённых равенств, любые две различные функции тригонометрической последовательности ортогональны на отрезке $[-\pi, \pi]$.

4.8.4 Вычисление коэффициентов тригонометрических рядов Фурье

Члены тригонометрического ряда $\frac{0_0}{2} + \sum_{n=1}^{\infty}(a_n\cos(nx) + b_n\sin(nx))$ являются периодическими функциями с общим периодом 2π, поэтому и сумма этого ряда S (x) также будет периодической функцией с периодом 2π.

Возникает вопрос: любую ли периодическую с периодом 2π функции можно представить в виде тригонометрического ряда Фурье? Ответ на этот вопрос дадим позднее.

Теперь же допустим, что 2π – периодическую функцию f (x) можно разложить в тригонометрический ряд Фурье, равномерно сходящийся на отрезке

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n\cos(nx) + b_n\sin(nx)) \tag{4.7}$$

Рассмотрим вопрос об определении коэффициентов a_0, a_n и b_n $(n = 1, 2, ...)$. Для этого применим теорему о почленном интегрировании функционального ряда. Проинтегрируем обе части равенства в пределах от $-\pi$ до π :

$$\int_{-\pi}^{\pi} f(x)\,dx = \frac{a_0}{2}\int_{-\pi}^{\pi} dx + \sum_{n=1}^{\infty}\left(a_n\int_{-\pi}^{\pi}\cos(nx)dx + b_n\int_{-\pi}^{\pi}\sin(nx)dx\right).$$

Из результатов вычисления интегралов, приведённых выше, следует, что все слагаемые, встречающиеся в правой части под знаком суммы равны нулю, поэтому

$$\int_{-\pi}^{\pi} f(x)dx = \pi a_0.$$

Следовательно,

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\,dx. \tag{4.8}$$

Для того чтобы найти $a_n\,(n = 1, 2, ...)$, обе части этого равенства умножим на cos(mx) и проинтегрируем на отрезке $[-\pi, \pi]$. Поскольку система тригонометрических функций ортогональна, то

$$\int_{-\pi}^{\pi}\cos(mx)\cos(nx)dx = 0, \int_{-\pi}^{\pi}\cos(mx)\sin(nx)dx = 0$$

для $\forall m, n \in \mathrm{N}$, если $m \neq n$.

Это означает что все с интегралы, встречающиеся в правой части, будут равны нулю; исключение составляет интеграл, который получается при m = n. Этот интеграл равен π. Поэтому

$$\int_{-\pi}^{\pi} f(x)\cos(nx)dx = a_n\int_{-\pi}^{\pi}\cos^2(nx)dx = \pi a_n,$$

откуда $a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx$, $n = 1, 2, \ldots$

Аналогично, умножив обе части равенства на sin(mx) и проинтегрировав на отрезке $[-\pi; \pi]$, получаем, что

$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx$, $n = 1, 2, \ldots$

Итак, если функцию f(x) можно представить в виде тригонометрического ряда, то коэффициенты a_n, a_n, b_n вычисляются по приведённым формулам и называются коэффициентами Фурье для функции f(x) (а ряд - соответственно рядом Фурье для f(x)).

Промежуток интегрирования $[-\pi, \pi]$ для периодической с периодом 2π функции можно заменить любым промежутком $[a, a + 2\pi]$, $a \in R$, длина которого равна 2π.

Функция f(x) называется кусочно-гладкой на отрезке [a,b] если функция f(x) и её производная на [a,b] имеют конечное число точек разрыва первого рода.

Достаточные условия разложимости функции в ряд Фурье даёт теорема Дирихле:.

Если f(x) - периодическая с периодом 2π кусочно-гладкая на $[-\pi; \pi]$ функция, то её ряд Фурье сходится в любой точке этого отрезка и его сумма равна:

1. функция f(x), когда x – точка непрерывности функции f(x);
2. $\frac{f(x-0)+f(x+0)}{2}$, когда x – точка разрыва функции f(x)

$$\frac{f(x-0)+f(x+0)}{2} = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n\cos(nx) + b_n\sin(nx)\right).$$

Отметим, что на практике чаще всего втречаются функции, которые удовлетворяют условиям теоремы Дирихле.

Пример: периодическую с периодом 2π функцию $f(x) = x$, $-\pi < x < \pi$ разложить в ряд Фурье.

Вычислим коэффициенты Фурье (используем Maxima):

```
(%i1) n:5;
```

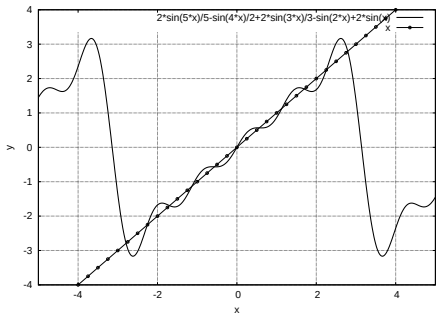


Рис. 4.9. График функции y=f(x) и суммы первых пяти членов ряда Фурье

(%o1) 5

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

(%o2) $f(x) := x$

```
(%i3) a0:1/%pi*integrate(f(x),x,-%pi,%pi);
```

(%o3) 0

```
(%i4) for k:1 thru n do a[k]:1/%pi*integrate(f(x)*cos(k*x),x,-%pi,%pi);
```

(%o4) *done*

```
(%i5) for k:1 thru n do b[k]:1/%pi*integrate(f(x)*sin(k*x),x,-%pi,%pi);
```

(%o5) *done*

```
(%i6) for k:1 thru n do display(a[k],b[k]);
```

(%o6) $a_1 = 0 b_1 = 2 a_2 = 0 b_2 = -1 a_3 = 0 b_3 = \frac{2}{3} a_4 = 0 b_4 = -\frac{1}{2} a_5 = 0 b_5 = \frac{2}{5} done$

```
(%i7) fun(x):=a0/2+sum(a[k]*cos(k*x),k,1,n)+sum(b[k]*sin(k*x),k,1,n);
```

(%o7) $fun(x) := \frac{a0}{2} + sum(a_k \, cos(k\,x), k, 1, n) + sum(b_k \, sin(k\,x), k, 1, n)$

```
(%i8) wxplot2d([f(x),fun(x)], [x,-5,5],
 [nticks,20]);
```

Данная функция f(x) удовлетворяет условиям теоремы Дирихле, её график изображён на рис.4.9.

4.8.5 Ряды Фурье для чётных и нечётных функций

Предположим, что f(x) – нечётная 2π- периодическая функция. В этом случае f(x)cos(nx) – чётная функция, поскольку f(-x)cos(-nx)=f(x)cos(nx), а f(x)sin(nx) – нечётная функция, так как f(-x)sin(-nx)= - f(x)sin(nx). Поэтому коэффициент ряда Фурье a_n, b_n равны:

$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx = \frac{2}{\pi}\int_{0}^{\pi} f(x)\cos(nx)dx$ (n=0,1,...),

$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx = 0$ (n=1,2,...).

Следовательно, ряд Фурье чётной функции содержит только косинусы, т.е.

$$f(x) = \frac{0_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx).$$

Аналогично, если f(x) – нечётная функция, то f(x)cos(nx) – нечётная, а f(x)sin(nx) – чётная функция.

Поэтому $a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx) = 0$ (n=0,1,...),

$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx = \frac{2}{\pi}\int_{0}^{\pi} f(x)\sin(nx)dx$ (n=1,2,...).

Следовательно, ряд Фурье нечётной функции содержит только синусы, т.е.

$$f(x) = \sum_{n=1}^{\infty} b_n \sin(nx).$$

ПРИМЕР. Разложить в ряд Фурье периодическую с периодом 2πфункцию, заданную на отрезке $[-\pi, \pi]$ равенством f(x)=x^2.

Данная функция является чётной (рис.), поэтому её ряд Фурье содержит только косинусы. Вычисляем коэффициенты этого ряда:

$$b_n = 0,\ n = 1, 2, \ldots$$

Для вычисления коэффициентов ряда Фурье создаём функцию fun, входными параметрами которой являются имя независимой переменной, число суммируемых членов ряда и символьное выражение, определяющее функцию, для которой строится разложение. Пример:

```
(%i1) fun(x,n,f):=(for k:0 thru n do a[k]:1/%pi*integrate(f*cos(k*x),x,-%pi,%pi),a[0]/2
 +sum(a[k]*cos(k*x),k,1,n));
```

(%o1)

$$fun\,(x, n, f) := (fork from 0 thru do a_k : \frac{1}{\pi}\, integrate\,(f\, cos\,(k\, x)\,, x, -\pi, \pi)\,, \frac{afun(x, n, f1, f2) := (fork : 0 thru do a[k] : 1}{2}$$

```
(%i2) fun(x,5,x^2);
```

(%o2)
$$-\frac{4\,cos\,(5\,x)}{25} + \frac{cos\,(4\,x)}{4} - \frac{4\,cos\,(3\,x)}{9} + cos\,(2\,x) - 4\,cos\,(x) + \frac{\pi^2}{3}$$

Для аналитического вычисления коэффициентов ряда Фурье функции $y = |x|$ немного функцию fun необходимо немного изменить, предусмотрев различные выражения для подинтегральной функции на полуинтервалах $[-\pi, 0)$ и $(0, \pi]$ (выражения f1 и f2 в списке параметров функции). Текст программы на макроязыке Maxima:

```
fun12(x,n,f1,f2):=(for k:0 thru n do
a[k]:1/%pi*(integrate(f1*cos(k*x),x,-%pi,0)+
             integrate(f2*cos(k*x),x,0,%pi)),
a[0]/2+sum(a[k]*cos(k*x),k,1,n)
)
```

Функция является $y = |x|$ также является чётной (рис.), поэтому её ряд Фурье содержит только косинусы. Результаты вычисления коэффициентов ряда Фурье для этой функции:

```
(%i1) fun12(x,5,-x,x);
```

(%o1) $$-\frac{4\,cos\,(5\,x)}{25\,\pi}-\frac{4\,cos\,(3\,x)}{9\,\pi}-\frac{4\,cos\,(x)}{\pi}+\frac{\pi}{2}$$

```
(%i2) fun12(x,7,-x,x);
```

(%o2) $$-\frac{4\,cos\,(7\,x)}{49\,\pi}-\frac{4\,cos\,(5\,x)}{25\,\pi}-\frac{4\,cos\,(3\,x)}{9\,\pi}-\frac{4\,cos\,(x)}{\pi}+\frac{\pi}{2}$$

4.8.6 Разложение функций в ряд Фурье на отрезке [0,π]

Пусть f(x) определена на отрезке $[0,\pi]$. Для того, чтобы функцию f(x) разложить в ряд Фурье на этом отрезке, доопределим эту функцию произвольным образом на отрезке $[-\pi,0]$. Таким образом, получаем функцию, которая$y=|x|$ уже определена на отрезке $[-\pi,\pi]$. Рассмотрим два случая:

Функцию f(x) , заданную на $[0,\pi]$, продолжим на отрезок $[-\pi,0]$ так, что вновь полученная функция $f_1(x)$, была чётной:

$$f_1(x)=\begin{cases}f(x), & x\in[0,\pi],\\ f(-x), & x\in[-\pi,0].\end{cases}$$

В таком случае говорят, что f(x) продолжена на $[-\pi,0]$ чётным образом. Поскольку $f_1(x)$ - чётная на $[-\pi,\pi]$ функция, то её ряд Фурье содержит только косинусы:

$$f_1(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}a_n\cos(nx).$$

Поскольку на отрезке $[0,\pi]$ имеет место равенство $f_1(x)=f(x)$, то ряд Фурье для функции $f_1(x)$ будет и рядом Фурье для f(x) на $[0,\pi]$

Функцию f(x), заданную на $[0,\pi]$, продолжим на отрезок $[-\pi,0]$ нечётным образом (рис.4):

$$f_2(x)=\begin{cases}f(x),5A;8x\in[0,\pi],\\ -f(-x),5A;8x\in[-\pi,0[.\end{cases}$$

Итак,

$$x=\frac{\pi}{2}-\frac{4}{\pi}\sum_{k=0}^{\infty}\frac{\cos((2k+1)x)}{(2k+1)^2},x\in[0,\pi]$$

Поскольку $f_2(x)$ - нечётная на $[-\pi,\pi]$ функция, то её ряд Фурье содержит только синусы:

$$f_2(x)=\sum_{n=1}^{\infty}b_n\sin(nx).$$

Так как $f_2(x)=f(x)$ при $\forall x\in[0,\pi]$, то полученный ряд Фурье для $f2_(x)$ и будет рядом Фурье для f(x) на $[0,\pi]$.

Пример: Функцию f(x)=2x+1, определённую на отрезке $[0,\pi]$, разложить в ряд Фурье: 1)по косинусам; 2)по синусам.

1) Функцию f(x) продолжим на $[-\pi,0]$ чётным образом, т.е.составим новую функцию f1(x) по формуле

$$f_1(x)=\begin{cases}-2x+1,x\in[-\pi,0]\\ 2x+1,x\in[0.\pi].\end{cases}$$

Вычисляем коэффициенты Фурье для этой функции при помощи функции fun12:

```
(%i1) fleft:-2*x+1;
```

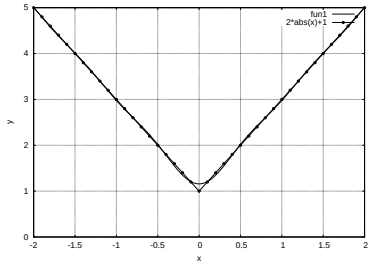


Рис. 4.10. График функции y=2x+1, продолженной чётным образом, и суммы семи членов соответствующего ряда

(%o1) $1-2\,x$

```
(%i2) fright:2*x+1;
```

(%o2) $2\,x+1$

```
(%i3) funcos(x,7,fleft,fright);
```

(%o3) $$-\frac{8\,cos\left(7\,x\right)}{49\,\pi}-\frac{8\,cos\left(5\,x\right)}{25\,\pi}-\frac{8\,cos\left(3\,x\right)}{9\,\pi}-\frac{8\,cos\left(x\right)}{\pi}+\frac{2\,\pi^2+2\,\pi}{2\,\pi}$$

Графическое сопоставление результатов суммирования ряда Фурье и аналитического выражения заданной функции представлены на рис.4.10

2) Функцию f(x) продолжим на $[-\pi,0]$ нечётным образом составим новую функцию $f_2(x)$ по формуле $f_2(x)=2x+1, x\in[-\pi,\pi]$.

Вычислим коэффициенты Фурье для этой функции, используя функцию fun12sin, аналогичгую приведённой выше. Пример:

```
(%i1) fleft:2*x-1;
```

(%o1) $2\,x-1$

```
(%i2) fright:2*x+1;
```

(%o2) $2\,x+1$

```
(%i3) f(x):=(if x>0 then fright else fleft);
```

(%o3) $$f\left(x\right):=ifx>0thenfrightelsefleft$$

```
(%i4) fun12sin(x,n,f1,f2):=(for k:1 thru n do b[k]:1/%pi*(integrate(f1*sin(k*x),x,-%pi,0)
 +integrate(f2*sin(k*x),x,0,%pi)),sum(b[k]*sin(k*x),k,1,n));
```

(%o4)

$$fun12sin\left(x,n,f1,f2\right):=\left(forkthrundob_k:\frac{1}{\pi}\left(integrate\left(f1\,sin\left(k\,x\right),x,-\pi,0\right)+integrate\left(f2\,sin\left(k\,x\right),x,0,\pi\right)\right),sum\left(\right.\right.$$

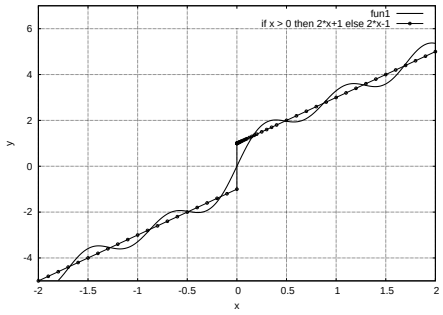


Рис. 4.11. Сравнение графика функции y=2x+1 при нечётном продолжении и суммы семи членов соответствующего ряда Фурье

```
(%i5) fun12sin(x,7,fleft,fright);
```

(%o5)

$$\frac{\left(\frac{2\,(2\,\pi+1)}{7}+\frac{2}{7}\right)\,sin\,(7\,x)}{\pi}+\frac{\left(\frac{1}{3}-\frac{2\,\pi+1}{3}\right)\,sin\,(6\,x)}{\pi}+\frac{\left(\frac{2\,(2\,\pi+1)}{5}+\frac{2}{5}\right)\,sin\,(5\,x)}{\pi}+\frac{\left(\frac{1}{2}-\frac{2\,\pi+1}{2}\right)\,sin\,(4\,x)}{\pi}+\frac{\left(\frac{2\,(2\,\pi+1)}{3}+\frac{2}{3}\right)\,sin\,(3\,x}{\pi}$$

Графическое сопоставление результатов суммирования ряда Фурье и аналитического выражения заданной функции представлены на рис.4.11

4.8.7 Ряд Фурье для функций с периодом 2l

Пусть f(x) – периодичная с периодом 2ℓ ($\ell \neq \pi$) функция, которая на отрезке $[-\ell, \ell]$ удовлетворяет условиям теоремы Дирихле. Разложим её на этом отрезке в ряд Фурье. Обозначим

$$x = \frac{\ell t}{\pi}. \tag{4.9}$$

Тогда

$$f(x) = f\left(\frac{\ell t}{\pi}\right) = \varphi(t)$$

Функция $\varphi(t)$- уже 2π- периодическая функция, так как

$$\varphi(t+2\pi) = f\left(\frac{\ell}{\pi}(t+2\pi)\right) = f\left(\frac{\ell t}{\pi}+2\ell\right) = f\left(\frac{\ell t}{\pi}\right) = \varphi(t).$$

Функцию $\varphi(t)$разложим в ряд Фурье на отрезке $[-\pi, \pi]$

$$\varphi(t) = f\left(\frac{\ell t}{\pi}\right) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos(nt) + b_n \sin(nt)\right).. \tag{4.10}$$

Коэффициенты этого ряда вычисляются по формулам:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\cos(nt)dt, n = 0, 1, \ldots, \tag{4.11}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\sin(nt)dt, n = 1, 2, ... \tag{4.12}$$

Возвращаясь к прежней переменной x, из равенства (4.9) имеем $t = \frac{\pi x}{\ell}$. Тогда ряд

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos(\frac{n\pi x}{\ell}) + b_n \sin(\frac{n\pi x}{\ell})\right). \tag{4.13}$$

В интегралах (4.11) и (4.12) произведём замену переменной:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\cos(nt)dt = \begin{cases} t=\frac{\pi x}{\ell} \\ dt=\frac{\pi}{\ell}dx \end{cases}$$

$= \frac{1}{\ell}\int_{-\ell}^{\ell} f(x)\cos(\frac{n\pi x}{\ell})dx$, n=0,1,...

Аналогично,Пример вычислений с Maxima:

$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\sin(t)dt = \frac{1}{\ell}\int_{-\ell}^{\ell} f(x)\sin(\frac{n\pi x}{\ell})dx$, n=1,2,... (**??**)

Если f(x) - чётная на $[-\ell,\ell]$ функция, то $b_n = 0$ (n=1,2,...)

$a_n \frac{2}{\ell}\int_0^{\ell} f(x)\cos(\frac{n\pi x}{\ell})dx$, n=0,1,...

Ряд Фурье такой функции имеет вид:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\frac{n\pi x}{\ell}).$$

Если f(x) – нечётная на $[-\ell,\ell]$ функция, то an=0 (n=0,1,2,...),

$b_n = \frac{2}{\ell}\int_0^{\ell} f(x)\sin(\frac{n\pi x}{\ell})dx$, n=1,2,...,

а сам ряд Фурье имеет вид:

$$f(x) = \sum_{n=1}^{\infty} b_n \sin(\frac{n\pi x}{\ell})dx.$$

Пример: Разложить в ряд Фурье периодическую с периодом T=2 функцию f(x), заданную формулой

$$f(x) = \begin{cases} 0, -1<x\leq 0; \\ x, 0<x\leq 1. \end{cases}$$

Эта функция на отрезке $[-1,1]$ удовлетворяет условиям теоремы Дирихле. Ряд Фурье для данной функции:

$$f(x) = \frac{1}{4} - \frac{2}{\pi^2}\sum_{k=0}^{\infty}\frac{\cos((2k+1)x)}{(2k+1)^2} + \frac{1}{\pi}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}\sin(k\pi x)$$

Сумма этого ряда в точках x=±1,±3,... равна $\frac{1}{2}$.

Рассмотрим видоизменение функции Maxima, необходимой для вычисления коэффициентов ряда Фурье для функции с периодом $[-\ell,\ell]$. Рассмотрим текст функции fun12l:

```
fun12l(x,n,l,f1,f2):=(for k:0 thru n do        a[k]:1/l*(integrate(f1*cos(%pi*k*x/l),x,-l,0)
 +integrate(f2*cos(%pi*k*x/l),x,0,l)),
for k:1 thru n do b[k]:1/l*(integrate(f1*sin(%pi*k*x/l),x,-l,0)+
integrate(f2*sin(%pi*k*x/l),x,0,l)),a[0]/2+sum(a[k]*cos(%pi*k*x/l),k,1,n)+
sum(b[k]*sin(%pi*k*x/l),k,1,n))$
```

Основное изменение по сравнение с вариантами, приведёнными выше - использование тригонометрических функций $\sin\left(\frac{\pi kx}{\ell}\right)$ и $\sin\left(\frac{\pi kx}{\ell}\right)$.

Вывод Maxima для первых семи членов ряда Фурье:

```
(%i6) fun12l(x,7,1,0,x);
```

(%o6)

$$\frac{sin\,(7\,\pi\,x)}{7\,\pi}-\frac{2\,cos\,(7\,\pi\,x)}{49\,\pi^2}-\frac{sin\,(6\,\pi\,x)}{6\,\pi}+\frac{sin\,(5\,\pi\,x)}{5\,\pi}-\frac{2\,cos\,(5\,\pi\,x)}{25\,\pi^2}-\frac{sin\,(4\,\pi\,x)}{4\,\pi}+\frac{sin\,(3\,\pi\,x)}{3\,\pi}-\frac{2\,cos\,(3\,\pi\,x)}{9\,\pi^2}-\frac{sin\,(2\,\pi\,x)}{2\,\pi}+\frac{sin\,(}{\pi}$$

Для построения графика собственно анализируемой функции (её представляет кусочно-непрерывная функция f(x)) и частичной суммы её ряда Фурье из результатов разложения формируем новую функцию g(x), после чего стандартной командой строим график:

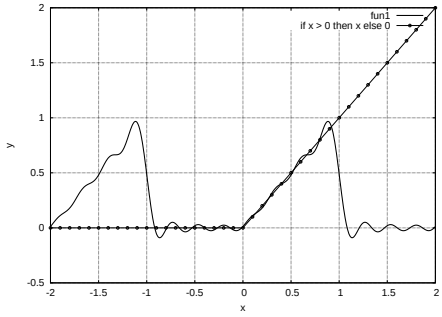


Рис. 4.12. График функции f(x)=0, -1<x<0; x, 0<x<1 и суммы первых семи членов ряда Фурье

```
(%i7) g(x):=''%$

(%i8) f(x):=(if x<0 then 0 else x)$

(%i9) wxplot2d([g(x),f(x)], [x,-2.2,1.6]);
```

Графическая иллюстрация, показывающая сопоставление рассматриваемой функции и ряда Фурье на заданном отрезке - на рис. 4.12.

4.8.8 Комплексная форма ряда Фурье

Пусть функция f(x) на $[-\pi, \pi]$ разложена в ряд Фурье

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos(x) + b_n \sin(x)\right). \tag{4.14}$$

Воспользуемся формулами Эйлера:

$$\cos(nx) = \frac{e^{inx} + e^{-inx}}{2}, \sin(nx) = \frac{e^{inx} - e^{-inx}}{2i}.$$

Подставим эти выражения в ряд (4.14), имеем:
$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \frac{e^{inx}+e^{-inx}}{2} + b_n \frac{e^{inx}-e^{-inx}}{2i}\right) = \frac{a_0}{2} +$
$+ \sum_{n=1}^{\infty} \left(a_n \frac{e^{inx}+e^{-inx}}{2} - ib_n \frac{e^{inx}-e^{-inx}}{2}\right) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n - ib_n}{2} \cdot e^{inx} + \frac{a_n + ib_n}{2} \cdot e^{-inx}\right).$
Обозначим:

$$\frac{a_0}{2} = c_0, \frac{a_n - ib_n}{2} = c_n, \frac{a_n + ib_n}{2} = c_{-n}. \tag{4.15}$$

Тогда

$$f(x) = c_0 + \sum_{n=1}^{\infty} \left(c_n \cdot e^{inx} + c_{-n} e^{-inx}\right) = c_0 + \sum_{n=1}^{\infty} c_n e^{inx} + \sum_{n=1}^{\infty} c_{-n} e^{-inx} =$$
$$= c_0 + \sum_{n=1}^{\infty} c_n e^{inx} + \sum_{n=-\infty}^{\infty} c_n e^{inx} = \sum_{n=-\infty}^{\infty} c_n e^{inx}.$$

Итак, получили

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}.$$

Выражение называется комплексной формой ряда Фурье функции f(x) с комплексными коэффициентами Фурье $_n$.

Коэффициенты Фурье cn выразим через интегралы.

$$c_n = \frac{1}{2}(a_n - ib_n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)\cos(nx) - i\sin(nx)dx =$$
$$\frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)\left[\cos(-nx) + i\sin(-nx)\right]dx =$$
$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx.$$

Эта формула верна при n=0, ±1,±2,...

Если f(x) – периодическая с периодом 2ℓ функция, то её комплексный ряд Фурье имеет вид:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{in\pi x}{\ell}},$$

а коэффициенты Фурье определяются по формуле

$$c_n = \frac{1}{2\ell}\int_{-\ell}^{\ell} f(x)e^{\frac{-in\pi x}{\ell}}dx.$$

4.8.9 Дополнительные возможности: пакет fourie

Пакет расширения fourie предназначен для расчёта коэффициентов тригонометрических рядов Фурье, а также интеграла Фурье. Функции, входящие в состав пакета, позволяют находить точное аналитическое выражение всех, а не первых нескольких коэффициентов ряда Фурье.

Вычислить коэффициенты ряда Фурье позволяет функция fourier (синтаксис вызова fourier (f, x, p)), которая возвращает список коэффициентов Фурье f(x), определенных на интервале [-p, p]. Собственно ряд Фурье позволяет построить функция fourexpand (синтаксис вызова fourexpand (l, x, p, limit)), которая конструирует и возвращает ряд Фурье, используя список коэффициентов Фурье l (limit может быть и бесконечным, равным inf).

Коэффициенты рядов Фурье по синусам и по косинусам вычисляются функциями fourcos (f, x, p) foursin (f, x, p) (синтаксис и аналогичны функции fourier).

Вычисления и подстановка $\cos n\pi$ и $\sin n\pi$ осуществляется специальной функцией foursimp (l). Управление подстановкой осуществляется посредством флагов sinnpiflag и cosnpiflag (если они установлены в true, вычисление и подстановка выполняются, это режим по умолчанию).

Для управления процессом разложения различных функций в ряд Фурье предусмотрены следующие функции:

1. remfun. Синтаксис вызова remfun (f, expr) или remfun (f, expr, x). Данная функция позволяет заменить все вхождения функции f(arg) в выражении expr на arg.(в форме remfun (f, expr, x) замена осуществляется, только если arg содержит x;

2. funp. Данная функция (синтаксис вызова funp (f, expr) или funp (f, expr, x)) возвращает true, если выражение expr содержит функцию f или конкретно f(x);

3. absint. Данная функция позволяет вычислить неопределённый или определённый интеграл абсолютных значений функции f (её определение может включать выражения abs (x), abs (sin (x)), abs (a) * exp (-abs (b) * abs (x)). Синтаксис вызова absint (f, x, halfplane) (halfplane = (pos, neg, both) - часть числовой оси), absint (f, x) (неопределённый интеграл по положительной полуоси), absint (f, x, a, b) (определённый интеграл).

Общую форму ряда Фурье (после подстановки и упрощения) позволяет построить функция totalfourier (f, x, p).

Коэффициенты интеграла Фурье на интервале $(-\inf, \inf)$ позволяет вычислить функция fourint (f, x), интеграла по косинусам или синусам на интервале $(0, \inf)$ - функции fourintcos (f, x) и fourintsin (f, x) соответственно.

Для испоьзования пакета fourie его необходимо предварительно загрузить командой

```
load("fourie")
```

Примеры использования пакета fourie:

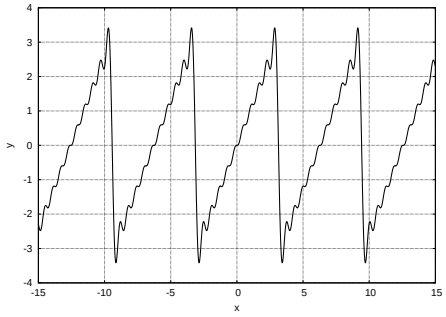


Рис. 4.13. График частичной суммы ряда Фурье для функции f(x)=x

```
(%i1) load("fourie");
```

(%o1) $/usr/share/maxima/5.13.0/share/calculus/fourie.mac$

```
(%i2) fourier(x,x,%pi);
```

(%t3) $$a_0 = 0a_n = 0Iscos\left(\pi\, n\right) positive, negative, orzero?$$

```
p;
```

(%o4) $$b_n = \frac{2\left(\frac{sin(\pi\, n)}{n^2} - \frac{\pi\, cos(\pi\, n)}{n}\right)}{\pi}[\%t2, \%t3, \%t4]$$

```
(%i5) foursimp(%);
```

(%o7) $$a_0 = 0a_n = 0b_n = -\frac{2\left(-1\right)^n}{n}[\%t5, \%t6, \%t7]$$

```
(%i8) fourexpand(%,x,%pi,10);
```

(%o8)
$$-\frac{sin\left(10\, x\right)}{5}+\frac{2\, sin\left(9\, x\right)}{9}-\frac{sin\left(8\, x\right)}{4}+\frac{2\, sin\left(7\, x\right)}{7}-\frac{sin\left(6\, x\right)}{3}+\frac{2\, sin\left(5\, x\right)}{5}-\frac{sin\left(4\, x\right)}{2}+\frac{2\, sin\left(3\, x\right)}{3}-sin\left(2\, x\right)+2\, sin\left(x\right)$$

График полученной функции приведен на рис.4.13

Глава 5

Численные методы и программирование с Maxima

5.1 Программирование на встроенном макроязыке

5.1.1 Условные операторы

Основная форма условного оператора $if cond_1 then expr_1 else expr_0$. Если условие $cond_1$ истинно, то выполняется выражение $expr_1$, иначе - выполняется выражение $expr_2$. Пакет Maxima позволяет использовать различные формы оператора if, например:

$$if cond_1 then expr_1 else if cond_2 then expr_2 else if ... else expr_0$$

Если выполняется условие $cond_1$, то выполняется выражение $expr_1$, иначе - проверяется условие $cond_2$, и если оно истинно - выполняется выражение $expr_2$, и т.д. Если ни одно из условий не является истинным - выполняется выражение $expr_0$.

Альтернативные выражения $expr_1, expr_2, ..., expr_k$ - произвольные выражения Maxima (в т.ч. вложенные операторы if). Условия - действительно или потенциально логические выражения, сводимые к значениям true или false. Способ интерпретации условий зависит от значения флага $prederror$. Если $prederror = true$, выдаётся ошибка, если значения какого-либо из выражений $cond_1, ..., cond_n$ отличается от true или false. Если $prederror = false$ и значения какого-либо из выражений $cond_1, ..., cond_n$ отличается от true или false, результат вычисления if - условное выражение..

5.1.2 Операторы цикла

Для выполнения итераций используется оператор do. Могут использоваться три варианта его вызова, отличающиеся условием окончания цикла:

```
for variable: initial_value step increment thru limit do body
for variable: initial_value step increment while condition do body
for variable: initial_value step increment unless condition do body
```

Здесь variable - переменная цикла; initia_value - начальное значение; increment - шаг (по умолчанию равен 1); limit - конечное значение переменной цикла; body - операторы тела цикла. Ключевые слова thru, while, unless указывают на способ завершения цикла: по достижении переменной цикла значения limit; пока выполняется условие condition; пока не будет достигнуто условие condition.

initial_value, increment, limit, и body могут быть произвольными выражениями. Контрольная переменная по завершении цикла предполагается положительной (при этом начальное значение может быть и отрицательным). Выражения limit, increment, условия завершения (condition) 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.

Контрольная переменная цикла - локальная внутри цикла, поэтому её из менение в цикле не влияет на контекст (даже при наличии вне цикла переменной с тем же именем).

Примеры:

```
(%i1) for a:-3 thru 26 step 7 do display(a)$
```

$$a = -3a = 4a = 11a = 18a = 25$$

```
(%i2) s: 0$ for i: 1 while i <= 10 do s: s+i;
```

(%o3) *done*

```
(%i4) s;
```

(%o4) 55

```
(%i5) series: 1$ term: exp (sin (x))$
(%i7) for p:1 unless p > 7 do (term: diff (term, x)/p, series: series + subst (x=0, term)*x^p)$
(%i8) series;
```

(%o8) $$\frac{x^7}{90} - \frac{x^6}{240} - \frac{x^5}{15} - \frac{x^4}{8} + \frac{x^2}{2} + x + 1$$

```
(%i9) for count: 2 next 3*count thru 20 do display (count)$
```

$$count = 2count = 6count = 18$$

Условия инициализации и завершения цикла можно опускать. Пример (цикл без явного указания переменной цикла)

```
(%i10) x:1000;
```

(%o10) 1000

```
(%i11) thru 20 do x: 0.5*(x + 5.0/x)$(%i12) x;
```

(%o12) 2.23606797749979

```
(%i12) float(sqrt(5));
```

(%o12) 2.23606797749979

За 20 итераций достигается точное значение $\sqrt{5}$.

Несколько более изощрённый пример - реализация метода Ньютона для уравнения с одной неизвестной (вычисляется та же величина - корень из пяти):

```
(%i1) newton (f, x):= ([y, df, dfx], df: diff (f ('x), 'x),
      do (y: ev(df), x: x - f(x)/y,
      if abs (f (x)) < 5e-6 then return (x)))

$(%i2) f(x):=x^2-5;
```

(%o2) $$f(x) := x^2 - 5$$

```
(%i3) float(newton(f,1000));
```

(%o3) $$2.236068027062195$$

Ещё одна форма опратора цикла характеризуется выбором значений переменной цикла из заданного списка. Синтаксис вызова: for variable in list end_tests do body

Проверка условия завершения end_tests до исчерпания списка list может отсутствовать. Пример:

```
(%i1) a:[];
```

(%o1) $$[]$$

```
(%i2) for f in [1,4,9,16] do a:cons(sqrt(f),a)$(%i3) a;
```

(%o3) $$[4, 3, 2, 1]$$

5.1.3 Блоки

Как в условных выражениях, так и в циклах вместо простых операторов можно писать составные операторы, т.е. блоки. Стандартный блок имеет вид: block([r,s,t],r:1,s:r+1,t:s+1,x:t,t*t); 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. Упрощенный блок имеет вид: (x:1,x:x+2,a:x); 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 и оператор "return". Оператор "return"прекращает выполнение текущего блока и возвращает в качестве значения блока свой аргумент block([],x:2,x:x*x, return(x), x:x*x);

В отсутствие оператора перехода на метку операторы в блоке выполняются последовательно. (В данном случае слово "метка"означает отнюдь не метку типа "%i5"или "%o7"). Оператор "go"выполняет переход на метку, располо- женную в этом же блоке: block([a],a:1,metka, a:a+1, if a=1001 then return(-a), go(metka)); -1001 В этом блоке реализован цикл, который завершается по достижении "перемен- ной цикла"значения 1001. Меткой может быть произвольный идентификатор. Следует иметь в виду, что цикл сам по себе является блоком, так что (в отли- чие от языка "C") 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 вну- три блока, содержит оператор "return то при исполнении оператора "re- turn"произойдет выход из цикла, но не выход из блока:

```
(%i1) block([],x:for i:1 thru 15 do if i=2 then return(555),display(x),777);
```

(%o1) $$x = 555777$$

```
(%i2) block([],x:for i:1 thru 15 do if i=52 then return(555),display(x),777);
```

(%o2) $$x = done777$$

Если необходимо выйти из нескольких вложенных блоков сразу (или несколь- ких блоков и циклов сразу) и при этом возвратить некоторое значение, то следует применять блок "catch"

```
(%i3) catch( block([],a:1,a:a+1, throw(a),a:a+7),a:a+9 );
```

(%o3) 2

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

(%o4) 2

```
(%i5) catch(block([],for i:1 thru 15 do if i=2 then throw(555)),777);
```

(%o5) 555

В данном блоке выполнение цикла завершается, как только значение i достигает 2. Возвращаемое блоком catch значение равно 555.

```
(%i6) catch(block([],for i:1 thru 15 do if i=52 then throw(555)),777);
```

(%o6) 777

В данном блоке выполнение 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".

Наконец, блок "errcatch"позволяет перехватывать некоторые (к сожалению, не все!) из ошибок, которые в нормальной ситуации привели бы к завершению счета. Пример:

```
(%i1) errcatch(a:1, b:0, log(a/b), c:7);
```

(%o1) $Divisionby0[]$

```
(%i2) c;
```

(%o2) c

Выполнение последовательности операций прерывается на первой операции, приводящей к ошибке. Остальные выражения блока не выполняются (значение с остаётся неопределённым). Сообщение об возникшей ошибке может быть выведено функцией errormsg().

5.1.4 Функции

Наряду с простейшим способом задания функции, Maxima допускает создание функции в виде последовательности операторов:

$$f(x) := (expr1, expr2, ..., exprn);$$

Значение, возвращаемое функцией - значение последнего выражения $exprn$.

Чтобы использлвать оператор *return* и изменять возвращаемое значение в зависимости от логики работы функции, следует использовать конструкцию *block*, например:

$$f(x) = block([], expr1, ..., if(a > 10)thenreturn(a), ..., exprn).$$

При $a > 10$ выполняется оператор *return* и функция возвращает значение a, в противном случае - значение выражения *exprn*. ции

Формальные параметры функции или блока - локальные, и являются видимыми только внутри них. Кроме того, при задании функции можно объявить локальные переменные (в квадратных скобках в начале объявления функции или бока). Пример:

$$block([a : a], expr1, ...a : a + 3, ..., exprn)$$

В данном случае при объявлении блока в локальной переменной a сохраняется значение глобальной переменной a, определённой извне блока. Пример:

```
(%i1) f(x):=([a:a],if a>0 then 1 else (if a<0 then -1 else 0));
```

(%o1) $$f(x) := ([a : a], if\ a > 0\ then\ 1\ else\ if\ a < 0\ then\ -1\ else\ 0)$$

```
(%i2) a:1;
```

(%o2) $$1$$

```
(%i3) f(0);
```

(%o3) $$1$$

```
(%i4) a:-4;
```

(%o4) $$-4$$

```
(%i5) f(0);
```

(%o5) $$-1$$

```
(%i6) a:0;
```

(%o6) $$0$$

```
(%i7) f(0);
```

(%o7) $$0$$

В данном примере значение переменной a задаётся вне тела функции, но результат, возвращаемый ею, зависит от значения a.

Начальные значения локальных переменных функции могут задаваться двумя способами:

- Задание функции $f(x) := (expr1, ..., exprn)$;, вызов функции $f(1)$; - начальное значение локальной переменной x равно 1.
- Задание блока $block([x : 1], expr1, ..., exprn)$, при этом начальное значение локальной переменной x также равно 1.

Наряду с именованными функциями, Maxima позволяет использовать и безымянные функции (лямбда-функции). Синтаксис использования лямбда-выражений (правда, при использовании с лямбда-выражениями всё-таки ассоциируется имя - см. пример):

$$f1 : lambda([x_1, ..., x_m], expr_1, ..., expr_n)$$
$$f2 : lambda([[L]], expr_1, ..., expr_n)$$
$$f3 : lambda([x_1, ..., x_m, [L]], expr_1, ..., expr_n)$$

Пример:

```
(%i1) f: lambda ([x], x^2);
```

(%o1) $lambda\left([x], x^2\right)$

```
(%i2) f(a);
```

(%o2) a^2

Более сложный пример (лямбда-выражения могут использоваться в контексте, когда ожидается имя функции):

```
(%i3) lambda ([x], x^2) (a);
```

(%o3) a^2

```
(%i4) apply (lambda ([x], x^2), [a]);
```

(%o4) a^2

```
(%i5) map (lambda ([x], x^2), [a, b, c, d, e]);
```

(%o5) $[a^2, b^2, c^2, d^2, e^2]$

Аргументы лямбда-выражений - локальные переменные. Другие переменные при вычислении лямбда-выражений рассматриваются как глобальные. Исключения отмечаются специальным символом - прямыми кавычками (см. лямбда-функцию g2 в примере).

```
(%i6) a: %pi$(%i7) b: %e$(%i8) g: lambda ([a], a*b);
```

(%o8) $lambda\left([a], a\, b\right)$

```
(%i9) b: %gamma$(%i10) g(1/2);
```

(%o10) $\frac{\gamma}{2}$

```
(%i11) g2: lambda ([a], a*''b);
```

(%o11) $lambda\left([a], a\,\gamma\right)$

```
(%i12) b: %e$(%i13) g2(1/2);
```

(%o13) $\frac{\gamma}{2}$

Лямбда-функции могут быть вложенными. При этом локальные переменные внешнего выражения доступны как глобальные для внутреннего (одинаковые имена переменных маскируются). Пример:

```
(%i1) h: lambda ([a, b], h2: lambda ([a], a*b), h2(1/2));
```

(%o1) $lambda\left([a, b], h2 : lambda\left([a], a\,b\right), h2\left(\frac{1}{2}\right)\right)$

```
(%i2) h(%pi, %gamma);
```

(%o2) $\frac{\gamma}{2}$

Подобно обычным функциям, лямбда-функции могут иметь список параметров переменной длины. Пример:

```
(%i1) f : lambda ([aa, bb, [cc]], aa * cc + bb);
```

(%o1) $lambda\left([aa, bb, [cc]], aa\,cc + bb\right)$

```
(%i2) f(3,2,a,b,c);
```

(%o2) $[3\,a + 2, 3\,b + 2, 3\,c + 2]$

Список $[cc]$ при вызове лямбда-функции f включает элемента: $[a, b, c]$. Формула для расчёта f применяется к каждому элементу списка.

Локальные переменные могут быть объявлены и посредством функции local (переменные $v_1, v_2, ..., v_n$ объявляются локальными вызовом $local(v_1, v_2, ..., v_n)$ независимо от контекста).

5.1.5 Транслятор и компилятор в MAXIM'е

Определив ту или иную функцию, можно заметно ускорить ее выполнение, если ее оттранслировать или откомпилировать. Это происходит потому, что если Вы не оттранслировали и не откомпилировали определенную Вами функцию, то при каждом очередном ее вызове MAXIMA каждый раз заново выполняет те действия, которые входят в определение функции, т.е. фактически разбирает соответствующее выражение на уровне синтаксиса MAXIM'ы.

5.1.5.1 Функция translate

Функция translate транслирует функцию MAXIM'ы на язык LISP. Например, выражение $f() := 1 + \ + \ 2 + "3 + "4 + \ 5 + "6 + "7$

$translate(f)$;

После этого функция (как правило) начинает считаться быстрее.

5.1.5.2 Функция compile

Функция compile сначала транслирует функцию MAXIM'ы на язык LISP, а затем компилирует эту функцию LISP'а до двоичных кодов и загружает их в память.

Пример:

```
(\%i9) compile(f);
Compiling /tmp/gazonk_1636_0.lsp.
End of Pass 1.
End of Pass 2.
OPTIMIZE levels:    Safety=2,
Space=3, Speed=3
Finished compiling /tmp/gazonk_1636_0.lsp.
(\%o92) [f]
```

После этого функция (как правило) начинает считаться еще быстрее, чем после трансляции. Следует иметь в виду, что как при трансляции, так и при компиляции MAXIMA старается оптимизировать функцию по скорости, не заботясь об аккуратности. Поэтому при работе с большими по объему функциями могут возникнуть чудеса. В этом случае следует отказаться от трансляции или компиляции, либо переписать функцию. Выигрыш во времени существенным образом зависит от типа машины, от вида функции, от того, декларирован ли тип функции и ее аргумента при определении функции, от типа аргумента, с которым вызывается функция. Если предполагается использовать функцию только для работы с действительными числами (например для вычисления определенного интеграла с помощью функции "romberg"или поиска корня с помощью функций "find_root"и "newton"), то обязательно следует декларировать тип аргумента и самой функции как "float". Это во много раз усилит эффект от трансляции или компиляции. Для того, чтобы дать общее представление о влиянии трансляции и компиляции на скорость счета разных типов функций и разных типов аргумента, приведем табличку с временами исполнения функций на одной конкретной машине. Были определены четыре разные функции, вычисляющие одно и то же выражение

$fl() := 1 + \ + "2 + "3 + "4 + "5 + "6 + "7 + "8 + "9$ $f2(x) := block([s], s : l, fori : lthru9dos : s + x\ i, s)$
$f3(x) := 1([], mode_declare([function(f), x], float), 1 + x + x"2 + x"3 + x"4 + x\ 5 + x"6 + x"7 + x"8 + x"9)$
$f4(x) := 1([s], mode_declare([function(f), x, s], float), s : l, fori : lthru9dos : s + x\ i, s)$

Форму записи двух последних функций ("f3"и "f4") следует воспринимать аксиоматически. Далее каждая из этих функций вызывалась с аналитическим аргументом "s"(кроме "f3"и "f4 для которых аналитический аргумент невозможен), целочисленным аргументом "7"и вещественным аргументом "0.3". После этого все четыре функции были оттранслированы ([t]) и эксперимент был повторен. Наконец, все четыре функции были откомпилированы ([c]) и эксперимент снова был повторен. Ниже приведены соответствующие (условные) времена исполнения функций.

Во-первых, хорошо заметно, насколько трудоемкой оказывается процедура упрощения. Функция "f1"задана в виде явной формулы, и ее вычисление сводится к однократной подстановке, в то время как функция "f2"требует упрощения на каждом обороте цикла, что фатально сказывается на скорости ее вычисления. Зато и эффект от трансляции или компиляции для неявных функций типа "f2"оказывается гораздо заметнее. Во-вторых, заметно, что экономия времени для "более простых"числовых аргументов (по сравнению с символьными аргументами) оказывается не такой уж радикальной, хотя и довольно существенной. Это связано с тем, что числа все равно рассматриваются как элемент аналитического выражения, хотя упрощение аналитического выражения, составленного исключительно из чисел, идет быстрее. В-третьих, заметно, что для функций, которые декларированы как вещественные функции от вещественных аргументов, вычисление от

целочисленного аргумента идет медленнее, чем от вещественного аргумента. В-четвертых, очень хорошо видно, что если Вы не выполняете компиляцию для вещественной функции, то Вы рискуете замедлить ее вычисление более чем в 6 раз.

5.2 Ввод-вывод в пакете Maxima

В этом разделе рассматриваются конструкции Maxima, позволяющие осуществить обмен данными между Maxima и другими приложениями.

5.2.1 Ввод-вывод данных в консоли

Основная функция для считывания вводимых пользователем функций: $read(expr_1, ..., expr_n)$. Вводимые выражения $expr_1, expr_2, ...$ при вводе интерпретируются. Поля ввода разделяются точками с запятой или знаком $. Аргументы функции read могут включать подсказку. Пример:

```
(%i1) a:42$(%i2) a:read("Значение a = ",a," введите новую величину");
```

$$a = 42$$

```
(p+q)^3;
```

$$(\%o2) \qquad (q+p)^3$$

```
(%i3) display(a);
```

$$(\%o3) \qquad a = (q+p)^3 done$$

Аналогичная функция readonly осуществляет только ввод данных (без их интерпретации). Пример (сравнение использования функций read и readonly):

```
(%i1) a:7$(%i2) readonly("Введите выражение:");
```

:

```
2^a;
```

$$(\%o2) \qquad 2^a$$

```
(%i3) read("Введите выражение:");
```

:

```
2^a;
```

$$(\%o3) \qquad 128$$

Вывод на экран осуществляется функцией display. Синтаксис её вызова: $display(expr_1, expr_2, ...)$.

Выражения из списка аргументов выводятся слева направо (сначала само выражение, а затем после знака раверства - его значение).

Аналогичня функция disp (синтаксис вызова: $disp(expr_1, expr_2, ...)$) выводит на экран только значение выражения после его интерпретации

Функция grind осуществляет вывод в консоль Maxima аналогично disp, но в форме, удобной для ввода с клавиатуры.

```
(%i1) a:1$ b:2$ c:3$(%i4) display(a,b,c);
```

(%o4) $a = 1b = 2c = 3done$

```
(%i5) disp(a,b,c);
```

(%o5) $123done$

```
(%i6) grind(a);
```

(%o6) $1done$

Управление консольным вводом/выводом осуществляется посредством установки флагов display2d, display_format_internal и т.п.

Вывод на экран длинных выражений по частям (одна часть над другой) осуществляется функцией dispterms(синтаксис displayterm(expr)).

Кроме того, для вывода результатов вычислений используется функция print. Синтаксис вызова: $print(expr_1, ..., expr_n)$. Выражения $expr_1, ..., expr_n$ интерпретируются и выводятся последовательно в строчку (в отличие от вывода, генерируемого функцией display). Функция print возвращает значение последнего интерпретированного выражения. Пример:

```
(%i1) a:1$ b:2$ c:(a^2+b^2)$(%i4) rez:print("Пример:",a,b,c);
```

(%o4) $: 1255$

```
(%i5) rez;
```

(%o5) 5

```
(%i6) display("Пример:",a,b,c);
```

(%o6) $:= \ : a = 1b = 2c = 5done$

5.2.2 Файловые операции ввода-вывода

5.2.2.1 Ввод-вывод текстовых данных

Сохранение текущего состояния рабочей области Maxima осуществляется при помощи функции save. Эта функция позволяет сохранить в файле отдельные объекты с указанными именами. Варианты вызова save:

$$save(filename, name_1, name_2, name_3, ...)$$

- сохраняет текущие значения переменных $name_1, name_2, name_3, \ldots$ в файле filename. Аргументы должны быть именами переменных, функций или других объектов. Если имя не ассоциируется с какой-либо величиной в памяти, оно игнорируется. Функция save возвращает имя файла, в который сохранены заданные объекты.

$$save(filename, values, functions, labels, ...)$$

- сохраняет все значения переменных, функций, меток и т.п.

$$save(filename, [m, n])$$

- сохраняет все значения меток ввода/вывода в промежутке от m до n (m, n - целые литералы).

$$save(filename, name_1 = expr_1, ...)$$

- позволяет сохранить объекты Maxima с заменой имени $expr_1$ на имя $name_1$.

$$save(filename, all)$$

- сохраняет все объекты, имеющиеся в памяти.

Глобальный флаг file_output_append управляет режимом записи. Еслии file_output_append=true, результаты вывода save добавляются в конец файла результатов. Иначе файл результата переписывается. Вне зависимости от file_output_append, если файл результатов не существует, то он создаётся.

Данные, сохранённные функцией save, могут быть снова загружены функцией load (см. ниже).

Варианты записи при помощи save могут совмещаться друг с другом (пример - команда save(filename, aa, bb, cc=42, functions, [11, 17]).

Загрузка предварительно сохранённого функцией save файла осушествляется функцией load(filename). Аналогичный синтаксис и у функции stringout, которая предназначена для вывода в файл выражений Maxima в формате, пригодном для последующего считывания Maxima. Синтаксис вызова stringout:

- stringout (filename, expr_1, expr_2, expr_3, ...)
- stringout (filename, [m, n])
- stringout (filename, input)
- stringout (filename, functions)
- stringout (filename, values)

Функция load (filename) вычисляет выражения в файле filename, создавая таким образом переменные, функции, и другие объекты Maxima. Если объект с некоторым именем уже присутствует в Maxima, при выполнении load он будет замещён считываемым. Пи поиске загружаемого файла Чтобы найти загружаемый файл, функция load использует переменные file_search , file_search_maxima и file_search_lisp как справочники поиска. Если загружаемый файл не найден, печатается сообщение об ошибке.

Загрузка работает одинаково хорошо для кода на Lisp и кода на макроязыке Maxima. Файлы, созданные функциями save, translate_file, compile_file , экономят, translate_file, и compile_file, которые содержат код на Lisp, или созданные при помощи функции stringout, который содержат код Maxima, может с равным успехом могут быть обработаны функцией load. Load использует функцию loadfile, чтобы загрузить файлы Lisp и batchload, чтобы загрузить файлы Maxima.

Load не распознаёт конструкции :lisp в файлах, содержащих код на Maxima, а также глобальные переменные _, __, %, и %th, пока не будут созданы соответствующие объекты в памяти.

Функция loadfile (filename) предназначена для загрузки файлов, содержащих код на Lisp, созданные функциями save, translate_file, compile_file. Для задач конечного пользователя удобнее функция load.

Протокол сессии Maxima может зарисываться при помощи функции writefile (он записывается в формате вывода на консоль). Для тех же целей используется функция appendfile (запись в конец существующего файла). Завершение записи и закрытие файла протокола осуществляется функцией closefile. Синтаксис вызова: writefile (filename), closefile (filename).

5.2.2.2 Ввод-вывод командных файлов

Основная функция, предназначенная дял ввода и интерпретации командных файлов - функция batch (filename) Функция batch читает выражения Maxima из файла filename и выполняет их. Функция batch отыскивает filename в списке file_search_maxima. имя файла filename включает последовательность выражений Maxima, каждое из которых должно оканчиваться ; или $. Специальная переменная % и функция %th обращаются к предыдущим результатам в пределах файла. Файл может включать конструкции:lisp. Пробелы, табуляции, сиволы конца строки в файле игнорирубтся. Подходящий входной файл может быть создан редактором текста или функцией stringout. Функция batch считывает каждое выражение из файла filename, показывает ввод в консоли, вычисляет соответствующие выражения и показывает вывод также в консоли. Метки ввода назначаются входным выражениям, метки вывода – результатам вычислений, функция batch интерпретирует каждое входное выражение, пока не будет достигнут конец файла. Если предполагается реакция пользователя (ввод с клавиатуры), выполнение batch приостанавливается до завершения ввода. Для остановки віполнения batch-файла используется Ctrl-C.

Функция batchload(filename) считывает и интерпретирует выражения из командного файла, но не выводит на консоль входных и выходных выражений. Метки ввода и вывода выражениям, встречающимся в командном файле, также не назначаются. Специальная переменная % и функция %th обращаются к предыдущим диалоговым меткам, не имея результатов в пределах файла. Кроме того, файл filename не может включать конструкции:lisp.

5.3 Встроенные численные методы

5.3.1 Численные методы решения уравнений

5.3.1.1 Решение уравнений с одним неизвестным

Для решения уравнения с одним неизвестным в пакете Maxima предусмотрена функция $find_root$. Синтаксис вызова:

$$find_root(expr, x, a, b)$$

$$find_root(f, a, b)$$

Поиск корня функции f или выражения expr относительно переменной x осуществляется в пределах $a \leqslant x \leqslant b$.

Для поиска корней используется метод деления пополам или, если исследуемая функция достаточно гладкая, метод линейной интерполяции.

5.3.2 Решение уравнений методом Ньютона: пакет newton1

Основная функция пакета newton1 предназначена для решения уравнений методом Ньютона. Синтаксис вызова:

```
newton (expr, x, x_0, eps)
```

Данная функция возвращает приближенное решение уравнения expr = 0 методом Ньютона, рассматривая expr как функцию одной переменной, x. Поиск начинается с x = x_0 и производится, пока не будет достигнуто условие abs (expr)< eps. Функция newton допускает наличие неопределенных переменных в выражении expr, при этом выполнение условия abs (expr)< eps, оценивается как истинное или ложное. Таким образом, нет необходимости оценивать expr только как число. Для использования пакета необходимо загрузить его командой load(newton1).

Примеры использования функции newton:

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

```
(%o3)                 1.2104963335033528E-4
(%i4) assume (a > 0);
(%o4)                         [a > 0]
(%i5) newton (x^2 - a^2, x, a/2, a^2/100);
(%o5)                  1.00030487804878 a
(%i6) ev (x^2 - a^2, x = %);
                                          2
(%o6)               6.098490481853958E-4 a

(%i1) load (newton1);
(%o1) /usr/share/maxima/5.10.0cvs/share/numeric/newton1.mac
(%i2) newton (cos (u), u, 1, 1/100);
(%o2)                   1.570675277161251
(%i3) ev (cos (u), u = %);
(%o3)                 1.2104963335033528E-4
(%i4) assume (a > 0);
(%o4)                         [a > 0]
(%i5) newton (x^2 - a^2, x, a/2, a^2/100);
(%o5)                  1.00030487804878 a
(%i6) ev (x^2 - a^2, x = %);
                                          2
(%o6)               6.098490481853958E-4 a
```

5.3.2.1 Решение уравнений с несколькими неизвестными: пакет mnewton

Мощная функция для решения систем нелинейных уравнений методом Ньютона входит в состав пакета ***mnewton***. Перед использованием пакет необходимо загрузить:

```
(%i1) load("mnewton");
```

(%o1) */usr/share/maxima/5.13.0/share/contrib/mnewton.mac*

После загрузки пакета mnewton становятся доступными основная функция - mnewton и ряд дополнительных переменных для управления ею: newtonepsilon (точность поиска, величина по умодчанию $10.0^{(} - fpprec/2)$), newtonmaxiter (максимальное число итераций, величина по умолчанию 50). Синтаксис вызова функции mnewton: mnewton (FuncList,VarList,GuessList) (FuncList - список функций, образующих решаемую систему уравнений, VarList - список имен переменной, и GuessList - список начальных приближений). Решение возвращается в том же самом формате, который использует функция solve(). Если решение не найдено, возвращается пустой список.

Пример использования функции mnewton:

```
(%i1) load("mnewton")$(%i2) mnewton([x1+3*log(x1)-x2^2, 2*x1^2-x1*x2-5*x1+1], [x1, x2], [5, 5]);
```

(%o2) $[[x1 = 3.756834008012769, x2 = 2.779849592817898]]$

```
(%i3) mnewton([2*a^a-5],[a],[1]);
```

(%o3) $[[a = 1.70927556786144]]$

Как видно из второго примера, функция mnewton может использоваться и для решения единичных уравнений.

5.3.3 Интерполяция

Для выполнения интегполяции функций, заданных таблично, в составе MAXIMA предусмотрен пакет расширения interpol, позволяющий выполнять линейную или полиномиальную интерполяцию. Пакет включает служебную функцию charfun2 (x, a, b),которая возвращает true, если число x принадлежит интервалу [a, b), и false в противном случае.

5.3.3.1 Линейная интерполяция

Линейная интерполяция выполняется функцией linearinterpol (синтаксис вызова: linearinterpol (points) или linearinterpol (points, option)). Аргумент points должен быть представлен в одной из следующих форм:

- матрица с двумя столбцами, например p:matrix([2,4],[5,6],[9,3]), при этом первое значение пары или первый столбец матрицы - это значения независимой переменной,
- список пар значений, например p: [[2,4],[5,6],[9,3]],
- список чисел, которые рассматриваются как ординаты интерполируемой функции, например p: [4,6,3], в этом случае абсциссы назначаются автоматически (принимают значения 1, 2, 3 и т.д.).

В качестве опции указывается имя независимой переменной, относительно которой строится интерполяционная функция.

Примеры выполнения линейной интерполяции:

```
(%i1) load("interpol");
```

(%o1) $/usr/share/maxima/5.13.0/share/numeric/interpol.mac$

```
(%i2) p: matrix([7,2],[8,2],[1,5],[3,2],[6,7])$(%i3) linearinterpol(p);
```

(%o3)
$$\left(\frac{13}{2}-\frac{3\,x}{2}\right)\,charfun2\left(x,-\infty,3\right)+2\,charfun2\left(x,7,\infty\right)+\left(37-5\,x\right)\,charfun2\left(x,6,7\right)+\left(\frac{5\,x}{3}-3\right)\,charfun2\left(x,3,6\right)$$

```
(%i4) f(x):=''%;
```

(%o4)
$$f\left(x\right):=\left(\frac{13}{2}-\frac{3\,x}{2}\right)\,charfun2\left(x,-\infty,3\right)+2\,charfun2\left(x,7,\infty\right)+\left(37-5\,x\right)\,charfun2\left(x,6,7\right)+\left(\frac{5\,x}{3}-3\right)\,charfun2\left(x,$$

```
(%i5) map(f,[7.3,25/7,%pi]);
```

(%o5)
$$[2,\frac{62}{21},\frac{5\,\pi}{3}-3]$$

5.3.3.2 Интерполяция полиномами Лагранжа

Интерполяция полиномами Лагранжа выполняется припомощи функции lagrange (синтаксис вызова lagrange (points) или lagrange (points, option)). Смысл параметров points и options аналогичен указанному выше. Пример использования интерпоояляции полиномами Лагранжа:

```
(%i1) load("interpol")$(%i2) p:[[7,2],[8,2],[1,5],[3,2],[6,7]]$(%i3) lagrange(p);
```

(%o3) $$\frac{73\,x^4}{420}-\frac{701\,x^3}{210}+\frac{8957\,x^2}{420}-\frac{5288\,x}{105}+\frac{186}{5}$$

```
(%i4) f(x):=''%;
```

(%o4) $$f\left(x\right):=\frac{73\,x^4}{420}-\frac{701\,x^3}{210}+\frac{8957\,x^2}{420}-\frac{5288\,x}{105}+\frac{186}{5}$$

```
(%i5) map(f,[2.3,5/7,%pi]);
```

(%o5) $$[-1.567534999999992,\frac{919062}{84035},\frac{73\,\pi^4}{420}-\frac{701\,\pi^3}{210}+\frac{8957\,\pi^2}{420}-\frac{5288\,\pi}{105}+\frac{186}{5}]$$

```
(%i6) %,numer;
```

(%o6) $$[-1.567534999999992,10.9366573451538,2.893196551256921]$$

5.3.3.3 Интерполяция сплайнами

Интерполяция кубическими сплайнами выполняется припомощи функции cspline (синтаксис вызова cspline(points) или cspline(points, option)). Смысл параметров points и options аналогичен указанному выше. Пример использования интерпооляции кубическими сплайнами:

```
(%i1) load("interpol")$(%i2) p:[[7,2],[8,2],[1,5],[3,2],[6,7]]$(%i3) cspline(p);
```

(%o3)
$$\left(\frac{1159\,x^3}{3288}-\frac{1159\,x^2}{1096}-\frac{6091\,x}{3288}+\frac{8283}{1096}\right)\,charfun2\left(x,-\infty,3\right)+\left(-\frac{2587\,x^3}{1644}+\frac{5174\,x^2}{137}-\frac{494117\,x}{1644}+\frac{108928}{137}\right)\,charfun2\left(x\right.$$

```
(%i4) f(x):=''%;
```

(%o4)
$$f\left(x\right):=\left(\frac{1159\,x^3}{3288}-\frac{1159\,x^2}{1096}-\frac{6091\,x}{3288}+\frac{8283}{1096}\right)\,charfun2\left(x,-\infty,3\right)+\left(-\frac{2587\,x^3}{1644}+\frac{5174\,x^2}{137}-\frac{494117\,x}{1644}+\frac{108928}{137}\right)\,ch$$

```
(%i5) map(f,[2.3,5/7,%pi]);
```

(%o5) $$[1.991460766423356,\frac{273638}{46991},-\frac{3287\,\pi^3}{4932}+\frac{2223\,\pi^2}{274}-\frac{48275\,\pi}{1644}+\frac{9609}{274}]$$

```
(%i6) %,numer;
```

(%o6) $$[1.991460766423356,5.823200187269903,2.227405312429507]$$

5.3.4 Оптимизация с использованием пакета lbfgs

Основная функция пакета (функция lbfgs (FOM, X, X0, epsilon, iprint)) позволяет найти приближенное решение задачи минимизации без ограничений целевой функции, определяемой выражением FOM, по списку переменных X с начальным приближением X0. Критерий окончания поиска определяется градиентом нормы целевой функции (градиент нормы FOM $<$ epsilon max(1, norm X)). Данная функция использует квазиньютоновский алгоритм с ограниченной памятью (алгоритм BFGS). Этот метод называют методом с ограниченным использованием памяти, потому что вместо полного обращения матрицы Гессе (гессиана) используется приближение с низким рангом. Каждая итерация алгоритма - линейный (одномерный) поиск, то есть, поиск вдоль луча в пространстве переменных X с направлением поиска, вычисленным на базе приближенного обращения матрицы Гессе. В результате успешного линейного поиска значение целевой функции (FOM) уменьшается. Обычно (но не всегда) норма градиента FOM также уменьшается.

Параметр функции iprint позволяет контролировать вывод сообщений о прогрессе поиска. Величина iprint[1] управляет частотой вывода (iprint[1] $<$ 0 - сообщения не выводятся; iprint[1] $=$ 0 - сообщения на первых и последних итерациях; iprint[1] $>$ 0 - вывод сообщений на каждой iprint[1] итерации). Величина iprint[2] управляет объёмом выводимой информации (если iprint[2] $=$ 0, выводится счётчик итераций, число вычислений целевой функции, её величину, величину нормы градиента FOM и длины шага). Увеличение iprint[2] (целая переменная, принимающая значения 0,1,2,3) влечёт за собой увеличение количества выводимой информации. Обозначения колонок выводимой информации:

- I - число итераций, которое увеличивается после каждого линейного поиска;
- NFN - количество вычислений целевой функции;
- FUNC - значение целевой функции в конце линейного поиска;
- GNORM - норма градиента целевой функции в конце очередного линейного поиска;
- STEPLENGTH - длина шага (внутренний параметр алгоритма поиска).

Функция lbfgs реализована разработчиками на Lisp путём перекодирования классического алгоритма, первоначально написанного на Фортране, поэтому сохранила некоторые архаичные черты. Однако используемый алгоритм обладает высокой надёжностью и хорошим быстродействием.

Рассмотрим примеры использования lbfgs.

Простейший пример - минимизация функции одной переменной. Необходимо найти локальный минимум функции $f(x) = x^3 + 3x^2 - 2x + 1$. Результаты расчётов:

```
(%i1) load (lbfgs);
```

(%o1) $/usr/share/maxima/5.13.0/share/lbfgs/lbfgs.mac$

```
(%i2) FOM:x^3+3*x^2-2*x+1;
```

(%o2) $$x^3 + 3\,x^2 - 2\,x + 1$$

```
(%i3) lbfgs(FOM,[x],[1.1], 1e-4, [1, 0]);
```

(%o3)
$**********N = 1NUMBEROFCORRECTIONS = 25INITIALVALU$

Рассмотрим результаты минимизации функции нескольких переменных при помощи lbfgs:

```
(%i1) load (lbfgs);
```

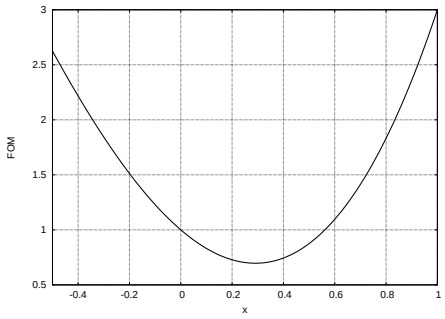


Рис. 5.1. График исследуемой функции в окрестности минимума

(%o1) $/usr/share/maxima/5.13.0/share/lbfgs/lbfgs.mac$

```
(%i2) FOM:2*x*y+8*y*z+12*x*z+1e6/(x*y*z);
```

(%o2) $$8\,y\,z + 12\,x\,z + \frac{1000000.0}{x\,y\,z} + 2\,x\,y$$

```
(%i3) lbfgs(FOM,[x,y,z],[1,1,1],1e-4,[-1,0]);
```

(%o3) $[x = 13.47613086835734, y = 20.21398622934409, z = 3.369022781547174]$

5.3.4.1 Оптимизация с ограничениями методом неопределённых множителей Лагранжа

Для решения задач минимизации с ограничениями в составе Maxima предусмотрен пакет augmented_lagrangian_ реализующий метод неопределённых множителей Лагранжа. Синтаксис вызова функции augmented_lagrangian_me
augmented_lagrangian_method (FOM, xx, C, yy);
augmented_lagrangian_method (FOM, xx, C, yy, optional_args).

Рассматриваемая функция возвращает приближенное решение задачи минимизации функции нескольких переменных с ограничениями-равенствами. Целевая функция задаётся выражением FOM, варьируемые переменные - списком xx, их начальные значения - списком yy, ограничения - списком C (предполагается, что ограничения приравниваются к 0). Переменные optional_args задаются в форме символ=значение. Распознаются следующие символы: niter - число итераций метода неопределённых множитетеё Лагранжа;
lbfgs_tolerance - точность поиска LBFGS;
iprint - тот же параметр, что и для lbfgs;
%lambda - начальное значение неопределённого множителя для метода Лагранжа.

Для использования функции augmented_lagrangian_method необходимо загрузить её командой load(augmented_lagrangian).

Данная реализация метода неопределённых множителей Лагранжа базируется на использовании квазиньютоновского метода LBFGS.

5.3.5 Численное интегрирование: пакет romberg

Для вычисления определённых интегралов численными методами в Maxima есть простая в использовании и довольно мощная функция romberg (перед использованием её необходимо загрузить). Синтаксис вызова romberg:

```
  romberg (expr, x, a, b)
romberg (F, a, b)
```

Функция romberg вычисляет определённые интегралы методом Ромберга. В форме romberg(expr, x, a, b) возвращает оценку полного интеграла выражения expr по переменной x в пределах от a до b. Выражение expr должно возвращать действительное значение (число с плавающей запятой). В форме romberg(F, a, b) функция romberg возвращает оценку интеграла функции F (x) по переменной x в пределах от a до b (x представляет собой неназванный, единственный аргумент F; фактический аргумент может быть отличен от x). Функция F должна быть функцией Maxima или Lisp, которая возвращает значение с плавающей запятой.

Точностью вычислений при выполнении romberg управляют глобальные переменные rombergabs, и rombergtol. Функцияromberg заканчивается успешно, когда абсолютное различие между последовательными приближениями - меньше чем rombergabs, или относительное различие в последовательных приближениях - меньше чем rombergtol. Таким образом, когда rombergabs равна 0.0 (это значение по умолчанию), только величина относительной ошибки влияет на выполнение функции romberg.

Функция romberg уменьшает шаг интегрирования вдвое по меньшей мере rombergit раз, поэтому максимальное количество вычислений подинтегральной составляет $2^r ombergit$. Если критерий точности интегрирования, установленный rombergabs и rombergtol, не удовлетворен, romberg печатает сообщение об ошибке. Функция romberg всегда делает по крайней мере rombergmin итерации; это - эвристическое правило, предназначенное, чтобы предотвратить преждевременное завершение выполнения функции, когда подынтегральное выражение является колебательным.

Вычисление при помощи romberg многомерных интегралов возможно, но заложенный разработчиками способ оценки точности приводит к тому, что методы, разработанные специально для многомерных задач, могут привести к той же самой точности с существенно меньшим количеством оценок функции.

Рассмотрим примеры вычисления интегралов с использованием romberg:

```
(%i1) load (romberg);
```

(%o1) $/usr/share/maxima/5.13.0/share/numeric/romberg.lisp$

```
(%i2) g(x, y) := x*y / (x + y);
```

(%o2) $$g\left(x,y\right):=\frac{x\,y}{x+y}$$

```
(%i3) estimate : romberg (romberg (g(x, y), y, 0, x/2), x, 1, 3);
```

(%o3) 0.81930228643245

```
(%i4) assume (x > 0);
```

(%o4) $[x > 0]$

```
(%i5) integrate (integrate (g(x, y), y, 0, x/2), x, 1, 3);
```

(%o5) $$-9\,log\left(\frac{9}{2}\right)+9\,log\left(3\right)+\frac{2\,log\left(\frac{3}{2}\right)-1}{6}+\frac{9}{2}$$

```
(%i6) float(%);
```

(%o6) 0.81930239639591

Как видно из полученных результатов вычисления двойного интеграла, точное и приближённое решение совпадают до 7 знака включительно.

Глава 6

Обрамление Maxima

6.1 Графические интерфейсы Maxima

6.1.1 Графический интерфейс wxMaxima

Для удобства работы сразу обратимся к графическому интерфейсу wxMaxima, т. к. он является наиболее дружественным для начинающих пользователей системы.

Достоинствами wxMaxima являются:

- возможность графического вывода формул;
- упрощенный ввод наиболее часто используемых функций (через диалоговые окна);
- возможность включения графических иллюстраций непосредственно в текст рабочей книги (при использовании формата wxMaxima)

6.1.1.1 Рабочее окно wxMaxima

Рассмотрим рабочее окно программы (6.1). Сверху вниз располагаются: текстовое меню программы - доступ к основным функциям и настройкам программы. В текстовом меню wxMaxima находятся функции для решения большого количества типовых математических задач, разделенные по группам: уравнения, алгебра, анализ, упростить, графики, численные вычисления. Ввод команд через диалоговые окна упрощает работу с программой для новичков.

Например, пункт меню Анализ/Интегрировать позволяет вычислить определенный или неопределенный интеграл. После ввода необходимых параметров, в рабочем окне мы увидим команду и результат вычисления:

```
(%i1) integrate(3*x+5/x,x);
```

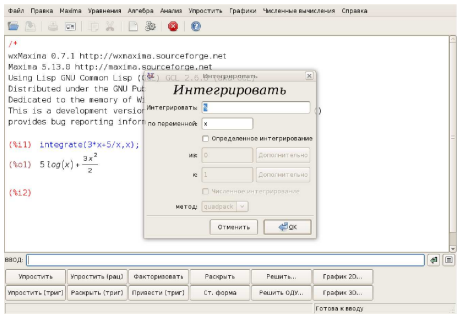


Рис. 6.1. Рабочее окно интерфейса wxMaxima

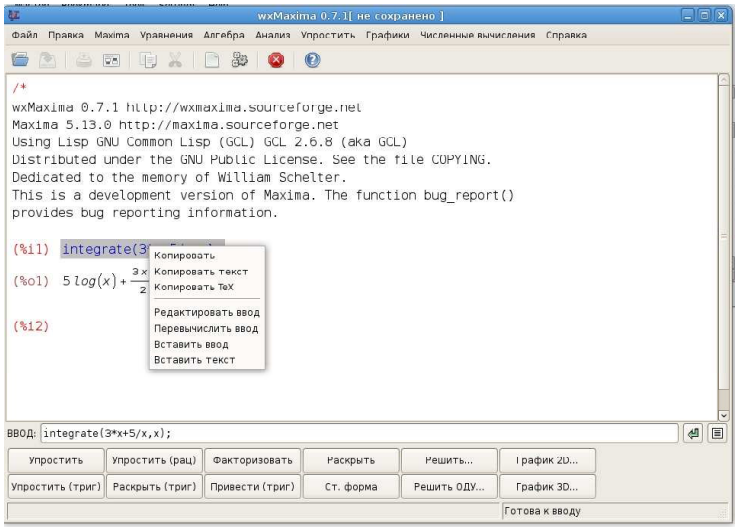


Рис. 6.2. Контекстное меню для поля ввода в рабочем окне wxMaxima

(%o1) $$5\,log\,(x)+\frac{3\,x^2}{2}$$

Основные возможности вычисления пределов представлены на вкладке wxMaxima.

Ниже располагается графическое меню - наиболее часто используемые функции для работы с файлами: открыть / сохранить / печать данных, а также функции правки - копировать / удалить / вставить текст и другие, затем расположено окно вывода результатов расчетов, поле ввода команд и кнопки доступа к часто используемым функциям расчетов (упрощение выражения, раскрытие скобок, построение графиков функциий и др.). При написании подпрограмм и длинных команд нажмите на кнопку I (многострочный ввод), которая располагается рядом с окном ввода команд.

При использовании интерфейса wxMaxima, Вы можете выделить в окне вывода результатов необходимую формулу и вызвав контекстное меню правой кнопкой мыши скопировать любую формулу в текстовом виде, в формате TEX или в виде графического изображения, для последующей вставки в какой-либо документ. Пример контекстного меню для поля ввода - на рис. 6.2, пример контекстного меню для поля вывода - на рис. 6.3.

Также в контекстном меню, при выборе результата вычисления, Вам будет предложен ряд операций с выбранным выражением (например, упрощение, раскрытие скобок, интегрирование, дифференцирование и др.).

6.1.1.2 Ввод простейших команд в wxMaxima

Все команды вводятся в поле ВВОД, разделителем команд является символ ; (точка с запятой) или символ $. После ввода команды необходимо нажать клавишу Enter для ее обработки и вывода результата. Если необходимо предотвратить вывод отклика команды, следует явно завершить её символом $. Современные версии wxMaxima автоматически завершают ввод, если это необходимо, символом ″; ″.

При работе с wxMaxima удобно использовать следующие клавиатурные комбинации (см. таблицу).

По умолчанию wxMaxima предполагает, что команда, вводимая при помощи кнопки, применяется к последнему выводу (т.е. аргкмент команды - %). Все кнопки или пункты меню в верхней или нижней части рабочего окна соответствуют той или иной команде Maxima.

Клавиатурная комбинация	**Команда**	**Примечание**
F1	Открыть браузер справки	
F2	Выбрать последний ввод	
F3	Перейти к окну вывода	
F4	Перейти в строку ввода	
F5	Скопировать выделение в поле ввода	Используется, когда выделено выражение в окне вывода
F6	Добавить новую текстовую группу	Позволяет ввести текстовый комментарий на поле вывола
F7	Ввести новую группу команд	Поволяет выбрать метку для группы команд
Enter	Редактировать текст	Применима, когда выбран редактируемый текст в окне ввода
Ctrl-Enter	Снова вычислить выражение	Применима, когда выбрана команда в окне ввода
Ctrl-Enter	Завершить редактирование	Применима, когда окно вывода в режиме редактирования
Esc	Завершить редактирование, но не посылать результаты Maxima	Применима, когда окно вывода в режиме редактирования
Up/Down	Перемещение по редактируемому тексту	Применима, когда окно вывода в режиме редактирования

Таблица 6.1. Клавиатурные комбинации для работы с wxMaxima

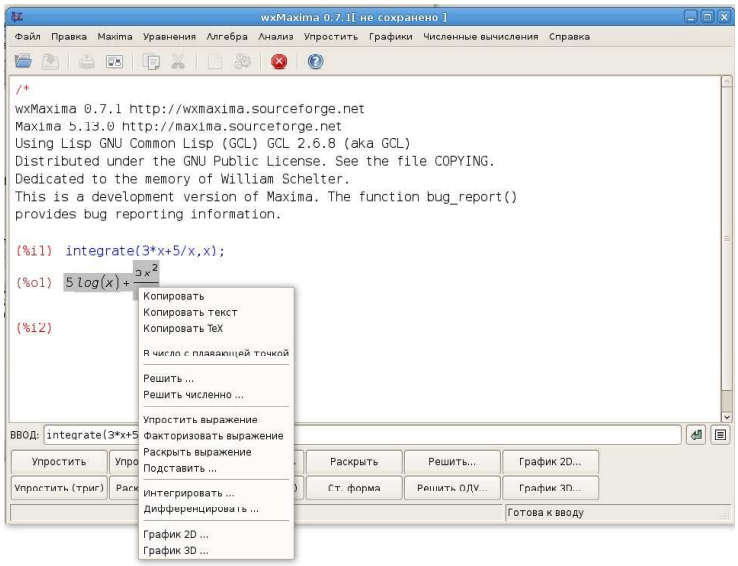


Рис. 6.3. Контекстное меню для поля вывода в рабочем окне wxMaxima

Оболочка wxMaxima допускает так называемое «автодополнение» в командной строке (что вызывается клавишей «Tab»). Нажатие Tab вызывает ту команду из уже введенных в этой сессии, которая начинается с заданных в командной строке символов.

Кроме того, wxMaxima предоставляет удобный интерфейс к документации по системе Maxima.

6.1.2 Графический интерфейс xMaxima

Интерфейс xMaxima фактически является специфичным видом веб-браузера, т.к. данный интерфейс предусматривает обмен данными с вычислительным ядром Maxima через сокет. Интерфейс довольно бедный и архаичный. Он предполагает, что пользователь владеет командами Maxima и макроязыком программирования. Общий вид рабочего окна xMaxima представлен на рис. 6.4. Пункты меню File, Edit, Options позволяют управлять сессией Maxima, сохранять и запускать batch-файлы. В рабочую книгу xMaxima можно втраивать графики в формате openmath (в зависимости от установки опции plot window). Пример рабочего окна xMaxima с простыми графиками представлен на рис. 6.5. График в рабочей книге можно вращать, редактировать, охранять в файл. Как и wxMaxima, интерфейс xMaxima предоставляет доступ к html-файла помощи по пакету Maxima.

6.1.3 Использование редактора TexMacs в качестве интерфейса Maxima

Широкие возможноси работы в Maxima и других математических пакетах предоставляет редактор TexMacs. Разработчик позиционирует его как Latex-редактор, однако это не совсем так. TexMacs использует собственный внутренний формат, но позволяет экспортировать документы в LaTeX (при этом полученных tex-файл очень похож на результат экспорта в tex документа OpenOffice). TexMacs хорошо локализован и полностью поддерживает русский язык, а также все возможности стандартного текстового процессора. В TexMacs реализован подход к структуре документа, во многом идентичный LaTeX, а также возможности ввода и редактирования сложных математических формул. Недостатком редактора является неудачный выбор способа локализации, что затрудняет открытие документов TexMacs при помощи других редакторов (OpenOffice) и др.

Важной особенностью TeXMacs является возможность встраивать в текст документа сессии работы с различными математическими пакетами (в т.ч. и Maxima). Общий вид рабочего окна TexMacs представлен на рис. 6.6

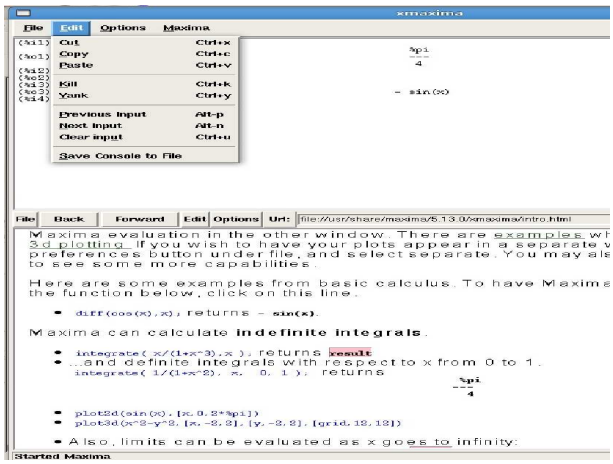


Рис. 6.4. Общий вид рабочего окна xMaxima

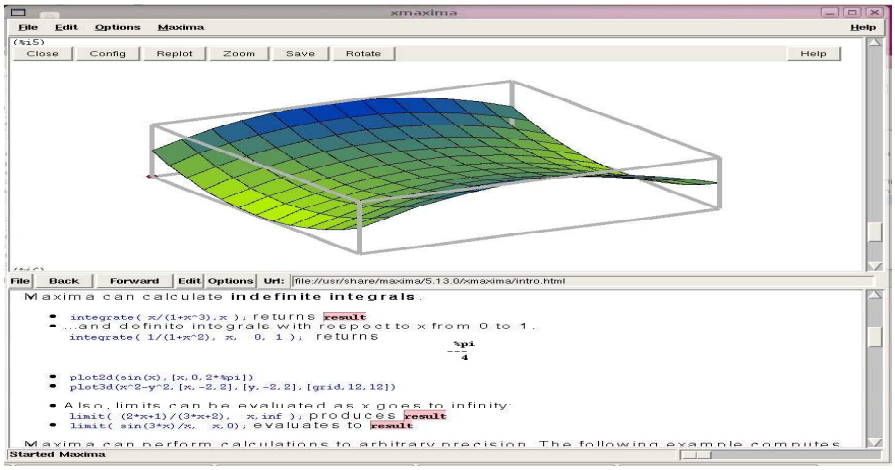


Рис. 6.5. Встроенный график в рабочей книге xMaxima

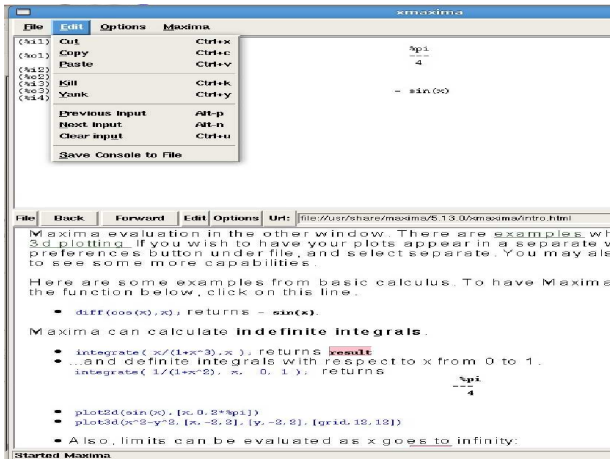


Рис. 6.6. Общий вид рабочего окна TexMacs с запущенной сессией Maxima

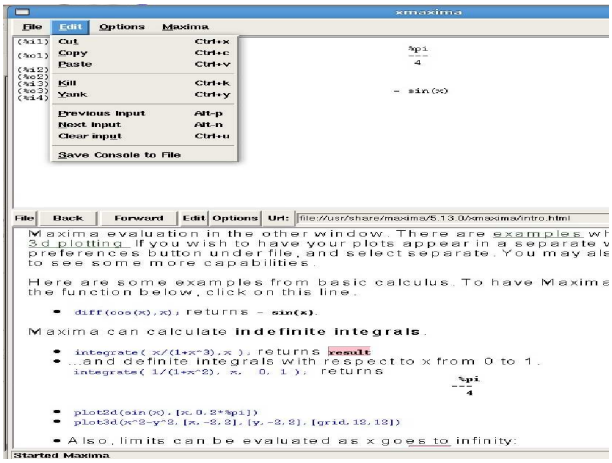


Рис. 6.7. Запуск сессии Maxima в текущем документе TexMacs

Последовательность вставки сессии Maxima в текст документа показана на рис. 6.7.

Возможность встраивать в текст документа графические иллюстрации, также возможность расщеплять сессию для ввода пояснений и комментариев делает TexMacs весьма привлекательным средством для работы с 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. При возникновении проблем с запуском Maxima-сессии из TeXmacs возможным решением является редактирование файла /usr/lib/texmacs/TeXmacs/bin/maxima_detect, в котором ссылку на #!/bin/sh заменить ссылкай на #!/bin/bash в самом начале файла

Окончательную версию TexMacs-документов целесообразно представлять в pdf-формате (этот редактор обеспечивает прямой экспорт в pdf). При сохранении документов в формате TexMacs и их последующем редактировании возможно и редактирование полей ввода сессии Maxima с пересчётом результатов.

6.1.4 Работа с Maxima из Emacs

Универсальный редактор Emacs также может использоваться в качестве front-end к Maxima. Для этого предусмотрено несколько режимов: maxima-mode, EMaxima и iMaxima.

Основной режим работы с Maxima в Emacs - maxima-mode. Этот режим запускается клавиатерной комбинацией M-x-maxima-mode (обычно нажатием alt-M-alt-x и после появления подсказки - набор maxima). Этот режим несколько аскетичен (похож на xMaxima), но достаточно удобен. Общий вид рабочего окна для данного режима представлен на рис. 6.8.

На рис. 6.8 показано также меню навигации по текщей сесии, позволяющее показывать необходимый участок сессии, сохранять часть результатов в протокол, повторять ввод уже использовавшихся в данной сессии команд и т.п. Графики в рабочую книгу, открытую в Emacs, не встраиваются. Сохранение копии рисунка должно выполняться средствами gnuplot или openmath.

Интерфейс EMaxima — скорее не самостоятельный режим, а надстройка над режимом LaTeX, которая наверняка понравится 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. Запуск режима осуществляется командой EMaxima-mode (M-x emaxima).

В простейшем случае с использованием EMaxima можно создать ячейку Maxima комбинацией C-c C-o («open cell»), ввести в ней любую команду или набор команд Максимы в текстовой нотации и получить результат вычисления этой команды либо в обычном текстовом виде нажатием C-c C-u c, либо в LaTeX-виде с помощью C-c C-u C (т. е. Ctrl-c Ctrl-u Shift-c). Здесь «u c» происходит от

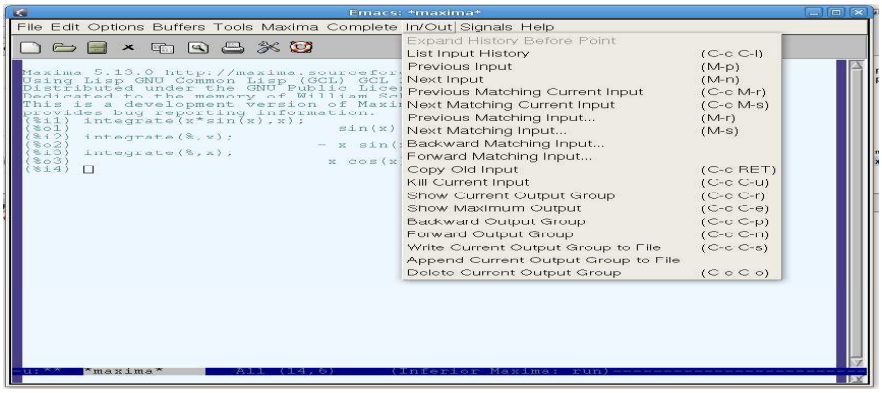


Рис. 6.8. Общий вид рабочего окна Emacs с запущенной сессией Maxima

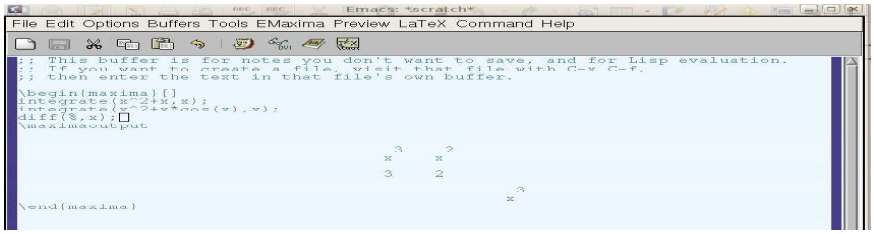


Рис. 6.9. Общий вид рабочего окна Emacs с запущенной сессией EMaxima

«update cell»; а смежные команды, генерирующие вывод в простой текстовой форме и в форме LaTeX, всегда привязаны в Emaxim'е к одинаковым строчной и заглавной буквам соответственно. Пример работы с EMaxima представлен на рис. 6.9, где показаны результаты создание ячейки с Maxima-кодом и результаты дополнения ячейки (команды можно выбирать из меню EMaxima в верхнейчасти рабочего окна).

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

Последний Emacs-интерфейс к Maxima — iMaxima — отличается от остальных самостоятельным (а не посредством LaTeX-документа, как в EMaxima) графическим представлением математических формул. Собственно, именно для этого он и создан, и его отличие от Maxima-mode заключается именно в возможности графического отображениия TeX-кода, генерируемого Maxima. Этот режим можно настроить таким образом, чтобы внутри него запускался режим Maxima (т. е. Maxima-Emacs), и пользоваться всеми командами последнего и их клавиатурными привязками. Т.е. фактически режим iMaxima в таком варианте можно рассматривать как графический интерфейс уже над Maxima-Emacs; именно это может добавить дополнительной привлекательности последнему. В отличие от всех рассмотренных выше интерфейсов, iMaxima — сторонний проект, разрабатываемый отдельно. Для его установки необходимо дополнительно установить пакет breqn, отвечающий за перенос строк в математических формулах в формате LaTeX. Инструкцию по установке самой iMaxima и breqn можно найти на сайте проекта.

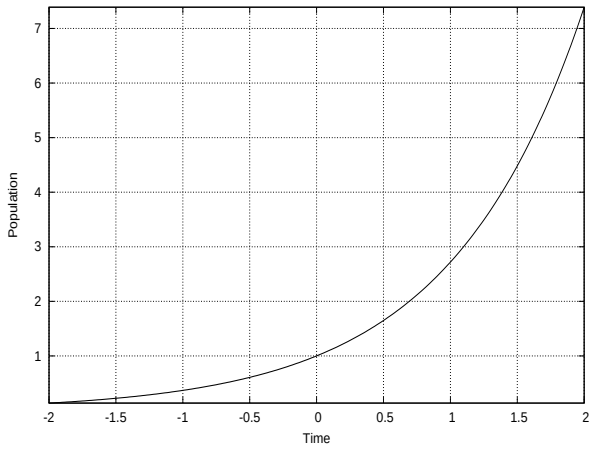


Рис. 6.10. График, построенный с помощью функции draw2d

6.2 Построение графических иллюстраций при помощи пакета draw

. В Maxima имеется несколько альтернативных библиотек для отображения графиков функций, наборов точек, трехмерных тел, градиентов и т.д. По умолчанию используется библиотека Plot, но для решения некоторых задач может оказаться удобнее библиотека Draw. Варианты использования команды plot2d рассмотрены выше, поэтому ниже иллюстрируются возможности draw. Библиотека draw построена на интерфейсе Maxima-gnuplot. Библиотека включает три основные функцииЮ, доступные на уровне Maxima: draw2d, draw3d, draw. Перед использованием draw необходимо загрузить командой

```
load("draw")
```

Рассмотрим несложный пример. На графике (рис. 6.10) показана кривая y=exp(x). График построен с использованием функции draw2d. Функции, заданные явно, указываются командой explicit. Для каждой функции указывается имя переменной и пределы изменения абсциссы. Пределы ординаты выбираются автоматически. Команда построения графика:

```
(%i4) draw2d(grid=true,xlabel = "Time",ylabel = "Population",explicit(exp(u),u,-2,2))$
```

На графике показана сетка (grid=true), а также метки осей (xlabel и ylabel).

Вывод графика на печать организуется при помощи указания типа терминала. Возможные варианты - screen (экран по умолчанию), png, jpg, eps, eps_color, gif, animated_gif, wxt, aquaterm.

Команда для вывода графика на печать имеет вид (указан терминал eps - encapsulated postscript):

```
(%i6) draw2d(terminal='eps,grid=true,xlabel = "Time",ylabel = "Population",explicit(exp(u),u,
 -2,2))$
```

По умолчанию файл сохраняется в файл maxima_out.eps; указание имени файла для вывода осуществляется командой

```
file\_name = "имя файла"
```

Построим аналогичный график (рис. 6.11), но с выводом кривых y=exp(x) и y=exp(-x) в одних осях с сохранением графика в файл draw_2.eps. Необходимая команда:

```
(%i7) draw2d(terminal='eps, file_name="draw_2",grid=true,xlabel = "x",ylabel = "y",explicit(exp(u),u
 -2,2),explicit(exp(-u),u,-2,2))$
```

С помощью пакета draw можно строить и графики функций, заданных неявно. В этом случае функция задаётся командой implicit, например (результат построения - на рис. 6.12:

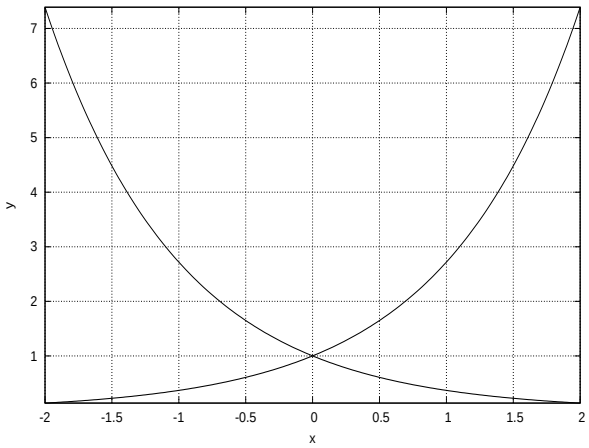


Рис. 6.11. График двух функций, построенный с помощью функции draw2d

```
(%i1) load(draw)$
(%i2) draw2d(terminal  = eps,
             grid      = true,
             line_type = solid,
             key       = "y^2=x^3-2*x+1",
             implicit(y^2=x^3-2*x+1, x, -4,4, y, -4,4),
             line_type = dots,
             key       = "x^3+y^3 = 3*x*y^2-x-1",
             implicit(x^3+y^3 = 3*x*y^2-x-1, x,-4,4, y,-4,4),
             title     = "Two implicit functions" )$
```

На графике хорошо видно, что кривые проведены разными линиями (одна сплошная, другая точечная). Для указания типа линиии использована опция line_type=тип линии (возможные значения - solid и dots).

Помимо графиков неявных функций, при помщи draw могут быть построены и графики параметрических функций или функций, заданных в полярных координатах. В этих случаях вместо команд explicit или implicit используются команды parametric и polar соответственно. Пример графика функции в полярнызх координатах - на рис. 6.13. Соответствующая команда:

```
draw2d(user_preamble = "set grid polar",
             nticks        = 200,
             xrange        = [-5,5],
             yrange        = [-5,5],
             color         = blue,
             line_width    = 3,
             title         = "Hyperbolic Spiral",
             polar(10/theta,theta,1,10*%pi) )$
```

В последнем примере указаны параметры построения графика: интервалы изменеия x и y, равные xrange и yrange, толщина линии line_width и её цвет color. Кроме того, важным параметром является опция user_preamble. Эта опция указывает команды gnuplot, определённые пользователем и выполняющиеся перед построением даннного графика. Для использования меток осей и заголовков на русском языке необходимо в user_preamble или в специальном файле .gnuplot (этот файл содержит команды gnuplot, выполняющиеся при старте программы) указать русскую кодировку командой "set encoding koi8r"или украинскую кодировку "set encoding koi8u". Кроме того, часто оказывается необзодимым указать и шрифт для вывода заголовка или меток.

Функция draw3d позволяет строить трёхмерные графики. Пример:

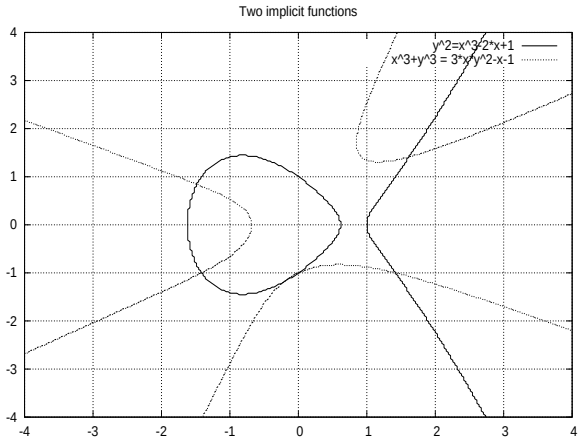


Рис. 6.12. График двух функций, заданных неявно

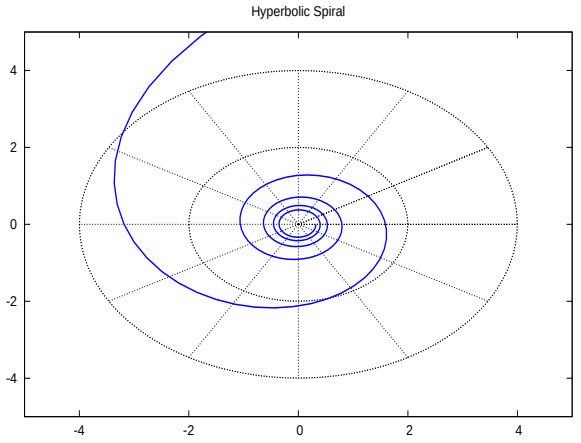


Рис. 6.13. График функции в полярных координатах

```
(%i8) draw3d(zlabel = "Z variable",ylabel = "Y variable",explicit(sin(x^2+y^2),x,-2,2,y,
 -2,2), xlabel = "X variable" )$
```

Метка оси z указывается командой zlabel=имя. Вывод графика на печать аналогичен указанному выше. Пример (с указанием, помимо меток осей, и названия графика командой title=имя) приведен на рис. 6.14.

Очевидно, что при помощи функции draw3d, можно строить и окрашенные поверхности (либо полутоновые). Для этого в качестве аргумента функции draw3d указывается опции enhanced3d (указывает на построение трёхмерной окрашенной поверхность) и palette (palette=color - цветная поверхность, palette=gray - оттенки серого). Пример поверхности, окрашенной оттенками серого - на рис. 6.15. Необходимая команда:

```
(%i12) draw3d(terminal='eps,surface_hide = true,enhanced3d  = true, palette = gray,   explicit(20*ex
 -x^2-y^2)-10,x,-3,3,y,-3,3))$
```

Пакет draw позволяет и строить несколько графиков на одном рисунке, а также предоставляет ряд других полезных возможностей, но для их использования необходимо ознакомиться с документацией, поставляемой с пакетом Maxima.

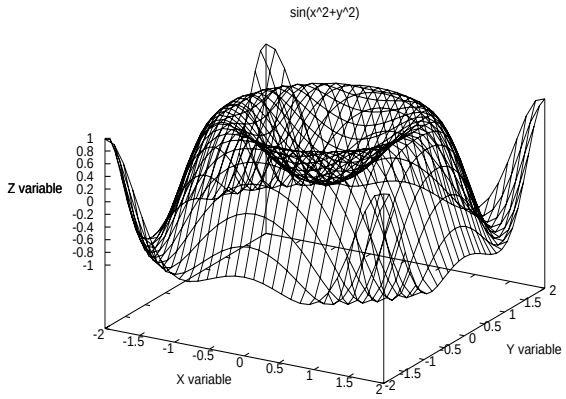


Рис. 6.14. Поверхность, построенная с помощью функции draw3d

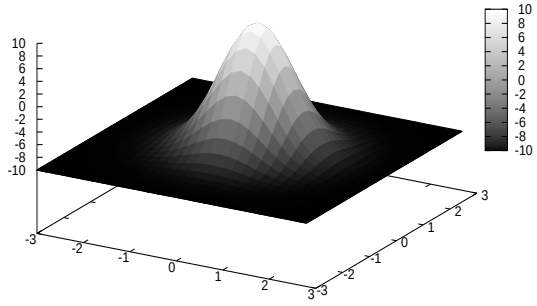


Рис. 6.15. Окрашенная поверхность, построенная с помощью функции draw3d

Глава 7

Моделирование с Maxima

7.1 Общие вопросы моделирования

7.1.1 Аналитические модели

Одним из главных результатов многовекового развития науки является познание и объяснение бесчисленного множества объективно существующих явлений и процессов, протекающих на разных уровнях живой и неживой природы. Компоненты теоретического арсенала современной науки – картины мира, теории, законы, принципы – все они от наиболее общих практически универсальных, таких как законы сохранения вещества и энергии, начала термодинамики, закон всемирного тяготения и других, до сугубо локальных, относящихся к узкому классу объектов или явлений, носят модельный характер. Таким образом в распоряжении исследователя, решающего на основе моделирования конкретную исследовательскую или прикладную задачу, сегодня находится огромное множество моделей-заготовок, которые, очевидно, могут и должны быть использованы. Наиболее благоприятной является ситуация, когда подлежащие описанию и исследованию свойства объекта удается представить непосредственно на основе ранее разработанных и практически достоверных модельных конструктов, являющихся элементами соответствующих областей теоретического знания (механики, термодинамики, электротехники и т.п.). В этом случае создаваемая конкретная модель должна быть охарактеризована как аналитическая (теоретическая). Она, как правило, не только описывает свойства и характеристики объекта, но вскрывает и в терминах соответствующих теорий выявляет сущность процессов, протекающих в исследуемом объекте. Все допущения и ограничения переносятся на модель.

На практике теоретические модели выступают в двух основных ролях.

Прежде всего, они образуют структурную основу и являются главным исходным материалом всех без исключения теоретических построений. Любая теория, относящаяся к сфере точных наук, есть не что иное, как система взаимосвязанных аналитических моделей, подчиненная регулятивным принципам и универсальным зависимостям более высокого уровня.

В поисковых областях научного знания теоретические модели, предназначенные для объяснения и описания явлений, не укладывающихся в существующие теоретические представления, играют роль главного инструмента познания. В сложившихся областях научного знания, главным образом, прикладного характера, таких как теоретическая механика, теоретическая электротехника и др. аналитические модели, в большей или меньшей мере дополняемые обобщенными экспериментальными данными, носят типовой канонический характер, они являются важнейшей составной частью понятийного аппарата, специфического языка и профессионального мышления. Вместе с тем, модели этого класса являются основой для решения множества конкретных прикладных задач, в частности инженерно-технического характера, относящимся к хорошо изученным, не слишком сложным объектам и носящих типовой или рутинный характер. Расчет прочностных характеристик конструкций, расчеты параметров и характеристик электрических цепей. В каждом конкретном случае модель исследуемого явления строится с учетом специфики природы и свойств объекта. Вместе с тем можно указать и некоторые общие методы и приемы.

В основе аналитических моделей, как правило, лежат так называемые балансовые соотношения, связывающие входные и выходные переменные или некоторые функционалы от этих переменных,

имеющие смысл обобщенных сил, обобщенных потоков или координат. Типичные примеры: условие равновесия сил или моментов, действующих на некоторую механическую систему, равенство масс исходных и конечных продуктов некоторой химической реакции, равенство нулю суммы ЭДС и падений напряжений в электрической цепи и т.п. Все эти и прочие им подобные соотношения по существу представляют собой частные проявления законов сохранения вещества и энергии. К этой основе добавляется необходимая дополнительная информация, не вытекающая из этих соотношений, источником которой может быть либо специфическая для данного класса объектов теория, либо эксперимент.

Возможности чисто теоретического решения вопроса уменьшаются с ростом сложности и новизны исследуемого объекта. Впрочем, опыт показывает, что нередко даже для широко используемых на практике и казалось бы, хорошо изученных объектов и процессов, например металлургических, чисто аналитическим путем построить удовлетворительную модель не удается и это побуждает исследователя к формированию модели преимущественно на экспериментальной основе, т.е. в классе идентифицируемых моделей.

7.1.2 Идентифицируемые модели

В основе всех ныне весьма многочисленных методов идентификации или опытного отождествления модели с объектом-оригиналом, лежит идея мысленного эксперимента с «черным ящиком» (Н. Винер). В предельном (теоретическом) случае «черный ящик» представляет собой некую систему, о структуре и внутренних свойствах которой неизвестно решительно ничего. Зато входы, т.е. внешние факторы, воздействующие на этот объект, и выходы, представляющие собой реакции на входные воздействия, доступны для наблюдений (измерений) в течение неограниченного времени. Задача заключается в том, чтобы по наблюдаемым данным о входах и выходах выявить внутренние свойства объекта или, иными словами, построить модель. Решение задачи допускает применение двух стратегий:

1.Осуществляется активный эксперимент. На вход подаются специальные сформированные тестовые сигналы, характер и последовательность которых определена заранее разработанным планом. Преимущество: за счет оптимального планирования эксперимента необходимая информация о свойствах и характеристиках объекта получается при минимальном объеме первичных экспериментальных данных и соответственно при минимальной трудоемкости опытных работ. Но цена за это достаточно высока: объект выводится из его естественного состояния (или режима функционирования), что не всегда возможно. 2.Осуществляется пассивный эксперимент. Объект функционирует в своем естественном режиме, но при этом организуются систематические измерения и регистрация значений его входных и выходных переменных. Информацию получают ту же, но необходимый объем данных существенно, на 2-3 порядка больше, чем в первом случае.

На практике при построении идентифицируемых моделей часто целесообразна смешанная стратегия эксперимента. По тем входным переменным конкретного объекта, которые это допускают (по условиям безопасности, техническим, экономическим соображениям и пр.), проводится активный эксперимент. Его результаты дополняют данными пассивного эксперимента, охватывающего все прочие значимые переменные. «Черный ящик» - теоретически граничный случай. На деле есть объем исходной информации. На практике приходится иметь дело с «серым», отчасти прозрачным ящиком.

Поэтому различают три основных класса постановки задачи идентификации объекта: 1. Для сложных и слабо изученных объектов системного характера достоверные исходные данные о внутренних свойствах и структурных особенностях исчезающе малы, почти отсутствуют. Поэтому задача идентификации, казалось бы, должна включать в себя с одной стороны, определение зависимостей, связывающих входы и выходы (обобщенного оператора), с другой определение внутренней структуры объекта. В такой постановке задача не разрешима даже теоретически. Непосредственным результатом идентификации является только определение зависимостей входы-выходы, причем не в параметрической форме – в виде таблиц или кривых. Для того, чтобы говорить о структуре модели, необходимо перейти к параметрической форме их представлений. Однако, как известно, однозначной связи между функциональной зависимостью и порождающей эту зависимость математической структурой не существует. Каждую непараметрическую зависимость вход-выход можно аппроксимировать различными способами и соответственно построить ряд практически равноцен-

ных моделей, характеризующихся собственной структурой, собственным набором параметров и их значений. Основанием для предпочтения той или иной параметрической модели могут быть только данные, внешние по отношению к процессу идентификации, например, основанные на теоретических соображениях. Если таких данных нет, то в рассматриваемой ситуации мы получаем чисто функциональную или имитационную модель, которая воспроизводит с тем или иным приближением характеристики объекта. 2. Второй класс задач идентификации характеризуется тем, что априорные данные о структуре моделируемого объекта, в принципе имеются. Однако, какой вклад в характеристики объекта или его модели вносит тот или иной компонент, заранее не известно и это надлежит определить на основе эксперимента наряду со значением соответствующих параметров. Типичный пример – исследование влияния на характеристики динамической системы, описанной в классе стационарных линейных моделей, старших членов соответствующих дифференциальных уравнений ради того, чтобы исключить малые, практически незначимые переменные, и снизив порядок уравнений, упростить модель. Задачи этого класса, связанные с уточнением структуры и оценивания параметров, часто встречаются на практике и характерны для объектов и процессов средней сложности, в частности технологических. 3. Третий класс задач связан с относительно простыми и хорошо изученными объектами, структура которых известна точно и речь идет только о том, чтобы по экспериментальным данным оценить значения всех или некоторых входящих в исследуемую структуру параметров (параметрическая идентификация). Очевидно, что модели данного класса тесно смыкаются с требующими экспериментального доопределения аналитическими моделями и четкой границы между ними не существует. Это наиболее массовый класс задач. Независимо от характера решаемой на основе идентификации задачи, построение модели базируется на результатах измерений соответствующих величин переменных.

С этим связаны два существенных обстоятельства: Во-первых, эксперимент должен быть обеспечен необходимыми средствами измерения надлежащей точности (датчики, измерительные преобразователи, средства регистрации). Во-вторых, измерительный комплекс со всеми его компонентами требует метрологического обеспечения, т.е. градуировки, аттестации и периодичности проверки.

Реальные свойства подавляющего большинства сложных объектов, а также неизбежные случайные погрешности измерений, лежащих в основе идентификации, придают последней статистический характер, что влечет за собой необходимость получения больших объемов первичных экспериментальных данных с их последующей обработкой. Поэтому на практике построение моделей путем идентификации неизбежно связано с использованием информационно-вычислительной техники как при получении первичных данных (автоматизация эксперимента), так и для их обработки и использования.

7.2 Статистические методы анализа данных

7.2.1 Ввод-вывод матричных даннх

Для чтения и записи матричных или потоковых данных в составе Maxima предусмотрен пакет numericalio.

Функции пакета рассчитаны на ввод/вывод данных, каждое поле которых предполагается атомом (в смысле lisp), т.е. целых чиесл, чисел с плавающей точкой, строк или символов. Атомы воспринимаются numericalio так же, как при интерактивном вводе в консоли или выполнении batch-файла. Возможно использование различных символов-сепараторов для разделения полей данных (параметр separator_flag).

Основны функции пакет numericalio:

- read_matrix (file_name) (другие формы - read_matrix (file_name, separator_flag), read_matrix (S), read_matrix (S, separator_flag)). Функция read_matrix считывает матрицу из файла. Здесь file_name - имя файла, из которого считываются данные, S - имя потока. Если не указан separator_flag, данные предполагаются разделёнными пробелами. Функция возвращает считанный объект.

- read_list (file_name) (другие формы - read_list (file_name, separator_flag), read_list (S), read_list (S, separator_flag)) - считываетсписок из файла или из потока.

- write_data (X, file_name) (другие формы - write_data (object, file_name, separator_flag), write_data (X, S), write_data (object, S, separator_flag))- осуществляет вывод объекта object (списка, матрицы, массива Lisp или Maxima и др.) в файл file_name (или объекта X в поток S). Матрицы выводятся по столбцам и строкам с использованием пробела или другого символа-разделителя (см. separator_flag).

Наряду с указанными простыми функциями, используются более специфичные - read_lisp_array, read_maxima_array, read_hashed_array, read_nested_list, предназначенные для считывания массивов в формате Lisp или Maxima, особенности применения которых не рассматриваются в данной книге.

7.2.2 Функции Maxima для расчёта описательной статистики

Система Maxima содержит ряд функций для выполнения статистических расчётов (описательной статистики), объединённые в пакет descriptive. Функции, входящие в состав descriptive, позволяют выполнить расчёт дисперсии, среднеквадратичного отклонения, медианы, моды и т.п. Названия функций и краткое описание выполняемых ими действий приведены в таблице.

Функция	**Выполняемые действия**	**Синтаксис вызова и примечания**
mean	вычисление среднего	mean (list) или mean (matrix)
geometric_mean	вычисление среднего геометрического	geometric_mean(list) или geometric_mean(matrix)
harmonic_mean	вычисление среднего гармонического	harmonic_mean(list) или harmonic_mean(matrix)
cor	Вычисляет корреляционную матрицу	cor (matrix) или cor (matrix, logical_value); logical_value равна true или false (false - при расчёте по ковариационной матрице)
cov, cov1	Вычисляет ковариационную матрицу	cov1 (matrix), cov (matrix)
median	Вычисляет медиану	median (list), median (matrix)
std std1	Вычисляет среднеквадратичное отклонение (корень квадратный из var или var1)	аналогично var
var, var1	Вычисляет дисперсию случайной величины	var1 (matrix), var (matrix), var1 (list), var (list)
central_moment	Вычисляет центральный момент порядка k	central_moment (list, k), central_moment (matrix, k)
noncentral_moment	Вычисляет момент порядка k	noncentral_moment (list, k), noncentral_moment (matrix, k)

skewness	Вычисление асимметрии	skewness (list), skewness (matrix)
kurtosis	Вычисление эксцесса	kurtosis(list), kurtosis(matrix)
quantile	Вычисление p-квантиля	quantile (list, p),quantile (matrix, p)
maxi, mini	Выбор наибольшего и наименьшего значения в выборке соответственно	maxi(list), maxi(matrix), mini(list), mini(matrix)
mean_deviation, median_deviation	Сумма абсолютных отклонений от среднего или медианы соответственно	Аналогично mean, median
range	Размах вариации выборки	range (list), range (matrix)
list_correlations	Возвращает список, включающий две матрицы - матрицу, обратную ковариационной, и матрицу частных коэффициентов корреляции	list_correlations (matrix), list_correlations (matrix, logical_value)logical_value равна true или false (false - при расчёте по ковариационной матрице)
subsample	Аналог функции submatrix	
global_variances	возвращаеь список, содержащий различные виды дисперсии	global_variances (matrix)

Построение графических иллюстраций производися при помощи функций dataplot(непосредственная визуализация данных), histogram (строит гистограмму), barsplot (также строит гистограмму, но по дискретным или нечисловым данным), boxplot (график Бокса-Вискера). Синтакисис вызова и параметры функций во многом аналогичны компонентам пакета draw (см. примеры ниже).

Рассмотрим пример использования функций из пакета descriptive для статистической обработки массива данных из файла biomed.dat (входит в состав пакета descriptive). Перед началом работы загружаем пакеты descriptive и numericalio.

```
(%i1) load(descriptive)$ load(numericalio)$
s:read_matrix (file_search ("wind.data"))$
(%i4) length(s);
  При помощи функции read_matrix считана матрица,
содержащая 100 строк и 5 столбцов.
```

(%o4) 100

```
(%i5) mean(s); рассчитываем среднее значение.
При обработке матрицы получаем список средних по столбцам.
```

(%o5) $[9.948499999999999, 10.1607, 10.8685, 15.7166, 14.8441]$

```
(%i6) median(s);
```

(%o6) $[10.06, 9.855, 10.73, 15.48, 14.105]$

```
(%i7) var(s);
```

(%o7) $[17.22190675000001, 14.98773651000001, 15.47572875, 32.17651044000001, 24.42307619000001]$

```
(%i8) std(s);
```

(%o8)
$[4.149928523480858, 3.871399812729242, 3.933920277534866, 5.672434260526957, 4.941970881136392]$

```
(%i9) mini(s);
```

(%o9) $[0.58, 0.5, 2.67, 5.25, 5.17]$

```
(%i10) maxi(s);
```

(%o10) $[20.25, 21.46, 20.04, 29.63, 27.63]$

```
(%i11) mini(%); При обработке списка и поиске в нём минимального элемента получаем одно значение!
```

(%o11) 20.04

Основная функция для вывода графических иллюстраций - функция dataplot. Синтаксис вызова:

```
dataplot (list)
dataplot (list, option_1, option_2, ...)
dataplot (matrix)
dataplot (matrix, option_1, option_2, ...)
```

Функция dataplot рассчитана на прямую визуализацию как нескольких наборов данных (представленных матрицей), так и одного набора данных (представленного списком). Допустимые опции:

- 'outputdev - устройство , на которое выводится диаграмма (возможные значения - "x "eps"и "png по умолчанию "x");
- 'maintitle - основной заготовок, по умолчанию ;
- 'axisnames - названия осей, по умолчанию ["x "y "z"],
- 'joined - флаг соединения точек линией (true или false, по умолчанию false);
- 'picturescales - масштабные коэффициенты осей, по умолчанию [1.0, 1.0];
- 'threedim - указывает, строить ли трехмерный график по матрице с тремя столбцами, по умолчанию true
- 'axisrot - углы, определябщие наклон оси зрения для трехмерного графика, по умолчанию [60, 30],

- 'nclasses - число классов для построения гистограммы, по умолчанию 10
- 'pointstyle - стиль точек (целое число, по умолчанию 1).

Одномерные массивы рассматриваются как временные ряды с равноотстоящими точками.

Для построения гистограмм используется функция histogram (синтаксис аналогичен dataplot). Основные опции идентичны опциям dataplot. Рассмотрим дополнительные опции, специфичные для histogram:

- 'relbarwidth - десятичное число, определяющее ширину столбцов гистограммы (по умолчанию 0,1; в общем случае от 0 до 1);
- 'barcolor - обозначение цвета столбцов (целое число, по умолчанию 1);
- 'colorintensity обозначение интенсивности цвета (по умолчанию 1; десятичное число от 0 до 1).

Опции функции barsplot идентичны опциям histogram.

Функция boxplot используется для сопоставления различных наборов данных. Синтаксис вызова: boxplot (data) или boxplot (data, option_1, option_2, ...). Параметр data - список или матрица с несколькими столбцами. Опции функции boxplot идентичны опциям dataplot.

Рассмотрим примеры использования графических утилит пакета descriptive.

Для дальнейшего использования считываем данные из файла wind.dat (это тестовый файл в составе пакета descriptive, содержит матрицу 100x5).

```
(%i1) load(descriptive)$ load(numericalio)$ s:read_matrix (file_search ("wind.data"))$
(%i4) x:makelist(s[k][1],k,1,length(s))$
(%i5) y:makelist(s[k][2],k,1,length(s))$
(%i6) m:makelist([x[k],y[k]],k,1,100)$
(%i7) xy:apply('matrix,m)$
```

Строим график (точечный) зависимости y от x (рис.7.1). Результаты сохраняются в файле dataplot.eps (имя файла - по умолчанию, он создаётся в домашнем каталоге пользователя). Необходимые команды:

```
(%i8) dataplot(xy,outputdev="eps");
```

(%o8) 0

Строим гистограмму частотного распределения вектора x (рис. 7.2).Результаты сохраняются в файле histogram.eps (имя файла - по умолчанию, он создаётся в домашнем каталоге пользователя). Необходимые команды:

```
(%i9) histogram(x,outputdev="eps");
```

(%o9) 0

Следующий пример - график Бокса-Уискера с аннотациями по осям (см. рис. 7.3). Необходимые команды:

```
(%i10) boxplot(s,'maintitle = "Тестовый график",'axisnames = ["Сезоны", ""],outputdev="eps");
```

График с использованием функции barsplot - на рис. 7.4. График построен командой (по умолчанию график созраняется в файл barsplot.eps):

```
s3 : read_matrix (file_search ("biomed.data"))$ barsplot (col (s3, 1), 'maintitle
  = "Группы пациентов",'axisnames=["Group", "# of individuals"], 'colorintensity=0.2,outputdev="eps"
```

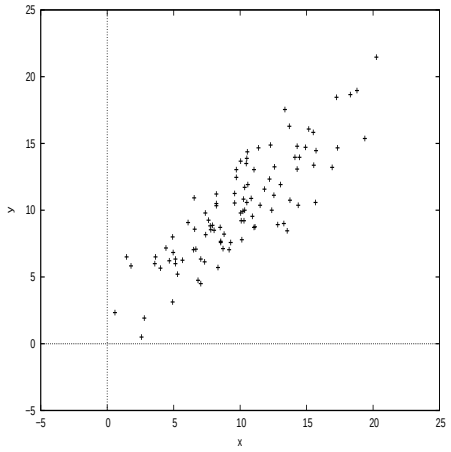


Рис. 7.1. Точечный график

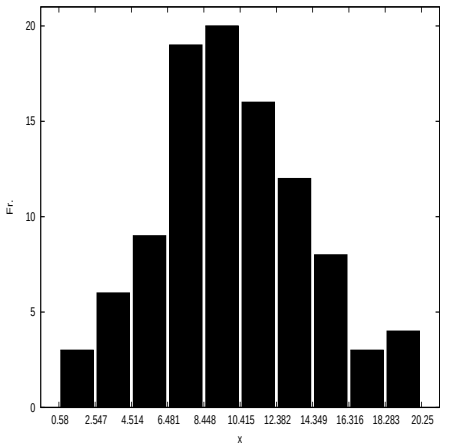


Рис. 7.2. Гистограмма

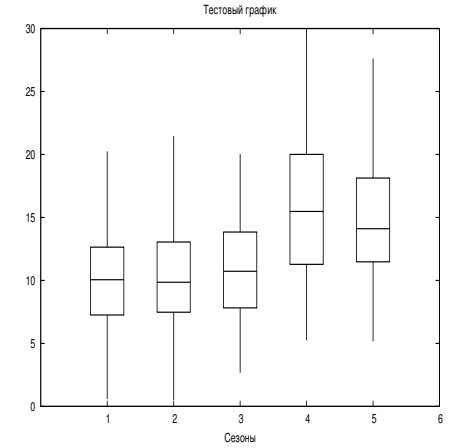


Рис. 7.3. График Бокса-Уискера

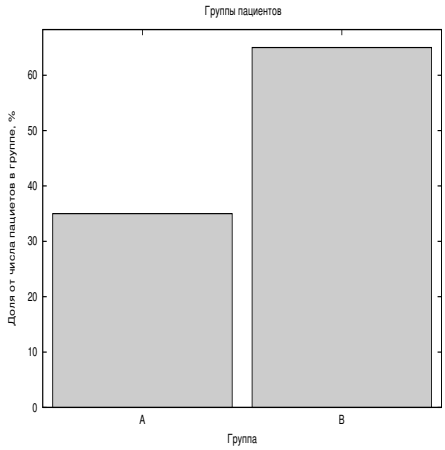


Рис. 7.4. Гистограмма распределения в группах

К сожалению, базовая программа вывода графики Maxima - gnuplot - написана очень давно, и воспринимает кириллические символы тольков кодировках KOI8-R или KOI8-U. Возможным решением (принятым для построения графиков в этой книге) является создание файла .gnuplot, содержащего следующие команды:

```
set encoding koi8r
```

Однако и в этом случае может возникнуть необходимость в редактировании файла maxout.gnuplot, содержащего команды для построения последнего графика.

7.2.3 Проверка статистических гипотез

Для проверки статистических гипотез в Maxima включён пакет stats. Он позволяет, в частности, проводить сопоставление средних или дисперсий двух выборок. Предусмотрена и проверка нормальности распределения, а также ряд других стндартных тестов. для использования stats пакет необходимо загрузить командой load("stats"); необходимые пакеты descriptive и distrib загружаются автоматически.

Функции пакета stats возвращают данные типа inference_result. Объекты этого типа содержат необходимые результаты для анализа статистических распределениц и проверки гипотез.

Функция test_mean позволяет оценить среднее значение и доверительный интервал по выборке. Синтаксис вызова:

```
test_mean (x) или test_mean (x, option_1, option_2, ...).
```

Функция test_mean использует проверку по критерию Стьюдента. Аргумент x - список или одномерная матрица с тестируемой выборкой. Возможно также использование центральной предельной теоремы (опция asymptotic). Опции test_mean:

- 'mean, по умолчанию 0, ожидаемое среднее значение;
- 'alternative, по умолчанию 'twosided, вид проверяемой гипотезы (возможные значения 'twosided, 'greater и 'less);
- 'dev, по умолчанию 'unknown, величина среднеквадратичного отклонения, если оно известно (unknjwn или положительное выражение);
- * 'conflevel, по умолчанию 95/100, уровень значимости для доверительного интервала (величина в пределах от 0 до 1);

- * 'asymptotic, по умолчанию false, указывает какой критерий использовать (t-критерий или центральную предельную теорему).

Результаты, кторые возвращает функция:

- 'mean_estimate: среднее по выборке;
- 'conf_level: уровень значимости, выбранный пользователем;
- 'conf_interval: оценка доверительного интервала;
- 'method: использованная процедура;
- 'hypotheses: проверяемые статистические гипотезы (нулевая H_0 и альтернативная H_1);
- 'statistic: число степеней свободы для проверки нулевой гипотезы;
- 'distribution: оценка распределения распределения выборки;
- 'p_value: вероятность ошибочного выбора гипотезы H_1, если выполняется H_0.

Примеры использования test_mean:
Выполняется t-тест с неизвестной дисперсией. Нулевая гипотеза H_0: среднее равно 50 против альтернативной гипотезы H_1: среднее меньше 50; в соответствии с результатов расчёта, величина вероятности p слишком велика, чтобы отвергнуть H_0.

```
(%i1) load("stats")$
   (%i2) data: [78,64,35,45,45,75,43,74,42,42]$
   (%i3) test_mean(data,'conflevel=0.9,'alternative='less,'mean=50);
             |                  MEAN TEST
             |
             |            mean_estimate = 54.3
             |
             |              conf_level = 0.9
             |
             | conf_interval = [minf, 61.51314273502712]
             |
   (%o3)     |  method = Exact t-test. Unknown variance.
             |
             | hypotheses = H0: mean = 50 , H1: mean < 50
             |
             |       statistic = .8244705235071678
             |
             |       distribution = [student_t, 9]
             |
             |        p_value = .7845100411786889
```

Следующий тест - проверка гипотезы H_0 (среднее равно 50) против албтернативной гипотезы H_1 среднее по выборке отлично от 50. В соответствии с величиной p«1 нулевая гипотеза. Данный тест применяется для больших выборок.

```
(%i1) load("stats")$
   (%i2) test_mean([36,118,52,87,35,256,56,178,57,57,89,34,25,98,35,
                98,41,45,198,54,79,63,35,45,44,75,42,75,45,45,
                45,51,123,54,151],
                'asymptotic=true,'mean=50);
             |                       MEAN TEST
             |
             |           mean_estimate = 74.88571428571429
```

```
                |
                |                    conf_level = 0.95
                |
                | conf_interval = [57.72848600856194, 92.04294256286663]
                |
    (%o2)       |    method = Large sample z-test. Unknown variance.
                |
                |       hypotheses = H0: mean = 50 , H1: mean # 50
                |
                |             statistic = 2.842831192874313
                |
                |             distribution = [normal, 0, 1]
                |
                |             p_value = .004471474652002261
```

Функция test_means_difference позволяет проверить, принадлежить ли выборки x1 и x2 к одной генеральной совокупности. Синтаксис вызова:

`test_means_difference (x1, x2) или test_means_difference (x1, x2, option_1, option_2, ...).`

Данная функция выполняет t-тест для сравнения средних по выборкам x1 и x2 (x1,x2 - списки или одномерные матрицы). Сравнение выборок может проводиться также на основании центральнйо предельной теоремы (для больших выблрок). Опции функции test_means_difference такие же, как и для test_mean, кроме оценок среднеквадратичных отклонений выборок (если они известны) Список опций:

- 'alternative, по умолчанию 'twosided, вид проверяемой гипотезы (возможные значения 'twosided, 'greater и 'less);
- 'dev1, dev2, по умолчанию 'unknown, величины среднеквадратичных отклонений для выборок x1 и x2, если они известны (unknown или положительное выражение);
- * 'conflevel, по умолчанию 95/100, уровень значимости для доверительного интервала (величина в пределах от 0 до 1);
- * 'asymptotic, по умолчанию false, указывает какой критерий использовать (t-критерий или центральную предельную теорему).

Вывод результатов test_means_difference не отличается от вывода результатов test_mean. Примеры использования test_means_difference: Для двух малых выборок проверяется гипотеза H_0 от равенстве средних против албтернативной гипотезы H_1: различие математических ожиданий статистически значимо, т.е. выборки принадлежатк разным генеральным совокупностям.

```
(%i1) load("stats")$
   (%i2) x: [20.4,62.5,61.3,44.2,11.1,23.7]$
   (%i3) y: [1.2,6.9,38.7,20.4,17.2]$
   (%i4) test_means_difference(x,y,'alternative='greater);
                |              DIFFERENCE OF MEANS TEST
                |
                |         diff_estimate = 20.31999999999999
                |
                |                 conf_level = 0.95
                |
                |    conf_interval = [- .04597417812882298, inf]
                |
   (%o4)        |        method = Exact t-test. Welch approx.
                |
```

```
                | hypotheses = H0: mean1 = mean2 , H1: mean1 > mean2
                |
                |           statistic = 1.838004300728477
                |
                |    distribution = [student_t, 8.62758740184604]
                |
                |            p_value = .05032746527991905
```

Оценка доверительного интервала для дисперсии выборки проводится при помощи функци test_variance. Стинтакис вызова:

```
test_variance (x) или test_variance (x, option_1, option_2, ...)
```

Данная функция использует тест χ^2. Предполагается, что распределение выборки x нормальное. Опции функции test_variance:

- 'mean, по умолчанию unknown, оценка математического ожидания (среднее по выборке), если оно известно;
- 'alternative, по умолчанию 'twosided, вид проверяемой гипотезы (возможные значения 'twosided, 'greater и 'less);
- 'variance, по умолчанию 1, это оценка дисперсии выборки для сравнения с фактической дисперсией;
- * 'conflevel, по умолчанию 95/100, уровень значимости для доверительного интервала (величина в пределах от 0 до 1).

Основной результат, возвращаемый функцией - оценка дисперсии выборки var_estimate и доверительный интервал для неё. Пример: Прверка, отличается ли дисперсия выборки с неизвестным математическим ожиданием от значения 200.

```
(%i1) load("stats")$
  (%i2) x: [203,229,215,220,223,233,208,228,209]$
  (%i3) test_variance(x,'alternative='greater,'variance=200);
                |                   VARIANCE TEST
                |
                |               var_estimate = 110.75
                |
                |                 conf_level = 0.95
                |
                |     conf_interval = [57.13433376937479, inf]
                |
  (%o3)         | method = Variance Chi-square test. Unknown mean.
                |
                |    hypotheses = H0: var = 200 , H1: var > 200
                |
                |                 statistic = 4.43
                |
                |              distribution = [chi2, 8]
                |
                |            p_value = .8163948512777689
```

Сравнение дисперсий двух выборок проводится при помощи функции test_variance (синтасис вызова

␣test_variance_ratio␣(x1,␣x2)␣или␣␣test_variance_ratio␣(x1,␣x2,␣option_1,␣option_2,␣...).

Данная функция предназначена для сопоставления дисперсий двух выборок с нормальным распределением по критерию Фишера (F-тест). Аргументы x1 и x2 - списки или одномерные матрицы, содержащие независимые выборки. Опции функции test_variance_ratio:

- 'mean1,mean2 , по умолчанию unknown, оценкb математических ожиданий выборок x1 и x2, если они известны;
- 'alternative, по умолчанию 'twosided, вид проверяемой гипотезы (возможные значения 'twosided, 'greater и 'less);
- * 'conflevel, по умолчанию 95/100, уровень значимости для доверительного интервала (величина в пределах от 0 до 1).

Основной результат, возвращаемый функцией test_variance_ratio - отношение дисперсий выборок ratio_estimate. Пример: проверяется гипотеза о равенстве дисперсий двух выборок по сравнению с албтернативной гипотезой о том, что дисперсия первой больше, чем дисперсия второй.

```
(%i1) load("stats")$
   (%i2) x: [20.4,62.5,61.3,44.2,11.1,23.7]$
   (%i3) y: [1.2,6.9,38.7,20.4,17.2]$
   (%i4) test_variance_ratio(x,y,'alternative='greater);
                 |              VARIANCE RATIO TEST
                 |
                 |       ratio_estimate = 2.316933391522034
                 |
                 |               conf_level = 0.95
                 |
                 |    conf_interval = [.3703504689507268, inf]
                 |
   (%o4)         | method = Variance ratio F-test. Unknown means.
                 |
                 | hypotheses = H0: var1 = var2 , H1: var1 > var2
                 |
                 |         statistic = 2.316933391522034
                 |
                 |            distribution = [f, 5, 4]
                 |
                 |          p_value = .2179269692254457
```

При отсутствии представлений о распределении выборки может использоваться непараметрический тест для сравнения средних. Оценка медианы неперывной выборки проводится при помощи функции test_sign (синтаксис вызова

␣test_sign␣(x)␣или␣test_sign␣(x,␣option_1,␣option_2,␣...).

Функция test_sign допускает две опции: alternative (аналогично test_mean) и median(по умолчанию 0, или оценка значения медианы для проверки статистической значимости). Результаты, кторые возвращает функция:

- 'med_estimate: медиана выбоки;
- 'method: использованная процедура;
- 'hypotheses: проверяемые статистические гипотезы (нулевая H_0 и альтернативная H_1);
- 'statistic: число степеней свободы для проверки нулевой гипотезы;
- 'distribution: оценка распределения распределения выборки;
- 'p_value: вероятность ошибочного выбора гипотезы H_1, если выполняется H_0.

Пример: проверка гипотезы H_0 о равенстве медианы выбоки 6, против альтернативной гипотезы H_1: медиана больше 6.

```
(%i1) load("stats")$
   (%i2) x: [2,0.1,7,1.8,4,2.3,5.6,7.4,5.1,6.1,6]$
   (%i3) test_sign(x,'median=6,'alternative='greater);
                 |                  SIGN TEST
                 |
                 |              med_estimate = 5.1
                 |
                 |      method = Non parametric sign test.
                 |
   (%o3)         | hypotheses = H0: median = 6 , H1: median > 6
                 |
                 |                statistic = 7
                 |
                 |      distribution = [binomial, 10, 0.5]
                 |
                 |         p_value = .05468749999999989
```

Аналогичная функция - test_signed_rank (x) (либо с указанием опций test_signed_rank (x, option_1, option_2, ...)), которая использует тест правила знаков Вилкоксона для оценки гипотезы о медиане непрерывной выборки. Опции и результаты функции test_signed_rank такие же, как и для функции test_sign. Пример: проверка гипотезы H_0: медиана равна 15 против албтернативной гипотезы H_1: медиана больше 15.

```
(%i1) load("stats")$
   (%i2) x: [17.1,15.9,13.7,13.4,15.5,17.6]$
   (%i3) test_signed_rank(x,median=15,alternative=greater);
                   |             SIGNED RANK TEST
                   |
                   |           med_estimate = 15.7
                   |
                   |           method = Exact test
                   |
   (%o3)           | hypotheses = H0: med = 15 , H1: med > 15
                   |
                   |              statistic = 14
                   |
                   |     distribution = [signed_rank, 6]
                   |
                   |            p_value = 0.28125
```

Непараметрическое сравненние медиан двух выборок реализовано в одной функции - test_rank_sum. В данной функции используется тест Вилкоксона-Манна-Уитни. U-критерий Манна-Уитни - непараметрический метод проверки гипотез, часто использующийся в качестве альтернативы t-тесту Стьюдента. Обычно этот тест используется для сравнения медиан двух распределений x1 и x2, не являющихся нормальными (отсутствие нормальности не позволяет применить t-тест). Синтаксис вызова:

␣test_rank_sum␣(x1,␣x2)␣или␣test_rank_sum␣(x1,␣x2,␣option_1).

Функция допускает лишь одну опцию: alternative (аналогично test_means_difference). Результаты, которые возвращает функция:

- 'method: использованная процедура;

- 'hypotheses: проверяемые статистические гипотезы (нулевая H_0 и альтернативная H_1);
- 'statistic: число степеней свободы для проверки нулевой гипотезы;
- 'distribution: оценка распределения распределения выборки;
- 'p_value: вероятность ошибочного выбора гипотезы H_1, если выполняется H_0.

Пример: проверка, одинаковы ли медианы выборок x1 и x2.

```
(%i1) load("stats")$
   (%i2) x:[12,15,17,38,42,10,23,35,28]$
   (%i3) y:[21,18,25,14,52,65,40,43]$
   (%i4) test_rank_sum(x,y);
                 |                  RANK SUM TEST
                 |
                 |              method = Exact test
                 |
                 | hypotheses = H0: med1 = med2 , H1: med1 # med2
   (%o4)         |
                 |                  statistic = 22
                 |
                 |        distribution = [rank_sum, 9, 8]
                 |
                 |          p_value = .1995886466474702
```

Для выборок большего объёма распределение выборок приблизительно нормальное. Сравниваем гипотезы H_0: медиана 1 = медиана 2 и H_1: медиана 1 < медиана 2.

```
(%i1) load("stats")$
   (%i2) x: [39,42,35,13,10,23,15,20,17,27]$
   (%i3) y: [20,52,66,19,41,32,44,25,14,39,43,35,19,56,27,15]$
   (%i4) test_rank_sum(x,y,'alternative='less);
               |                   RANK SUM TEST
               |
               |          method = Asymptotic test. Ties
               |
               |  hypotheses = H0: med1 = med2 , H1: med1 < med2
   (%o4)       |
               |                 statistic = 48.5
               |
               | distribution = [normal, 79.5, 18.95419580097078]
               |
               |           p_value = .05096985666598441
```

Проверка нормальности распределения осуществляется функцией test_normality (x). В этой функции реализован тест Шапиро-Уилка. Выборка x (список или одномерная матрица) должна быть размером не менее 2, но не более 5000 элементов (иначе выдаётся сообщениеоб ошибке). Функция возвращает два значения: statistic - величина W-статистики и величина вероятности p (если p больше принятого уровня значимости, нулевая гипотеза о нормальности распределения выборки x не отвергается). Статистика W характеризует близость выборочного распределения к нормальному (чем ближе W к 1, тем меньше вероятность ошибочно принять гипотезу о нормальности распределения). Пример: проверка гипотезы о нормальном распределения генеральной совокупности по заданной выборке.

```
(%i1) load("stats")$
   (%i2) x:[12,15,17,38,42,10,23,35,28]$
   (%i3) test_normality(x);
```

```
                          |      SHAPIRO - WILK TEST
                          |
(%o3)                     | statistic = .9251055695162436
                          |
                          |  p_value = .4361763918860381
```

7.2.4 Расчёт коэффициентов линейной регрессии

Коэффициенты и оценка статистической значимости для простейшей линейной регрессии могут оцениваться при помощи функции simple_linear_regression из пакета stats. Функция вычисляет коэффициенты и параметры линейной регрессии y=a0+a1*x (т.е. только простейшей). Синтаксис вызова:

`simple_linear_regression (x) или simple_linear_regression (x option_1).`

Опции функции simple_linear_regression: conflevel (уровень значимости, обычно 0.95, см. выше) и regressor (по умолчанию x, имя независимой переменной). Рассматриваемая функция выводит большое количество статистических параметров:

1. model: полученное уравнение регрессии;
2. means: среднее;
3. variances: дисперсии обоих переменных;
4. correlation: коэффициент корреляции;
5. adc: коэффициент детеринации;
6. a_estimation: оценка параметра a;
7. a_conf_int: доверительный интервал для a;
8. b_estimation: оценка параметра b;
9. b_conf_int: доверительный интервал для b;
10. hypotheses: нулевая и альтернативная гипотеза относительно параметра b;
11. statistic: статистические характеристики выборки, использованные для проверки нулевой гипотезы;
12. distribution: распределение выборки;
13. p_value: величина вероятности для проверки гипотезы о статистической значимости b;
14. v_estimation: оценка остаточной дисперсии;
15. v_conf_int: доверительный интервал для остаточной дисперсии
16. cond_mean_conf_int: доверительный интервал для среднего;
17. new_pred_conf_int: доверительный интервал для нового предсказания;
18. residuals: список, содержащий остатки.

По умолчанию на консоль выводятся только парметры 1, 4, 14, 9, 10, 11, 12, и 13 в этом списке. Остальные параметры скрыты, но доступ к ним обеспечивается при помощи функций items_inference или take_inference.

Задаёмся исходными данными

```
(%i9) s:[[125,140.7], [130,155.1], [135,160.3], [140,167.2], [145,169.8]]
$(%i10) z:simple_linear_regression(s,conflevel=0.99);
```

Вычисляем коэффициенты и прочие параметры регрессии

```
(%i10) z:simple_linear_regression(s,conflevel=0.99);
```

$$(\%o10)\quad \left(\begin{array}{c} SIMPLELINEARREGRESSION \\ model = 1.405999999999986\,x - 31.18999999999804 \\ correlation = 0.96116852552552 \\ v\_estimation = 13.57966666666665 \\ b\_conf\_int = [0.044696336625253, 2.767303663374718] \\ hypotheses = H0 : b = 0, H1 : b\#0 \\ statistic = 6.032686683658114 \\ distribution = [student\_t, 3] \\ p\_value = 0.0038059549413203 \end{array}\right)$$

```
(%i11) z:simple_linear_regression(s,conflevel=0.95);
```

$$(\%o11)\quad \left(\begin{array}{c} SIMPLELINEARREGRESSION \\ model = 1.405999999999986\,x - 31.18999999999804 \\ correlation = 0.96116852552552 \\ v\_estimation = 13.57966666666665 \\ b\_conf\_int = [0.66428743645021, 2.147712563549759] \\ hypotheses = H0 : b = 0, H1 : b\#0 \\ statistic = 6.032686683658114 \\ distribution = [student\_t, 3] \\ p\_value = 0.0038059549413203 \end{array}\right)$$

Некоторые дополнительные параметры:

```
(%i5) take_inference(model,z), x=133;
```

$$(\%o5)\qquad 155.808$$

```
(%i6) take_inference(means,z);
```

$$(\%o6)\qquad [135.0, 158.62]$$

```
(%i7) take_inference(new_pred_conf_int,z), x=133;
```

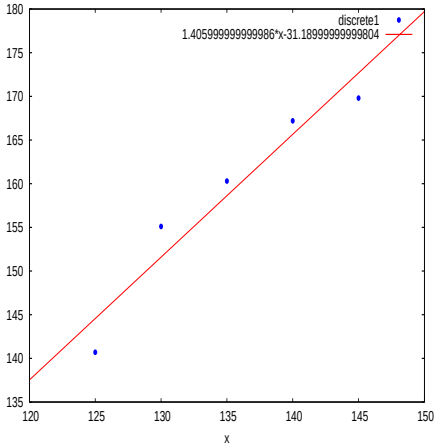


Рис. 7.5. Прростая линейная регрессия

(%o7) [132.0728595995113, 179.5431404004887]

Графическая иллюстрация построенной линейной зависимости - на рис. 7.5. Использованная команда:

```
(%i11) plot2d([[discrete, s], take_inference(model,z)], [x,120,150], [style, [points], [lines]],
              [gnuplot_term, ps], [gnuplot_out_file, "regress.eps"])$
```

7.2.5 Использование метода наименьших квадратов

Пакет Maxima включает мощный модуль для линейного и нелинейного оценивания параметров различных моделей с использованием метода наименьших квадратов - пакет lsqares.

Основная функция пакета lsqares - это функция lsquares_estimates. Синтаксис вызова:

␣lsquares_estimates␣(D,␣x,␣e,␣a)␣или␣␣lsquares_estimates␣(D,␣x,␣e,␣a,␣initial␣=␣L,␣tol␣=␣t)

Функция предназначениа для оценки параметров, лучше всего соответствующие уравнению e в переменных x по набору данныч D определяются методом методом наименьших квадратов. Функция lsquares_estimates сначала пытается отыскать ищет точное решение, и если это не удаётся, ищет приблизительное решение. Возвращаемое значение – список вида [a = ..., b = ..., c = ...]. Элементы списка обеспечивают минимум среднеквадратичнойц ошибки. Данные D должны быть матрицей. Каждый ряд - одна запись или один случай, каждый столбец соответствует значениям некоторой переменной. Список переменных x дает название для каждого столбца D (даже для столбцов, которые не входят в анализ). Список параметров содержит названия параметров, для которых отысиваются оценки. Уравнение e является выражением или уравнением в переменных x и a; если e записано не в форме уравнения, его рассматривают как уравнение e = 0. в Если некоторое точное решение может быть найдено при помощи solve, данные D могут содержать и нечисловые значения. Дополнительные аргументы lsquares_estimates определены как уравнения и передаются «дословно» функции lbfgs, которая используется, чтобы найти оценки численным методом, когда точный результат не найден. Однако, если никакое точное решение не найдено, у каждого элемента D должно быть числовое значение, в том числе константы (такие как %pi и %e) или числовые литералы (целые числа, rationals, обычные плавания, и bigfloats). Численные расчеты выполнены с обычной арифметикой с плавающей запятой, таким образом все другие виды чисел преобразованы к значениям с плавающей точкой. Для работы с lsquares_estimates необходимо загрузить эту функцию командой load(lsquares). Пример (точное решение):

```
(%i1) load (lsquares)$
(%i2) M : matrix ([1, 1, 1], [3/2, 1, 2], [9/4, 2, 1], [3, 2, 2], [2, 2, 1]);
                            [ 1  1  1 ]
                            [         ]
                            [ 3       ]
                            [ -  1  2 ]
                            [ 2       ]
                            [         ]
(%o2)                       [ 9       ]
                            [ -  2  1 ]
                            [ 4       ]
                            [         ]
                            [ 3  2  2 ]
                            [         ]
                            [ 2  2  1 ]
(%i3) lsquares_estimates (M, [z, x, y], (z + D)^2 = A*x + B*y + C, [A, B, C, D]);
                  59        27      10921        107
(%o3)     [[A = - --, B = - --, C = -----, D = - ---]]
                  16        16      1024         32
```

Другой пример (точное решение отсутствует, отыскивается приближенное):

```
(%i1) load (lsquares)$
(%i2) M : matrix ([1, 1], [2, 7/4], [3, 11/4], [4, 13/4]);
                            [ 1  1  ]
                            [       ]
                            [    7  ]
                            [ 2  -  ]
                            [    4  ]
                            [       ]
(%o2)                       [    11 ]
                            [ 3  -- ]
                            [    4  ]
                            [       ]
                            [    13 ]
                            [ 4  -- ]
                            [    4  ]
(%i3) lsquares_estimates (M, [x, y], y = a*x^b + c, [a, b, c], initial = [3, 3, 3], iprint = [-1, 0]
(%o3) [[a = 1.387365874920637, b = .7110956639593767, c = - .4142705622439105]]
```

Для расчёта невязок для уравнения e при подстановке в него данных, содержащихся в матрице D, можно использовать функцию lsquares_residuals (D, x, e, a) (смысл параметров тот же, что ид ля функциии lsquares_estimates). Пимер использования функции lsquares_estimates и lsquares_residuals (те же данные, что использованы для расчёта параметров простой линейной регрессии):

```
(%i1) load (lsquares)$(%i2) s:[[125,140.7], [130,155.1], [135,160.3], [140,167.2], [145,169.8]];
```

(%o2) $[[125, 140.7], [130, 155.1], [135, 160.3], [140, 167.2], [145, 169.8]]$

```
(%i3) D:apply(matrix,s);
```

(%o3)
$$\begin{pmatrix} 125 & 140.7 \\ 130 & 155.1 \\ 135 & 160.3 \\ 140 & 167.2 \\ 145 & 169.8 \end{pmatrix}$$

```
(%i4) a : lsquares_estimates (D, [y,x], y = A + B*x, [A, B]);
```

(%o4)
$$[[A = \frac{8231525}{267474}, B = \frac{87875}{133737}]]$$

```
(%i5) float(%);
```

(%o5)
$$[[A = 30.77504729431646, B = 0.65707321085414]]$$

```
(%i6) lsquares_residuals (D, [y,x], y = A + B*x, first(a));
```

(%o6)
$[1.774751938506171, -2.687102297793416, -1.103882994234965, -0.63768814912851, 2.653921502650718]$

Остальные функции, входящие в состав пакеты lsquares, по синтаксису использования и идее реализации аналогичны приведенным (см. документацию разработчика).

7.3 Моделирование динамических систем

Многие модели, основанные на нелинейных дифференциальных уравнениях, демонстрируют совершенно удивительные свойства, причем решение большинства из них можно получить лишь численно.

Модели, основанные на задачах Коши для ОДУ, часто называют динамическими системами, подчеркивая, что, как правило, они содержат производные по времени t и описывают динамику некоторых параметров. Проблемы, связанные с динамическими системами, на самом деле весьма разнообразны и зачастую не сводятся к простому интегрированию ОДУ.

7.3.1 Моделирование системы химических реакций

Рассмотрим другой пример. Исследуем систему из трех дифференциальных уравнений, описывающих модель химической кинетики:

$A \to B$

$B \to C$

Система соответствующих дифференциальных уравнений

$\frac{\mathrm{dc}_A}{\mathrm{dt}} = -k_1 c_A$

$\frac{\mathrm{dc}_B}{\mathrm{dt}} = k_1 c_A - k_2 c_B$

$\frac{\mathrm{dc}_C}{\mathrm{dt}} = k_2 c_B$

Начальные условия:

$c_A = 1, c_B = 0, c_C = 0$

Результаты решения приведены на рис. 7.6.

```
(%i1) eq1:'diff(ca(t),t)=-k1*ca(t); eq2:'diff(cb(t),t)=k1*ca(t)-k2*cb(t); eq3:'diff(cc(t),t)=k2*cb(t
```

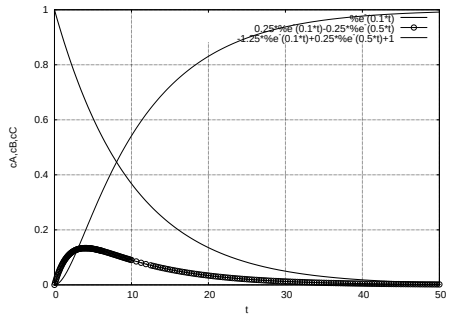


Рис. 7.6. Кинетика химических реакций

(%o3) $$\frac{d}{d\,t}\,ca\,(t) = -k1\,ca\,(t)\;\frac{d}{d\,t}\,cb\,(t) = k1\,ca\,(t) - k2\,cb\,(t)\;\frac{d}{d\,t}\,cc\,(t) = k2\,cb\,(t)$$

```
(%i4) atvalue(ca(t),t=0,1);
```

(%o4) $$1$$

```
(%i5) atvalue(cb(t),t=0,0); atvalue(cc(t),t=0,0);
```

(%o6) $$00$$

```
(%i7) sol:desolve([eq1,eq2,eq3],[ca(t),cb(t),cc(t)]);
```

(%o7) $$[ca\,(t) = e^{-k1\,t}, cb\,(t) = \frac{k1\,e^{-k1\,t}}{k2-k1} - \frac{k1\,e^{-k2\,t}}{k2-k1}, cc\,(t) = \frac{k1\,e^{-k2\,t}}{k2-k1} - \frac{k2\,e^{-k1\,t}}{k2-k1} + 1]$$

```
(%i8) ratsimp(sol);
```

(%o8)
$$[ca\,(t) = e^{-k1\,t}, cb\,(t) = \frac{\left(k1\,e^{k2\,t} - k1\,e^{k1\,t}\right)\,e^{-k2\,t-k1\,t}}{k2-k1}, cc\,(t) = \frac{\left(\left((k2-k1)\,e^{k1\,t} - k2\right)\,e^{k2\,t} + k1\,e^{k1\,t}\right)\,e^{-k2\,t-k1\,t}}{k2-k1}]$$

```
(%i9) k1:0.1; k2:0.5; ev((sol));
```

(%o11) $$0.10.5[ca\,(t) = e^{-0.1\,t}, cb\,(t) = 0.25\,e^{-0.1\,t} - 0.25\,e^{-0.5\,t}, cc\,(t) = -1.25\,e^{-0.1\,t} + 0.25\,e^{-0.5\,t} + 1]$$

```
(%i12) plot2d([%e^(-0.1*t),0.25*%e^(-0.1*t)-0.25*%e^(-0.5*t),-1.25*%e^(-0.1*t)+0.25*%e^(
 -0.5*t)+1],[t,0,50],[gnuplot_preamble,"set grid"],[gnuplot_term,"png size 500,500"],[gnuplot_out_fi
```

Несколько более сложная задача - моделирование кинетики параллельно-последовательных реакций, протекающих по схеме: $A \to B$

$B + C \to A$

$B \to C$

В зависимости от констант скорости химических реакций данная сиситема может быть довольно жёсткой.

Пример командного файла для решения жёсткой системы ОДУ в Maxima (данная система нелинейна, поэтому используем метод Рунге-Кутта, однако расчёты затрудняются жёсткостью системы):

```
load("dynamics");
load("draw");
k1:0.1; k2:100; k3:10;
eq1:-k1*ca+k3*cb*cc;
eq2:k1*ca-k3*cb*cc-k2*cb;
eq3:k2*cb;
t_range:[t,0,100,0.01];
sol: rk([eq1,eq2,eq3],[ca,cb,cc],[1,0,0],t_range)$
len:length(sol);
t:makelist(sol[k][1],k,1,len)$
ca:makelist(sol[k][2],k,1,len)$
cb:makelist(sol[k][3],k,1,len)$
cc:makelist(sol[k][4],k,1,len)$
draw2d(title="Chemical system",xlabel="ca",ylabel="cb",grid=true,points_joined = true,
points(t,ca),points(t,cb), points(t,cc),terminal=eps);
```

Данная система достаточно трудно решается при помощи функции rk. Увеличение констант до k2=1000 и k3=100 делает задачу практически неразрешимой средствами пакета dynamics.

Простейшим классическим примером существования автоколебаний в системе химических реакций является тримолекулярная модель «Брюсселятор», предложенная в Брюсселе Пригожиным и Лефевром (1965). Основной целью при изучении этой модели было установление качественных типов поведения, совместимых с фундаментальными законами химической и биологической кинетики. В этом смысле блюсселятор играет роль базовой модели, такую же как гармонический осциллятор в физике, или модели Вольтерра в динамике популяций. Во 2-й части лекций мы остановимся на пространственно-временных свойствах распределенной системы, локальным элементом которой является брюсселятор. Здесь мы рассмотрим свойства брюсселятора как автоколебательной системы.

Описание модели брюсселятора в Maxima приведено в следующем командном файле:

```
load("dynamics");
load("draw");
B:0.5;
eq1:-(B+1)*y0+y0^2*y1+1;
eq2:B*y0-y0^2+1;
t_range:[t,0,10,0.1];
sol: rk([eq1,eq2],[y0,y1],[1,1],t_range)$
len:length(sol);
t:makelist(sol[k][1],k,1,len)$
y0:makelist(sol[k][2],k,1,len)$
y1:makelist(sol[k][3],k,1,len)$
draw2d(title="Brusselator",xlabel="t",ylabel="y0,y1",
grid=true,points_joined = true,
points(t,y0),points(t,y1),terminal=eps);
```

Графическая иллюстрация (автоколебательный режим в системе) - на рис. 7.7 , а фазовые портреты - на рис.7.7 и рис. 7.7.

При проведении расчётов следует обратить внимание на жёсткость системы ОДУ, описывающей брюсселятор. в частности, при B=2.5 для потроения приведённой иллюстрации необходимо было уменьшить шаг по времени до 0.002. При очередном запуске командного файла, содержащего команды загрузки пакетов, рекомендуется перезапустить Maxima.

7.3.2 Фазовые портреты динамических систем

Для изучения динамических систем центральным моментом является анализ фазовых портретов, т. е. решений, получающихся при выборе всевозможных начальных условий.

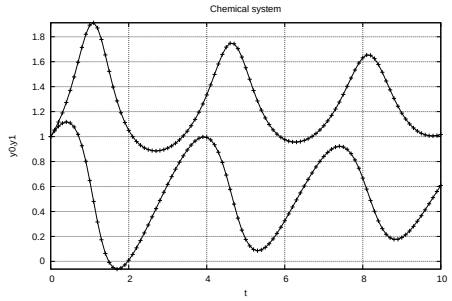


Рис. 7.7. Изменение концентраций при моделировании автоколебательной химической реакции (брюсселятора)

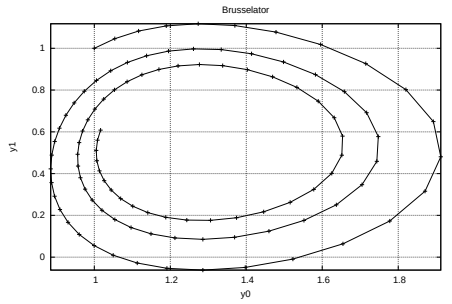


Рис. 7.8. Фазовый портрет для брюсселятора (B=0.5)

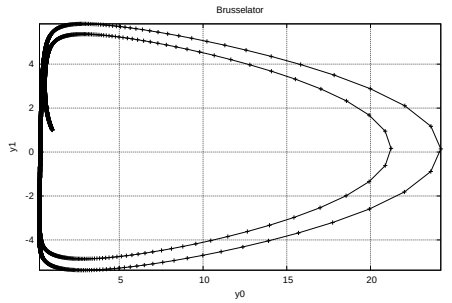


Рис. 7.9. Фазовый портрет для брюсселятора (B=2.5)

Решение ОДУ часто удобнее изображать не в виде графика y0 (t), y1(t), ..., а в фазовом пространстве, по каждой из осей которого откладываются значения каждой из найденных функций. При таком построении графика аргумент t будет присутствовать на нем лишь параметрически.

Как правило, решение задач Коши для ОДУ и их систем — задача хорошо разработанная и с вычислительной точки зрения довольно простая. На практике чаще встречаются другие, более сложные задачи, в частности, исследование поведения динамической системы в зависимости от начальных условий. При этом в большинстве случаев бывает необходимым изучить только асимптотическое решение ОДУ, т.е. y(t->∞), называемое аттрактором. Очень наглядным образом можно визуализировать такую информацию на фазовой плоскости, во многом благодаря тому, что существует всего несколько типов аттракторов, и для них можно построить четкую классификацию.

С одной стороны, каждое решение будет выходить из точки, координаты которой являются начальными условиями, но, оказывается, для большинства ОДУ целые семейства траекторий будут заканчиваться в одних и тех же аттракторах (стационарных точках или предельных циклах). Множество решений, вычисленное для всевозможных начальных условий, образует фазовый портрет

динамической системы. С вычислительной точки зрения задача исследования фазового портрета часто сводится к обычному сканированию семейств решений ОДУ при разных начальных условиях.

Дальнейшее усложнение задач анализа фазовых портретов связано с их зависимостью от параметров, входящих в систему ОДУ. В частности, при плавном изменении параметра модели может меняться расположение аттракторов на фазовой плоскости, а также могут возникать новые аттракторы и прекращать свое существование старые. В первом случае, при отсутствии особенностей, будет происходить простое перемещение аттракторов по фазовой плоскости (без изменения их типов и количества), а во втором — фазовый портрет динамической системы будет коренным образом перестраиваться. Критическое сочетание параметров, при которых фазовый портрет системы качественно меняется, называется в теории динамических систем точкой бифуркации.

Рассмотрим несколько наиболее известных классических примеров динамических систем, имея в виду. Это модели динамики популяций (Лотка-Вольтерры), генератора автоколебаний (Ван дер Поля), турбулентной конвекции (Лоренца) и химической реакции с диффузией (Пригожина). Для изучения динамических систем разработана специальная теория, центральным моментом которой является анализ фазовых портретов, т. е. решений, получающихся при выборе всевозможных начальных условий.

7.3.3 Модель динамики популяций

Модель взаимодействия "хищник—жертва"независимо предложили в 1925— 1927 гг. Лотка и Вольтерра. Два дифференциальных уравнения моделируют временную динамику численности двух биологических популяций жертв x и хищников y. Предполагается, что жертвы размножаются с постоянной скоростью, а их численность убывает вследствие поедания хищниками. Хищники же размножаются со скоростью, пропорциональной количеству пищи, и умирают естественным образом.

Модель была создана для биологических систем, но с определенными корректурами применима к конкуренции фирм, строительству финансовых пирамид, росту народонаселения, экологической проблематике и др.

Эта модель Вольтерра-Лотка с логистической поправкой описывается системой уравнений
с условиями заданной численности "жертв"и "хищников"в начальный момент t=0.

Решая эту задачу при различных значениях, получаем различные фазовые портреты (обычный колебательный процесс и постепенная гибель популяций). Результаты приведены на рисунке.

```
(%i1) a:4; b:2.5; c:2; d:1; alpha=0;
```

(%o5) $$42.521alpha = 0$$

```
(%i6) eq1:'diff(y(t),t)=(a-b*z(t))*y(t)-alpha*y(t)^2; eq2:'diff(z(t),t)=(-c+d*y(t))*z(t)
 -alpha*y(t)^2;
```

(%o7) $$\frac{d}{dt}\,y\left(t\right) = y\left(t\right)\,\left(4 - 2.5\,z\left(t\right)\right) - alpha\,y\left(t\right)^2 \frac{d}{dt}\,z\left(t\right) = \left(y\left(t\right) - 2\right)\,z\left(t\right) - alpha\,y\left(t\right)^2$$

```
(%i8) atvalue(y(t),t=0,3); atvalue(z(t),t=0,1);
```

(%o9) 31

```
(%i10) desolve([eq1,eq2],[y(t),z(t)]);
```

(%o10)

$$'rat'replaced 2.5 by 5/2 = 2.5[y\left(t\right) = ilt\left(-\frac{5\,laplace\left(y\left(t\right)\,z\left(t\right),t,lvar\right) + 2\,alpha\,laplace\left(y\left(t\right)^2,t,lvar\right) - 6}{2\,lvar - 8}, lvar, t\right), z\left(t\right)$$

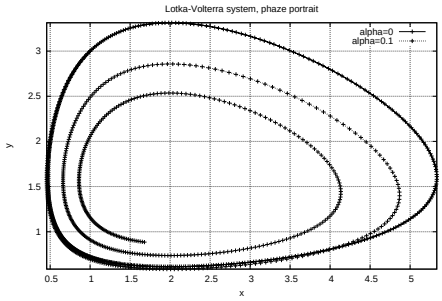


Рис. 7.10. Фазовый портрет для системы Лотка-Вольтерра

```
(%i11)
```

Очевидная проблема - неразрешимость данной системы в явном виде методом преобразования Лапласа, т.к. она нелинейна.

Используем численный метод Рунге-Кутта из пакета dynamics.

Результаты решения для значений alpha=0 и alpha=0.1 представлены на рис. 7.10, 7.11, 7.12. .

Рассмотрим командный файл для задачи моделирования системы Лотка-Вольтерра в Maxima:

```
a:4; b:2.5; c:2; d:1; alpha1:0;
ode1:(a-b*x)*y-alpha1*y^2$ ode2:(-c+d*y)*x$
alpha2:0.1;
ode3:(a-b*x)*y-alpha2*y^2$ ode4:(-c+d*y)*x$
load("dynamics");
t1:[]$ xg1:[]$ yg1:[]$ t2:[]$ xg2:[]$ yg2:[]$
l1:rk([ode1,ode2],[y,x],[1,3],[t,0,6,0.01])$
l2:rk([ode3,ode4],[y,x],[1,3],[t,0,6,0.01])$
for i:1 thru length(l1) do(t1:append(t1,[l1[i][1]]), xg1:append(xg1,[l1[i][2]]), yg1:append(yg1,[l1[
for i:1 thru length(l2) do(t2:append(t2,[l2[i][1]]), xg2:append(xg2,[l2[i][2]]), yg2:append(yg2,[l2[
load("draw");
draw2d(terminal='eps, file_name="lotka1",grid=true,xlabel = "x",
      ylabel = "y", title="Lotka-Volterra system, phaze portrait",
      key= "alpha=0",points_joined = true, color = black, points(xg1,yg1),
      points_joined = true, line_type=dots, color = black,
      key="alpha=0.1", points(xg2,yg2))$
draw2d(terminal='eps, file_name="lotka2",grid=true,xlabel = "t",
       ylabel = "x,y",  title="Lotka-Volterra system, alpha=0",
       key= "x(t)",points_joined = true, color = black, points(t1,xg1),
       points_joined = true, color = black, key= "y(t)", points(t1,yg1))$
draw2d(terminal='eps, file_name="lotka3",grid=true,xlabel = "t",
       ylabel = "x,y",  title="Lotka-Volterra system, alpha=0.1",
       key= "x(t)",points_joined = true, color = black, points(t2,xg2),
       points_joined = true, color = black, key= "y(t)", points(t2,yg2))$
```

Дифференциальные уравнения формируются символьными выражениями, определяющими правые части. Порядок следования выражений для расчёта правых частей ОДУ в первом списке функции rk должен соответствовать друг другу. Следует обратить внимание, что результат выполнения функции rk - двухуровневый список (каждый элемент списков l1 и l2 - также список, содержащий значение независимой переменной, и соответствующие значения искомых функций), поэтому оп преобразуется в векторы, испльзуемые для построения графических иллюстраций.

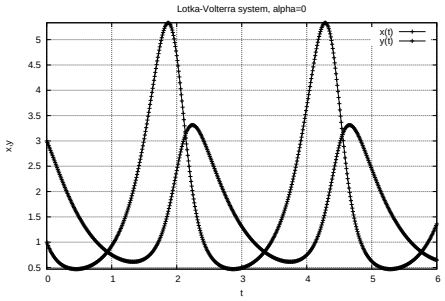


Рис. 7.11. Решения системы Лотка-Вольтерра в зависимости от времени (параметр alpha равен 0)

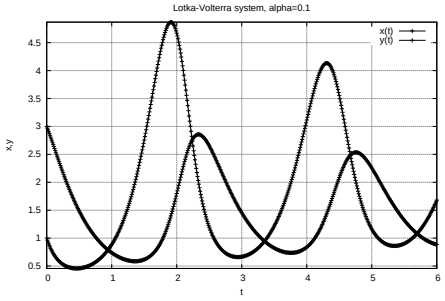


Рис. 7.12. Решения системы Лотка-Вольтерра в зависимости от времени (параметр alpha равен 0,1)

7.3.4 Движение твердого тела

Рассмотрим пример построения трехмерного фазового портрета. Находим решение задачи Эйлера свободного движения твердого тела:

$$\frac{dx_1}{dt} = x_2 x_3$$

,

$$\frac{dx_2}{dt} = -x_1 x_3$$

,

$$\frac{dx_3}{dt} = -0.51 x_1 x_2$$

.

```
(%i1) eq1:'diff(x1(t),t)=x2(t)*x3(t); eq2:'diff(x2(t),t)=-x1(t)*x3(t); eq3:'diff(x3(t),t)=
 -0.51*x1(t)*x2(t);
```

$$(\%o3) \qquad \frac{d}{d\,t}\,x1\,(t) = x2\,(t)\;x3\,(t)\;\frac{d}{d\,t}\,x2\,(t) = -x1\,(t)\;x3\,(t)\;\frac{d}{d\,t}\,x3\,(t) = -0.51\,x1\,(t)\;x2\,(t)$$

```
(%i4) load("dynamics")$ l: rk([y*z, -x*z,0.51*x*y],[x,y,z],[1,2,3],[t,0,4,0.1])$
```

Фазовый портрет для данной динамической системы (трехмерная кривая) представлен на рис.7.13.

7.3.5 Аттрактор Лоренца

Одна из самых знаменитых динамических систем предложена в 1963 г. Лоренцем в качестве упрощенной модели конвективных турбулентных движений жидкости в нагреваемом сосуде тороидальной формы. Система состоит из трех ОДУ и имеет три параметра модели. Задаём правые части уравнений модели Лоренца:

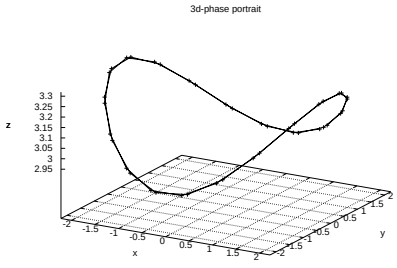


Рис. 7.13. Фазовый портрет трехмерной динамической системы

```
(%i2) eq: [s*(y-x), x*(r-z) -y, x*y - b*z];
```

(%o2) $$[s\,(y-x)\,,x\,(r-z)-y,x\,y-b\,z]$$

Задаём временные параметры решения

```
(%i3) t_range: [t,0,50,0.05];
```

(%o3) $$[t,0,50,0.05]$$

Задаём параметры системы

```
(%i4) s: 10.0; r: 28.0; b: 2.6667;
```

(%o6) 10.028.02.6667

Задаём начальные значения x,y,z

```
(%i7) init: [1.0,0,0];
```

(%o7) $$[1.0,0,0]$$

Выполняем решение

```
(%i8) sol: rk(eq, [x,y,z],init,t_range)$\\
(%i9) len:length(sol);
```

(%o9) 1001

Выдкляем компоненты решения и строим графические иллюстрации

```
(%i10) t:makelist(sol[k][1],k,1,len)$\\
(%i11) x:makelist(sol[k][2],k,1,len)$\\
(%i12) y:makelist(sol[k][3],k,1,len)$
(%i13) z:makelist(sol[k][4],k,1,len)$
(%i14) plot2d([discrete,t,x]);
```

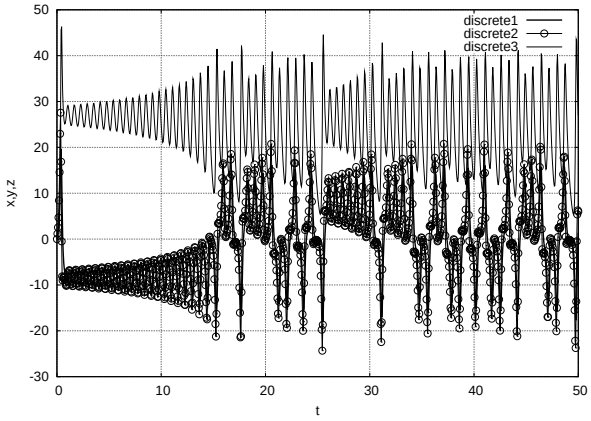


Рис. 7.14. Пример формирования динамического хаоса (аттрактор Лоренца)

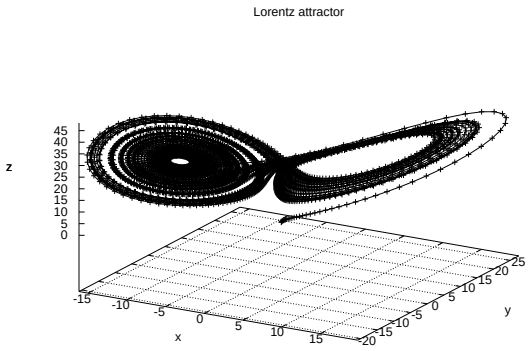


Рис. 7.15. Трехмарный фазовый портрет (аттрактор Лоренца)

(%o14)

```
(%i15) plot2d([discrete,t,y]);
```

Результаты решения (хаотические колебания x,y,z) представлен на рис. 7.14 и 7.15(фазовый портрет системы). На рисунках объединены в одних осях кривые x(t), y(t), z(t).

Решением системы Лоренца при определенном сочетании параметров является странный аттрактор (или аттрактор Лоренца) — притягивающее множество траекторий на фазовом пространстве, которое по виду идентично случайному процессу. В некотором смысле аттрактор Лоренца является стохастическими автоколебаниями, которые поддерживаются в динамической системе за счет внешнего источника.

Решение в виде странного аттрактора появляется только при некоторых сочетаниях параметров. Перестройка типа фазового портрета происходит в области промежуточных г. Критическое сочетание параметров, при которых фазовый портрет системы качественно меняется, называется в теории динамических систем точкой бифуркации. Физический смысл бифуркации в модели Лоренца, согласно современным представлениям, описывает переход ламинарного движения жидкости к турбулентному.

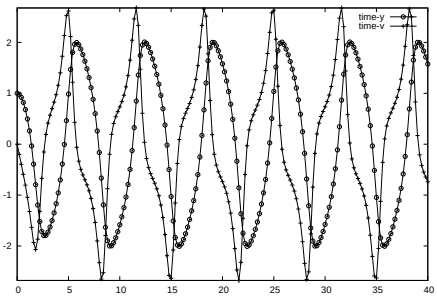


Рис. 7.16. Решение уравнения Ван дер Поля

7.3.6 Модель автоколебательной системы: уравнение Ван дер Поля

Рассмотрим решение уравнения Ван дер Поля, описывающего электрические колебания в замкнутом контуре, состоящем из соединенных последовательно конденсатора, индуктивности, нелинейного сопротивления и элементов, обеспечивающих подкачку энергии извне. Неизвестная функция времени y(t) имеет смысл электрического тока, а в параметре ц заложены количественные соотношения между составляющими электрической цепи, в том числе и нелинейной компонентой сопротивления:

$$\frac{d^2y(t}{dx^2} - \mu(1 - y(t)^2)\frac{dy(t)}{dt} + y(t) = 0$$

Решением уравнения Ван дер Поля являются колебания, вид которых для μ=1 показан на рис. 7.16 . Они называются автоколебаниями и принципиально отличаются от рассмотренных ранее (например, численности популяций в модели Вольтерра) тем, что их характеристики (амплитуда, частота, спектр) не зависят от начальных условий, а определяются исключительно свойствами самой динамической системы. Через некоторое время расчетов после выхода из начальной точки решение выходит на один и тот же цикл колебаний, называемый предельным циклом. Аттрактор типа предельного цикла является замкнутой кривой на фазовой плоскости. К нему асимптотически притягиваются все окрестные траектории, выходящие из различных начальных точек, как изнутри (рис. 7.17), так и снаружи предельного цикла.

Использованный командный файл Maxima (для построения графической иллюстрации использован пакет draw):

```
load("dynamics");
mu:1;
s:rk ([v,mu*(1-y^2)*v-y],[y,v],[1,0],[t,0,40,0.2])$
time:makelist(s[k][1],k,1,length(s))$
y:makelist(s[k][2],k,1,length(s))$
v:makelist(s[k][3],k,1,length(s))$
load("draw");
draw2d(points_joined = true, point_type=6,
       key= "y-v", xlabel="y",ylabel="v",
       points(y,v), terminal = eps)$
```

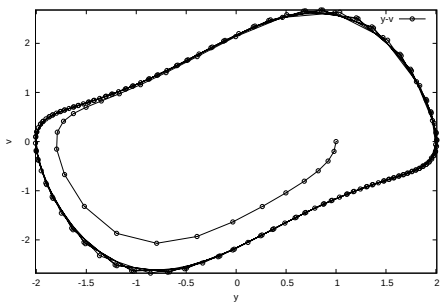


Рис. 7.17. Фазовый портрет уравнения Ван дер Поля

Глава 8

Решение физических и математических задач с Maxima

Доступная литература и сеть Интернет в качестве "электронного помощника"студентов и школьников обычно позиционирует пакет MathCad, изредка - Maple или Mathematica. Материал данной главы содержит ряд разнородных задач, которые решались разными авторами вручную или при помощи MathCad

8.1 Операции с полиномами и рациональными функциями

Рассмотрим решение с помощью Maxima нескольких задач из классического сборника под редакцией М.И. Сканави. В Maxima "пошаговое"упрощение выражений с поледовательным использованием стандартного набора примитивов (формул суммы или разности кубов, формул возведения суммы или разности в степень и т.п.) выполнить сложно, поэтому результат являтся фактически справочным, на который следует ориентироваться при решении вручную, при помощи ручки и бумаги.

8.1.1 Упрощение алгебраических выражений

Пример 1. Упростить выражение и вычислить его, если даны числовые значения параметров:

```
(%i12) g:(1/a-1/(b+c))/(1/a+1/(b+c))*(1+(b^2+c^2-a^2)/2/b/c)/((a-b-c)/a/b/c);
```

(%o12)
$$\frac{a\,b\,c\left(\frac{1}{a}-\frac{1}{c+b}\right)\left(\frac{c^2+b^2-a^2}{2\,b\,c}+1\right)}{(-c-b+a)\left(\frac{1}{c+b}+\frac{1}{a}\right)}$$

```
(%i13) ratsimp(%);
```

(%o13)
$$-\frac{a\,c+a\,b-a^2}{2}$$

```
(%i14) %,a=0.02,b=-11.05,c=1.07;
```

(%o14) 0.1

Пример 2. Упростить выражение и вычислить его, если даны числовые значения параметров:

```
(%i1) (sqrt(x)+1)/(x*sqrt(x)+x+sqrt(x))/(1/(x^2-sqrt(x)));
```

(%o1) $$\frac{\left(\sqrt{x}+1\right)\,\left(x^2-\sqrt{x}\right)}{x^{\frac{3}{2}}+x+\sqrt{x}}$$

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

(%o2) $$x-1$$

Пример 3. Сделать указанную подстановку и результат упростить:

```
(%i3) expr:(x^3-a^(-2/3)*b^(-1)*(a^2+b^2)*x+b^(1/2))/(b^(3/2)*x^2);
```

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

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

(%o4) $$\frac{a^{\frac{2}{3}}\,b\,x^3+\left(-b^2-a^2\right)\,x+a^{\frac{2}{3}}\,b^{\frac{3}{2}}}{a^{\frac{2}{3}}\,b^{\frac{5}{2}}\,x^2}$$

```
(%i5) radcan(%);
```

(%o5) $$\frac{a^{\frac{2}{3}}\,b\,x^3+\left(-b^2-a^2\right)\,x+a^{\frac{2}{3}}\,b^{\frac{3}{2}}}{a^{\frac{2}{3}}\,b^{\frac{5}{2}}\,x^2}$$

Без указанной подстановки упрощение посредством комбинации функций ratsimp и radcan не удаётся.

```
(%i6) %,x=a^(2/3)*b^(-1/2);
```

(%o6) $$\frac{\frac{a^{\frac{2}{3}}\,\left(-b^2-a^2\right)}{\sqrt{b}}+a^{\frac{2}{3}}\,b^{\frac{3}{2}}+\frac{a^{\frac{8}{3}}}{\sqrt{b}}}{a^2\,b^{\frac{3}{2}}}$$

```
(%i7) ratsimp(%);
```

Конечный результат оказывается простым

(%o7) $$0$$

8.1.2 Разложение полиномов и рациональных выражений на множители

8.1.3 Решение алгебраических уравнений

Maxima (как и любой другой пакет символьной математики) не всегда способен получить окончательное решение. Однако полученный результат может оказаться всё же проще, чем исходная задача.

Пример (также из сборника под ред. М.И. Сканави): Решить уравнение $\sqrt{(}x-2)=x-4$:

```
(%i1) solve([sqrt(x-2)=x-4],[x]);
```

(%o1) $$[x = \sqrt{x-2} + 4]$$

Уравнение имеет одно решение: $X = 6$, однако для отыскания его с помощью Maxima придётся прибегнуть к замене исходного уравнения его следствием:

```
(%i3) solve([(x-2)=(x-4)^2],[x]);
```

(%o3) $$[x = 6, x = 3]$$

Решения для дальнейшего использования можно извлечь из списка функцией ev:

```
(%i1) sol:solve([x-2=(x-4)^2],[x]);
```

(%o1) $$[x = 6, x = 3]$$

```
(%i2) ev(x,sol[1]);
```

(%o2) $$6$$

```
(%i3) ev(x,sol[2]);
```

(%o3) $$3$$

Ещё два примера решения алгебраических уравнений:

```
(%i1) eq:7*(x+1/x)-2*(x^2+1/x^2)=9;
```

(%o1) $$7\left(x+\frac{1}{x}\right)-2\left(x^2+\frac{1}{x^2}\right)=9$$

```
(%i2) sol:solve([eq],[x]);
```

(%o2) $$[x = 2, x = \frac{1}{2}, x = -\frac{\sqrt{3}\,i-1}{2}, x = \frac{\sqrt{3}\,i+1}{2}]$$

```
(%i3) x1:ev(x,sol[1]); x2:ev(x,sol[2]);

(комплексные корни не рассматриваем)
```

(%o4) $$2\frac{1}{2}$$

Уравнения с радикалами перед решением в Maxima приходится преобразовывать к степенной форме (для выделения левой и правой части выражения используют функции lhs и rhs соответственно):

```
(%i1) eq:sqrt(x+1)+sqrt(4*x+13)=sqrt(3*x+12);
```

(%o1) $$\sqrt{4\,x+13}+\sqrt{x+1}=\sqrt{3\,x+12}$$

```
(%i2) eq1:lhs(eq)^2=rhs(eq)^2;
```

(%o2) $$\left(\sqrt{4\,x+13}+\sqrt{x+1}\right)^2=3\,x+12$$

```
(%i3) solve([eq1],[x]);
```

(%o3) $$[x=-\sqrt{x+1}\,\sqrt{4\,x+13}-1]$$

```
(%i4) eq2:x+1=rhs(%[1])+1;
```

(%o4) $$x+1=-\sqrt{x+1}\,\sqrt{4\,x+13}$$

```
(%i5) eq3:lhs(eq2)^2=rhs(eq2)^2;
```

(%o5) $$(x+1)^2=(x+1)\,(4\,x+13)$$

Последняя команда позволила получить степенное уравнение, разрешимое аналитически в Maxima (для этого потребовалось дважды возвести в квадрат исзодное уравнение).

```
(%i6) solve([eq3],[x]);
```

(%o6) $$[x=-4,x=-1]$$

Проверку решения выполняем при помощи функции ev.
Решение $x=-4$ не удовлетворяет исходному уравнению.

```
(%i7) ev(eq,%[1]);
```

(%o7) $$2\,\sqrt{3}\,i=0$$

Решение $x=-1$ превращает мсходное уравнение в верное равенство:

```
(%i8) ev(eq,%o6[2]);
```

(%o8) $$3=3$$

Рассмотрим ещё один пример, иллюстрирующий замену и подстановку при решении алгебраических уравнений:

```
(%i1) eq:sqrt(x+3-4*sqrt(x-1))+sqrt(x+8-6*sqrt(x-1))=1;
```

Исходное уравнение:

(%o1) $$\sqrt{x-4\,\sqrt{x-1}+3}+\sqrt{x-6\,\sqrt{x-1}+8}=1$$

Выполним замену $\sqrt{x+1}=z$, $z=x^2+1$:

```
(%i2) eq1:subst(z,sqrt(x-1),eq);
```

(%o2) $$\sqrt{-4\,z+x+3}+\sqrt{-6\,z+x+8}=1$$

```
(%i3) eq2:subst(z^2+1,x,eq1);
```

(%o3) $$\sqrt{z^2-4\,z+4}+\sqrt{z^2-6\,z+9}=1$$

Упрощаем полученный результат:

```
(%i4) radcan(%);
```

(%o4) $$2\,z-5=1$$

```
(%i5) solve([%],z);
```

(%o5) $$[z=3]$$

```
(%i6) solve([sqrt(x-1)=3],[x]);
```

(%o6) $$[x=10]$$

```
Выполним проверку
(%i7) ev(eq,%[1]);
```

(%o7) $$1=1$$

Значительная часть тригонометрических уравнений школьного курса также разрешимы в Maxima, но непосредственное решение удаётся получить далеко не всегда.

Примеры:

```
(%i1) solve([sin(%pi/6-x)=sqrt(3)/2],[x]);
```

(%o1) $$`solve'\,is\,using\,arc-trig\,functions\,to\,get\,a\,solution.\,Some\,solutions\,will\,be\,lost.[x=-\frac{\pi}{6}]$$

Большинство тригонометрических уравнений в Maxima (кроме простейших) приходится решать приведением их к алгебраическим.

Логарифмические и показательные уравнения также решаются в Maxima путём замены переменных и сведения к алгебраическим (см. выше специфические функции для упрощения логарифмических выражений).

8.2 Некоторые физические задачи

Применение систем символьной математики в преподавании физики и химии позволяет сосредоточиться на содержательной части преподаваемого материала. Кроме того, учащиеся получают возможность решать куда более сложные задачи, чем при ручных расчётах. Наличие в Maxima чётко выраженного алгоритмического языка (в отличие от Matcad) существенно снижает риск подмены понятий, когда пробелы собственного подхода к решению задачи учащиеся относят на наличие ошибок и неточностей программного обеспечения. Идеи рассмотренных задач взяты из известных руководств по использованию MathCad, однако, по мнению автора, использование Maxima может быть не менее, а во многих случаях и более эффективным.

8.2.1 Вычисление средней квадратичной скорости молекул

Выражение, содержащие переменные, по существу может использоваться в качестве функции пользователя в Maxima.

Рассмотрим возможность вычисления среднеквадратичной скорости молекул для различных газов. спользуемая формула:

$$v = \sqrt{\frac{3RT}{M}}$$

, где R = 8.314 Дж/(моль $\cdots$ K), T - абсолютная температура, M - молярная масса.

Вычислим среднеквадратичную скорость молекул CO_2 (M=0.044 кг/моль) при температуре 273 K:

```
(%i1) v:sqrt(3*R*T/M);
```

(%o1) $$\sqrt{3}\sqrt{\frac{RT}{M}}$$

```
(%i2) vCO2:float(v),M=0.044,T=273,R=8.314;
```

(%o2) 393.3875604633079

Расчёт для нескольких различных газов несложно провести, варьируя молярную массу:

```
(%i3) vVozd:float(v),M=0.029,T=273,R=8.314;
```

(%o3) 484.5604478145187

```
(%i4) vH2:float(v),M=0.002,T=273,R=8.314;
```

(%o4) 1845.151213315592

8.2.2 Распределение Максвелла

Аналогично предыдущему расчёту создадим выражение, описывающее распределение Максвелла (см. блок команд Maxima ниже).

```
(%i1) fun:4*%pi*(M/2/%pi/R/T)^(3/2)*exp(-M*v^2/2/R/T)*v^2;
```

(%o1) $$\frac{2\,v^2\left(\frac{M}{RT}\right)^{\frac{3}{2}}e^{-\frac{v^2\,M}{2\,R\,T}}}{\sqrt{2}\,\sqrt{\pi}}$$

Для анализа полученных выражений в формулу распределения Максвелла пожставляем только температуру:

```
(%i2) fun70:fun,T=70;
```

(%o2) $$\frac{v^2\left(\frac{M}{R}\right)^{\frac{3}{2}}e^{-\frac{v^2\,M}{140\,R}}}{35\,\sqrt{2}\,\sqrt{70}\,\sqrt{\pi}}$$

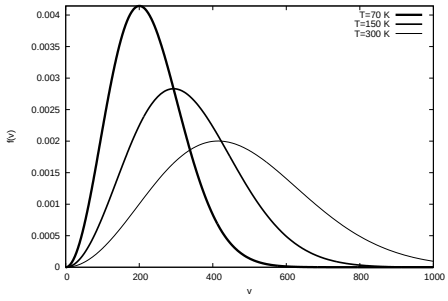


Рис. 8.1. Распределение Максвелла по скоростям молекул воздуха для различных температур

```
(%i3) fun150:fun,T=150;
```

(%o3)
$$\frac{v^2\left(\frac{M}{R}\right)^{\frac{3}{2}}e^{-\frac{v^2\,M}{300\,R}}}{375\,\sqrt{2}\,\sqrt{6}\,\sqrt{\pi}}$$

```
(%i4) fun300:fun,T=300;
```

(%o4)
$$\frac{v^2\left(\frac{M}{R}\right)^{\frac{3}{2}}e^{-\frac{v^2\,M}{600\,R}}}{1500\,\sqrt{2}\,\sqrt{3}\,\sqrt{\pi}}$$

Для построения графика зависимости функции распределения от температуры подставляем молярную массу воздуха и величину универсальной газовой постоянной (см. результаты на рис. 8.1):

```
r
(%i5) plot2d([fun70,fun150,fun300],[v,0,1000]),M=0.029,R=8.314;
```

Можно изучить влияние температуры на форму кривой, а также на положение максимума функции распределения. С помощью интегрирования f(v) можно посчитать долю молекул, обладающих скоростями в каком-либо интервале, а также определить среднюю и среднюю квадратичную скорости молекул. Пример:

```
(%i6) integrate(v*v*fun,v,0,inf);
```

$$IsM\ RTpositive, negative, orzero?$$

```
p;
```

(%o6)
$$\frac{3\,R\,T}{M}$$

Таким образом, $\langle v^2\rangle = \frac{3RT}{M}$, откуда среднеквадратичная скорость молекул газа $\langle\sqrt{v^2}\rangle = v_{sq} = \sqrt{frac3RTM}$.

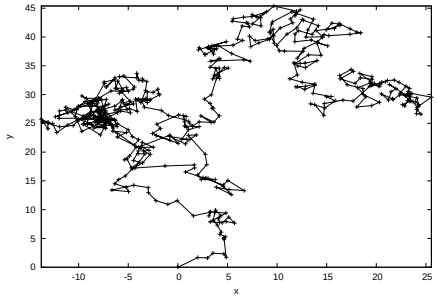


Рис. 8.2. Траектория броуновского движения модельной частицы

8.2.3 Броуновское движение

Наличие генератора случайных чисел дает возможность моделировать движение броуновской частицы.

Эйнштейн первый рассчитал параметры броуновского движения, показав, что нерегулярное перемещение частиц, взвешенных в жидкости, вызвано случайными ударами соседних молекул, совершающих тепловое движение. В соответствии с теорией Смолуховского-Эйнштейна, среднее значение квадрата смещения броуновской частицы (s^2) за время t прямо пропорционально температуре T и обратно пропорционально вязкости жидкости h, размеру частицы r и постоянной Авогадро N_A:

$$s^2 = \frac{2RTt}{6\pi hrN_A},$$

где R — газовая постоянная. Броуновские частицы имеют размер порядка 0,1–1 мкм, т.е. от одной тысячной до одной десятитысячной доли миллиметра.

Построим несколько упрощённую модель броуновского движения, предполагая, что смещение частицы по каждой из координат - нормально распределённая случайная величина с нулевым математическим ожиданием. Для генерации случайных чисел используем пакет distrib, включающий необходимые функции (использован генератор random_normal).

```
(%i1) load("distrib");
```

(%o1) */usr/share/maxima/5.17.1/share/contrib/distrib/distrib.mac*

```
(%i2) x:0$ y:0$ xy:[[0,0]]$ m:0$ s:1$
Nmax:500$ for i:1 thru Nmax do (x:x+random_normal(m,s),
y:y+random_normal(m,s), xy:append(xy,[[x,y]]));
```

(%o8) *done*

```
(%i9) plot2d([discrete,xy]);
```

Результат построения графика приведен на рис. 8.2.

8.3 Пример построения статистической модели

Рассмотрим построение задачи с практическим содержанием.

В таблице 8.1 приведены данные (взяты из статьи В.Ф. Очкова - http://twt.mpei.ac.ru/ochkov/) о зависимости цены подержанного автомобиля от его пробега и "возраста"(времени использования). В статье-первоисточнике задача исследования этой зависимости решалась средствами MathCad.

Рассмотрим её решение средствами Maxima.

Возраст (лет)	Пробег (миль)	Цена ($)	Возраст (лет)	Пробег (миль)	Цена ($)
11.5	88000	1195	13.5	103000	750
10.5	82000	1295	10.5	65000	1495
12.5	97000	800	10.5	70000	1495
8.5	51000	2295	10.5	80000	1495
9.5	79000	1995	6.5	57000	2695
13.5	120000	495	11.5	101000	895
3.5	39000	4995	10.5	78000	1295
6.5	52000	2695	9.5	84000	1995
4.5	39000	3995	4.5	46000	3675
12.5	92000	795	11.5	108000	975
7.5	41000	3495	13.5	124000	850
10.5	77000	1595	6.5	56000	3495
12.5	83000	895	9.5	67000	2495
4.5	38000	3990	6.5	43000	3400
13.5	92000	795	11.5	78000	1295

Таблица 8.1. Стоимость подержанного автомобиля в зависимости от его возраста и пробега

В дальнейшем предполагается, что исходные данные для решения подготовлены в виде файла cars.txt. Для считывания используем пакет numericalio. В памяти данные представляются матрицей, а для построения отдельных графиков - списками (переменные age, mile,price - см. ниже).

```
(%i1) load("draw");
```

(%o1) */usr/share/maxima/*5.13.0*/share/draw/draw.lisp*

```
(%i2) load("numericalio");
```

(%o2) */usr/share/maxima/*5.13.0*/share/contrib/numericalio/numericalio.lisp*

```
(%i3) data:read_matrix("cars1.txt")$
(%i4) age:makelist(data[k,1], k, 1, 30)$
(%i5) mile:makelist(data[k,2], k, 1, 30)$
(%i6) price:makelist(data[k,3], k, 1, 30)$
```

Простейшую линейную регрессию можно построить, используя функцию simple_linear_regression (пакет stats). Построим зависимость цены автомобиля от его стоимости и пробега:

```
(%i21) xy:makelist([age[k],price[k]], k, 1, 30)$(%i22) simple_linear_regression(xy);
```

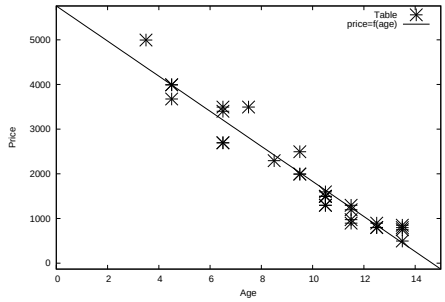


Рис. 8.3. Зависимость цены подержанного автомобиля от его возраста

(%o22)
$$\left(\begin{array}{c} SIMPLELINEARREGRESSION \\ model = 5757.594446543255 - 392.7181715149224\,x \\ correlation = -0.96881779424672 \\ p\_value = 0.0 \end{array}\right)$$

Фактический вывод результатов несколько сокращён.

Построим аналогичную зависимость цены автомобиля от пробега, но не в линейной, а в экспоненциальной форме:

```
(%i26) xy:makelist([mile[k],log(price[k])], k, 1, 30)$(%i27) simple_linear_regression(xy);
```

(%o27)
$$\left(\begin{array}{c} SIMPLELINEARREGRESSION \\ model = 9.174960600286802 - 2.3747715120748162\,10^{-5}\,x \\ correlation = -0.93011255642444 \\ p\_value = 5.9285909514983359\,10^{-14} \end{array}\right)$$

Полученные зависимости представлены в виде выражений Maxima:

```
(%i28) fun1:5757.6-392.7*x$(%i29) exp(9.175);
```

(%o29) 9652.768071616591

```
(%i30) fun2:9653*exp(-2.375*10^(-5)*x);
```

(%o30) $9653\,e^{-2.3750000000000001\,10^{-5}\,x}$

Проиллюстрируем полученные результаты графически (рис. 8.3 и рис. 8.4):

```
(%i34) draw2d(terminal=eps,key="Table",xlabel="Age",ylabel="Price",point_size = 3,point_type=3,point
```

```
(%i41) draw2d(terminal=eps,key="Table",xlabel="Mile",ylabel="Price",point_size = 3,point_type=3,poin
```

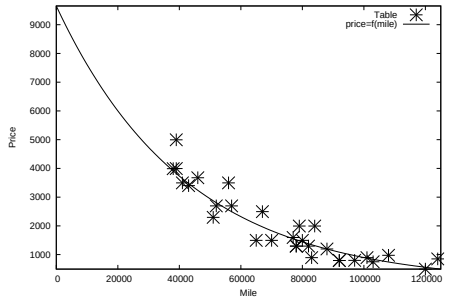


Рис. 8.4. Зависимость цены подержанного автомобиля от его пробега

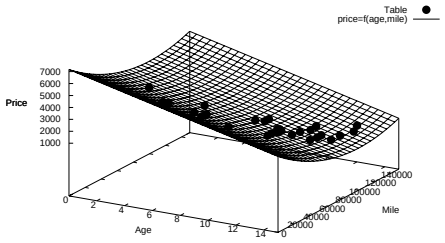


Рис. 8.5. Иллюстрация зависимости отклика (цены подержанного автомобиля) от двух независимых факторов (возраста и пробега автомобиля)

Для построения модели в виде зависимости цены автомобиля от пробега и возраста одновременно целесообразно использовать более сложную функцию lsquares_estimates (пакет lsquares). Искомая модель была представлена уравнением:

$$Price = a + b * Age + c * Mile + d * Mile^2)$$

Необходимые команды Maxima:

```
(%i5) lsquares_estimates (data, [x,y,z], z=a+b*x+c*y+d*y^2, [a,b,c,d]);
```

(%o5)
$$[[a = \frac{36712000090549571}{5117101479342}, b = -\frac{80056614985946}{284283415519}, c = -\frac{194393701258481}{3411400986228000}, d = \frac{2937180994967}{10234202958684000000}]]$$

```
(%i6) float(%);
```

(%o6)
$$[[a = 7174.37405506961, b = -281.6084604857839, c = -0.056983539033745, d = 2.8699655525931528\,10^{-7}]]$$

Следует отметить, что сильно нелинейные задачи решаются при помощи lsquares_estimate медленно, поэтому результаты построения модели сильно зависят от обоснованности постановки задачи оценивания. ллюстрация Графическая иллюстрация представлена на рис.

Приложения

Таблица 1: **Сокращённый список основных функций Maxima**

Функция или переменная	**Краткое описание**
;, $, , %	простейшие команды , см. стр. 17
addcol	Функция добавляет столбец к матрице , см. стр. 30-35
addrow	Функция добавляет строку к матрице , см. стр. 30-35
algsys	Функция решает полиномиальные системы уравнений. Допускаются системы из одного уравнения с одной неизвестной. Кроме того, допускаются недоопределенные системы , см. стр. 63-67
allroots	Функция, которая находит и печатает все (в том числе и комплексные) корни полиномиального уравнения с действительными либо комплексными коэффициентами , см. стр. 63-67
antidiff	Функция выполняет интегрирование выражений с произвольными функциями, перед ее первым вызовом следует загрузить пакет "antid см. стр. 98-102
append	Функция позволяет склеивать два списка , см. стр. 22
arrayinfo	Функция печатает информацию о массиве — его вид, число индексов, размер , см. стр. 26
arrays	Переменная содержит список имен массивов первого и второго видов, определенных на данный момент , см. стр. 29
array	Функция определяет массив с данным именем, определенным количеством индексов и заданным размером , см. стр. 26
assume	Функция вводит информацию о переменной в базу данных , см. стр. 36-40

atom	Функция возвращает "true если аргумент не имеет структуры, т.е. составных частей (например, число или переменная не имеют структуры) , см. стр. **??**
atvalue	Функция позволяет задать значение функции и ее производных при некоторых значениях аргументов , см. стр. 126
at	Функция вычисляет значение выражения в заданной точке с учетом свойства , см. стр. 126
augmented_lagrangian_method	Функция осуществляет минимизацию ФНП с ограничениями, см. стр. 161
batch	Функция запускает файл с программой. Операторы выполняются один за другим либо до конца файла, либо до синтаксической ошибки, либо до некорректной операции, см. стр. 156
bc2	Функция позволяет учесть краевые условия в решениях дифференциальных уравнений второго порядка , см. стр. 116
cabs	Функция возвращает модуль комплексного выражения, см. стр. 56-58
carg	Функция возвращает фазу комплексного выражения, см. стр. 56-58
cfdisrep	Функция преобразует список (как правило результат выполнения функции cf) в собственно цепную дробь , см. стр. 114
cf	Функция Создает цепную дробь, аппроксимирующую данное выражение. Выражение должно состоять из целых чисел, квадратных корней целых чисел и знаков арифметических операций. Возвращаемый результат - список , см. стр. 114
cfdirep	Функция преобразует список в собственно цепную дробь, см. стр. 114
changevar	реализует замену переменных в интеграле , см. стр. 98-102
charpoly	Функция является до некоторой степени избыточной — она вычисляет характеристический полином матрицы (корни этого полинома — собственные значения матрицы), см. стр. 58- 63
closefile	Функция прекращает вывод в файл , см. стр. 154
COl	Функция выделяет заданный столбец матрицы , см. стр. 30-35
combine	Функция объединяет слагаемые с идентичным знаменателем , см. стр. 36-40

compile	Функция сначала транслирует функцию Maxima на язык LISP, а затем компилирует эту функцию LISP'а до двоичных кодов и загружает их в память , см. стр. 152
conjugate	Функция для вычисления комплексно-сопряжённых выражений , см. стр. 56-58
cons	Функция позволяет добавлять элемент в начало списка , см. стр. 23
contrib_ode	Функция решает дифференциальные уравнения (больше возможностей, чем у ode2), см. стр. 128-131
copylist	Функция создаёт копию списка , см. стр. 21
create_list	Функция создаёт список , см. стр. 22
copymatrix	Функция Создаёт копию матрицы , см. стр. 30-35
cspline	Функция строит сплайн-интерполяцию , см. стр. 159
define	Функция позволяет преобразовать выражение в функцию , см. стр. 46
demoivre	Функция заменяет все экспоненты с мнимыми показателями на соответствующие тригонометрические функции , см. стр. 56 - 58
denom	Функция выделяет знаменатель , см. стр. 40
depends	Функция позволяет декларировать, что переменная зависит от одной или нескольких других переменных , см. стр. 94
desolve	Функция решает дифференциальные уравнения и системы дифференциальных уравнений методом преобразования Лапласа, см. стр. 127
determinant	Функция вычисляет детерминант матрицы, см. стр. 58- 63
diff	Функция выполняет дифференцирование , см. стр. 79
display2d	Переменная включает или выключает "двумерное"рисование дробей, степеней, и т.п. Изначально установлено значение "true см. стр. 153
display	Функция печатает значения своих аргументов вместе с их именем, каждое в отдельной строке, см. стр. 153
disp	Функция печатает значения своих аргументов, причем каждое значение печатается в отдельной строке? см. стр. 153

divide	Функция позволяет вычислить частное и остаток от деления одного многочлена на другой, см. стр. 36 - 40
draw2d	строит двумерные графики, см. стр. 170-172
draw3d	строит трёхмерные графики , см. стр. 170-172
echelon	Функция преобразует матрицу к верхней треугольной, см. стр. 58- 63
eigenvalues	Функция аналитически вычисляет собственные значения матрицы, см. стр. 58- 63
eigenvectors	Функция аналитически вычисляет собственные значения и собственные вектора матрицы, если это возможно , см. стр. 58- 63
eliminate	Функция исключает из системы уравнений указанные переменные. Оставшиеся уравнения приводятся к виду с нулевой правой частью, которая опускается , см. стр. 67
endcons	Функция позволяет добавлять элемент в конец списка, см. стр. 23
ev	Функция является основной функцией, обрабатывающей выражения , см. стр. см. стр. 36-40
expand	Функция раскрывает скобки, см. стр. 36-40
exponentialize	Функция приводит комплексное выражение к экспоненнциальной форме , см. стр. 56-58
express	Функция преобразует дифференциальные операторы в выражения, см. стр. 95
factor	Функция представляет в виде произведения некоторых сомножителей заданное выражение, см. стр. 36 - 40
factorsum	Функция факторизует отдельные слагаемые в выражении , м. стр. 36 - 40
fillarray	Функция позволяет заполнять одноиндексные массивы третьего вида из списка , см. стр. 29
find_root	Функция находит корень уравнения на заданном интервале методом деления отрезка пополам , см. стр. 156
first	Функция выделяет первый элемент списка , см. стр. 23
float	Функция конвертирует любые числа в выражениях в числа машинной точности , см. стр. 18
fourier	Функция позволяет вычислить коэффициенты ряда Фурье см. стр. **??**
foursimp	Функция позволяет упростить коэффициенты ряда Фурье см. стр. **??**

fullratsimp	Функция вызывает функцию "ratsimp"до тех пор, пока выражение не перестанет меняться , 41-43
genmatrix	Функция возвращает матрицу заданной размерности, составленную из элементов индексного массива , см. стр. 30-35
gfactorsum	Функция представляет в виде сомножителей слагаемые выражения с комплексными числами, см. стр. 36-40
gfactor	Функция представляет в виде сомножителей выражение с комплексными числами, см. стр. 36-40
gradef	Функция определяет результат дифференцирования функции по своим аргументам , см. стр. 94
gramschmidt	Функция вычисляет ортонормированную систему векторов, см. стр. 58-63
ic1	Функция позволяет учесть начальное условие в решениях дифференциальных уравнений первого порядка , см. стр. 116
ic2	Функция позволяет учесть начальные условия в решениях дифференциальных уравнений второго порядка , см. стр. 116
ident	Функция возвращает единичную матрицу заданной размерности , см. стр. 30-35
ilt	Функция реализует обратное преобразование Лапласа , см. стр. 102-103
imagpart	Функция возвращает действительную часть выражения , см. стр. 56-58
integrate	Функция выполняет интегрирование заданного выражения по указанной переменной (неопределенная константа не добавляется). Можно также указать пределы интегрирования — в этом случае вычисляется определенный интеграл , см. стр. 98-102
invert	функция выполняет обращение матрицы , см. стр. 30 - 35
join	функция выполняет компоновку списков , см. стр. 23
kill	Функция уничтожает всю информацию (как свойства, так и присвоенное значение) об объекте или нескольких объектах , см. стр. 19
lagrange	Функция строит интерполяцию полиномом Лагранжа, см. стр. 158

lambda	создает лямбда-выражение (безымянную функцию). Лямбда-выражение может использоваться в некоторых случаях как обычная функция, см. стр. **??**
laplace	Функция реализует прямое преобразование Лапласа , см. стр. 102-103
last	Функция выделяет последний элемент списка , см. стр. 23
lbfgs	Функция осуществляет минимизацию ФНП, см. стр. 160
ldisplay	Функция печатает значения своих аргументов вместе с их именем и метками "%t см. стр. 153
ldisp	Функция печатает значения своих аргументов вместе с метками "%t см. стр. 153
length	Функция возвращает длину списка , см. стр. 21
lhs	Функция выделяет левую часть уравнения, см. стр. 63-67
limit	функция осуществляет вычисление пределов , см. стр. 70
linearinterpol	Функция строит линейную интерполяцию , см. стр. 158
linsolve	Функция решает системы линейных и полиномиальных уравнений. Допускаются недоопределенные системы, см. стр. 63-67
listarray	Функция печатает содержимое массивов первого и второго видов , см. стр. 26
load	Функция загружает тот или иной файл : load(somefile); Тип загрузки зависит от типа файла (макрос Maxima, программа на Lisp, бинарный файл) , см. стр. 155
logcontract	Функция компактифицирует логарифмы в данном выражении , см. стр. 45-45
make_array	Функция оздает массивы третьего вида, содержимое которых печатается автоматически , см. стр. 29
makelist	Функция позволяет создавать списки , см. стр. 21
map	Функция применяет заданную функцию к каждому элементу списка , см. стр. 25
matrix	Функция возвращает матрицу, заданную поэлементно , см. стр. 30-35
matrixmap	Функция для заполнения матрицы значениями некоторой функции , см. стр. 30-35

mattrace	Функция вычисляет след матрицы (сумму ее диагональных элементов) , см. стр. 30-35
max	перебирает свои аргументы и находит максимальное число , см. стр. 25
member	Функция возвращает "true если ее первый аргумент является элементом заданного списка, и "false"в противном случае , см. стр. 23
min	перебирает свои аргументы и находит минимальное число , см. стр. 25
minor	вычисляет миноры матрицы, см. стр. 58-63
mnewton	Функция находит корень системы уравнений многомерным методом Ньютона. Для использования функции необходимо сначала загрузить пакет "mnewton см. стр. 157
multthru	Функция умножает каждое слагаемое в сумме на множитель, причем при умножении скобки в выражении не раскрываются , см. стр. 36-40
newton	Функция находит корень указанной функции методом Ньютона , см. стр. 156
nroots	Функция, которая возвращает количество действительных корней полиномиального уравнения с действительными коэффициентами, которые локализованы в указанном интервале, см. стр. 63-67
num	Функция выделяет числитель , см. стр. 40
ode2	Функция решает дифференциальные уравнения первого и второго порядков, см. стр. 115-124
odelin	Функция решает однородные линейные уравнения первого и второго порядка, и возвращает фундаментальное решение ОДУ см. стр. 128-131
pade	Функция аппроксимирует отрезок ряда Тейлора дробно-рациональной функцией , см. стр. 113
part	Функция позволяет выделить тот или иной элемент часть списка , см. стр. 23
plog	представляет основную ветвь комплексного логарифма , см. стр. 56-58
plot2d, wxplot2d	строит двумерные графики , см. стр. 48-51
plot3d, wxplot3d	строит трёхмерные графики , см. стр. 48-51
polarform	Функция приводит комплексное выражение к тригонометрической форме , см. стр. 56-58
polyfactor	Переменная определяет форму выдачи функции "allroots см. стр. 63-67
powerseries	Функция строит разложение в степенной ряд, см. стр. 107

print	печатает значения всех своих аргументов в одну строку , см. стр. 153
product	Функция реализует цикл умножения , см. стр. ??
properties	Функция печатает свойства переменной, см. стр. 98-102
radcan	Функция упрощает выражения со вложенными степенями и логарифмами , см. стр. 45- 45
ratepsilon	Переменная задает точность преобразования действительного числа в рациональное , см. стр. 41
ratexpand	Функция раскрывает скобки в выражении. Отличается от функции "expand"тем, что приводит выражение к канонической форме , см. стр. 41-43
ratfac	Переменная включает или выключает частичную факторизацию выражений при сведении их к CRE. Изначально установлено значение "false см. стр. 41-43
ratsimpexpons	Переменная управляет упрощением показателей степени в выражениях , см. стр. 41-43
ratsimp	Функция приводит все куски (в том числе аргументы функций) выражения, которое не является дробно-рациональной функцией, к каноническому представлению, производя упрощения, которые не делает функция "rat". Повторный вызов функции может изменить результат, т.е. упрощение не идет до конца , см. стр. 41-43
ratsubst	Функция Реализует синтаксическую подстановку для рациональных выражений , ссм. стр. 41-43
ratvars	Функция позволяет изменить алфавитный порядок "гдавности"переменных, принятый по умолчанию , см. стр. 41-43
rat	Функция приводит выражение к каноническому представлению и снабжает его меткой "/R/". Она упрощает любое выражение, рассматривая его как дробно-рациональную функцию, т.е. работает с арифметическими операциями и с возведением в целую степень , см. стр. 41-43
realpart	Функция возвращает действительную часть комплексного выражения , см. стр. 56-58
read	основная функция для считывания вводимых пользователем выражений, см. стр. 153
read_matrix, read_list	функция для ввода массивов чисел, см. стр. 176
realroots	Функция выдает действительные корни полиномиального уравнения с действительными коэффициентами , см. стр. 63-67

rectform	Функция Приводит комплексное выражение к алгебраической форме , см. стр. 56-58
remarray	Функция уничтожает массив или массивы , см. стр. 29
remove	Функция удаляет свойство переменной , см. стр. 98-102
residue	Функция позволяет вычислять вычеты на комплексной плоскости, см. стр. 56-58
rest	Функция выделяет остаток после удаления первого элемента списка , см. стр. ??
reverse	Функция меняет порядок элементов в списке на обратный , см. стр. 23
rhs	Функция выделяет правую часть уравнения, см. стр. 63-67
romberg	Функция численно находит определенный интеграл функции на заданном отрезке. При этом используется алгоритм Ромберга, см. стр. 161
rk	Функция реализует метод Рунге-Кутта решения ОДУ, см. стр. 131
row	Функция выделяет заданную строку матрицы , см. стр. 30-35
save	сохраняет текущие значения рабочей области в файл, см. стр. 154
solve	Функция решает уравнения и системы уравнений , см. стр. 47
sort	Функция упорядочивает элементы списка , см. стр. 23
sublist	Функция составляет список из тех элементов исходного списка, для которых заданная логическая функция возвращает значение , см. стр. ??
submatrix	Функция выделяет из матрицы подматрицу , 30-35
subst	Функция Реализует синтаксическую подстановку , см. стр. 40
Sum	Функция реализует цикл суммирования , см. стр. 24
taylor	Функция Возвращает разложение функции в ряд Тейлора , см. стр. 106
tlimit	Функция отличается от функции "limit"только алгоритмом — она использует разложение выражения в ряд Тейлора , см. стр. 109
totalfourier	Функция позволяет вычислить построить ряд Фурье см. стр. ??

translate	Функция транслирует функцию Maxima на язык LISP , см. стр. 152
transpose	Функция ранспонирует матрицу , см. стр. 58- 63
trigexpand	Переменная управляет работой функции "trigexpand см. стр. 43 - 43
trigexpand	Функция раскладывает все тригонометрические функции от сумм в суммы произведений тригонометрических функций , см. стр. 43 - 44
trigreduce	Функция свертывает все произведения тригонометрических функций в тригонометрические функции от сумм, см. стр. 43-44
trigsimp	Функция только применяет к выражению правило sin2(x) + cos2 (x) = 1 , см. стр. 43-44
trirat	Функция пытается свести выражение с тригонометрическими функциями к некому универсальному каноническому виду (в общем, пытается упростить выражение) , см. стр. 43 - 44
uniteigenvectors	Функция отличается от функции "eigenvectors"тем, что возвращает нормированные на единицу собственные вектора, см. стр. 58- 63
writefile	Функция начинает запись выходных данных Maxima в указанный файл , см. стр. 155
write_matrix, write_list	функция для вывода массивов чисел, см. стр. 176
xthru	ФуНКЦИЯ приводит выражение к общему знаменателю, не раскрывая скобок и не факторизовать слагаемые , ссм. стр. 36-40
zeromatrix	Функция возвращает матрицу заданной размерности, составленную из нулей , см. стр. 30
“	Две одиночные кавычки "a вызывают дополнительное вычисление в момент обработки a , см. стр. 17
‘	Одиночная кавычка ’ предотвращает вычисление , см. стр. 17

Таблица 2: **Перечень основных пакетов расширения Maxima**

Наименование пакета	**Краткое описание функций пакета**
augmented_lagrangian	Минимизация функции нескольких переменных с ограничениями методом неопределённых множителей Лагранже (используется совместно с lbfgs)

bode	Построение диаграмм Боде (узкоспециальный пакет)
contrib_ode	Дополнительные функции для аналитического решения обыкновенных дифференциальных уравнений
descriptive	Описательная статистика, оценка параметров распределения (генеральной совокупности) по выборке (см. стр. 182-**??**)
. diag	Пакет для операций с некоторыми видами диагональных матриц
distrib	Пакет, содержащий функции для расчёта различных распределений вероятностей и их параметров (нормальное распределение, распределение Стьюдента и т.п.)
draw	Интерфейс Maxima-Gnuplot. Предназначен для подготовки иллюстраций полирафического качества
Dynamics	Различные функции, в т.ч. графические, относящиеся к моделированию динамических систем и фракталов
f90	Экспорт кода Maxima в кон на Фортран90
ggf	Пакет включает единственную функцию, позволяющую оперировать с произволящими функциями последовательностей (узкоспециальный пакет)
graphs	Пакет, включающий функции для работы с графами
grobner	Функции для того, чтобы работать с базисо Грёбнера (Groebner)
Impdiff	вычисление производных неявных функций нескольких переменных
implicit_plot	Графики неявных функций
interpol	Пакет, включающий функции интерполяции (линейной, полиномами Лагранжа, сплайнами)
lapack	Функции пакета LAPACK для решения задач линейной алгебры
Lbfgs	пакет минимизации функций нескольких переменных квазиньютоновским методом (L-BFGS)
lindstedt	Пакет, рассчитанный на интерпретацию некоторый типов начальных условий для ОДУ, описывающийх колебания
lsquares	Функции для оценки параметров различных зависимостей методом наименьших квадратов (см. стр. 191-193)

makeOrders	Пакет включает одну функцию для операций с полиномами
mnewton	Метод Ньютона для решения систем налинейных уравнений
numericalio	Чтение и запись файлов (преимущественно с матричными числовыми данными)
opsubst	Пакет содержит одну функцию opsubst, позволяющую выполнять замену в выражениях (по возможностям мало отличается от subst)
orthopoly	Пакет, содержащий функции для операций с ортогональными полиномами (Лежандра, Чебышева и др.)
plotdf	Пакет, позволяющий строить поле направлений для решения автономных систем (интересный, но довольно узкоспециальный пакет)
romberg	Пакет, включающий ряд функций для численного интегрирования
simplex	Пакет, предназначенный для решения задач линейного программирования
solve_rec	Пакет, содержащий функции для упрощения рекуррентных выражений
stats	Пакет, включающий функции для статистической проверки гипотез (о равенстве математических ожиданий или дисперсий выборок и т.п. - см. стр. 189-??)
stirling	Расчёт гамма-функции
stringproc	Пакет, включающий функции для обработки строк
unit	Пакет, включающий функции для операций с единицами измерения
zeilberger	Функции для гипергеометрического суммирования

Таблица 3. **Список основных математических констант, доступных в Maxima**

Обозначение в Maxima	**Математическое содержание**
%e	основание натуральных логарифмов
%i	мнимая единица ($\sqrt{(}-1)$)
inf	положительная бесконечность (на действительной оси)
minf	отрицательная бесконечность (на действительной оси)
infinite	бесконечность (на комплексной плоскости)
% phi	Золотое сечение (ϕ)
% pi	Постоянная π - отношение длтны окружности к её диаметру
%gamma	Постоянная Эйлера (γ)
false, true	логические (булевы) величины

Таблица 4. **Список основных математических функций, доступных в Maxima**

Обозначение в Maxima	**Математическое содержание**
sqrt	квадратный корень
sin	синус
cos	косинус
tan	тангенс
cot	котангенс
sec	секанс
csc	косеканс
asin	арксинус
acos	арккосинус
atan	арктангенс
acot	арккотангенс
asec	арксеканс
acsc	арккосеканс
exp	экспонента
log	натуральный логарифм
sinh	гиперболический синус
cosh	гиперболический косинус
tanh	гиперболический тангенс
asinh	обратный гиперболический синус
acosh	обратный гиперболический косинус
atanh	обратный гиперболический тангенс
floor	округление до целого с недостатком
ceiling	округление до целого с избытком
fix	целая часть
float	преобразование к формвту с плавающей точкой
abs	абсолютная величина

Литература

[1] Документация по текущей версии пакета: http://maxima.sourceforge.net/docs/manual/en/maxima.html

[2] В.А. Ильина, П.К. Силаев. система аналитических вычислений Maxima для физиков-теоретиков. М.:МГУ им. М.В. Ломоносова, 2007. - 113 с. http://tex.bog.msu.ru/numtask/max07.ps

[3] Статьи Тихона Тарнавского http://maxima.sourceforge.net/ru/maxima-tarnavsky-1.html

[4] http://www.pmtf.msiu.ru/chair31/students/spichkov/maxima2.pdf (Методическое пособие по изучению математического пакета Maxima) Математический практикум с применением пакета Maxima. (PDF)

[5] Н.А. Стахин. ОСНОВЫ РАБОТЫ С СИСТЕМОЙ АНАЛИТИЧЕСКИХ (СИМВОЛЬНЫХ) ВЫЧИСЛЕНИЙ MAXIMA (ПО для решения задач аналитических (символьных) вычислений).- Москва: Федеральное агентство по образованию, 2008 - 86 с.

[6] Книга по Maxima (электронное руководство) http://maxima.sourceforge.net/docs/maximabook/maxima 19-Sept-2004.pdf

[7] Книга Gilberto E. Urroz http://www.neng.usu.edu/cee/faculty/gurro/Maxima.html

[8] Аладьев В. З. Системы компьютерной алгебры: Maple: искусство программирования / В. З. Аладьев. — М.: Лаборатория базовых знаний, 2006. — 792 с.

[9] Васильев А. Н. Mathcad 13 на примерах / А. Н. Васильев. — СПб.: БХВ-Петербург, 2006. — 528 с.

[10] Воробъев Е. М. Введение в систему символьных, графических и численных вычислений Mathematica 5 / Е. М. Воробъев. — М.: ДИАЛОГ-МИФИ, 2005. — 368 с.

[11] Говорухин В. Н. Введение в Maple. Математический пакет для всех / В. Н. Говорухин, В. Г. Цибулин. — М.: Мир, 1997. — 208 с.

[12] Гурский Д. А. Вычисления в MathCAD / Д. А. Гурский. — Мн.: Новое знание, 2003. — 814 с.

[13] Гурский Д. Mathcad для студентов и школьников. Популярный самоучитель / Д. Гурский, Е. Турбина. — СПб.: Питер, 2005. — 400 с.

[14] Дьяконов В. П. Maple 9 в математике, физике и образовании / В. П. Дьяконов. — М.: СОЛОН-Пресс, 2004. — 688 с.

[15] Дьяконов В. П. Справочник по MATHCAD PLUS 7.0. PRO / В. П. Дьяконов. — М.: СК Пресс, 1998. — 352 с.

[16] Очков В. Ф. Mathcad 12 для студентов и инженеров / В. Ф. Очков. — СПб.: БХВ-Петербург, 2005. — 464 с.

[17] Плис А. И. MATHCAD 2000. Математический практикум для экономистов и инженеров / А. И. Плис, Н. А. Сливина. — М.: Финансы и статистика, 2000. — 656 с.

[18] Половко А. М. Derive для студента / А. М. Половко. — СПб.: БХВ-Петербург, 2005. — 352 с.

[19] Половко А. М. Mathcad для студента / А. М. Половко, И. В. Ганичев. — СПб.: БХВ-Петербург, 2006. — 336 с.

[20] Сдвижков О. А. Математика на компьютере: Maple 8 / О. А. Сдвижков. — М.: СОЛОН-Пресс, 2003. — 176 с.

[21] Новые информационные технологии: Учеб. пособие / Под ред. В.П. Дьяконова; Смол. гос. пед. ун-т. - Смоленск, 2003. - Ч. 3: Основы математики и математическое моделирование / В.П. Дьяконов, И.В. Абраменкова, А.А. Пеньков. - 192 с.: ил.

[22] Майер, Р.В. Решение физических задач с помощью пакета MathCAD [Электронный ресурс] / Р.В.Майер. — Глазов: ГГПИ, 2006. — 37 с. - http://maier-rv.glazov.net/math/math1.htm

Предметный указатель

Учебное издание

Серия «Библиотека ALT Linux»

Чичкарев Евгений Анатольевич

Компьютерная математика с Maxima: Руководство для школьников и студентов

Редактор серии: К. А. Маслинский
Редактор: В. М. Жуков
Оформление обложки: В. Меламед
Вёрстка:

Подписано в печать 00.00.08. Формат 70x100/16.
Гарнитура Computer Modern. Печать офсетная. Бумага офсетная.
Усл. печ. л. 00,0. Тираж 0000 экз. Заказ

ООО «Альт Линукс Технолоджи»
Адрес для переписки: 119334, Москва, 5-й Донской проезд, д. 21Б, стр. 21
Телефон: (495) 662-38-83. E-mail: sales@altlinux.ru
http://altlinux.ru