

Система аналитических вычислений MAXIMA и её использование в научных исследования

по материалам сайта
<http://www.intuit.ru/studies/curriculum/17584/courses/726/info>

Введение	7
1. Возникновение и развитие систем компьютерной математики	8
2. Основы Maxima	20
3. Задачи высшей математики с Maxima.....	68
4. Численные методы и программирование с Maxima	171
5. Обрамление Maxima	194
6. Моделирование с Maxima	212
7. Решение физических и математических задач с Maxima.....	251
8. Реализация некоторых численных методов	262
9. Справочный материал	296
Литература	309

Оглавление

Введение	7
1. Возникновение и развитие систем компьютерной математики	8
1.1 Определение систем компьютерной алгебры	8
1.1.1 Недостатки численных расчётов.....	9
1.1.2 Отличия символьных вычислений от численных	10
1.2 Классификация, структура и возможности систем компьютерной математики	12
1.2.1 Классификация систем компьютерной математики.....	12
1.2.2 Задачи систем компьютерной алгебры.....	13
1.2.3 Место компьютерной алгебры в информатике	13
1.2.4 Взаимосвязь систем компьютерной алгебры и традиционных математических дисциплин.....	14
1.2.5 Возможности повышения эффективности решения математических и вычислительных задач	15
1.3 Коммерческие и свободно распространяемые системы компьютерной математики.....	16
2. Основы Maxima	20
2.1 Структура Maxima.....	20
2.1.1 Области математики, поддерживаемые в Maxima	20
2.2 Достоинства программы	20
2.3 Установка и запуск программы	21
2.4 Интерфейс wxMaxima	21
2.5 Интерфейс командной строки. Ввод простейших команд Maxima.....	22
2.5.1 Обозначение команд и результатов вычислений	23
2.6 Числа, операторы и константы	24
2.6.1 Ввод числовой информации.....	24
2.6.2 Арифметические операции	24
2.6.3 Константы	25
2.7 Типы данных, переменные и функции	25
2.7.1 Списки.....	26
2.7.1.1 Функции для элементарных операций со списками.....	26
2.7.1.2 Функции, оперирующие с элементами списков.....	29
2.7.2 Массивы	30
2.7.3 Матрицы и простейшие операции с ними	34
2.7.4 Математические функции	38
2.7.5 Вычисление и преобразование аналитических выражений.....	39
2.7.6 Преобразование рациональных выражений	44
2.7.7 Преобразование тригонометрических выражений.....	47
2.7.8 Преобразование степенных и логарифмических выражений	48
2.7.9 Пользовательские функции.....	49
2.8 Решение задач элементарной математики.....	50
2.8.1 Нахождение корней уравнений и систем алгебраических уравнений.....	50
2.9 Построение графиков и поверхностей.....	51
2.9.1 Построение графика явной функции $y = f(x)$	52
2.9.2 Построение графиков функций, заданных параметрически.....	54
2.9.3 Построение кривых в полярной системе координат	56
2.9.4 Построение трёхмерных графиков	57
2.10 Draw графика	63
2.10.1 Сцены	63
2.10.2 Основные функции и примеры	63
3. Задачи высшей математики с Maxima.....	68

3.1	Операции с комплексными числами	68
3.1.1	Представление комплексных чисел	68
3.1.2	Функции для работы с комплексными числами.....	70
3.2	Задачи линейной алгебры	72
3.2.1	Простейшие операции с матрицами	72
3.2.2	Обращение матриц и вычисление определителей.....	73
3.2.3	Характеристический полином, собственные числа и собственные векторы матрицы	73
3.2.4	Ортогонализация	74
3.2.5	Преобразование матрицы к треугольной форме	75
3.2.6	Вычисление ранга и миноров матрицы	75
3.2.7	Решение матричных уравнений	76
3.2.8	Специальные функции для решения систем линейных и полиномиальных уравнений	77
3.3	Классификация и основные свойства функций	80
3.3.1	Основные свойства функций.....	81
3.3.2	Предел функции и его свойства.....	82
3.3.2.1	Предел функции в бесконечности.....	82
3.3.2.2	Предел функции в точке.....	82
3.3.2.3	Односторонние пределы.....	82
3.3.2.4	Теоремы о пределах.....	83
3.3.2.5	Вычисление пределов различных классов функций	84
3.3.2.6	Предел и непрерывность функции.....	84
3.3.2.7	Пределы рациональных дробей	84
3.3.2.8	Пределы, содержащие иррациональные выражения.....	85
3.3.2.9	Пределы тригонометрических выражений.....	85
3.3.2.10	Пределы экспоненциальных выражений	86
3.3.2.11	Бесконечно малые и бесконечно большие функции	88
3.3.2.12	Эквивалентные бесконечно малые. Их применение к вычислению пределов... ..	89
3.3.2.13	Бесконечно большие функции. Связь между бесконечно малыми и бесконечно большими величинами.....	90
3.3.3	Непрерывные функции.....	90
3.3.3.1	Непрерывность функции в точке	90
3.3.3.2	Свойства непрерывных функций	92
3.3.3.3	Точки разрыва функций и их классификация.....	93
3.3.4	Дифференцирование с помощью пакета Maxima.....	94
3.3.4.1	Вычисление производных и дифференциалов	95
3.4	Экстремумы функций	96
3.4.1	Отыскание максимумов и минимумов.....	96
3.4.1.1	Теорема Ферма.....	97
3.4.1.2	Необходимое условие экстремума.....	98
3.4.1.3	Первое достаточное условие экстремума	100
3.4.1.4	Второе достаточное условие экстремума	101
3.4.1.5	Схема исследования функции $y = f(x)$ на экстремум	102
3.4.1.6	Нахождение наибольших и наименьших значений функции	104
3.4.2	Выпуклость функции.....	105
3.4.2.1	Необходимые и достаточные условия выпуклости (вогнутости) функции	105
3.4.2.2	Точки перегиба	107
3.4.2.3	Исследование функции на выпуклость и наличие точек перегиба.....	107
3.4.2.4	Общая схема исследования функций и построения графика.....	108
3.4.2.5	Асимптоты графика функции	111
3.4.2.6	Свойства функций, непрерывных на отрезке. Теоремы Вейерштрасса	113

3.4.3 Дифференцирование функций нескольких переменных	114
3.5 Аналитическое и численное интегрирование	117
3.5.1 Основные команды	117
3.5.2 Интегралы, зависящие от параметра. Ограничения для параметров.....	118
3.5.3 Основные приёмы интегрирования	119
3.5.4 Преобразование Лапласа	121
3.6 Методы теории приближения в численном анализе	122
3.6.1 Приближённое вычисление математических функций.....	123
3.6.2 Приближённое вычисление определённых интегралов.....	130
3.7 Преобразование степенных рядов	132
3.8 Решение дифференциальных уравнений в Maxima	135
3.8.1 Основные определения.....	135
3.8.2 Функции для решения дифференциальных уравнений в Maxima	136
3.8.3 Решение основных типов дифференциальных уравнений.....	138
3.8.3.1 Уравнения с разделяющимися переменными.....	138
3.8.3.2 Однородные уравнения	138
3.8.3.3 Линейные уравнения первого порядка	139
3.8.3.4 Уравнения в полных дифференциалах	140
3.8.3.5 Уравнения Бернулли.....	141
3.8.3.6 Уравнения высших порядков	141
3.8.3.7 Уравнения с постоянными коэффициентами	142
3.8.3.8 Уравнения с переменными коэффициентами.....	143
3.8.3.9 Уравнение Эйлера.....	144
3.8.3.10 Граничные задачи	144
3.8.4 Операторный метод решения.....	144
3.8.5 Дополнительные возможности решения ОДУ	148
3.8.5.1 Пакет contrib_ode	148
3.8.5.2 Уравнения Клеро и Лагранжа	149
3.8.5.3 Другие задачи с использованием contrib_ode.....	150
3.8.5.4 Решение однородных линейных уравнений	151
3.8.6 Численные методы решения ОДУ	152
3.9 Ряды Фурье по ортогональным системам	153
3.9.1 Понятие ряда Фурье	153
3.9.2 Вычисление коэффициентов тригонометрических рядов Фурье	155
3.9.3 Ряды Фурье для чётных и нечётных функций.....	158
3.9.4 Разложение функций в ряд Фурье на отрезке $[0, \pi]$	161
3.9.5 Ряд Фурье для функций с периодом $2l$	164
3.9.6 Комплексная форма ряда Фурье	166
3.9.7 Дополнительные возможности: пакет fourie	168
3.9.8 Дополнительные возможности: обобщённые ряды Фурье	170
4. Численные методы и программирование с Maxima	171
4.1 Программирование на встроенном макроязыке.....	171
4.1.1 Условные операторы	171
4.1.2 Операторы цикла	172
4.1.3 Блоки	174
4.1.4 Функции	176
4.1.5 Транслятор и компилятор в Maxima	178
4.1.5.1 Функция translate	178
4.1.5.2 Функция compile	179
4.2 Ввод-вывод в пакете Maxima	181
4.2.1 Ввод-вывод данных в консоли.....	181
4.2.2 Файловые операции ввода-вывода.....	183

4.2.2.1 Ввод-вывод текстовых данных	183
4.2.2.2 Ввод-вывод командных файлов	184
4.3 Встроенные численные методы	185
4.3.1 Численные методы решения уравнений	185
4.3.1.1 Решение уравнений с одним неизвестным	185
4.3.2 Решение уравнений методом Ньютона: пакет newton1	185
4.3.2.1 Решение уравнений с несколькими неизвестными: пакет mnewton	186
4.3.3 Интерполяция	187
4.3.3.1 Линейная интерполяция	187
4.3.3.2 Интерполяция полиномами Лагранжа	188
4.3.3.3 Интерполяция сплайнами	188
4.3.4 Оптимизация с использованием пакета lbfgs	189
4.3.4.1 Оптимизация с ограничениями методом неопределённых множителей Лагранжа	192
4.3.5 Численное интегрирование: пакет romberg	192
5. Обрамление Maxima	194
5.1 Классические графические интерфейсы Maxima	194
5.1.1 Графический интерфейс wxMaxima	194
5.1.1.1 Рабочее окно wxMaxima	194
5.1.2 Графический интерфейс xMaxima	199
5.1.3 Использование редактора TeXmacs в качестве интерфейса Maxima	199
5.1.4 Работа с Maxima из Emacs	201
5.2 Работа с Maxima в KDE: интерфейс Cantor	204
5.2.1 Документ Cantor	204
5.2.2 Настройка	205
5.2.3 Прочие возможности Cantor	205
5.3 Интегрированная среда Sage	205
5.4 Построение графических иллюстраций: пакет draw	206
6. Моделирование с Maxima	212
6.1 Общие вопросы моделирования	212
6.1.1 Аналитические модели	213
6.1.2 Идентифицируемые модели	214
6.2 Статистические методы анализа данных	216
6.2.1 Ввод-вывод матричных данных	216
6.2.2 Функции Maxima для расчёта описательной статистики	217
6.2.3 Проверка статистических гипотез	226
6.2.4 Расчёт коэффициентов линейной регрессии	233
6.2.5 Использование метода наименьших квадратов	235
6.3 Моделирование динамических систем	238
6.3.1 Моделирование системы химических реакций	238
6.3.1.1 Пример командного файла для решения жёсткой системы	239
6.3.2 Фазовые портреты динамических систем	242
6.3.3 Модель динамики популяций	243
6.3.4 Движение твердого тела	247
6.3.5 Аттрактор Лоренца	248
6.3.6 Модель автоколебательной системы: уравнение Ван дер Поля	249
7. Решение физических и математических задач с Maxima	251
7.1 Операции с полиномами и рациональными функциями	251
7.1.1 Упрощение алгебраических выражений	251
7.1.2 Разложение полиномов и рациональных выражений на множители	252
7.1.2.1 Решение алгебраических уравнений	252
7.2 Некоторые физические задачи	254

7.2.1 Вычисление средней квадратичной скорости молекул.....	255
7.2.2 Распределение Максвелла	255
7.2.3 Броуновское движение	257
7.3 Пример построения статистической модели.....	258
8. Реализация некоторых численных методов	262
8.1 Программирование методов решения нелинейных уравнений в Maxima	262
8.1.1 Метод половинного деления	264
8.1.2 Метод простых итераций	265
8.1.3 Метод Ньютона (метод касательных).....	266
8.1.4 Модифицированный метод Ньютона (метод секущих)	267
8.1.5 Метод хорд.....	268
8.2 Численное интегрирование.....	268
8.2.1 Обзор методов интегрирования	269
8.2.2 Метод прямоугольников	269
8.2.3 Метод средних прямоугольников	270
8.2.4 Метод трапеций	271
8.2.5 Метод Симпсона.....	271
8.3 Методы решения систем линейных уравнений	273
8.3.1 Общая характеристика и классификация методов решения.....	273
8.3.2 Метод Гаусса	273
8.3.3 Метод квадратного корня.....	276
8.3.4 Корректность постановки задачи и понятие обусловленности	279
8.3.5 О вычислительных затратах.....	281
8.4 Итерационные методы	282
8.4.1 Матричная формулировка итерационных методов решения систем линейных уравнений	282
8.4.2 Метод простой итерации.....	284
8.4.3 Метод Зейделя	286
8.5 Решение обыкновенных дифференциальных уравнений	288
8.5.1 Методы решения задачи Коши	288
8.5.2 Метод рядов, не требующий вычисления производных правой части уравнения	291
8.5.2.1 Метод Эйлера.....	291
8.5.2.2 Метод Рунге-Кутты	293
9. Справочный материал	296
9.1 Сокращённый список основных функций Maxima.....	296
9.2 Перечень основных пакетов расширения Maxima.....	306
9.3 Список основных математических констант, доступных в Maxima	308
9.4 Список основных математических функций, доступных в Maxima	309
Литература	309

Введение

В настоящее время получили большое распространение специальные программные средства, позволяющие провести весь цикл разработки какой-либо математической модели: от поиска и просмотра необходимой литературы до непосредственного решения задачи (аналитического и/или численного) и подготовки отчёта или статьи к печати. К такому средству относится система аналитических вычислений **Maxima** (и вычислительная среда Octave). Эта программная среда (пакет) — хороший выбор для проведения любой учебной задачи или серьёзного 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.

В настоящее время компьютерные программы этого класса (проприетарные — **Maple, Mathematica, MATLAB, MathCad, Maxima** и др., или с открытым кодом **Maxima**) находят самое широкое применение в научных исследованиях, становятся одним из обязательных компонентов компьютерных технологий, используемых в образовании.

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

Для школьников системы компьютерной математики (**СКМ**) являются незаменимым помощником в изучении математики, физики, информатики, освобождая их от рутинных расчётов и сосредотачивая их внимание на сущности метода решения той или иной задачи. Применение **СКМ** позволяет решать целый спектр новых трудоёмких, 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, приобретённых 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, **СКМ** — хорошее средство создания графических иллюстраций и документов, содержащих

математические формулы и выкладки. В настоящее время для проведения расчётов по всевозможным техническим дисциплинам студентами широко используется пакет **MatCad**, в основе которого лежит ядро **Maple**. При некотором навыке и наличии документации связка **Maxima+TexMacs** или ядро **Maxima**+интерфейс **wxMaxima** вполне разумная замена **MathCad** в Unix-среде. А наличие универсального интерфейса в виде **TexMacs** или **Emacs** позволяет объединять в одном документе расчёты, выполненные в **Maxima**, **Octave**, **Axiom** и т.п.

Для научных работников и инженеров **СКМ** незаменимое средство анализа постановки всевозможных задач моделирования.

Под системами компьютерной математики понимают программное обеспечение, которое позволяет не только выполнять численные расчёты на компьютере, но и производить аналитические (символьные) преобразования различных математических и графических объектов. 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 и использование которых в настоящее время требует квалифицированного применения компьютера. Это касается редактирования текста, изготовления графических материалов, ведения библиографии, размещения электронных версий в Интернете, поиска статей и их просмотра. Де-факто сейчас стандартными системами подготовки научно-технических публикаций 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.

1. Возникновение и развитие систем компьютерной математики

1.1 Определение систем компьютерной алгебры

История математики насчитывает около трёх 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 берёт своё 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 всё здание современной математики.

Мы живём во время начала нового периода развития математики, который связан с изобретением и применением компьютеров. Прежде всего, компьютер предоставил возможность производить сложнейшие численные расчёты 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 даёт в распоряжение будущего инженера, экономиста, научного работника не только определённую сумму знаний, но и развивает в нём 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 её помощью лучше всего развиваются способности логического мышления, концентрации внимания, аккуратности и усидчивости.

Компьютерная алгебра — область математики, лежащая на стыке алгебры и вычислительных методов. Для неё, как и для любой области, лежащей на стыке различных наук, трудно определить чёткие 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.

1.1.1 Недостатки численных расчётов

Большинство первых систем компьютерной математики (**Eureka**, **Mercury**, **Excel**, **Lotus-123**, **MathCad** для MS-DOS, **PC MatLab** и др.) предназначались для численных расчётов. Они как бы превращали компьютер в большой программируемый калькулятор, способный быстро и автоматически (по введённой 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 вычисления 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 приближённых 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 трудоёмка 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 учёные 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 серьёзных 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.

1.1.2 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 чём 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). Ещё 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 определённой степени даётся 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 задаётся 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 объём памяти ЭВМ.

В научных исследованиях и технических расчётах специалистам приходится гораздо больше заниматься преобразованиями формул, чем собственно численным счётом. 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, ещё в 50-х годах прошлого века. Активная разработка систем компьютерной алгебры началась в конце 60-х годов. С тех пор создано значительное количество различных систем, получивших различную степень распространения; некоторые системы продолжают развиваться, другие отмирают, и постоянно появляются новые.

1.2 Классификация, структура и возможности систем компьютерной математики

1.2.1 Классификация систем компьютерной математики

В настоящее время системы компьютерной математики (СКМ) можно разделить на семь основных классов: системы для численных расчётов, табличные процессоры, матричные системы, системы для статистических расчётов, системы для специальных расчётов, системы для аналитических расчётов (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 даёт 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 присутствие им удобства работы.

Функции и процедуры, включённые в ядро, выполняются предельно быстро. Поэтому объём 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 счёт 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 её 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 ещё 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 ведёт поиск иных алгебраических методов для улучшения или даже замены таких алгоритмов.

1.2.2 Задачи систем компьютерной алгебры

Первые ЭВМ изначально создавались для того, чтобы проводить сложные расчёты, 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). Такая их особенность использовалась в разных областях знаний. Развитие компьютерных математических систем привело к появлению отдельного класса программ, который получил названия Системы Компьютерной Алгебры (СКА).

Главная задача **СКА** — это 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 определённых и неопределённых интегралов, работа с множествами, вычисления и работа с матрицами. В дополнение к перечисленному, большинство **СКА** поддерживают разнообразные численные операции: расчёт значений выражений при определённых значениях переменных, построение графиков на плоскости и в пространстве.

Большинство **СКА** включают в себя высокоуровневый язык программирования, который позволяет реализовать свои собственные алгоритмы. Наука, которая изучает алгоритмы, применяемые в **СКА**, называется компьютерной алгеброй.

1.2.3 Место компьютерной алгебры в информатике

Компьютерная алгебра есть та 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 всё больший интерес к компьютерной алгебре. Опираясь на противопоставление, можно сказать, что компьютерная алгебра рассматривает такие объекты, которые имеют слишком вычислительный характер, чтобы встречаться в книгах по алгебре, и слишком алгебраический характер, чтобы быть представленными в учебниках по

информатике. Многие алгоритмы компьютерной алгебры можно рассматривать как полу численные (в смысле Кнута).

1.2.4 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 с терминами 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 интегрирования 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 объёмом 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. Современные 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.

1.2.5 Возможности повышения эффективности решения 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 плодотворных применений. Характерной особенностью проблематики символьных преобразований является сочетание весьма тонких математических и алгоритмических методов с самыми современными методами программирования, эффективно реализующими не численную 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, особое 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 счёте 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 всё более и более удаляются от неё. 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 её 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, С), предназначена только для численных расчётов, 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 **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.

1.3 Коммерческие и свободно распространяемые системы компьютерной математики

СКМ были созданы в 70-ые годы и развивались в рамках проектов, связанных с искусственным интеллектом. Поэтому сфера применения их достаточно большая 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 использование систем компьютерной математики. Синтаксис языков систем 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 пользовательских процедур — 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 бывшем СССР получила система **Reduce**. Она первоначально предназначалась для решения физических задач, разрабатывалась на наиболее широко распространённых компьютерах, разработка до определённого времени не носила коммерческого характера (система до конца 80-х годов распространялась бесплатно). Открытый характер системы позволил привлечь к её разработке огромную армию пользователей, обогативших систему многочисленными пакетами для решения отдельных задач.

Macsyma, так же, как и **Reduce**, является "старой" системой. В отличие от системы **Reduce**, **Macsyma** разрабатывалась с самого начала как коммерческий продукт. В ней более тщательно проработаны алгоритмические вопросы, её эффективность существенно выше, но меньшее её 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. Она развивается "вширь и вглубь", даже её 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**. Её успех в значительной степени

объясняется её широкими графическими возможностями а также электронной документацией, которую можно рассматривать как электронную библиотеку, посвящённую различным разделам математики и информатики.

Особое место среди систем компьютерной алгебры занимает система **Axiom**. В отличие от остальных систем, представляющих собой пакеты программ, общение с которыми осуществляется на некотором алголо-подобном языке, система **Axiom**, развившаяся из системы **Scratchpad-II**, имеет дело с более привычными для математиков объектами. В частности, в ней ключевым понятием 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 мощных компьютеров, 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 её 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 **СКА** изначально разработана Стивеном Вольфрандом, которая продаётся компанией **Wolfram Research**. Он начал работу над **Mathematica** в 1986 году, а выпустил в 1988 году.

Mathematica не только СКА, но и мощный язык программирования. Этот язык программирования реализован на основе объектно ориентированного варианта языка С, 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 программ, 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 СКА очень похожая на **Mathematica**. Распространяется компанией **Mathsoft**. **MathCad** ориентирован на поддержку концепций рабочего листа. Уравнения и выражения отображаются на рабочем листе так, как они выглядели бы на какой-нибудь презентации, а не так, как выглядят на языке программирования. Некоторые задачи, которые выполняет программа: решение дифференциальных уравнений, графики 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 расчётов! Для последних они не слишком удобны: уравнения спрятаны в ячейках, сложно вставить комментарии. Это делает работу довольно затруднительной, а устранять ошибки и разбираться в чьих-то вычислениях вообще сложно. Электронные таблицы трудны для понимания и повторного использования другими пользователями.

Yacas — это Open Source СКА общего назначения. Базируется на собственном языке программирования, главной целью при разработке этого языка 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 своё существование в конце 1960 года в **MIT**. **Macsyma** первая создала систему компьютерной алгебры, она проложила путь для таких программ 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 сохранили её в рабочем состоянии. На сегодняшний день пакет достаточно активно развивается, и во многих отношениях не уступает таким развитым системам компьютерной математики, как **Maple** или **Mathematica**.

2. Основы Maxima

2.1 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.

2.1.1 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 вычисление определённых и неопределённых интегралов
- Решение интегральных уравнений
- Решение алгебраических уравнений и их систем
- Операции со степенными рядами и рядами Фурье
- Операции с матрицами и списками, большая библиотека функций для решения задач линейной алгебры
- Операции с тензорами
- Теория чисел, теория групп, абстрактная алгебра

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

2.2 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™**);
- 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**);

2.3 Установка и запуск программы

Скачать последнюю версию программы можно с её сайта в сети Интернет:

<http://maxima.sourceforge.net/>. Русская локализация сайта:

<http://maxima.sourceforge.net/ru/>.

2.4 Интерфейс wxMaxima

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

Достоинствами **wxMaxima** являются:

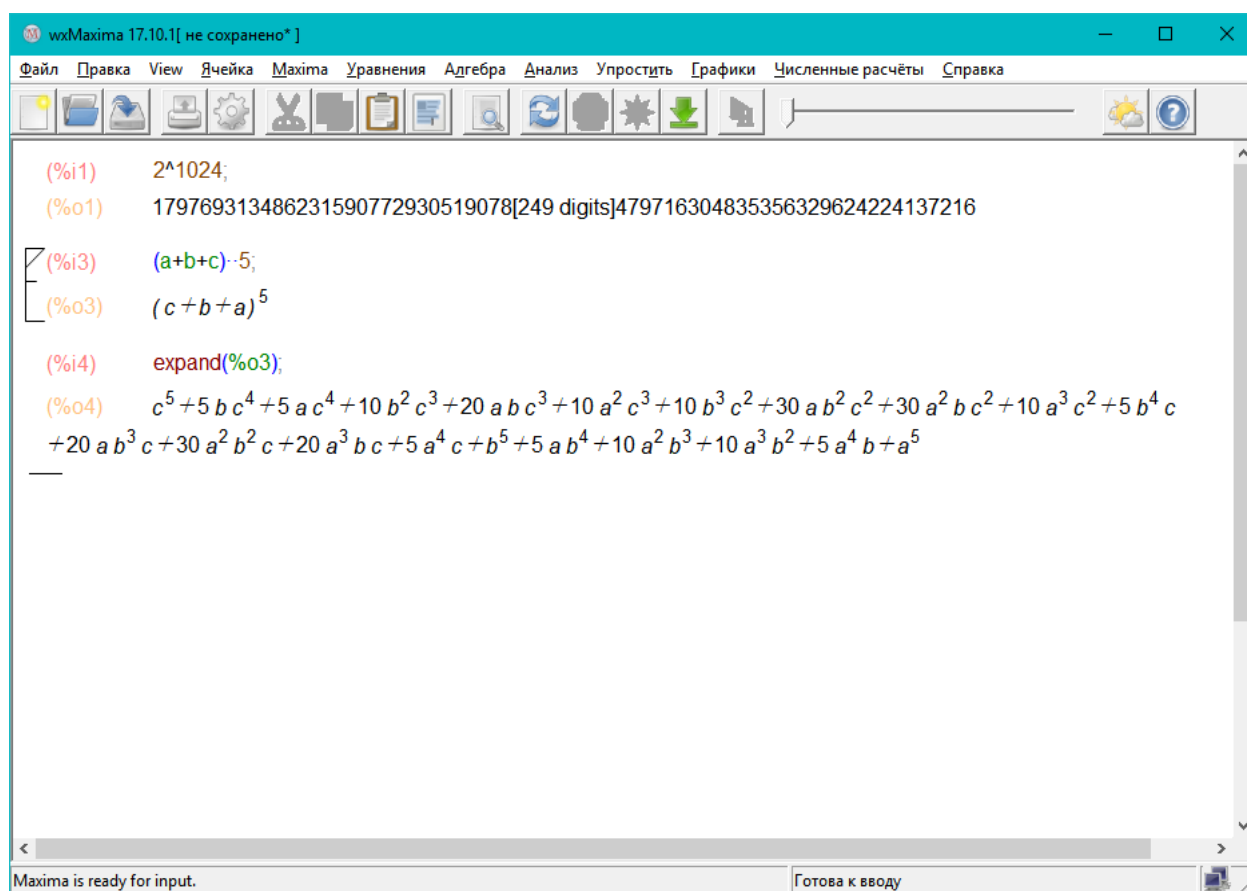
- возможность графического вывода формул (см. иллюстрации ниже)
- упрощённый ввод наиболее часто используемых функций (через диалоговые окна), а не набор команд, как в классической **Maxima**.
- разделение окна ввода данных и области вывода результатов (в классической **Maxima** эти области объединены,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

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, разделённые 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.).



2.5 Интерфейс командной строки. Ввод простейших команд Maxima

Все команды вводятся в поле ВВОД, разделителем команд является символ ; (точка с запятой). После ввода команды необходимо нажать клавишу **Enter** для её обработки и вывода результата. В ранних версиях **Maxima** и некоторых её 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
```

2.5.1 Обозначение команд и результатов вычислений

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

Результат вычисления также имеет порядковый номер, например (%o1), (%o2) и т.д., где **i** — сокращение от англ. input (ввод), а **o** — англ. output (вывод). Этот механизм позволяет избежать в последующих вычислениях повторения полной записи уже выполненных команд, например (%i1)+(%i2) будет означать добавление к выражению первой команды — выражения второй и последующего вычисления результата. Также можно использовать и номера результатов вычислений, например (%o1) * (%o2). Для последней выполненной команды в Maxima есть специальное обозначение — %.

Пример:

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

```
(%i1) diff(x^2*exp(-x), x);
(%o1) 2xe^{-x} - x^2e^{-x}
(%i2) f(x) := '%;
(%o2) f(x) := 2xe^{-x} - x^2e^{-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
```

2.6 Числа, операторы и константы

2.6.1 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)          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
```

2.6.2 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 нахождения факториала обозначается восклицательным знаком, например 5!. Для увеличения приоритета операции, как и в математике, используются круглые скобки: (). Список основных арифметических и логических операторов приведён в табл. 2.1 и табл. 2.2 ниже.

Таблица 2.1. Арифметические операторы

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

Таблица 2.2. 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

2.6.3 Константы

В **Maxima** для удобства вычислений имеется ряд встроенных констант. Самые распространённые из них показаны в табл. 2.3:

Таблица 2.3. Основные константы Maxima

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

2.7 Типы данных, переменные и функции

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

`kill(x)` — удалить значение переменной x ;

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

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

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

2.7.1 Списки

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

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

2.7.1.1 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:

- $\text{makelist}(\text{expr}, i, i_0, i_1)$ — возвращает список, j -й элемент которого равен $\text{ev}(\text{expr}, i = j)$, при этом индекс j меняется от i_0 до i_1
- $\text{makelist}(\text{expr}, x, \text{list})$ — возвращает список, j -й элемент которого равен $\text{ev}(\text{expr}, x = \text{list}[j])$, при этом индекс j меняется от 1 до $\text{length}(\text{list})$.

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 $\text{create_list}(\text{form}, x_1, \text{list}_1, \dots, x_n, \text{list}_n)$.

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** ($\text{list}_1, \dots, \text{list}_n$)

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, komponуя элементы двух списков поочерёдно в порядке следования, позволяет функция `join(l,m)`. Новый список содержит l_1 , затем m_1 , затем l_2, m_2 и т.д.

Пример:

```
(%i1) join([1, 2, 3], [10, 20, 30]);
(%o1)      [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(expr1, expr2)` 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]
```

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

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

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

Пример:

```
(%i1) product (x + i*(i+1)/2, i, 1, 4);
(%o1)      (x + 1) (x + 3) (x + 6) (x + 10)
(%i2) sum (x + i*(i+1)/2, i, 1, 4);
(%o2)      4x + 20
(%i3) product (i^2, i, 1, 4);
(%o3)      576
(%i4) sum (i^2, i, 1, 4);
(%o4)      30
```

2.7.1.2 Функции, оперирующие с элементами списков

Функция $map(f, expr_1, \dots, expr_n)$ позволяет применить функцию (оператор, символ операции) f к частям выражений $expr_1, expr_2, \dots, expr_n$. 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 $(F, [x_1, \dots, x_n])$ вычисляется выражение $F(arg_1, \dots, arg_n)$). 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*.

2.7.2 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 массива. Такие переменные рассматриваются как элементы массивов с неопределёнными размерностями (так называемые хэш-массивы). Размеры неопределённых массивов растут динамически по мере присваивания значений элементам. Интересно, что индексы массивов с неопределёнными границами не обязательно должны быть числами. Для повышения эффективности вычислений рекомендуется преобразовывать

массивы с неопределёнными границами в обычные массивы (для этого используется функция **array**).

Создание массива производится функцией **array**. Синтаксис обращения к функции:

array(name, dim1, ..., dimn)

— создание массива с именем **name** и размерностями **dim1, ..., dimn**;

array(name, type, dim1, ..., dimn)

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

array([name1, ..., name_n], dim1, ..., dimn)

— создание нескольких массивов одинаковой размерности.

Индексы обычного массива — целые числа, изменяющиеся от 0 до **dimn**

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, преобразует массив в список. Синтаксис вызова: **listarray(a)** ; .

Аргумент **a** может быть определённым или неопределённым массивом, функцией массива или функцией с индексами. Порядок включения элементов массива в список — по строкам.

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

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)      ccx,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)      ddx (y) := yx
(%i2) dd [1] (4);
(%o2)      4
(%i3) dd [a+b];
(%o3)      lambda ([y], yb+a)
(%i4) arrayinfo (dd);
(%o4)      [hashed, 1, [1], [b + a]]
(%i5) listarray (dd);
(%o5)      [lambda ([y], y), lambda ([y], yb+a)]
```

Функция **make_array**(type, dim1, ..., dimn) создаёт и возвращает массив Lisp. Тип массива может быть **any**, **flonum**, **fixnum**, **hashed**, **functional**. Индекс *i* может изменяться в пределах от 0 до **dimn-1**

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

Присваивание **y: make_array(...)** создаёт ссылку на массив. Когда массив больше не нужен, ссылка уничтожается присваиванием **y:false**, память освобождается затем сборщиком мусора.

Примеры:


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

Переменная **arrays** содержит список имён массивов первого и второго видов, определённых 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)      Lisp Array #(0 0 0 0 0 0 0 0)
(%i5) fillarray (a2, [1, 2, 3, 4, 5]);
(%o5)      Lisp Array #(1 2 3 4 5 5 5 5)
```

Как видно из рассмотренных 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, dim₁, ..., dim_n)**. Новый массив заполняется элементами старого по строкам. Если размер старого массива меньше, чем нового, остаток нового заполняется нулями или **false** (в зависимости от типа массива).

2.7.3 Матрицы и простейшие операции с ними

В **Maxima** определены прямоугольные матрицы.

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

Синтаксис вызова: **matrix(row1, ..., rown)**.

Каждая строка — список выражений, все строки одинаковой длины. На множестве матриц определены операции сложения, вычитания, умножения и деления. *Эти операции выполняются поэлементно*, если операнды — две матрицы, скаляр и матрица или матрица и скаляр.

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

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

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

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

Пример создания двух матриц:

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

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

```

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

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

```

Выполнение арифметических операций с матрицами:

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

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

```

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

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

```

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

```
(%o5) 
$$\begin{bmatrix} 17\pi & 3e \\ -a8 & 11b \end{bmatrix}$$

```

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

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

```

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) $x \cdot y$;

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

(%i12) $y \cdot x$;

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

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

Возведение в степень -1 даёт обратную матрицу:

(%i13) x^{-1} ;

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

(%i14) $x \cdot (x^{-1})$;

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

Стоит обратить внимание, что операции x^{-1} и x^{-1} дают разный результат!

Пример:

(%i2) x^{-1} ;

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

(%i3) x^{-1} ;

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

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 элементы 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)  $\pi$ 
```

```
(%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**(n,m) .

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

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

```

Функция **ident** возвращает единичную матрицу заданной размерности. Синтаксис вызова **ident**(n).

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

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

```

Функция **copymatrix**(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)  $\begin{bmatrix} 1 & 2 \\ 3 & 10 \end{bmatrix}$ 
(%i4) a;
```

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

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

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

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

- $addcol(M, list_1, \dots, list_n)$
- $addrow(M, list_1, \dots, list_n)$

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

Пример:

```
(%i1) a:matrix([1,2],[3,4]);
(%o1)  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 
(%i2) b:addrow(a,[10,20]);
(%o2)  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 10 & 20 \end{bmatrix}$ 
(%i3) addcol(b,[x,y,z]);
(%o3)  $\begin{bmatrix} 1 & 2 & x \\ 3 & 4 & y \\ 10 & 20 & z \end{bmatrix}$ 
```

Функция **submatrix(M)** возвращает новую матрицу, состоящую из подматрицы заданной. Синтаксис вызова:

- $submatrix(i_1, \dots, i_m, M, j_1, \dots, j_n)$
- $submatrix(i_1, \dots, i_m, M)$
- $submatrix(M, j_1, \dots, j_n)$

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

Пример: удаляем третью строку и третий столбец:

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

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

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) [ 1 2 ]
      [ 3 4 ]

(%i2) f(x):=x^2;
(%o2)          f(x) := x^2

(%i3) matrixmap(f,a);
(%o3) [ 1 4 ]
      [ 9 16 ]
```

Для работы с матрицами существует ещё много функций, но они относятся к решению различных задач линейной алгебры, поэтому обсуждаются ниже, в 3.2.

2.7.4 Математические функции

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

- тригонометрические функции: *sin* (синус), *cos* (косинус), *tan* (тангенс), *cot* (котангенс);

- обратные тригонометрические функции: *asin* (арксинус), *acos* (арккосинус), *atan* (арктангенс), *acot* (арккотангенс);
- *sec* (секанс, $sec(x) = \frac{1}{cos(x)}$), *csc* (косеканс, $csc(x) = \frac{1}{sin(x)}$);
- *sinh* (гиперболический синус), *cosh* (гиперболический косинус), *tanh* (гиперболический тангенс), *coth* (гиперболический котангенс), *sech* (гиперболический секанс), *csch* (гиперболический косеканс);
- *log* (натуральный логарифм);
- *sqrt* (квадратный корень);
- *mod* (остаток от деления);
- *abs* (модуль);
- *min*(x_1, \dots, x_n) и *max*(x_1, \dots, x_n) — нахождение минимального и максимального значения в списке аргументов;
- *sign* (определяет знак аргумента: *pos* — положительный, *neg* — отрицательный, *pnz* — не определён, *zero* — значение равно нулю);
- Специальные функции — функции Бесселя, гамма-функция, гипергеометрическая функция и др.;
- Эллиптические функции различных типов.

2.7.5 Вычисление и преобразование аналитических выражений

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

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

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

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

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

expr, **flag1**, **flag2**, ...;

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

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

На выражение **expr** по умолчанию действует функция упрощения. Необходимость выполнения упрощения регулируется флагом **simp** - если установить **simp: false** упрощение будет отключено.

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

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

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

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

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

Пример:

```
(%i1) ev((a+b)^2, expand);
(%o1)          b^2 + 2ab + 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 + 14x + 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) +  $\frac{d}{dw}$  sin(w) + (w + 1)^2
(%i2) ev(%, sin, expand, diff, x=2, y=1);
(%o2)  cos(w) + w^2 + 2w + cos(1) + 1.909297426825682
```

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

```
(%i1) f:a+2*a+3*a+4*a;
(%o1)          10a
(%i2) simp:false;
(%o2)          false
(%i3) f:a+2*a+3*a+4*a;
(%o3)          a + 2a + 3a + 4a
```

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

Пример:

```
(%i1) x^3-1, factor;
(%o1)      (x - 1) (x^2 + x + 1)
(%i2) factor(x^3-1);
(%o2)      (x - 1) (x^2 + x + 1)
```

Ещё примеры факторизации различных выражений:

```
(%i3) factor(-8*y - 4*x + z^2*(2*y + x));
(%o3)      (2y + 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)      (e^x + 1) (e^2x - e^x + 1)
```

Пример использования функции **gfactor**:

```
(%i6) gfactor(x^2+a^2);
(%o6)      (x - i a) (x + i a)
(%i7) gfactor(x^2+2*i*x*a-a^2);
(%o7)      (x + i a)^2
```

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

```
(%i1) expand((x-a)^3);
(%o1)      x^3 - 3ax^2 + 3a^2x - a^3
(%i2) expand((x-a)*(y-b)*(z-c));
(%o2)      xyz - ayz - bxz + abz - cxy + acy + bcx - abc
(%i3) expand((x-a)*(y-b)^2);
(%o3)      xy^2 - ay^2 - 2bxy + 2aby + b^2x - ab^2
```

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

```
(%i8) expand ((x + 1)*((u + v)^2 + a*(w + z)^2));
(%o8)
axz^2+az^2+2awxz+2awz+aw^2x+v^2x+2uvx+u^2x+aw^2+v^2+2uv+u^2
(%i9) factorsum(%);
(%o9) (x + 1) (a(z + w)^2 + (v + u)^2)
```

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

```
gfactorsum(a^3+3*a^2*b+3*a*b^2+b^3+x^2+2*i*x*y-y^2 );
(%o10) (b + a)^3 - (y - i x)^2
```

Функция **combine** объединяет слагаемые с идентичным знаменателем

```
(%i5) combine (x/(1+x^2)+y/(1+x^2));
(%o5)
      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)
      (y + x)^2 + 1
      -----
     (y + x)^12
```

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

```
(%o1)
      1          (x + 2)^20 - 2 y          x
      ----- + ----- - -----
     (y + x)^19  (y + x)^20          (y + x)^20
```

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

```
(%o2)
      (x + 2)^20 - y
      -----
     (y + x)^20
```

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

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

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

Пример:

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

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

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

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

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

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

(%i4) **multthru** (%);

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

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]

(%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);

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

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

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

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

(%o27) [x - 1]

Подстановки осуществляются функцией **subst**.

Вызов этой функции: **subst** (a, b, c)

- подставляем a вместо b в выражении c.

Пример:

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

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

2.7.6 Преобразование рациональных выражений

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

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

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

(%i2) **num**(expr);

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

(%i3) **denom**(expr);

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

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

Синтаксис вызова:

- **rat**(expr)
- **rat**(expr, x₁, ..., x_n)

Переменные упорядочиваются в соответствии со списком x₁, ..., 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)
$$\frac{(y + a) (2y + x) \left(\frac{(x-2y)^4}{(x^2-4y^2)^2} + 1 \right)}{4y^2 + x^2}$$

(%i2) **rat**(%);

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

После указания порядка использования переменных получаем следующее выражение:

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

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

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 $\text{ratvars}(m, n, a, b)$ упорядочивает переменные m, n, a, b в порядке возрастания приоритета.

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

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

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

$$(\%01) \quad \sin\left(\frac{x}{x^2 + x}\right) = e^{(\log(x)+1)^2 - \log(x)^2}$$

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

$$(\%02) \quad \sin\left(\frac{1}{x+1}\right) = e^{x^2}$$

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

$$(\%03) \quad \frac{(x-1)^{\frac{3}{2}} - \sqrt{x-1}(x+1)}{\sqrt{(x-1)(x+1)}}$$

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

$$(\%04) \quad \frac{2\sqrt{x-1}}{\sqrt{x^2-1}}$$

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

$$(\%05) \quad x^{\frac{a^2+1}{a}}$$

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);
```

$$(\%01) \quad \frac{(x^{\frac{a}{2}} - 1)^2 (x^{\frac{a}{2}} + 1)^2}{x^a - 1}$$

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

$$(\%02) \quad \frac{x^{2a} - 2x^a + 1}{x^a - 1}$$

```
(%i3) fullratsimp(expr);
(%o3)          x^a - 1
(%i4) rat(expr);
(%o4)          (x^(a/2))^4 - 2(x^(a/2))^2 + 1
              x^a - 1
```

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

```
(%i1) fullratsimp( exp((x^(a/2)-1)^2 * (x^(a/2)+1)^2 /
                    (x^a-1) ) );
(%o1)          e^(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 отличаться 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)          x - 1          1
              (x + 1)^2 + x - 1
(%i3) expand(expr);
(%o3)          x          1          1
              x^2 + 2x + 1 - x^2 + 2x + 1 + x - 1
(%i4) ratexpand(expr);
(%o4)          2x^2          2
              x^3 + x^2 - x - 1 + 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(x)^4 + cos(x)^3 + cos(x)^2 + cos(x) + 1

(%i3) ratsubst(1 -- sin(x)^2, cos(x)^2, %);
(%o3)  sin(x)^4 - 3 sin(x)^2 + cos(x) (2 - sin(x)^2) + 3
```

2.7.7 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(x)^2 + 3 cos(x)^2 + x
(%i2) trigexpand(sin(10*x+y));
(%o2)  cos(10x) sin(y) + sin(10x) cos(y)
(%i3) trigexpand(sin(3*x)+cos(4*x));
(%o3)  sin(x)^4 - sin(x)^3 - 6 cos(x)^2 sin(x)^2 + 3 cos(x)^2 sin(x) + cos(x)^4
```

Функция **trigreduce(expr)** свёртывает все произведения тригонометрических и гиперболических функций в комбинации соответствующих функции от сумм. Функция работает не до конца, так что повторный вызов может изменить выражение.

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

Примеры:

```
(%i8) trigreduce(cos(x)^4 + cos(x)^3 + cos(x)^2 +
              cos(x) + 1);
(%o8)
cos(4x) + 4 cos(2x) + 3 cos(3x) + 3 cos(x) + cos(2x) + 1
----- + ----- + ----- + cos(x) + 1
      8           4           2

(%i9) trigreduce(-sin(x)^2 + 3*cos(x)^2 + x);
(%o9)
cos(2x)
----- + 3 (----- + 1/2) + x - 1/2
      2
```

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

Пример :

```
(%i1) trigsimp(sin(x)^2+3*cos(x)^2);
(%o1)          2*cos(x)^2 + 1
(%i2) trigsimp(sinh(x)^2+3*cosh(x)^2);
(%o2)          4*cosh(x)^2 - 1
```

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

Пример:

```
(%i1) trigrat((1+sin(2*b)-cos(2*b))/sin(b));
(%o1)          2*sin(b) + 2*cos(b)
```

2.7.8 Преобразование степенных и логарифмических выражений

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

Примеры:

```
(%i1) (log(x+x^2)-log(x))^a/log(1+x)^(a/2);
(%o1)          (log(x^2+x)-log(x))^a
                log(x+1)^(a/2)
(%i2) radcan(%);
(%o2)          log(x+1)^(a/2)
(%i10) (%e^x-1)/(1+%e^(x/2));
(%o10)          e^x - 1
                e^(x/2) + 1
(%i11) radcan(%);
(%o11)          e^(x/2) - 1
```


Функция **logcontract(expr)** рекурсивно сканирует выражение **expr**, преобразуя выражения вида $a_1 * \log(b_1) + a_2 * \log(b_2) + c$ к форме $\log(\text{ratsimp}(b_1^{a_1} * b_2^{a_2})) + c$.

Пример:

```
(%i1) 2*(a*log(x)+3*b*log(y));
(%o1) 2(3b log(y) + a log(x))
(%i2) logcontract(%);
(%o2) b log(y^6) + a log(x^2)
```

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

```
(%i1) declare(n, integer);
(%o1) done
(%i2) logcontract(3*a*n*log(x));
(%o2) a log(x^{3n})
```

2.7.9 Пользовательские функции

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

Пример:

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

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:

- $define(f(x_1, \dots, x_n), expr)$

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

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 (аналогично 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)
```

Создание функции-массива:

```
(%i1) define (G2 [x, y], x.y - y.x);
(%o1)      G2x,y := x.y - y.x
```

Создание массива:

```
(%i2) define (arraymake (F, [u]), cos(u) + 1);
(%o2)      Fu := 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)
```

2.8 Решение задач элементарной математики

2.8.1 Нахождение корней уравнений и систем алгебраических уравнений

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

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

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

solve([eqn₁, eqn₂, . . . , eqn_n], [x₁, x₂, . . . , x_n]) — решение системы уравнений.

В качестве параметров:

- в первых квадратных скобках указывается список уравнений через запятую — $L_{\text{expr}_1} = R_{\text{expr}_2}, L_{\text{expr}_1} = R_{\text{expr}_2}, \dots$;

- во вторых квадратных скобках указывается — список переменных, через запятую.

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) $\left[x = -\frac{\sqrt{b^2 + 4a} + b}{2}, x = \frac{\sqrt{b^2 + 4a} - b}{2} \right]$

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

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 выдаётся соответствующее сообщение только и одно из решений.

Пример:

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

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

(%o13) $[x = 0]$

Также Maxima позволяет находить комплексные корни

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

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

2.9 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* (Тсl/Тк программа с графическим интерфейсом пользователя; элемент xMaxima)
- *gnuplot* (мощная утилита для построения графиков, обмен с Maxima — через канал)
- *mgplot* (Тк-интерфейс к *gnuplot* с рудиментарным графическим интерфейсом 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 трехмерных графиков).

При использовании **wxMaxima** кроме них используются ещё две аналогичные команды: **wxplot2d** и **wxplot3d**. Все команды позволяют либо вывести график на экран, либо (в зависимости от параметров функции) в файл.

2.9.1 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])
```

несколько графиков:

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

или

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

несколько графиков:

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

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

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

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

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

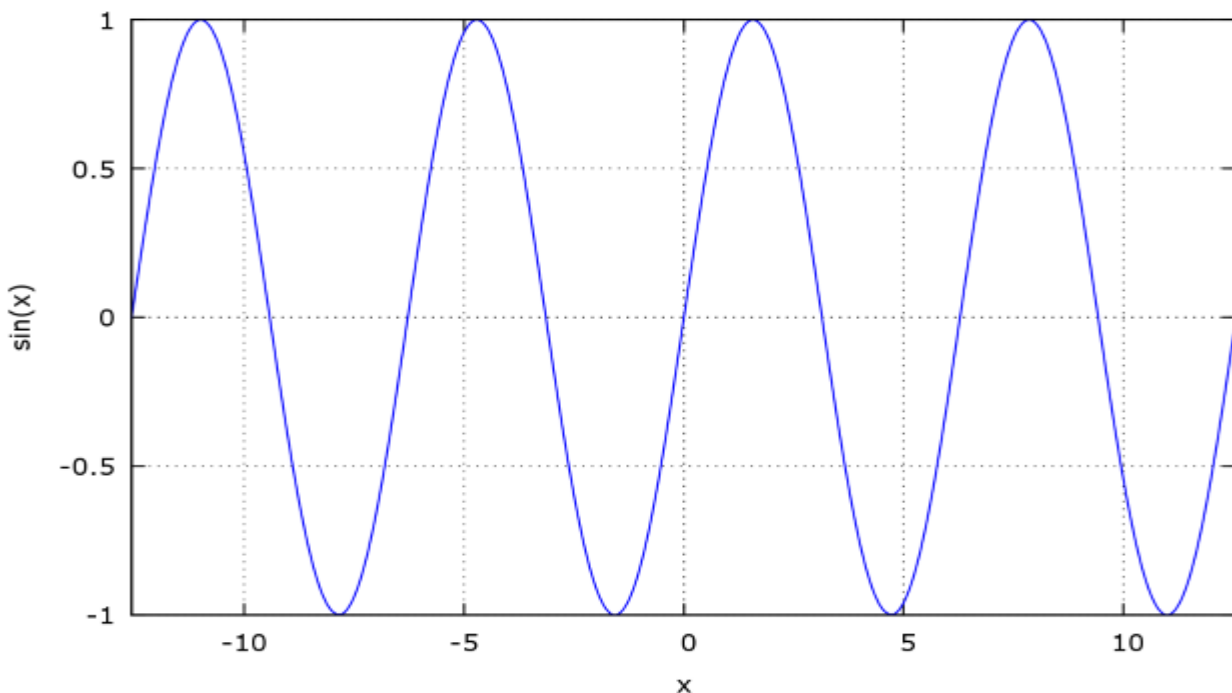


Рис. 2.1. Простейшая команда построения графика

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

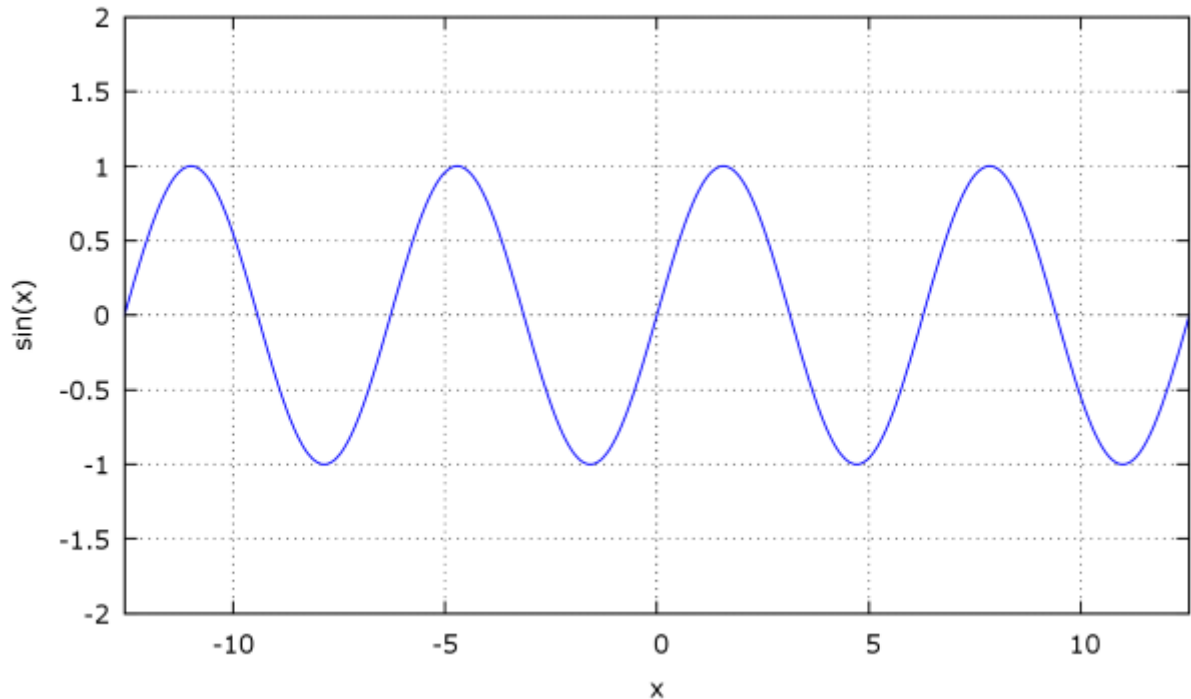
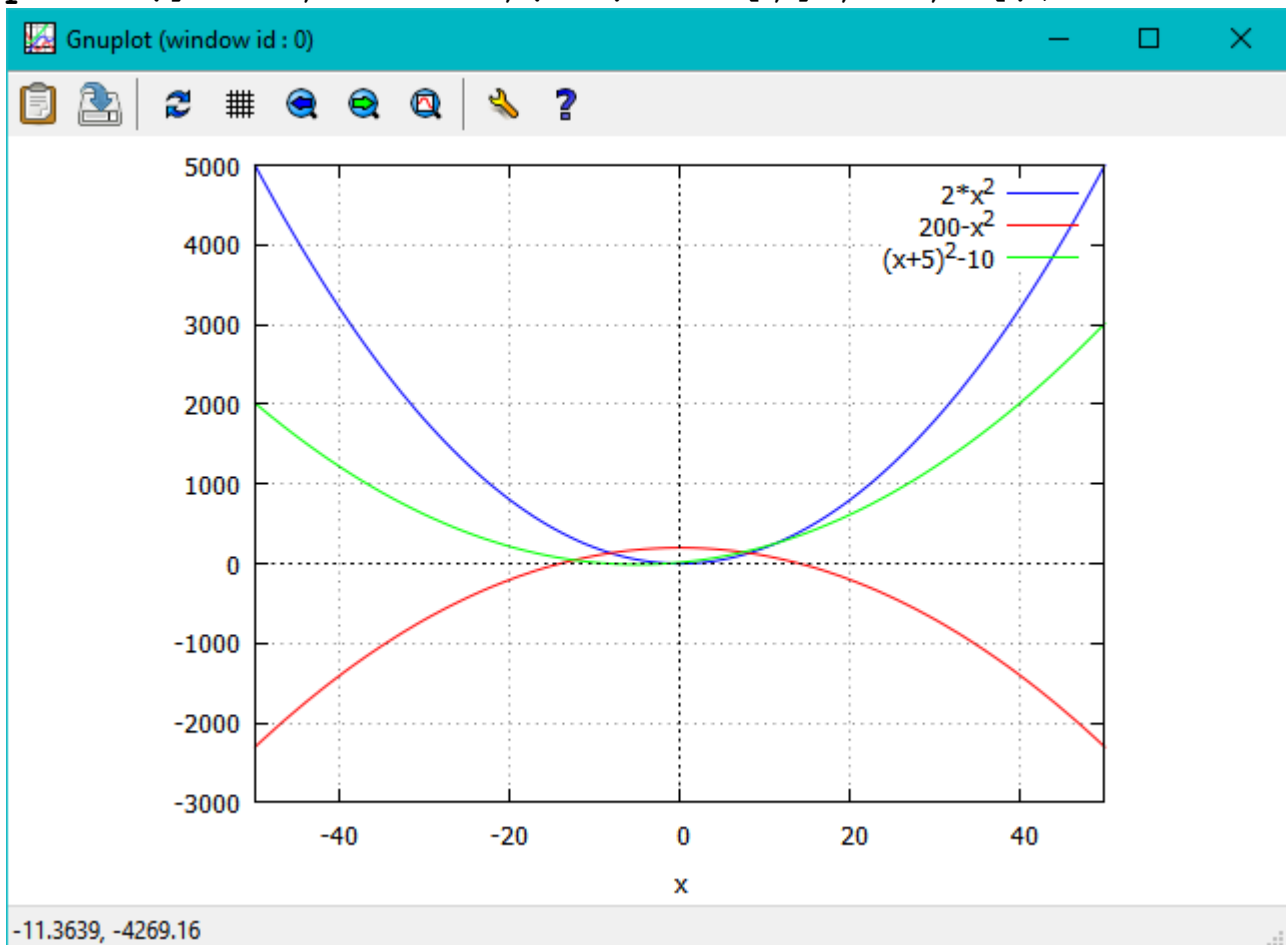


Рис. 2.2. Простейшая команда построения графика с указанием интервала по оси $0y$

Пример построения несколько графиков на одном рисунке:

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



2.9.2 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 рис. 2.3. Команда построения графика:

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

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

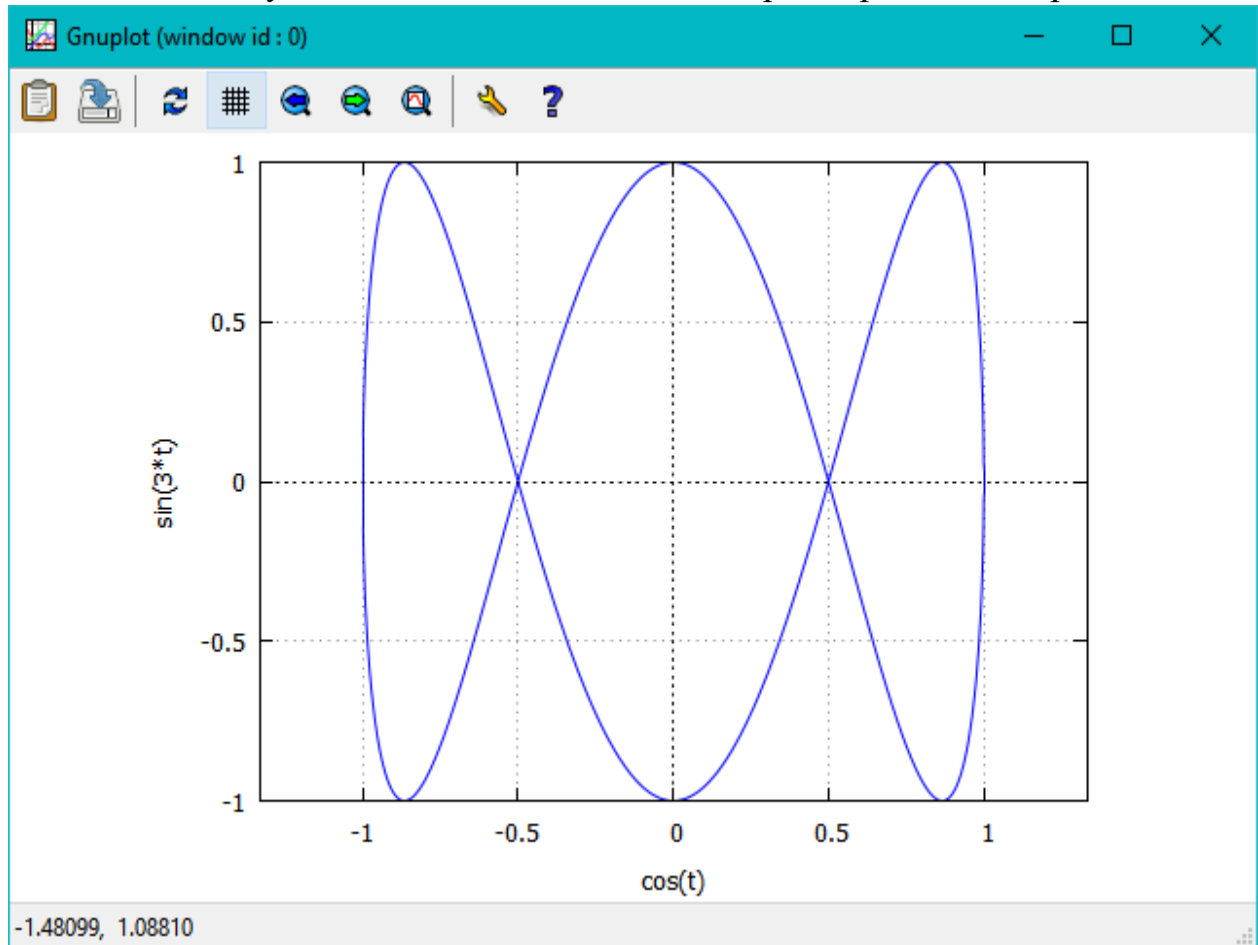


Рис. 2.3. Простейшая команда построения графика функции, заданной параметрически

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:

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

Результат приведён на рис. 2.4.

В данном примере в одних осях строятся два графика.

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

Второй график строится по уравнению функции $2 * \pi * \sqrt{1/980}$ с указанием стиля **lines**.

Опция **legend** указывает подписи кривых.

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

Формирование массива для построения графика осуществляется следующим образом:

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

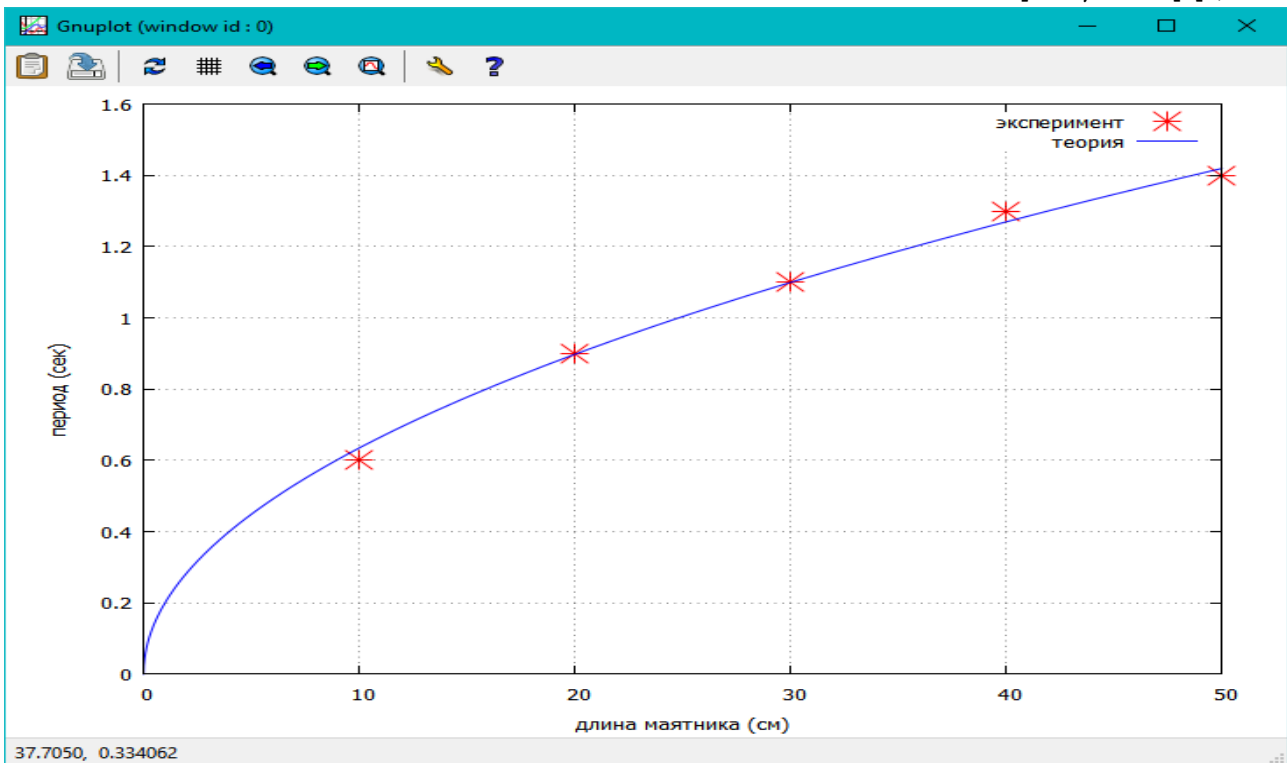


Рис. 2.4. Совмещение на одном графике действия серии опций

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.рис. 2.5):

```
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] 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, "test1.png" ]
```

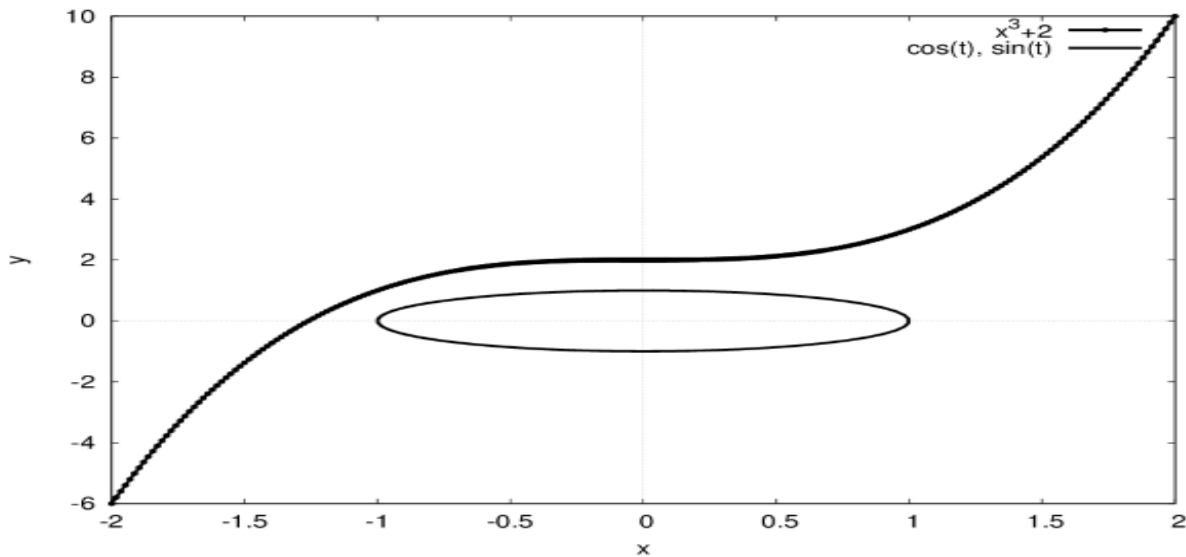


Рис. 2.5. Совмещение на одном графике параметрической и заданной явно кривых

2.9.3 Построение кривых в полярной системе координат

Для построения графика в полярных координатах нужно задать изменение значений полярного радиуса и полярного угла. Пусть $r=r(f)$ ($a \leq f \leq b$) — зависимость полярного радиуса r от полярного угла f . Тогда график этой функции в полярных координатах можно построить, задав у функции **plot2d** опцию [**gnuplot_postamble**, "**set polar; set zeroaxis**"].

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

Пример: построить в полярных координатах графики трёх функций $r = 6\cos(\varphi)$, $r = \varphi$, $r = 2\sin(\varphi)$, $0 \leq \varphi \leq 2\pi$.

Для создания графика используем команду:

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

Результат приведён на рис. 2.6.

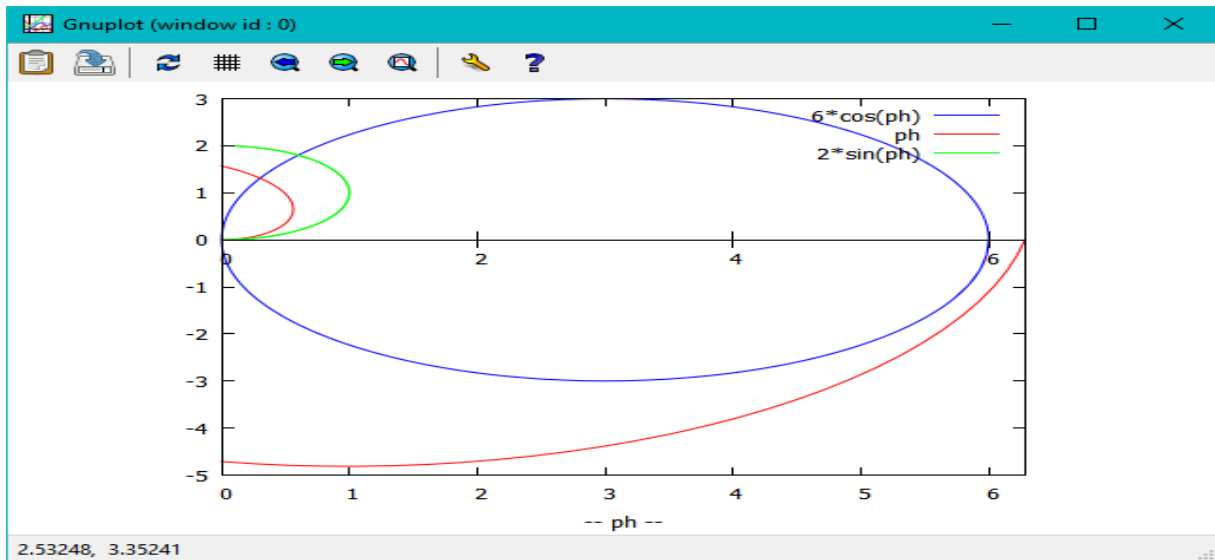


Рис. 2.6. Совмещение на одном графике нескольких параметрических кривых.

Еще один пример:

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

Результат приведён на рис. 2.7.

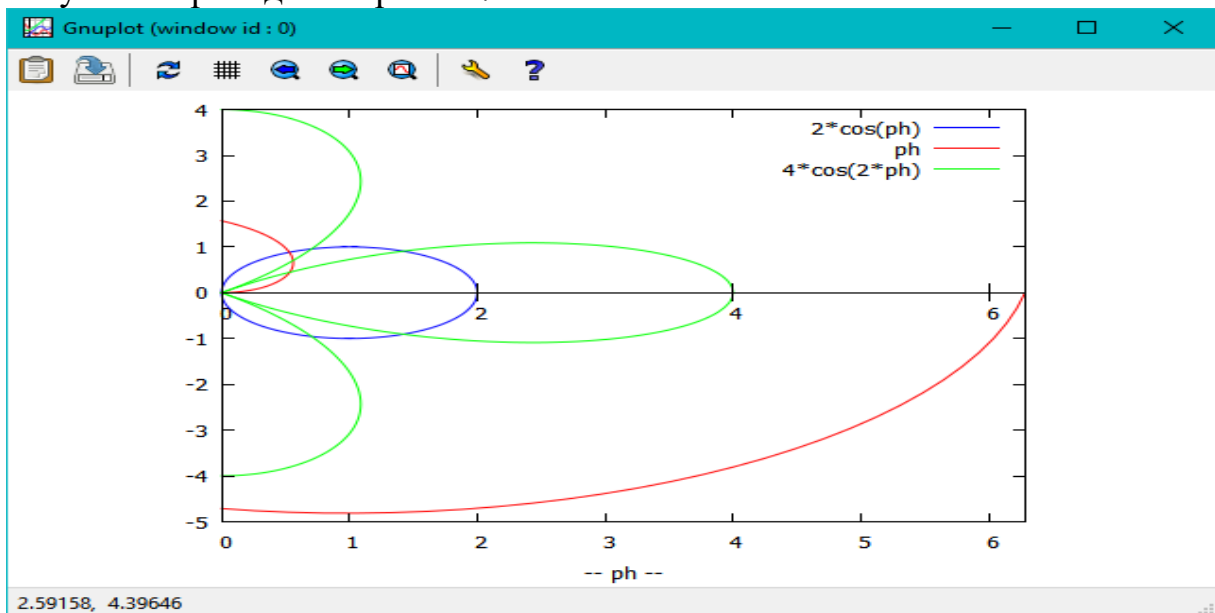


Рис. 2.7. Совмещение на одном графике нескольких параметрических кривых.

2.9.4 Построение трёхмерных графиков

Основная команда для построения трёхмерных графиков — **plot3d**. Рассмотрим технологию построения графиков с использованием интерфейса **plot3d**. Поверхность функции в цветном изображении строится с использованием опции **gnuplot_pm3d** (рис. 2.8).

Пример:

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

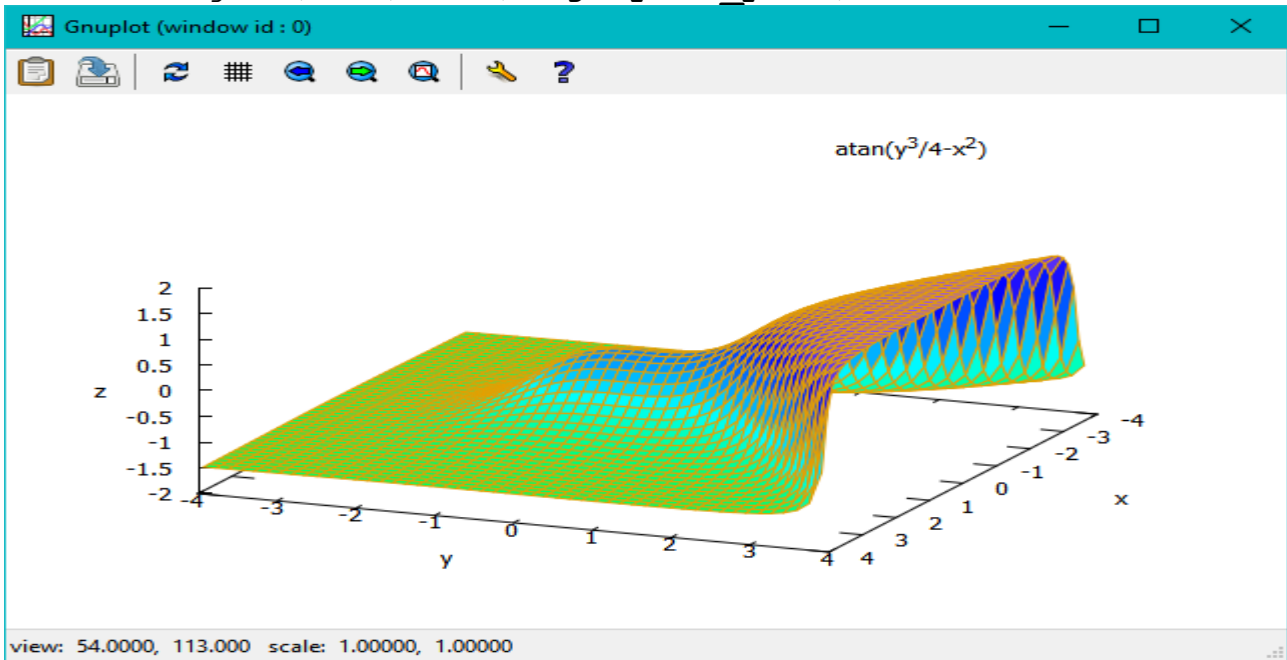


Рис. 2.8. График функции двух переменных с окраской поверхности

С использованием этой опции и особенностей программы **gnuplot** можно построить и изображение линий уровня функции. Пример (рис. 2.9):

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

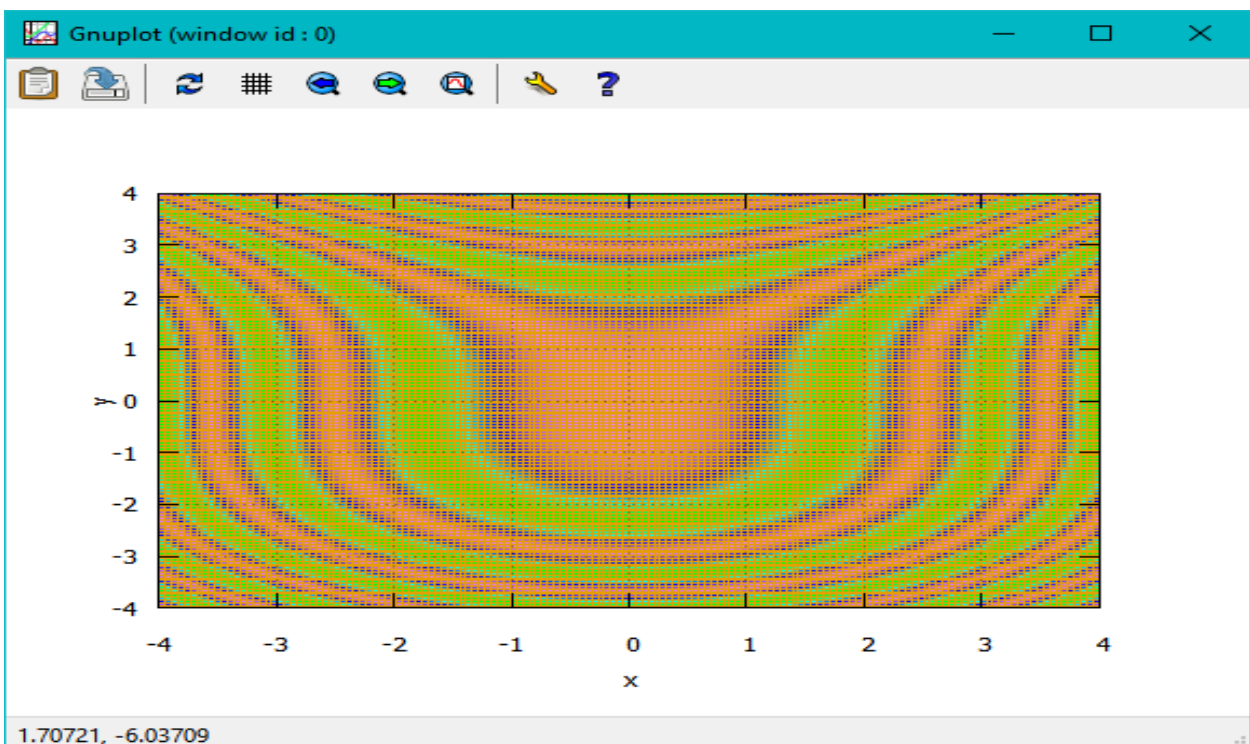


Рис. 2.9. График линий уровня функции двух переменных с окраской поверхности

Более строгий результат можно получить, используя стандартный формат функции `plot3d`. Пример (рис. 2.10):

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

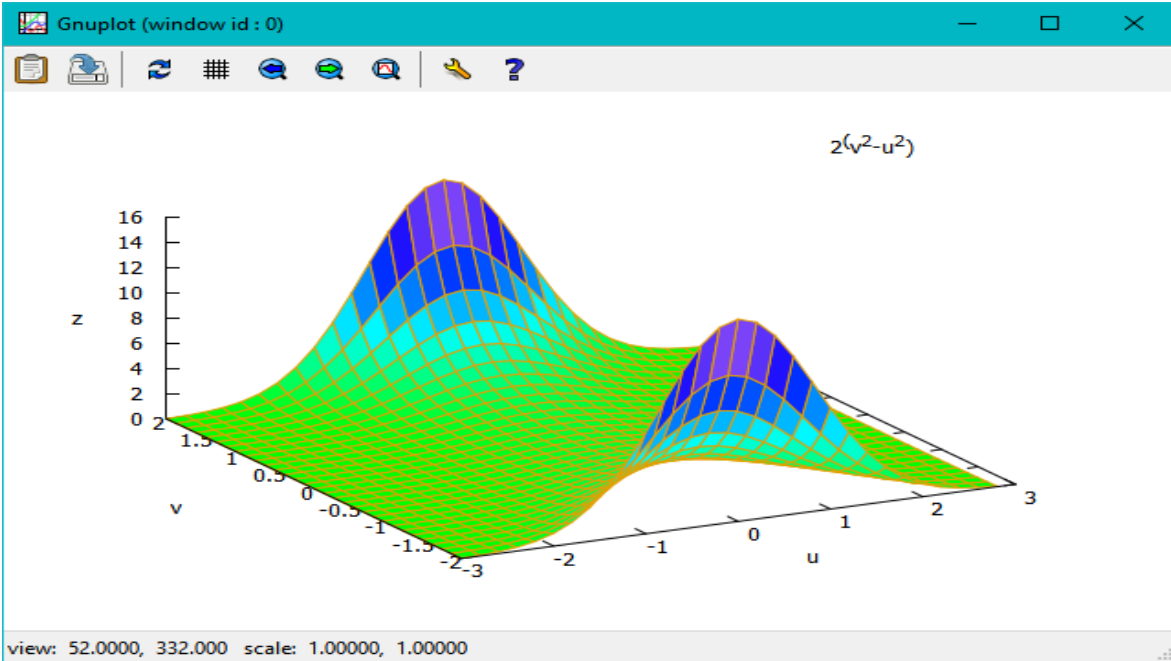


Рис. 2.10. Простой график функции двух переменных

Для вывода графика в файл можно использовать опции `gnuplot:` установить терминал `gnuplot` и имя файла результата. Необходимая команда:

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

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

Пример (вывод графики в формате `openmath` — рис. 2.11):

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

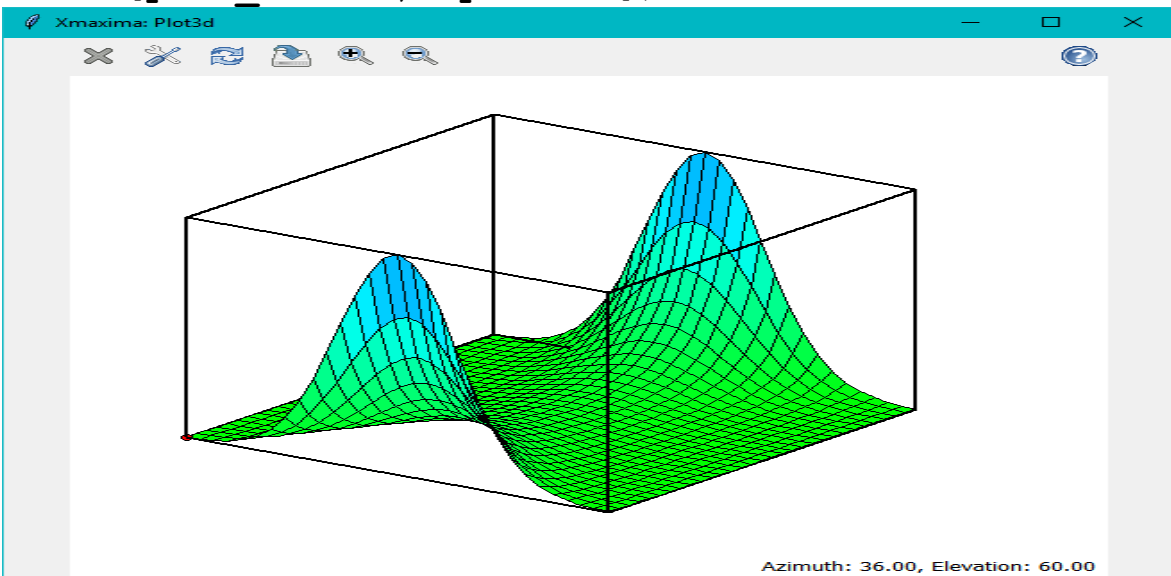


Рис. 2.11. Простой график функции двух переменных (формат OpenMath)

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

График параметрически заданной функции строится так:

```
plot3d([выражение1, выражение2, выражение3],
       [переменная1, начало, конец],
       [переменная2, начало, конец]),
```

где выражения отвечают, по порядку, $x(u, v)$, $y(u, v)$, $z(u, v)$

Пример:

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

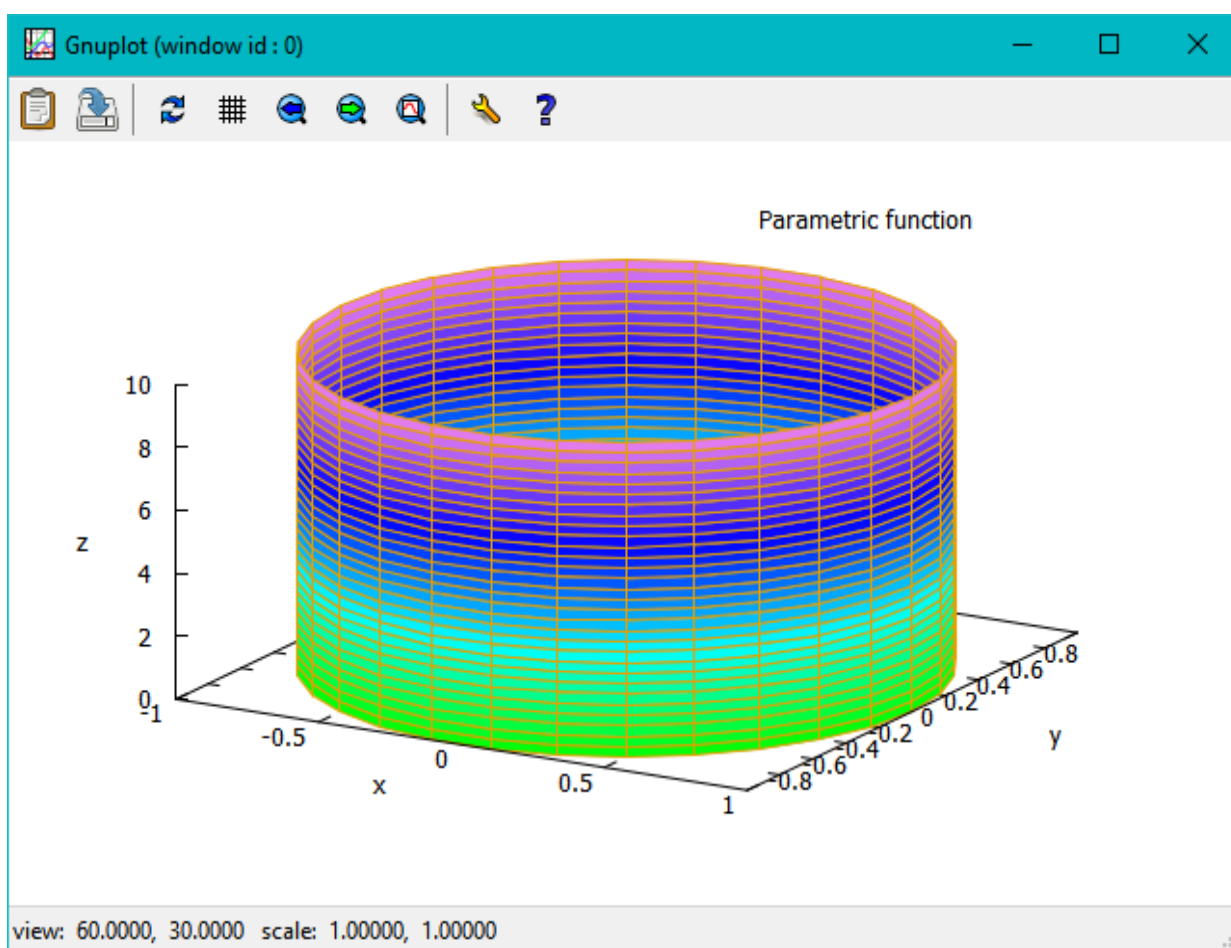


Рис. 2.12 График параметрически заданной функции

С помощью параметрической формы можно строить и пространственные кривые. Для этого просто нужно задать второй, фиктивный, параметр, чтобы Maxima «не ругалась» на неправильный синтаксис вызова функции:

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

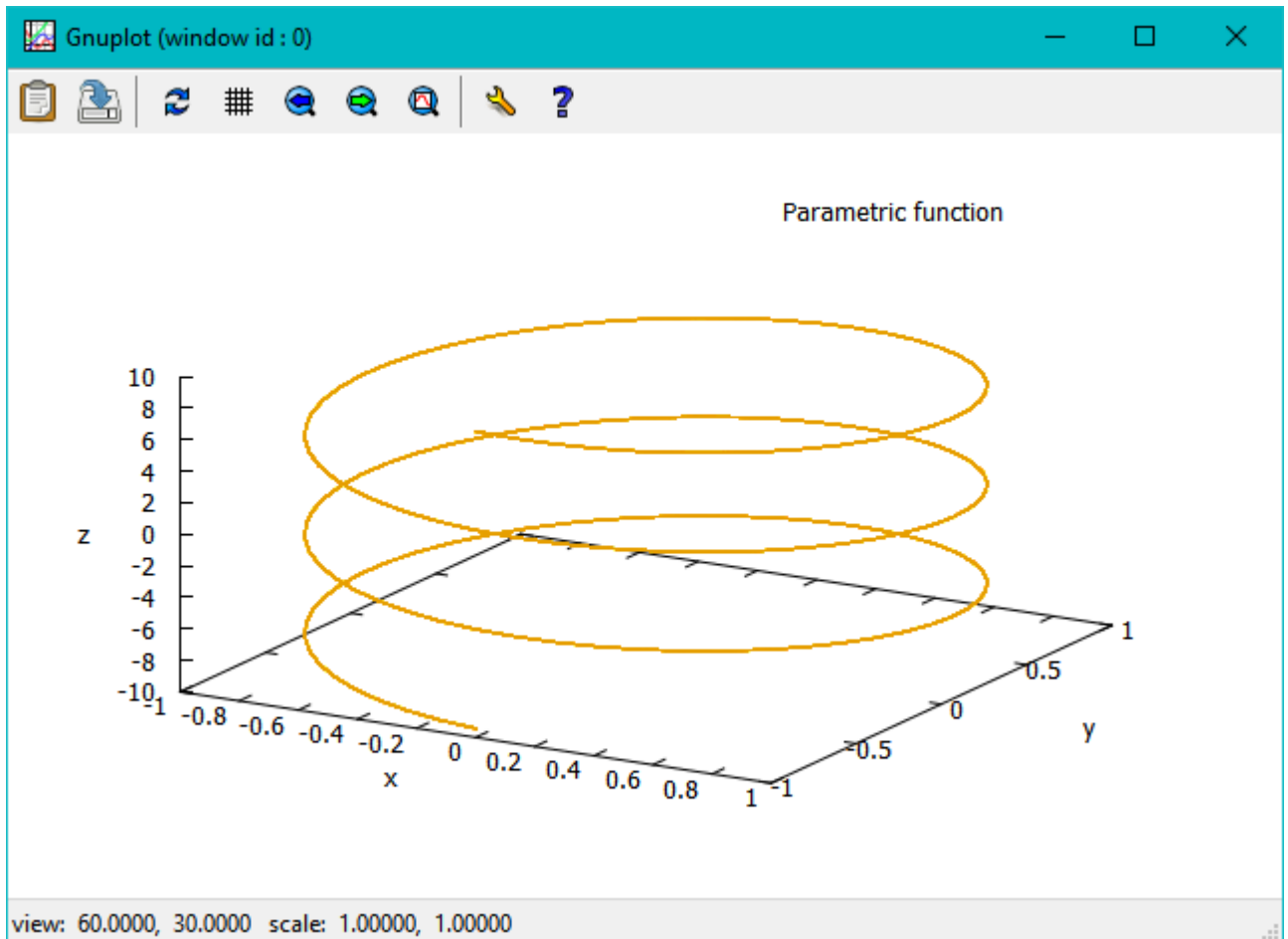


Рис 2.13. График пространственной кривой.

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

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)`).

Рассмотрим опцию - универсальную, пригодную для всех форматов и преобразующую не результирующее изображение, а сам процесс построения графика; точнее, систему координат. Называется эта опция `transform_xy`, по умолчанию она равна `false`. Передавать ей нужно выражение, сгенерированное функцией

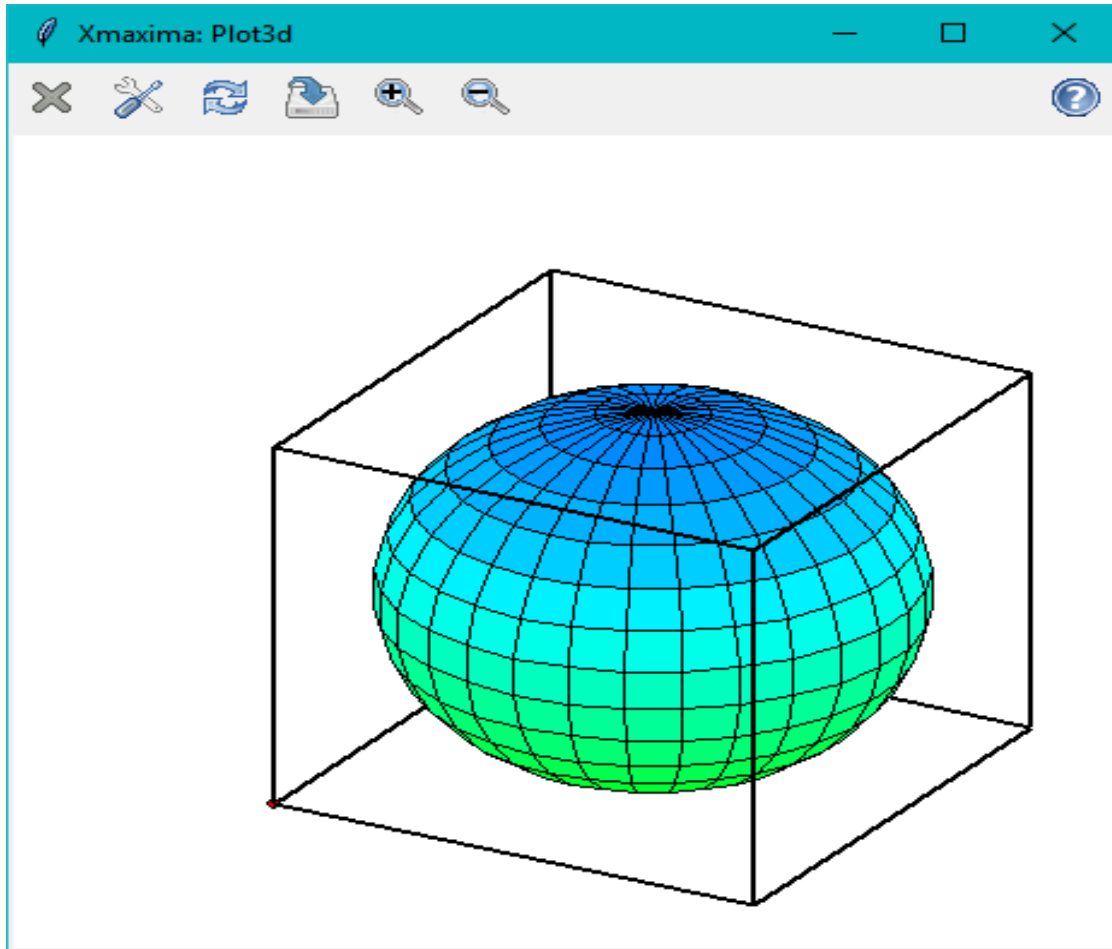
```
make_transform([x, y, z], f1(x, y, z),
               f2(x, y, z), f3(x, y, z)).
```

Кроме того, существует одно встроенное преобразование, известное как `polar_xy` и соответствующее

`make_transform([r, th, z], r*cos(th), r*sin(th), z)`,
то есть переходу к полярной цилиндрической системе координат.

В качестве примера использования `transform_xy` рассмотрим преобразование к полярным сферическим координатам (во встроенном виде его нет):

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



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

Если нужно постоянно работать со сферическими координатами, можно задать, например:

```
spherical_xy:make_transform([t, f, r],
  r*sin(f)*sin(t), r*cos(f)*sin(t), r*cos(t))
```

и затем при построении графиков писать
`[transform_xy, spherical_xy]`.

2.10 Draw графика

2.10.1 Сцены

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

Функция **gr2d** создает объект, описывающий 2D-сцену. Аргументы - это графические параметры, графические объекты или списки, содержащие как графические параметры, так и объекты (*graphic options, graphic objects*). Эта сцена интерпретируется последовательно: графические параметры влияют на те графические объекты, которые расположены справа. Некоторые графические параметры влияют на глобальное появление сцены.

Это список графических объектов (*graphic objects*), доступных для сцен в двух измерениях: - *bars, ellipse, explicit, image, implicit, label, parametric, points, polar, polygon, quadrilateral, rectangle, triangle, vector, and geomap* (this one defined in package *worldmap*).

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

Функция **gr3d** создает объект, описывающий 3D-сцену. Аргументы - это графические параметры, графические объекты или списки, содержащие как графические параметры, так и объекты. Эта сцена интерпретируется последовательно: графические параметры влияют на те графические объекты, которые расположены справа. Некоторые графические параметры влияют на глобальное появление сцены.

Это список графических объектов, доступных для сцен в трех измерениях: *cylindrical, elevation_grid, explicit, implicit, label, mesh, parametric, parametric_surface, points, quadrilateral, spherical, triangle, tube, vector, and geomap*.

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

2.10.2 Основные функции и примеры

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

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

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

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

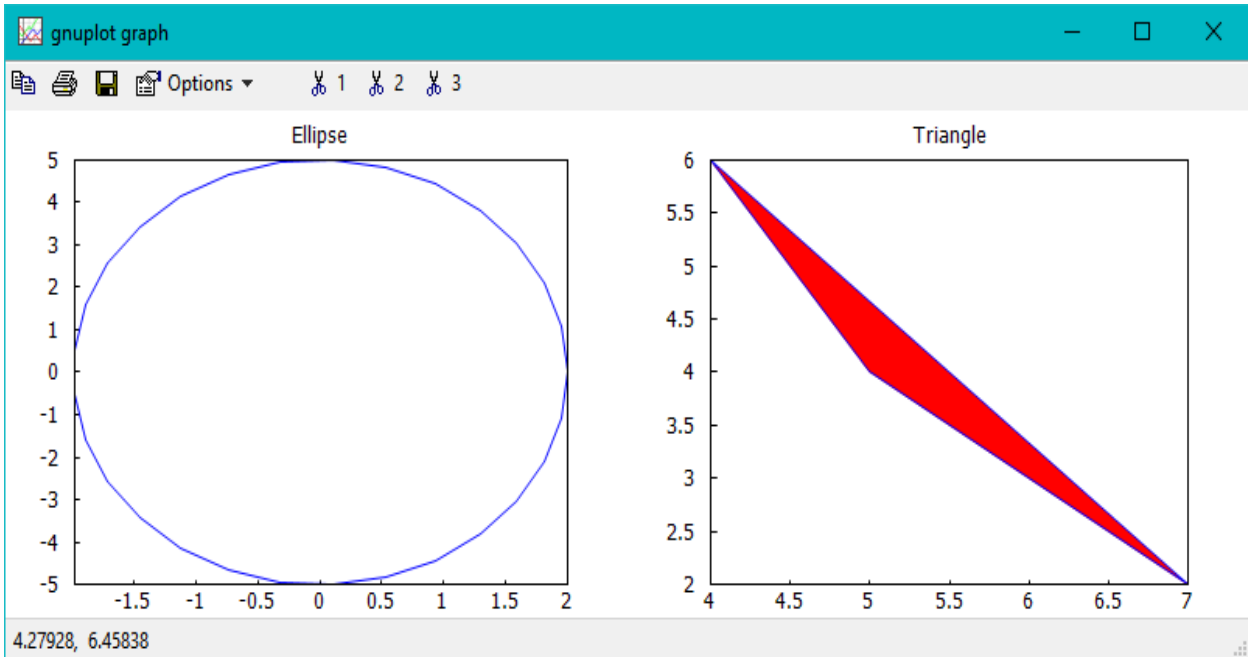
Примеры:

```
kill(all);
load(draw) $
scene1: gr2d(title="Ellipse",
```

```

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

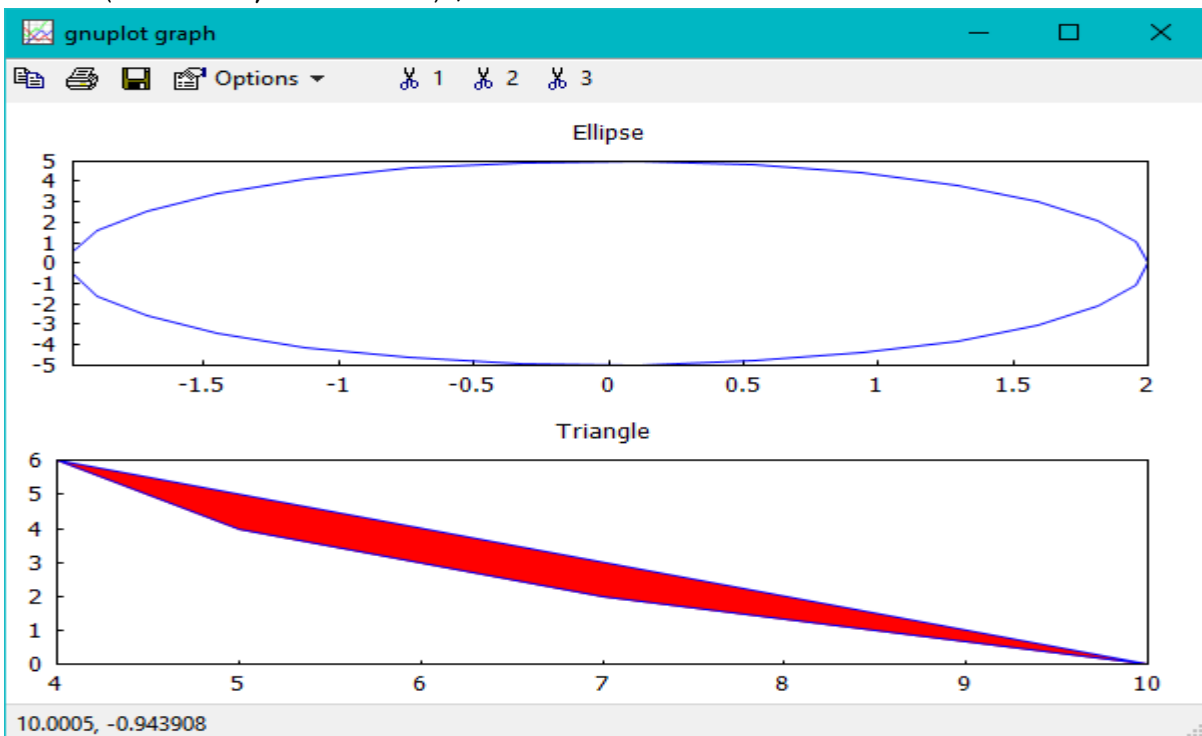
```



```

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

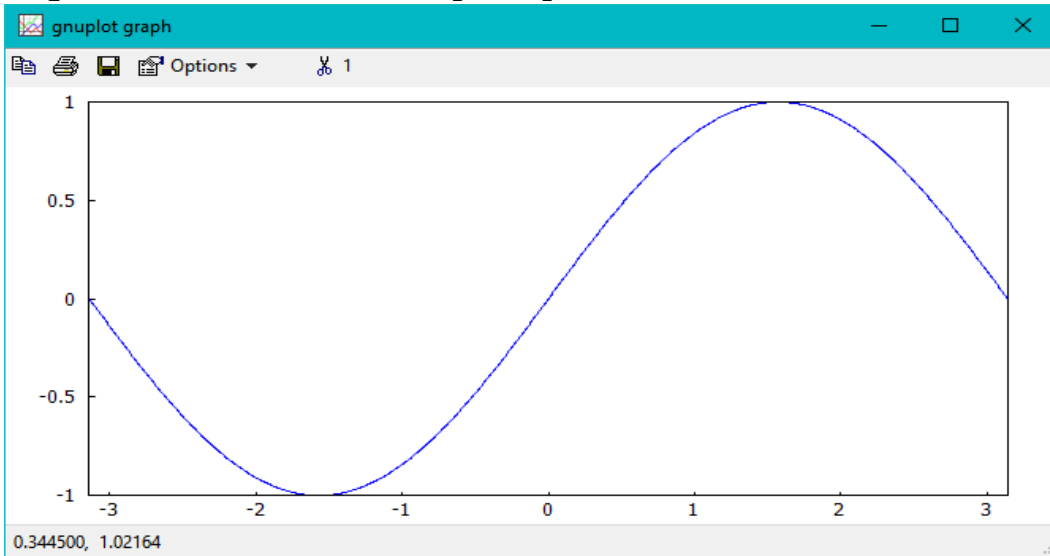
```



Построение простой (явной) функции при помощи `draw2d`:

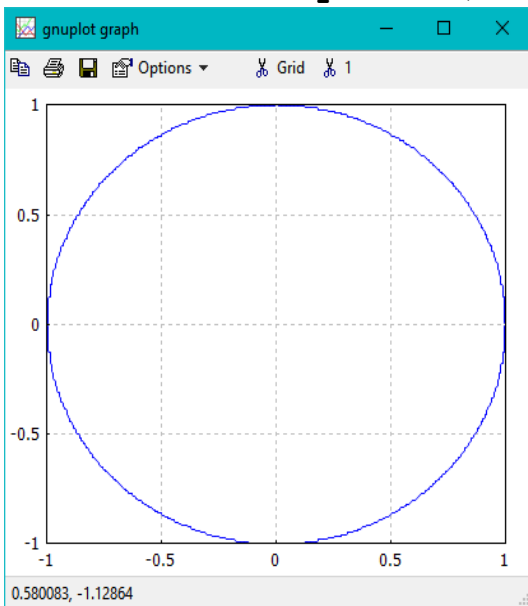
```
draw2d(nticks=300, explicit(sin(x), x, -%pi, %pi));
```

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



Построение неявной функции (окружности). Уравнение функции задается через функцию `implicit`:

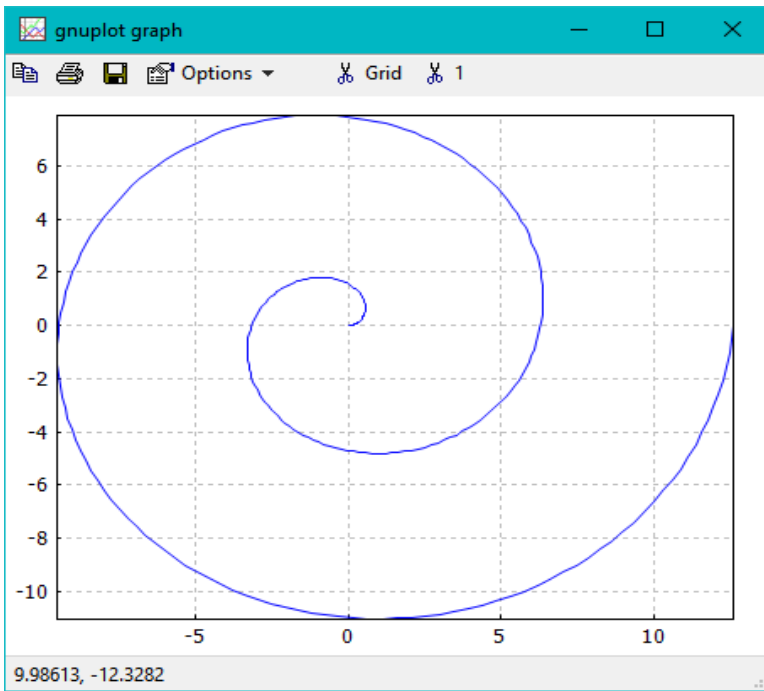
```
draw2d(nticks=300, grid=true,
        implicit(x^2+y^2=1, x, -1, 1, y, -1, 1));
```



Использование функции `draw2d` для построения графика функции в полярных координатах:

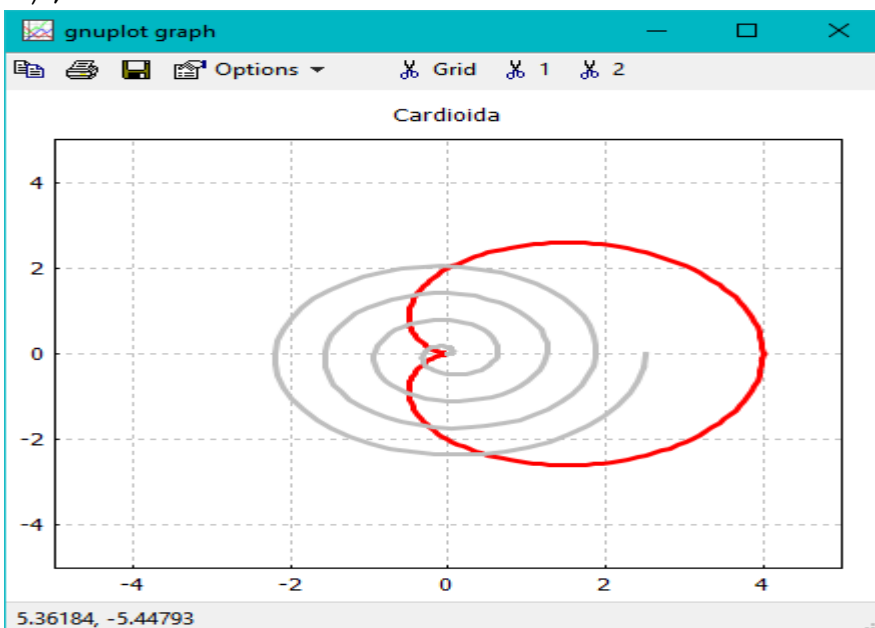
```
draw2d(nticks=300, grid=true, polar(t, t, 0, 4*%pi));
```

Здесь функция задается, как функция радиуса $r(t)$. Функция конструируется в виде вызова `polar(t, t, 0, 4*%pi)`, у которой первый параметр - это функция $r(t)$, второй параметр - это обозначение полярного угла t , далее границы области определения угла.



Можно рисовать несколько графиков:

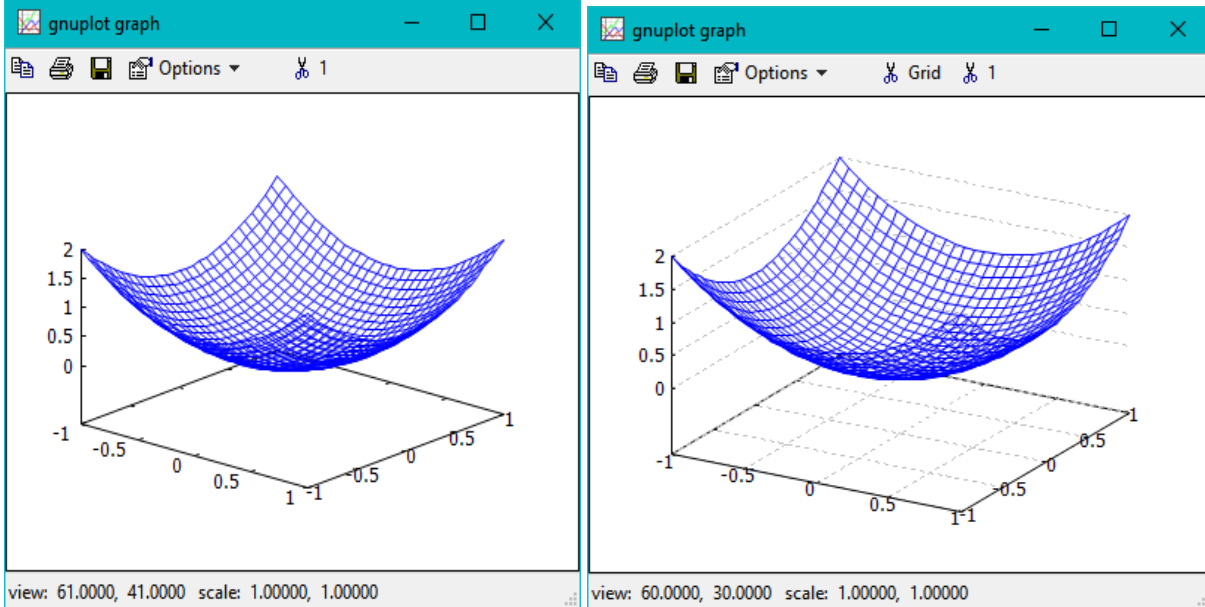
```
load(draw);
draw2d(user_preamble="set grid",
  nticks = 200,
  xrange = [-5,5],
  yrange = [-5,5],
  color = red,
  line_width = 3,
  title = "Cardioida",
  polar ( 2 * (1+cos(theta)),theta, 0, 2*pi),
  color=grey,
  polar ( 0.1*theta,theta, 0, 8*pi)
);
```



Пример построения поверхности двумя способами.

Функция $Z=f(x, y)$ конструируется при помощи вызова `explicit(x^2+y^2, x, -1, 1, y, -1, 1)`. Второй график рисуется с «координатной сеткой»:

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

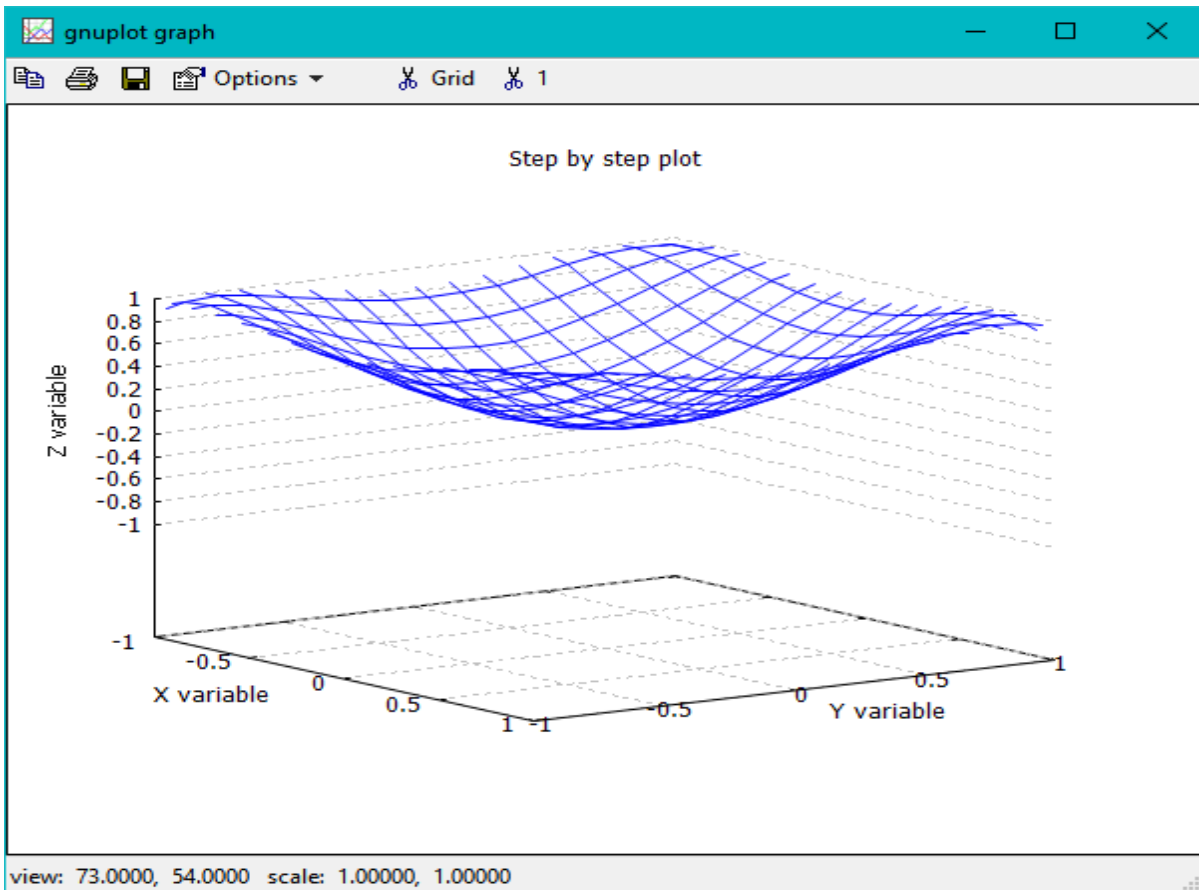


Изображение графика можно «анимировать» и записать в файл:

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

Функция **draw3d** позволяет строить трёхмерные графики. Пример:

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



3. Задачи высшей математики с Maxima

3.1 Операции с комплексными числами

3.1.1 Представление комплексных чисел

Значение целой положительной степени комплексного аргумента проще всего вычислять в тригонометрической форме. Если $z = x + iy = r(\cos(\varphi) + i \sin(\varphi))$ (здесь $r = \sqrt{x^2 + y^2}$ — модуль

комплексного числа, $\varphi = \operatorname{arctg} \frac{y}{x}$ — его аргумент), то для любого целого положительного числа n имеет место формула:
 $w = f(z) = z^n = r^n(\cos(n\varphi) + i \sin(n\varphi))$.

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 $f(z) = \sqrt[n]{z}$ также удобно вычислять в тригонометрической форме. Если $z = x + iy = r(\cos(\varphi) + i \sin(\varphi))$, то для любого целого положительного числа n имеет место формула:

$$f(z) = \sqrt[n]{z} = \sqrt[n]{r(\cos \varphi + i \sin \varphi)} = \sqrt[n]{r} \left(\cos \frac{\varphi + 2k\pi}{n} + i \sin \frac{\varphi + 2k\pi}{n} \right), \text{ т.е.}$$

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

$$f(z) = \exp(z) \text{ вычисляются по формуле}$$

$$f(z) = e^z = e^{x+iy} = e^x(\cos(y) + i \sin(y)).$$

Логарифмом комплексного числа z называется такое число w , что $e^w = z$.

Значения логарифмической функции $f(z) = Ln(z)$ вычисляются по формуле $Ln(z) = \ln(|z|) + iArgz = \ln(|z|) + iargz + 2k\pi$, $k = 0, 1, 2, \dots$

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 и произведения i (мнимой единицы) 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 - 4x + 13 = 0
(%i2) solve(eq,x);
(%o2)      [x = 2 - 3i, x = 3i + 2]
(%i3) x1:%o2[1] $ x2:%o2[2];
(%o3)      x = 3i + 2
(%i4) print(x1,x2);
(%o4)      x = 2 - 3i, x = 3i + 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 =  $e^{\frac{2i\pi}{5}}$ , x =  $e^{\frac{4i\pi}{5}}$ , x =  $e^{-\frac{4i\pi}{5}}$ , x =  $e^{-\frac{2i\pi}{5}}$ , 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.

3.1.2 Функции для работы с комплексными числами

Упрощение частных, корней, и других функций комплексных выражений может обычно достигаться при использовании функций *realpart*, *imagpart*, *rect form*, *polar form*, *abs*, *carg*.

Вычисление модуля комплексного числа осуществляется функцией *abs*. Аргумент комплексного выражения вычисляется при помощи функции *carg*. Комплексный аргумент — θ в пределах $[-\pi, \pi]$ таким образом, что $r \exp(\theta i) = z$, где r — модуль комплексного числа z . Следует учитывать, что *carg* — вычислительная функция, не предназначенная для упрощения комплексных выражений. (в некоторых случаях удобно использовать опцию *numer*, установка которой заставляет представлять результаты в формате с плавающей точкой — см. пример ниже).

Пример:

```
(%i1) carg (1);
(%o1) 0
(%i2) carg (1 + %i);
(%o2)  $\frac{\pi}{4}$ 
(%i3) carg (exp (%i)), numer;
(%o3) 1.0
(%i4) carg (exp (%pi * %i));
(%o4)  $\pi$ 
(%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: e^{a+ib} упрощает к виду $e^a \cdot (\cos(b) + i \cdot \sin(b))$, 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) -e3i
(%i3) demoivre (exp (%pi+3/2 * %pi * %i));
(%o3) -eπi
```

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 чисто мнимые (*imaginary*).

Функция *plog(x)* представляет основную ветвь комплексного логарифма, соответствующую $-\pi < \text{carg}(x) \leq +\pi$, например:

```
(%i1) a:1+%i;
(%o1) i + 1
(%i2) plog(a);
(%o2)  $\frac{\log(2)}{2} + \frac{i\pi}{4}$ 
```

Функция *polar form(expr)* возвращает выражение $r e^{i\theta}$, эквивалентное *expr* (параметры *r* и *θ* действительны).

Преобразование комплексного выражения к алгебраической форме осуществляется функцией *rect form(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, z₀)* вычисляет остаток в комплексной плоскости для выражения *expr*, когда переменная *z* принимает значение *z₀*. 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) $-\frac{a^3}{6}$

3.2 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.

3.2.1 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-множителей должны быть согласованы.

Рассмотрим несколько примеров.

Создание двух прямоугольных матриц:

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

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

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

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

Функция *transpose* транспонирует матрицу:

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

(%o1) (1 2 3) (%o2) $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

Умножение матрицы на число:

```
(%i2) c:b*2;
```

(%o2) $\begin{pmatrix} 4 & 4 \\ 6 & 6 \\ 8 & 8 \end{pmatrix}$

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

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


$$(\%04) \begin{pmatrix} 6 & 6 \\ 9 & 9 \\ 12 & 12 \end{pmatrix}$$

(%i5) a+b;

fullmap: arguments must have same formal structure.
 - an error. To debug this try: debugmode(true);

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

(%i6) f:a.b;

$$(\%06) \begin{pmatrix} 20 & 20 \\ 47 & 47 \end{pmatrix}$$

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

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

3.2.2 Обращение матриц и вычисление определителей

Для обращения матриц используется функция *invert*. Пример:

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

b:invert(a);

b.a;

$$(\%01) \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad (\%02) \begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix} \quad (\%03) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Определитель вычисляется функцией *determinant*:

(%i4) determinant(a);

$$(\%04) -2$$

3.2.3 Характеристический полином, собственные числа и собственные векторы матрицы

Характеристический полином матрицы вычисляется функцией *charpoly*(M, x) (M — матрица, x — переменная, относительно которой строится полином).

Пример:

(%i6) charpoly(a, x);

$$(\%06) (1-x)(4-x)-6$$

(%i7) ratsimp(%);

$$(\%07) x^2 - 5x - 2$$

Корни характеристического полинома являются собственными числами матрицы.

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

Функция *eigenvectors* аналитически вычисляет собственные значения и собственные вектора матрицы, если это возможно. Она возвращает список, первый элемент которого — список собственных 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) 
$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$$

(%i2) eigenvalues(a);
(%o2) [[0,6],[2,1]]
(%i3) eigenvectors(a);
(%o3) [[[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.

3.2.4 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 подписки 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],[ $\frac{2^2 3}{7}$ ,  $\frac{3}{7}$ ,  $-\frac{23}{7}$ ]]
(%i4) ratsimp(%[1].%[2]);
```

(%04) 0

3.2.5 Преобразование матрицы к треугольной форме

Преобразование матрицы к треугольной форме осуществляется методом исключения Гаусса посредством функции *echelon*(*M*) (аналогичный результат даёт 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.

3.2.6 Вычисление ранга и миноров матрицы

Для расчёта ранга матрицы (порядка наибольшего 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.

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

(%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

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

3.2.7 Решение матричных уравнений

Пусть дано матричное уравнение $AX = B$, где A — квадратная матрица размерности n ; B — матрица размерности $n \times k$; X — неизвестная матрица размерности $n \times k$. 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 уравнения $AX = B$, 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) \begin{pmatrix} 1 & 2 & 2 \\ -1 & -1 & 3 \\ 2 & 5 & 0 \end{pmatrix}
```

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

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

Проверим существование и единственность решения:

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

```
(%o3) -9
```

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

```
(%i4) A1:invert(A); x:A1.B;
```

```
(%o4) \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} \quad (%o5) \begin{pmatrix} 18 & -\frac{82}{9} \\ -7 & \frac{40}{9} \\ 3 & \frac{1}{9} \end{pmatrix}
```

Выполним проверку:

```
(%i6) A.x-B;
```

```
(%o6) \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}
```

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

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

```
(%i10) C:matrix([10,0,-2],[5,1,4]); x:C.A1; x.A-C;
(%o10)  $\begin{pmatrix} 10 & 0 & -2 \\ 5 & 1 & 4 \end{pmatrix}$ 
(%o11)  $\begin{pmatrix} 16 & -\frac{34}{3} & -\frac{26}{3} \\ 9 & -\frac{14}{3} & -\frac{13}{3} \end{pmatrix}$  (%o12)  $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ 
```

В общем случае (когда A — вырожденная матрица, или A — не квадратная матрица) матричное уравнение $AX = B$ можно решить при помощи функции *solve*.

Синтаксис вызова: $\text{solve}([eq_1, eq_2, \dots, eq_n], [x_1, x_2, \dots, x_m])$, где $[eq_1, eq_2, \dots, eq_n]$ — список уравнений, $[x_1, x_2, \dots, x_m]$ — список неизвестных, относительно которых осуществляется решение.

3.2.8 Специальные функции для решения систем линейных и полиномиальных уравнений

Функция $\text{linsolve}([expr_1, expr_2, \dots, expr_m], [x_1, x_2, \dots, x_n])$ решает список одновременных линейных уравнений $[expr_1, expr_2, \dots, expr_m]$ относительно списка переменных $[x_1, \dots, x_n]$.

Выражения $[expr_1, \dots, expr_m]$ 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]);
(%o1) z + y + x + t = 6
(%o2) z - 2y + 2x + 3t = 2
(%o3) 2z - y + 3x - t = 8
(%o4) [x = -\frac{3\%r1 - 14}{4}, y = -\frac{\%r1 - 10}{4}, z = \%r1, t = 0]
```

Таким образом общее решение имеет вид:

$$x = (14 - 3c)/4, y = (10 - c)/4, z = c, t = 0,$$

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 = 11/4, y = 9/4, z = 1, t = 0]
```

Способ представления решения зависит от флага *linsolve_params* (по умолчанию *true*). Если указанный флаг установлен в *true*, решение недоопределённых систем включает параметры 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 $algsys([expr_1, expr_2, \dots, expr_m], [x_1, x_2, \dots, x_n])$ решает систему $[expr_1 = 0, expr_2 = 0, \dots, expr_m = 0]$ полиномиальных уравнений относительно списка переменных $[x_1, \dots, x_n]$.

Выражения $[expr_1, \dots, expr_n]$ могут быть представлены и в виде уравнений. Количество уравнений может превышать количество неизвестных и 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);
```

```
(%o6) 2(1 - a1)x - 2a2(x - 1)
```

```
(%o7) a2 - a1
```

```
(%o8) a1(-y - x^2 + 1)
```

```
(%o9) a2(y - (x - 1)^2)
```

```
(%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);
(%o1)  x4 + 1
(%o2)  [x = 0.70710678118655%i + 0.70710678118655,
x = 0.70710678118655 - 0.70710678118655%i,
x = 0.70710678118655%i - 0.70710678118655,
x = -0.70710678118655%i - 0.70710678118655]
```

Количество действительных 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)      x2 + x + 1 = (x - 1)3
(%i2) lhs (eqn);
(%o2)      x2 + x + 1
(%i3) rhs (eqn);
(%o3)      (x - 1)3
```

Упрощение систем уравнений достигается функцией *eliminate*, позволяющей исключить те или иные переменные.

Вызов $eliminate([eqn_1, \dots, eqn_n], [x_1, \dots, x_k])$ исключает переменные $[x_1, \dots, x_k]$ 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;
(%o1)          z + x y + 2 x^2
(%o2)          -z + 5y + 3x - 1
(%o3)          z^2 - y^2 + x + 5
(%i4) eliminate ([expr3, expr2, expr1], [y, z]);
(%o4) [7425x^8 - 1170x^7 + 1299x^6 + 12076x^5 + 22887x^4 - 5154x^3 - 1291x^2 + 7688x + 15376]
```

3.3 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 уравнением $x^3 + y^2 - x = 0$. 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:

1. степенная функция $y = x^r, r \in R$;
2. показательная функция $y = a^x (a > 0, a \neq 1)$
3. логарифмическая функция $y = \log_a x (a > 0, a \neq 1)$;
4. тригонометрические функции
 $y = \sin x, y = \cos x, y = \operatorname{tg} x, y = \operatorname{ctg} x$;
5. обратные тригонометрические функции
6. $y = \arcsin x, y = \arccos x, y = \operatorname{arctg} x, y = \operatorname{arcctg} x$

Из основных элементарных функций новые элементарные функции могут быть получены при помощи:

1. алгебраических действий;
2. 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, функция

$$y = \frac{\sqrt{x} + \arcsin x^5}{\ln^3 x + x^3 + x^7}$$

является элементарной.

Примером неэлементарной функции является функция $y = \operatorname{sign} x$.

3.3.1 Основные свойства функций

- **Четность и нечетность.** Функция $y = f(x)$ называется четной, если $f(-x) = f(x)$ и нечетной, если $f(-x) = -f(x)$. В противном случае функция 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.
- **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).

- **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 $x \in \mathbb{R}$.
- **Периодичность.** Функция $y = f(x)$ называется периодической с периодом $T \neq 0$ на промежутке X , для любого $x \in X$ выполняется равенство $f(x + T) = f(x)$.

3.3.2 Предел функции и его свойства

3.3.2.1 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:

$$\lim_{x \rightarrow \infty} f(x) = A$$

или $f(x) \rightarrow A$ при $x \rightarrow \infty$.

3.3.2.2 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 $x \neq a$.

Предел функции обозначается следующим образом:

$$\lim_{x \rightarrow a} f(x) = A$$

или $f(x) \rightarrow A$ при $x \rightarrow a$.

3.3.2.3 Односторонние пределы.

Если $x > a$ и $x \rightarrow a$, то употребляют запись $x \rightarrow a + 0$. Если $x < a$ и $x \rightarrow a$, то употребляют запись $x \rightarrow a - 0$.

Выражения $\lim_{x \rightarrow a+0} f(x)$ и $\lim_{x \rightarrow a-0} f(x)$ называются соответственно пределами функции $f(x)$ в точке a справа и слева.

Если существует предел $\lim_{x \rightarrow a} f(x)$, то существуют и односторонние пределы $\lim_{x \rightarrow a+0} f(x)$ и $\lim_{x \rightarrow a-0} f(x)$ и

$$\lim_{x \rightarrow a-0} f(x) = \lim_{x \rightarrow a} f(x) = \lim_{x \rightarrow a+0} f(x).$$

Это равенство выполняется также, если пределы слева и справа равны¹.

3.3.2.4 Теоремы о пределах

1. Предел суммы двух функций равен сумме пределов этих функций, если те существуют, то есть

$$\lim_{x \rightarrow x_0} [f(x) + \psi(x)] = A + B,$$

где $A = \lim_{x \rightarrow x_0} f(x)$, $B = \lim_{x \rightarrow x_0} \psi(x)$.

2. Предел произведения двух функций равен произведению пределов этих функций.

$$\lim_{x \rightarrow x_0} [f(x) \cdot \psi(x)] = A \cdot B.$$

3. Предел частного двух функций равен частному пределов этих функций.

$$\lim_{x \rightarrow x_0} \frac{f(x)}{\psi(x)} = \frac{A}{B},$$

причем $B \neq 0$.

4. Если,

$$\lim_{u \rightarrow u_0} f(u) = A;$$

$$\lim_{x \rightarrow x_0} \psi(x) = u_0,$$

то предел сложной функции

$$\lim_{x \rightarrow x_0} f[\psi(x)] = A.$$

3.3.2.5 Вычисление пределов различных классов функций

Предел выражения $f(x)$ при $x \rightarrow a$ вычисляется с помощью функции $\text{limit}(f(x), x, a)$;

Рассмотрим пример: вычислить предел $\lim_{x \rightarrow 0} \frac{\sin x}{x}$.

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:

$\lim_{x \rightarrow +\infty} e^x$, $\lim_{x \rightarrow -\infty} e^x$, $\lim_{x \rightarrow 0-0^x} \frac{1}{x}$, $\lim_{x \rightarrow 0+0^x} \frac{1}{x}$.

Предел неограниченной функции на бесконечности:

```
(%i2) limit(exp(x), x, inf);
```

```
(%o2) inf
```

```
(%i3) limit(exp(x), x, minf);
```

```
(%o3) 0
```

Пределы при $x \rightarrow 0$ слева и справа:

```
(%i3) limit(1/x, x, 0, minus);
```

```
(%o3) -inf
```

```
(%i4) limit(1/x, x, 0, plus);
```

```
(%o4) inf
```

3.3.2.6 Предел и непрерывность функции

Вычислить пределы $\lim_{x \rightarrow 8} \sqrt[3]{x}$ и $\lim_{x \rightarrow -8} \sqrt[3]{x}$.

```
(%i8) limit(x^(1/3), x, 8);
```

```
(%o8) 2
```

```
(%i9) limit(x^(1/3), x, -8);
```

```
(%o9) -2
```

3.3.2.7 Пределы рациональных дробей

Вычислить предел $\lim_{x \rightarrow -1} \frac{x^3 - 3x - 2}{(x^2 - x - 2)^2}$.

```
(%i10) y(x) := (x^3 - 3*x - 2) / (x^2 - x - 2)^2; limit(y(x), x, -1);
```

```
(%o10) y(x) := (x^3 - 3x - 2) / (x^2 - x - 2)^2
```

$$(\%o11) \quad -\frac{1}{3}$$

При операциях с рациональными дробями и выделения носителей нуля² целесообразно использовать факторизацию выражений, например: вычисление предела непосредственно

```
(%i16) limit((x^2-4)/(x^2-3*x+2), x, 2);
```

$$(\%o16) \quad 4$$

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 $x \rightarrow 2$, т.е. выражение $x - 2$.

$$(\%o17) \quad \frac{x+2}{x-1}$$

```
(%i18) limit(%, x, 2);
```

$$(\%o18) \quad 4$$

3.3.2.8 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 $x \rightarrow 1$. При вычислении предела непосредственно имеем:

```
(%i1) limit((sqrt(x)-1)/(x-1), x, 1);
```

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

Для упрощения и сокращения носителей нуля используется функция *radcan*:

```
(%i2) factor((sqrt(x)-1)/(x-1));
```

$$(\%o2) \quad \frac{\sqrt{x}-1}{x-1}$$

```
(%i3) radcan(%);
```

$$(\%o3) \quad \frac{1}{\sqrt{x}+1}$$

```
(%i4) limit(%, x, 1);
```

$$(\%o4) \quad \frac{1}{2}$$

3.3.2.9 Пределы тригонометрических выражений

Первым замечательным пределом называется предел

$$\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1.$$

Рассмотрим примеры нахождения некоторых пределов с использованием первого замечательного предела.

Пример. Найти предел

$$\lim_{x \rightarrow 0} \frac{\sin 5x}{x}.$$

$$\lim_{x \rightarrow 0} \frac{\sin 5x}{x} = \lim_{x \rightarrow 0} 5 \frac{\sin 5x}{5x} = 5 \lim_{t \rightarrow 0} \frac{\sin t}{t} = 5,$$

где $t = 5x$.

Расчёт с использованием **Maxima**:

```
(%i1) limit(sin(5*x)/x, x, 0);
(%o1) 5
```

Пример. Найти предел $\lim_{x \rightarrow 0} \frac{1 - \cos 2x}{x^2}$.

$$\lim_{x \rightarrow 0} \frac{1 - \cos 2x}{x^2} = \lim_{x \rightarrow 0} \frac{2 \sin x^2}{x^2} = \lim_{t \rightarrow 0} \frac{2 \sin t}{t} = 2,$$

где $t = x^2$.

Расчёт с использованием **Maxima**:

```
(%i4) limit((1-cos(2*x))/x^2, x, 0);
(%o4) 2
```

3.3.2.10 Пределы экспоненциальных выражений

Вторым замечательным пределом называется предел

$$\lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x = e = 2.718281828 \dots$$

Можно показать, что функция

$$y(x) = \left(1 + \frac{1}{x}\right)^x$$

при $x \rightarrow +\infty$ и при $x \rightarrow -\infty$ также имеет предел, равный e .

$$e = \lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x.$$

Заменяя x на $x = 1/t$ получим еще одну запись числа e

$$e = \lim_{t \rightarrow 0} (1 + t)^{1/t}.$$

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 называется 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:

$$\operatorname{sh}(x) = \frac{e^x - e^{-x}}{2};$$

$$\operatorname{ch}(x) = \frac{e^x + e^{-x}}{2};$$

$$\operatorname{th}(x) = \frac{\operatorname{sh}(x)}{\operatorname{ch}(x)}.$$

Рассмотрим примеры нахождения некоторых пределов с использованием второго замечательного предела.

Пример. Найти предел $\lim_{x \rightarrow 0} \frac{\ln(1+x)}{x}$.

$$\lim_{x \rightarrow 0} \frac{\ln(1+x)}{x} = \lim_{x \rightarrow 0} \ln(1+x)^{1/x} = \lim_{x \rightarrow 0} \ln e = 1.$$

$$\text{Итак } \lim_{x \rightarrow 0} \frac{\ln(1+x)}{x} = 1.$$

Пример. Найти предел $\lim_{x \rightarrow 0} \frac{a^x - 1}{x}$.

Пусть $a^x - 1 = u$. Тогда

$$a^x = 1 + u; \quad x = \frac{\ln(1+u)}{\ln a}$$

$$\lim_{x \rightarrow 0} \frac{a^x - 1}{x} = \lim_{u \rightarrow 0} \frac{u \ln a}{\ln(1+u)} = \ln a \cdot \lim_{u \rightarrow 0} \frac{u}{\ln(1+u)} = \ln a \cdot 1 = \ln a.$$

$$\text{Итак } \lim_{x \rightarrow 0} \frac{a^x - 1}{x} = \ln a.$$

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)
```

$$\lim_{x \rightarrow \infty} \left(\frac{x}{2+x} \right)^{3x}.$$

Найдем предел $\lim_{x \rightarrow \infty} \left(\frac{x}{2+x} \right)^{3x}$. Аналитический расчёт даёт следующий результат:

$$\lim_{x \rightarrow \infty} \left(\frac{x}{2+x} \right)^{3x} = \exp \left[\lim_{x \rightarrow \infty} \left(\frac{x}{2+x} - 1 \right) 3x \right] = e^{-6}.$$

Используя **Maxima**, получаем:

```
(%i7) limit((x/(2+x))^(3*x), x, inf);
```

```
(%o7) e^-6
```

3.3.2.11 Бесконечно малые и бесконечно большие функции

Сравнение бесконечно малых функций.

Рассмотрим предел частного от деления двух бесконечно малых $\alpha(x)$ и $\beta(x)$ при $x \rightarrow a$.

$$A = \lim_{x \rightarrow a} \frac{\alpha(x)}{\beta(x)}$$

Предел отношения двух бесконечно малых величин 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 $x \rightarrow a$.
Если $A = 1$, то $\alpha(x)$ и $\beta(x)$ называют эквивалентными и пишут $\alpha(x) \sim \beta(x)$ при $x \rightarrow a$.
2. Если $A = 0$, то $\alpha(x)$ называют бесконечно малой более высокого порядка, чем $\beta(x)$ и пишут $\alpha(x) = o[\beta(x)]$ при $x \rightarrow a$.
Если существует действительное число $r > 0$ такое, что
3. $\lim_{x \rightarrow a} \frac{\alpha(x)}{[\beta(x)]^r} \neq 0$
то $\alpha(x)$ называют бесконечно малой порядка r относительно $\beta(x)$ при $x \rightarrow a$.
4. Если $A \rightarrow \infty$ при $x \rightarrow a$, 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 стремится ни к какому пределу, например, если взять $\alpha = x$ и $\beta = x \sin \frac{1}{x}$, то их отношение, равное $\sin \frac{1}{x}$, при $x \rightarrow 0$ 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 $x \rightarrow 0$

```
(%i26) f(x) := sin(3*x) * sin(5*x) $ g(x) := (x-x^3)^2 $
```

Вычислим предел отношения $f(x)/g(x)$ при $x \rightarrow 0$

```
(%i28) limit(f(x)/g(x), x, 0);
```

```
(%o28)
```

15

Результат, равный постоянному числу, свидетельствует о том, что рассматриваемые бесконечно малые одного порядка.

3.3.2.12 Эквивалентные бесконечно малые. Их применение к вычислению пределов.

При вычислении пределов полезно иметь в виду эквивалентность следующих бесконечно малых величин:
 $\sin x \sim x$; $\operatorname{tg} x \sim x$; $\arcsin x \sim x$; $\operatorname{arctg} x \sim x$; $\ln(1+x) \sim x$, при $x \rightarrow 0$.

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 $x \rightarrow 0$.

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

$$\lim_{x \rightarrow 0} \frac{\alpha(x)}{\beta(x)} = \lim_{x \rightarrow 0} \frac{x^2 \sin^2 x}{x \operatorname{tg} x} = \lim_{x \rightarrow 0} \frac{x^2 \cdot x^2}{x \cdot x} = \lim_{x \rightarrow 0} \frac{x^4}{x^2} = \lim_{x \rightarrow 0} x^2 = 0.$$

Таким образом, $\alpha(t) = o[\beta(t)]$ при $t \rightarrow 0$. 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 $x \rightarrow 0$.

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

$$\lim_{x \rightarrow 0} \frac{\alpha(x)}{\beta(x)} = \lim_{x \rightarrow 0} \left[\frac{1}{x} \sin \left(\frac{x}{\sqrt{x+1} + 1} \right) \right] = \frac{1}{2}.$$

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 \rightarrow 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) f(x):= \frac{\tan(x) - \sin(x)}{x - \sin(x)}
```

```
(%i2) tlimit(f(x), x, 0);
```

(2)

3

3.3.2.13 Бесконечно большие функции.

Связь между бесконечно малыми и бесконечно большими величинами.

Функция $f(x)$ называется бесконечно большой величиной при $x \rightarrow a$, 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 $x \rightarrow a$ означает следующее: $\lim_{x \rightarrow a} f(x) = \infty$ или $f(x) \rightarrow \infty$ при $x \rightarrow a$.

Пример: $y = \operatorname{tg} x$ бесконечно большая при $x \rightarrow \pi/2$.

Замечание: Функция может быть неограниченной, но не бесконечно большой. Например, функция $y = x \sin x$ — не ограничена на $(-\infty, \infty)$, но не бесконечно большая при $x \rightarrow \infty$.

Если функция $\alpha(x)$ есть бесконечно малая величина при $x \rightarrow a$ ($x \rightarrow \infty$), то функция $f(x) = \frac{1}{\alpha(x)}$ является бесконечно большой при $x \rightarrow a$ ($x \rightarrow \infty$).

И обратно, если функция $f(x)$ бесконечно большая при $x \rightarrow a$ ($x \rightarrow \infty$), то функция $\alpha(x) = \frac{1}{f(x)}$ есть величина бесконечно малая при $x \rightarrow a$ ($x \rightarrow \infty$).

Например, функция $y = \cos x$ — бесконечно малая при $x \rightarrow \pi/2$, тогда функция $\frac{1}{\cos x}$ — бесконечно большая. Функция $y = \frac{1}{2x - 7}$ — бесконечно малая при $x \rightarrow \infty$, тогда функция $y = 2x - 7$ — бесконечно большая при $x \rightarrow \infty$.

3.3.3 Непрерывные функции

Понятие непрерывности функции, так же как и понятие предела, является одним из основных понятий математического анализа.

3.3.3.1 Непрерывность функции в точке

Дадим два определения понятия непрерывности функции в точке.

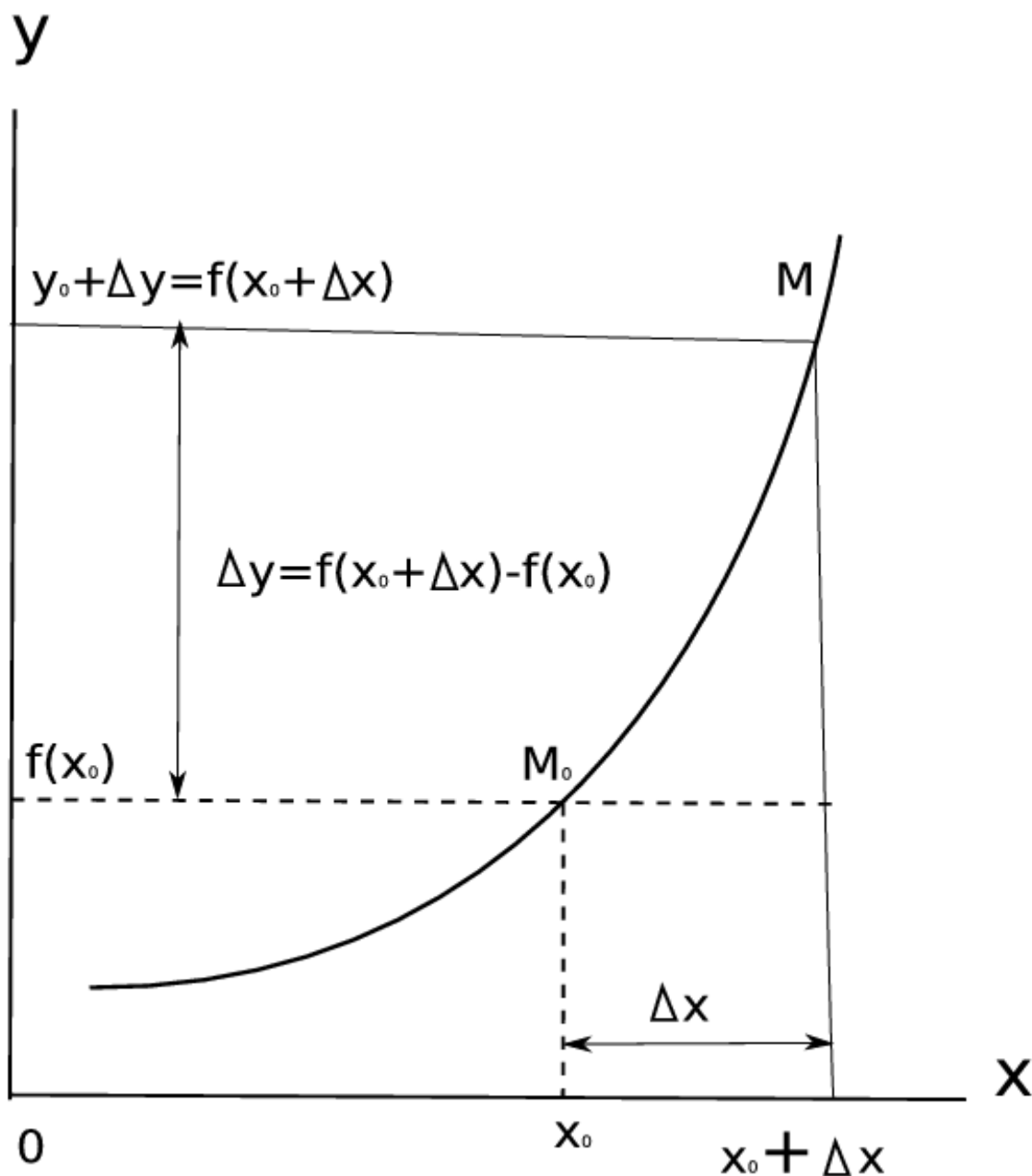


Рис. 3.1. Определение непрерывной функции

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 $\lim_{x \rightarrow a} f(x)$ этот предел равен значению функции $f(x)$ в точке a , т.е. $\lim_{x \rightarrow a} f(x) = f(a)$. Очевидно, что непрерывность функции в данной точке выражается непрерывностью ее графика при прохождении данной точки.

Рассмотрим 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. 3.1).

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: $\lim_{\Delta x \rightarrow 0} \Delta y = 0$.

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) := 1 / (1 + exp(1/(1-x)))
```

```
(%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.

3.3.3.2 Свойства непрерывных функций

1. Если функции $f(x)$ 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 x_0 .

Свойство 3 может быть записано в виде:

$$\lim_{x \rightarrow x_0} f[\psi(x)] = f \left[\lim_{x \rightarrow x_0} \psi(x) \right],$$

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.

3.3.3.3 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 $f(a-0) = \lim_{x \rightarrow a-0} f(x)$ и $f(a+0) = \lim_{x \rightarrow a+0} f(x)$, 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, 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 $x = 1$.

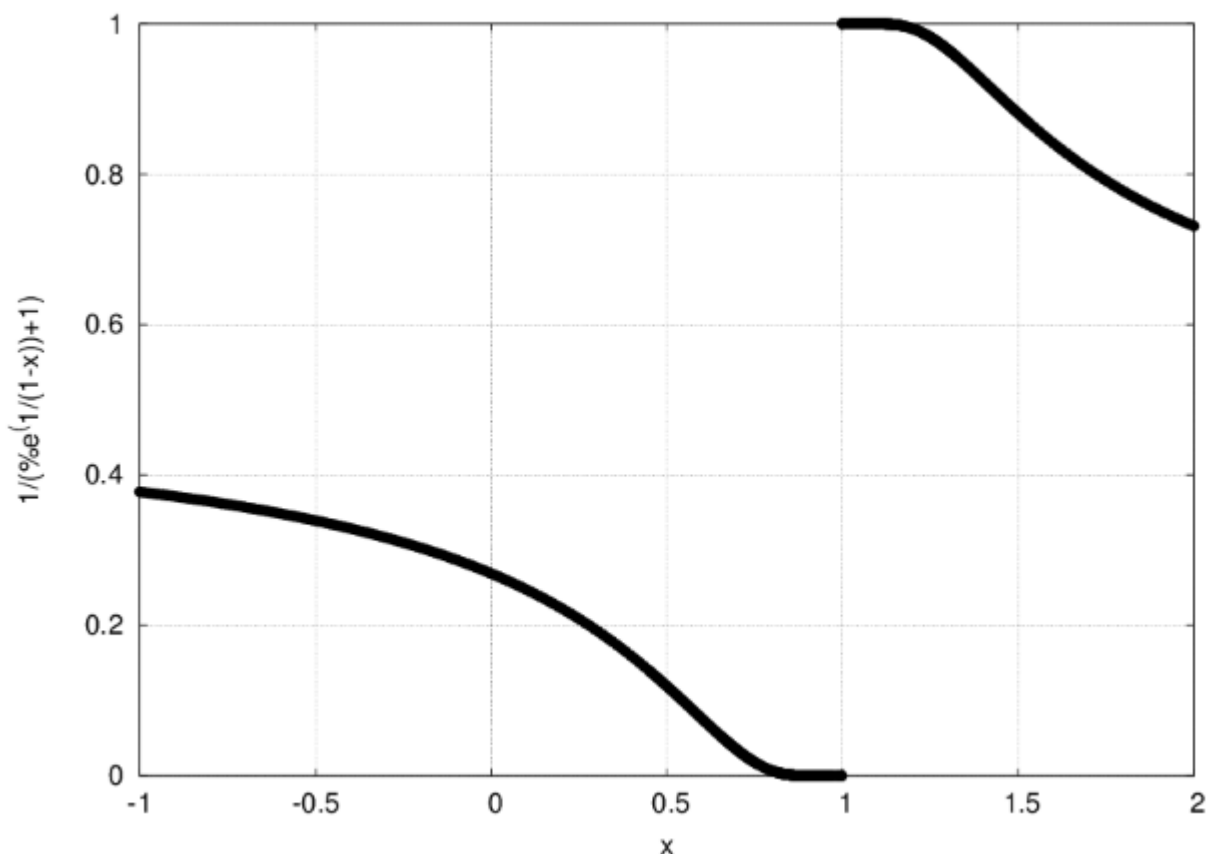


Рис. 3.2. Разрыв исследуемой функции

Так как пределы $\lim_{x \rightarrow 0-0} f(x)$ и $\lim_{x \rightarrow 0+0} f(x)$ не совпадают, но оба конечны, делаем вывод о наличии точки разрыва первого рода при $x = 1$.

Графическую иллюстрацию получаем при помощи **wxMaxima** (см. рис. 3.2).

3.3.4 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 выражения.
- *trigsim* (<выражение >), *trigexpand* (<выражение >) — упрощение или подстановка тригонометрического выражения.
- *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 , x_0 , n); — разложить функцию $f(x)$ по формуле Тейлора с центром в точке x_0 до порядка n включительно.

3.3.4.1 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(9x^2)
```

```
(%i4) d1:diff(f(x),x,1);
```

```
(%o4)          18x cos(9x^2)
```

```
(%i5) d2:diff(f(x),x,2);
```

```
(%o5)          18 cos(9x^2) - 324x^2 sin(9x^2)
```

```
(%i6) d3:diff(f(x),x,3);
```

```
(%o6)          -972x sin(9x^2) - 5832x^3 cos(9x^2)
```

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)           $x e^{xy} del(y) + y e^{xy} del(x)$ 
```

Пример:

```
(%i10) diff(exp(x*y*z));
```

(%o10)

$$x y e^{xyz} \operatorname{del}(z) + x z e^{xyz} \operatorname{del}(y) + y z e^{xyz} \operatorname{del}(x)$$

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 дифференциальное выражение:

(%i19) $\operatorname{diff}(f(x, z), x, 2) + \operatorname{diff}(f(x, z), z, 3) +$
 $\operatorname{diff}(f(x, z), x) * x^2;$ (%o19) $x^2 (z^2 + 2xz) + 2z$

Производим формальное дифференцирование, не вычисляя непосредственно результат:

(%i20) $'\operatorname{diff}(f(x, z), x, 2) + '\operatorname{diff}(f(x, z), z, 3) +$
 $'\operatorname{diff}(f(x, z), x) * x^2;$

(%o20)

$$\frac{d^3}{dz^3} (xz^2 + x^2z) + \frac{d^2}{dx^2} (xz^2 + x^2z) + x^2 \left(\frac{d}{dx} (xz^2 + x^2z) \right)$$

3.4 Экстремумы функций

3.4.1 Отыскание максимумов и минимумов

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 $f(x) \leq f(x_0)$ (см. рис. 3.3).

Определение 2. Точка x_1 называется точкой **минимума** функции $f(x)$, если в некоторой окрестности точки x_1 выполняется неравенство $f(x) \geq f(x_1)$ (см. рис. 3.3).

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.

Наличие 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)).

3.4.1.1 Теорема Ферма

Теорема Ферма. 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. $f'(x_0) = 0$.

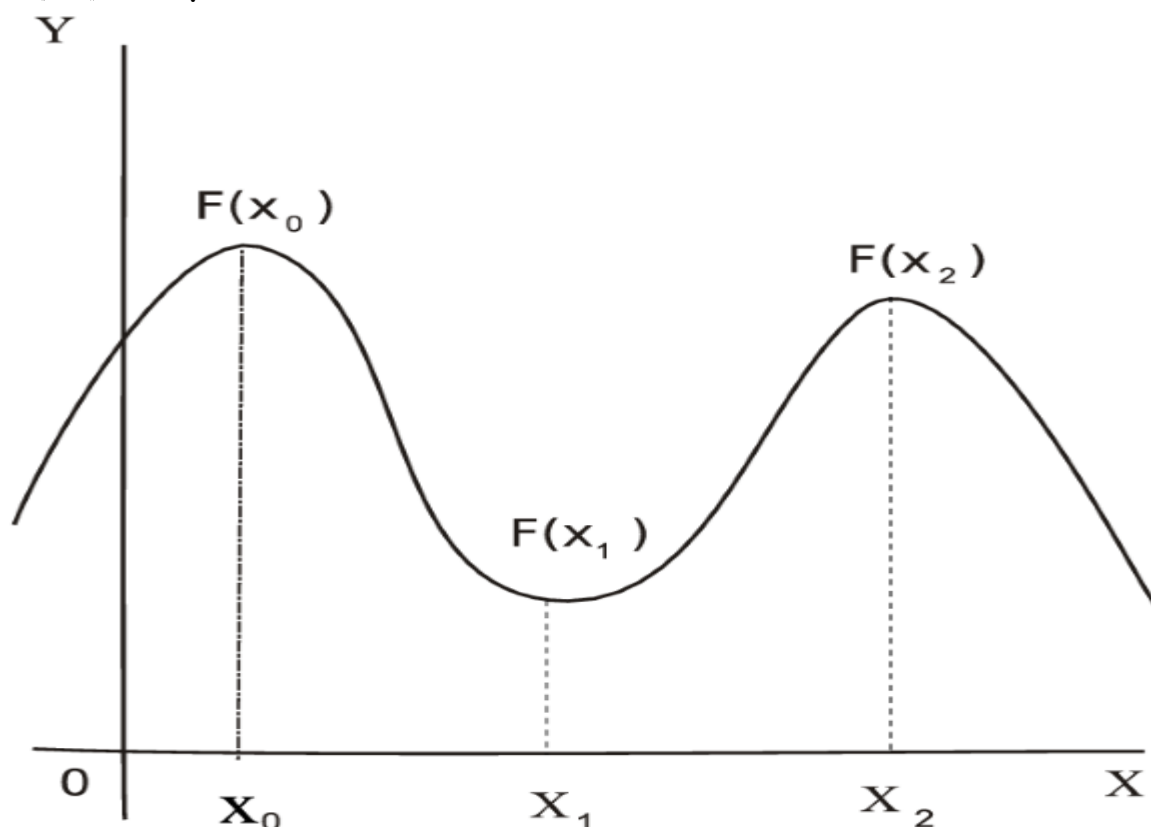


Рис. 3.3. Экстремумы функции

Пусть функция $y = f(x)$ дифференцируема на промежутке X и в точке $x_0 \in X$ принимает наименьшее значение (см. рис. 3.4).

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

$$\frac{\Delta y}{\Delta x} \geq 0 \text{ при } \Delta x > 0 \text{ (справа от } x_0\text{);}$$

$$\frac{\Delta y}{\Delta x} \leq 0 \text{ при } \Delta x < 0 \text{ (слева от } x_0\text{)}.$$

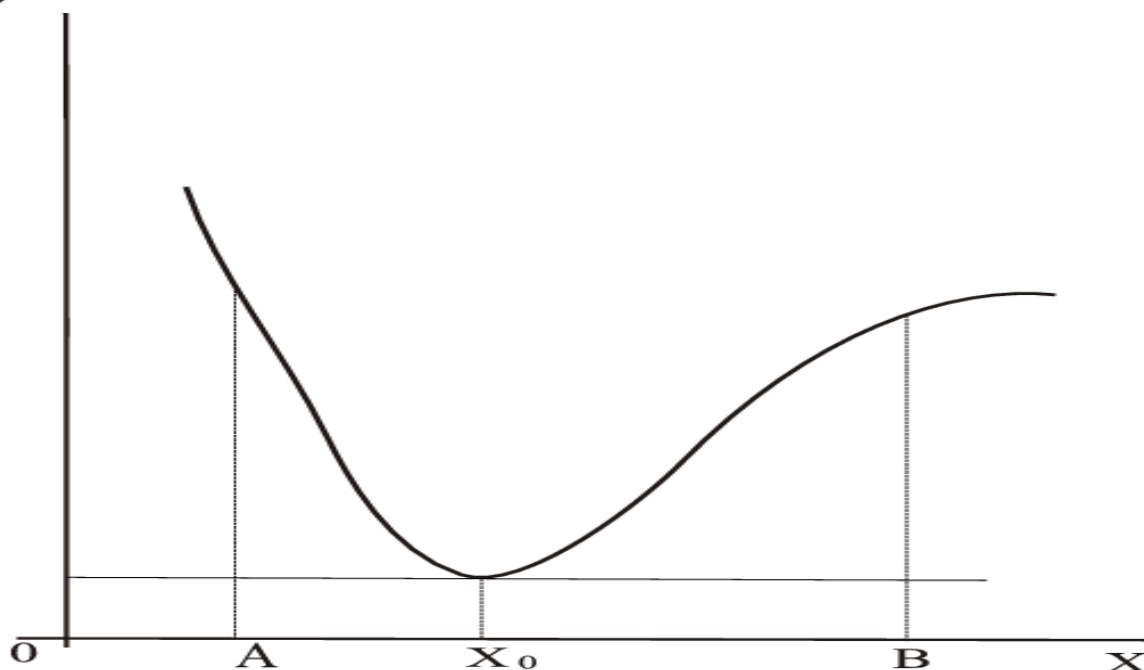


Рис. 3.4. Иллюстрация теоремы Ферма

Переходя к пределу справа и слева получим

$$\lim_{\Delta x \rightarrow 0^+} \frac{\Delta y}{\Delta x} \geq 0 \text{ и } \lim_{\Delta x \rightarrow 0^-} \frac{\Delta y}{\Delta x} \leq 0.$$

Так как функция дифференцируема на промежутке X , то пределы справа и слева равны

$$\lim_{\Delta x \rightarrow 0^+} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0^-} \frac{\Delta y}{\Delta x}.$$

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.

3.4.1.2 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

$$y' = \frac{2}{3\sqrt[3]{x}}, \quad y'(0) = \infty$$

бесконечна:

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. $y = x^2 + 1$; 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 $y(x) = x^3 - 3x^2 + 3x + 2$

Задаём исследуемую функцию

```
(%i1) f(x) := x^3 - 3*x^2 + 3*x + 2;
```

```
(%o1) f(x) := x^3 - 3x^2 + 3x + 2
```

Производную в форме функции определяем **явно**, используя функцию *define*

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

```
(%o2) df(x) := 3x^2 - 6x + 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 — $x = 1$.

3.4.1.3 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. 3.5). 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.

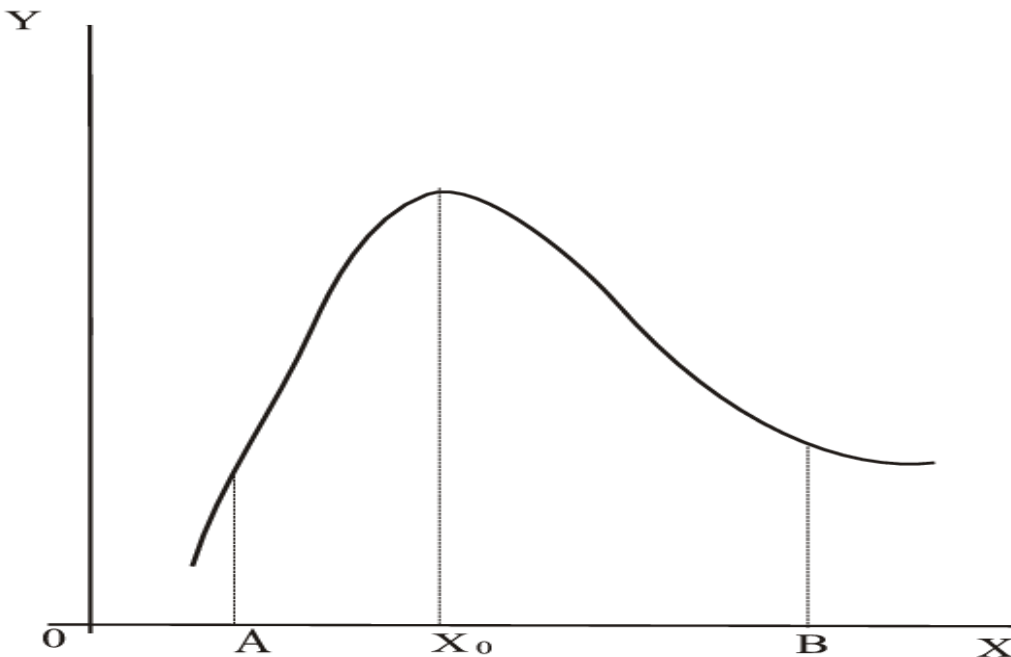


Рис. 3.5. Необходимое условие экстремума

3.4.1.4 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 — точка минимума 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

$$y(x) = x^3 - 3x^2 + 3x + 2$$

Как установлено выше, имеется одна критическая точка: $x = 1$.

Задаёмся функцией $d^2f(x)$

```
(%i4) define(d2f(x), diff(df(x), x));
```

```
(%o4)          d2f(x) := 6x - 6
```

Вычисляем значение второй производной в критической точке:

```
(%i5) map(d2f, %o3);
```

```
(%o5)          [6x - 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 $d^2f(x)$ применяется ко всем элементам списка, полученного при решении уравнения $f'(x) = 0$ (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 (см. рис. 3.6).

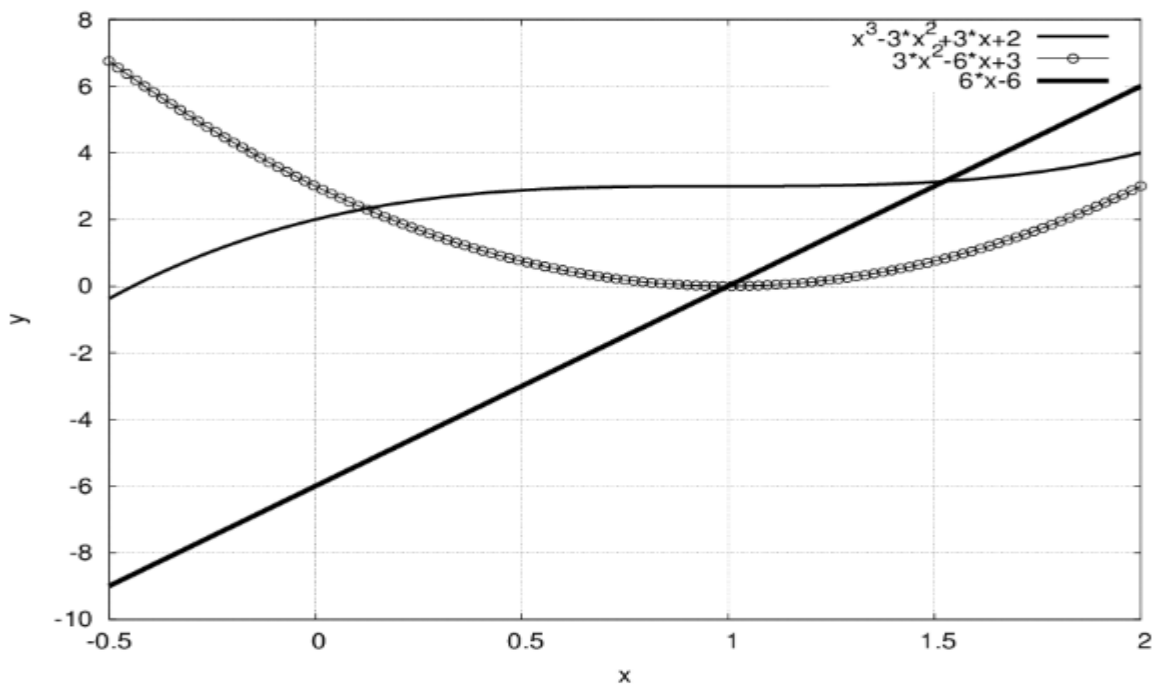


Рис. 3.6. Пример исследования функции

3.4.1.5 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 $x_2 = \frac{1}{4}$.

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.
$$y_{\min} = y\left(\frac{1}{4}\right) = -\frac{27}{256}.$$

Выполним тот же расчёт при помощи **Maxima**

```
(%i13) f(x) := x*(x-1)^3;
(%o13) f(x) := x(x-1)^3
(%i14) define(df(x), diff(f(x), x));
(%o14) df(x) := 3(x-1)^2*x + (x-1)^3
(%i15) solve(df(x)=0, x);
(%o15) [x = 1/4, x = 1]
(%i16) define(d2f(x), diff(df(x), x));
(%o16) d2f(x) := 6(x-1)x + 6(x-1)^2
(%i17) map(d2f, %o15);
(%o17) [6(x-1)x + 6(x-1)^2 = 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) 7
(%i19) df(1/3);
(%o19) 4/27
```

Производная в окрестности точки $x = 1$ не меняет знак, поэтому экстремум у исследуемой функции один — точка $x = \frac{1}{4}$. Так как $d^2f(\frac{1}{4}) > 0$, $x = \frac{1}{4}$ — точка минимума. Иллюстрация полученного результата — на рис. 3.7.

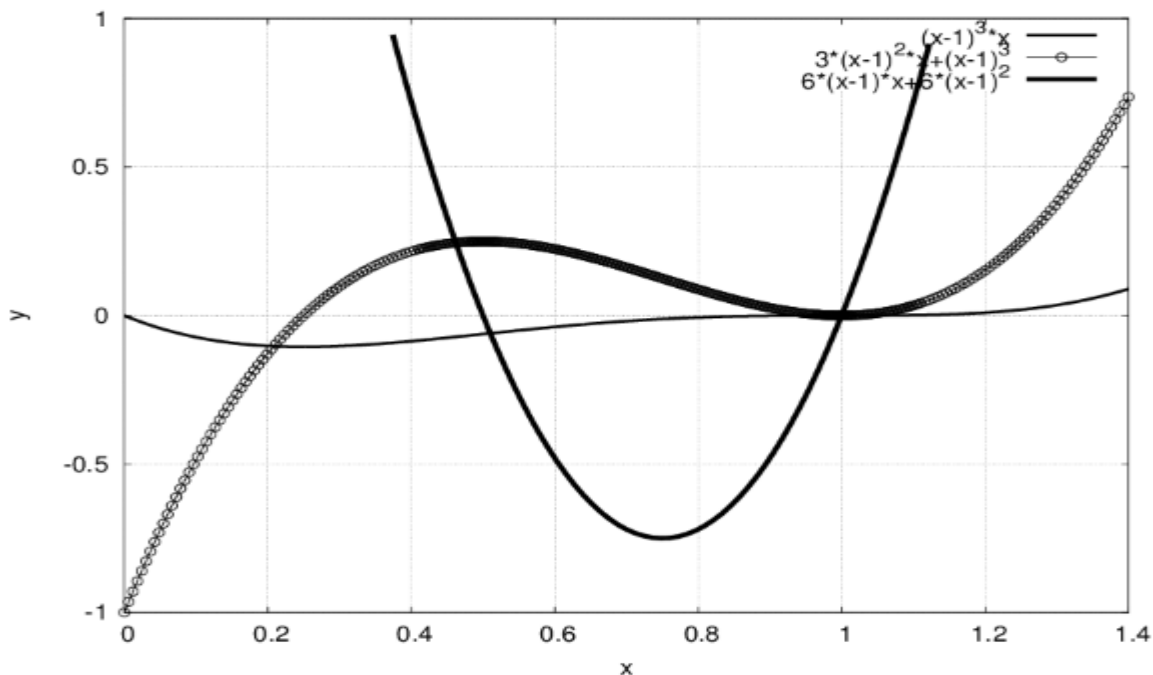


Рис. 3.7. Пример исследования функции на экстремум

3.4.1.6 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(x) := 3x^2 - 6x
(%i30)      define(df(x), diff(f(x), x));
(%o30)      df(x) := 6x - 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)      [3x^2 - 6x = -3, 3x^2 - 6x = 0, 3x^2 - 6x = 9]
```

Результат — наибольшие и наименьшие значения — находим в списке полученных значений.

3.4.2 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. 3.8а).

График функции $y = f(x)$ называется **вогнутым** в интервале (a, b) , если он расположен выше касательной, проведенной в любой точке этого интервала (см. рис. 3.8б).

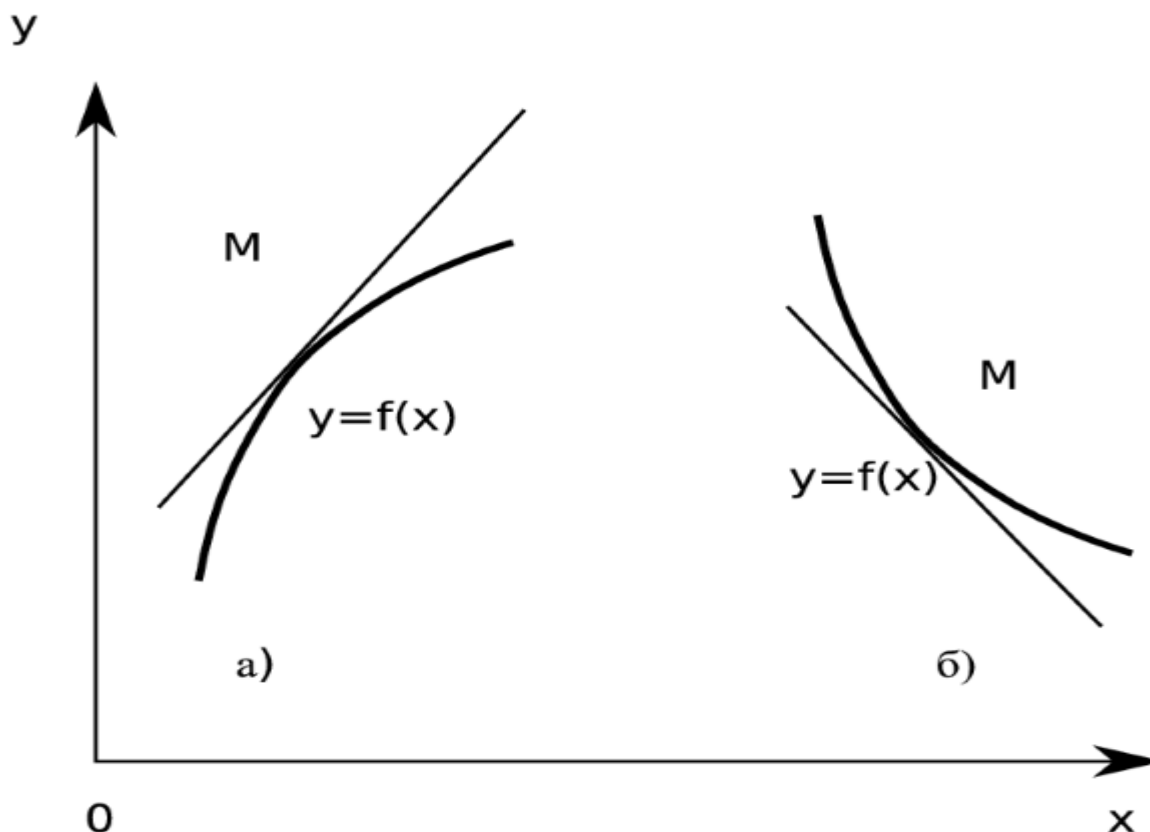


Рис. 3.8. Выпуклые и вогнутые функции.

3.4.2.1 Необходимые 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. Возьмем 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 . Возьмем 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 $x \rightarrow x_0$ 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 $(-\infty, \infty)$.

3.4.2.2 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 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 производная $f''(x_0)$ может не существовать.

Геометрическая интерпретация точек перегиба иллюстрируется рис. 3.9

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.

3.4.2.3 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.

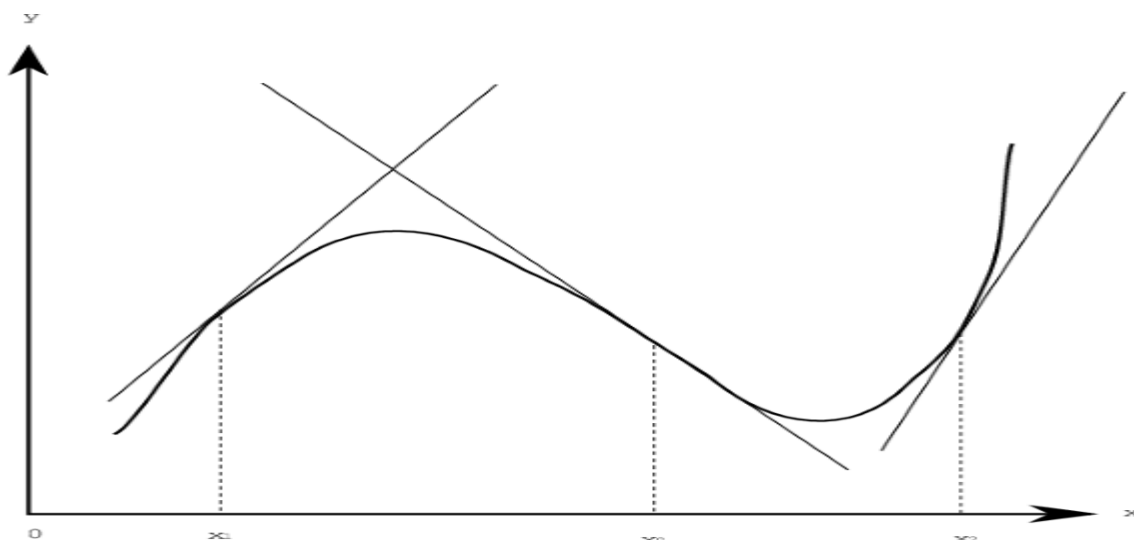


Рис. 3.9. Точки перегиба.

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.

3.4.2.4 Общая схема исследования функций и построения графика

При исследовании функции и построении ее графика рекомендуется 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 симметричен 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.

$$y(x) = f(x) = \frac{1 + x^2}{1 - x^2}$$

Пример. Исследовать функцию и построить ее график.

1. Область определения функции — $(-\infty, -1) \cup (-1, 1) \cup (1, \infty)$.

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 ∞ .

$$\lim_{x \rightarrow 1-0} y(x) = \lim_{x \rightarrow -1+0} y(x) = \infty; \quad \lim_{x \rightarrow 1+0} y(x) = \lim_{x \rightarrow -1-0} y(x) = -\infty.$$

4. Поведение функции в бесконечности.

$$\lim_{x \rightarrow \pm\infty} y(x) = -1,$$

поэтому график функции имеет горизонтальную асимптоту $y = -1$.

5. Экстремумы и интервалы монотонности. Находим первую производную

$$y'(x) = \frac{4x}{(1 - x^2)^2}.$$

$y'(x) < 0$ при $x \in (-\infty, -1) \cup (-1, 0)$, поэтому в этих интервалах функция $y(x)$ убывает.

$y'(x) > 0$ при $x \in (0, 1) \cup (1, \infty)$, 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 $x \in (-\infty, -1) \cup (1, \infty)$, 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 $y(x) = \frac{1+x^2}{1-x^2}$ изображен на рис. 3.10.

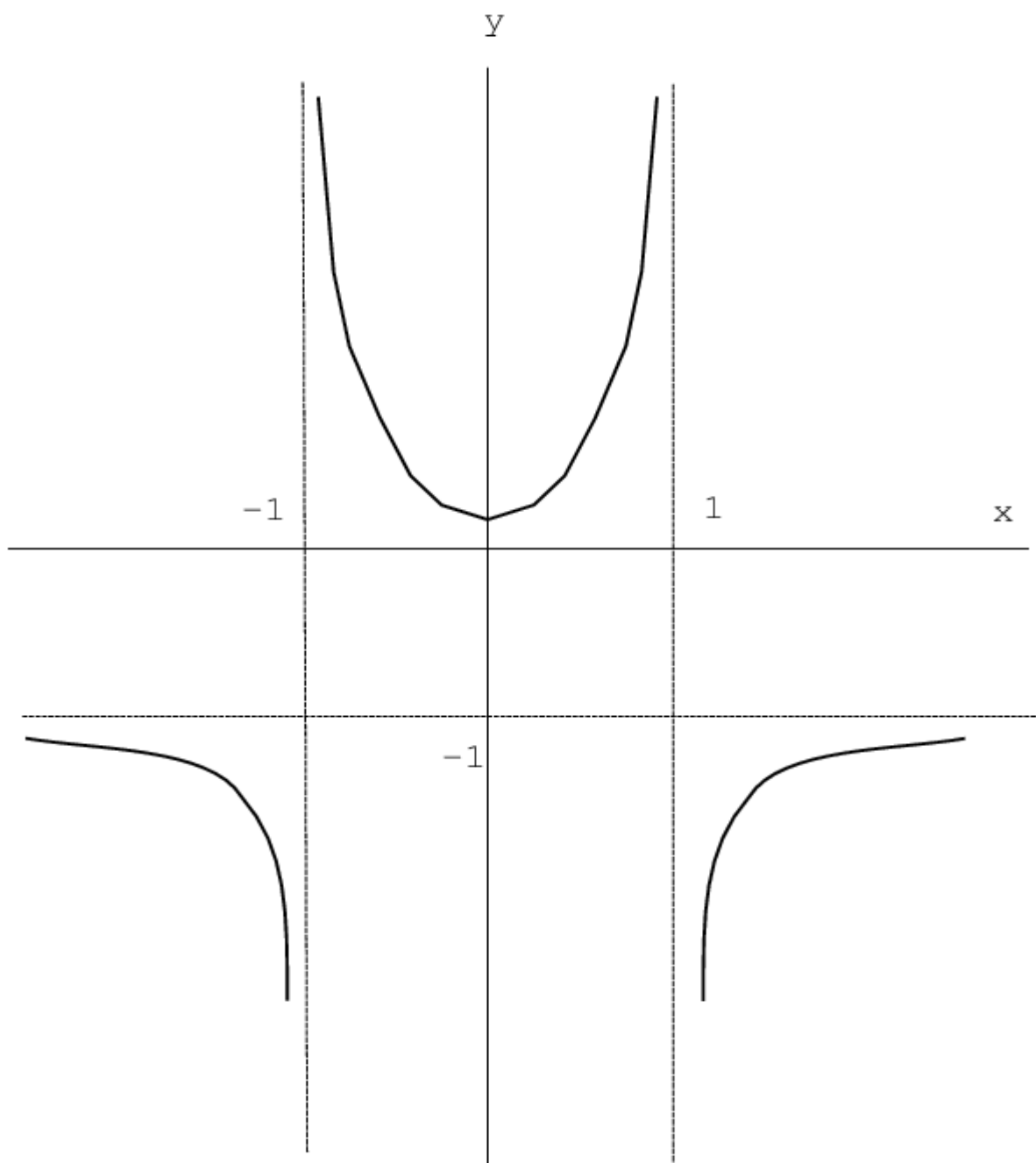


Рис. 3.10. График функции

3.4.2.5 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. 3.11а), горизонтальные (см. рис. 3.11б) и наклонные (см. рис. 3.11в).

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 $x \rightarrow x_0 - 0$ (слева) или $x \rightarrow x_0 + 0$ (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

$$\lim_{x \rightarrow \mp\infty} f(x) = b.$$

Тогда прямая $y = b$ есть горизонтальная асимптота графика функции $y = f(x)$.

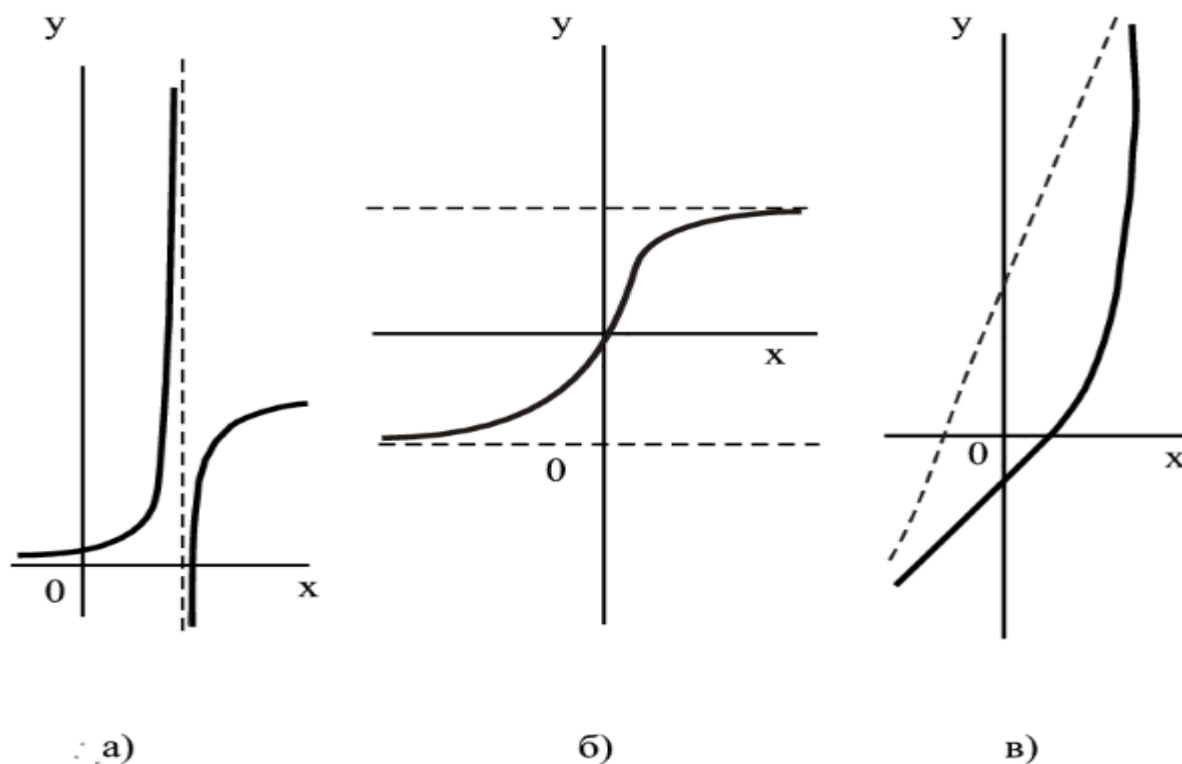


Рис. 3.11. Асимптоты

Теорема 3. Пусть функция $y = f(x)$ определена при достаточно больших x и существуют конечные пределы

$$\lim_{x \rightarrow \mp\infty} \frac{f(x)}{x} = k$$

и

$$\lim_{x \rightarrow \mp\infty} [f(x) - kx] = b.$$

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 $\lim_{x \rightarrow -d/c} f(x)$.

Перепишем дробно-рациональную функцию в виде:

$$y(x) = \frac{ax + b}{c(x + d/c)}$$

Так как $ad - bc \neq 0$ то при $x \rightarrow d/c$ числитель дробно-рациональной функции не стремится к нулю. Поэтому прямая $x = -d/c$ — асимптота графика дробно-рациональной функции.

Найдём предел $\lim_{x \rightarrow \pm\infty} f(x)$.

$$\lim_{x \rightarrow \pm\infty} \frac{ax + b}{cx + d} = \lim_{x \rightarrow \pm\infty} \frac{a + b/x}{c + d/c} = \frac{a}{c}$$

$y = a/c$ — является горизонтальной асимптотой дробно-рациональной функции.

Пример. Найти асимптоты кривой

$$y(x) = \frac{x^3}{x^2 + 1}.$$

$$\lim_{x \rightarrow \pm\infty} \frac{f(x)}{x} = \lim_{x \rightarrow \pm\infty} \frac{x^2}{x^2 + 1} = 1.$$

Поэтому $k = 1$.

Теперь ищем b .

$$b = \lim_{x \rightarrow \pm\infty} \left[\frac{x^3}{x^2 + 1} - x \right] = \lim_{x \rightarrow \pm\infty} \left(\frac{-x}{x^2 + 1} \right)$$

Функция $y(x) = \frac{x^3}{x^2 + 1}$ имеет наклонную асимптоту $y = x$.

3.4.2.6 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 $m \leq f(x) \leq M$ при $a \leq x \leq b$ (см. рис. 3.12а).
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. 3.12б).
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. 3.12в).

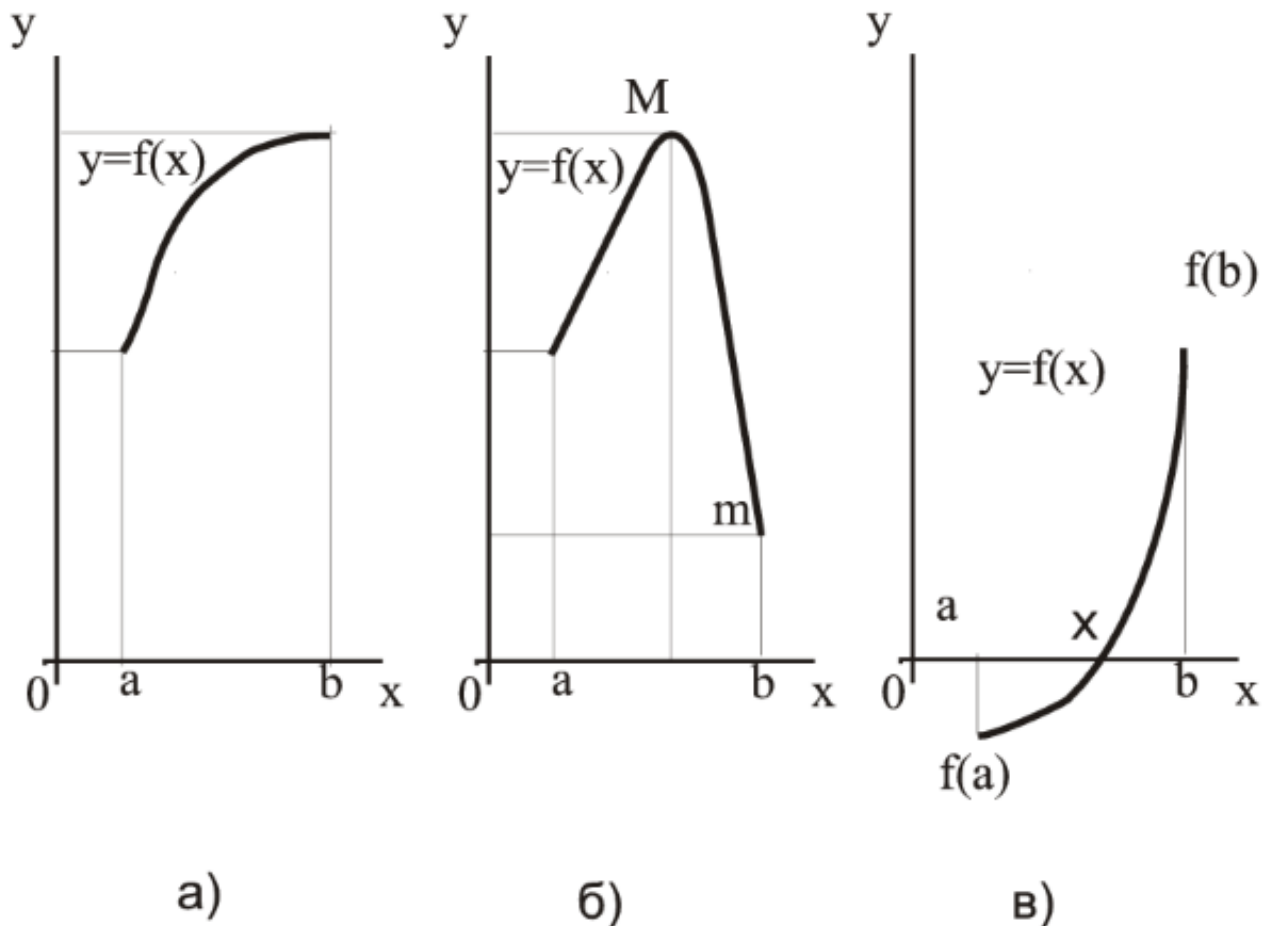


Рис. 3.12. Иллюстрации к теоремам Вейерштрасса

3.4.3 Дифференцирование функций нескольких переменных

Для определения набора частных производных функции нескольких переменным (компонентов градиента) используется функция *gradef* в формате *gradef*($f(x_1, \dots, x_n), g_1, \dots, g_m$) или *gradef*($a, x, expr$)

Выражение *gradef*($f(x_1, \dots, x_n), g_1, \dots, g_m$) определяет g_1, g_2, \dots, g_m как частные производные функции $f(x_1, x_2, \dots, x_n)$ по переменным x_1, x_2, \dots, x_n соответственно.

Зависимости между переменными можно явно указать при помощи функции *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) (d/dx y) + 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 (3 z^2 + 2 y^2 + x^2)
(%i3) express(%);
(%o3)          [d/dx (3 z^2 + 2 y^2 + x^2), d/dy (3 z^2 + 2 y^2 + x^2), d/dz (3 z^2 + 2 y^2 + x^2)]
```

```
(%i4) ev(%,diff);
```

```
(%o4) [2x, 4y, 6z]
```

Вычисление дивергенции

```
(%i5) div([x^2, 2*y^2, 3*z^2]);
```

```
(%o5) div([x^2, 2y^2, 3z^2])
```

```
(%i6) express(%)
```

```
(%o6)  $\frac{d}{dz}(3z^2) + \frac{d}{dy}(2y^2) + \frac{d}{dx}x^2$ 
```

```
(%i7) ev(%,diff);
```

```
(%o7) 6z + 4y + 2x
```

Вычисление вихря:

```
(%i8) curl([x^2, 2*y^2, 3*z^2]);
```

```
(%o8) curl([x^2, 2y^2, 3z^2])
```

```
(%i9) express(%)
```

```
(%o9)  $[\frac{d}{dy}(3z^2) - \frac{d}{dz}(2y^2), \frac{d}{dz}x^2 - \frac{d}{dx}(3z^2), \frac{d}{dx}(2y^2) - \frac{d}{dy}x^2]$ 
```

```
(%i10) ev(%,diff);
```

```
(%o10) [0, 0, 0]
```

Вычисление оператора Лапласа:

```
(%i13) laplacian(x^2+2*y^2+3*z^2);
```

```
(%o13) laplacian(3z^2 + 2y^2 + x^2)
```

```
(%i14) express(%)
```

```
(%o14)  $\frac{d^2}{dz^2}(3z^2 + 2y^2 + x^2) + \frac{d^2}{dy^2}(3z^2 + 2y^2 + x^2) + \frac{d^2}{dx^2}(3z^2 + 2y^2 + x^2)$ 
```

```
(%i15) ev(%,diff);
```

```
(%o15) 12
```

Рассмотрим пример исследования функции нескольких переменных:
исследовать на экстремум функцию

$$f(x, y) = y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12$$

Загружаем пакет **vect**

```
(%i1) load("vect")$
```

Определяем исследуемое выражение и вычисляем его градиент:

```
(%i2) f:x^3-9/2*x^2+6*x+y^2-4*y-12;
```

```
(%o2)  $y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12$ 
```

```
(%i3) grad(f);
```

$$(\%o3) \quad \text{grad} \left(y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12 \right)$$

(%i4) express(%);

$$(\%o4) \quad \left[\frac{d}{dx} \left(y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12 \right), \right.$$

$$\left. \frac{d}{dy} \left(y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12 \right), \right.$$

$$\left. \frac{d}{dz} \left(y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12 \right) \right]$$

(%i5) ev(% , diff);

$$(\%o5) \quad [3x^2 - 9x + 6, 2y - 4, 0]$$

Выделяем из полученного списка частные производные и решаем систему $f_x(x, y) = 0; f_y(x, y) = 0$

(%i6) dfdx:%o5[1];

$$(\%o6) \quad 3x^2 - 9x + 6$$

(%i7) dfdy:%o5[2];

$$(\%o7) \quad 2y - 4$$

(%i8) solve([dfdx=0, dfdy=0], [x, y]);

$$(\%o8) \quad [[x = 1, y = 2], [x = 2, y = 2]]$$

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);

$$(\%o9) \quad 6x - 9$$

(%i10) C:diff(dfdy, y);

$$(\%o10) \quad 2$$

(%i11) B:diff(dfdx, y);

$$(\%o11) \quad 0$$

(%i12) A*C-B^2;

$$(\%o12) \quad 2(6x - 9)$$

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. 3.13).

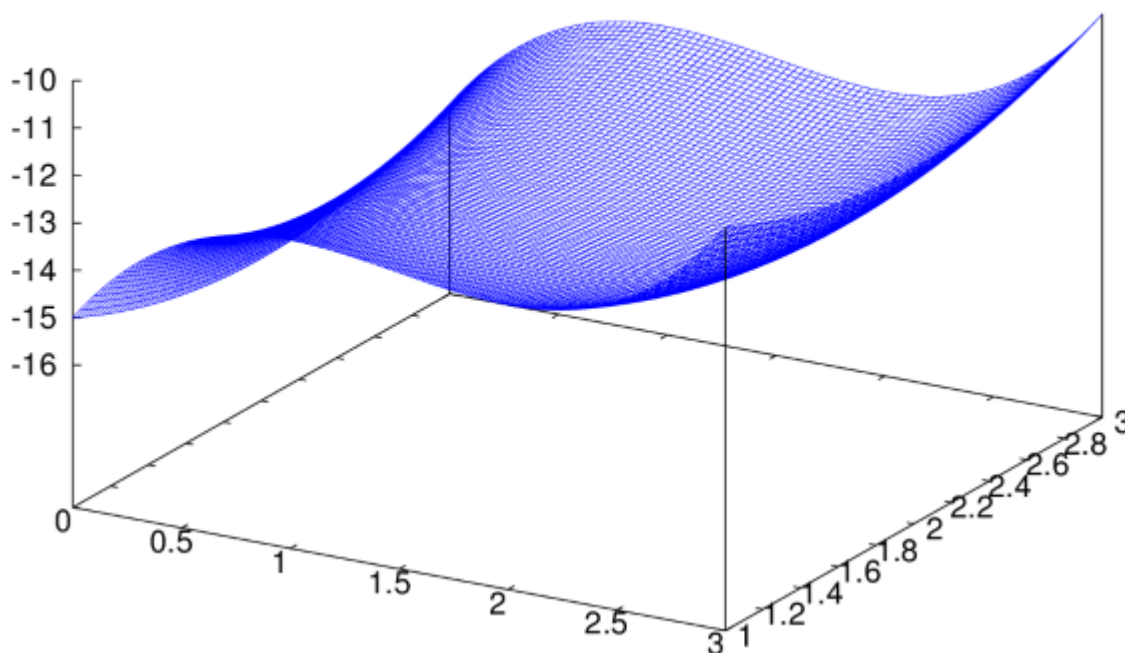


Рис. 3.13. Поиск экстремума функции нескольких переменных

3.5 Аналитическое и численное интегрирование

3.5.1 Основные команды

Неопределённый интеграл $\int f(x)dx$ вычисляется с помощью команды *integrate*(*f*, *x*), где *f* — подинтегральная функция, *x* — переменная интегрирования.

Для вычисления определённого интеграла $\int_a^b f(x)dx$ в команде *integrate* добавляются пределы интегрирования, например,

$$\int_0^\pi (1 + \cos(x))^2 dx = \frac{3}{2}\pi$$

Несобственные интегралы с бесконечными пределами интегрирования вычисляются, если в параметрах команды *integrate* указывать, например, *x*, 0, *inf*.

Численное интегрирование выполняется функцией *romberg* или при помощи функций пакета *quadpack*.

3.5.2 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

$$\int_0^{+\infty} e^{-ax} dx = \lim_{x \rightarrow \infty} -\frac{e^{-ax} - 1}{a}.$$

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 (2a+2)/5 an integer? no;
Is 2a-3 positive, negative, or zero? neg;
(%o2)  $\beta\left(a+1, \frac{3}{2}-a\right)$ 
(%i3) properties(a);
(%o3) [database info, a > 1]
```

Вернёмся 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);
(%o1) [a > 0] (%o2)  $\frac{1}{a}$ 
```

Отменить принятые ограничения на значения параметров можно, используя функцию `forget`.

Пример:

```
(%i1) assume(n+1>0); integrate((a+b)*x^(n+1), x);
(%o1) [n > -1] (%o2)  $\frac{(b+a)x^{n+2}}{n+2}$ 
```

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]
```

```
Is n + 2 zero or nonzero? zero;
```

```
(%o4) (b + a) log(x)
```

Результат, который получен, совершенно другой!

3.5.3 Основные приёмы интегрирования

В **Maxima** имеется функция, предназначенная для выполнения расчётов 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 её 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) ez(x) sin(x)  $\left(\frac{d}{dx} z(x)\right)$ 
```

```
(%i3) a1: antid(expr, x, z(x));
```

```
(%o3) [ez(x) sin(x), -ez(x) cos(x)]
```

При помощи пакета **antid** можно выполнить формальное интегрирование по частям,:

```
(%i1) expr:u(x)*diff(v(x), x);
```

```
(%o1) u(x)  $\left(\frac{d}{dx} v(x)\right)$ 
```

```
(%i2) a:antid(expr, x, v(x));
```

```
(%o2) antid  $\left(u(x) \left(\frac{d}{dx} v(x)\right), x, v(x)\right)$ 
```

```
(%i3) b:antidiff(expr, x, v(x));
```

```
(%o3) antidiff  $\left(u(x) \left(\frac{d}{dx} v(x)\right), x, v(x)\right)$ 
```

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 вычисление 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, поэтому 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)
$$x^4 - 4x^3 + 2x^2 - 7x - 4$$

```
(%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)
$$\frac{\log(x-4)}{73} - \frac{\int \frac{x^2+4x+18}{x^3+2x+1} dx}{73}$$

Возможным решением является упрощение интеграла, сопровождающееся понижением степени рационального выражения в знаменателе. При этом

необходимо установить в *true* значение переменной *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 - 4x^3 + 2x^2 - 7x - 4
(%i2) polyfactor:true$ ffact:allroots(f);
(%o3) 1.0(x-3.9999999999999997)(x+0.4533976515164)
(x^2 - 0.45339765151641x + 2.205569430400593)
(%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 рациональными 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)      cos(6x)  cos(4x)  cos(2x)
           24      16      8
(%i2) integrate(1/cos(x)^3,x);
(%o2)      log(sin(x)+1)  log(sin(x)-1)  sin(x)
           4              4              2 sin(x)^2 - 2
(%i3) integrate(x^3*log(x),x);
(%o3)      x^4 log(x)  x^4
           4          16
```

3.5.4 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 удалиться.

Кроме того, вычисление преобразования Лапласа возможно и для дифференциальных уравнений и интегралов типа свёртки.

(%i1) laplace(c,t,s);

(%o1)
$$\frac{c}{s}$$

(%i2) laplace(erf(t),t,s);

(%o2)
$$\frac{e^{\frac{s^2}{4}} \left(1 - \operatorname{erf}\left(\frac{s}{2}\right)\right)}{s}$$

(%i3) laplace(sin(t)*exp(-a*t),t,s);

(%o3)
$$\frac{1}{s^2 + 2as + a^2 + 1}$$

Функция $\operatorname{ilt}(expr,t,s)$ 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)
$$\frac{2}{s^2 + 8s + 20}$$

(%i4) ilt(%,s,t);

(%o4)
$$e^{-4t} \sin(2t)$$

3.6 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 результата. 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 объясняется 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, приближённому вычислению интегралов и решению дифференциальных уравнений. Степенные ряды и разложения по многочленам Чебышёва широко используются при вычислении значений функции с заданной степенью точности. Они являются эффективным вычислительным средством при решении широкого круга научно-технических задач.

3.6.1 Приближённое 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

$$f(x) = \sum_{i=0}^{\infty} u_i(x) = \sum_{i=0}^{\infty} a_i(x-x_0)^i = a_0 + a_1(x-x_0) + a_2(x-x_0)^2 + \dots + a_n(x-x_0)^n + \dots,$$

мы получим, что точное значение $f(x_1)$ равно сумме этого ряда при $x = x_1$

$$f(x_1) = \sum_{i=0}^{\infty} a_i(x_1-x_0)^i = a_0 + a_1(x_1-x_0) + a_2(x_1-x_0)^2 + \dots + a_n(x_1-x_0)^n + \dots,$$

а приближённое — частичной сумме $S_n(x_1)$

$$f(x_1) \approx S_n(x_1) = \sum_{i=0}^n a_i(x_1-x_0)^i = a_0 + a_1(x_1-x_0) + a_2(x_1-x_0)^2 + \dots + a_n(x_1-x_0)^n.$$

Для погрешности приближения мы имеем выражение в виде остатка ряда

$$f(x_1) - S_n(x_1) = r_n(x_1),$$

где

$$r_n(x_1) = \sum_{i=1}^{\infty} x_1^{n+i} = a_{n+1}x_1^{n+1} + a_{n+2}x_1^{n+2} + \dots$$

Для знакопеременных рядов с последовательно убывающими членами

$$|r_n(x)| = \left| \sum_{i=1}^{\infty} u_{n+i}(x) \right| < |u_{n+1}(x)|.$$

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 Чебышёва.

Для приближённого нахождения значений функции посредством степенных рядов, как правило, используются её 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 определённой 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

$$f(x_0) + \frac{f'(x_0)}{1!} (x - x_0) + \frac{f''(x_0)}{2!} (x - x_0)^2 + \dots + \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n.$$

Формула Тейлора для функции $f(x)$ — это представление функции в виде суммы её многочлена Тейлора степени n ($n = 0, 1, 2, \dots$) 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 её 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

$$r_n(x) = o((x - x_0)^n), \quad x \rightarrow x_0.$$

Получаем, что

$$P_n(x) = f(x_0) + \frac{f'(x_0)}{1!} (x - x_0) + \frac{f''(x_0)}{2!} (x - x_0)^2 + \dots + \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n.$$

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

$$r_n(x) = \frac{f^{(n+1)}(x_0 + \theta(x - x_0))}{(n + 1)!} (x - x_0)^{(n+1)},$$

$$0 < \theta < 1, \quad x \in (x_0 - \delta, x_0 + \delta).$$

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 $f(x) = a_0x^n + a_1x^{n-1} + \dots + a_n$. Тогда при любых x и h имеет место следующая формула:

$$\begin{aligned} f(x + h) &= a_0(x + h)^n + a_1(x + h)^{n-1} + \dots + a_n = \\ &= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \dots + \frac{f^{(k)}(x)}{k!}h^k + \dots + \frac{f^{(n)}(x)}{n!}h^n. \end{aligned}$$

Рядом Маклорена для функции $f(x)$ называется её 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.

Приведём разложения по формуле Маклорена для основных элементарных математических функций:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + o(x^n),$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} + o(x^{2n}),$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} + o(x^{2n+1}),$$

$$(1+x)^\alpha = 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!} x^2 + \dots + \frac{\alpha(\alpha-1)\dots(\alpha-n+1)}{n!} x^n + o(x^n),$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + \dots + (-1)^{n-1} \frac{x^n}{n} + o(x^n).$$

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: Найти многочлен Тейлора 9-ой степени экспоненциальной функции e^x в начале координат.

```
(%i29) taylor(exp(x), x, 0, 9);
```

```
(%o29)
```

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + \frac{x^8}{40320} + \frac{x^9}{362880} + \dots$$

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: 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);
```

$$(\%o1) 1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+\frac{x^6}{720}+\frac{x^7}{5040}+\frac{x^8}{40320}+\dots$$

Вычисляем частичную сумму ряда при $x = 1$:

```
(%i2) ev(t, x=1);
```

$$(\%o2) \frac{109601}{40320}$$

Значение e в форме с плавающей точкой находим, используя функцию *float*:

```
(%i3) float(%);
```

$$(\%o3) 2.71827876984127$$

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 произведём по формуле остаточного члена ряда Маклорена

$$|f(x_1) - S_n(x_1)| = |r_n(x_1)| = \left| \frac{f^{n+1}(c)}{(n+1)!} \right|,$$

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 окрестности a). Результатом выполнения команды *powerseries* может быть построение её ряда Тейлора в общей форме, например:

(%i1) `powerseries(sin(x), x, 0);`

(%o1)
$$\sum_{i2=0}^{\infty} \frac{(-1)^{i2} x^{2i2+1}}{(2i2+1)!}$$

(%i2) `powerseries(sin(x^2), x, 0);`

(%o2)
$$\sum_{i3=0}^{\infty} \frac{(-1)^{i3} x^{2(2i3+1)}}{(2i3+1)!}$$

Для разложения в ряд Тейлора функции нескольких переменных используется функция *taylor* с указанием списка переменных в форме: *taylor(expr, [x₁, x₂, ...], [a₁, a₂, ...], [n₁, n₂, ...])*

Пример: 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)
$$x - x^2 + x^3 - x^4 + x^5 - x^6 + \dots$$

Пример: Найти разложение функции $\arccos(x)$ в ряд Маклорена.

(%i6) `taylor(acos(x), x, 0, 12);`

(%o6)
$$\frac{\pi}{2} - x - \frac{x^3}{6} - \frac{3x^5}{40} - \frac{5x^7}{112} - \frac{35x^9}{1152} - \frac{63x^{11}}{2816} + \dots$$

Пример: 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)

$$1 + e^2 + e^2(x-2) + \frac{e^2(x-2)^2}{2} + \frac{e^2(x-2)^3}{6} + \frac{e^2(x-2)^4}{24} + \frac{e^2(x-2)^5}{120} + \dots$$

Пример: 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. Приведём пример функции, не имеющей разложения в ряд Тейлора и для которой команда *taylor* не даёт результата: $f(x) = 1/x^2 + x$

```
(%i8) taylor(1/x^{2}+x, x, 0, 7);
```

```
(%o8)          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)
  2 2^2 + 1      (2 - 2 2^2) (x - 2)      (2^2 + 2) (x - 2)^2
  ----- - ----- + ----- + ...
      2^2          2 2^2                    8 2^2
```

```
(%i14) ratsimp(%);
```

```
(%o14)
2^{-2-3} ((2^2 + 2) x^2 + (2^{2+3} - 4 2^2 - 8 2) x + 4 2^2 + 12 2 + 8)
```

Пакет **Maxima** даёт 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 её разложений Тейлора различных степеней.

Пример: Сравним функцию $\cos(x)$ с её разложением Маклорена 4-ой степени на интервале $[-5, 5]$.

Построим разложение

```
(%i15) appr:taylor(cos(x), x, 0, 5);
```

```
(%o15)          2      4
              x  - x
              2   24
              + ...
```

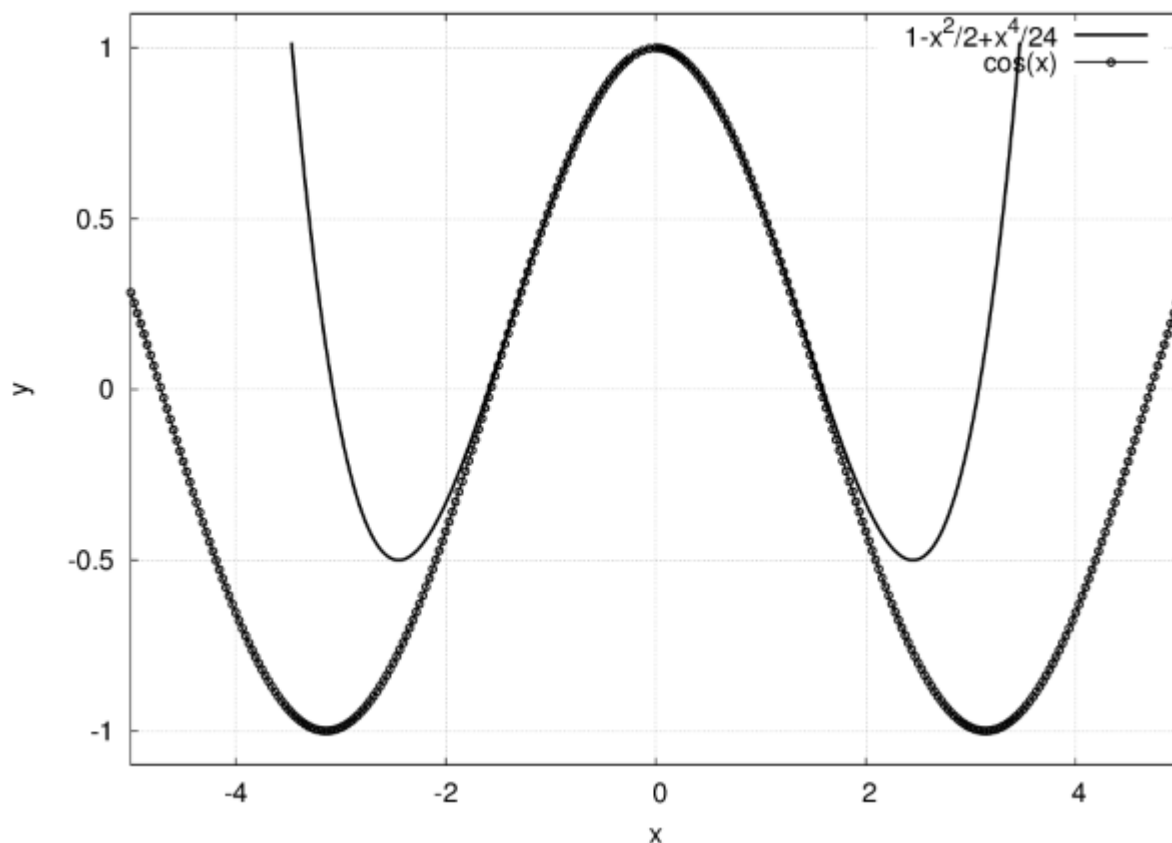



Рис. 3.14. Сопоставление разложения в ряд Маклорена и функции $y=\cos(x)$

Построим график (экранный формат, 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 её разложения практически совпадают, однако при возрастании x начинают отличаться.

Пример: Сравним функцию $\cos(x)$ с её 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);
```

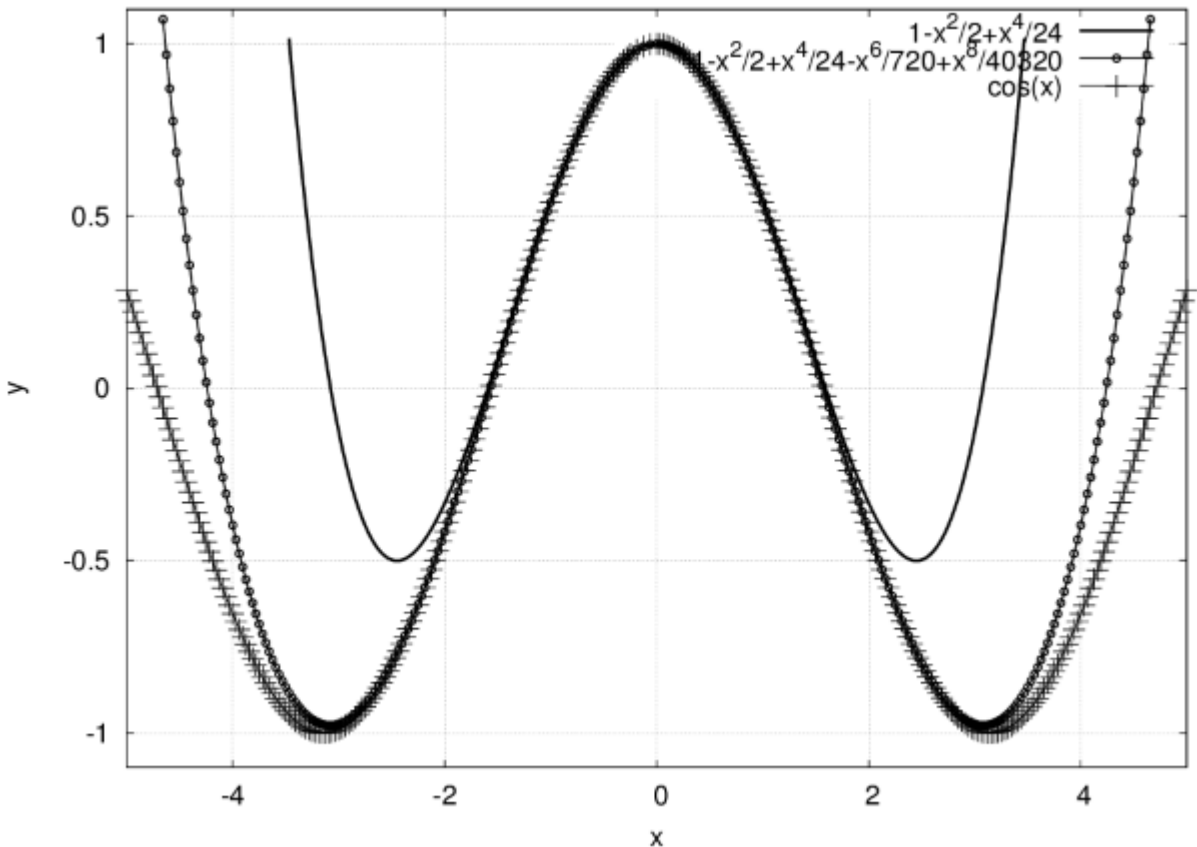


Рис. 3.15. Сопоставление двух разложений в ряд Маклорена и функции $y=\cos(x)$

$$(\%018) \quad 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \frac{x^8}{40320} + \dots$$

Пример показывает, что при использовании разложения Тейлора более высокой степени точность приближения возрастает и удаётся 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*).

3.6.2 Приближённое вычисление определённых интегралов

Степенные ряды эффективны и удобны при приближённом вычислении определённых интегралов, не выражающихся в конечном виде через элементарные функции. Для вычисления $\int_0^x f(t)dt$ подынтегральная функция $f(t)$ раскладывается в степенной ряд. Если

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n + \dots, \quad |x| < R,$$

то при $|x| < R$ степенной ряд можно интегрировать почленно. Получаем метод вычисления интеграла $\int_0^x f(t)dt$ с любой наперёд заданной точностью

$$\int_0^x f(t)dt = a_0x + a_1\frac{x^2}{2} + a_2\frac{x^3}{3} + \dots + a_n\frac{x^{n+1}}{n+1} + \dots$$

Пример: Приближённое 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

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, \quad |x| < \infty,$$

то

$$e^{-x^2/2} = 1 - \frac{x^2}{2} + \frac{x^4}{2^2 2!} - \frac{x^6}{2^3 3!} + \dots$$

Подставив этот ряд под знак интеграла и произведя почленное интегрирование получаем

$$\phi(x) = \frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt = \frac{2}{\sqrt{2\pi}} \left[x - \frac{x^3}{3 \cdot 2} + \frac{x^5}{5 \cdot 2^2 \cdot 2!} - \frac{x^7}{7 \cdot 2^3 \cdot 3!} + \dots \right]$$

Так как это знакопеременный ряд с последовательно убывающими слагаемыми, то погрешность вычисления интеграла последовательно убывает и не превышает последнего слагаемого.

Рассмотрим пример приближённого представления интеграла в виде полинома некоторой степени в том случае, когда он не вычисляется в замкнутой аналитической форме.

$$\int_0^1 e^{-x^2/2} dx$$

Пример: Вычислить $\int_0^1 e^{-x^2/2} dx$, оценить достигнутую точность

Используем разложение подинтегральной 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);

$$(%o1) \quad f(x) := \exp\left(\frac{-x^2}{2}\right)$$

(%i2) taylor(f(x), x, 0, 8);

$$(%o2) \quad 1 - \frac{x^2}{2} + \frac{x^4}{8} - \frac{x^6}{48} + \frac{x^8}{384} + \dots$$

Интегрируя в пределах от 0 до 1, получаем числовой результат:

(%i3) integrate(%, x, 0, 1);

$$(%o3) \quad \frac{103499}{120960}$$

(%i4) float(%);

(%o4) 0.85564649470899

Точность расчёта оцениваем, интегрируя в пределах от 0 до a :

(%i5) integrate(%o2, x, 0, a);

(%o5)
$$\frac{35a^9 - 360a^7 + 3024a^5 - 20160a^3 + 120960a}{120960}$$

При $a = 1$ находим:

(%i6) expand(%);

(%o6)
$$\frac{a^9}{3456} - \frac{a^7}{336} + \frac{a^5}{40} - \frac{a^3}{6} + a$$

(%i7) float(1/3456);

(%o7) 2.8935185185185184 10⁻⁴

Таким образом, точность расчёта значения интеграла $\int_0^1 e^{-x^2/2} dx$, не хуже 0,0003. Окончательно

$$\int_0^1 e^{-x^2/2} dx = 2.8935 \pm 0.0003$$

3.7 Преобразование степенных рядов

Пакет **Maxima** позволяет не только строить разложение различных функций в степенные ряды, но и представления их в виде дробнорациональной 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

$$R(x) = \frac{\sum_{k=0}^L p_k x^k}{1 + \sum_{k=1}^M q_k x^k}$$
, чьё разложение в степенной ряд совпадает со степенным рядом $f(x)$ с точностью до коэффициента при x^{L+M} .

Паде-аппроксимант задаётся 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 чём 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 задаём 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 точно 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 значения функции.

Паде-аппроксимация наиболее эффективна для функций, имеющих 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. Её 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.

Пример:

```
(%i1) t:taylor(exp(x), x, 0, 3);
```

```
(%o1) 1 + x +  $\frac{x^2}{2}$  +  $\frac{x^3}{6}$  + ...
```

```
(%i2) pade(t, 1, 2);
```

```
(%o2) [ $\frac{2x + 6}{x^2 - 4x + 6}$ ]
```

```
(%i3) taylor(sin(x)/x, x, 0, 7);
```

```
(%o3) 1 -  $\frac{x^2}{6}$  +  $\frac{x^4}{120}$  -  $\frac{x^6}{5040}$  + ...
```

```
(%i4) pade(%, 2, 4);
```

```
(%o4) [ $-\frac{620x^2 - 5880}{11x^4 + 360x^2 + 5880}$ ]
```

```
(%i5) taylor(1/(cos(x) - sec(x))^3, x, 0, 5);
```

$$\frac{1}{x^6} + \frac{1}{2x^4} + \frac{11}{120x^2} - \frac{347}{15120} - \frac{6767x^2}{604800} - \frac{15377x^4}{7983360} + \dots$$

(%i6) pade(%, 3, inf);

$$\left[-\frac{120}{41x^{10} + 60x^8 + 120x^6}, \frac{8806092x^2 - 16847160}{1353067x^{10} - 382512x^8 + 16847160x^6} \right]$$

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 $a + 1/(b + 1/(c + \dots))$) представляется списком $[a, b, c, \dots]$. 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 $a + 1/(b + 1/(c + \dots))$.

Примеры использования функций *cf* и *cfdisrep*:

(%i1) cf ([1, 2, -3] + [1, -2, 1]);

(%o1) [1, 1, 1, 2]

(%i2) cfdisrep (%);

$$1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}$$

(%i3) cflength: 3;

(%o3) 3

(%i4) cf (sqrt (3));

(%o4) [1, 1, 2, 1, 2, 1, 2]

(%i5) cfdisrep (%);

$$1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{1 + \frac{1}{2 + \frac{1}{1 + \frac{1}{2}}}}}}$$

(%i6) ev (%, numer);

(%o6) 1.731707317073171

3.8 Решение дифференциальных уравнений в Maxima

3.8.1 Основные определения

Дифференциальным уравнением называется уравнение вида $F(x, y, y', \dots, y^{(n)}) = 0$, где $F(t_0, t_1, \dots, t_{n+1})$ — функция, определенная в некоторой области D пространства R^{n+2} , x — независимая переменная, y — функция от $x, y', \dots, y^{(n)}$ — 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: $F(x, f(x), f'(x), \dots, f^{(n)}(x)) = 0$.

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: 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: 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: 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: 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 дифференциального 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 $D = (x, y; a < x < b; c < y < d)$, 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.

3.8.2 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 $\frac{dx}{dt} = F(t, x)$, так и в неявной $M \frac{dy}{dt} = F(t, x)$, где M — матрица, — т.н. решатель ОДУ (*solverODE*), 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" производной (т.е. с производной, вычисление которой запрещено с помощью одиночной 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, допускающие понижение порядка.

Тип используемого метода сохраняется в переменной *method*. При использовании интегрирующего множителя он сохраняется в переменной *int factor*. 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)      y = (%c - e^-x) e^2x
```

Вариант с явным указанием зависимости $y = y(x)$:

```
(%i1) depends(y,x);
(%o1)      [y(x)]
(%i2) ode2(diff(y,x)=2*y+exp(x),y,x);
(%o2)      y = (%c - e^-x) e^2x
```

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)      y = %k1 e^2x + %k2 e^x
```

Параметры *%k1* и *%k2* — постоянные интегрирования для уравнений второго порядка.

Рассмотрим варианты вычисления частных решений: для уравнения первого порядка

```
(%i5) ic1(%o1,x=1,y=1);
(%o5)      y = e^-2 ((e+1) e^2x - e^(x+2))
```

для уравнения второго порядка

```
(%i6) ic2(%o4,x=0,y=1,diff(y,x)=1);
```

$$(\%06) \quad y = e^x$$

3.8.3 Решение основных типов дифференциальных уравнений

3.8.3.1 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);
```

$$(\%01) \quad \frac{d}{dx} y = \frac{\sqrt{1-y^2}}{\sqrt{1-x^2}}$$

```
(%i2) rez:ode2(difur1,y,x);
```

$$(\%02) \quad \operatorname{asin}(y) = \operatorname{asin}(x) + \%c$$

Отыскиваем различные варианты частных решений:

```
(%i3) ic1(rez,x=0,y=0);
```

$$(\%03) \quad \operatorname{asin}(y) = \operatorname{asin}(x)$$

```
(%i4) ic1(rez,x=0,y=1);
```

$$(\%04) \quad \operatorname{asin}(y) = \frac{2 \operatorname{asin}(x) + \pi}{2}$$

3.8.3.2 Однородные уравнения

Под однородными уравнениями понимаются уравнения вида $y' = f\left(\frac{y}{x}\right)$. 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:

```
(%i1) homode:'diff(y,x) = (y/x)^2 + 2*(y/x);
```

$$(\%o1) \quad \frac{d}{dx} y = \frac{y^2}{x^2} + \frac{2y}{x}$$

(%i2) ode2(homode, y, x);

$$(\%o2) \quad -\frac{xy + x^2}{y} = \%c$$

Находим частное решение:

(%i3) ic1(%, x=2, y=1);

$$(\%o3) \quad -\frac{xy + x^2}{y} = -6$$

Более общий вариант дифференциальных уравнений, уравнения вида: $y' = \frac{a_1x + b_1y + c_1}{a_2x + b_2y + c_2}$ — сводим их к однородным. **Maxima** не способна решать такие уравнения при помощи *ode2* непосредственно, а лишь после необходимого преобразования.

3.8.3.3 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;

$$(\%o1) \quad \frac{d}{dx} y - \frac{y}{x} = x$$

(%i2) ode2(lineq1, y, x);

$$(\%o2) \quad y = x(x + \%c)$$

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;

$$(\%o3) \quad y^2 - (2xy + 3) \left(\frac{d}{dx} y \right) = 0$$

(%i4) ode2(lineq2, y, x);

$$(\%o4) \quad \frac{xy + 1}{y^3} = \%c$$

3.8.3.4 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 $\mu = \mu(x, y)$ (интегрирующий 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 **Mathima**:

Для решения в **Mathima** дифференциальное уравнение 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) (y^2 + x^2) \left( \frac{d}{dx} y \right) + \frac{y^3}{3} + x^2 y + 2 x y = 0
```

```
(%i2) ode2(deq, y, x);
```

```
(%o2) \frac{e^x y^3 + 3 x^2 e^x y}{3} = %c
```

Указание на интегрирующий множитель

```
(%i3) intfactor;
```

```
(%o3) e^x
```

Указание на использованный метод

```
(%i4) method;
```

```
(%o4) 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) (4 y^3 + 6 x^2 y) \left( \frac{d}{dx} y \right) + 6 x y^2 + 3 x^2 = 0
```

```
(%i6) ode2(deq1, y, x);
(%o6)       $y^4 + 3x^2y^2 + x^3 = \%c$ 
```

Указание на использованный метод

```
(%i7)      method;
(%o7)      exact
```

3.8.3.5 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 0 и 1³.

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)       $\frac{d}{dx}y = \frac{4y}{x} + x\sqrt{y}$ 
```

```
(%i2) ode2(deq, y, x);
```

```
(%o2)       $y = x^4 \left( \frac{\log(x)}{2} + \%c \right)^2$ 
```

```
(%i3) method;
```

```
(%o3)      bernoulli
```

```
(%i4) de1: 'diff(y, x) + y/x = -x*y^2;
```

```
(%o4)       $\frac{d}{dx}y + \frac{y}{x} = -xy^2$ 
```

```
(%i5) ode2(de1, y, x);
```

```
(%o5)       $y = \frac{1}{x(x + \%c)}$ 
```

3.8.3.6 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.

3.8.3.7 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 :

1. r_1, r_2 — вещественные и различные. Решение представляется в форме $y = k_1 \cdot e^{r_1 \cdot x} + k_2 \cdot e^{r_2 \cdot x}$.
2. $r_1 = r_2$ — корни вещественные одинаковые. Решение представляется в форме $y = (k_1 + k_2 \cdot x) e^{r_1 \cdot x}$.
3. 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 однородного 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)
$$2 \left(\frac{d^2}{dx^2} y \right) - \frac{d}{dx} y - y = 4x e^{2x}$$

(%i2) ode2(de1, y, x);

(%o2)
$$y = \frac{(20x - 28) e^{2x}}{25} + \%k1 e^x + \%k2 e^{-\frac{x}{2}}$$

Частное решение неоднородного уравнения сохраняется в переменной *yp*:

(%i3) yp;

(%o3)
$$\frac{(20x - 28) e^{2x}}{25}$$

Неоднородное уравнение с кратными корнями характеристического уравнения:

(%i1) de2: 'diff(y, x, 2) - 2*'diff(y, x) + y = x*exp(x);

(%o1)
$$\frac{d^2}{dx^2} y - 2 \left(\frac{d}{dx} y \right) + y = x e^x$$

(%i2) ode2(de2, y, x);

(%o2)
$$y = \frac{x^3 e^x}{6} + (\%k2 x + \%k1) e^x$$

(%i3) yp;

(%o3)
$$\frac{x^3 e^x}{6}$$

Неоднородное уравнение с комплексными корням:

(%i4) de3: 'diff(y, x, 2) + y = x*sin(x);

$$(\%04) \quad \frac{d^2}{dx^2} y + y = x \sin(x)$$

(%i5) ode2(de3, y, x);

$$(\%05) \quad y = \frac{2x \sin(x) + (1 - 2x^2) \cos(x)}{8} + \%k1 \sin(x) + \%k2 \cos(x)$$

(%i6) yr;

$$(\%06) \quad \frac{2x \sin(x) + (1 - 2x^2) \cos(x)}{8}$$

3.8.3.8 Уравнения с переменными коэффициентами

Аналогично уравнению с постоянными коэффициентами, общее решение однородного уравнения $y'' + p(x)y' + q(x)y = 0$ имеет вид $y = C_1 y_1 + C_2 y_2$, 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:

(%i3) difur: x^2*'diff(y, x, 2) - x*'diff(y, x) = 3*x^3;

$$(\%03) \quad x^2 \left(\frac{d^2}{dx^2} y \right) - x \left(\frac{d}{dx} y \right) = 3x^3$$

(%i4) ode2(difur, y, x);

$$(\%04) \quad y = x^3 + \%k2 x^2 - \frac{\%k1}{2}$$

Пример:

(%i3) difur1: x*'diff(y, x, 2) + 'diff(y, x) = x^2;

$$(\%03) \quad x \left(\frac{d^2}{dx^2} y \right) + \frac{d}{dx} y = x^2$$

(%i4) ode2(difur1, y, x);

$$(\%04) \quad y = \%k1 \log(x) + \frac{x^3}{9} + \%k2$$

3.8.3.9 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 $y = k_1x^r + k_2\ln(x)x^r$.

Неоднородное уравнение типа Эйлера сводится 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 (d^2/dx^2 y) + x (d/dx y) + 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))
```

3.8.3.10 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)      d^2/dx^2 y + y (d/dx y)^3 = 0
```

```
(%i2) ode2(%,y,x);
```

```
(%o2)      (y^3 + 6 %k1 y) / 6 = x + %k2
```

```
(%i3) bc2(%,x=0,y=1,x=1,y=3);
```

```
(%o3)      (y^3 - 10 y) / 6 = x - 3/2
```

3.8.4 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 непрерывные 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:

$$pF(p) - f(0) = f'(t)$$

$$p^2 F(p) - pf(0) - f'(0) = f''(t)$$

и т.д., поэтому использование преобразования Лапласа для решения систем ОДУ требует задания начальных условий.

Использование *desolve* ограничивается одним из свойств преобразования Лапласа: если $Lf(t) = F(s)$, то $Ltf(t) = -F(s)$. 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)      
$$\frac{d}{dx} f(x) = \frac{d}{dx} g(x) + \sin(x)$$

```

```
(%i2) de2:diff(g(x), x, 2)=diff(f(x), x) - cos(x);
```

```
(%o2)      
$$\frac{d^2}{dx^2} g(x) = \frac{d}{dx} f(x) - \cos(x)$$

```

```
(%i3) desolve([de1, de2], [f(x), g(x)]);
```

```
(%o3) [f(x) = e^x \left( \frac{d}{dx} g(x) \Big|_{x=0} \right) - \frac{d}{dx} g(x) \Big|_{x=0} + f(0),
```

```
g(x) = e^x \left( \frac{d}{dx} g(x) \Big|_{x=0} \right) - \frac{d}{dx} 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)      
$$\frac{d^2}{dx^2} f(x) + f(x) = 2x$$

```

```
(%i2) desolve(de3, f(x));
```

```
(%o2)      
$$f(x) = \sin(x) \left( \frac{d}{dx} f(x) \Big|_{x=0} - 2 \right) + f(0) \cos(x) + 2x$$

```

Для указания начальных условий используется функция *atvalue*.

Синтаксис вызова:

```
tvalue(expr, [x1 = a1, ..., xm = am], c)atvalue(expr, x1 = a1, c)
```

Функция *atvalue* присваивает значение *c* выражению *expr* в точке $x = a$. Выражение *expr* — функция $f(x_1, \dots, x_m)$ или производной $\text{diff}(f(x_1, \dots, x_m), x_1, n_1, \dots, x_n, n_n)$ 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)      
$$\frac{d}{dx} f(x) = \frac{d}{dx} g(x) + \sin(x)$$

```

```
(%i2) de2:diff(g(x), x, 2)=diff(f(x), x) - cos(x);
```

$$(\%02) \quad \frac{d^2}{dx^2} g(x) = \frac{d}{dx} f(x) - \cos(x)$$

(%i3) atvalue(f(x), x=0, 1);

$$(\%03) \quad 1$$

(%i4) atvalue(g(x), x=0, 2);

$$(\%04) \quad 2$$

(%i5) atvalue(diff(g(x), x), x=0, 3);

$$(\%05) \quad 3$$

(%i6) desolve([de1, de2], [f(x), g(x)]);

$$(\%06) \quad [f(x) = 3e^x - 2, g(x) = \cos(x) + 3e^x - 2]$$

Управление начальными условиями осуществляется при помощи функций *properties* и *printprops*. 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([a1, ..., an], 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 *a1, ..., an* осуществляется вызовом *remove(a1, p1, ..., an, pn)*; удаление списка свойств — вызовом *remove([a1, ..., am], [p1, ..., pn], ...)*).

Пример синтаксиса и использования рассмотренных функций:

(%i1) eq1: 'diff(f(x), x) = 'diff(g(x), x) + sin(x);

$$(\%01) \quad \frac{d}{dx} f(x) = \frac{d}{dx} g(x) + \sin(x)$$

(%i2) eq2: 'diff(g(x), x, 2) = 'diff(f(x), x) - cos(x);

$$(\%02) \quad \frac{d^2}{dx^2} g(x) = \frac{d}{dx} f(x) - \cos(x)$$

(%i3) atvalue('diff(g(x), x), x=0, a);

$$(\%03) \quad a$$

(%i4) atvalue(f(x), x=0, 1);

$$(\%04) \quad 1$$

(%i5) properties(f);

$$(\%o2) \quad x \left(\frac{d}{dx} y \right)^2 - (xy + 1) \left(\frac{d}{dx} y \right) + y = 0$$

(%i3) contrib_ode(eqn, y, x);

$$(\%t3) \quad x \left(\frac{d}{dx} y \right)^2 - (xy + 1) \left(\frac{d}{dx} y \right) + y = 0$$

first order equation not linear in y'

$$(\%o3) \quad [y = \log(x) + \%c, y = \%c e^x]$$

(%i4) method;

$$(\%o4) \quad \text{factor}$$

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.

3.8.5.2 Уравнения Клеро и Лагранжа

Уравнение Клеро

(%i1) load("contrib_ode")\$

(%i2) eqn: 'diff(y, x)^2+x*'diff(y, x)-y=0;

$$(\%o2) \quad \left(\frac{d}{dx} y \right)^2 + x \left(\frac{d}{dx} y \right) - y = 0$$

(%i3) contrib_ode(eqn, y, x);

$$(\%t3) \quad \left(\frac{d}{dx} y \right)^2 + x \left(\frac{d}{dx} y \right) - y = 0$$

first order equation not linear in y'

$$(\%o3) \quad [y = \%c x + \%c^2, y = -\frac{x^2}{4}]$$

(%i4) method;

$$(\%o4) \quad \text{clairault}$$

Уравнение Лагранжа

(%i5) leq: y=(1+'diff(y, x))*x+('diff(y, x))^2;

$$(\%o5) \quad y = \left(\frac{d}{dx} y \right)^2 + x \left(\frac{d}{dx} y + 1 \right)$$

(%i6) contrib_ode(leq, y, x);

$$(\%t6) \quad y = \left(\frac{d}{dx} y \right)^2 + x \left(\frac{d}{dx} y + 1 \right)$$

first order equation not linear in y'

```
(%o6)
[[x = e^{-%t} (%c - 2 (%t - 1) e^{%t}), y = (%t + 1) 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) 
$$\frac{d}{dx} y = (y + x)^2$$

```

```
(%i9) contrib_ode(eqn, y, x);
```

```
(%o9) [[x = %c - atan(sqrt(%t)), y = -x - sqrt(%t)],
```

```
[x = atan(sqrt(%t)) + %c, y = sqrt(%t) - x]]
```

```
(%i10) method;
```

```
(%o10) lagrange
```

3.8.5.3 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) 
$$(-y + 2x + 4) \left( \frac{d}{dx} y \right) - 2y + x + 5 = 0$$

```

```
(%i12) contrib_ode(eqn, y, x);
```

```
(%o12) 
$$\left[ \frac{\log\left(3 - \frac{2(2x+4)-x-5}{-y+2x+4}\right) - 3\log\left(1 - \frac{2(2x+4)-x-5}{-y+2x+4}\right) + 2\log\left(-\frac{2(2x+4)-x-5}{4(-y+2x+4)}\right)}{2} = \right.$$


$$\left. \log(x + 1) + %c \right]$$

```

```
(%i13) method;
```

```
(%o13) abel2
```

```
(%i14) eqn1: 'diff(y, x) = (1-3*x-3*y) / (1+x+y);
```

```
(%o14) 
$$\frac{d}{dx} y = \frac{-3y - 3x + 1}{y + x + 1}$$

```

```
(%i15) contrib_ode(eqn1, y, x);
```

```
(%o15) 
$$\left[ \frac{2\log(\bar{y} + x - 1) + y + 3x}{2} = %c \right]$$

```

```
(%i16) method;
```

```
(%o16) lie
```

3.8.5.4 Решение однородных линейных уравнений

Другие полезные функции пакета `contrib_ode`: *odelin* и *odecheck*.

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);
```

```
...trying factor method...
solving 7 equations in 4 variables...
trying the Bessel solver...solving 1 equations in 2
variables...
trying the F01 solver...
solving 1 equations in 3 variables...
trying the spheroidal wave solver...
solving 1 equations in 4 variables...
trying the square root Bessel solver...
solving 1 equations in 2 variables...
trying the 2F1 solver...
solving 9 equations in 5 variables
```

```
(%o4)

$$\frac{\text{gauss}_a(-6, -2, -3, -x)}{x^4}, \frac{\text{gauss}_b(-6, -2, -3, -x)}{x^4}$$

```

Примечание: функции gauss_a и gauss_b — специальные функции, представляющие собой решения гипергеометрического уравнения.

Функция *odecheck* позволяет подставить в ОДУ найденное решение.

Пример:

```
(%i1) load("contrib_ode")$
(%i2) eqn:(1+x^2)*'diff(y,x,2)-2*x*'diff(y,x);
```

```
(%o2)

$$(x^2 + 1) \left( \frac{d^2}{dx^2} y \right) - 2x \left( \frac{d}{dx} y \right)$$

```

```
(%i3) odelin(eqn,y,x);
...trying factor method...solving 7 equations in 4
variables
```

```
(%o3)

$$1, x (x^2 + 3)$$

```

```
(%i4) ode_check(eqn,y=x*(x^2+3));
```

```
(%o4)

$$0$$

```

```
(%i5) ode_check(eqn,y=1);
```

```
(%o5)

$$0$$

```

3.8.6 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-Кутты.

Начиная с версии 5.12, **Maxima** включает пакет `dynamics` (его необходимо загружать перед использованием). Помимо метода Рунге-Кутты, пакет `dynamics` включает ряд функций для построения различных фракталов.

Метод Рунге-Кутты реализует функция `rk`. Синтаксис вызова её вызова: `rk([eq],[vars],[init],[trange])`, где `eq` — список правых частей уравнений; `vars` — список зависимых переменных; `init` — список начальных значений; `trange` — список `[t, t0, tend, ht]`, содержащий символьное обозначение независимой переменной (`t`), её начальное значение (`t0`), конечное значение (`tend`), шаг интегрирования (`ht`).

Пример:

Решить ОДУ

$$\frac{dx}{dt} = 4x^2 - 4y^2; \quad \frac{dy}{dt} = y^2 - x^2 + 1;$$

при $t = [0 \dots 4]$, $x(0) = -1,25$, $y(0) = 0,75$.

Используем пакет `dynamics`.

```
(%i1) load('dynamics')$
```

Выбираем шаг интегрирования 0,02.

```
(%i2) sol:rk([4*x^2-4*y^2,y^2-x^2+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")$
```

```
(%i2) rp1:4*x^2-4*y^2;
```

```
(%o2) 4x^2 - 4y^2
```

```
(%i3) rp2:y^2-x^2+1;
```

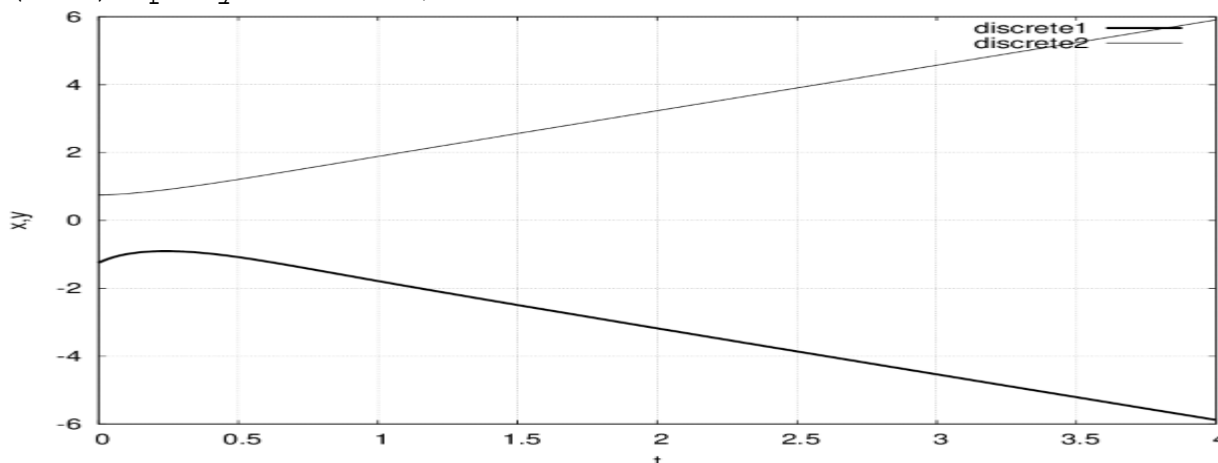


Рис. 3.16. Пример графического решения системы ОДУ численным методом


```
(%o3)          y2 - x2 + 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. 3.16

Аналогичный, хотя и несколько более сложный пример — моделирование аттрактора Лоренца.

3.9 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.

3.9.1 Понятие ряда Фурье

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 выполняется условие

$$\int_a^b f(x)g(x)\rho(x)dx = 0,$$

где $\rho(x)$ — весовая функция.

Функциональная последовательность

$\{\varphi_n(x)\} = \{\varphi_0(x), \varphi_1(x), \dots, \varphi_n(x), \dots\}$ называется ортогональной

на $[a, b]$, если выполняется условие:

$$\int_a^b \varphi_n(x)\varphi_m(x)\rho(x)dx = 0, \forall n \neq m.$$

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 из тригонометрических функций $1, \cos(x), \sin(x), \cos(2x), \sin(2x), \dots, \cos(nx), \sin(nx), \dots$ ортогональна на отрезке $[-\pi, \pi]$ с весовой функцией $\rho(x) = 1$.

Проверим свойство ортогональности, вычисляя соответствующие интегралы. При $m \neq n$ получаем:

$$\begin{aligned} \int_{-\pi}^{\pi} 1 \cdot \sin(nx) dx &= -\frac{\cos(nx)}{n} \Big|_{-\pi}^{\pi} = 0, \\ \int_{-\pi}^{\pi} 1 \cdot \sin(nx) dx &= -\frac{\cos(nx)}{n} \Big|_{-\pi}^{\pi} = 0, \\ \int_{-\pi}^{\pi} \cos(nx) dx &= \frac{1}{n} \sin(nx) \Big|_{-\pi}^{\pi} = 0, \forall n \in N; \\ \int_{-\pi}^{\pi} \cos(mx) \cdot \cos(nx) dx &= \\ \frac{1}{2} \int_{-\pi}^{\pi} (\cos((m-n)x) + \cos((m+n)x)) dx &= \\ \frac{1}{2} \left(\frac{\sin((m-n)x)}{m-n} + \frac{\sin(m+n)x}{m+n} \right) \Big|_{-\pi}^{\pi} &= 0 \end{aligned}$$

Если же $m = n$, то

$$\begin{aligned} \int_{-\pi}^{\pi} \cos^2(mx) dx &= \frac{1}{2} \int_{-\pi}^{\pi} (1 + \cos(2mx)) dx = \\ \frac{1}{2} \left(x + \frac{\sin(2mx)}{2m} \right) \Big|_{-\pi}^{\pi} &= \pi \end{aligned}$$

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 подинтегральная 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 $[-\pi, \pi]$.

Другой широко используемой последовательностью ортогональных функций является последовательность полиномов Лежандра. Полином Лежандра степени n можно представить через формулу Родрига в виде:

$$P_n(z) = \frac{1}{2^n n!} \frac{d^n}{dz^n} (z^2 - 1)^n.$$

Они также могут быть вычислены по рекуррентной формуле:

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x).$$

Полиномы Лежандра ортогональны на отрезке $[-1, 1]$ с весом $\rho(x) = 1$:

$$\int_{-1}^1 P_k(x) P_l(x) dx = \begin{cases} \frac{2}{2k+1}, & \text{если } k = l \\ 0, & \text{если } k \neq l \end{cases}.$$

Ещё одной важной последовательностью ортогональных функций является последовательность полиномов Чебышёва. Полиномы Чебышёва первого рода $T_n(x)$ степени n можно определить с помощью равенства:

$$T_n(\cos(\theta)) = \cos(n\theta),$$

или, что почти эквивалентно,

$$T_n(z) = \cos(n \arccos(z)).$$

Они также могут быть вычислены по рекуррентной формуле:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Полиномы Чебышёва ортогональны на отрезке $[-1, 1]$ с весом

$$\rho(x) = \frac{1}{\sqrt{1-x^2}}.$$

$$\int_{-1}^1 T_k(x) T_l(x) \frac{1}{\sqrt{1-x^2}} dx = \begin{cases} \frac{\pi}{2}, & \text{если } k = l \neq 0 \\ \pi, & \text{если } k = l = 0 \\ 0, & \text{если } k \neq l \end{cases}$$

3.9.2 Вычисление коэффициентов тригонометрических рядов Фурье

Члены тригонометрического ряда

$$\frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(nx) + b_n \sin(nx))$$

являются периодическими функциями с общим периодом 2π , поэтому и сумма этого ряда $S(x)$ также будет периодической функцией с периодом 2π .

Предположим, что 2π -периодическую функцию $f(x)$ можно разложить в тригонометрический ряд, равномерно сходящийся на отрезке $[-\pi, \pi]$.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(nx) + b_n \sin(nx)) \quad (3.1)$$

Рассмотрим вопрос об определении коэффициентов a_0, a_n и $b_n (n = 1, 2, \dots)$. 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,

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx. \quad (3.2)$$

Для того чтобы найти $a_n (n = 1, 2, \dots)$, обе части этого равенства умножим на $\cos(mx)$ и проинтегрируем на отрезке $[-\pi, \pi]$. Поскольку система тригонометрических функций ортогональна, то

$$\int_{-\pi}^{\pi} \cos(mx) \cos(nx) dx = 0, \quad \int_{-\pi}^{\pi} \cos(mx) \sin(nx) dx = 0$$

для $\forall m, n \in \mathbb{N}$, 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

$$\int_{-\pi}^{\pi} f(x) \cos(nx) dx = a_n \int_{-\pi}^{\pi} \cos^2(nx) dx = \pi a_n,$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx, \quad n = 1, 2, \dots$$

откуда

Аналогично, умножив обе части равенства на $\sin(mx)$ и проинтегрировав на отрезке $[-\pi; \pi]$, получаем, что

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx, \quad n = 1, 2, \dots$$

Итак, если функцию $f(x)$ можно представить в виде тригонометрического ряда, то коэффициенты a_0, 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 $[a, a + 2\pi]$, $a \in \mathbb{R}$, 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 кусочногладкая на $[-\pi; \pi]$ функция, то её ряд Фурье сходится в любой точке этого отрезка и его сумма равна:

1. значению функции $f(x)$, когда x — точка непрерывности функции $f(x)$;

$$\frac{f(x-0) + f(x+0)}{2}$$
2. $\frac{f(x-0) + f(x+0)}{2}$, когда x — точка разрыва функции $f(x)$, при этом

$$\frac{f(x-0) + f(x+0)}{2} = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(nx) + b_n \sin(nx)).$$

Отметим, что на практике чаще всего встречаются 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;
(%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]);
a1 = 0 b1 = 2 a2 = 0 b2 = -1 a3 = 0 b3 = 2/3 a4 = 0 b4 = -1/2 a5 = 0 b5 = 2/5
(%o6) 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) := 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 в сравнении с графиком частичной суммы ряда Фурье $fun(x)$ изображён на рис. 3.17.

3.9.3 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 верно равенство $f(-x) \cos(-nx) = f(x) \cos(nx)$, а $f(x) \sin(nx)$ — нечётная функция, так как $(-x) \sin(-nx) = -f(x) \sin(nx)$. Поэтому коэффициенты ряда Фурье a_n, b_n равны:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx = \frac{2}{\pi} \int_0^{\pi} f(x) \cos(nx) dx \quad (n = 0, 1, \dots),$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx = 0 \quad (n = 1, 2, \dots).$$

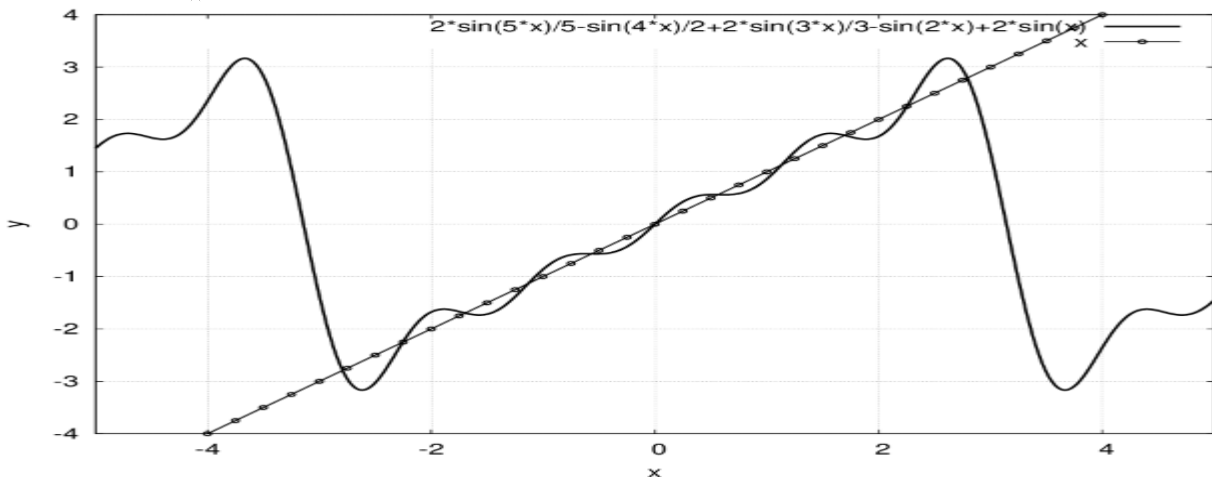


Рис. 3.17. График функции $y = f(x)$ и суммы первых пяти членов ряда Фурье

Следовательно, ряд Фурье чётной функции содержит только косинусы, т.е.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx)$$

.. Аналогично, если $f(x)$ — нечётная функция, то $f(x) \cos(nx)$ — нечётная, а $f(x) \sin(nx)$ — чётная функция.

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx = 0 \quad (n = 0, 1, \dots),$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx = \frac{2}{\pi} \int_0^{\pi} f(x) \sin(nx) dx \quad (n =$$

Поэтому $1, 2, \dots)$

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 2π функцию, заданную на отрезке $[-\pi, \pi]$ равенством $f(x) = x^2$.

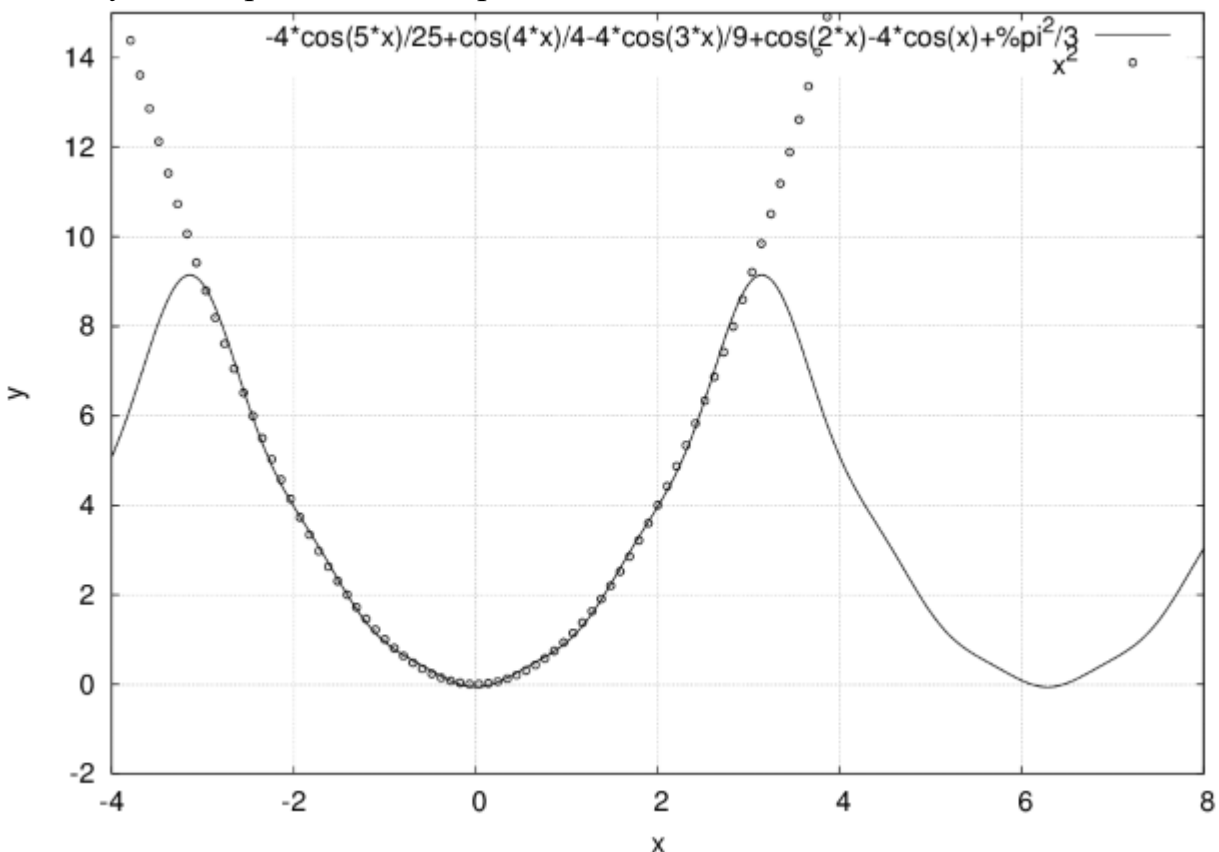


Рис. 3.18. График функции (точки) и суммы первых пяти членов ряда Фурье (сплошная линия)

Данная функция $y = x^2$ является чётной (рис. 3.18), поэтому её ряд Фурье содержит только косинусы. Вычисляем коэффициенты этого ряда: $b_n = 0, n = 1, 2, \dots$

Для вычисления коэффициентов a_n ряда Фурье создаём функцию *fun*, входными параметрами которой являются имя независимой переменной (в примере это x), число суммируемых членов ряда (n , в дальнейшем функция вызывается при $n = 5$) и символьное выражение, определяющее функцию, для которой строится разложение (f , функция *fun* вызывается с $f = x^2$).

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))$
(%i2) fun(x,5,x^2);
```

```
(%o2) 
$$-\frac{4 \cos(5x)}{25} + \frac{\cos(4x)}{4} - \frac{4 \cos(3x)}{9} + \cos(2x) - 4 \cos(x) + \frac{\pi^2}{3}$$

```

Для аналитического вычисления коэффициентов ряда Фурье функции $y = |x|$ функцию *fun* необходимо немного изменить, предусмотрев различные выражения для подинтегрального выражения на полуинтервалах $[-\pi, 0)$ и $(0, \pi]$ (выражения f_1 и f_2 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))$
```

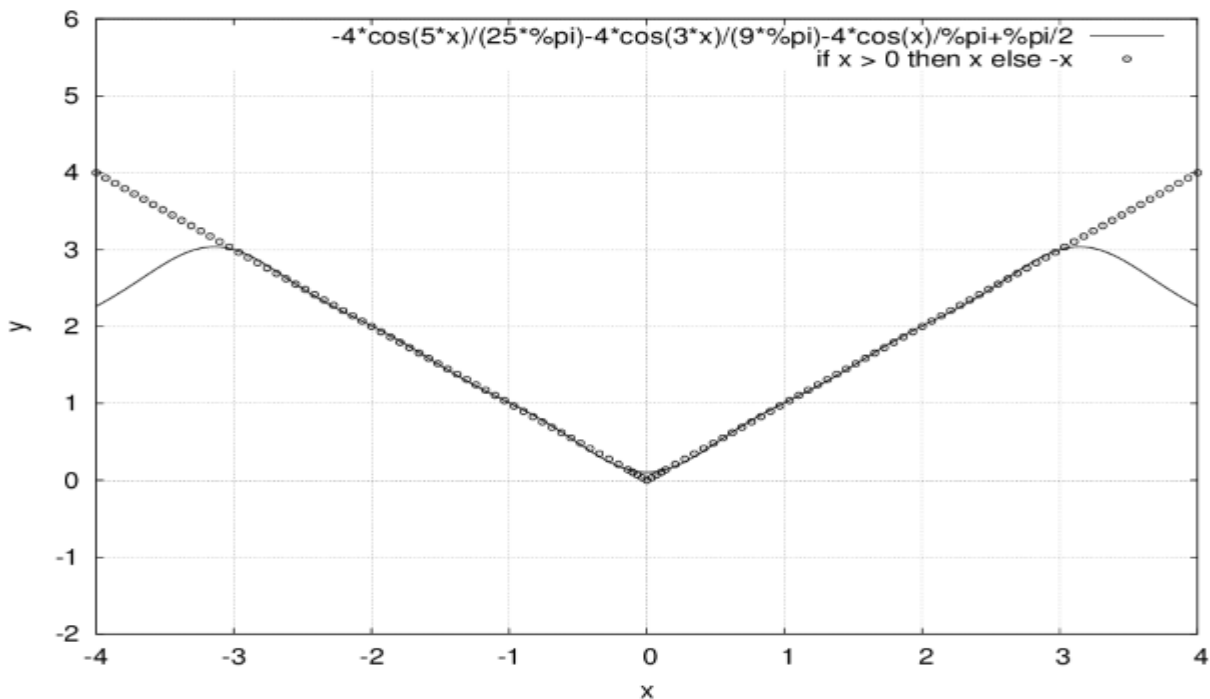


Рис. 3.19. График функции $y = |x|$ (точки) и суммы первых пяти членов ряда Фурье (сплошная линия)

Функция является $y = |x|$ также является чётной (рис. 3.19), поэтому её ряд Фурье содержит только косинусы.

Результаты вычисления коэффициентов ряда Фурье для этой функции:

```
(%i1) fun12(x, 5, -x, x);
```

$$(\%o1) \quad -\frac{4 \cos(5x)}{25\pi} - \frac{4 \cos(3x)}{9\pi} - \frac{4 \cos(x)}{\pi} + \frac{\pi}{2}$$

Для построения графика функции $y = |x|$ создаём функцию $fg(x)$, которая использована для построения графика на рис. 3.19.

```
(%i3) fg(x):=if x>0 then x else -x$
```

3.9.4 Разложение функций в ряд Фурье на отрезке $[0, \pi]$

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 интервале $[-\pi, 0]$. Рассмотрим два случая:

Функцию $f(x)$, заданную на $[0, \pi]$, продолжим на интервал $[-\pi, 0]$ так, что вновь полученная функция $f_1(x)$, была чётной:

$$f_1 = \begin{cases} f(-x), & \text{если } x \in [-\pi, 0] \\ f(x), & \text{если } x \in [0, \pi] \end{cases}.$$

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 интервал $[-\pi, 0]$ нечётным образом:

$$f_2 = \begin{cases} -f(-x), & \text{если } x \in [-\pi, 0[\\ f(x), & \text{если } x \in [0, \pi] \end{cases}.$$

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 $f_2(x)$ 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 $f_1(x)$ по формуле:

$$f_1(x) = \begin{cases} -2x + 1, & \text{если } x \in [-\pi, 0[\\ 2x + 1, & \text{если } x \in [0, \pi] \end{cases}.$$

Вычисляем коэффициенты Фурье для этой функции при помощи функции `fun12`:

```
(%i1) fleft:-2*x+1;
```

```
(%o1) 1 - 2x
```

```
(%i2) fright:2*x+1;
```

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

```
(%i3) funcos(x, 7, fleft, fright);
```

```
(%o3) 
$$\frac{8 \cos(7x)}{49\pi} + \frac{8 \cos(5x)}{25\pi} + \frac{8 \cos(3x)}{9\pi} + \frac{8 \cos(x)}{\pi} + \frac{2\pi^2 + 2\pi}{2\pi}$$

```

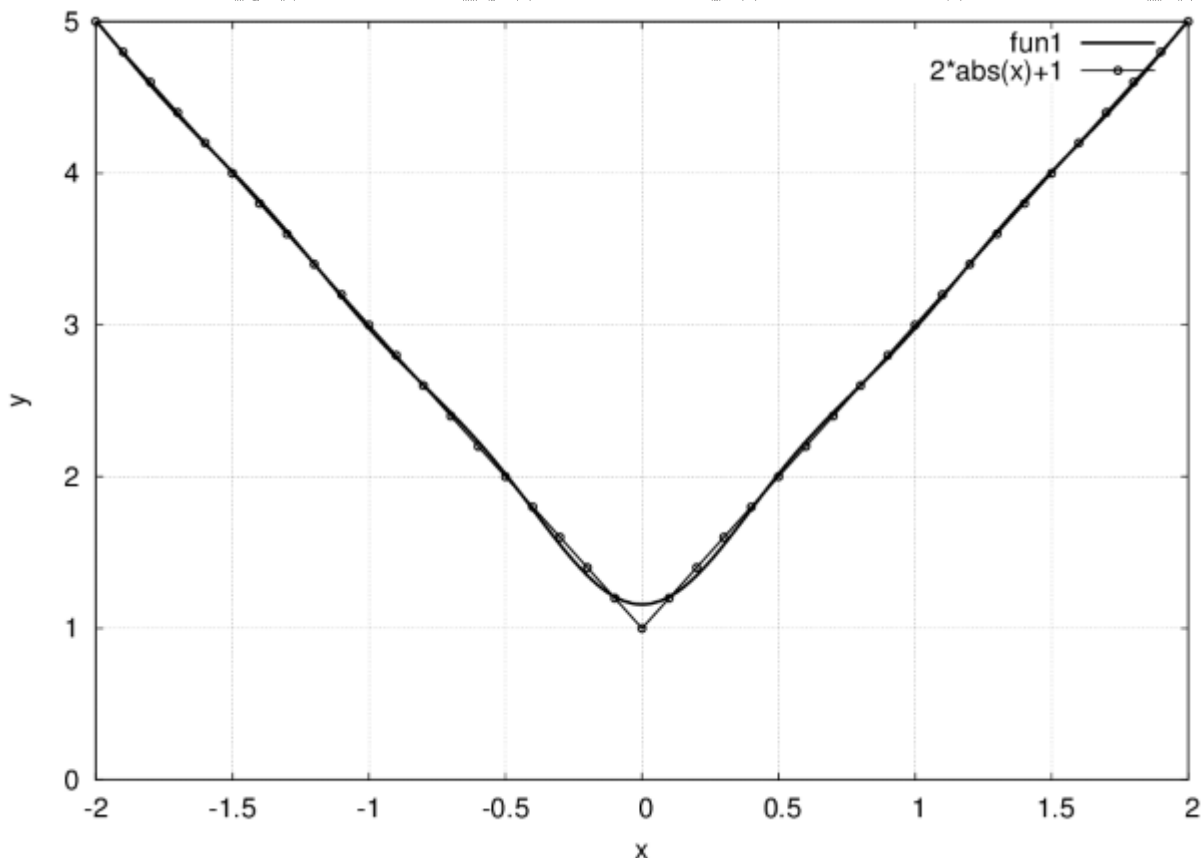


Рис. 3.20. График функции $y = 2|x| + 1$, продолженной чётным образом, и суммы семи членов соответствующего ряда

Графическое сопоставление результатов суммирования ряда Фурье и аналитического выражения заданной функции представлены на рис. 3.20

2) Функцию $f(x)$ продолжим на $[-\pi, 0]$ нечётным образом. Составим новую функцию $f_2(x)$ по формуле $f_2(x) = \begin{cases} 2x-1, & x \in [-\pi, 0[\\ 2x+1, & x \in [0, \pi] \end{cases}$

Вычислим коэффициенты Фурье для этой функции, используя функцию `fun12sin`, аналогичную приведённой выше.

Пример:

```
(%i1) fleft:2*x-1$
(%i2) fright:2*x+1$
(%i3) f(x):=(if x>0 then fright else fleft)$
(%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))$
(%i5) fun12sin(x,7,fleft,fright);
(%o5) 
$$\frac{\left(\frac{2(2\pi+1)}{7}+\frac{2}{7}\right)\sin(7x)}{\pi} + \frac{\left(\frac{1}{3}-\frac{2\pi+1}{3}\right)\sin(6x)}{\pi} + \frac{\left(\frac{2(2\pi+1)}{5}+\frac{2}{5}\right)\sin(5x)}{\pi} +$$


$$\frac{\left(\frac{1}{2}-\frac{2\pi+1}{2}\right)\sin(4x)}{\pi} + \frac{\left(\frac{2(2\pi+1)}{3}+\frac{2}{3}\right)\sin(3x)}{\pi} - 2\sin(2x) + \frac{(4\pi+4)\sin(x)}{\pi}$$

```

Графическое сопоставление результатов суммирования ряда Фурье и аналитического выражения заданной функции представлены на рис. 3.21

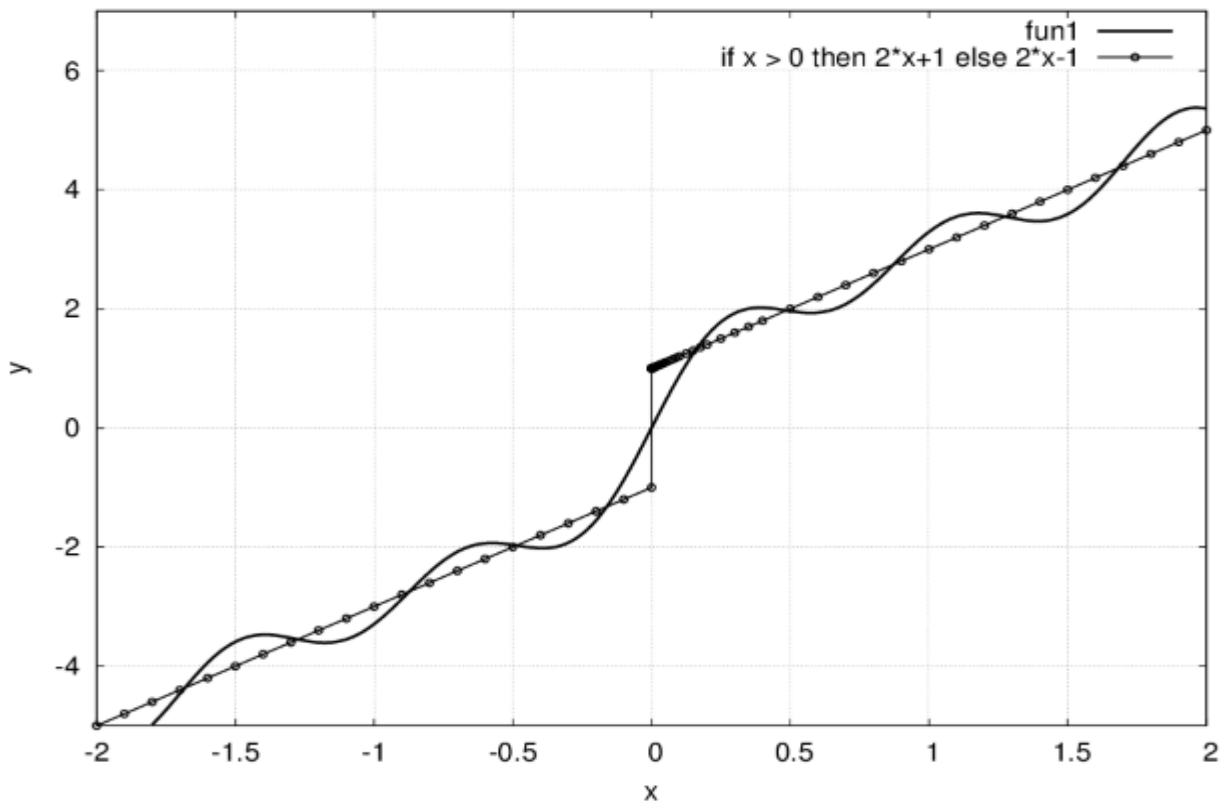


Рис. 3.21. Сравнение графика функции $y = 2x + 1$ при нечётном продолжении и суммы семи членов соответствующего ряда Фурье

3.9.5 Ряд Фурье для функций с периодом 2ℓ

Пусть $f(x)$ — периодическая 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

$$x = \frac{\ell t}{\pi}. \quad (3.3)$$

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 $[-\pi, \pi]$

$$\varphi(t) = f\left(\frac{\ell t}{\pi}\right) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(nt) + b_n \sin(nt)) \dots \quad (3.4)$$

Коэффициенты этого ряда вычисляются по формулам:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right) \cos(nt) dt, \quad n = 0, 1, \dots, \quad (3.5)$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right) \sin(nt) dt, \quad n = 1, 2, \dots \quad (3.6)$$

Возвращаясь к прежней переменной x , из равенства (3.3) имеем $t = \frac{\pi x}{\ell}$. Тогда ряд (3.4) можно представить в виде

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos\left(\frac{n\pi x}{\ell}\right) + b_n \sin\left(\frac{n\pi x}{\ell}\right) \right). \quad (3.7)$$

В интегралах (3.5) и (3.6) произведём замену переменной:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right) \cos(nt) dt = \frac{1}{\ell} \int_{-\ell}^{\ell} f(x) \cos\left(\frac{n\pi x}{\ell}\right) dx, \quad n = 0, 1, \dots$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right) \sin(nt) dt = \frac{1}{\ell} \int_{-\ell}^{\ell} f(x) \sin\left(\frac{n\pi x}{\ell}\right) dx, \quad n = 1, 2, \dots$$

Если $f(x)$ — чётная на $[-\ell, \ell]$ функция, то $b_n = 0 (n = 1, 2, \dots)$, а $a_n = \frac{2}{\ell} \int_0^\ell f(x) \cos\left(\frac{n\pi x}{\ell}\right) dx, (n = 0, 1, \dots)$, ряд Фурье такой функции имеет вид:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{\ell}\right).$$

Если $f(x)$ — нечётная на $[-\ell, \ell]$ функция, то $a_n = 0 (n = 0, 1, 2, \dots)$, $b_n = \frac{2}{\ell} \int_0^\ell f(x) \sin\left(\frac{n\pi x}{\ell}\right) dx, (n = 1, 2, \dots)$, ряд Фурье имеет вид:

$$f(x) = \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi x}{\ell}\right).$$

Пример: Разложить в ряд Фурье периодическую с периодом $T = 2$ функцию $f(x)$, заданную формулой

$$f(x) = \begin{cases} x, & \text{если } 0 < x \leq 1 \\ 0, & \text{если } -1 < x \leq 0 \end{cases}.$$

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 $x = \pm 1, \pm 3, \dots$ 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 сравнению с вариантами, приведёнными выше — использование тригонометрических функций $\sin\left(\frac{\pi k x}{\ell}\right)$ и $\cos\left(\frac{\pi k x}{\ell}\right)$

Вывод **Maxima** для первых семи членов ряда Фурье:

```
(%i6) fun121(x, 7, 1, 0, x);
(%o6) 
$$\frac{\sin(7\pi x) 2\cos(7\pi x) \sin(6\pi x) \sin(5\pi x) 2\cos(5\pi x) \sin(4\pi x) \sin(3\pi x) 2\cos(3\pi x) \sin(2\pi x) \sin(\pi x) 2\cos(\pi x) 1}{7\pi \quad 49\pi^2 \quad 6\pi \quad 5\pi \quad 25\pi^2 \quad 4\pi \quad 3\pi \quad 9\pi^2 \quad 2\pi \quad \pi \quad \pi^2} 1$$

```

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:

```
(%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. 3.22.

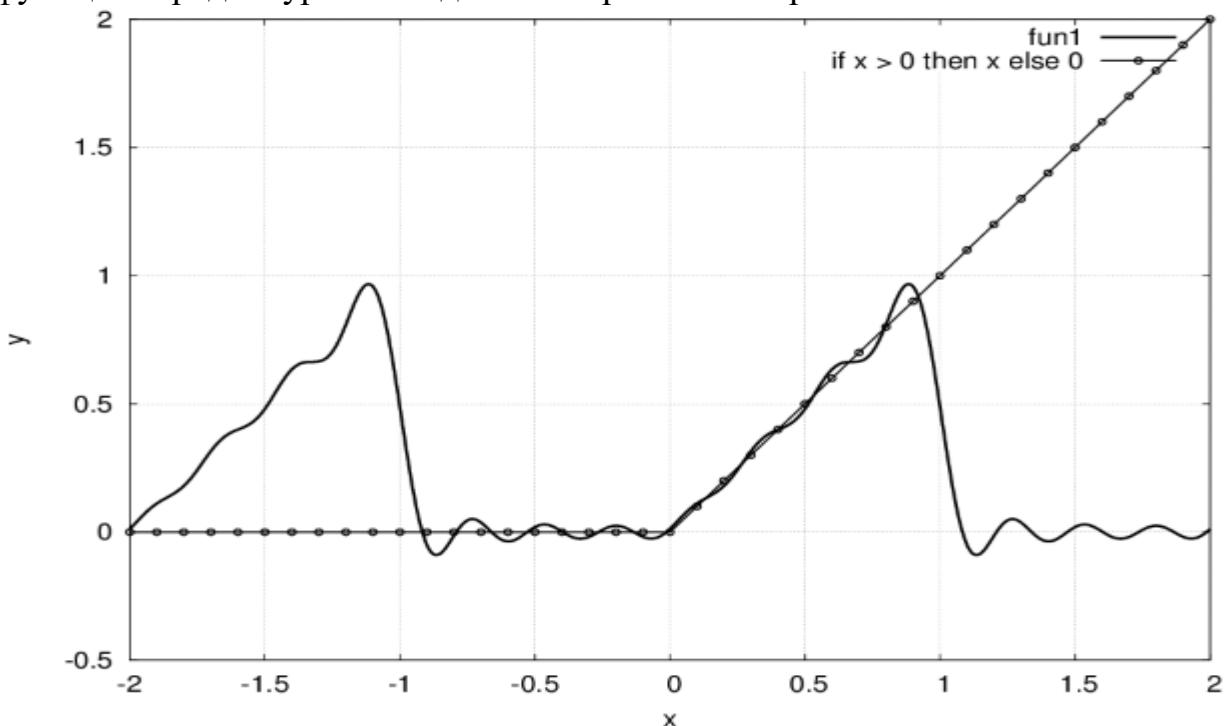


Рис. 3.22. График функции $f(x)$ и суммы первых семи членов ряда Фурье

3.9.6 Комплексная форма ряда Фурье

Пусть функция $f(x)$ на $[-\pi, \pi]$ разложена в ряд Фурье

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(x) + b_n \sin(x)). \quad (3.8)$$

Воспользуемся формулами Эйлера:

$$\cos(nx) = \frac{e^{inx} + e^{-inx}}{2}, \quad \sin(nx) = \frac{e^{inx} - e^{-inx}}{2i}.$$

Подставим эти выражения в ряд (3.8), имеем:

$$\begin{aligned}
 f(x) &= \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \frac{e^{inx} + e^{-inx}}{2} + b_n \frac{e^{inx} - e^{-inx}}{2i} \right) = \\
 &= \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \frac{e^{inx} + e^{-inx}}{2} - ib_n \frac{e^{inx} - e^{-inx}}{2} \right) = \\
 &= \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(\frac{a_n - ib_n}{2} \cdot e^{inx} + \frac{a_n + ib_n}{2} \cdot e^{-inx} \right).
 \end{aligned}$$

Обозначим:

$$\frac{a_0}{2} = c_0, \quad \frac{a_n - ib_n}{2} = c_n, \quad \frac{a_n + ib_n}{2} = c_{-n}. \quad (3.9)$$

Тогда

$$\begin{aligned}
 f(x) &= c_0 + \sum_{n=1}^{\infty} (c_n \cdot e^{inx} + c_{-n} e^{-inx}) = \\
 &= c_0 + \sum_{n=1}^{\infty} c_n e^{inx} + \sum_{n=1}^{\infty} c_{-n} e^{-inx} = \\
 &= c_0 + \sum_{n=1}^{\infty} c_n e^{inx} + \sum_{n=-\infty}^{-1} c_n e^{inx} = \sum_{n=-\infty}^{\infty} c_n e^{inx}.
 \end{aligned}$$

Следовательно

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx} \quad (3.10)$$

Выражение (3.10) называется комплексной формой ряда Фурье функции $f(x)$ с комплексными коэффициентами Фурье c_n . Коэффициенты Фурье c_n вычисляются по формулам ($n = 0, \pm 1, \pm 2, \dots$):

$$\begin{aligned}
 c_n &= \frac{1}{2} (a_n - ib_n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) [\cos(nx) - i \sin(nx)] dx = \\
 &= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) [\cos(-nx) + i \sin(-nx)] dx = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx.
 \end{aligned}$$

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

$$c_n = \frac{1}{2\ell} \int_{-\ell}^{\ell} f(x) e^{\frac{-in\pi x}{\ell}} dx.$$

Пример: Разложить в ряд Фурье с комплексными коэффициентами периодическую с периодом $l = 2$ функцию, заданную на отрезке $[-1, 1]$ равенством $f(x) = x^2$.

```
(%i1) n:5$ f:x^2$ l:1$ c(k):=
1/2/l*integrate(f*exp(-i*pi*k*x/l),x,-l,l)$
z:makelist(k-6,k,1,2*n+1)$
cr:makelist(c(z[k]),k,1,2*n+1)$
fk:makelist(cr[k]*exp(i*pi*z[k]*x/l),k,1,2*n+1)$
```

```
g:sum(fk[k],k,1,2*n+1)$
gend:trigreduce(ratsimp(rectform(g)));
```

(%o9)

$$\frac{-144 \cos(5 \pi x) + 225 \cos(4 \pi x) - 400 \cos(3 \pi x) + 900 \cos(2 \pi x) - 3600 \cos(\pi x) + 300 \pi^2}{900 \pi^2}$$

В данном примере члены частичной суммы ряда Фурье представляются списком. В представленном вычислении $z = -5, -4, \dots, 4, 5$. Список *sr* содержит коэффициенты ряда в комплексной форме (при суммировании от $-n$ до n индекс элемента ряда содержится в $z[k]$). Собственно члены ряда Фурье скомпонованы в список *fk*, после суммирования которого получаем сумму ряда (выражение *g*). Для построения графика $g(x)$ необходимо упростить выражение *g* (см. пример, результат упрощения — выражение *gend*). Очевидно, что для просмотра промежуточных результатов (они довольно объёмные) терминальные символы \$ можно заменить на ;.

3.9.7 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.

Функция *fourier* позволяет вычислить коэффициенты ряда Фурье (синтаксис вызова: *fourier(f, x, p)*), которая возвращает список коэффициентов Фурье $f(x)$, определённых 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 *sinnpi flag* и *cosnpi flag* (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 использования пакета *fourie* его необходимо предварительно загрузить командой $load("fourie")$.

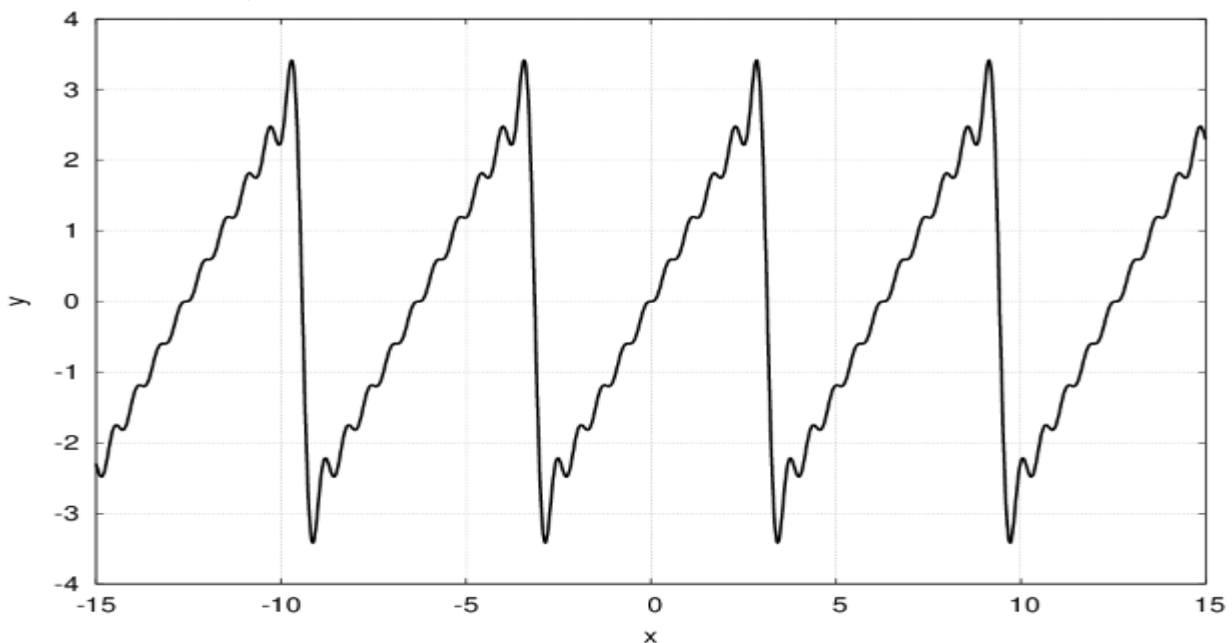


Рис. 3.23. График частичной суммы ряда Фурье для функции $f(x) = x$, построенной при помощи пакета *fourie*

Примеры использования пакета `fourie` (график полученной функции приведён на рис. 3.23):

```
(%i1) load("fourie")$ fourier(x,x,%pi);
(%t2) a0 = 0
(%t3) an = 0
(%t4) bn =  $\frac{2 \left( \frac{\sin(\pi n)}{n^2} - \frac{\pi \cos(\pi n)}{n} \right)}{\pi}$ 
(%o4) [%t2, %t3, %t4]
(%i5) foursimp(%);
(%t5) a0 = 0
(%t6) an = 0
(%t7) bn =  $-\frac{2(-1)^n}{n}$ 
(%o7) [%t5, %t6, %t7]
(%i8) fourexpand(% , x, %pi, 10);
(%o8)  $-\frac{\sin(10x)}{5} + \frac{2\sin(9x)}{9} - \frac{\sin(8x)}{4} + \frac{2\sin(7x)}{7} - \frac{\sin(6x)}{3} + \frac{2\sin(5x)}{5} - \frac{\sin(4x)}{2} + \frac{2\sin(3x)}{3} - \sin(2x) + 2\sin(x)$ 
```

3.9.8 Дополнительные возможности: обобщённые ряды Фурье

Как указывалось выше, наряду с тригонометрической ортонормированной системой функций достаточно широко используются и другие (в частности, полиномы Лежандра, Чебышёва, Эрмита и др.). Рассмотрим представление функции обобщённым рядом Фурье по полиномам Лежандра.

Вычисление значений ортогональных полиномов в **Maxima** осуществляется при помощи пакета `orthopoly`, который позволяет оперировать полиномами Чебышёва, Лежандра, Эрмита, Якоби и др., а также рядом сферических функций.

Интегрируемая на интервале $(-1, 1)$ кусочно-непрерывная функция может быть представлена обобщённым рядом Фурье (в данном случае — по полиномам Лежандра):

$$f(x) = \sum_{n=0}^{\infty} c_n P_n(x),$$

где $P_n(x)$ — полином Лежандра степени n , c_n — коэффициенты Фурье для разложения по полиномам Лежандра. Значения c_n вычисляются по формуле:

$$c_n = \frac{2n+1}{2} \int_{-1}^1 f(x) P_n(x) dx.$$

Пример вычисления разложения функции $y = e^x$ на интервале $(-1, 1)$ в ряд по полиномам Лежандра представлен следующими командами:

```
(%i1) load(orthopoly)$ n:5$ f:exp(x)$ l:1$
c(m) := (2*m+1)/2*integrate(f*legendre*p (m, x), x, -1, 1)$
```

```

z:makelist(k-1, k, 1, n+1)$
cr:makelist(c(z[k]), k, 1, n+1)$
fk:makelist(cr[k]*legendre_p (z[k], x), k, 1, n+1)$
g:sum(fk[k], k, 1, n+1)$

```

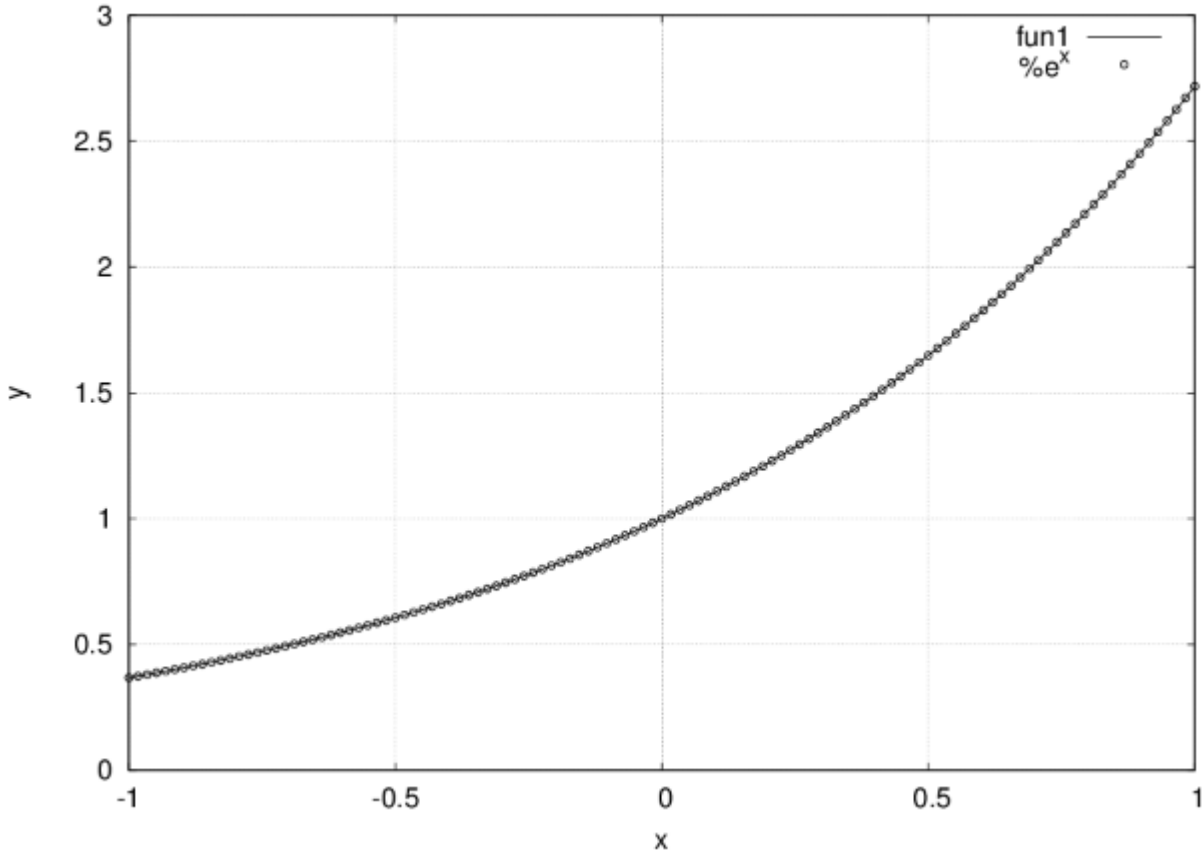


Рис. 3.24. График частичной суммы обобщённого ряда Фурье для функции

График полученного выражения g в сравнении с функцией e^x показан на рис. 3.24.

Как видно из рисунка, графики экспоненты и полученного разложения совпадают. В совпадении результатов можно убедиться, сопоставив выражение g (после упрощения) и разложение экспоненты в ряд Тейлора.

4. Численные методы и программирование с Maxima

4.1 Программирование на встроенном макроязыке

4.1.1 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 $expr_1$ elseif $cond_2$ then $expr_2$ elseif ... 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 $expr_1, expr_2, \dots, expr_k$ — 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 $cond_1, \dots, cond_n$ отличается от *true* или *false*. Если *prederror = false* и значения какого-либо из выражений $cond_1, \dots, cond_n$ отличается от *true* или *false*, результат вычисления *if* — условное выражение.

4.1.2 Операторы цикла

Для выполнения итераций используется оператор **text tt do**. Могут использоваться три варианта его вызова, отличающиеся условием окончания цикла:

```
for variable : init,aluestepincrement thru limit do body
for variable : init,aluestepincrement while condition do body
for variable : init,aluestepincrement unless condition do body
```

Здесь *variable* — переменная цикла; *init,alue* — 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*.

Параметры *init,alue, 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 изменение 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 = -3
a = 4
a = 11
a = 18
a = 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 = 2
count = 6
count = 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 оператора цикла характеризуется **выбором значений переменной цикла из заданного списка**. Синтаксис вызова:

```
for variable in list end_ests do body
```

Проверка условия завершения *end_ests* 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]
```

4.1.3 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, расположенную в этом же блоке:

```
(%i1) block([a],a:1,metka, a:a+1,
            if a=1001 then return(-a),go(metka));
(%o1) - 1001
```

В этом блоке реализован цикл, который завершается по достижении "переменной цикла" значения 1001. Меткой может быть произвольный идентификатор.

Следует иметь в виду, что цикл сам по себе является блоком, так что (в отличие 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 внутри блока, содержит оператор **return**, то при

исполнении оператора **return** 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);
x = 555
(%o1)          777
(%i2) block([],x:for i:1 thru 15 do
      if i=52 then return(555),display(x),777);
x = done
(%o2)          777
```

Если необходимо выйти из нескольких вложенных блоков сразу (или нескольких блоков и циклов сразу) и при этом вернуть 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);
expt: undefined: 0 to a negative exponent.
(%o1)          []
(%i2) c;
(%o2)          c
```

Выполнение последовательности операций прерывается на первой операции, приводящей к ошибке. Остальные выражения блока не выполняются

(значение s остаётся неопределённым). Сообщение об возникшей ошибке может быть выведено функцией `errmsg()`.

4.1.4 Функции

Наряду с простейшим способом задания функции, **Maxima** допускает создание функции в виде последовательности операторов:

$$f(x) := (expr_1, expr_2, \dots, expr_n);$$

Значение, возвращаемое функцией — значение последнего выражения $expr_n$.

Чтобы использовать оператор **return** и изменять возвращаемое значение в зависимости от логики работы функции, следует применять конструкцию **block**, например:

$$f(x) = \text{block} ([], expr_1, \dots, \text{if} (a > 10) \text{ then } \text{return}(a), \dots, expr_n).$$

При $a > 10$ выполняется оператор **return** и функция возвращает значение a , в противном случае — значение выражения $expr_n$.

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 блока.

Пример:

$$\text{block} ([a : a], expr_1, \dots, a : a+3, \dots, expr_n)$$

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)
```

$$f(x) := ([a : a], \text{if } a > 0 \text{ then } 1 \text{ else if } a < 0 \text{ then } -1 \text{ else } 0)$$

```
(%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:

1. Задание функции $f(x) := (expr_1, \dots, expr_n)$; вызов функции $f(1)$; — начальное значение локальной переменной x равно 1.
2. Задание блока **block** ($[x: 1], expr_1, \dots, expr_n$), 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([x1, ..., xm], expr1, ..., exprn)
f2 : lambda([[L]], expr1, ..., exprn)
f3 : lambda([x1, ..., xm, [L]], expr1, ..., exprn)
```

Пример:

```
(%i1) f : lambda ([x], x^2);
(%o1) lambda ([x], x^2)
(%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 g^2 в примере).

```
(%i6) a : %pi$ b : %e$ g : lambda ([a], a*b);
(%o8) lambda ([a], a b)
(%i9) b : %gamma$ g (1/2);
(%o10)  $\frac{\gamma}{2}$ 
(%i11) g2 : lambda ([a], a*"b");
(%o11) lambda ([a], a  $\gamma$ )
(%i12) b : %e$ 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 ([a, b], h2: lambda ([a], a*b), h2( $\frac{1}{2}$ ))
```

```
(%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 ([aa, bb, [cc]], aa*cc + bb)
```

```
(%i2) f(3, 2, a, b, c);
```

```
(%o2) [3a + 2, 3b + 2, 3c + 2]
```

Список $[cc]$ при вызове лямбда-функции f включает три 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, \dots, v_n объявляются локальными вызовом $local(v_1, v_2, \dots, v_n)$ независимо от контекста.

4.1.5 Транслятор и компилятор в Maxima

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 **Maxima**.

4.1.5.1 Функция translate

Функция *translate* транслирует функцию **Maxima** на язык **Lisp**. Например, выражение: $f(x) := 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$ транслируется командой: `translate(f)`. После этого функция, как правило, начинает вычисляться быстрее.

Пример, иллюстрирующий выигрыш по времени после трансляции функции:

```
(%i1) f(n) := block([sum, k], sum:0,
for k:1 thru n do (sum:sum+k^2), sum)$
```

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

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

Для выполнения тестов использовался один и тот же компьютер и Maxima 5.24. При непосредственном обращении к функции f время вычисления $f(1000000)$ составило 7,86 с, после трансляции — 3,19 с. Для оценки времени вычисления использована функция *time*.

```
(%i2) f(1000000);
(%o2) 333333833333500000
(%i3) time(%o2);
(%o3) [7.86]
(%i4) translate(f);
(%o4) [f]
(%i5) f(1000000);
(%o5) 333333833333500000
(%i6) time(%o5);
(%o6) [3.19]
```

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

4.1.5.2 Функция *compile*

Функция *compile* сначала транслирует функцию Maxima на язык Lisp, а затем компилирует эту функцию Lisp 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. Например, после компиляции функции f из последнего примера время вычисления $f(1000000)$ составило 2.17 с.

Следует иметь в виду, что как при трансляции, так и при компиляции Maxima старается оптимизировать функцию по скорости. Однако Maxima работает преимущественно с целыми числами произвольной длины либо

текстовыми выражениями. Поэтому при работе с большими по объёму функциями могут возникнуть проблемы, связанные с преобразованием типов данных. В этом случае следует отказаться от трансляции или компиляции, либо переписать функцию, упорядочив использование типов.

Пример: Рассмотрим две функции, вычисляющие одно и то же выражение.

В функции $f2$ явно указано, что функция возвращает действительные значения (в формате с плавающей точкой)

```
f1(x,n):=block([sum,k], sum:1,
  for k:1 thru n do (sum:sum+1/x^k), sum)$
```

```
f2(x,n):=block([sum,k],
  mode_declare([function(f2),x], float),
  sum:1, for k:1 thru n do (sum:sum+1/x^k), sum)$
```

Время выполнения функции $f1$ при запуске $f1(5, 10000)$ составило 1,8 с. После компиляции время выполнения составило 1,49 с, после трансляции — 1,39 с. Попытка обратиться к откомпилированной функции $f1$ командой $f1(5.0, 10000.0)$ завершилась неудачей вследствие возникающей ошибки (плавающее переполнение).

При использовании функции с декларированным типом результата ($f2$) время выполнения $f2(5, 10000)$ оказалось меньше, чем $f1(1,65$ с вместо 1,8 с). Однако время выполнения той же функции после трансляции или компиляции превышает 10 с. Следует учесть, что в данном случае результат расчёта — рациональное число. Преобразование его к форме с плавающей точкой при вычислении очередного значения суммы требует дополнительных вычислительных затрат. При обращении к $f2$ с действительными аргументами $f2(5.0, 10000.0)$ время счёта составило всего 0,16 с.

Для функции, возвращающей результат, который представляется в виде числа с плавающей точкой, компиляция или трансляция может дать уменьшение времени счёта в несколько раз.

Пример: Рассмотрим функции, вычисляющую действительное выражение (в данном случае суммируются иррациональные числа)

```
f3(x,n):=block([sum,k],
  mode_declare([function(f3),x], float),
  sum:1, for k:1 thru n do (sum:sum+sqrt(x^k)), sum)$
```

Время вычисления выражения $f3(5, 2000)$ для неоткомпилированной и не оттранслированной функции составило 7,47 с., после трансляции время вычисления $f3(5, 2000)$ составило 0,03 с, после компиляции — 0,02 с.

Рассмотрим ещё один пример:

```
f4(x,n):=block([sum,k], sum:1,
  for k:1 thru n do (sum:sum+k/x), sum)$
```

Время вычисления выражения $f4(5, 1000000)$ составило 10,89 с, время вычисления выражения $f4(5.0, 1000000)$ составило 6,71 с. После трансляции

f_4 время вычисления выражения $f_4(5, 1000000)$ составило 9,1 с (выигрыш по времени практически отсутствует), а для $f_4(5.0, 1000000)$ — 2,49 с (выигрыш по времени за счёт выполнения вычислений с плавающей точкой примерно в 2,5 раза).

4.2 Ввод-вывод в пакете *Maxima*

В этом разделе рассматриваются конструкции, позволяющие осуществить обмен данными между *Maxima* и другими приложениями.

4.2.1 Ввод-вывод данных в консоли

Основная функция для считывания вводимых пользователем данных: $read(expr_1, \dots, expr_n)$. Вводимые выражения $expr_1, expr_2, \dots$ 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")
;
Значение a = 42 введите новую величину (p+q)^3;
(%o2) (q + p)^3
(%i3) display(a);
a = (q + p)^3
(%o3) 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:");
```

Введите выражение: 2^a ;

```
(%o2) 2^a
(%i3) read("Введите выражение:");
```

Введите выражение: 2^a ;

```
(%o3) 128
```

Вывод на экран осуществляется функцией $display$. Синтаксис её вызова: $display(expr_1, expr_2, \dots)$.

Выражения из списка аргументов выводятся слева направо (сначала само выражение, а затем после знака равенства — его значение).

Аналогичная 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);
a = 1
b = 2
c = 3
(%o4) done
(%i5) disp(a,b,c);
1
2
3
(%o5) done
(%i6) grind(a);
1
(%o6) done
```

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 *dispterm*. Синтаксис вызова:

dispterm(expr).

Кроме того, для вывода результатов вычислений используется функция *print*. Синтаксис вызова:

print(expr₁, ..., expr_n).

Выражения *expr₁, ..., 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);
```

Пример: 1 2 5

```
(%o4) 5
(%i5) rez;
(%o5) 5
```

```
(%i6) display("Пример:", a, b, c);
```

Пример:=Пример:

$a = 1$

$b = 2$

$c = 5$

(%o6) done

4.2.2 Файловые операции ввода-вывода

4.2.2.1 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*¹:

save(filename, name₁, name₂, name₃, ...)

— сохраняет текущие значения переменных *name₁, name₂, name₃, ...* 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, name₁ = expr₁, ...)

— позволяет сохранить объекты **Maxima** с заменой имени *expr₁* на имя *name₁*.

save(filename, all)

— сохраняет все объекты, имеющиеся в памяти.

Глобальный флаг *file_output_append* управляет режимом записи. Если *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, сохранённые функцией *save*, могут быть снова загружены функцией *load* (см. ниже).

Варианты записи при помощи *save* могут совмещаться друг с другом (пример: *save(filename, aa, bb, cc = 42, functions, [11,17])*).

Загрузка предварительно сохранённого функцией *save* файла осуществляется 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, expr1, expr2, expr3, ... )
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. 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* содержат код на **Lisp**, а созданные при помощи функции *stringout* содержат код **Maxima**. Все эти файлы могут с равным успехом 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 записываться при помощи функции *write file* (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 *close file*.

Синтаксис вызова:
write file(filename), *close file(filename)*.

4.2.2.2 Ввод-вывод командных файлов

Основная функция, предназначенная для 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, символы конца строки в файле игнорируются. 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 выполнения 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.

4.3 Встроенные численные методы

4.3.1 Численные методы решения уравнений

4.3.1.1 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 $a \leq x \leq b$.

Для поиска корней используется метод деления пополам или, если исследуемая функция достаточно гладкая, метод линейной интерполяции.

4.3.2 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/5.26.0/share/numeric/newton1.mac
(%i2) newton (cos (u), u, 1, 1/100);
(%o2) 1.570675277161251
(%i3) ev (cos (u), u = %);
(%o3) 1.2104963335033529 10-4
(%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 = %);
(%o6) 6.098490481853958 10-4 a2
```

4.3.2.1 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 умолчанию $10.0^{-\frac{fpprec}{2}}$), $newtonmaxiter$ (максимальное число итераций, величина по умолчанию 50).

Синтаксис вызова: $mnewton(FuncList, VarList, GuessList)$, где $FuncList$ — 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 = 2.779849592817897]]
(%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.

4.3.3 Интерполяция

Для выполнения интерполяции функций, заданных таблично, в составе **Maxima** предусмотрен пакет расширения `interp`, 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.

4.3.3.1 Линейная интерполяция

Линейная интерполяция выполняется функцией `linearinterp` или `linearinterp(points)`. Синтаксис вызова: `linearinterp(points, option)`.

Аргумент `points` должен быть представлен в одной из следующих форм:

- матрица с двумя столбцами, например `r:matrix([2,4], [5,6], [9,3])`, при этом первое значение пары или первый столбец матрицы — это значения независимой переменной,
- список пар значений, например `r:[[2,4], [5,6], [9,3]]`,
- список чисел, которые рассматриваются как ординаты интерполируемой функции, например `r:[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("interp") $
(%i2) r: matrix([7,2], [8,2], [1,5], [3,2], [6,7]) $
(%i3) linearinterp(r);
(%o3) (13/2 - 3x/2) charfun2(x, -∞, 3) + 2 charfun2(x, 7, ∞) +
(37 - 5x) charfun2(x, 6, 7) + (5x/3 - 3) charfun2(x, 3, 6)
```

```
(%i4) f(x) := "%";
(%o4) f(x) := (13 - 3x) charfun2(x, -∞, 3) + 2 charfun2(x, 7, ∞) + (37 - 5x) charfun2(x, 6, 7) + (5x/3 - 3) charfun2(x, 3, 6)
(%i5) map(f, [7.3, 25/7, %pi]);
(%o5) [2, 62/21, 5π/3 - 3]
```

4.3.3.2 Интерполяция полиномами Лагранжа

Интерполяция полиномами Лагранжа выполняется при помощи функции *lagrange*.

Синтаксис вызова: *lagrange(points)* или *lagrange(points, option)*.

Смысл параметров *points* и *options* аналогичен указанному выше.

Пример использования интерполяции полиномами Лагранжа:

```
(%i1) load("interpol") $
(%i2) p: [[7, 2], [8, 2], [1, 5], [3, 2], [6, 7]] $
(%i3) lagrange(p);
(%o3) (x-7)(x-6)(x-3)(x-1) / (7(x-8)(x-7)(x-3)(x-1)) - (x-8)(x-6)(x-3)(x-1) / (30(x-8)(x-7)(x-6)(x-1)) +
(x-8)(x-6)(x-3)(x-1) / (60(x-8)(x-7)(x-6)(x-1)) + (x-8)(x-7)(x-6)(x-3) / (84(x-8)(x-7)(x-6)(x-3))
(%i4) f(x) := "%";
(%o4) f(x) := (x-7)(x-6)(x-3)(x-1) / (7(x-8)(x-7)(x-3)(x-1)) - (x-8)(x-6)(x-3)(x-1) / (30(x-8)(x-7)(x-6)(x-1)) +
(x-8)(x-6)(x-3)(x-1) / (60(x-8)(x-7)(x-6)(x-1)) + (x-8)(x-7)(x-6)(x-3) / (84(x-8)(x-7)(x-6)(x-3))
(%i5) map(f, [2.3, 5/7, %pi]);
(%o5) [-1.567535, 10.9366573451538, 2.893196551256924]
(%i6) %, numer;
(%o6) [-1.567535, 10.9366573451538, 2.893196551256924]
```

4.3.3.3 Интерполяция сплайнами

Интерполяция кубическими сплайнами выполняется при помощи функции *cspline*.

Синтаксис вызова: *cspline(points)* или *cspline(points, option)*.

Смысл параметров *points* и *options* аналогичен указанному выше.

Пример использования интерполяции кубическими сплайнами:

```
(%i1) load("interpol") $
(%i2) p: [[7, 2], [8, 2], [1, 5], [3, 2], [6, 7]] $
(%i3) cspline(p);
```

```

(%o3)  $\left(\frac{1159x^3}{3288} - \frac{1159x^2}{1096} - \frac{6091x}{3288} + \frac{8283}{1096}\right) \text{char fun2}(x, -\infty, 3) +$ 
 $\left(-\frac{2587x^3}{1644} + \frac{5174x^2}{137} - \frac{494117x}{1644} + \frac{108928}{137}\right) \text{char fun2}(x, 7, \infty) +$ 
 $\left(\frac{4715x^3}{1644} - \frac{15209x^2}{274} + \frac{579277x}{1644} - \frac{199575}{274}\right) \text{char fun2}(x, 6, 7) +$ 
 $\left(-\frac{3287x^3}{4932} + \frac{2223x^2}{274} - \frac{48275x}{1644} + \frac{9609}{274}\right) \text{char fun2}(x, 3, 6)$ 
(%i4) f(x) := "%";
%o4)  $f(x) := \left(\frac{1159x^3}{3288} - \frac{1159x^2}{1096} - \frac{6091x}{3288} + \frac{8283}{1096}\right) \text{char fun2}(x, -\infty, 3) +$ 
 $\left(-\frac{2587x^3}{1644} + \frac{5174x^2}{137} - \frac{494117x}{1644} + \frac{108928}{137}\right) \text{char fun2}(x, 7, \infty) +$ 
 $\left(\frac{4715x^3}{1644} - \frac{15209x^2}{274} + \frac{579277x}{1644} - \frac{199575}{274}\right) \text{char fun2}(x, 6, 7) +$ 
 $\left(-\frac{3287x^3}{4932} + \frac{2223x^2}{274} - \frac{48275x}{1644} + \frac{9609}{274}\right) \text{char fun2}(x, 3, 6)$ 
(%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]

```

4.3.4 Оптимизация с использованием пакета lbfgs

Основная функция пакета ($lbfgs(FOM, X, X0, \epsilon, iprint)$) 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.

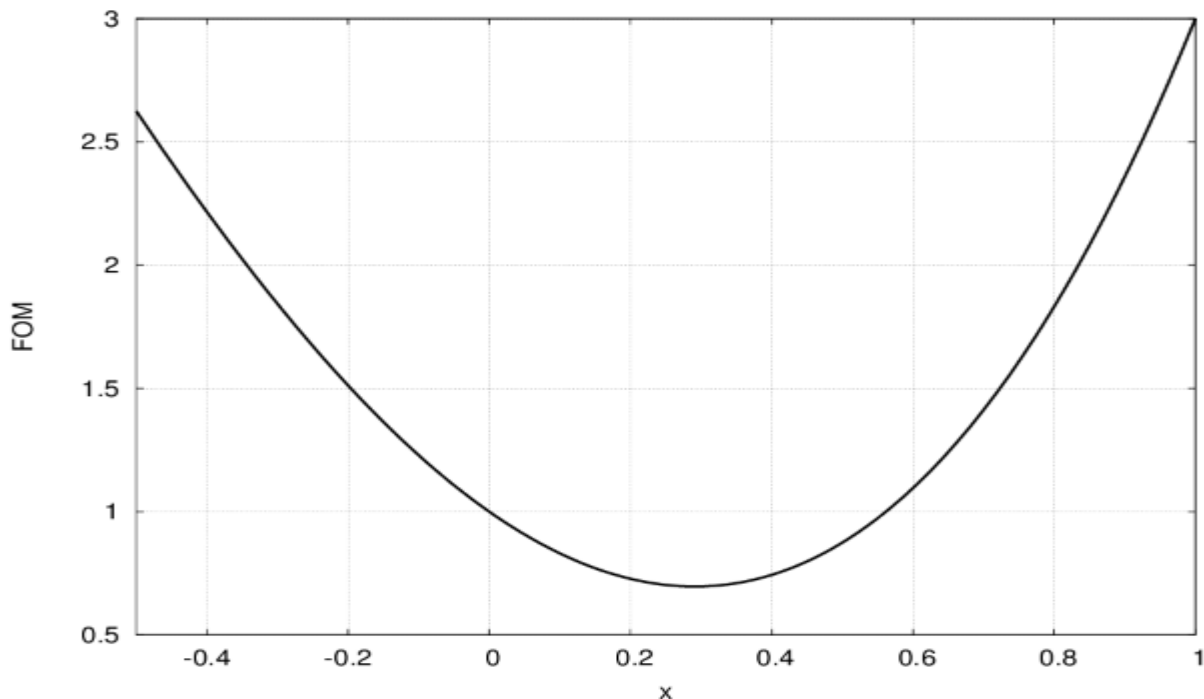


Рис. 4.1. График исследуемой функции в окрестности минимума

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)
      x3 + 3x2 - 2x + 1
(%i3) lbfgs(FOM, [x], [1.1], 1e-4, [1, 0]);

*****
N=      1      NUMBER OF CORRECTIONS=25
      INITIAL VALUES
F=  3.7610000000000001D+00      GNORM=  8.2300000000000001D+00
*****
I  NFN      FUNC      GNORM
STEPLength
1   2      8.309999999999997D-01      1.3700000000000000D+00
1.215066828675577D-01
2   3      7.056026396574796D-01      3.670279947916664D-01
1.0000000000000000D+00
3   4      6.967452517789576D-01      3.053950958095847D-02
1.0000000000000000D+00
4   5      6.966851926112383D-01      5.802032710369720D-04
1.0000000000000000D+00
5   6      6.966851708806983D-01      8.833119583551152D-07
1.0000000000000000D+00
THE MINIMIZATION TERMINATED WITHOUT DETECTING ERRORS.
IFLAG = 0
(%o3)      [x = 0.29099433470072]
```

Рассмотрим результаты минимизации функции нескольких переменных при помощи *lbfgs*:

```
(%i1) load (lbfgs)$
(%i2) FOM:2*x*y+8*y*z+12*x*z+1e6/(x*y*z);
(%o2)
      8yz + 12xz +  $\frac{1000000.0}{xyz}$  + 2xy
(%i3) lbfgs(FOM, [x, y, z], [1, 1, 1], 1e-4, [-1, 0]);
%o3) [x = 13.47613086835734, y = 20.21398622934409,
z = 3.369022781547174]
```

4.3.4.1 Оптимизация с ограничениями методом неопределённых множителей Лагранжа

Для решения задач минимизации с ограничениями в составе **Maxima** предусмотрен пакет `augmented_lagrangian_method`, реализующий метод неопределённых множителей Лагранжа.

Синтаксис вызова функции:

- `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, представленными в виде равенств. 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 множителей 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.

4.3.5 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(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 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 $rombertol$. Функция $romberg$ заканчивается успешно, когда абсолютное различие между последовательными приближениями — меньше чем $rombergabs$, или относительное различие в последовательных приближениях — меньше чем $rombertol$. 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 функции составляет $2^{rombergit}$. Если критерий точности интегрирования, установленный $rombergabs$ и $rombertol$, 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 подинтегральное 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)
      x y
g(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) 
$$-9 \log\left(\frac{9}{2}\right) + 9 \log(3) + \frac{2 \log\left(\frac{3}{2}\right) - 1}{6} + \frac{9}{2}$$

```

```
(%i6) float(%);
```

```
(%o6) 0.81930239639591
```

Как видно из полученных результатов вычисления двойного интеграла, точное и приближённое решение совпадают до 7 знака включительно.

5. Оформление Maxima

5.1 Классические графические интерфейсы Maxima

5.1.1 Графический интерфейс wxMaxima

Для удобства работы сразу обратимся к графическому интерфейсу **wxMaxima**, т. к. он является наиболее дружелюбным 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**)

5.1.1.1 Рабочее окно wxMaxima

Рассмотрим рабочее окно программы (см. рис. 5.1 и рис. 5.2). 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, разделённые по группам: уравнения, алгебра, анализ, упростить, графики, численные вычисления. Ввод команд через диалоговые окна упрощает работу с программой для новичков.

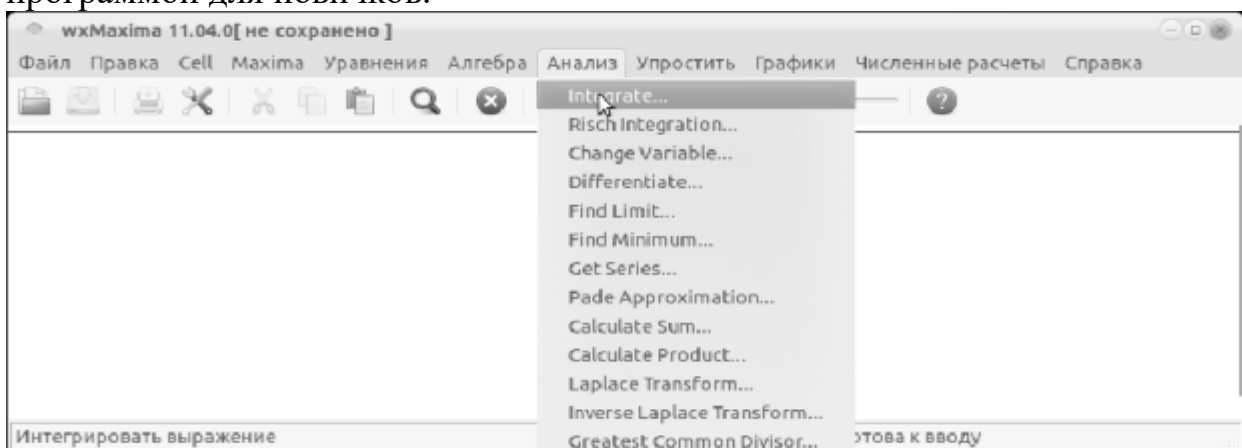


Рис. 5.1. Интерфейс wxMaxima, выбор команды интегрирования.

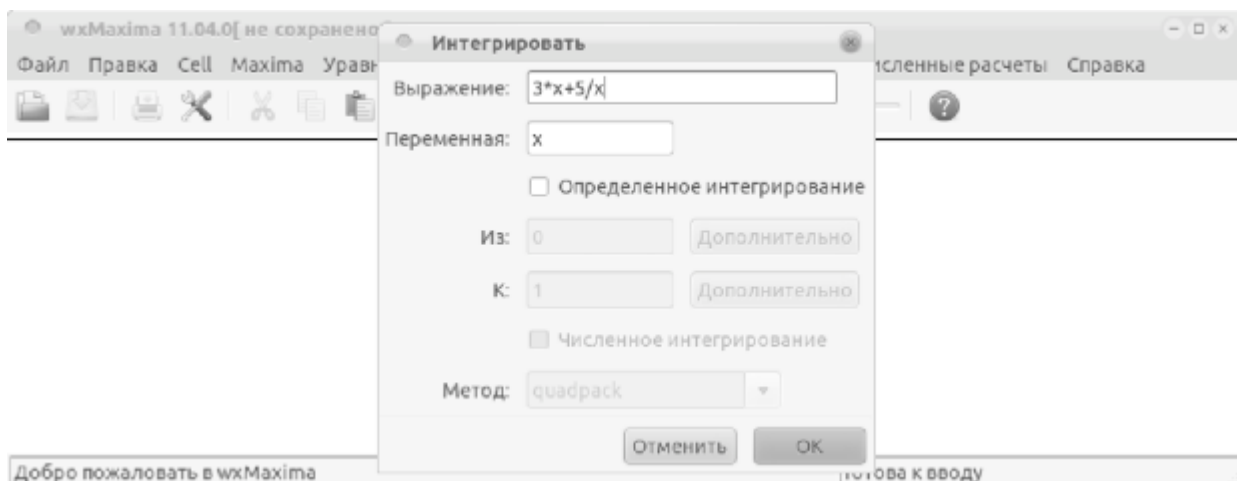


Рис. 5.2. Интерфейс wxMaxima, вычисление интеграла.

Например, пункт меню Анализ/Интегрировать позволяет вычислить определённый или неопределённый интеграл. После ввода необходимых параметров, в рабочем окне мы увидим команду и результат вычисления:

```
(%i1) integrate(3*x+5/x, x);
```

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

Пример использования команд меню для вычисления предела

$$\lim_{x \rightarrow 0} \frac{\sin(2x)}{x}$$

представлен на рис. 5.3 и рис. 5.4. Следует отметить, что оболочка **wxMaxima** при вызове команды и соответствующего диалогового окна генерирует текстовую команду, интерпретируемую вычислительным ядром **Maxima**. Передаваемая ядру **Maxima** строка выводится в командное окно аналогично команде, введённой вручную. После генерации и первого выполнения команды (или набора команд) можно дополнять и редактировать автогенерированную команду, рассматривая её в качестве шаблона.

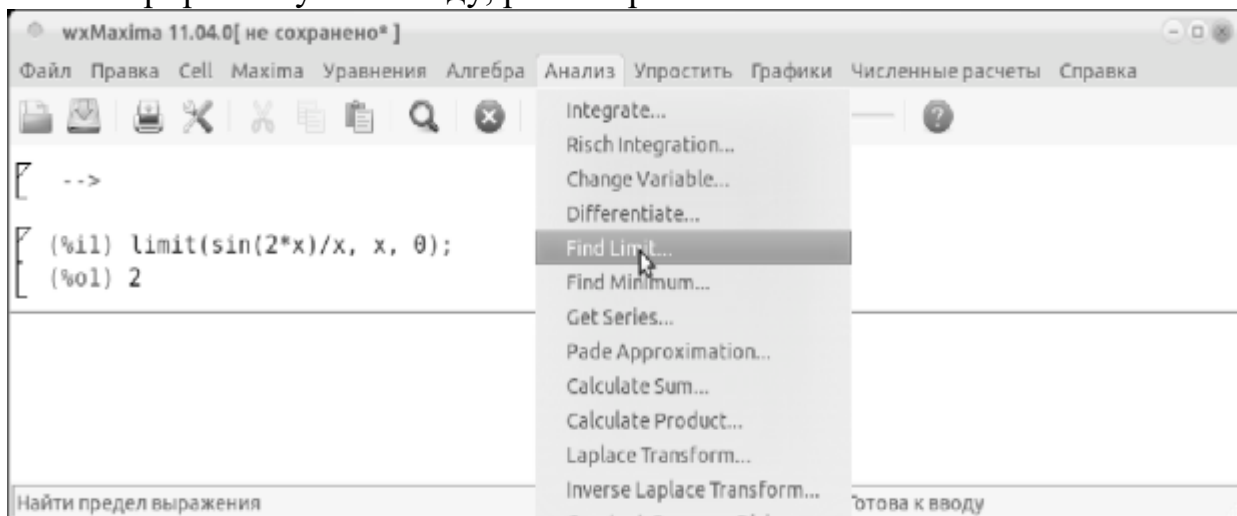


Рис. 5.3. Интерфейс wxMaxima, выбор команды find limit.

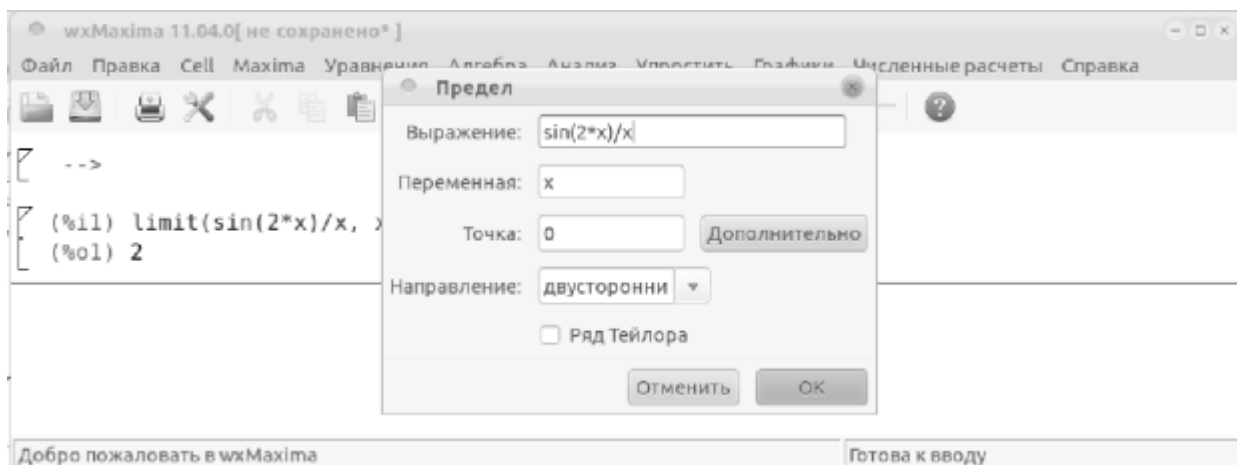


Рис. 5.4. Интерфейс wxMaxima, окно ввода — вычисление предела.

Ниже располагается графическое меню основных команд с пиктограммами, соответствующими наиболее часто используемым функциям для работы с файлами: открыть / сохранить / печать данных, а также функциям правки — копировать / удалить / вставить текст и другие.

Центральную часть рабочего окна **wxMaxima** занимает командное окно (псевдотерминал), в которое вводятся команды системы и выводятся результаты.

В последних версиях интерфейсного пакета **wxMaxima** поддерживается концепция ячеек (cells) в рабочей книге. Ячейка включает либо набор команд **Maxima**, либо результаты их выполнения (в т. ч. графики). Кроме того, по аналогии с Maple и Mathematica **wxMaxima** поддерживает текстовые ячейки (text cells) для пояснений и комментариев, а также ячейки для заголовков и номеров секций (title cells, section cells, subsection cells). Пример книги Maxima с ячейками указанных типов представлен на рис. 5.5. Допускается вставка изображений в рабочую книгу (также в специальные ячейки).

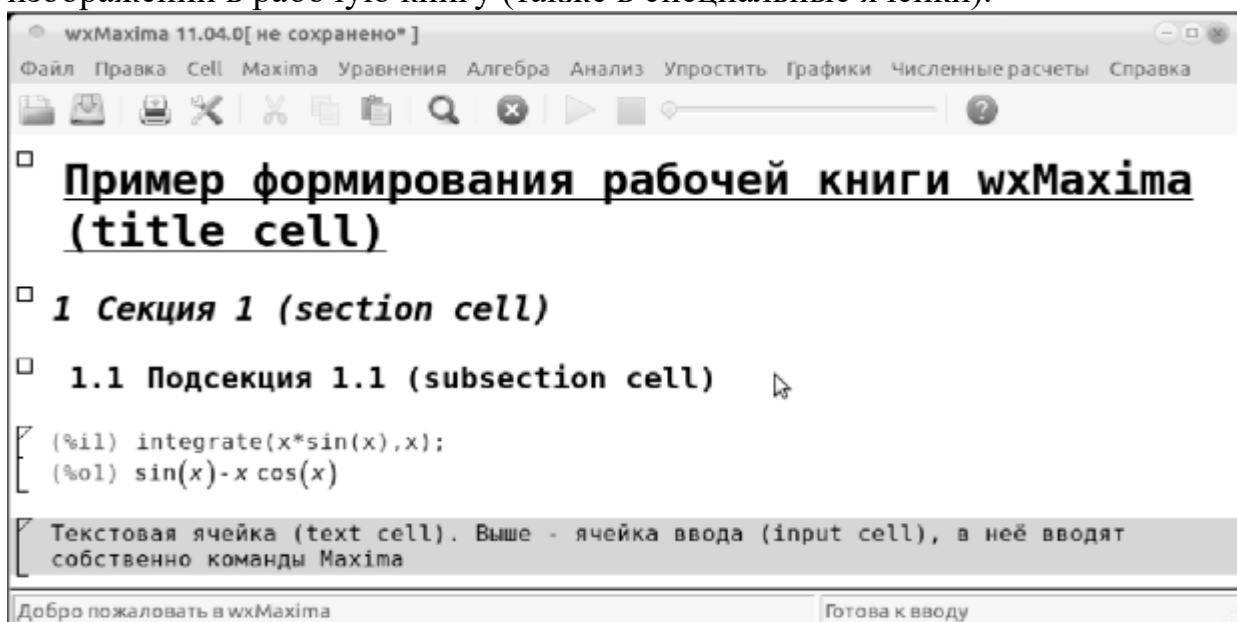


Рис. 5.5. Пример вставки ячеек различных типов в книгу wxMaxima.

При сохранении книги (в формате wxm) в файл выводятся только входные ячейки (input). Поэтому при работе с сохранённым документом не обязательно интерпретировать все ячейки, хотя это возможно — команда Evaluate all cells из меню Cells).

Рабочую книгу **Maxima** можно экспортировать в форматы html или pdflatex.

Интерпретация текущей ячейки, в которой может быть несколько команд, осуществляется после нажатия комбинации клавиш Ctrl+Enter, либо командой меню Cells. Если необходимо предотвратить вывод отклика команды, следует явно завершить её символом \$. Современные версии **wxMaxima** автоматически завершают ввод, если это необходимо, символом ";".

При использовании интерфейса **wxMaxima** можно выделить в командном окне 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 контекстных меню при работе с **wxMaxima** смотри на рис. 5.6, рис. 5.7 и рис. 5.8.

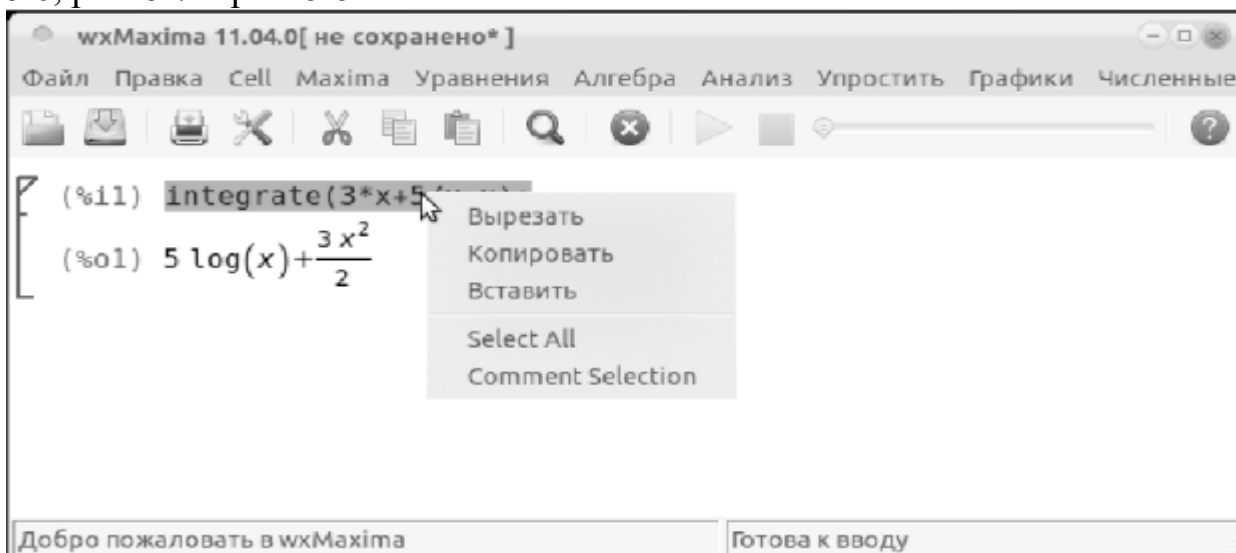


Рис. 5.6. Интерфейс wxMaxima. Контекстное меню строки ввода.

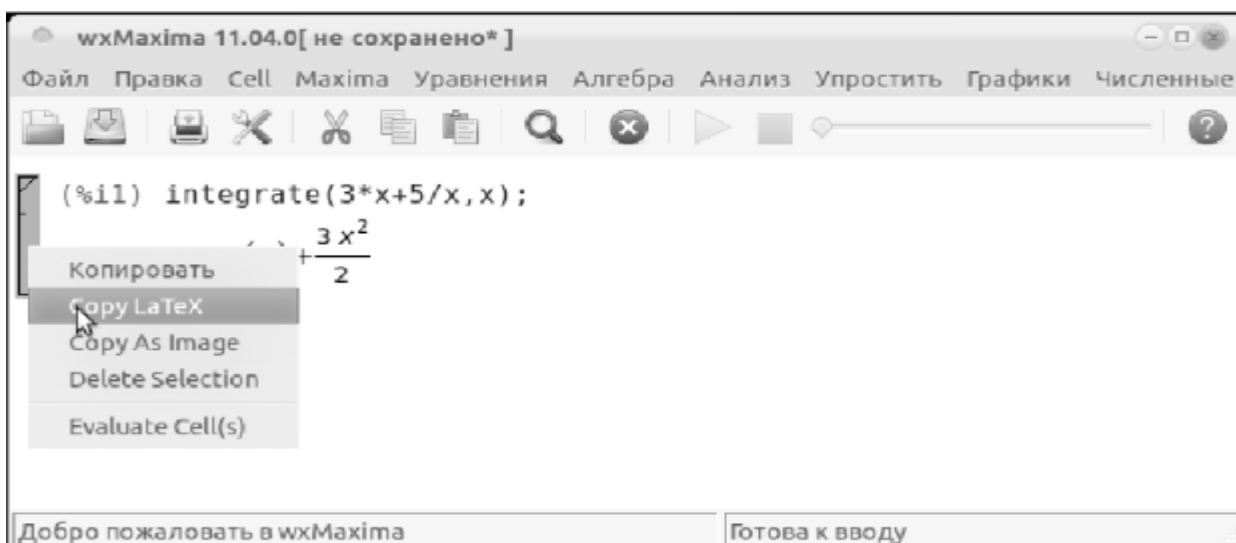


Рис. 5.7. Интерфейс wxMaxima. Контекстное меню ячейки.

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

По умолчанию **wxMaxima** предполагает, что команда, вводимая при помощи кнопки, применяется к последнему выводу (т. е. аргумент команды — %). Все кнопки или пункты меню в верхней или нижней части рабочего окна соответствуют той или иной команде **Maxima**.

Кроме того, **wxMaxima** предоставляет удобный интерфейс к документации по системе **Maxima**.

Меню правка \rightarrow настройки обеспечивает достаточно широкие возможности настройки графического интерфейса **wxMaxima**. Предусмотрены три группы параметров:

- опции, определяющие отдельные особенности выполнения команд;
- опции вызова вычислительного ядра **Maxima**;
- опции, определяющие стиль графического интерфейса (язык, шрифты, цветовую гамму и т. п.).

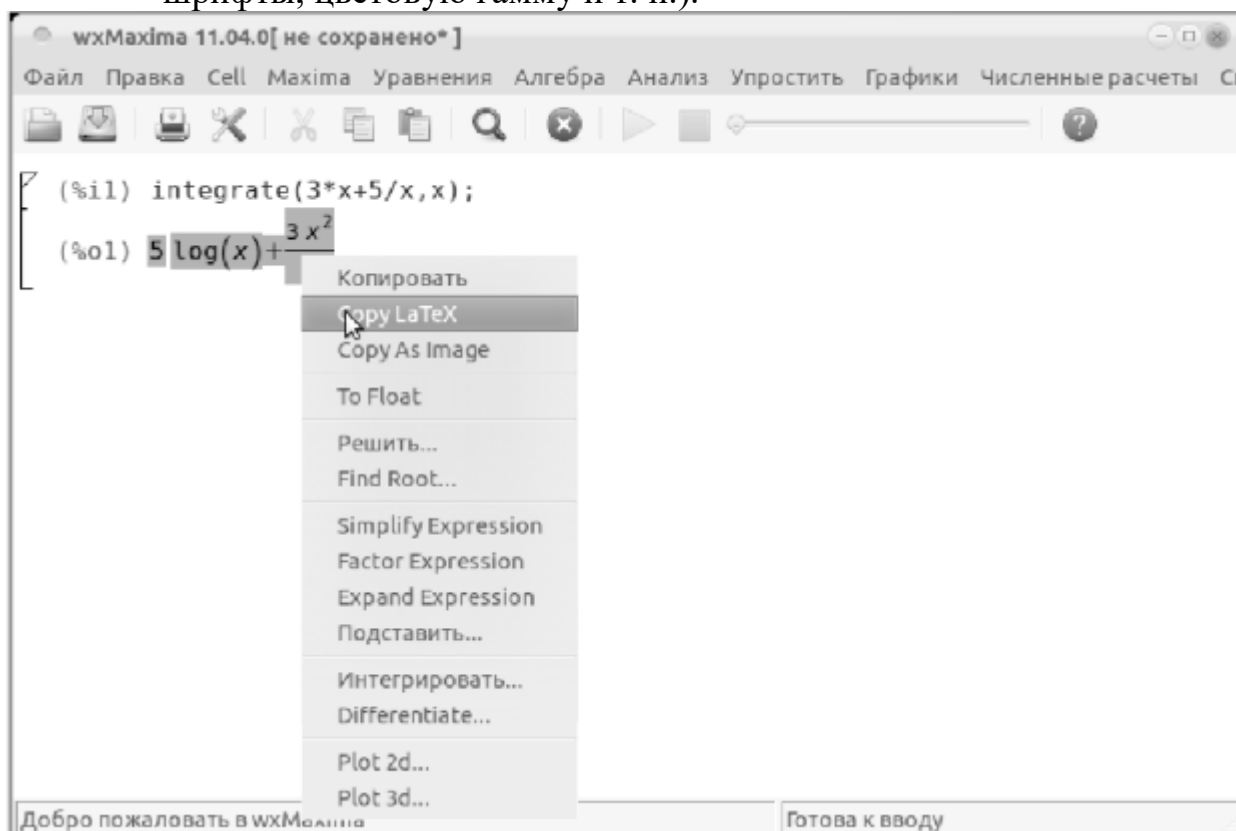


Рис. 5.8. Интерфейс **wxMaxima**. Контекстное меню строки вывода.

Управление процессом вычислений осуществляется командами пункта главного меню **Maxima**. Пользователю предоставляются следующие возможности:

- прервать вычисления, перезапустить **Maxima**, очистить память;
- просмотреть содержимое памяти (переменные, функции, определения и т. п.);
- изменить формат просмотра результатов.

5.1.2 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 отличается простотой (точнее, минимализмом). В последних версиях **xMaxima** при старте открываются одновременно окно браузера системы помощи и консоль команд.

Предполагается, что пользователь владеет командами **Maxima** и макроязыком программирования. Общий вид командного окна **xMaxima** представлен на рис. 5.9. Пункты меню File, Edit, Options позволяют управлять сессией **Maxima**, сохранять и запускать batch- файлы. В рабочую книгу **xMaxima** можно встраивать графики в формате openmath (в зависимости от установки опции plot window). Пример рабочего окна **xMaxima** с простыми графиками представлен на рис. 5.10. График в рабочей книге можно вращать, редактировать, охранять в файл. Как и **wxMaxima**, интерфейс **xMaxima** предоставляет доступ к html-файла помощи по пакету **Maxima**.

```

xmaxima: console
Файл Редактировать Настройки Maxima Справка
(%i1) integrate(x*sin(x), x);
(%o1)          sin(x) - x cos(x)
(%i2) integrate(%, x);
(%o2)          - x sin(x) - 2 cos(x)
(%i3) integrate(%, x);
(%o3)          x cos(x) - 3 sin(x)
(%i4) plot2d(%, [x, -%pi, %pi], [plot_format, openmath]);
(%i5)
Maxima запущена
  
```

Рис. 5.9. Общий вид рабочего окна xMaxima

5.1.3 Использование редактора TeXmacs в качестве интерфейса Maxima

Широкие возможности 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 полученный \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

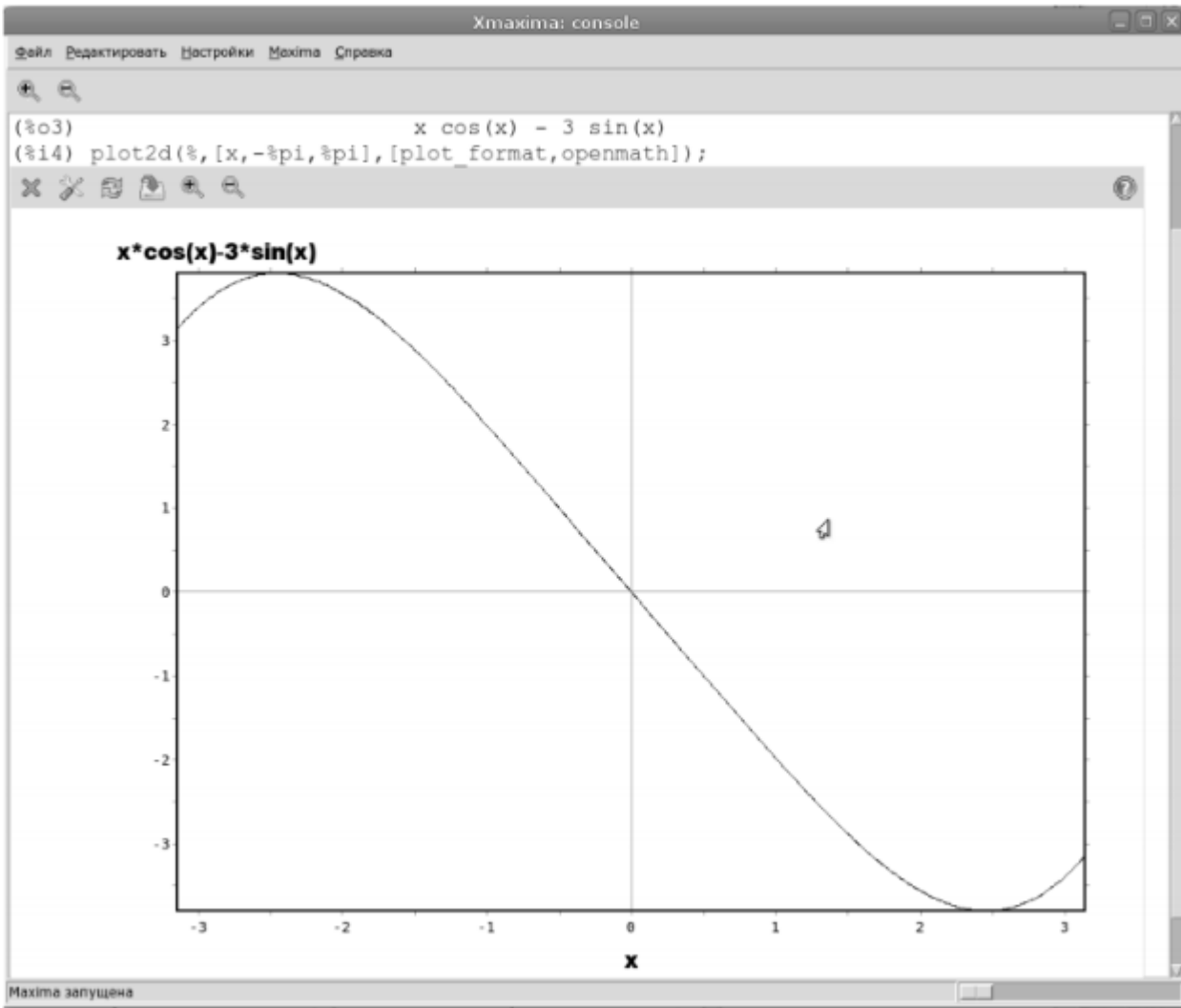


Рис. 5.10. Встроенный график в рабочей книге xMaxima

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. 5.11. Последовательность вставки сессии **Maxima** в текст документа показана на рис. 5.12.

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**.

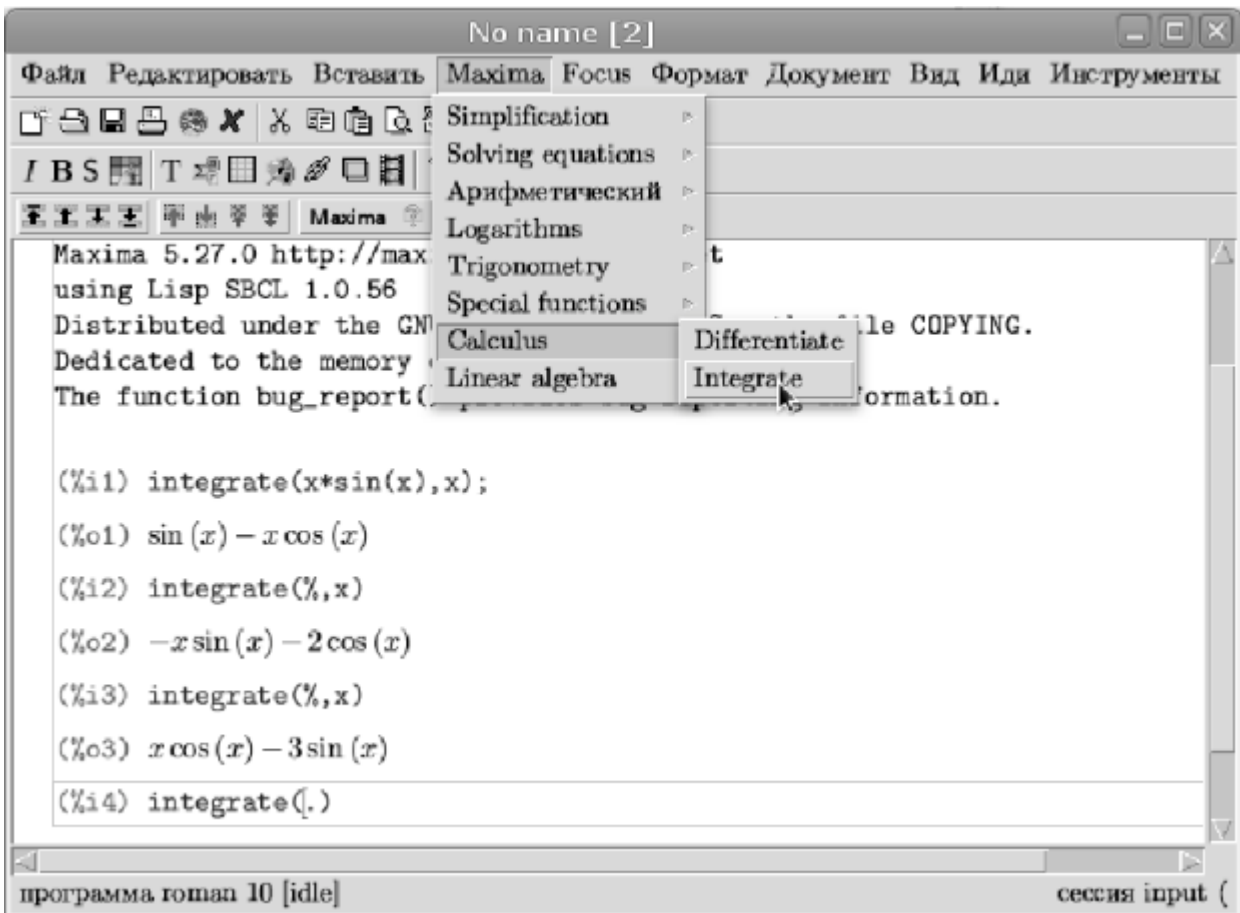


Рис. 5.11. Рабочее окно TeXmacs с запущенной сессией Maxima

Для решения проблем с запуском **Maxima**-сессии из **TeXmacs** возможным решением является редактирование файла `/usr/lib/texmacs/TeXmacs/bin/maxima_detect`, в котором надо ссылку на `#!/bin/sh` заменить ссылкой 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.

5.1.4 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 клавиатурной 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. 5.13. На этом же рисунке видно меню навигации по текущей сессии, позволяющее показывать необходимый участок сессии, сохранять часть результатов в протокол, повторять ввод уже использовавшихся в данной сессии команд и т.п.

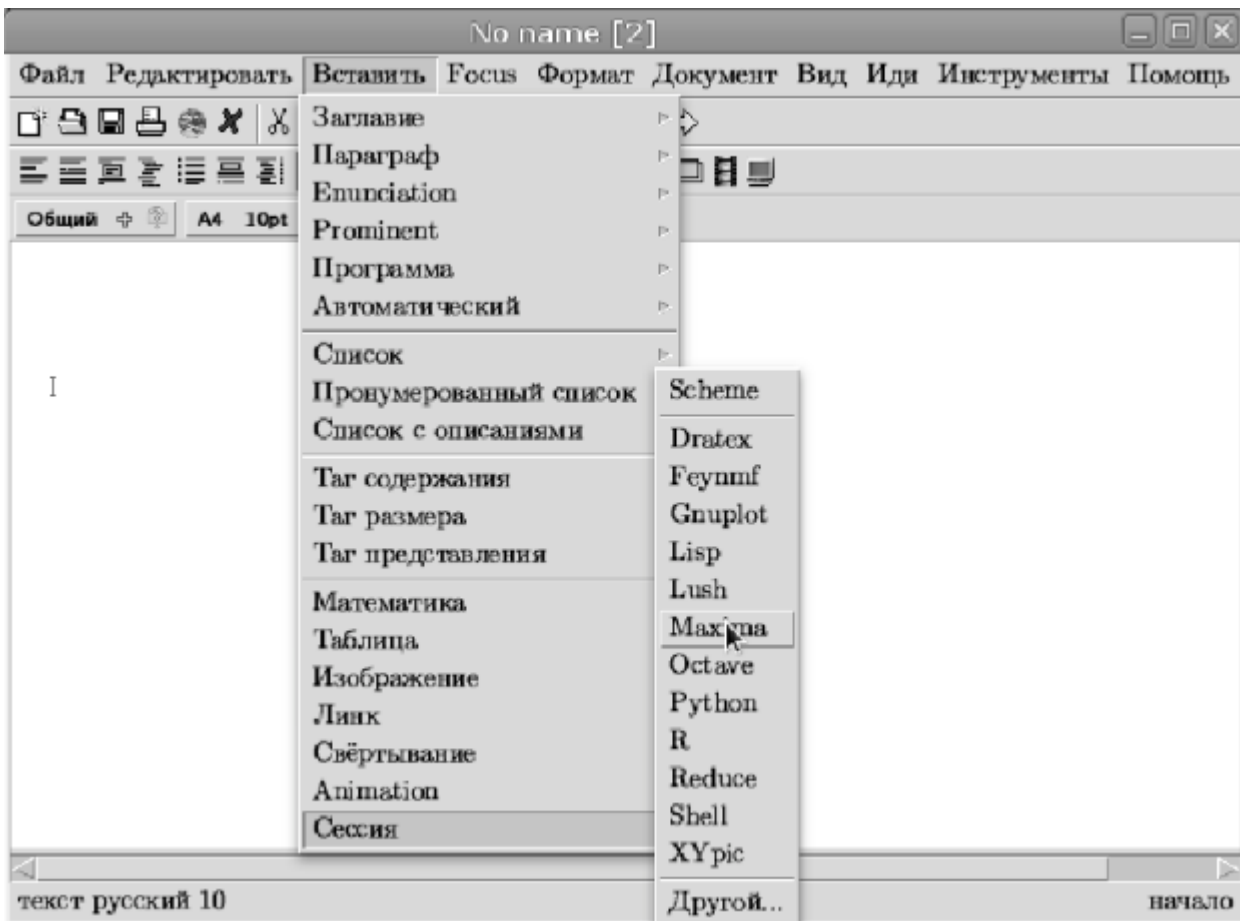


Рис. 5.12. Запуск сессии Maxima в текущем документе TeXmacs

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 Maxima 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 "update cell"; а смежные команды, генерирующие вывод в простой текстовой форме и в форме \LaTeX , всегда привязаны в EMaxima к одинаковым строчной и заглавной буквам соответственно. Пример работы с EMaxima представлен на рис. 5.14, где показаны результаты создание ячейки с

Maxima-кодом и результаты дополнения ячейки (команды можно выбирать из меню EMaxima в верхней части рабочего окна).

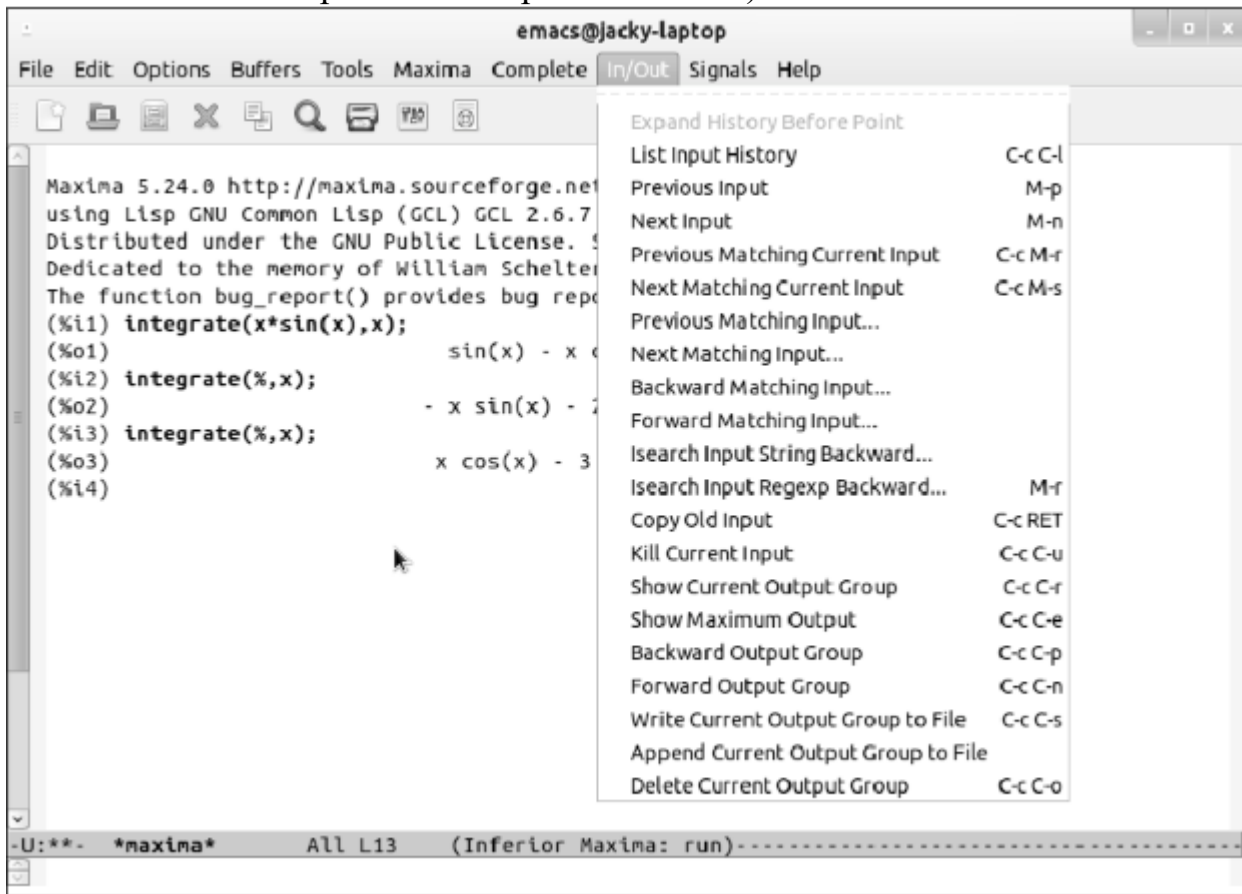


Рис. 5.13. Рабочее окно Emacs с запущенной сессией Maxima

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 отображения \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.

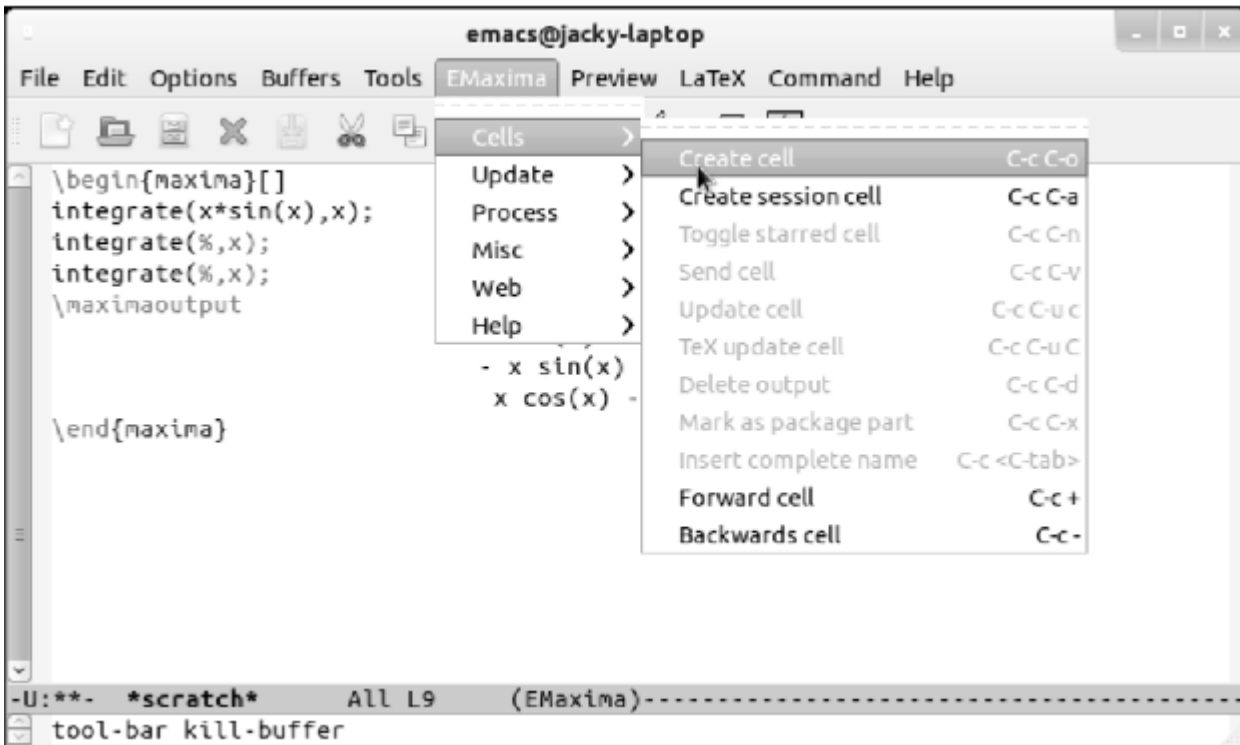


Рис. 5.14. Рабочее окно Emacs с запущенной сессией EMaxima

5.2 Работа с Maxima в KDE: интерфейс Cantor

Интерфейс пользователя **Cantor** состоит из трёх частей:

- Панель вкладок, при помощи которой можно переключаться между документами;
- Панель справки, где будет показано описание команды, если ввести "? команда";
- Панель текущего документа с меню команд, напоминающим интерфейс **wxMaxima**.

Пакет **Cantor** рассчитан на рабочий стол KDE, поэтому графический интерфейс написан с использованием библиотек Qt.

5.2.1 Документ Cantor

В **Cantor** вы работаете с документом. В нём можно вводить выражения, производить вычисления и видеть результаты. Аналогично **wxMaxima** или **Emacs**, документ **Cantor** включает ячейки, содержащие команды текущего пакета и результаты их выполнения. Наряду с командами и результатами, можно вводить и комментарии.

Набор доступных в выражениях команд зависит от используемой системы компьютерной алгебры, поэтому полезно знать синтаксис конкретной системы. Если вы знаете название команды, можно посмотреть её описание, введя "? команда". Чтобы посмотреть примеры документов **Cantor**, выберите пункт меню Файл → Загрузить примеры... и загрузите документы, опубликованные другими пользователями.

5.2.2 Настройка

В меню Настройка можно настроить внешний вид текущего документа. Параметр Показывать результаты с помощью \LaTeX влияет на то, в каком виде будут показаны результаты вычислений. Если он включён, результат будет обработан системой \LaTeX для создания визуально понятных формул.

Например, $3 \cdot x^2 \cdot \sqrt{2} \cdot x + \frac{2}{3}$ превратится в

$$3x^2 \cdot \sqrt{2} \cdot x + \frac{2}{3}$$

Подсветка синтаксиса повышает читаемость кода, выделяя цветом ключевые слова и парные скобки. Если включить параметр Автодополнение, **Cantor** будет показывать возможные продолжения вводимой вами команды при нажатии клавиши Tab. Если существует только одно продолжение команды, при нажатии клавиши Tab название команды будет автоматически введено полностью.

Параметр «Показывать номера строк» позволяет добавить нумерацию введённых выражений. Нумерацию можно использовать для подстановки предыдущих результатов в новое выражение.

5.2.3 Прочие возможности Cantor

Документы **Cantor** сохраняются во встроенном формате cws. Конечный документ пользователя можно сохранять в формате \LaTeX . При работе с **Maxima** в качестве backend графические документы можно встраивать в документ, при помощи контекстного меню они сохраняются в формат eps.

Опцию Встраивать график в файл можно отключить, при этом графики будут воспроизводиться программой gnuplot в отдельном окне. При наличии pdf-псевдопринтера можно распечатать документ **Cantor** в формате pdf (в т. ч. включённые в текст графики и комментарии).

5.3 Интегрированная среда Sage

Sage (англ. "Мудрец") — система компьютерной алгебры покрывающая много областей математики, включая алгебру, комбинаторику, вычислительную математику и матанализ. Первая версия **Sage** была выпущена 24 февраля 2005 года в виде свободного программного обеспечения с лицензией GNU GPL. Первоначальной целью проекта было "создание открытого программного обеспечения альтернативного системам Magma, Maple, Mathematica, и MATLAB". Разработчиком **Sage** является Уильям Стейн — математик Университета Вашингтона (Официальный сайт: <http://sagemath.org>).

Многочисленные возможности **Sage** включают:

- Интерфейс notebook для просмотра и повторного использования введённых команд и полученных результатов, включая графики и текстовые аннотации, доступный из большинства современных веб-браузеров. Доступно защищённое соединение через протокол HTTPS,

когда конфиденциальность имеет значение. Так же **Sage** может выполняться как локально, так и удалённо.

- Интерфейс ввода на основе командной строки, с использованием мультипарадигменного языка IPython.
- Поддержка параллельных вычислений с использованием как многоядерных процессоров, так и многопроцессорных систем и систем распределённых вычислений.
- Внутренняя инфраструктура на python, поддерживающая взаимодействие с математическими пакетами на python: SymPy, SciPy и NumPy.
- Различные статистические библиотеки функций, использующие функциональность R и SciPy.
- Возможность построения плоских и трёхмерных графиков для функций и данных.
- Средства работы с матрицами и массивами данных с поддержкой разрежённых массивов.
- Набор инструментов для добавления собственного пользовательского интерфейса к вычислениям и приложениям.
- Сетевые инструменты для соединения с базами данных SQL, поддержка сетевых протоколов, включая HTTP, NNTP, IMAP, SSH, IRC, FTP.

Sage — сам по себе мощное средство благодаря многочисленным объектно-ориентированным возможностям и большому объёму возможностей, реализованному на python для решения всевозможных задач. Однако следует учитывать, что основная идея **Sage** — интеграция всевозможных математических пакетов, как открытых, так и проприетарных. Наряду с функцией интеграции, **Sage** включает достаточно развитые собственные возможности — многочисленные функции и структуры данных. Преобразование результатов, полученных, например, в **Maxima**, к структурам **Sage** может оказаться достаточно сложной задачей.

5.4 Построение графических иллюстраций: пакет draw

В **Maxima** имеется несколько альтернативных библиотек для отображения графиков функций, наборов точек, трёхмерных 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 функции, доступные на уровне **Maxima**: draw2d, draw3d, draw. Перед использованием draw необходимо загрузить командой load("draw").

Рассмотрим несложный пример. На графике (рис. 5.15) 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*).

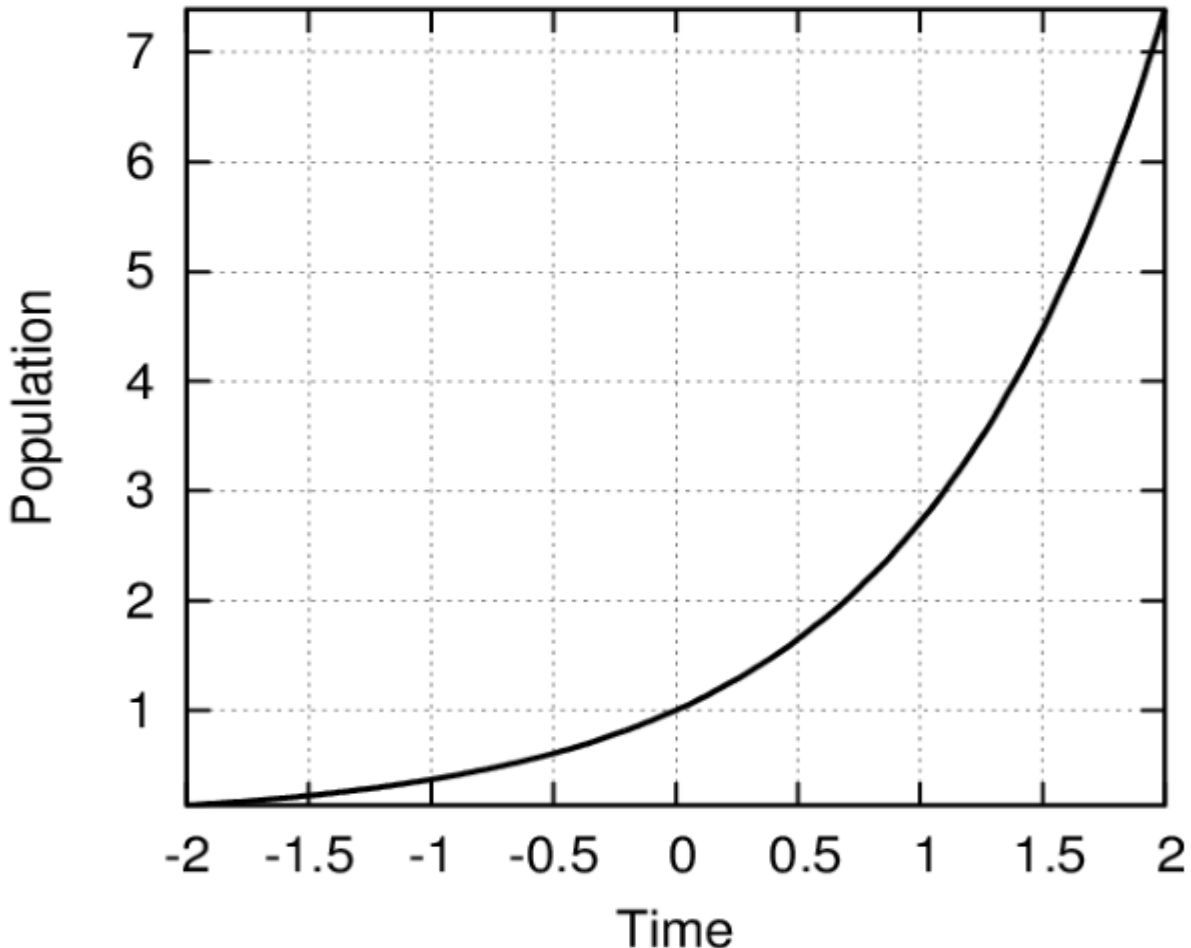


Рис. 5.15. График, построенный с помощью функции *draw2d*

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 рисунок сохраняется в файл *maxima_out.eps*; указание имени файла для вывода осуществляется командой *file_name* = "имя файла".

Построим аналогичный график (рис. 5.16), 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))$
```

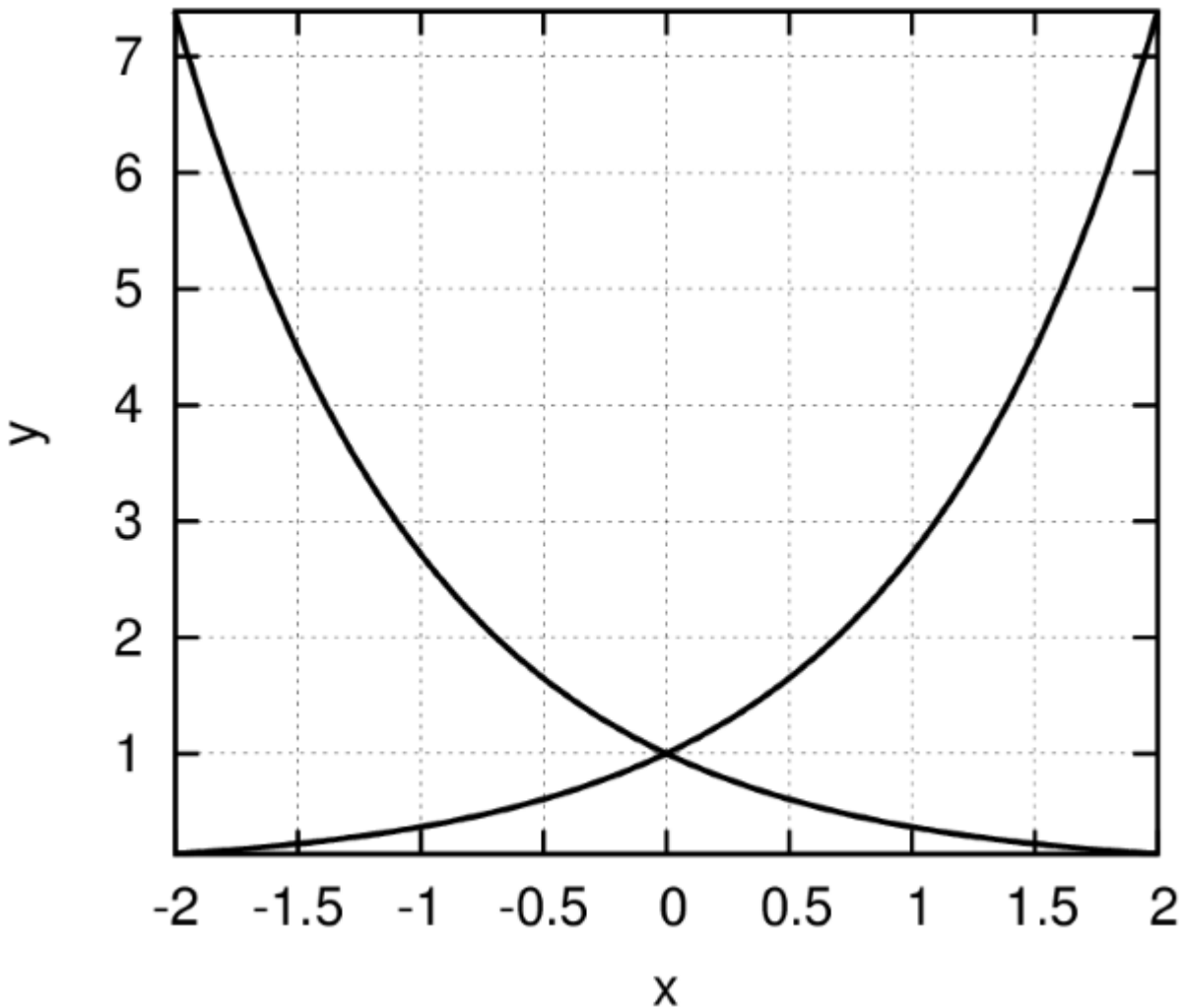


Рис. 5.16. График двух функций (использована функция draw2d)

С помощью пакета draw можно строить и графики функций, заданных неявно. В этом случае функция задаётся командой implicit, например (результат построения — на рис. 5.17):

```
(%i1) load(draw)$
(%i2) draw2d(grid = true, title = "Two implicit
functions",
  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))$
```

На графике хорошо видно, что кривые проведены разными линиями (одна сплошная, другая точечная). Для указания типа линии использована опция line_type=тип линии (возможные значения — solid и dots).

Помимо графиков неявных функций, при помощи 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 полярных координатах — на рис. 5.18. 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))$
```

Two implicit functions

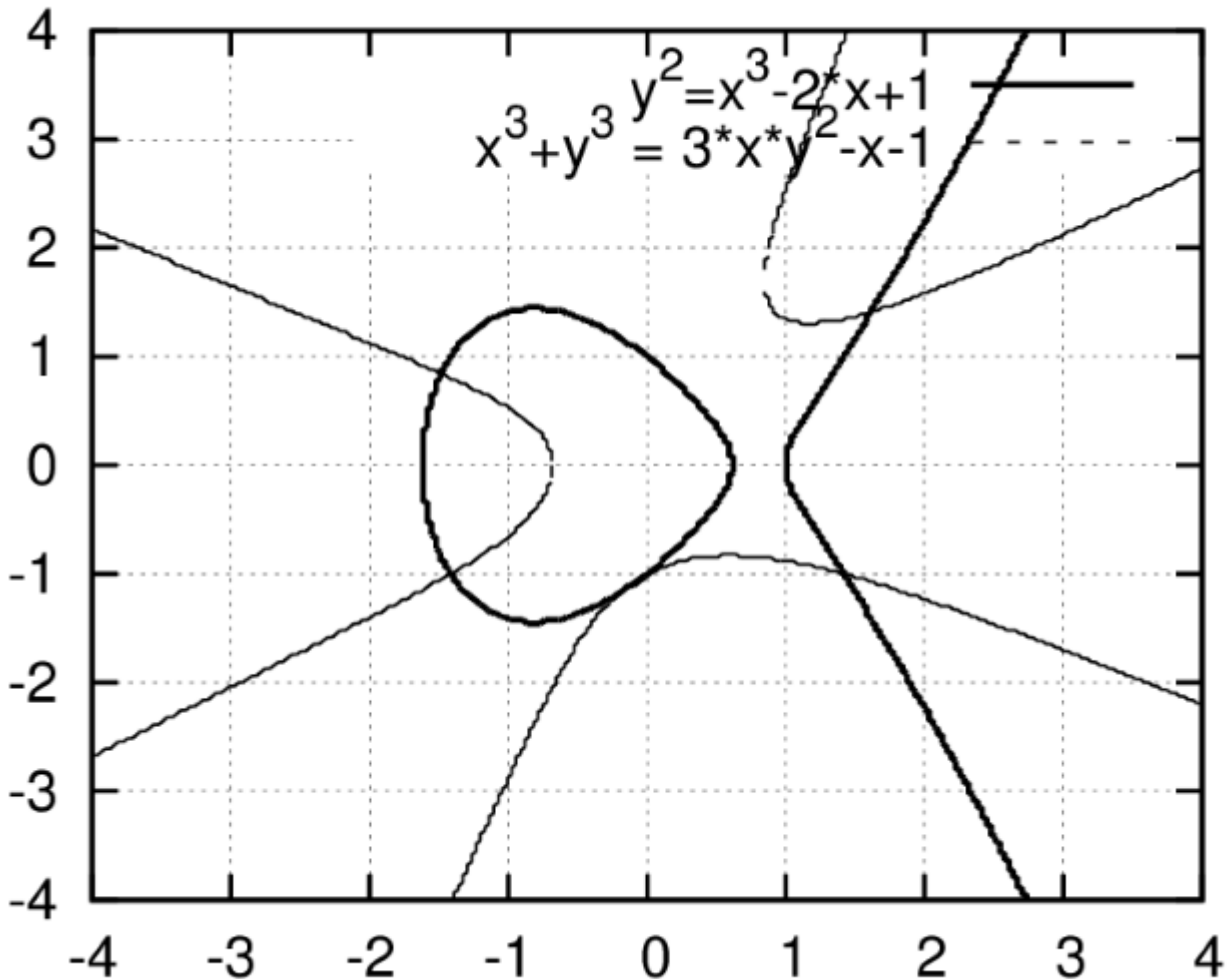


Рис. 5.17. График двух функций, заданных неявно

В последнем примере указаны параметры построения графика: интервалы изменения 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 данного 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 koï8r` или украинскую кодировку `set encoding koï8u`**. Кроме того, часто оказывается необходимым указать и шрифт для вывода заголовка или меток.

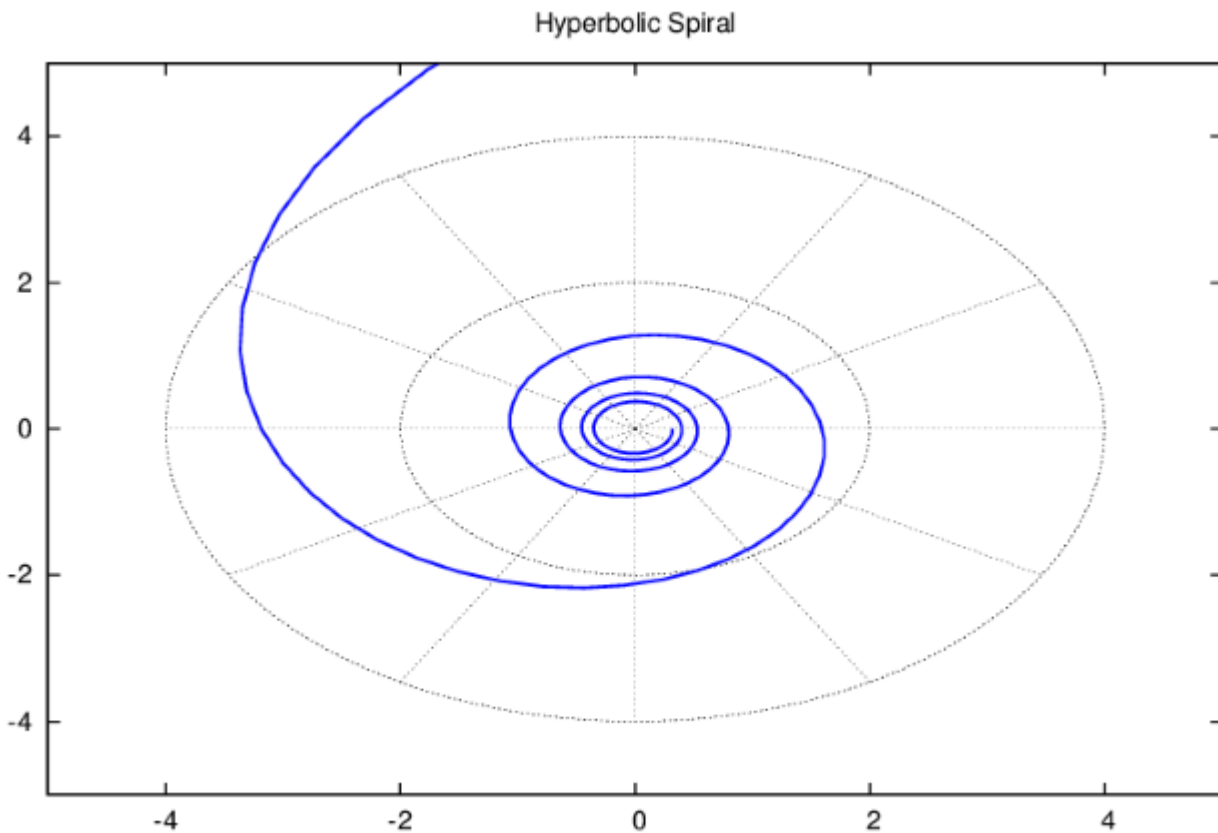


Рис. 5.18. График функции в полярных координатах

Функция `draw3d` позволяет строить трёхмерные графики. **При-** мер:

```
(%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. 5.19.

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. 5.20. Необходимая команда:

```
(%i12) draw3d(terminal=eps, surface_hide = true,
  enhanced3d = true,
  palette=gray, explicit(20*exp(-x^2-y^2)-10, x, -
3, 3, y, -3, 3))$
```

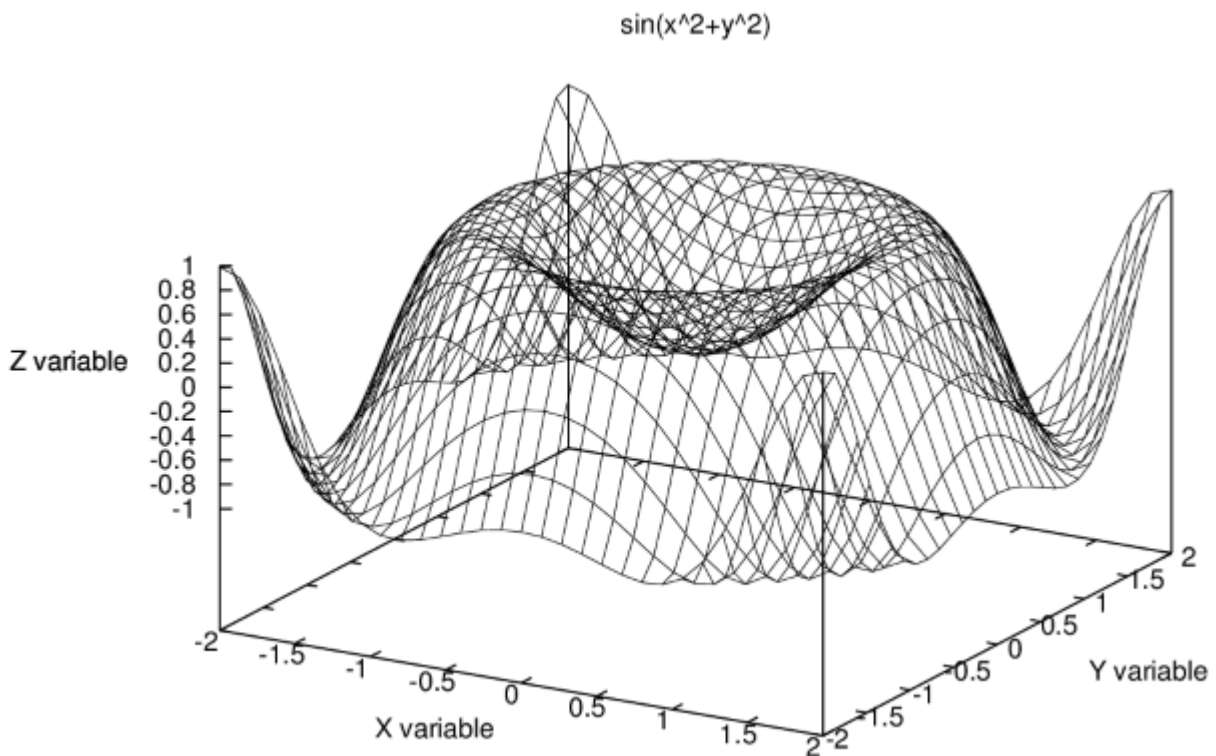


Рис. 5.19. Поверхность, построенная с помощью функции draw3d

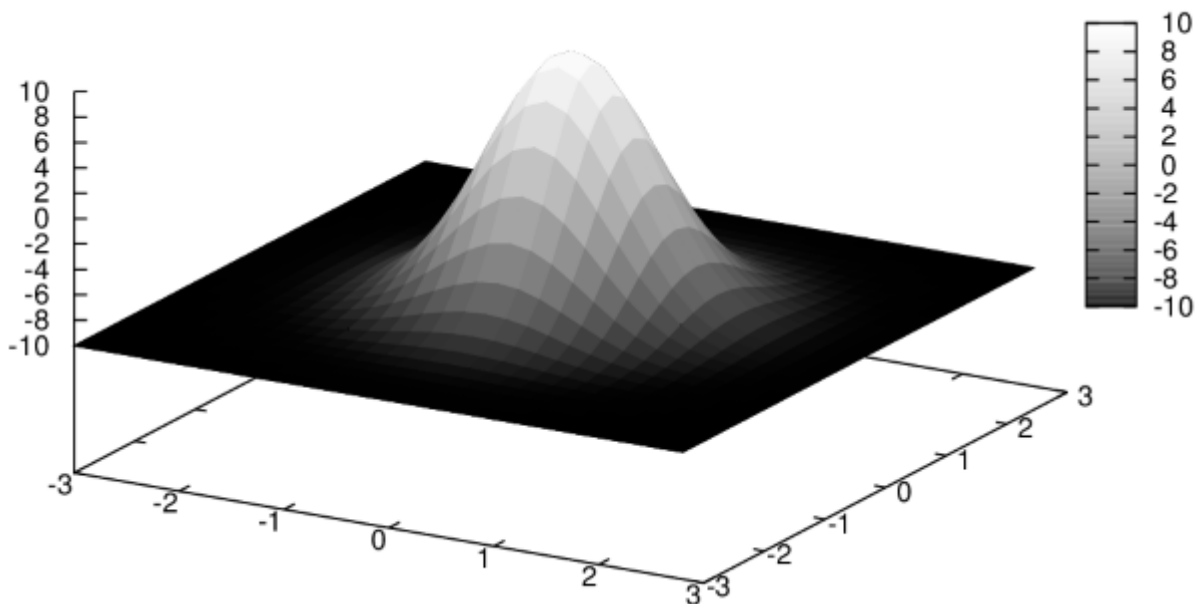


Рис. 5.20. Окрашенная поверхность (использована функция draw3d).

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**.

6. Моделирование с Maxima

6.1 Общие вопросы моделирования

На сегодняшний день существует обширная литература по различным аспектам моделирования систем, природных, технических и экономических объектов.

Одно из широко применяемых универсальных программных средств для решения задач моделирования — пакет Matlab и его расширения для визуального построения блочных моделей — пакет Simulink. Подход, связанный с построением блочных моделей, развивался на протяжении многих лет, и имеет обширную теоретическую базу (см., например, известный учебник по моделированию систем [23]).

Рассматриваемый в данной книге пакет **Maxima** не рассчитан на построение блочных моделей технических систем или систем массового обслуживания, но может быть очень полезен для построения и анализа аналитических или эмпирических идентифицируемых моделей.

Построение, классификация и идентификация математических моделей — широкая область научной и практической деятельности, широко освещённая в литературе различного назначения.

Определим модель как изображение существенных сторон реальной системы (или конструируемой системы), в удобной форме отражающее информацию о ней. Модели бывают концептуальные, физические или математические (другие названия: феноменологические, эмпирические и аналитические) в зависимости от того, какая сторона явления в данном случае наиболее существенна, от методов, которые можно использовать при построении модели, от количества и качества имеющейся информации (см. [24]).

По мнению П. Эйкхоффа, при построении модели информация должна быть представлена в удобной форме. Это существенно, так как модель должна создать предпосылки для следующих решений. Если модель слишком сложна, её полезность становится сомнительной. Относительная простота является главной характеристикой модели. Модель представляет собой упрощённое отображение действительности. Во многих случаях, для того чтобы модель была полезной, её сложность должна находиться в определённом соотношении со сложностью описываемого объекта (пример: биологические системы).

По способу представления информации об исследуемом объекте и способу построения модели делятся на следующие группы:

- словесные или вербальные модели (описания объекта моделирования на естественном языке);
- физические модели, предполагающие представление основных свойств объекта моделирования каким-либо материальным объектом (моделью, макетом и т.п.);

- формальные модели, представляющие собой описание объекта моделирования на формальном языке, к этой группе относятся математические модели.

В свою очередь, математические модели делятся на (см. [25]) графические, табличные, алгоритмические, аналитические.

Средствами **Maxima** можно строить довольно широкий круг различных математических моделей.

Процедуру построения модели во многих источниках называют идентификацией, при этом данный термин часто относится к построению аналитических математических моделей динамических объектов.

Обычно идентификация — многоэтапная процедура. Основные её этапы следующие:

1. Структурная идентификация заключается в определении структуры математической модели на основании теоретических соображений
2. Параметрическая идентификация включает в себя проведение идентифицирующего эксперимента и определение оценок параметров модели по экспериментальным данным.
3. Проверка адекватности — проверка качества модели в смысле выбранного критерия близости выходов модели и объекта.

6.1.1 Аналитические модели

Аналитические модели представляют собой отражение взаимосвязей между переменными объекта в виде математической формулы или группы таких формул. Моделирование основано на двух основополагающих признаках:

- на принципе практической ограниченности количества фундаментальных законов природы;
- на принципе подобия, означающем, что явления различной физической природы могут описываться одинаковыми математическими зависимостями.

Для аналитического моделирования характерно то, что процессы функционирования элементов системы записываются в виде некоторых функциональных соотношений (алгебраических, интегродифференциальных, конечно-разностных и т.п.) или логических условий. Аналитическая модель может быть исследована следующими методами (см. [26]):

- аналитическим, когда стремятся получить в общем виде явные зависимости для искомых характеристик;
- численным, когда, не умея решать уравнений в общем виде, стремятся получить числовые результаты при конкретных начальных данных;
- качественным, когда, не имея решения в явном виде, можно найти некоторые свойства решения (например, оценить устойчивость решения).

Наиболее полное исследование процесса функционирования системы можно провести, если известны явные зависимости, связывающие искомые характеристики с начальными условиями, параметрами и переменными

системы. Однако такие зависимости удаётся получить только для сравнительно простых систем. При усложнении систем исследование их аналитическим методом наталкивается на значительные трудности, которые часто бывают непреодолимыми. Поэтому, желая использовать аналитический метод, в этом случае идут на существенное упрощение первоначальной модели, чтобы иметь возможность изучить хотя бы общие свойства системы. Такое исследование на упрощённой модели аналитическим методом помогает получить ориентировочные результаты для определения более точных оценок другими методами.

Возможности чисто теоретического построения математической модели 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, чисто аналитическим путём построить удовлетворительную модель не удаётся и это побуждает исследователя к формированию модели преимущественно на экспериментальной основе, т.е. в классе эмпирических (идентифицируемых) моделей.

По степени соответствия модели реальному объекту модели можно разделить:

- - на состоятельные — опирающиеся на законы, характеризующие объект моделирования в области их применимости;
- - аппроксимации — построенные на основе приближенных или эмпирических формул, характеризующих объект (их, в отличие от первых, называют несостоятельными — см. [26]).

6.1.2 Идентифицируемые модели

В основе всех ныне весьма многочисленных методов идентификации или опытного отождествления модели с объектом-оригиналом, лежит идея мысленного эксперимента с "чёрным ящиком" (Н. Винер). В предельном (теоретическом) случае "чёрный 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 (см. [27]).

Модель чёрного ящика является начальным этапом изучения сложных систем.

Исследование объекта моделирования 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 счёт оптимального планирования

эксперимента необходимая информация о свойствах и характеристиках объекта получается при минимальном объёме первичных экспериментальных данных и соответственно при минимальной трудоёмкости 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 своём естественном режиме, но при этом организуются систематические измерения и регистрация значений его входных и выходных переменных. Информацию получают ту же, но необходимый объём данных обычно больше, чем в первом случае.

На практике при построении идентифицируемых (эмпирических) моделей часто целесообразна смешанная стратегия эксперимента. При наличии возможности свободного манипулирования параметрами объекта моделирования проводится активный эксперимент. Его результаты дополняют данными пассивного эксперимента, охватывающего все прочие значимые переменные. "Чёрный ящик" — теоретически граничный случай. На деле есть объём исходной информации. На практике приходится иметь дело с "серым" (отчасти прозрачным) ящиком.

Построение модели сводится к следующим этапам (см. [24]):

1. выбор структуры модели из физических соображений;
2. подгонка параметров к имеющимся данным (оценивание);
3. проверка и подтверждение модели (диагностическая проверка);
4. использование модели по её назначению.

Исходя из перечня научных направлений и разнообразия приложений, по этим этапам нельзя дать каких-либо общих рекомендаций. Структура модели выбирается на основе исходной (априорной) информации о системе и преследуемых целях. На практике отыскание подходящей модели может быть достаточно трудной задачей даже для узкой прикладной области.

Различают три основных класса постановки задачи идентификации объекта:

Для сложных и слабо изученных объектов системного характера достоверные исходные данные о внутренних свойствах и структурных особенностях обычно отсутствуют.

Поэтому задача идентификации включает в себя как определение внутренней структуры объекта, так и определение зависимостей, связывающих входы и выходы (обобщённого оператора).

На начальной стадии моделирования строятся эмпирические идентифицируемые модели (на основе статистической обработки экспериментальных результатов).

1. Второй класс задач идентификации характеризуется тем, что имеются априорные данные о структуре моделируемого объекта, в принципе имеются. Однако не определён вклад компонентов объекта в его результирующие характеристики. 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.

2. Третий класс задач связан с относительно простыми и хорошо изученными объектами, структура которых известна точно и речь идёт 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 чёткой 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.

Реальные свойства подавляющего большинства сложных объектов, а также неизбежные случайные погрешности измерений, лежащих в основе идентификации, придают последней статистический характер, что влечёт за собой необходимость получения больших объёмов первичных экспериментальных данных с их последующей обработкой. Поэтому на практике построение моделей путём идентификации неизбежно связано с использованием компьютеров, как при получении первичных данных (автоматизация эксперимента), так и для их обработки и использования.

В **Maxima** включено достаточно большое количество средств для решения задач моделирования, параметрической идентификации, исследования моделей.

6.2 Статистические методы анализа данных

6.2.1 Ввод-вывод матричных данных

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 чисел, 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*).

Основные функции пакета **numericalio**:

- *read_matrix(file_name)*

другие формы вызова —

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)*

— считывает список 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.

6.2.2 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 табл. 6.1.

Таблица 6.1. Функции пакета *descriptive*

Функция	Выполняемые действия	Синтаксис вызова и примечания
<i>mean</i>	Вычисление среднего	<i>mean(list)</i> или <i>mean(matrix)</i>
<i>geometric_mean</i>	Вычисление среднего геометрического	<i>geometric_mean(list)</i> или <i>geometric_mean(matrix)</i>
<i>harmonic_mean</i>	Вычисление среднего гармонического	<i>harmonic_mean(list)</i> или <i>harmonic_mean(matrix)</i>
<i>cor</i>	Вычисляет	<i>cor(matrix)</i> или

Функция	Выполняемые действия	Синтаксис вызова и примечания
	корреляционную матрицу	$cor(matrix, logical_value)$ $logical_value$ равна <i>true</i> или <i>false</i> (при расчёте по ковариационной матрице)
<i>cov, cov1</i>	Вычисляет ковариационную матрицу	$cov1(matrix), cov(matrix)$
<i>median</i>	Вычисляет медиану	$median(list), median(matrix)$
<i>std, std1</i>	Вычисляет среднеквадратичное отклонение (корень квадратный из <i>var</i> или <i>var1</i>)	аналогично <i>var</i>
<i>var, var1</i>	Вычисляет дисперсию случайной величины	$var1(list), var(list)$ $var1(matrix), var(matrix)$,
<i>central_moment</i>	Вычисляет центральный момент порядка <i>k</i>	$central_{m}oment(list, k)$, $central_{m}oment(matrix, k)$
<i>noncentral_mom</i>	Вычисляет момент порядка <i>k</i>	$noncentral_{m}oment(list, k)$, $noncentral_{m}oment(matrix, k)$
<i>skewness</i>	Вычисление асимметрии	$skewness(list), skewness(matrix)$
<i>kurtosis</i>	Вычисление эксцесса	$kurtosis(list), kurtosis(matrix)$
<i>quantile</i>	Вычисление <i>P</i> -квантиля	$quantile(list, p)$, $quantile(matrix, p)$
<i>maxi, mini</i>	Выбор наибольшего и наименьшего значения в выборке	$maxi(list), maxi(matrix)$, $mini(list), mini(matrix)$

Функция	Выполняемые действия	Синтаксис вызова и примечания
	соответственно	
<i>mean_deviation</i>	Сумма абсолютных отклонений от среднего или медианы соответственно	Аналогично <i>mean, median</i>
<i>range</i>	Размах вариации выборки	<i>range(list), range(matrix)</i>
<i>list_correlations</i>	Возвращает список, включающий две матрицы — матрицу, обратную ковариационной, и матрицу частных коэффициентов корреляции	<i>list_correlations(matrix), list_correlations(matrix, logical_value)</i> , где <i>logical_value</i> равна <i>true</i> или <i>false</i> (при расчёте по ковариационной матрице)
<i>subsample</i>	Аналог функции <i>submatrix</i>	
<i>global_variances</i>	Возвращает список, содержащий различные виды дисперсии	<i>global_variances(matrix)</i>

Построение графических иллюстраций производится при помощи функций *scatterplot* (непосредственная визуализация данных), *histogram* (строит гистограмму), *barsplot* (также строит гистограмму, но по дискретным или нечисловым данным), *boxplot* (график Бокса- Вискера), *piechart* (круговая диаграмма). Синтаксис вызова и параметры функций во многом аналогичны компонентам пакета *draw* (см. примеры ниже).

Основные общие опции (унаследованные от *draw*):

- *terminal* — устройство, на которое выводится диаграмма (возможные значения — *eps* и *png*, по умолчанию диаграмма выводится на экран, другие варианты);
- *file_name* — имя файла, в который выводить гистограмму (расширение устанавливается по опции *terminal*)

- title — основной заготовок;
- xlabel, ylabel — названия (метки) осей,

Рассмотрим пример использования функций пакета `descriptive` для статистической обработки массивов данных. Данные берем из файлов, входящих в состав пакета `descriptive` (файлы `biomed.data`, `wind.data` и др.). Перед началом работы загружаем необходимые пакеты `descriptive` и `numericalio`. При помощи функции `read_matrix` считывается матрица, содержащая 100 строк и 5 столбцов.

```
(%i1) load(descriptive)$ load(numericalio)$
      s:read_matrix (file_search ("wind.data"))$
      length(s);
(%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) [17.221906750000001, 14.987736510000001, 15.47572875,
32.176510440000001, 24.423076190000001]
(%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
```

Для построения диаграмм разброса (xy -диаграмм) предназначена функция `scatterplot`. Синтаксис вызова:

```
scatterplot(list)
scatterplot(list, option1, option2, ...)
scatterplot(matrix)
scatterplot(matrix, option1, option2, ...)
```

Данные для функции *scatterplot* могут представляться вектором (списком) или матрицей. Одномерные массивы рассматриваются как временные ряды с равноотстоящими точками.

Основные опции, специфичные для данной функции:

- *point_size* — размер точки на графике (целое положительное число);
- *point_type* — вид точки (отсутствие точек — *none* (-1), *dot* (0), *plus* (1), *multiply* (2), *asterisk* (3), *square* (4), *filled_square* (5), *circle* (6), *filled_circle* (7), *up_triangle* (8), *filled_up_triangle* (9), *down_triangle* (10), *filled_down_triangle* (11), *diamant* (12), *filled_diamant* (13));
- *color* — цвет точки (тот же набор цветов, что и в пакете *draw*);
- *grid* — наличие сетки на графике (*true/false*).

Для построения гистограмм используется функция *histogram* (синтаксис вызова аналогичен *scatterplot* и основные опции идентичны опциям *scatterplot*). Рассмотрим дополнительные опции, специфичные для *histogram*:

- *nclasses* (по умолчанию 10) — число классов гистограммы, или список с указанием пределов классов и их число, или только пределы;
- *frequency* — указывает масштаб шкалы ординат, возможные значения: абсолютный, относительный и процентный (*default*, *absolute*, *persent*);
- *htics* (*default*, *auto*) — формат промежуточных делений на гистограмме, возможные значения — *auto*, *endpoints*, *intervals*, или список меток;

При построении гистограммы доступны также локальные и глобальные опции пакета *draw*.

Для графического представления описательной статистики служит диаграмма Бокса-Уискера ("ящик-с-усами"), которая является удобным способом наглядно представить статистические данные пятью параметрами: наименьшее и наибольшее значения выборки, нижний, средний и верхний квартили. На данной диаграмме могут быть показаны и выбросы (если они есть).

Для построения диаграмм Бокса-Уискера используется функция *boxplot*. Синтаксис вызова: *boxplot(data)* или *boxplot(data, option₁, option₂, ...)*

Параметр *data* — список или матрица с несколькими столбцами. Опции функции *boxplot* идентичны опциям *scatterplot*.

Столбчатые диаграммы (обычно частотные) строятся для данных, разбитых на категории, при помощи функции *barsplot*. Эти диаграммы позволяют графически отобразить различия между данными категорий.

Достаточно распространённым способом графического изображения структуры статистических совокупностей является секторная диаграмма, так

как идея целого очень наглядно выражается кругом, который представляет всю совокупность. Относительная величина каждого значения изображается в виде сектора круга, площадь которого соответствует вкладу этого значения в сумму значений. Этот вид графиков строится в **Maxima** функцией *piechart*.

Рассмотрим примеры использования графических утилит пакета **descriptive**.

Для дальнейшего использования считываем данные из файла `wind.data` (это тестовый файл в составе пакета `descriptive`, содержит матрицу 100×5).

```
(%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)$
```

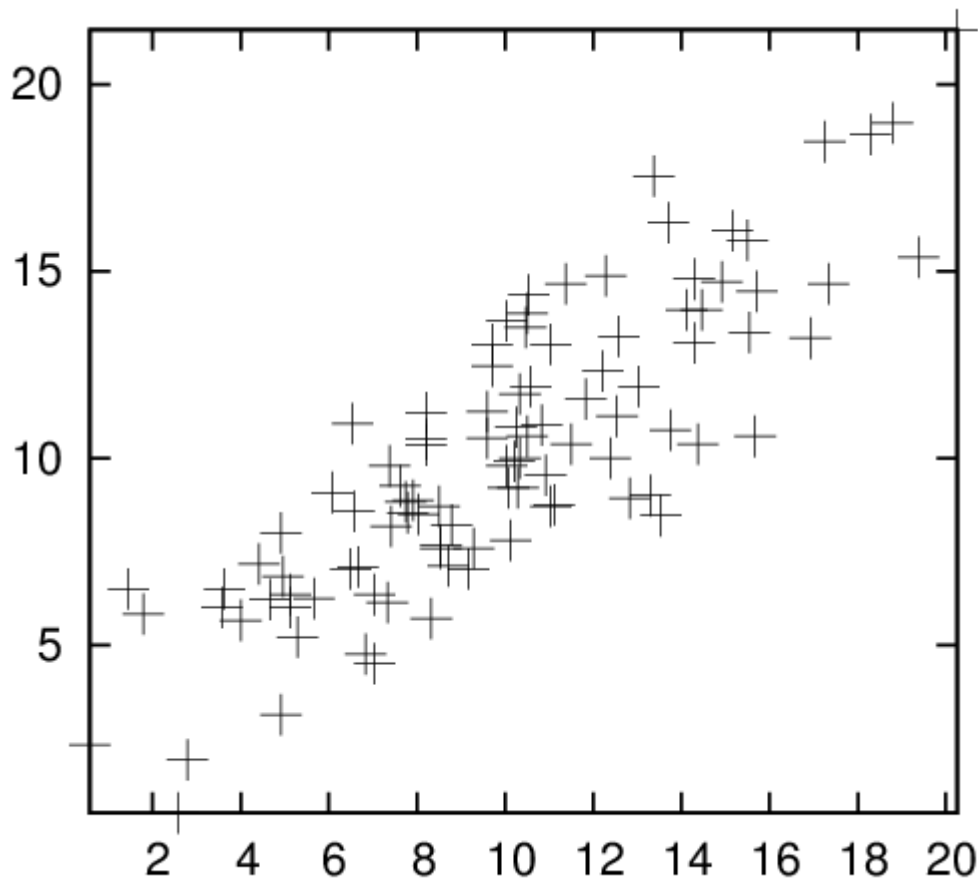


Рис. 6.1. Точечный график

Строим график (точечный) зависимости y от x (см. рис.6.1). Результаты сохраняются в файле `maxima_out.eps` (имя файла — по умолчанию, он создаётся в домашнем каталоге пользователя). Необходимые команды:

```
(%i8) scatterplot(xy,terminal=eps);
```

Считывая данные из файла `pidigits.data`, строим гистограмму частотного распределения десятичных знаков числа π (см. рис.6.2). Результаты сохраняются в файле `histogram.eps` (имя файла задаётся опцией `file_name = "histogram"`, он создаётся в домашнем каталоге пользователя). Необходимые команды:

```
load (descriptive)$ s1 : read_list (file_search
("pidigits.data"))$
histogram (s1, nclasses=8, title="pi digits",
xlabel="digits",
ylabel="Absolute frequency", fill_color=grey,
fill_density=0.6,
terminal=eps, file_name="histogram")$
```

Следующий пример — график Бокса-Уискера с аннотациями по осям (см. рис.6.3). Необходимые команды (результат сохраняется в файле `boxwisker.eps`, англоязычные наименования заменены на русские переводы):

```
(%i10) boxplot(s,title = "Test plot", xlabel =
"Seasons",
terminal=eps,file_name="boxwisker")$
```

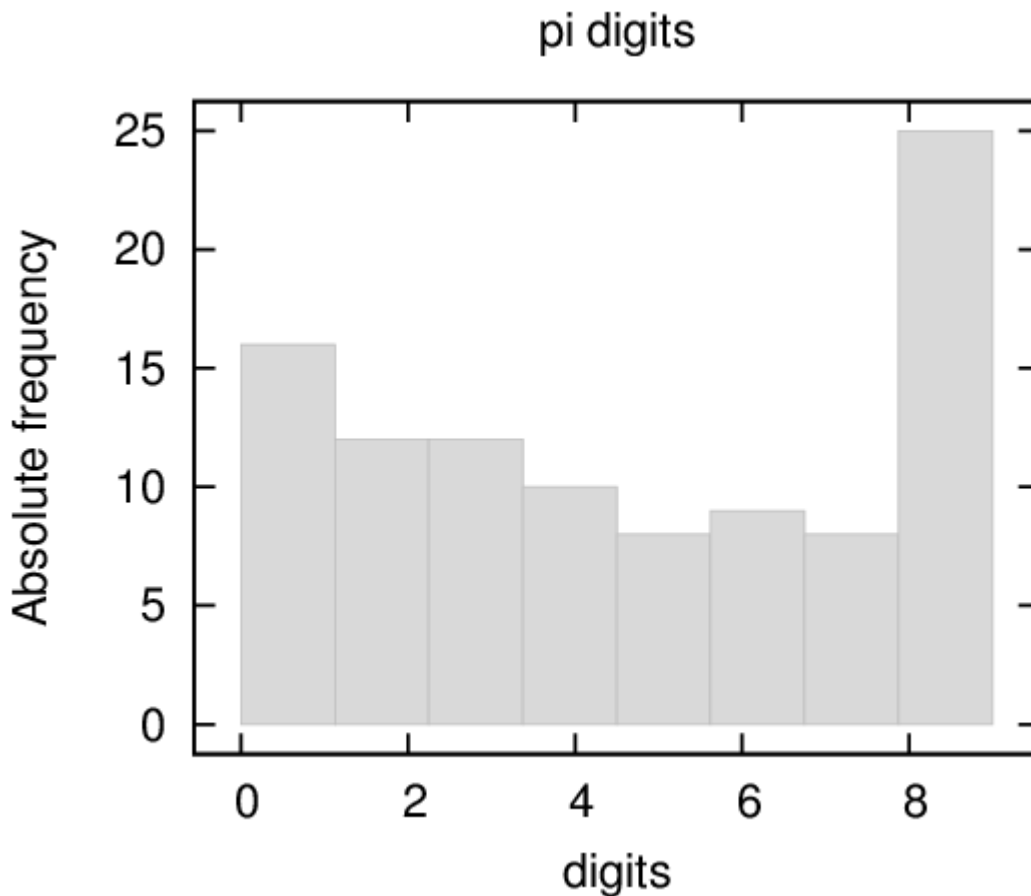


Рис. 6.2. Гистограмма

Пример построения столбчатой диаграммы с использованием функции *barsplot* — на рис.6.4. График построен командой (результат сохраняется в файл *barsplot.eps*):

```
load (descriptive)$
l1:makelist(random(8),k,1,50)$
l2:makelist(random(8),k,1,100)$
barsplot(l1,l2, box_width=1/2,
fill_density=3/4,sample_keys=["A","B"],
      bars_colors=[grey10,grey50], terminal=eps,
file_name="barsplot")$
```

Основные опции команды *barsplot*:

- *box_width* — относительная ширина прямоугольников (по умолчанию 3/4, величина в пределах [0, 1]);
- *grouping* — индикатор, показывающий, как представляются множественные образцы (возможные значения *clustered* и *stacked*);
- *groups_gap* — натуральное число, представляющее разрыв между двумя соседними группами (относительная величина, по умолчанию 1);
- *bars_colors* — список цветов для множественных образцов (по умолчанию []);
- *start_at* — указывает начало графика по оси \mathcal{X} (по умолчанию 0).

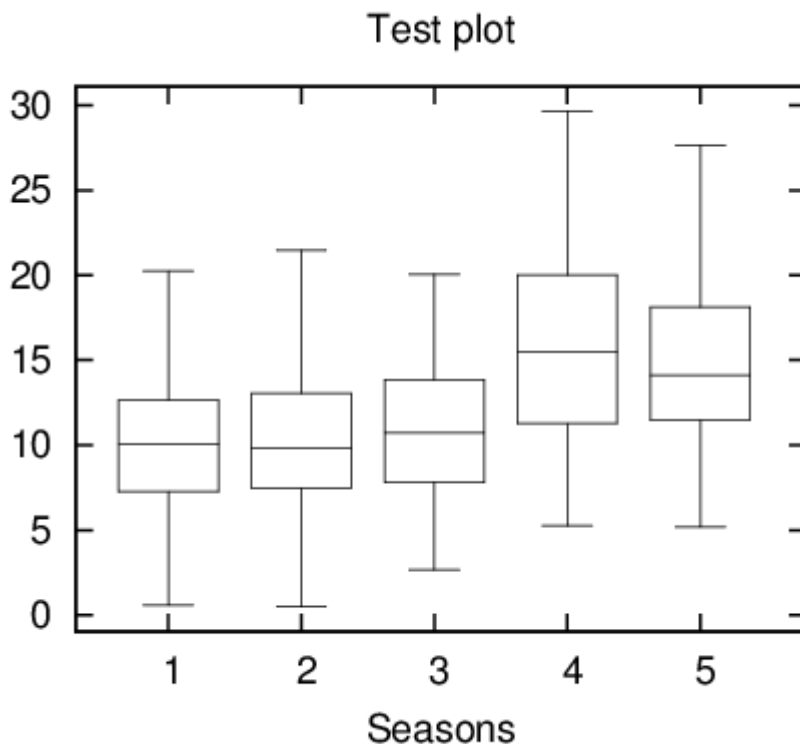


Рис. 6.3. График Бокса-Уискера

С функцией *barsplot* можно использовать и опции пакета *draw*.

Для построения круговых (секторных) диаграмм используется функция *piechart*.

Пример использования *piechart*:

```
load (descriptive)$
s1 : read_list (file_search ("pidigits.data"))$
piechart(s1, xrange=[-1.1, 1.3], yrange=[-1.1, 1.1],
         title="Digit frequencies in pi")$
```

Цвета секторов и радиус диаграммы описываются опциями *sector_colors* и *pie_radius*.

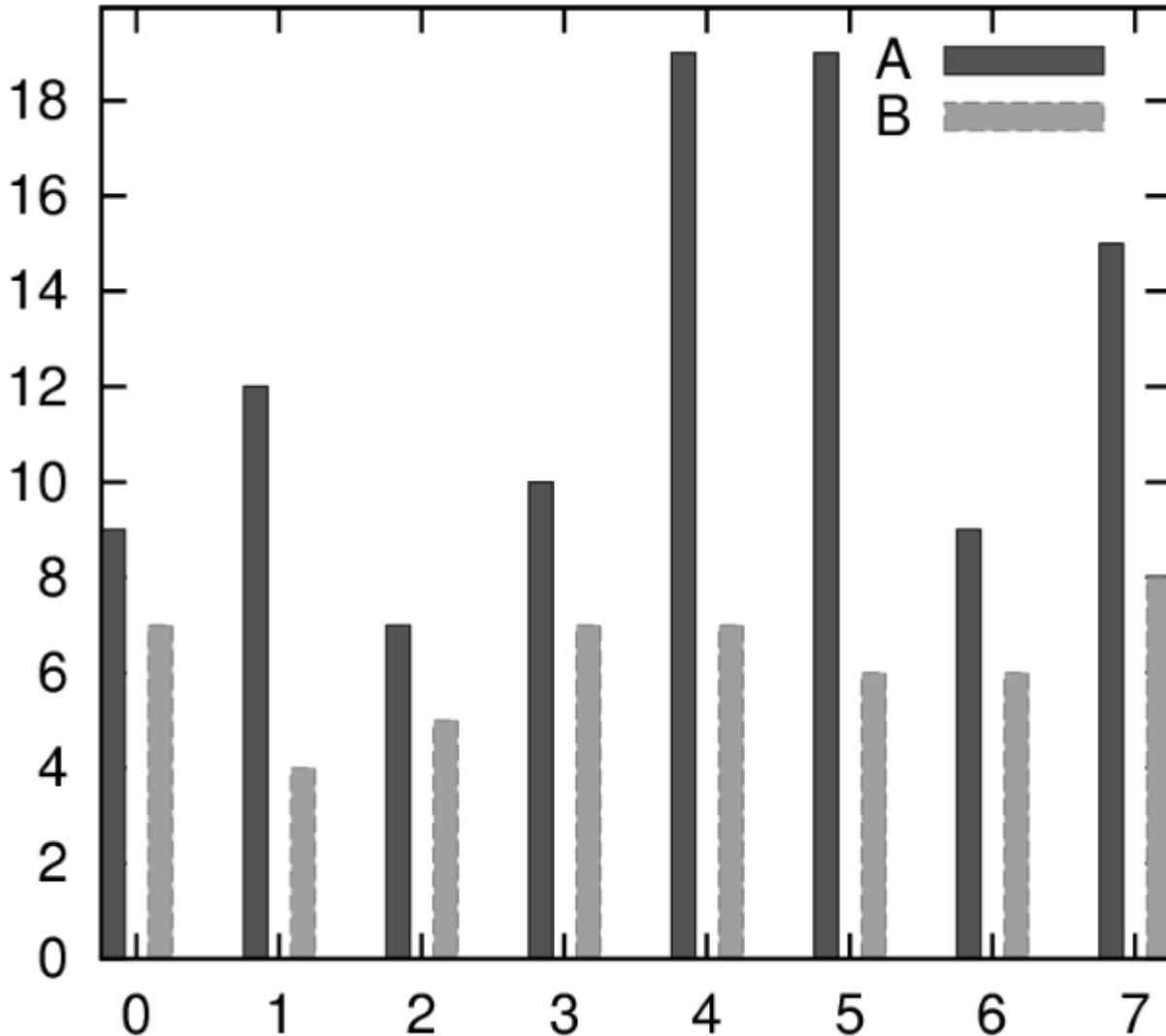


Рис. 6.4. Гистограмма распределения в группах

К сожалению, базовая программа вывода графики **Maxima** — **gnuplot** — написана очень давно, и воспринимает кириллические символы только в кодировках KOI8-R или KOI8-U (в последних версиях — и utf8). Возможным решением (принятым для построения графиков здесь) является создание файла **.gnuplot**, содержащего следующие команды: **set encoding koi8r** или **set encoding utf8**.

Однако и в этом случае может возникнуть необходимость в редактировании файла **maxout.gnuplot**, содержащего команды для построения последнего графика.

6.2.3 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 стандартных 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 распределений и проверки гипотез.

Функция *test_mean* позволяет оценить среднее значение и доверительный интервал по выборке. Синтаксис вызова:

`test_mean(x)` или `test_mean(x, option1, option2, ...)`.

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 ('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).

Результаты, которые 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 результатами 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);
```

```
(%o3) (
      MEAN TEST
      mean_estimate = 54.3
      conf_level = 0.9
      conf_interval = [-∞, 61.51314273502714]
      method = Exact t - test. Unknown variance.
      hypotheses = H0 : mean = 50, H1 : mean < 50
      statistic = .8244705235071678
      distribution = [student_t, 9]
      p_value = .7845100411786887
    )
```

Следующий тест — проверка гипотезы H_0 (среднее равно 50) против альтернативной гипотезы H_1 среднее по выборке отлично от 50. В соответствии с величиной $p \ll 1$ 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);
```

```
(%o2) (
      MEAN TEST
      mean_estimate = 74.88571428571429
      conf_level = 0.95
      conf_interval = [57.72848600856193, 92.04294256286664]
      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, принадлежат ли выборки x_1 и x_2 к одной генеральной совокупности.

Синтаксис вызова: *test_means_difference*(x_1, x_2) или *ttest_means_difference*($x_1, x_2, option_1, option_2, \dots$).

Данная функция выполняет t-тест для сравнения средних по выборкам x_1 и x_2 (x_1, x_2 — списки или одномерные матрицы). Сравнение выборок может проводиться также на основании центральной предельной теоремы (для больших выборок). 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 x_1 и x_2 , 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 альтернативной гипотезы H_1 : различие математических ожиданий статистически значимо, т.е. выборки принадлежат к 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);
```

```
(%o4) (
    DIFFERENCE OF MEANS TEST
    diff_estimate = 20.319999999999999
    conf_level = 0.95
    conf_interval = [-.04597417812881588, ∞]
    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 функции *test_variance*.

Синтаксис 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: Проверка, 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
);
```

```
(%o3) (
      VARIANCE TEST
      var_estimate = 110.75
      conf_level = 0.95
      conf_interval = [57.13433376937479, ∞]
      method = Variance Chi – square test. Unknown mean.
      hypotheses = H0: var = 200, H1: var > 200
      statistic = 4.4300000000000001
      distribution = [chi2, 8]
      p_value = .8163948512777688
    )
```

Сравнение дисперсий двух выборок проводится при помощи функции *test_variance_ratio*. Синтаксис вызова

test_variance_ratio(*x*₁, *x*₂) или
test_variance_ratio(*x*₁, *x*₂, *option*₁, *option*₂, ...).

Данная функция предназначена для сопоставления дисперсий двух выборок с нормальным распределением по критерию Фишера (*F*-тест). Аргументы *x*₁ и *x*₂ — списки или одномерные матрицы, содержащие независимые выборки.

Опции функции *test_variance_ratio*:

- 'mean1, 'mean2, по умолчанию 'unknown, оценки математических ожиданий выборок x_1 и x_2 , 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 альтернативной 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);
```

```
(%o4) (
      VARIANCE RATIO TEST
      ratio_estimate = 2.316933391522034
      conf_level = 0.95
      conf_interval = [.3703504689507263, ∞]
      method = Variance ratio F – test. Unknown means.
      hypotheses = H0: var1 = var2, H1: var1 > var2
      statistic = 2.316933391522034
      distribution = [f, 5, 4]
      p_value = .2179269692254463
    )
```

При отсутствии представлений о распределении выборки может использоваться непараметрический тест для сравнения средних. Оценка медианы непрерывной выборки проводится при помощи функции *test_sign*.

Синтаксис вызова

```
test_sign(x) или
test_sign(x, option1, option2, ...).
```

Функция *test_sign* допускает две опции: *alternative* (аналогично *test_mean*) и *median* (по умолчанию 0, или оценка значения медианы для проверки статистической значимости).

Результаты, которые возвращает функция:

- 'med_estimate: медиана выборки;
- '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 выборки 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);
```

(%o3)
$$\left(\begin{array}{l} \text{SIGN TEST} \\ \text{med_estimate} = 5.1 \\ \text{method} = \text{Non parametric sign test.} \\ \text{hypotheses} = H_0 : \text{median} = 6, H_1 : \text{median} > 6 \\ \text{statistic} = 7 \\ \text{distribution} = [\text{binomial}, 10, 0.5] \\ \text{p_value} = .05468749999999989 \end{array} \right)$$

Аналогичная функция — $\text{test_signed_rank}(x)$

(либо с указанием опций $\text{test_signed_rank}(x, \text{option}_1, \text{option}_2, \dots)$),

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 альтернативной 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);
```

(%o3)
$$\left(\begin{array}{l} \text{SIGNED RANK TEST} \\ \text{med_estimate} = 15.7 \\ \text{method} = \text{Exacttest} \\ \text{hypotheses} = H_0 : \text{med} = 15, H_1 : \text{med} > 15 \\ \text{statistic} = 14 \\ \text{distribution} = [\text{signed_rank}, 6] \\ \text{p_value} = 0.28125 \end{array} \right)$$

Непараметрическое сравнение 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 x_1 и x_2 , не являющихся нормальными (отсутствие нормальности не позволяет применить t - тест).

Синтаксис вызова:

`test_rank_sum(x1, x2)` или `test_rank_sum(x1, x2, option1)`.

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 x_1 и x_2 .

```
(%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);
```

```
(%o4) (
      RANK SUM TEST
      method = Exact test
      hypotheses = H0 : med1 = med2, H1 : med1 # med2
      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');
```

```
(%o4) (
      RANK SUM TEST
      method = Asymptotic test. Ties
      hypotheses = H0 : med1 = med2, H1 : med1 < med2
      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 сообщение об 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 распределении генеральной совокупности по заданной выборке.

```
(%i1) load("stats") $
(%i2) x: [12, 15, 17, 38, 42, 10, 23, 35, 28] $
(%i3) test_normality(x);
(%o3) (SHAPIRO - WILK TEST)
      (statistic = .9251055695162439
       p_value = .4361763918860427)
```

6.2.4 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 $y = a_0 + a_1 \cdot x$ (т.е. только простейшей).

Синтаксис вызова:

simple_linear_regression(x) или
simple_linear_regression(x, option₁).

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:

- 'model: полученное уравнение регрессии;
- 'means: среднее;
- 'variances: дисперсии обоих переменных;
- 'correlation: коэффициент корреляции;
- 'adc: коэффициент детерминации;
- 'a_estimation: оценка параметра a ;
- 'a_conf_int: доверительный интервал для a ;
- 'b_estimation: оценка параметра b ;
- 'b_conf_int: доверительный интервал для b ;

- 'hypotheses: нулевая и альтернативная гипотеза относительно параметра b ;
- 'statistic: статистические характеристики выборки, использованные для проверки нулевой гипотезы;
- 'distribution: распределение выборки;
- 'p_value: величина вероятности для проверки гипотезы о статистической значимости b ;
- 'v_estimation: оценка остаточной дисперсии;
- 'v_conf_int: доверительный интервал для остаточной дисперсии
- 'cond_mean_conf_int: доверительный интервал для среднего;
- 'new_pred_conf_int: доверительный интервал для нового предсказания;
- 'residuals: список, содержащий остатки.

По умолчанию на консоль выводятся только параметры 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]]$
```

Вычисляем коэффициенты и прочие параметры регрессии

```
(%i10) z:simple_linear_regression(s,conflevel=0.99);
```

```
(%o10)
```

```

      SIMPLE LINEAR REGRESSION
      model = 1.405999999999986 x - 31.189999999999804
      correlation = 0.96116852552552
      v_estimation = 13.579666666666665
      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

```

```
(%i11) z:simple_linear_regression(s,conflevel=0.95);
```

```
(%o11)
```

```

      SIMPLE LINEAR REGRESSION
      model = 1.405999999999986 x - 31.189999999999804
      correlation = 0.96116852552552
      v_estimation = 13.579666666666665
      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

```

Некоторые дополнительные параметры:

```
(%i5) take_inference(model,z), x=133;
```

```
(%o5) 155.808
```

```
(%i6) take_inference(means,z);
```

```
(%o6) [135.0, 158.62]
```

```
(%i7) take_inference(new_pred_conf_int,z), x=133;
```

```
(%o7) [132.0728595995113, 179.5431404004887]
```

Графическая иллюстрация построенной линейной зависимости см. на рис.6.5. Использованная команда:

```
(%i11) plot2d([[discrete,s],
take_inference(model,z)],
[x,120,150],[style,[points],[lines]],
[gnuplot_term,ps],
[gnuplot_out_file,"regress.eps"])$
```

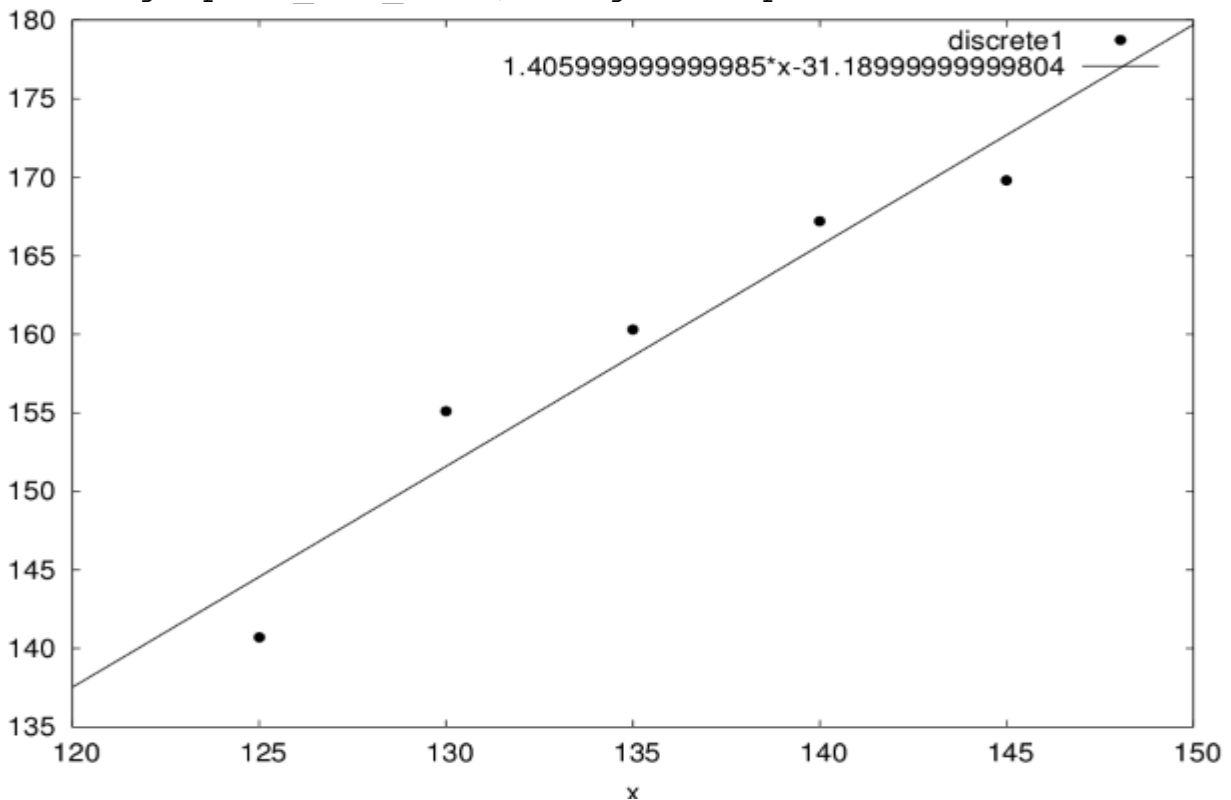


Рис. 6.5. Простая линейная регрессия

6.2.5 Использование метода наименьших квадратов

Пакет **Maxima** включает мощный модуль для линейного и нелинейного оценивания параметров различных моделей с использованием метода наименьших квадратов — пакет `lsquares`.

Основная функция пакета `lsquares` — это функция *lsquares_estimates*.

Синтаксис вызова:

lsquares_estimates(*D*, *x*, *e*, *a*) или

lsquares_estimates(*D*, *x*, *e*, *a*, *initial* = *L*, *tol* = *t*)

Функция предназначена для оценки параметров, лучше всего соответствующих уравнению e в переменных x и a по набору данных D , которые определяются методом наименьших квадратов. Функция *lsquares_estimates* сначала пытается отыскать точное решение, и если это не удаётся, ищет приближительное решение. Возвращаемое значение — список вида $[a = \dots, b = \dots, c = \dots]$. Элементы списка обеспечивают минимум среднеквадратичной 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 отыскиваются оценки. Уравнение e является выражением или уравнением в переменных x и a ; если e записано не в форме уравнения, его рассматривают как уравнение $e = 0$. 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, рациональные, с плавающей точкой, и с плавающей точкой повышенной точности). Численные расчеты выполняются с обычной арифметикой с плавающей точкой, таким образом все другие виды чисел преобразуются 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]);
```

$$(\%o2) \begin{pmatrix} 1 & 1 & 1 \\ \frac{3}{2} & 1 & 2 \\ \frac{9}{4} & 2 & 1 \\ 3 & 2 & 2 \\ 2 & 2 & 1 \end{pmatrix}$$

```
(%i3) lsquares_estimates(M,[z,x,y],(z+D)^2=A*x+B*y+C,
                          [A,B,C,D]);
```

$$(\%o3) \left[A = -\frac{59}{16}, B = -\frac{27}{16}, C = \frac{10921}{1024}, D = -\frac{107}{32} \right]$$

Другой пример (точное решение отсутствует, отыскивается приближенное):

```
(%i1) load (lsquares)$
```

```
M : matrix ([1, 1], [2, 7/4], [3, 11/4],
            [4, 13/4]);
```

$$(\%o2) \begin{pmatrix} 1 & 1 \\ 2 & \frac{7}{4} \\ 3 & \frac{11}{4} \\ 4 & \frac{13}{4} \end{pmatrix}$$

```
(%i3) lsquares_estimates(M, [x, y], y=a*x^b+c, [a, b, c],
            initial=[3, 3, 3], iprint=[-1, 0]);
```

```
(%o3) [[a = 1.375751433061394, b = .7148891534417651,
c = -.4020908910062951]]
```

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 и для функции $lsquares_estimates$).

Пример 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]);
```

$$(\%o4) \quad \left[\left[A = \frac{8231525}{267474}, B = \frac{87875}{133737} \right] \right]$$

```
(%i5) float(%);
```

$$(\%o5) \quad \left[\left[A = 30.77504729431646, B = 0.65707321085414 \right] \right]$$

```
(%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 пакета `lsquares`, по синтаксису использования и идее реализации аналогичны приведенным (см. документацию разработчика).

6.3 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, подчёркивая, 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.

6.3.1 Моделирование системы химических реакций

Рассмотрим другой пример. Исследуем систему из трех дифференциальных уравнений, описывающих модель химической кинетики:



Система соответствующих дифференциальных уравнений

$$\frac{dc_A}{dt} = -k_1 c_A$$

$$\frac{dc_B}{dt} = k_1 c_A - k_2 c_B$$

$$\frac{dc_C}{dt} = k_2 c_B$$

Начальные условия:

$$c_A = 1, c_B = 0, c_C = 0$$

Результаты решения приведены на рис.6.6.

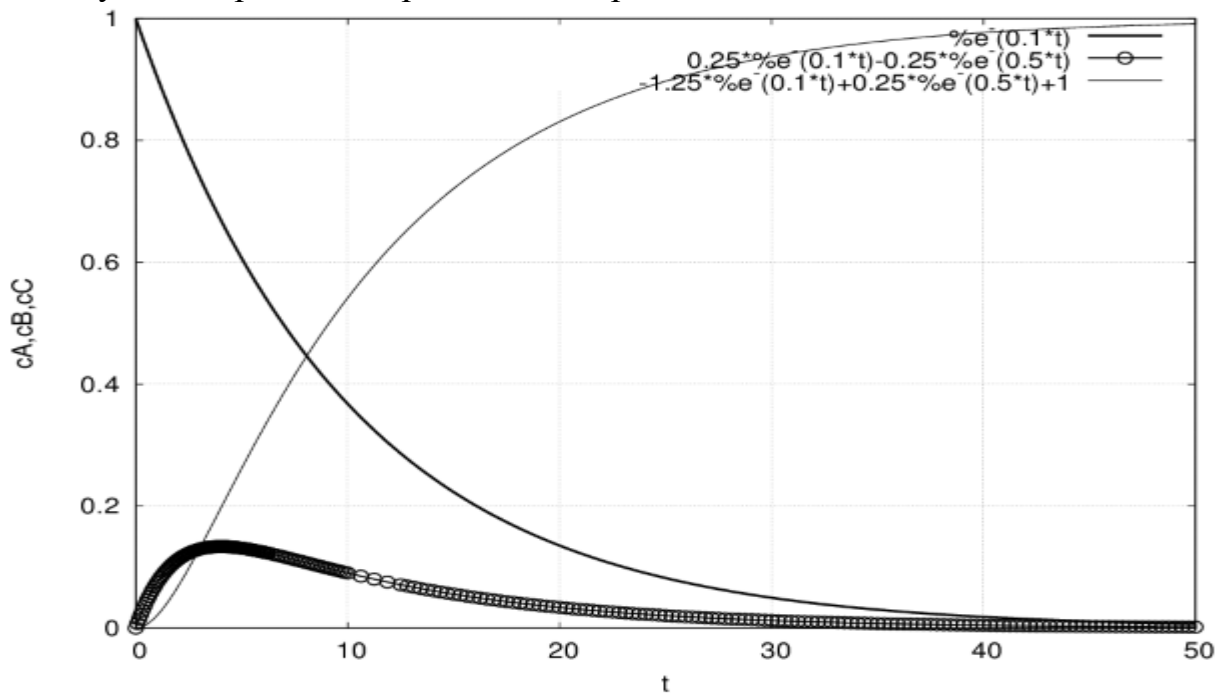


Рис. 6.6. Кинетика химических реакций

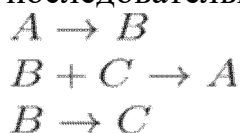
```
(%i1) eq1: 'diff(ca(t), t) = -k1*ca(t);
eq2: 'diff(cb(t), t) = k1*ca(t) - k2*cb(t);
eq3: 'diff(cc(t), t) = k2*cb(t);
```

```

(%o1)  $\frac{d}{dt} ca(t) = -k1 ca(t)$ 
(%o2)  $\frac{d}{dt} cb(t) = k1 ca(t) - k2 cb(t)$ 
(%o3)  $\frac{d}{dt} cc(t) = k2 cb(t)$ 
(%i4) atvalue(ca(t), t=0, 1);
(%o4) 1
(%i5) atvalue(cb(t), t=0, 0);
      atvalue(cc(t), t=0, 0);
(%o5) 0
(%o6) 0
(%i7) sol:desolve([eq1,eq2,eq3],[ca(t),cb(t),cc(t)]);
(%o7)  $[ca(t) = e^{-k1t}, cb(t) = \frac{k1 e^{-k1t}}{k2-k1} - \frac{k1 e^{-k2t}}{k2-k1},$ 
 $cc(t) = \frac{k1 e^{-k2t}}{k2-k1} - \frac{k2 e^{-k1t}}{k2-k1} + 1]$ 
(%i8) ratsimp(sol);
(%o8)  $[ca(t) = e^{-k1t}, cb(t) = \frac{(k1 e^{k2t} - k1 e^{k1t}) e^{-k2t-k1t}}{k2-k1},$ 
 $cc(t) = \frac{(((k2-k1) e^{k1t} - k2) e^{k2t} + k1 e^{k1t}) e^{-k2t-k1t}}{k2-k1}]$ 
(%i9) k1:0.1; k2:0.5; ev((sol));
(%o9) 0.1
(%o10) 0.5
(%o11)  $[ca(t) = e^{-0.1t}, cb(t) = 0.25 e^{-0.1t} - 0.25 e^{-0.5t}, cc(t) =$ 
 $-1.25 e^{-0.1t} + 0.25 e^{-0.5t} + 1]$ 
(%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_file, "chem.png"]);

```

Несколько более сложная задача — моделирование кинетики параллельно-последовательных реакций, протекающих по схеме:



В зависимости от констант скорости химических реакций данная система может быть довольно жёсткой.

6.3.1.1 Пример командного файла для решения жёсткой системы

ОДУ в **Maxima** (данная система нелинейна, поэтому используем метод Рунге-Кутты, 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 $k_2 = 1000$ и $k_3 = 100$ 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 брюсселятор играет роль базовой модели, такую же как гармонический осциллятор в физике, или модели Вольтерра в динамике популяций.

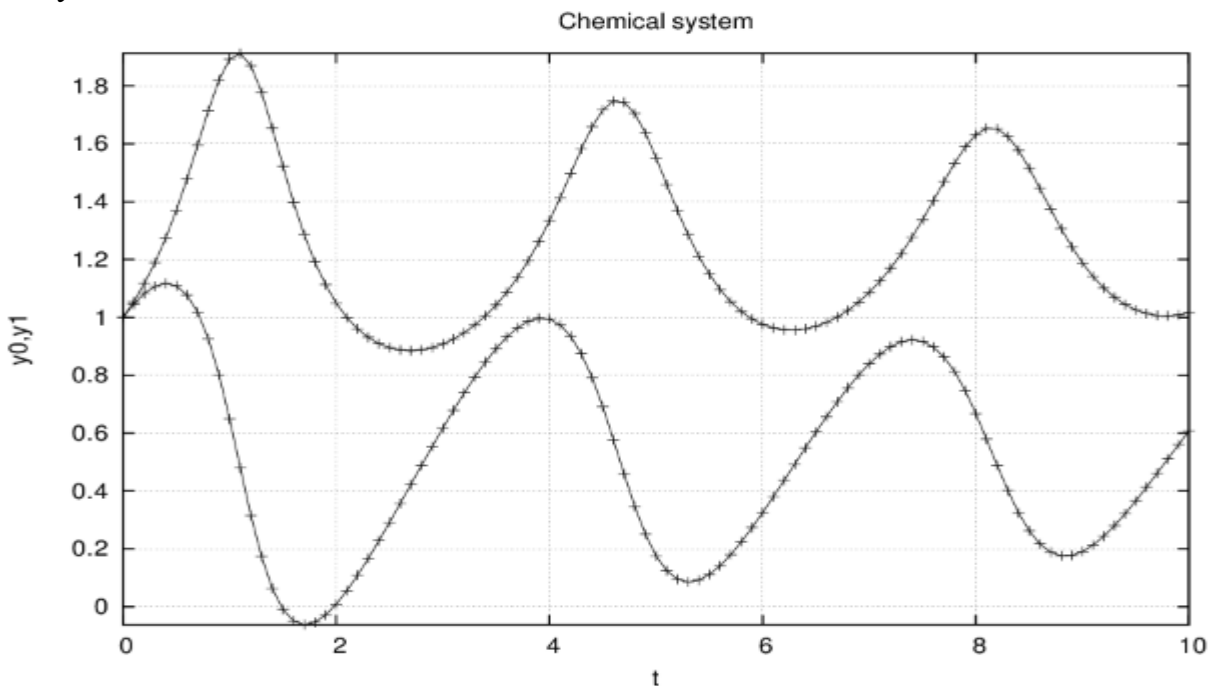


Рис. 6.7. Изменение концентраций при моделировании автоколебательной химической реакции (брюсселятора)

Здесь брюсселятор рассматривается как пример 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.67., а фазовые портреты — на рис.6.8 и рис.6.9.

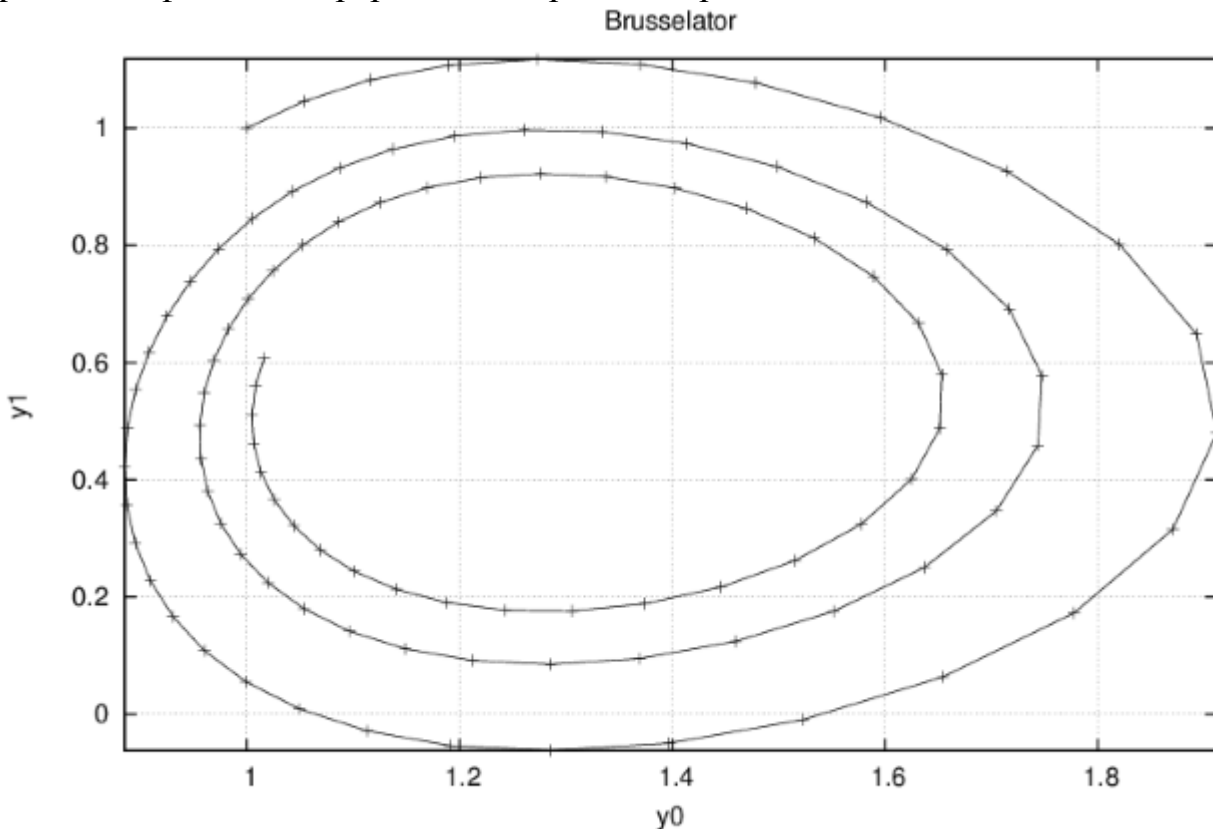


Рис. 6.8. Фазовый портрет для брюсселятора ($B=0.5$)

Brusselator

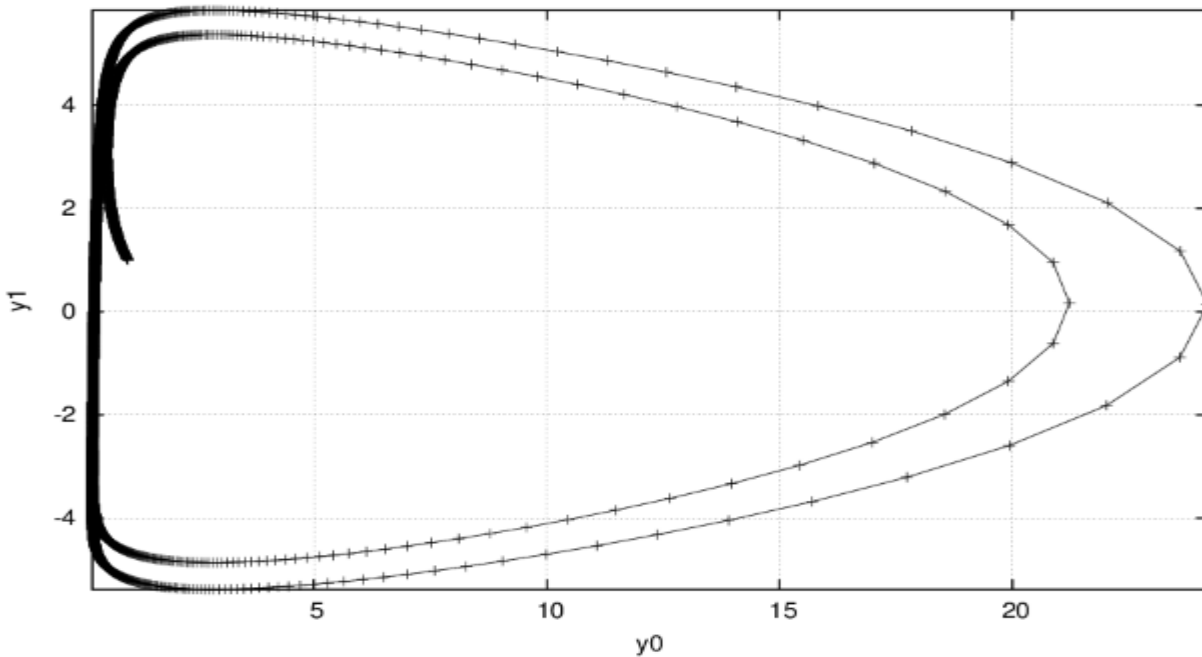


Рис. 6.9. Фазовый портрет для брюсселятора ($B=2.5$)

При проведении расчётов следует обратить внимание на жёсткость системы ОДУ, описывающей брюсселятор, в частности, при $B = 2.5$ для построения приведённой иллюстрации необходимо было уменьшить шаг по времени до 0.002. При очередном запуске командного файла, содержащего команды загрузки пакетов, рекомендуется перезапустить **Maxima**.

6.3.2 Фазовые портреты динамических систем

Для изучения динамических систем центральным моментом является анализ фазовых портретов, т. е. решений, получающихся при выборе всевозможных начальных условий.

Решение ОДУ часто удобнее изображать не в виде графика $y_0(t), y_1(t), \dots$, 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. $y(t \rightarrow \infty)$, 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.

6.3.3 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

$$\begin{aligned} \frac{d}{dt} y(t) &= y(t) (a - b z(t)) - \alpha y(t)^2 \\ \frac{d}{dt} z(t) &= (-c + d y(t)) z(t) - \alpha z(t)^2 \end{aligned}$$

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 рис.6.10, рис.6.11 и рис.6.12.

```
(%i1) a:4$ b:2.5$ c:2$ d:1$ alpha=0$
      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;
      atvalue(y(t),t=0,3); atvalue(z(t),t=0,1);
(%o6)  $\frac{d}{dt}y(t) = y(t)(4 - 2.5z(t)) - \alpha y(t)^2$ 
(%o7)  $\frac{d}{dt}z(t) = (y(t) - 2)z(t) - \alpha y(t)^2$ 
(%o8) 3
(%o9) 1
(%i10) desolve([eq1,eq2],[y(t),z(t)]);
rat: replaced -2.5 by -5/2 = -2.5
rat: replaced -2.5 by -5/2 = -2.5
rat: replaced -2.5 by -5/2 = -2.5
rat: replaced 2.5 by 5/2 = 2.5
(%o10)
[y(t) = ilt  $\left( \frac{5 \text{laplace}(y(t)z(t),t,g2176)+2\alpha \text{laplace}(y(t)^2,t,g2176)-6}{2g2176-8}, g2176, t \right),$ 
z(t) = ilt  $\left( \frac{\text{laplace}(y(t)z(t),t,g2176)-\alpha \text{laplace}(y(t)^2,t,g2176)+1}{g2176+2}, g2176, t \right)]$ 
```

Очевидная проблема — неразрешимость данной системы в явном виде методом преобразования Лапласа, т.к. она нелинейна.

Используем численный метод Рунге-Кутты из пакета **dynamics**.

Результаты решения для значений $\alpha = 0$ и $\alpha = 0.02$ представлены на рис.6.11 и рис.6.12.

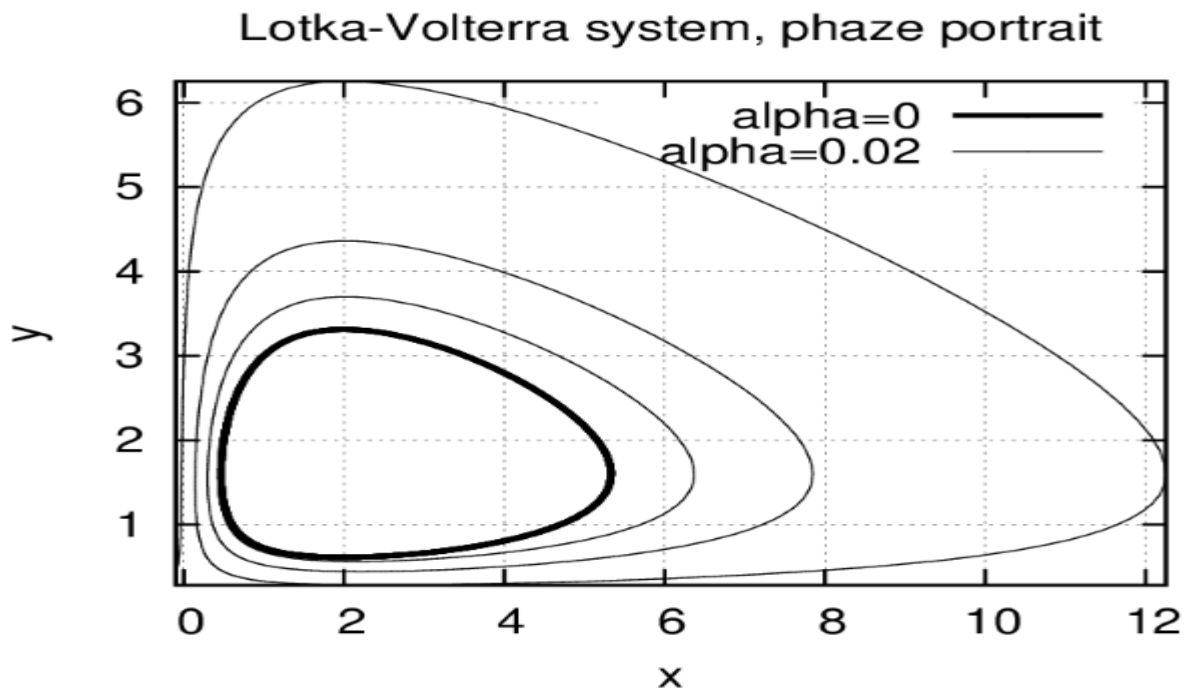


Рис. 6.10. Фазовый портрет для системы Лотка-Вольтерра

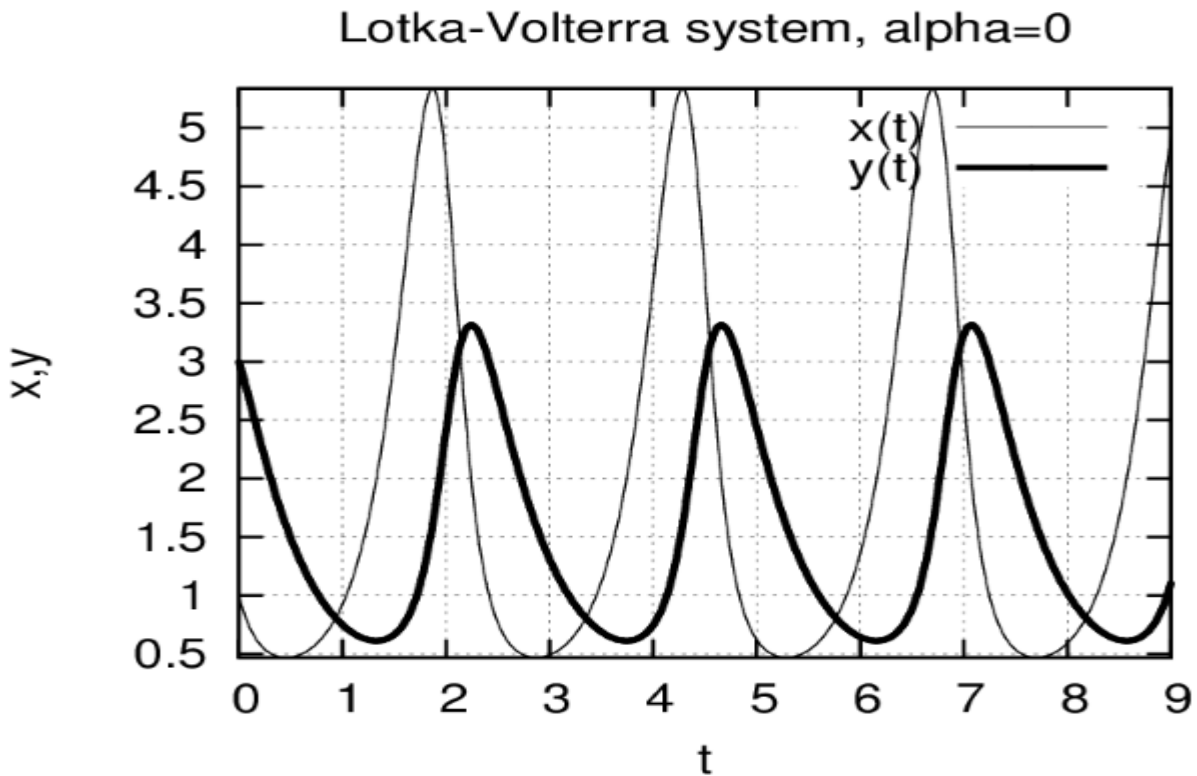


Рис. 6.11. Решения системы Лотка-Вольтерра в зависимости от времени ($\alpha = 0$)

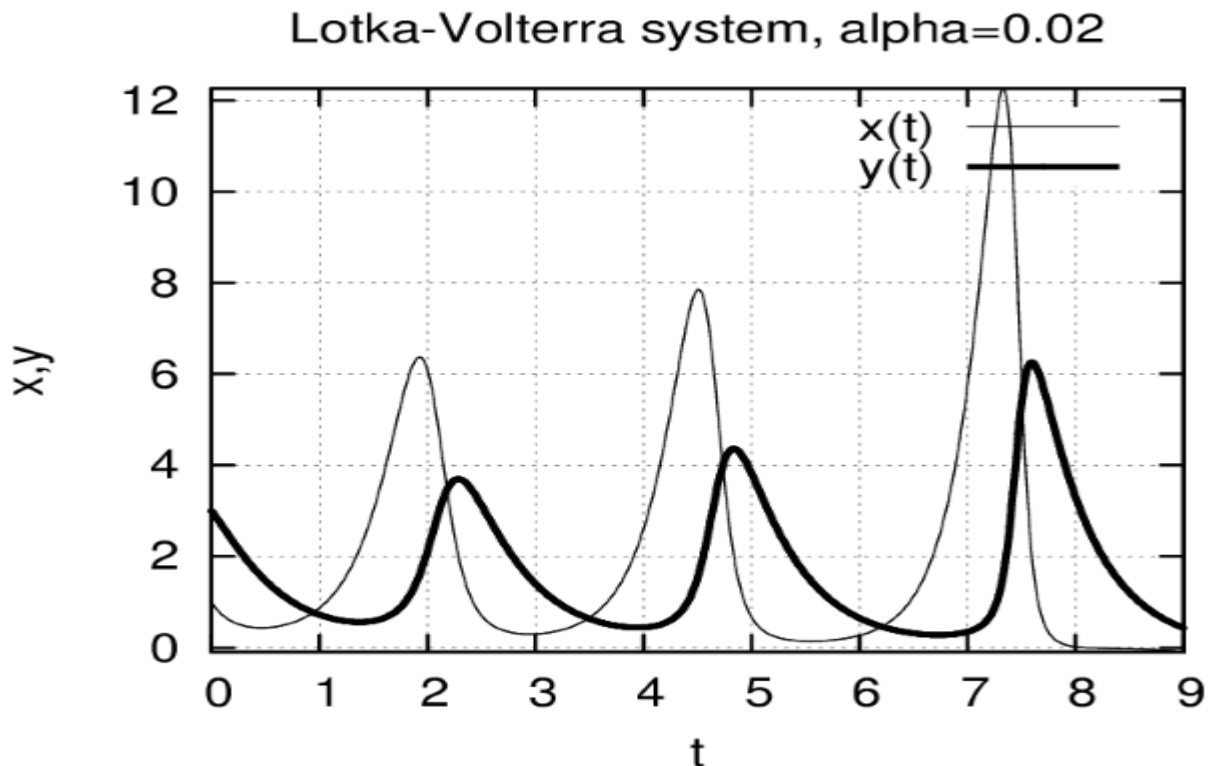


Рис. 6.12. Решения системы Лотка-Вольтерра в зависимости от времени ($\alpha = 0, 1$)

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*x^2$ ode2:(-c+d*y)*x-alpha1*y^2$
```

```

alpha2:0.02;
ode3:(a-b*x)*y-alpha2*x^2$ ode4:(-c+d*y)*x-alpha2*y^2$
load("dynamics");
t1:[]$ xg1:[]$ yg1:[]$ t2:[]$ xg2:[]$ yg2:[]$
l1:rk([ode1,ode2],[y,x],[1,3],[t,0,9,0.01])$
l2:rk([ode3,ode4],[y,x],[1,3],[t,0,9,0.01])$
for i:1 thru length(l1) do(t1:append(t1,[l1[i][1]]),
    xg1:append(xg1,[l1[i][2]]),
    yg1:append(yg1,[l1[i][3]]));
for i:1 thru length(l2) do(t2:append(t2,[l2[i][1]]),
    xg2:append(xg2,[l2[i][2]]),
    yg2:append(yg2,[l2[i][3]]));
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,
    point_type = none,
    line_width = 4,color = black, points(xg1,yg1),
    points_joined = true, color = black,
    point_type = none,
    line_width = 1,key="alpha=0.02", 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, line_width = 1,
    color = black,point_type = none, points(t1,xg1),
    points_joined = true, line_width = 4,
    point_type = none,
    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=0.02",
    key= "x(t)",points_joined = true,
    point_type = none,
    line_width = 1, color = black, points(t2,xg2),
    points_joined = true, line_width = 4,
    point_type = none,
    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 он преобразуется в векторы, используемые для построения графических иллюстраций.

6.3.4 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);
```

```
(%o1)  $\frac{d}{dt} x_1(t) = x_2(t) x_3(t)$ 
```

```
(%o2)  $\frac{d}{dt} x_2(t) = -x_1(t) x_3(t)$ 
```

```
(%o3)  $\frac{d}{dt} x_3(t) = -0.51 x_1(t) x_2(t)$ 
```

```
(%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.6.13.

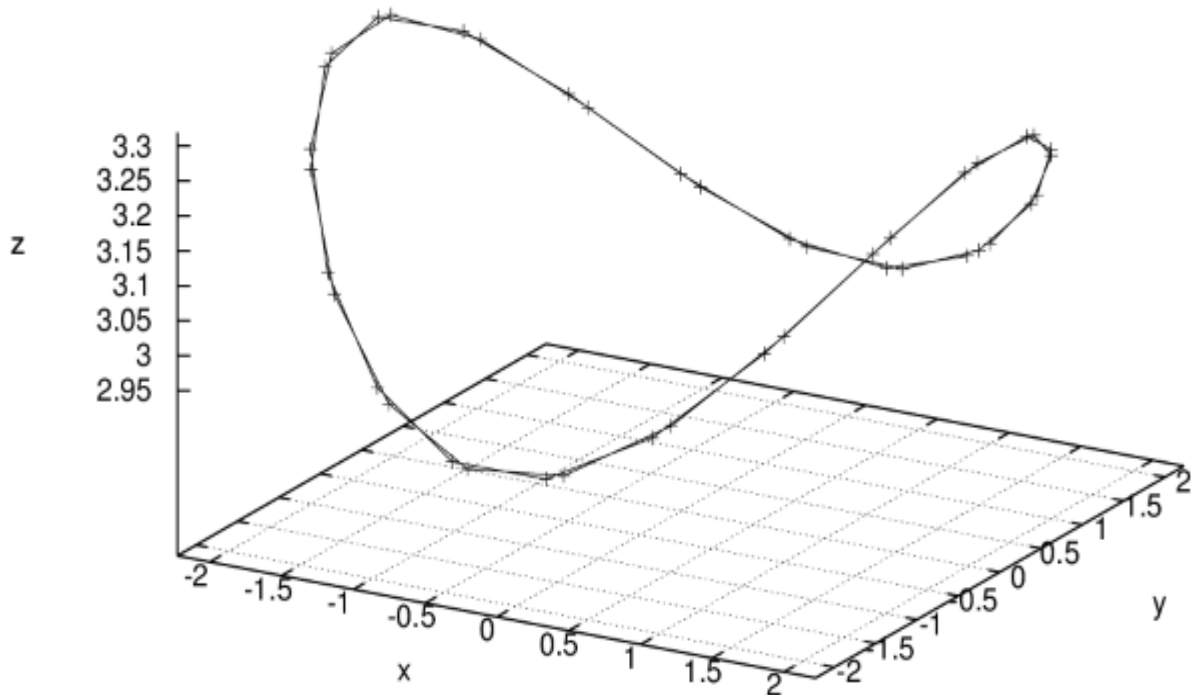


Рис. 6.13. Фазовый портрет трехмерной динамической системы

6.3.5 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:

```
(%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
```

Выделяем компоненты решения и строим графические иллюстрации:

```
(%i10) t: makelist(sol[k][1], k, 1, len) $
x: makelist(sol[k][2], k, 1, len) $
y: makelist(sol[k][3], k, 1, len) $
z: makelist(sol[k][4], k, 1, len) $
plot2d([discrete, t, x]) $ plot2d([discrete, t, y]) $
```

Результаты решения (хаотические колебания x, y, z) представлен на рис.6.14 и рис.6.15 (фазовый портрет системы). На рисунках объединены в одних осях кривые $x(t), y(t), z(t)$.

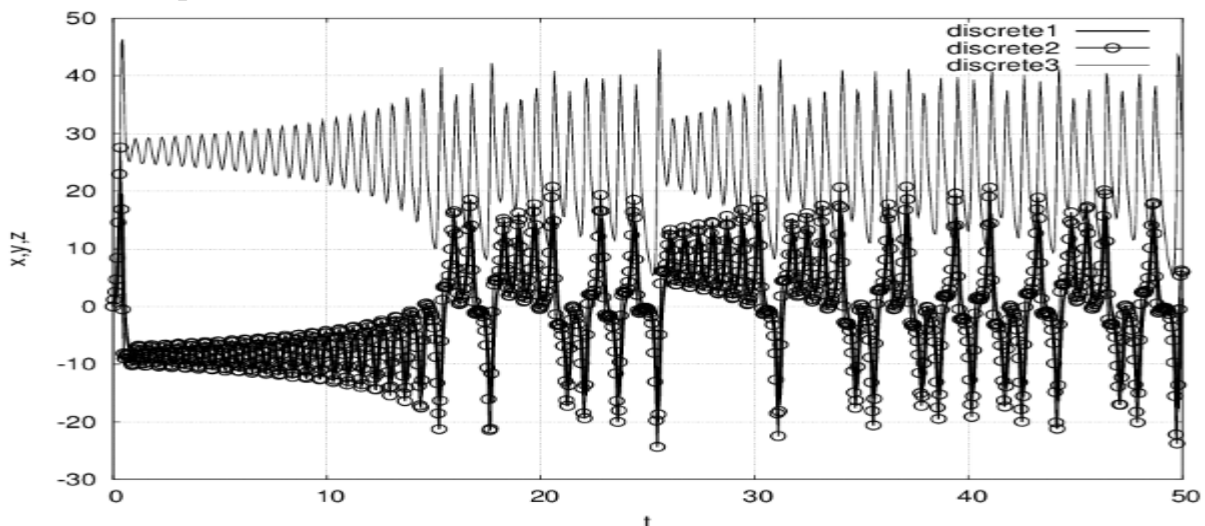


Рис. 6.14. Пример формирования динамического хаоса (аттрактор Лоренца)

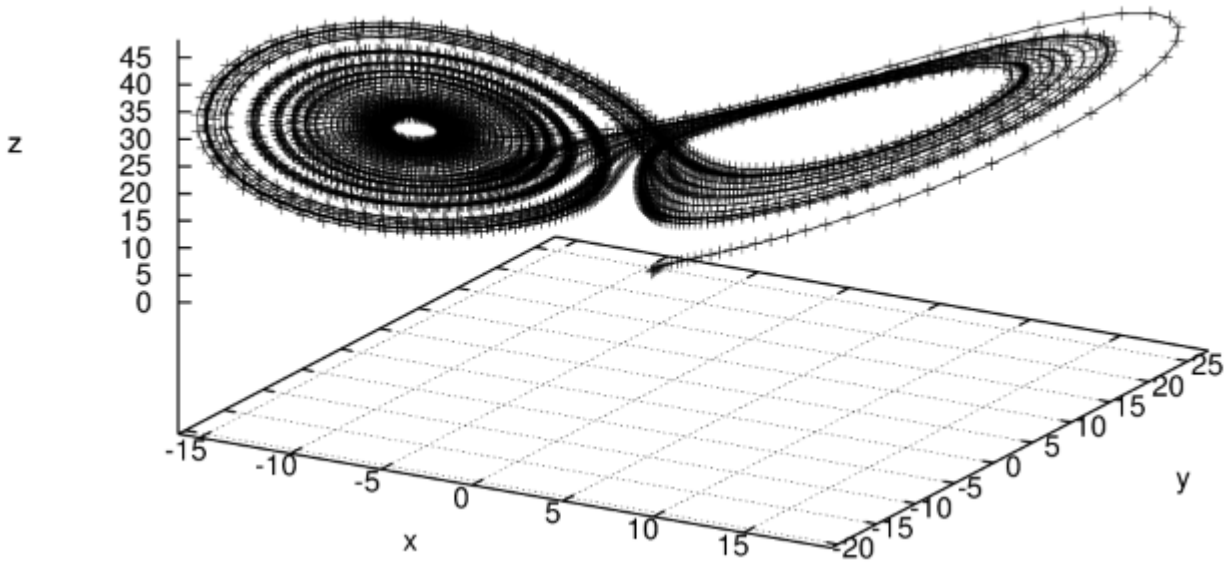


Рис. 6.15. Трехмерный фазовый портрет (аттрактор Лоренца)

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 значениях параметра r . 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.

6.3.6 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. Незвестная функция времени $y(t)$ имеет смысл электрического тока, а в параметре μ заложены количественные соотношения между составляющими электрической цепи, в том числе и нелинейной компонентой сопротивления:

$$\frac{d^2 y(t)}{dt^2} - \mu(1 - y(t)^2) \frac{dy(t)}{dt} + y(t) = 0$$

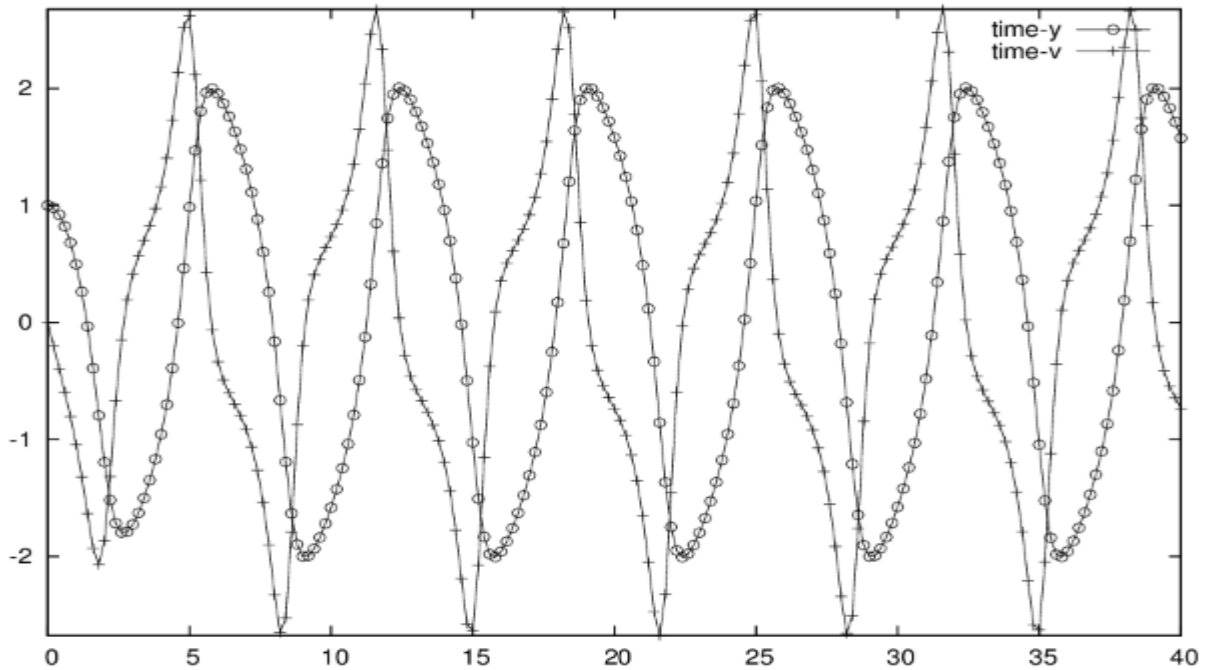


Рис. 6.16. Решение уравнения Ван дер Поля

Решением уравнения Ван дер Поля являются колебания, вид которых для $\mu = 1$ показан на рис.6.16. 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.6.17), так и снаружи предельного цикла.

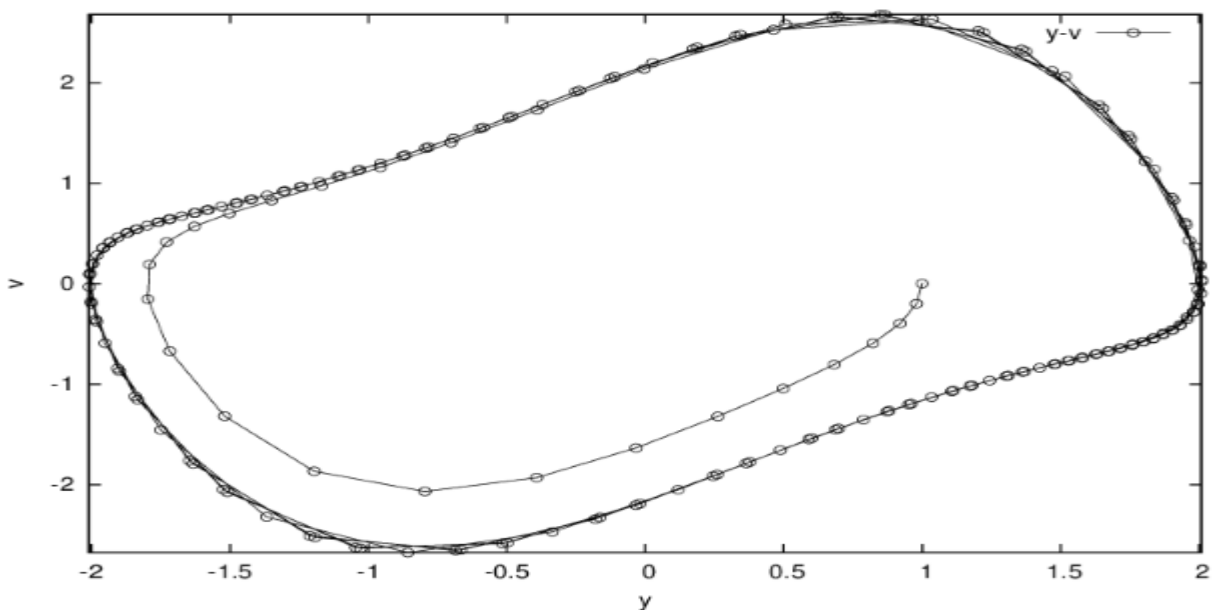


Рис. 6.17. Фазовый портрет уравнения Ван дер Поля

Использованный командный файл **Maxima** (для построения графической иллюстрации использован пакет `draw`):

```
load("dynamics")$ load("draw")$
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))$
draw2d(points_joined = true, point_type=6, key= "y-v",
        xlabel="y", ylabel="v", points(y,v), terminal=eps)$
```

7. Решение физических и математических задач с Maxima

7.1 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 последовательным использованием стандартного набора примитивов (формул суммы или разности кубов, формул возведения суммы или разности в степень и т.п.) выполнить сложно, поэтому результат являются фактически справочным, на который следует ориентироваться при решении вручную, при помощи ручки и бумаги.

7.1.1 Упрощение алгебраических выражений

Пример:

Упростить выражение и вычислить его, если даны числовые значения параметров:

```
(%i1) 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);
```

$$(\%o1) \frac{abc \left(\frac{1}{a} - \frac{1}{c+b} \right) \left(\frac{c^2+b^2-a^2}{2bc} + 1 \right)}{(-c-b+a) \left(\frac{1}{c+b} + \frac{1}{a} \right)}$$

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

$$(\%o2) -\frac{ac+ab-a^2}{2}$$

```
(%i3) %,a=0.02,b=-11.05,c=1.07;
```

```
(%o3) 0.1
```

Пример: 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)));
```

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

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

```
(%o2) x - 1
```

Пример: 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) 
$$\frac{x^3 - \frac{(b^2 + a^2)x}{a^{2/3}b} + \sqrt{b}}{b^{3/2}x^2}$$

```

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

```
(%o4) 
$$\frac{a^{2/3}bx^3 + (-b^2 - a^2)x + a^{2/3}b^{3/2}}{a^{2/3}b^{5/2}x^2}$$

```

```
(%i5) radcan(%);
```

```
(%o5) 
$$\frac{a^{2/3}bx^3 + (-b^2 - a^2)x + a^{2/3}b^{3/2}}{a^{2/3}b^{5/2}x^2}$$

```

Без указанной подстановки упрощение посредством комбинации функций *ratsimp* и *radcan* не удаётся.

```
(%i6) %, x=a^(2/3)*b^(-1/2);
```

```
(%o6) 
$$\frac{\frac{a^{2/3}(-b^2 - a^2)}{\sqrt{b}} + a^{2/3}b^{3/2} + \frac{a^{8/3}}{\sqrt{b}}}{a^2b^{3/2}}$$

```

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

Конечный результат оказывается простым

```
(%o7) 0
```

7.1.2 Разложение полиномов и рациональных выражений на множители

7.1.2.1 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. Сканава): 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{4x+13}+\sqrt{x+1}=\sqrt{3x+12}
(%i2) eq1:lhs(eq)^2=rhs(eq)^2;
(%o2) \left(\sqrt{4x+13}+\sqrt{x+1}\right)^2=3x+12
(%i3) solve([eq1],[x]);
(%o3) [x = -\sqrt{x+1}\sqrt{4x+13}-1]
(%i4) eq2:x+1=rhs(%[1])+1;
(%o4) x+1=-\sqrt{x+1}\sqrt{4x+13}
(%i5) eq3:lhs(eq2)^2=rhs(eq2)^2;
(%o5) (x+1)^2=(x+1)(4x+13)
```

Последняя команда позволила получить степенное уравнение, разрешимое аналитически в **Maxima** (для этого потребовалось дважды возвести в квадрат исходное 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 исходное 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) 
$$\sqrt{-4z + x + 3} + \sqrt{-6z + x + 8} = 1$$

```

```
(%i3) eq2:subst(z^2+1,x,eq1);
```

```
(%o3) 
$$\sqrt{z^2 - 4z + 4} + \sqrt{z^2 - 6z + 9} = 1$$

```

Упрощаем полученный результат:

```
(%i4) radcan(%);
```

```
(%o4) 
$$2z - 5 = 1$$

```

```
(%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]);
```

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

```
(%o1) [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).

7.2 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 в программном обеспечении.

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.

7.2.1 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. Используемая формула: $v = \sqrt{\frac{3RT}{M}}$, где $R = 8.314 \text{ Дж}/(\text{моль} \cdot \text{К})$, T — абсолютная температура, M — молярная масса.

Вычислим среднеквадратичную скорость молекул CO_2 ($M = 0.044 \text{ кг/моль}$) при температуре 273 К:

```
(%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
```

7.2.2 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)  $\frac{2v^2 \left(\frac{M}{RT}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{2RT}}}{\sqrt{2} \sqrt{\pi}}$ 
```

Для анализа полученных выражений в формулу распределения Максвелла подставляем только температуру:

```
(%i2) fun70:fun,T=70;
```

```
(%o2) 
$$\frac{v^2 \left(\frac{M}{R}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{140 R}}}{35 \sqrt{2} \sqrt{70} \sqrt{\pi}}$$

(%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) 
$$\frac{v^2 \left(\frac{M}{R}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{600 R}}}{1500 \sqrt{2} \sqrt{3} \sqrt{\pi}}$$

```

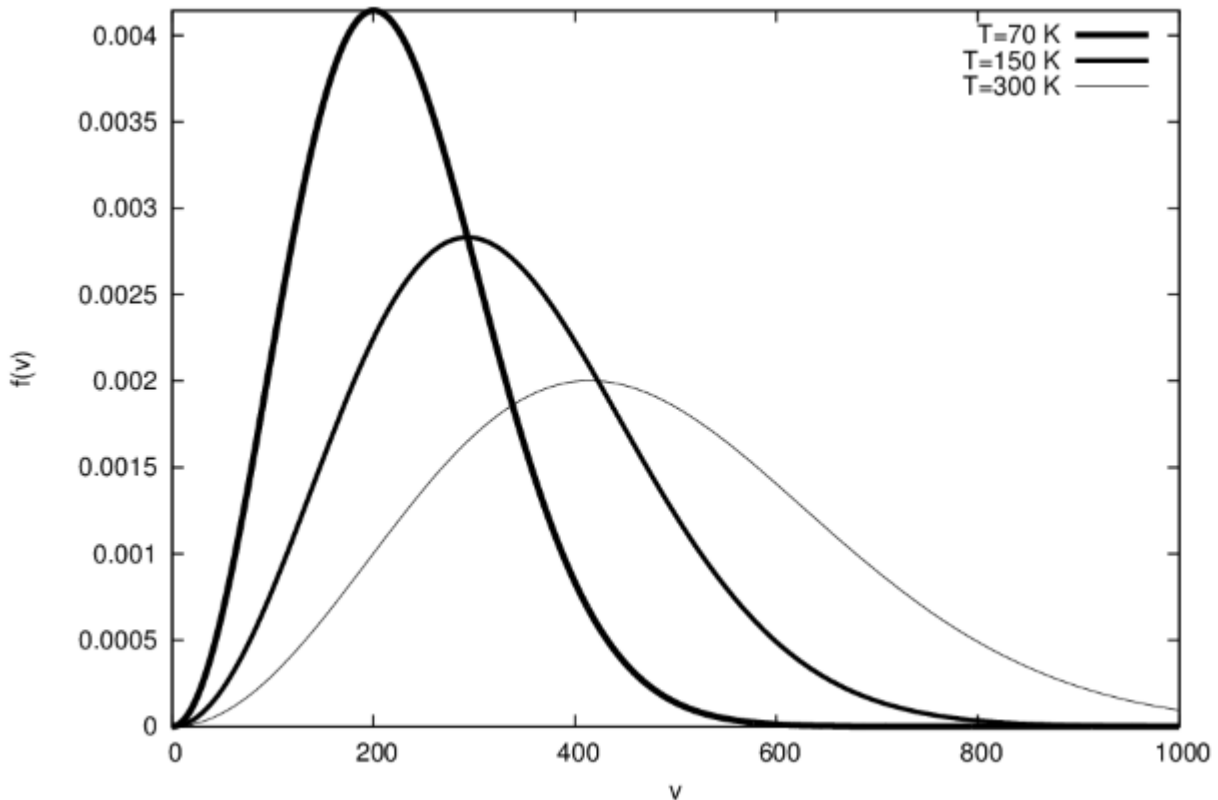


Рис. 7.1. Распределение Максвелла по скоростям молекул воздуха для различных температур

Для построения графика зависимости функции распределения от температуры подставляем молярную массу воздуха и величину универсальной газовой постоянной (см. результаты на рис. 7.1):

```
(%i5) plot2d([fun70, fun150, fun300], [v, 0, 1000]),
M=0.029, R=8.314;
```

Можно изучить влияние температуры на форму кривой, а так-же 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);
 Is M\R\,T positive, negative, or zero?
 p;

$$(\%o6) \frac{3RT}{M}$$

Таким образом, $\langle v^2 \rangle = \frac{3RT}{M}$, откуда среднеквадратичная скорость молекул
 газа $\langle \sqrt{v^2} \rangle = v_{sq} = \sqrt{\frac{3RT}{M}}$.

7.2.3 Броуновское движение

Наличие генератора случайных чисел дает возможность моделировать движение броуновской частицы.

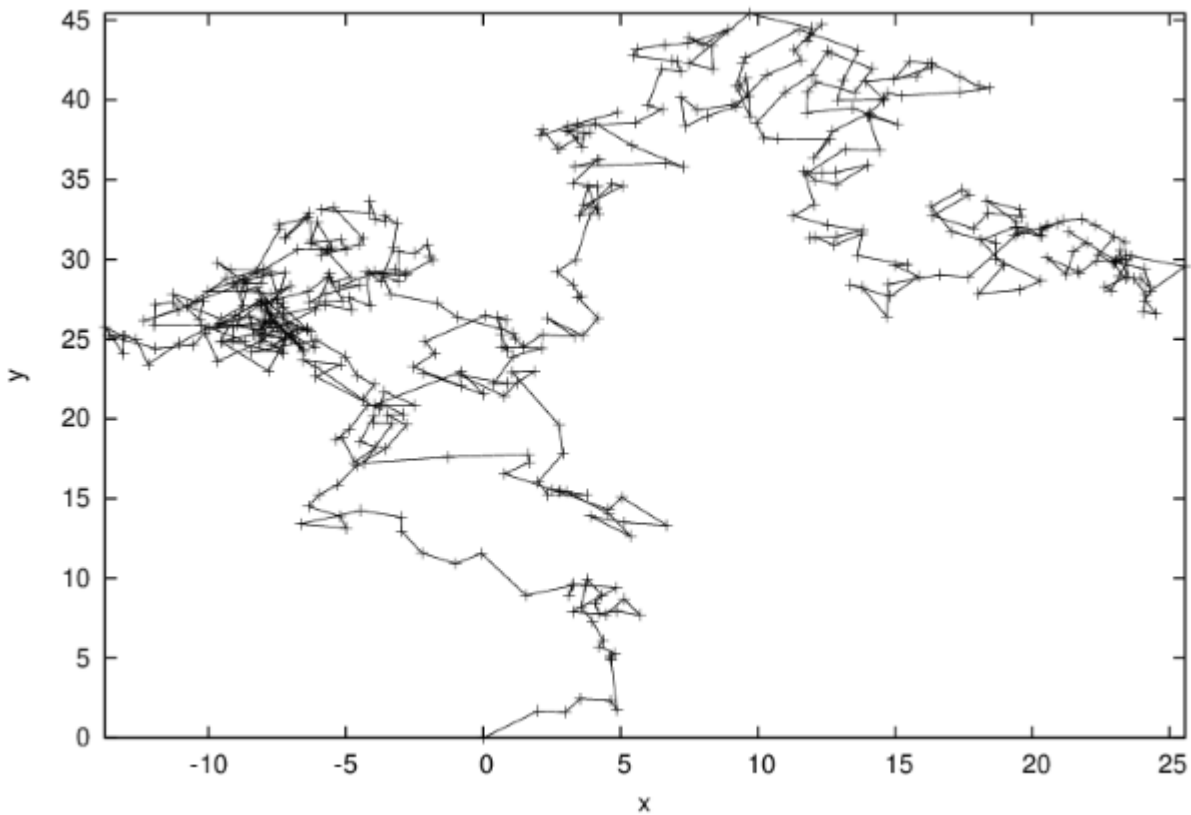


Рис. 7.2. Траектория броуновского движения модельной частицы

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 : $s^2 = \frac{2RTt}{6\pi hr N_A}$, 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")$
      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]]))$
      plot2d([discrete,xy]);
```

Результат построения графика приведен на рис. 7.2.

7.3 Пример построения статистической модели

Рассмотрим построение задачи с практическим содержанием.

В табл. 7.1 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**.

Таблица 7.1. Стоимость подержанного автомобиля в зависимости от его возраста и пробега

Возраст (лет)	Пробег (миль)	Цена (\$)
11.5	88000	1195
10.5	82000	1295
12.5	97000	800
8.5	51000	2295
9.5	79000	1995
13.5	120000	495
3.5	39000	4995
6.5	52000	2695
4.5	39000	3995
12.5	92000	795
7.5	41000	3495
10.5	77000	1595
12.5	83000	895
4.5	38000	3990
13.5	92000	795
13.5	103000	750
10.5	65000	1495
10.5	70000	1495

Возраст (лет)	Пробег (миль)	Цена (\$)
10.5	80000	1495
6.5	57000	2695
11.5	101000	895
10.5	78000	1295
9.5	84000	1995
4.5	46000	3675
11.5	108000	975
13.5	124000	850
6.5	56000	3495
9.5	67000	2495
6.5	43000	3400
11.5	78000	1295

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")$
(%i2) load("numericalio")$
(%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);
```

```
(%o22) (
    SIMPLE LINEAR REGRESSION
    model = 5757.594446543255 - 392.7181715149224 x
    correlation = -.9688177942467208
    v_estimation = 95364.34912839333
    b_conf_int = [-431.5987157329751, -353.8376272968697]
    hypotheses = H0 : b = 0, H1 : b # 0
    statistic = 20.69021212080514
    distribution = [student_t, 28]
    p_value = 0.0
)
```

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) SIMPLE LINEAR REGRESSION
      model = 9.174960600286802 - 2.3747715120748164 10-5 x
      correlation = -.9301125564244438
      v_estimation = .05467789749118319
      b_conf_int = [-2.7377780810631264 10-5, -2.0117649430865062 10-5]
      hypotheses = H0 : b = 0, H1 : b#0
      statistic = 13.40058098403749
      distribution = [student_t, 28]
      p_value = 5.928590951498336 10-14
```

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. 7.3 и рис. 7.4):

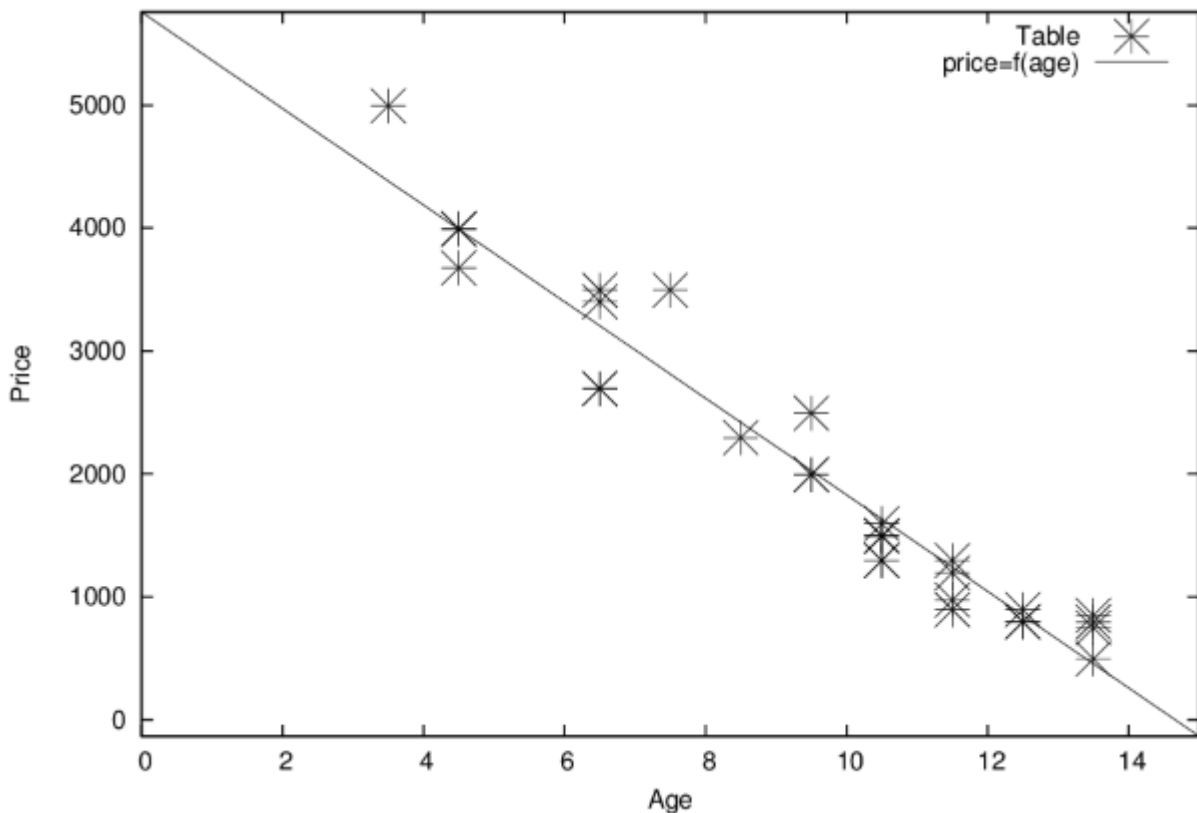


Рис. 7.3. Зависимость цены подержанного автомобиля от его возраста

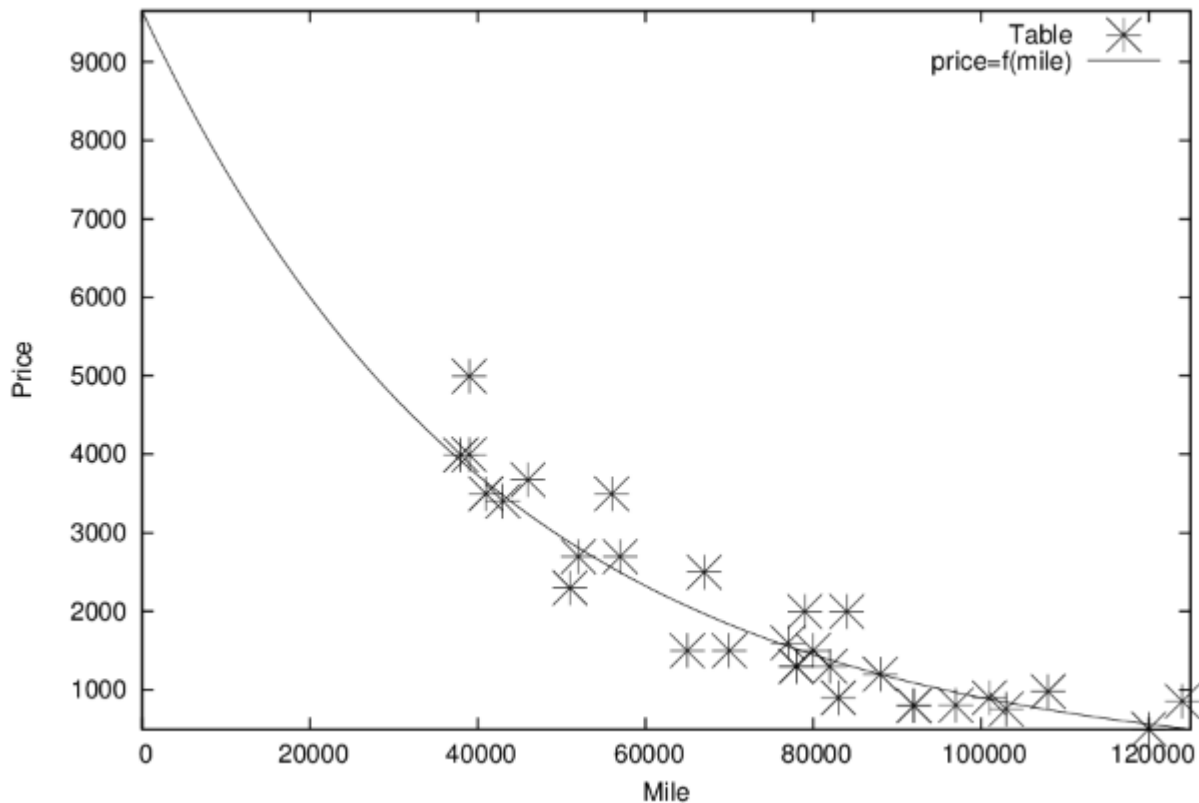


Рис. 7.4. Зависимость цены подержанного автомобиля от его пробега

```
(%i34) draw2d (terminal=eps, key="Table",
              xlabel="Age", ylabel="Price",
              point_size = 3, point_type=3, points (age, price),
              key="price=f (age) ", explicit (fun1, x, 0, 15));
(%i41) draw2d (terminal=eps, key="Table",
              xlabel="Mile", ylabel="Price",
              point_size = 3, point_type=3, points (mile, price),
              key="price=f (mile) ", explicit (fun2, x, 0, 125000));
```

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. Графическая иллюстрация представлена на рис. 7.5.

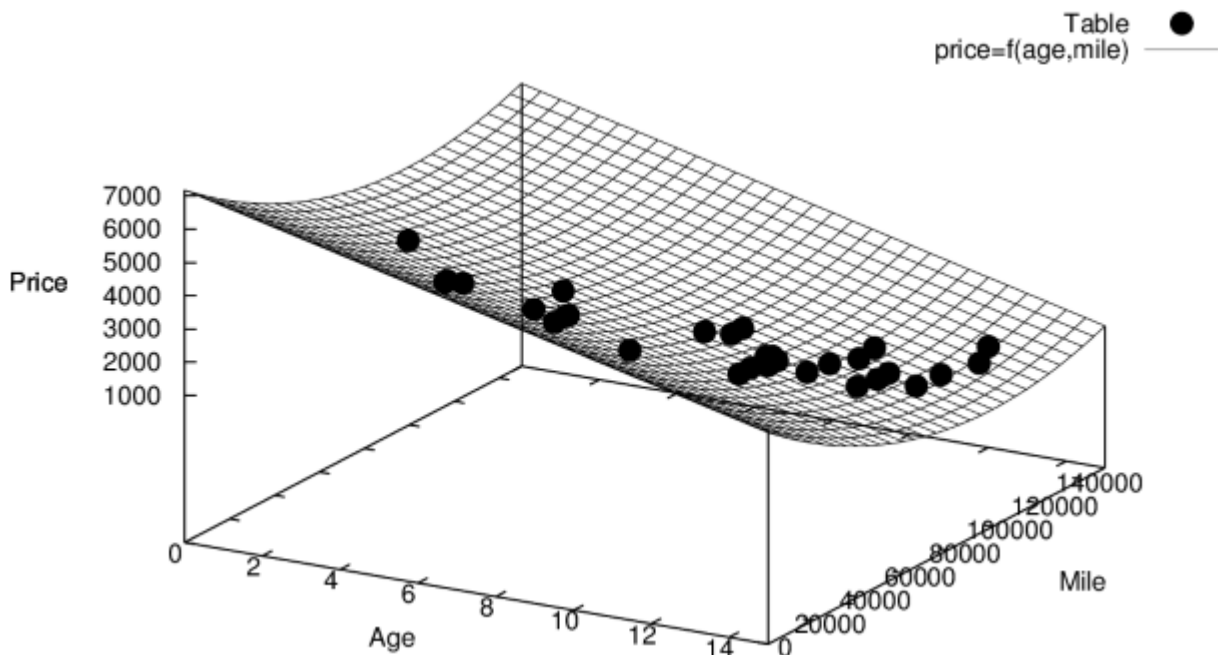


Рис. 7.5. Иллюстрация зависимости отклика (цены подержанного автомобиля) от двух независимых факторов (возраста и пробега автомобиля)

8. Реализация некоторых численных методов

8.1 Программирование методов решения нелинейных уравнений в Maxima

Необходимость отыскания корней нелинейных уравнений встречается в целом ряде задач: расчетах систем автоматического управления и регулирования, собственных колебаний машин и конструкций, в задачах кинематического анализа и синтеза, плоских и пространственных механизмов и других задачах.

Пусть дано нелинейное уравнение $f(x) = 0$, и необходимо решить это уравнение, т. е. найти его корень \bar{x} .

Если функция имеет вид многочлена степени

$$m, f(x) = a_0x^m + a_1x^{m-1} + a_2x^{m-2} + \dots + a_{m-1}x + a_m,$$

где a_i — коэффициенты многочлена, $i = \overline{1, m}$, то уравнение $f(x) = 0$ имеет m корней (основная теорема алгебры).

Если функция $f(x)$ включает в себя тригонометрические или экспоненциальные функции от некоторого аргумента x , то уравнение

$f(x) = 0$ называется трансцендентным уравнением. Такие уравнения обычно имеют бесконечное множество решений.

Как известно, не всякое уравнение может быть решено точно. В первую очередь это относится к большинству трансцендентных уравнений.

Доказано также, что нельзя построить формулу, по которой можно было бы решать произвольные алгебраические уравнения степени, выше четвертой.

Однако точное решение уравнения не всегда является необходимым. Задачу отыскания корней уравнения можно считать практически решенной, если мы сумеем найти корни уравнения с заданной степенью точности. Для этого используются приближенные (численные) методы решения.

Большинство употребляющихся приближенных методов решения уравнений являются, по существу, способами уточнения корней. Для их применения необходимо знание интервала изоляции $[a, b]$, в котором лежит уточняемый корень уравнения.

Процесс определения интервала изоляции $[a, b]$, содержащего только один из корней уравнения, называется отделением этого корня.

Процесс отделения корней проводят исходя из физического смысла прикладной задачи, графически, с помощью таблиц значений функции $f(x)$ или при помощи специальной программы отделения корней. Процедура отделения корней основана на известном свойстве непрерывных функций: если функция непрерывна на замкнутом интервале $[a, b]$ и на его концах имеет различные знаки, т.е. $f(a) \cdot f(b) < 0$, то между точками a и b имеется хотя бы один корень уравнения $f(x) = 0$. Если при этом функция $f(x)$ на отрезке $[a, b]$ монотонна, то указанный корень единственный.

Процесс определения корней алгебраических и трансцендентных уравнений состоит из двух этапов:

1. отделение корней, — т.е. определение интервалов изоляции $[a, b]$, внутри которого лежит каждый корень уравнения;
2. уточнение корней, — т.е. сужение интервала $[a, b]$ до величины равной заданной степени точности ε .

Для алгебраических и трансцендентных уравнений пригодны одни и те же методы уточнения приближенных значений действительных корней:

- метод половинного деления (метод дихотомии);
- метод простых итераций;
- метод Ньютона (метод касательных);
- модифицированный метод Ньютона (метод секущих);
- метод хорд и др.

8.1.1 Метод половинного деления

Рассмотрим следующую задачу: дано нелинейное уравнение $f(x) = 0$, необходимо найти корень уравнения, принадлежащий интервалу $[a, b]$, с заданной точностью ε .

Для уточнения корня методом половинного деления последовательно осуществляем следующие операции:

- вычисляем значение функции $f(x)$ в точках a и $t = (a + b)/2$;
- если $f(a) \cdot f(t) < 0$, то корень находится в левой половине интервала $[a, b]$, поэтому отбрасываем правую половину интервала и принимаем $b = t$;
- если условие $f(a) \cdot f(t) < 0$ не выполняется, то корень находится в правой половине интервала $[a, b]$, поэтому отбрасываем левую половину интервала $[a, b]$ за счёт присваивания $a = t$.

В обоих случаях новый интервал $[a, b]$ в 2 раза меньше предыдущего.

Процесс сокращения длины интервала неопределённости циклически повторяется до тех пор, пока длина интервала $[a, b]$ не станет равной либо меньшей заданной точности, т.е. $|b - a| \leq \varepsilon$.

Реализация метода половинного деления в виде функции **Maxima** представлена в следующем примере:

собственно функция `bisect`, в которую передаётся выражение f , определяющее уравнение, которое необходимо решить

```
(%i1)
bisect(f, sp, eps) := block([a, b], a: sp[1], b: sp[2], p: 0,
  while abs(b-a) > eps do (
    p: p+1, c: (a+b)/2,
    fa: float(subst(a, x, f)), fc: float(subst(c, x, f)),
    if fa*fc < 0 then b: c else a: c
  ),
float(c));

(%o1) bisect(f, sp, eps) := block([a, b], a: sp_1, b: sp_2, p:
0,
while |b - a| > eps do(p: p+1, c: (a+b)/2, fa: float(subst(a, x, f)),
fc: float(subst(c, x, f)), if fa*fc < 0 then b: c else a:
c), float(c))
```

последовательность команд, организующая обращение к `bisect` и результаты вычислений

```
(%i2) f: exp(-x) - x$
```



```
a:-1$ b:2$ eps:0.000001$ xrez:bisect(f,[-2,2],eps)$
print("Решение ",xrez," Невязка ",subst(xrez,x,f))$
Решение 0.56714344024658 Невязка - 2.34815726529724610-7
(%o2) - 2.348157265297246 10-7
```

В представленном примере решается уравнение $e^{-x} - x = 0$. Поиск корня осуществляется на отрезке $[-2, 2]$ с точностью 0.000001.

Следует отметить особенность программирования для **Maxima**, заключающуюся в том, что решаемое уравнение задаётся в виде математического выражения (т.е. фактически текстовой строки). Числовое значение невязки уравнения вычисляется при помощи функции `subst`, посредством которой выполняется подстановка значений $x = a$ или $x = c$ в заданное выражение. Вычисление невязки после решения осуществляется также путём подстановки результата решения `xrez` в выражение `f`.

8.1.2 Метод простых итераций

В ряде случаев весьма удобным приёмом уточнения корня уравнения является метод последовательных приближений (метод итераций).

Пусть с точностью ε необходимо найти корень уравнения $f(x) = 0$, принадлежащий интервалу изоляции $[a, b]$. Функция $f(x)$ и ее первая производная непрерывны на этом отрезке.

Для применения этого метода исходное уравнение $f(x) = 0$ должно быть приведено к виду $x = \varphi(x)$.

В качестве начального приближения может быть выбрана любая точка интервала $[a, b]$.

Далее итерационный процесс поиска корня строится по схеме:

$$x_1 = f(x_0),$$

$$x_2 = f(x_1),$$

...

$$x_n = f(x_{n-1})$$

В результате итерационный процесс поиска реализуется рекуррентной формулой. Процесс поиска прекращается, как только выполняется условие $|x_n - x_{n-1}| \leq \varepsilon$ или число итераций превысит заданное число N .

Для того, чтобы последовательность x_1, x_2, \dots, x_n приближалась к искомому корню, необходимо, чтобы выполнялось условие сходимости $|\varphi'(x)| < 1$.

Пример реализации метода итераций представлен ниже:

```
(%i1) f:exp(-x)-x$
beta:0.1$ x1:1$ x0:0$ eps:0.000001$ p:0$
while abs(x1-x0)>eps do
  (x0:x1, p:p+1,
  x1:float(x0+beta*(subst(x0,x,f))))$
```

```
print("Число итераций ", p, " ", "Решение ", float(x1),
      " Невязка ", float(abs(x1-x0))) $
Число итераций 67 Решение 0.56714848327814
Невязка 9.650298036234517107
```

8.1.3 Метод Ньютона (метод касательных)

Рассмотренные ранее методы решения нелинейных уравнений являются методами прямого поиска. В них для нахождения корня используется нахождение значения функции в различных точках интервала $[a, b]$.

Метод Ньютона относится к градиентным методам, в которых для нахождения корня используется значение производной.

Рассмотрим нелинейное уравнение $f(x) = 0$, для которого необходимо найти корень на интервале $[a, b]$ с точностью ε .

Метод Ньютона основан на замене исходной функции $f(x)$, на каждом шаге поиска касательной, проведённой к этой функции. Пересечение касательной с осью X даёт приближение корня.

Выберем начальную точку $x_0 = b$ (конец интервала изоляции). Находим значение функции в этой точке и проводим к ней касательную, пересечение которой с осью X даёт первое приближение корня x_1 :

$$x_1 = x_0 - h_0, \quad \text{где}$$

$$h_0 = \frac{f(x_0)}{\operatorname{tg}(\alpha)} = \frac{f(x_0)}{f'(x_0)}.$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Поэтому

В результате, итерационный процесс схождения к корню реализуется рекуррентной формулой

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Процесс поиска продолжаем до тех пор, пока не выполнится условие:

$$|x_{n+1} - x_n| \leq \varepsilon, \quad \text{откуда} \quad \left| \frac{f(x_n)}{f'(x_n)} \right| \leq \varepsilon.$$

Метод обеспечивает быструю сходимость, если выполняется условие: $f(x_0) \cdot f''(x_0) > 0$, т.е. первую касательную рекомендуется проводить в той точке интервала $[a, b]$, где знаки функции $f(x_0)$ и ее кривизны $f''(x_0)$ совпадают.

Пример реализации метода Ньютона в **Maxima** представлен ниже:

```
(%i1) newton(f, x0, eps) := block([df, xn, xn0, r, p],
  xn0:x0, df:diff(f, x),
  p:0, r:1,
  while abs(r)>eps do (
```

```

    p:p+1, xn:xn0-
float(subst(xn0,x,f)/subst(xn0,x,df)),
    print("x0,x1 ",xn0,xn),r:xn-xn0, xn0:xn
),
[xn,p])$

```

Последовательность команд для обращения к функции *newton* и результаты вычислений представлены в следующем примере:

```

(%i2) f:exp(-x)-x$
eps:0.000001$ xrez:newton(f,1,eps)$
print("Решение ",xrez[1]," Число итераций ",xrez[2],
" Невязка ",subst(xrez[1],x,f))$
x0,x1 1.53788284273999
x0,x1 0.53788284273999 0.56698699140541
x0,x1 0.56698699140541 0.56714328598912
x0,x1 0.56714328598912 0.56714329040978
Решение 0.56714329040978 Число итераций 4 Невязка 0.0

```

Особенности приведённого примера — промежуточная печать результатов и возвращаемое значение в виде списка, что позволяет одновременно получить как значение корня, так и необходимое для достижения заданной точности число итераций. Существенному уменьшению числа итераций способствует и аналитическое вычисление производной.

8.1.4 Модифицированный метод Ньютона (метод секущих)

В этом методе для вычисления производных на каждом шаге поиска используется численное дифференцирование по формуле:

$$f'(x) \approx \frac{\Delta f(x)}{\Delta x}$$

Тогда рекуррентная формула метода Ньютона приобретёт вид:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{f(x_n)\Delta x}{\Delta f(x_n)} =$$

$$= x_n - \frac{f(x_n)\Delta x}{f(x_n+\Delta x) - f(x_n)},$$

где $\Delta x \approx \varepsilon$.

Реализация данного метода в **Maxima** представлено ниже:

```

(%i1) secant(f, sp, eps) := block([x0,x1,d,y,r],
    x0:sp[1],x1:sp[2],
    p:0, r:x1-x0, d:float(subst(x0,x,f)),
    while abs(r)>eps do (
        p:p+1, y:float(subst(x1,x,f)), r:r/(d-y)*y,
        d:y, x1:x1+r
    ),
    x1)$
(%i2) f:exp(-x)-x$
eps:0.000001$ xrez:secant(f,[-2,2],eps)$
print("Решение ",xrez," Невязка ",subst(xrez,x,f))$

```

Решение 0.56714329040978 Невязка $-1.1102230246251565 \cdot 10^{-16}$

Особенности программирования для **Maxima**, использованные в этом примере, аналогичны приведённым выше в примере, касающемся метода половинного деления.

8.1.5 Метод хорд

Метод основан на замене функции $f(x)$ на каждом шаге поиска хордой, пересечение которой с осью X дает приближение корня.

При этом в процессе поиска семейство хорд может строиться:

- при фиксированном левом конце хорд, т.е. $z = a$, тогда начальная точка $x_0 = b$;
- при фиксированном правом конце хорд, т.е. $z = b$, тогда начальная точка $x_0 = a$.

В результате итерационный процесс схождения к корню реализуется рекуррентной формулой:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(a)}(x_n - a) \quad \text{для случая а);}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(b)}(x_n - b) \quad \text{для случая б);}$$

Процесс поиска продолжается до тех пор, пока не выполнится условие $|x_{n+1} - x_n| \leq \varepsilon$ или $|h| \leq \varepsilon$.

Метод обеспечивает быструю сходимость, если $f(z) \cdot f''(z) > 0$, т.е. хорды фиксируются в том конце интервала $[a, b]$, где знаки функции $f(z)$ и ее кривизны $f''(z)$ совпадают.

8.2 Численное интегрирование

Задача численного интегрирования состоит в замене исходной подинтегральной функции $f(x)$, для которой трудно или невозможно записать первообразную в аналитике, некоторой аппроксимирующей функцией $\varphi(x)$. Такой функцией обычно является полином (кусочный полином)

$$\varphi(x) = \sum_{i=1}^n c_i \phi_i(x)$$

Таким образом

$$I = \int_a^b f(x) dx = \int_a^b \phi(x) dx + R,$$

$$R = \int_a^b r(x) dx$$

где R — априорная погрешность метода на интервале интегрирования, а $r(x)$ — априорная погрешность метода на отдельном шаге интегрирования.

8.2.1 Обзор методов интегрирования

Методы вычисления однократных интегралов называются квадратурными (для кратных интегралов — кубатурными), и делятся на следующие группы:

- Методы Ньютона-Котеса. Здесь $\varphi(x)$ — полином различных степеней. Сюда относятся метод прямоугольников, трапеций, Симпсона.
- Методы статистических испытаний (методы Монте-Карло). Здесь узлы сетки для квадратурного или кубатурного интегрирования выбираются с помощью датчика случайных чисел, ответ носит вероятностный характер. В основном применяются для вычисления кратных интегралов.
- Сплайновые методы. Здесь $\varphi(x)$ — кусочный полином с условиями связи между отдельными полиномами посредством системы коэффициентов.
- Методы наивысшей алгебраической точности. Обеспечивают оптимальную расстановку узлов сетки интегрирования и выбор

весовых коэффициентов $\rho(x)$ в задаче $\int_a^b \phi(x)\rho(x)dx$ (характерный пример — метод Гаусса).

8.2.2 Метод прямоугольников

Различают метод левых, правых и средних прямоугольников. Суть метода ясна из рисунка. На каждом шаге интегрирования функция аппроксимируется полиномом нулевой степени — отрезком, параллельным оси абсцисс.

Формулы метода прямоугольников можно получить из анализа разложения функции $f(x)$ в ряд Тейлора вблизи некоторой точки $x = x_i$:

$$f(x)|_{x=x_i} = f(x_i) + (x - x_i) f'(x_i) + \frac{(x - x_i)^2}{2!} f''(x_i) + \dots$$

Рассмотрим диапазон интегрирования от x_i до $x_i + h$, где h — шаг интегрирования. Вычислим интеграл от исследуемой функции на этом промежутке:

$$\begin{aligned} \int_{x_i}^{x_i+h} f(x)dx &= x \cdot f(x_i)|_{x_i}^{x_i+h} + \frac{(x-x_i)^2}{2} f'(x_i)|_{x_i}^{x_i+h} + \\ &+ \frac{(x-x_i)^3}{3 \cdot 2!} f''(x_i)|_{x_i}^{x_i+h} + \dots = \\ &= f(x_i)h + \frac{h^2}{2} f'(x_i) + O(h^3) = f(x_i)h + r_i. \end{aligned}$$

таким образом, на базе анализа ряда Тейлора получена формула правых (или левых) прямоугольников и априорная оценка погрешности T на отдельном шаге интегрирования. Основным критерий, по которому судят о точности алгоритма — степень при величине шага в формуле априорной оценки погрешности. В случае равного шага h на всем диапазоне интегрирования общая формула имеет вид

$$\int_a^b f(x)dx = h \sum_{i=0}^{n-1} f(x_i) + R,$$

где n — число разбиений интервала интегрирования,
 $R = \sum_{i=0}^{n-1} r_i = \frac{h}{2} \cdot h \sum_{i=0}^{n-1} f'(x_i) = \frac{h}{2} \int_a^b f'(x)dx$.
 Полученная оценка справедлива при наличии непрерывной производной подинтегральной функции $f'(x)$.

8.2.3 Метод средних прямоугольников

Здесь на каждом интервале значение функции считается в средней точке

$$\int_{x_i}^{x_i+h} f(x)dx = hf(\bar{x}) + r_i$$

отрезка $[x_i, x_i + h]$, то есть x_i .

Разложение функции в ряд Тейлора показывает, что в случае средних прямоугольников точность метода существенно выше:

$$r = \frac{h^3}{24} f''(\bar{x}), \quad R = \frac{h^2}{24} \int_a^b f''(x)dx.$$

Пример функции **Maxima**, реализующей метод средних прямоугольников, представлен ниже:

```
(%i1)  intpr(f,n,a,b):=block([h,i,s,_x],h:(b-a)/n,
_x:a+h/2, s:0,
  for i:1 thru n do
(s:s+float(subst(_x,x,f)),_x:_x+h),s:s*h)$
(%i2)  intpr(x^2,100,-1,1);
(%o2)  0.6666
```

8.2.4 Метод трапеций

Аппроксимация в этом методе осуществляется полиномом первой степени.

На единичном интервале

$$\int_{x_i}^{x_i+h} f(x)dx = \frac{h}{2} (f(x_i) + f(x_i + h)) + r_i.$$

В случае равномерной сетки ($h = \text{const}$)

$$\int_a^b f(x)dx = h \left(\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n) \right) + R$$

$$\text{При этом } r_i = -\frac{h^3}{12}f''(x_i), \quad R = -\frac{h^3}{12} \int_a^b f''(x)dx.$$

Погрешность метода трапеций в два раза выше, чем у метода средних прямоугольников. Однако на практике найти среднее значение на элементарном интервале можно только у функций, заданных аналитически (а не таблично), поэтому использовать метод средних прямоугольников удаётся далеко не всегда. В силу разных знаков погрешности в формулах трапеций и средних прямоугольников истинное значение интеграла обычно лежит между двумя этими оценками.

Пример реализации метода трапеций в виде функции приведен ниже:

```
(%i1) f:x^2;
(%o1) x^2
(%i2) inttrap(f,n,a,b):=block([h,i,s],h:(b-a)/n,
s:(float(subst(a,x,f))+float(subst(b,x,f)))/2,
for i:1 thru n-1 do (s:s+float(subst(a+i*h,x,f))),
s:s*h)$
(%i3) inttrap(f,100,-1,1);
(%o3) 0.6668
```

Подинтегральная функция задаётся в виде выражения **Maxima**. Выражение, определяющее подинтегральную функцию, можно задавать и непосредственно при обращении к методу, как в следующем примере:

```
(%i4) inttrap(x*sin(x),100,-1,1);
(%o4) 0.60242947746101
```

8.2.5 Метод Симпсона

При использовании данного метода подинтегральная функция $f(x)$ заменяется интерполяционным полиномом второй степени $P(x)$ — параболой, проходящей через три соседних узла. Рассмотрим два шага интегрирования ($h = \text{const} = x_{i+1} - x_i$), то есть три узла x_0, x_1, x_2 , через которые проведем параболу, воспользовавшись уравнением Ньютона:

$$P(x) = f_0 + \frac{x - x_0}{h} (f_1 - f_0) + \frac{(x - x_0)(x - x_1)}{2h^2} (f_0 - 2f_1 + f_2).$$

Пусть $z = x - x_0$, тогда

$$\begin{aligned} P(z) &= f_0 + \frac{z}{h} (f_1 - f_0) + \frac{z(z-h)}{2h^2} (f_0 - 2f_1 + f_2) = \\ &= f_0 + \frac{z}{2h} (-3f_0 + 4f_1 - f_2) + \frac{z^2}{2h^2} (f_0 - 2f_1 + f_2) \end{aligned}$$

Воспользовавшись полученным соотношением, вычислим интеграл по данному интервалу:

$$\begin{aligned} \int_{x_0}^{x_2} P(x) dx &= \int_0^{2h} P(z) dz = \\ &= 2hf_0 + \frac{(2h)^2}{4h} (-3f_0 + 4f_1 - f_2) + \frac{(2h)^3}{6h^2} (f_0 - 2f_1 + f_2) = \\ &= 2hf_0 + h(-3f_0 + 4f_1 - f_2) + \frac{4h}{3} (f_0 - 2f_1 + f_2) = \\ &= \frac{h}{3} (6f_0 - 9f_0 + 12f_1 - 3f_2 + 4f_0 - 8f_1 + 4f_2). \end{aligned}$$

$$\int_{x_0}^{x_2} f(x) dx = \frac{h}{3} (f_0 + 4f_1 + f_2) + r$$

В итоге

Для равномерной сетки и чётного числа шагов n формула Симпсона принимает вид:

$$\int_a^b f(x) dx = \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + \dots + 2f_{n-2} + 4f_{n-1} + f_n) + R,$$

$$\text{где } r = -\frac{h^5}{90} f^{IV}(x_i), \quad R = -\frac{h^4}{180} \int_a^b f^{IV}(x) dx$$

в предположении непрерывности четвёртой производной подинтегральной функции.

Реализация метода Симпсона средствами **Maxima** представлена в следующем примере:

```
(%i1) intsimp(f,n,a,b):=block([h,h2,i,_x,s],h:(b-a)/n,
h2:h/2,
s:(float(subst(a,x,f))+float(subst(b,x,f)))/2+
2*float(subst(a+h2,x,f)),
_x:a,
for i:1 thru n-1 do (_x:_x+h,
s:s+2*float(subst(_x+h2,x,f))+float(subst(_x,x,f))),
s:s*h/3)$
(%i2) intsimp(x^2,100,-1,1);
(%o2) 0.6666666666666667
```


8.3 Методы решения систем линейных уравнений

8.3.1 Общая характеристика и классификация методов решения

Рассмотрим систему линейных алгебраических уравнений (сокращенно — СЛАУ):

$$A \cdot \bar{x} = \bar{f}, \quad (8.1)$$

где A — матрица $m \times m$, $\bar{x} = (x_1, x_2, \dots, x_m)^T$ — искомый вектор, $\bar{f} = (f_1, f_2, \dots, f_m)^T$ — заданный вектор. Будем предполагать, что определитель матрицы A отличен от нуля, т.е. решение системы (8.1) существует.

Методы численного решения системы (8.1) делятся на две группы: прямые методы ("точные") и итерационные методы.

Прямыми методами называются методы, позволяющие получить решение системы (8.1) за конечное число арифметических операций. К этим методам относятся метод Крамера, метод Гаусса, LU-метод и т.д.

Итерационные методы (методы последовательных приближений) состоят в том, что решение системы (8.1) находится как предел последовательных приближений $\bar{x}^{(n)}$ при $n \rightarrow \infty$, где n — номер итерации. При использовании методов итерации обычно задается некоторое малое число ε и вычисления проводятся до тех пор, пока не будет выполнена оценка $\|\bar{x}^{(n)} - \bar{x}\| < \varepsilon$. К этим методам относятся метод Зейделя, Якоби, метод верхних релаксаций и т.д.

Следует заметить, что реализация прямых методов на компьютере приводит к решению с погрешностью, т.к. все арифметические операции над переменными с плавающей точкой выполняются с округлением. В зависимости от свойств матрицы исходной системы эти погрешности могут достигать значительных величин.

8.3.2 Метод Гаусса

Запишем систему (8.1), в развернутом виде

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1m}x_m = f_1,$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2m}x_m = f_2,$$

.....

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mm}x_m = f_m.$$

Метод Гаусса состоит в последовательном исключении неизвестных из этой системы. Предположим, что $a_{11} \neq 0$. Последовательно умножая первое

$$-\frac{a_{i1}}{a_{11}}$$

уравнение на $\frac{a_{i1}}{a_{11}}$ и складывая с i -м уравнением, исключим x_1 из всех уравнений кроме первого. Получим систему

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1m}x_m = f_1,$$

$$a_{22}^{(1)}x_2 + \dots + a_{2m}^{(1)}x_m = f_2^{(1)},$$

.....

$$a_{m2}^{(1)}x_2 + \dots + a_{mm}^{(1)}x_m = f_m^{(1)}.$$

где

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}, \quad f_i^{(1)} = f_i - \frac{a_{i1}f_1}{a_{11}}, \quad i, j = 2, 3, \dots, m.$$

Аналогичным образом из полученной системы исключим x_2 . Последовательно, исключая все неизвестные, получим систему треугольного вида

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1k}x_k + \dots + a_{1m}x_m = f_1,$$

$$a_{22}^{(1)}x_2 + \dots + a_{2k}^{(1)}x_k + \dots + a_{2m}^{(1)}x_m = f_2^{(1)},$$

.....,

$$a_{m-1,m-1}^{(m-1)}x_{m-1} + a_{m-1,m}^{(m-1)}x_m = f_{m-1}^{(m-1)},$$

$$a_{m,m}^{(m-1)}x_m = f_m^{(m-1)}.$$

Описанная процедура называется прямым ходом метода Гаусса. Заметим, что ее выполнение было возможно при условии, что все $a_{i,i}^{(l)}$, $l = 1, 2, \dots, m-1$ не равны нулю. Выполняя последовательные подстановки в последней системе, (начиная с последнего уравнения) можно получить все значения неизвестных.

$$x_m = \frac{f_m^{(m-1)}}{a_{m,m}^{(m-1)}},$$

$$x_i = \frac{1}{a_{i,i}^{(i-1)}} \left(f_i^{(i-1)} - \sum_{j=i-1}^m a_{ij}^{(i-1)} x_j \right).$$

Эта процедура получила название обратный ход метода Гаусса.

Метод Гаусса может быть легко реализован на компьютере. При выполнении вычислений, как правило, промежуточные значения матрицы A не представляют интереса. Поэтому численная реализация метода сводится к преобразованию элементов массива размерности $m \times (m+1)$, где $m+1$ -й столбец содержит элементы правой части системы.

Для контроля ошибки реализации метода используются, так называемые, контрольные суммы. Схема контроля основывается на следующем очевидном положении. Увеличение значения всех неизвестных на единицу равносильно замене данной системы контрольной системой, в которой свободные члены равны суммам всех коэффициентов соответствующей строки. Создадим дополнительный столбец, хранящий сумму элементов матрицы по строкам. На каждом шаге реализации прямого хода метода Гаусса будем выполнять

преобразования и над элементами этого столбца, и сравнивать их значение с суммой по строке преобразованной матрицы. В случае не совпадения значений счет прерывается.

Один из основных недостатков метода Гаусса связан с тем, что при его реализации накапливается вычислительная погрешность.

$$\frac{m^3}{3}$$

Для систем порядка m число действий умножения и деления близко $\frac{m^3}{3}$ и быстро растет с величиной m .

Для того, чтобы уменьшить рост вычислительной погрешности применяются различные модификации метода Гаусса. Например, метод Гаусса с выбором главного элемента по столбцам, в этом случае на каждом этапе прямого хода строки матрицы переставляются таким образом, чтобы диагональный угловой элемент был максимальным. При исключении соответствующего неизвестного из других строк деление будет производиться на наибольший из возможных коэффициентов и, следовательно, относительная погрешность будет наименьшей.

Пример реализации метода Гаусса в **Maxima** приведен в функции ниже (применен метод без выбора главного элемента):

```
(%i1) gauss(a0,b0,n):=block([a,b,i,j,k,d],
  a:copymatrix(a0), b:copymatrix(b0),
  x:copymatrix(b0),
  for i:1 thru n-1 do
  (
    for k:i+1 thru n do
    (
      d:a[k,i]/a[i,i],
      for j:i+1 thru n do (a[k,j]:a[k,j]-
        a[i,j]*d),
      b[k,1]:b[k,1]-b[i,1]*d
    )
  ),
  for i:n thru 1 step -1 do
  (
    for j:i+1 thru n do (
      b[i,1]:b[i,1]-a[i,j]*x[j,1]),
    x[i,1]:b[i,1]/a[i,i]
  ),
  x)$
```

Пример обращения к функции, реализующей метод Гаусса:

```
(%i2) aa:matrix([3,1,1],[1,3,1],[1,1,3]);
bb:matrix([6],[6],[8]);
zz:gauss(aa,bb,3);
```

$$(\%o2) \begin{pmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{pmatrix} (\%o3) \begin{pmatrix} 6 \\ 6 \\ 8 \end{pmatrix} (\%o4) \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

Проверка вычислений показывает, что перемножение матрицы A на вектор решения zz дает вектор, совпадающий с вектором правых частей bb .

(%i5) aa.zz;

$$(\%o5) \begin{pmatrix} 6.0 \\ 6.0 \\ 8.0 \end{pmatrix}$$

Существует метод Гаусса с выбором главного элемента по всей матрице. В этом случае переставляются не только строки, но и столбцы. Использование модификаций метода Гаусса приводит к усложнению алгоритма увеличению числа операций и соответственно к росту времени счета.

Выполняемые в методе Гаусса преобразования прямого хода, приведшие матрицу A системы к треугольному виду позволяют вычислить определитель матрицы

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ 0 & a_{22}^{(1)} & \dots & a_{2m}^{(1)} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{m,m}^{(m-1)} \end{vmatrix} = a_{11} \cdot a_{22}^{(1)} \cdot \dots \cdot a_{m,m}^{(m-1)}$$

Метод Гаусса позволяет найти и обратную матрицу. Для этого необходимо решить матричное уравнение

$$A \cdot X = E,$$

где E – единичная матрица. Его решение сводится к решению m систем

$$A\bar{x}^{(j)} = \bar{\delta}^{(j)}, \quad j = 1, 2, \dots, m,$$

у вектора $\bar{\delta}^{(j)}$ j -я компонента равна единице, а остальные компоненты равны нулю.

8.3.3 Метод квадратного корня

Метод квадратного корня основан на разложении матрицы A в произведение

$$A = S^T S,$$

где S — верхняя треугольная матрица с положительными элементами на главной диагонали, S^T — транспонированная к ней матрица.

Пусть A — матрица размером $m \times m$. Тогда

$$(S^T S)_{ij} = \sum_{k=1}^m s_{ik}^T s_{kj} \quad (8.2)$$

Из условия (8.2) получаются уравнения

$$\sum_{k=1}^m s_{ik}^T s_{kj} = a_{ij}, \quad i, j = 1, 2, \dots, m \quad (8.3)$$

Так как матрица A симметричная, не ограничивая общности, можно считать, что в системе (8.3) выполняется неравенство $i \leq j$. Тогда (8.3) можно переписать в виде

$$\sum_{k=1}^{i-1} s_{ik}^T s_{kj} + s_{ii} s_{ij} + \sum_{k=i+1}^m s_{ik}^T s_{kj} = a_{ij}$$

$$s_{ii} s_{ij} + \sum_{k=1}^{i-1} s_{ik}^T s_{kj} = a_{ij}, \quad i \leq j.$$

В частности, при $i = j$ получится

$$|s_{ii}|^2 = a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2$$

$$s_{ii} = \left(\left| a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 \right| \right)^{1/2}$$

Далее, при $i < j$ получим

$$s_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} s_{ik}^T s_{kj}}{s_{ii}}$$

По приведённым формулам находятся рекуррентно все ненулевые элементы матрицы S .

Обратный ход метода квадратного корня состоит в последовательном решении двух систем уравнений с треугольными матрицами.

$$S^T y = f,$$

$$Sx = y.$$

Решения этих систем находятся по рекуррентным формулам:

$$\left\{ \begin{array}{l} y_i = \frac{f_i - \sum_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = 2, 3, \dots, m \\ y_1 = \frac{f_1}{s_{11}} \end{array} \right.$$

$$\left\{ \begin{array}{l} x_i = \frac{y_i - \sum_{k=i+1}^m s_{ik} x_k}{s_{ii}}, \quad i = m-1, m-2, \dots, 1 \\ x_m = \frac{y_m}{s_{mm}} \end{array} \right.$$

Всего метод квадратного корня при факторизации $A = S^T S$ требует $\frac{m^3}{3}$ операций умножения и деления и m операций извлечения квадратного корня.

Пример функции, реализующей метод квадратного корня:

```
(%i1) holetsk(a0,b0):=block([L,Lt,x,y,i,j,k,n],
  n:length(a0), L:zeromatrix(n,n),
  for i:1 thru n do (
    for j:1 thru i-1 do (
      s:0,
      for k:1 thru j-1 do (s:s+L[i,k]*L[j,k]),
      L[i,j]:1/L[j,j]*(a0[i,j]-s)
    ),
    s:0,
    for k:1 thru i-1 do (s:s+L[i,k]^2),
    L[i,i]:sqrt(a0[i,i]-s)
  ),Lt:transpose(L),
  y:zeromatrix(n,1), x:zeromatrix(n,1),
  for i:1 thru n do (
    s:0,
    for k:1 thru i-1 do (s:s+L[i,k]*y[k,1]),
    y[i,1]:(b0[i,1]-s)/L[i,i]
  ),
  for i:n thru 1 step -1 do (
    s:0,
    for k:n thru i+1 step -1 do
(s:s+Lt[i,k]*x[k,1]),
    x[i,1]:(y[i,1]-s)/Lt[i,i]
  ),x
  )$
```

Тест данной функции (решение системы $Ax = B$, B — матрица $n \times 1$, A — квадратная симметричная матрица $n \times n$, результат решения — вектор $n \times 1$):

```
(%i2) A:matrix([4,1,1],[1,4,1],[1,1,4])$
B:matrix([1],[1],[1])$
```

Результаты вычислений:

```
(%i4) x:holetsk(A,B);
```

```
(%o4)  $\begin{pmatrix} 1 \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}$ 
```

Проверка:

```
(%i5) A.x;
```

$$(\%05) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

8.3.4 Корректность постановки задачи и понятие обусловленности

При использовании численных методов для решения тех или иных математических задач необходимо различать свойства самой задачи и свойства вычислительного алгоритма, предназначенного для ее решения.

Говорят, что задача поставлена корректно, если решение существует и единственно и если оно непрерывно зависит от входных данных. Последнее свойство называется также устойчивостью относительно входных данных.

Корректность исходной математической задачи еще не гарантирует хороших свойств численного метода ее решений и требует специального исследования.

Известно, что решение задачи (8.1) существует тогда и только тогда, когда $\det A \neq 0$. В этом случае можно определить обратную матрицу A^{-1} и решение записать в виде $\bar{x} = A^{-1}\bar{f}$.

Исследование устойчивости задачи (8.1) сводится к исследованию зависимости ее решения от правых частей \bar{f} и элементов a_{ij} матрицы A . Для того чтобы можно было говорить о непрерывной зависимости вектора решений от некоторых параметров, необходимо на множестве m -мерных векторов принадлежащих линейному пространству \mathbb{H} , ввести метрику.

В линейной алгебре предлагается определение множества метрик l_p —

норма $\|\bar{x}\|_p = \left(\sum_{i=1}^m |x_i|^p \right)^{1/p}$ из которого легко получить наиболее часто используемые метрики

- при $p = 1, \|\bar{x}\|_1 = \sum_{i=1}^m |x_i|$,
- при

$$p = 2, \|\bar{x}\|_2 = \left(\sum_{i=1}^m |x_i|^2 \right)^{1/2}$$

- при

$$p \rightarrow \infty, \|\bar{x}\|_\infty = \lim_{p \rightarrow \infty} \left(\sum_{i=1}^m |x_i|^p \right)^{1/p}$$

$$\|A\| = \sup_{0 \neq x \in \mathbb{H}} \frac{\|Ax\|}{\|x\|},$$

Подчиненные нормы матриц определяемые как соответственно записываются в следующем виде:

$$\|A\|_1 = \max_{1 \leq j \leq m} \sum_{i=1}^m |a_{ij}|,$$

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^m |a_{ij}|,$$

$$\|A\|_2 = \sqrt{\rho(A^T A)} = \sqrt{\sum_{i=1}^m \sum_{j=1}^m a_{ij}^2}.$$

Обычно рассматривают два вида устойчивости решения системы (8.1):

- по правым частям;
- по коэффициентам системы (8.1) и по правым частям.

Наряду с исходной системой (8.1) рассмотрим систему с "возмущенными" правыми частями

$$A \cdot \tilde{x} = \tilde{f},$$

где $\tilde{f} = \bar{f} + \delta \bar{f}$ возмущенная правая часть системы, а $\tilde{x} = \bar{x} + \delta \bar{x}$ возмущенное решение.

Можно получить оценку, выражающую зависимость относительной погрешности решения от относительной погрешности правых частей

$$\frac{\|\delta \bar{x}\|}{\|\bar{x}\|} \leq M_A \frac{\|\delta \bar{f}\|}{\|\bar{f}\|},$$

где $M_A = \|A\| \cdot \|A^{-1}\|$ — число обусловленности матрицы A (в современной литературе это число обозначают как $\text{cond}(A)$). Если число обусловленности велико ($M_A \sim 10^k$, $k > 2$), то говорят, что матрица A плохо обусловлена. В этом случае малые возмущения правых частей системы (8.1), вызванные либо неточностью задания исходных данных, либо вызванные погрешностями вычисления существенно влияют на решение системы.

Если возмущение внесено в матрицу A , то для относительных возмущений решения имеет место следующая оценка:

$$\frac{\|\delta \bar{x}\|}{\|\bar{x}\|} \leq \frac{M_A}{1 - M_A \frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta \bar{f}\|}{\|\bar{f}\|} \right).$$

В **Maxima** матричные нормы вычисляются посредством функции *mat_norm*. Синтаксис вызова: *mat_norm(M, type)*, где *M* — матрица, *type* — тип нормы, *type* может быть равен 1 (норма $\|A\|_1$), *inf* (норма $\|A\|_\infty$), *frobenius* (норма $\|A\|_2$).

Пример вычисления указанных видов нормы в **Maxima**:

```
(%i1) A:matrix([1,2,3],[4,5,6],[7,8,9]);
```

```
(%o1)  $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ 
```

```
(%i2) mat_norm(A,1);
```

```
(%o2) 18
```

```
(%i3) mat_norm(A,inf);
```

```
(%o3) 24
```

```
(%i4) mat_norm(A,frobenius);
```

```
(%o4)  $\sqrt{285}$ 
```

Вычислим число обусловленности для плохо и хорошо обусловленных матриц:

```
(%i1) A:matrix([1,1],[0.99,1]);
```

```
(%o1)  $\begin{pmatrix} 1 & 1 \\ 0.99 & 1 \end{pmatrix}$ 
```

```
(%i2) nrA:mat_norm(A,frobenius);
```

```
(%o2) 1.995018796903929
```

```
(%i3) A1:invert(A);
```

```
(%o3)  $\begin{pmatrix} 99.99999999999992 & -99.99999999999992 \\ -98.99999999999992 & 99.99999999999992 \end{pmatrix}$ 
```

```
(%i4) nrA1:mat_norm(A1,frobenius);
```

```
(%o4) 199.5018796903927
```

```
(%i5) MA:nrA*nrA1;
```

```
(%o5) 398.00999999999997
```

Таким образом, для плохо обусловленной матрицы число обусловленности достигает почти 400.

Аналогичным путём (с использованием нормы Фробениуса) для матрицы

$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ число обусловленности составило $MB = 2$.

8.3.5 О вычислительных затратах

Один из важных факторов предопределяющих выбор того или иного метода при решении конкретных задач, является вычислительная эффективность метода. Учитывая, что операция сложения выполняется намного быстрее, чем

операция умножения и деления, обычно ограничиваются подсчетом последних. Для решения СЛАУ методом Гаусса без выбора главного элемента требуется

$$\frac{m^3}{3} + m^2 - \frac{m}{3}$$

$$\frac{m^3}{6} + \frac{3m^2}{2} + \frac{m}{3}$$

умножений и делений, решение СЛАУ методом квадратного корня требует $\frac{m^3}{6} + \frac{3m^2}{2} + \frac{m}{3}$ и m операций извлечения корней. При больших значениях размерности m , можно сказать, что вычислительные затраты на операции умножения и деления в методе Гаусса составляют

$$O\left(\frac{m^3}{3}\right), \text{ в методе квадратных корней } O\left(\frac{m^3}{6}\right).$$

величину

8.4 Итерационные методы

В приближенных или итерационных методах решение системы линейных алгебраических уравнений является пределом итерационной последовательности, получаемой с помощью этих методов. К ним относятся: метод простой итерации, метод Зейделя и др. Итерационные методы выгодны для системы специального вида, со слабо заполненной матрицей очень большого вида порядка $10^3 \dots 10^5$.

Для итерационных методов характерно то, что они требуют начальных приближений значений неизвестных, решение ищется в виде последовательности, постепенно улучшающихся приближений, и кроме того, итерационный процесс должен быть сходящимся. В вычислительной практике процесс итерации обычно продолжается до тех пор, пока два последовательных приближения не совпадут в пределах заданной точности.

8.4.1 Матричная формулировка итерационных методов решения систем линейных уравнений

При использовании СКМ **Maxima** вполне обосновано использование и матричной формулировки итерационных методов.

Рассмотрим решение системы $Ax = f$ (A — квадратная матрица, f — вектор правых частей, x — вектор неизвестных). Обозначим $A = L + D + U$, где L — нижняя треугольная матрица с нулевыми диагональными элементами; D — диагональная матрица; U — верхняя треугольная матрица с нулевыми диагональными элементами.

Для решения этой системы рассмотрим итерационный процесс

$$x^{i+1} = x^i - H_i(Ax^i - f),$$

где $\det(H_i) \neq 0$, или

$$x^{i+1} = P_i x^i + d_i, \quad x(0) = x_0,$$

где $P_i = I - P_i A$ — оператор i -го шага итерационного процесса; $d_i = P_i f$.

Итерационный процесс сходящийся, если последовательность $\{x_i\}$ сходится к решению x^* при любом x_0 .

Если матрица не зависит от номера итерации, итерационный процесс называется стационарным:

$$x^{i+1} = Px^i + d. \quad (8.4)$$

Необходимым и достаточным условием сходимости стационарного процесса является выполнение условия $\rho(P) < 1$, где $\rho(P)$ — спектральный радиус матрицы P (наибольшее по модулю собственное число матрицы P).

С использованием введённых обозначений метод простой итерации (метод Якоби) даётся формулой:

$$P = I - D^{-1}A, \quad D = \text{diag}(a_{ii}), \quad H = D^{-1},$$

а метод Гаусса–Зейделя — формулой:

$$P = -(D + L)^{-1}U, \quad H = (D + L)^{-1}.$$

Рассмотрим поэлементные расчетные соотношения для методов Якоби и Гаусса–Зейделя.

Все элементы главной диагонали матрицы $I = D^{-1}A$ равны нулю, остальные элементы равны $-\frac{a_{ij}}{a_{ii}}$, $i, j = \overrightarrow{1, n}$. Свободный член уравнения

$$(8.4) \text{ равен } \frac{f_i}{a_{ii}}.$$

Таким образом, для метода Якоби итерационный процесс записывается в виде

$$x^{k+1} = Cx^k + E, \quad \text{где}$$

$$C_{ij} = -\frac{a_{ij}}{a_{ii}}, \quad k = 0, 1, \dots, \quad i, j = \overrightarrow{1, n}; \quad E_i = \frac{f_i}{a_{ii}}.$$

Для метода Гаусса–Зейделя $x^{k+1} = -(D + L)^{-1}Ux^k + (D + L)^{-1}f$, или $(D + L)x^{k+1} = -Ux^k + b$, $x^{k+1} = Bx^{k+1} + Ex^k + e$, где

$$B = \begin{bmatrix} 0 & 0 & \dots & 0 \\ c_{21} & 0 & \dots & 0 \\ c_{31} & c_{32} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ c_{n-1,1} & c_{n-1,2} & \dots & 0 \\ c_{n1} & c_{n2} & \dots & 0 \end{bmatrix}, \text{ и } E = \begin{bmatrix} 0 & c_{21} & \dots & \dots & c_{1n} \\ 0 & 0 & \dots & \dots & c_{2n} \\ 0 & \dots & \dots & \dots & \dots \\ \vdots & & & \dots & c_{n-1,n} \\ 0 & & & & 0 \end{bmatrix}.$$

Рассмотрим решение конкретной системы уравнений $Ax = b$ методом Якоби:

$$8x_1 - x_2 + 2x_3 = 8,$$

$$x_1 + 9x_2 + 3x_3 = 18,$$

$$2x_1 - 3x_2 + 10x_3 = -5.$$

Вычисляем элементы матрицы B и вектора e :

$$B = \begin{bmatrix} 0 & \frac{1}{8} & -\frac{2}{8} \\ -\frac{1}{9} & 0 & -\frac{3}{9} \\ -0,2 & 0,3 & 0 \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ 2 \\ 0,5 \end{bmatrix}.$$

Вычислим значения x по формуле $x^{k+1} = Bx^k + e$. Для решения использована следующая последовательность команд Maxima:

преобразование заданных матриц

```
(%i1) A:matrix([8,-1,2],[1,9,3],[2,-3,10])$
      b:matrix([8],[18],[5])$
      A0:matrix([A[1,1],A[1,1],A[1,1]],
                [A[2,2],A[2,2],A[2,2]],
                [A[3,3],A[3,3],A[3,3]])$
      B:-A/A0+diagmatrix(3,1)$
      e:b/matrix([A[1,1]],[A[2,2]],[A[3,3]])$ x:e$
```

собственно вычисление решения

```
(%i7) xt:float(B.x+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      x0:xt$
      xt:float(B.xt+e)$
      x1:xt$
      r:x1-x0$
      float(r); /* оценка сходимости*/
      float(A.x1-b); /* оценка невязки*/
```

```
(%o18) (
        -1.2272477756924971 10-5
        -1.3018148195786949 10-4
        6.2047575160040225 10-5
      )
```

```
(%o19) (
        2.5427663227794994 10-4
        1.738702477211973 10-4
        3.6599949035931445 10-4
      )
```

8.4.2 Метод простой итерации

Для решения системы линейных алгебраических уравнений (8.1) итерационным методом её необходимо привести к нормальному виду:

$$\bar{x} = P\bar{x} + \bar{g} \quad (8.5)$$

Стационарное итерационное правило получаем, если матрица B и вектор \bar{g} не зависят от номера итерации: $\bar{x}^{k+1} = P\bar{x}^k + \bar{g}$. Нестационарное итерационное правило получаем если матрица B или вектор \bar{g} изменяются с ростом номера итерации: $\bar{x}^{k+1} = P_k\bar{x}^k + \bar{g}^k$.

Стационарное итерационное правило обычно называют методом простой итерации. Предел итерационной последовательности является точным решением системы (8.5) или (8.1).

Для того, чтобы метод простой итерации сходился при любом начальном приближении, необходимо и достаточно, чтобы все собственные значения матрицы B были по модулю меньше единицы.

В силу того, что проверить сформулированное выше условие достаточно сложно на практике применяют следующие достаточные признаки:

- для того чтобы метод простой итерации сходился, достаточно, чтобы какая-либо норма матрицы P была меньше единицы;
- для того чтобы метод простой итерации сходился, достаточно, чтобы выполнялось одно из следующих условий:

$$\begin{aligned} \sum_{j=1}^n |P_{ij}| &< 1, \quad i = \overline{1, n} \\ & ; \\ \sum_{i=1}^n |P_{ij}| &< 1, \quad j = \overline{1, n} \\ & ; \\ \sum_{i,j=1}^n |P_{ij}|^2 &< 1 \end{aligned}$$

Для определения скорости сходимости можно воспользоваться следующей теоремой: если какая-либо норма матрицы P , согласованная с данной нормой вектора, меньше единицы, то имеет место следующая оценка погрешности метода простой итерации:

$$\|\bar{x}^* - \bar{x}^k\| < \|P\|^k \cdot \|\bar{x}^k\| + \frac{\|P\|^k \cdot \|\bar{g}\|}{1 - \|P\|},$$

где \bar{x}^* — точное решение системы (8.1).

Другими словами, условие сходимости выполняется, если выполняется условие доминирования диагональных элементов матрицы исходной системы

$$A \text{ по строкам или столбцам: } \sum_{i,j=1}^n |a_{ij}| < |a_{ii}| \quad \text{или} \quad \sum_{i,j=1}^n |a_{ij}| \leq |a_{jj}|$$

В этом случае легко можно перейти от системы вида (8.1) к системе (8.5). Для этого разделим i -ое уравнение системы на a_{ii} и выразим x_i :

$$x_i = \frac{f_i}{a_{ii}} - \frac{a_{i1}}{a_{ii}}x_1 - \dots - \frac{a_{in}}{a_{ii}}x_n,$$

т.е. для матрицы P будет выполнено одно из условий сходимости, где

$$P = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix}.$$

Пример реализации метода простой итерации средствами **Maxima** с печатью промежуточных результатов представлен в скрипте ниже:

```
(%i1)
  iterpr(a0,b0,x,n,eps):=block([a,b,x0,i,j,s,sum,p],
  a:copymatrix(a0), b:copymatrix(b0),
x0:copymatrix(x),
  sum:1, p:0,
  while sum>eps do (
    sum:0, p:p+1, print("p= ",p," x= ",float(x)),
    for i:1 thru n do (
      s:b[i,1],
      for j:1 thru n do (s:s-a[i,j]*x0[j,1]),
      s:s/a[i,i], x[i,1]:x0[i,1]+s, sum:sum+abs(s)
    ),
    x0:copymatrix(x)
  ),
float(x))$
```

8.4.3 Метод Зейделя

В методе Зейделя система (8.1) также приводится к системе (8.5). Но при вычислении последующей компоненты вектора используются уже вычисленные компоненты этого вектора.

Итерационная формула метода в скалярной форме записывается следующим образом:

$$x_i^{(k+1)} = \sum_{j=1}^i p_{ij} x_i^{(k+1)} + \sum_{j=i+1}^n p_{ij} x_j^{(k)} + g_i.$$

Установим связь между методом Зейделя и методом простой итерации. Для этого матрицу B представим в виде суммы двух матриц: $P = H + F$, где

$$H = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ p_{21} & 0 & 0 & \dots & 0 \\ p_{31} & p_{32} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ p_{n1} & p_{n2} & p_{n3} & \dots & 0 \end{bmatrix}; \quad F = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \dots & p_{1n} \\ 0 & p_{22} & p_{23} & \dots & p_{2n} \\ 0 & 0 & p_{33} & \dots & p_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & p_{nn} \end{bmatrix}.$$

Итерационная формула метода Зейделя в матричной форме записывается в виде:

$$\bar{x}^{(k+1)} = H\bar{x}^{(k+1)} + F\bar{x}^{(k)} + \bar{g}, \quad \text{или}$$

$$(E - H)\bar{x}^{(k+1)} = F\bar{x}^{(k)} + \bar{g}, \quad \text{откуда}$$

$$\bar{x}^{(k+1)} = (E - H)^{-1}F\bar{x}^{(k)} + (E - H)^{-1}\bar{g},$$

т.е. метод Зейделя эквивалентен методу простой итерации с матрицей $(E - H)^{-1}F$.

Исходя из полученной аналогии методов Зейделя и простой итерации, можно сформулировать следующий признак сходимости метода Зейделя: для того чтобы метод Зейделя сходился, необходимо и достаточно, чтобы все собственные значения матрицы $(E - H)^{-1}F$ по модулю были меньше единицы.

Другими словами, чтобы метод Зейделя сходился, необходимо и достаточно, чтобы все корни уравнения по модулю были меньше единицы, т.к.

$$\begin{aligned} & \left| (E - H)^{-1}F - \lambda E \right| = \\ & \left| (E - H)^{-1} (E - H) \left[(E - H)^{-1}F - \lambda E \right] \right| = \\ & \left| (E - H)^{-1} \right| \left| F + \lambda H - \lambda E \right| = \\ & \left| F + \lambda H - \lambda E \right| = 0. \end{aligned}$$

Сформулируем достаточный признак сходимости: для того, чтобы метод Зейделя сходился, достаточно, чтобы выполнялось одно из условий:

- $\|P\|_1 = \max_i \sum_{j=1}^n |p_{ij}| < 1$
- $\|P\|_2 = \max_j \sum_{i=1}^n |p_{ij}| < 1$
- $\|P\|_3 = \sqrt{\sum_{i,j=1}^n |p_{ij}|^2} < 1$

При использовании метода Зейделя итерационный процесс сходится к единственному решению быстрее метода простых итераций.

Пример реализации метода Зейделя:

```
(%i1) seidel(a0,b0,x,n,eps):=block([a,b,i,j,s,sum,p],
a:copymatrix(a0), b:copymatrix(b0),
sum:1, p:0,
while sum>eps do (
sum:0, p:p+1, print("p= ",p),
for i:1 thru n do
```

```
(
    s:b[i,1],
    for j:1 thru n do (s:s-a[i,j]*x[j,1]),
    s:s/a[i,i], x[i,1]:x[i,1]+s, sum:sum+abs(s)
)
),
float(x))$
```

Пример решения простой системы методом Зейделя:

```
(%i2) aa:matrix([3,1,1],[1,3,1],[1,1,3]);
bb:matrix([6],[6],[8]);
x:matrix([3],[3],[3]);
zz:seidel(aa,bb,x,3,0.0000001);
```

```
(%o2)  $\begin{pmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{pmatrix}$  (%o3)  $\begin{pmatrix} 6 \\ 6 \\ 8 \end{pmatrix}$  (%o4)  $\begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}$ 
```

```
p = 1 p = 2 p = 3 p = 4 p = 5 p = 6
p = 7 p = 8 p = 9 p = 10 p = 11 p = 12
```

```
(%o5)  $\begin{pmatrix} 1.000000000753427 \\ 0.99999999214211 \\ 2.000000002368154 \end{pmatrix}$ 
```

8.5 Решение обыкновенных дифференциальных уравнений

8.5.1 Методы решения задачи Коши

Среди задач, с которыми приходится иметь дело в вычислительной практике, значительную часть составляют различные задачи, сводящиеся к решению обыкновенных дифференциальных уравнений. Обычно приходится прибегать к помощи приближенных методов решения подобных задач. В случае обыкновенных дифференциальных уравнений в зависимости от того, ставятся ли дополнительные условия в одной или нескольких точках отрезка изменения независимой переменной, задачи обычно подразделяются на одноточечные (задачи с начальными условиями или задачи Коши) и многоточечные. Среди многоточечных задач наиболее часто в прикладных вопросах встречаются так называемые граничные задачи, когда дополнительные условия ставятся на концах рассматриваемого отрезка. В дальнейшем ограничимся рассмотрением численных методов решения задачи Коши. Для простоты изложения методов решения задачи будем рассматривать случай одного обыкновенного дифференциального уравнения первого порядка.

Пусть на отрезке $x_0 \leq x \leq L$ требуется найти решение $y(x)$ дифференциального уравнения

$$y' = f(x, y), \quad (8.6)$$

удовлетворяющее при $x = x_0$ начальному условию $y(x_0) = y_0$.

Будем считать, что условия существования и единственности решения поставленной задачи Коши выполнены.

На практике найти общее либо частное решение задачи Коши удается для весьма ограниченного круга задач, поэтому приходится решать эту задачу приближенно.

Отрезок $[x_0, L]$ накрывается сеткой (разбивается на интервалы) чаще всего с постоянным шагом h ($h = x_{n+1} - x_n$), и по какому-то решающему правилу находится значение $y_{n+1} = y(x_{n+1})$. Таким образом, результатом решения задачи Коши численными методами является таблица, состоящую из двух векторов: $x = (x_0, x_1, \dots, x_n)$ — вектора аргументов и соответствующего ему вектора значений искомой функции $y = (y_0, y_1, \dots, y_n)$.

Численные методы (правила), в которых для нахождения значения функции в новой точке используется информация только об одной (предыдущей) точке, называются одношаговыми.

Численные методы (правила), в которых для нахождения значения функции в новой точке используется информация о нескольких (предыдущих) точках, называются многошаговыми.

Из общего курса обыкновенных дифференциальных уравнений широкое распространение получил аналитический метод, основанный на идее разложения в ряд решения рассматриваемой задачи Коши. Особенно часто для этих целей используется ряд Тейлора. В этом случае вычислительные правила строятся особенно просто.

Приближенное решение $y_m(x)$ исходной задачи ищут в виде

$$y_m(x) = \sum_{i=0}^m \frac{(x - x_0)^i}{i!} y^{(i)}(x_0), \quad (8.7)$$

$$x_0 \leq x \leq b.$$

Здесь $y^{(0)}(x_0) = y(x_0)$, $y^{(1)}(x_0) = y'(x_0) = f(x_0, y_0)$, а значения $y^{(i)}(x_0)$, $i = 2, 3, \dots, m$ находят по формулам, полученным последовательным дифференцированием заданного уравнения:

$$\begin{aligned} y^{(2)}(x_0) &= y''(x_0) = f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0); \\ y^{(3)}(x_0) &= y'''(x_0) = f_{x^2}(x_0, y_0) + 2f(x_0, y_0) f_y(x_0, y_0) + \\ &+ f^2(x_0, y_0) f_{y^2}(x_0, y_0) + f_y(x_0, y_0) [f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0)]; \end{aligned} \quad (8.8)$$

...

$$y^{(m)}(x_0) = F_m(f; f_x; f_{x^2}; f_{xy}; f_{y^2}; \dots; f_{x^{m-1}}; f_{y^{m-1}})|_{x=x_0, y=y_0}.$$

Для значений x , близких к x_0 , метод рядов (8.7) при достаточно большом значении m дает обычно хорошее приближение к точному решению $y(x)$ задачи (8.6). Однако с ростом расстояния $|x - x_0|$ погрешность приближения искомой функции рядом возрастает по абсолютной величине (при одном и том же количестве членов ряда), и правило (8.7) становится вовсе неприемлемым, когда x выходит из области сходимости соответствующего ряда (8.7) Тейлора.

Если в выражении (8.7) ограничиться $m = 1$, то для вычисления новых значений $y(x)$ нет необходимости пересчитывать значение производной, правда и точность решения будет невысока.

При использовании системы компьютерной алгебры более естественным выглядит метод последовательных приближений Пикара.

Рассмотрим интегрирование единичного дифференциального уравнения $\frac{dy}{dx} = f(x, y)$ на отрезке $[x_0, x]$ с начальным условием $y(x_0) = y_0$. При формальном интегрировании получим:

$$\int_{x_0}^x \frac{dy}{dx} dx = \int_{x_0}^x f(x, y) dx$$

Процедура последовательных приближений метода Пикара реализуется согласно следующей схеме

$$y_{n+1}(x) - y(x_0) = \int_{x_0}^x f(x, y_n(x)) dx$$

В качестве примера рассмотрим решение уравнения $y' = -y$, при $y(0) = 1, x_0 = 0$.

```
(%i1)  rp:-y$ y0:1$ x0:0$ /*rp-правая часть уравнения
*/
```

```
(%i4)  y1:y0+integrate(subst(y0,y,rp),x,x0,x);
/* y1 - первое приближение */
```

```
(%o4)  1 - x
```

```
(%i5)  y2:y0+integrate(subst(y1,y,rp),x,x0,x);
/* y2 - второе приближение */
```

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

```
(%i6)  y3:y0+integrate(subst(y2,y,rp),x,x0,x);
```

```
(%o6)   $1 - \frac{x^3 - 3x^2 + 6x}{6}$ 
```

```
(%i7)  expand(%); /* Очевидное решение рассматриваемого
ОДУ -
```

```
экспонента y=exp(-x).
```

В результате использование метода Пикара
получаем решение в виде ряда Тейлора */

$$(\%07) \quad -\frac{x^3}{6} + \frac{x^2}{2} - x + 1$$

8.5.2 Метод рядов, не требующий вычисления производных правой части уравнения

Естественно поставить задачу о таком усовершенствовании приведенного выше одношагового метода, которое сохраняло бы основные его достоинства, но не было бы связано с нахождением значений производных правой части уравнения

$$y_m(x_{n+1}) \approx \sum_{i=0}^m \frac{h^i}{i!} y^{(i)}(x_n), \quad (8.9)$$

где $x_{n+1} = x_n + h$.

Чтобы выполнить это условие (последнее), производные $y^{(i)}(x)$, $i = 2, 3, \dots, m$, входящие в правую часть уравнения (8.9), можно заменить по формулам численного дифференцирования их приближенными выражениями через значение функции y' и учесть, что $y'(x) = f[x, y(x)]$.

8.5.2.1 Метод Эйлера

В случае $m = 1$ приближенное равенство (8.9) не требует вычисления производных правой части уравнения и позволяет с погрешностью порядка h^2 находить значение $y(x_n + h)$ решения этого уравнения по известному его значению $y(x_n)$. Соответствующее одношаговое правило можно записать в виде

$$y_{n+1} = y_n + h f_n. \quad (8.10)$$

Это правило (8.10) впервые было построено Эйлером и носит его имя. Иногда его называют также правилом ломаных или методом касательных. Метод Эйлера имеет относительно низкий порядок точности — h^2 на одном шаге. Практическая оценка погрешности приближенного решения может быть получена по правилу Рунге.

Пример реализации метода Эйлера средствами **Maxima** приведён в следующем примере:

```
(%i1)
euler1(rp, fun, y0, x0, xend, h) := block([OK, _x, _y, _y1, rez]
,
  _x:x0, _y:y0, rez:[_y], OK:-1, eps:0.1e-7,
  while OK<0 do (
    if ((_x+h>xend) or (abs(_x+h-xend)<eps))
      then (h:xend-_x, _x:xend, OK:1)
      else (_x:_x+h),
```

```

_y1:makelist(float(_y[i]+h*subst([fun[i]=_y[i],x=_x],
rp[i])),i,1,length(_y)),rez:append(rez,[_y1]),
_y:_y1
),
rez
) $

```

Правые части решаемых дифференциальных уравнений передаются в функцию *euler1* в списке *rp*. По умолчанию предполагается, что список имён зависимых переменных — *fun*, имя независимой переменной — *x*. Начальные значения независимой и зависимых переменных — список *y0* и скалярная величина *x0*, граница интервала интегрирования — величина *xend*, шаг интегрирования — *h*.

Следующий пример — обращение к функции *euler1*. Приведено решение системы из трёх дифференциальных уравнений на интервале $[0, 1]$ с шагом $h = 0.1$:

$$\begin{cases} \frac{dy}{dx} = -2y, \\ \frac{dv}{dx} = -5v, \\ \frac{dz}{dx} = 3x. \end{cases}$$

С начальными условиями $y(0) = 1.0; v(0) = 1.0; z(0) = 0$, решением уравнений данной системы будут функции:

$$\begin{cases} y(x) = e^{-2x}, \\ v(x) = e^{-5x}, \\ z(x) = \frac{3}{2} \cdot x^2. \end{cases}$$

```

(%i2) euler1([-2*y,-5*v,3*x],[y,v,z],[1,1,0],0,1,0.1);
(%o2) [[1, 1, 0], [0.8, 0.5, 0.03], [0.64, 0.25, 0.09], [0.512, 0.125, 0.18],
[0.4096, 0.0625, 0.3], [0.32768, 0.03125, 0.45], [0.262144, 0.015625, 0.63],
[0.2097152, 0.0078125, 0.84], [0.16777216, 0.00390625, 1.08], [0.134217728,
0.001953125, 1.35], [0.1073741824, 9.76562499999999913 * 10^-4, 1.65]]

```

Проверить решение можно сравнивая графики точного решения и множества вычисленных приближенных значений. Пример последовательности команд, позволяющих выделить отдельные компоненты решения системы ОДУ и построить график точного и приближенного решения третьего уравнения системы, представлен ниже (точные решения — списки *yf*, *vf*, *zf*;

приближенные решения — списки yT, vT, zT ; список значений независимой переменной — xg).

```
(%i3) rez:euler1([-2*y,-
5*v,3*x],[y,v,z],[1,1,0],0,1.0,0.1)$
n:length(rez)$
yr:makelist(rez[k][1],k,1,n)$
vr:makelist(rez[k][2],k,1,n)$
zr:makelist(rez[k][3],k,1,n)$
xg:makelist(0.1*(k-1),k,1,n)$
yf:makelist(exp(-2*xg[k]),k,1,n)$
vf:makelist(exp(-5*xg[k]),k,1,n)$
zf:makelist(3*xg[k]^2/2,k,1,n)$
plot2d([[discrete,xg,zr],[discrete,xg,zf]],
[style,points,lines])$
```

Уменьшение шага h приводит к уменьшению погрешности решения (в данном примере — шаг 0.1).

8.5.2.2 Метод Рунге-Кутты

Изложим идею метода на примере задачи Коши:

$$y' = f(x, y);$$

$$x_0 \leq x \leq b;$$

$$y(x_0) = y_0.$$

Интегрируя это уравнение в пределах от x до $x+h$ ($0 < h < 1$), получим равенство

$$y(x+h) = y(x) + \int_x^{x+h} f[t, y(t)] dt, \quad (8.10)$$

которое посредством последнего интеграла связывает значения решения рассматриваемого уравнения в двух точках, удаленных друг от друга на расстояние шага h .

Для удобства записи выражения (8.11) используем обозначение $\Delta y = y(x+h) - y(x)$ и замену переменной интегрирования $t = x+h$. Окончательно получим:

$$\Delta y = h \int_0^1 f[x + \alpha h, y(x + \alpha h)] d\alpha \quad (8.12)$$

В зависимости от способа вычисления интеграла в выражении (8.12) получают различные методы численного интегрирования обыкновенных дифференциальных уравнений.

Рассмотрим линейную комбинацию величин ϕ_i , $i = 0, 1, \dots, q$, которая будет являться аналогом квадратурной суммы и позволит вычислить приближенное значение приращения Δy :

$$\Delta y \approx \sum_{i=0}^q a_i \phi_i,$$

где

$$\phi_0 = hf(x, y);$$

$$\phi_1 = hf(x + \alpha_1 h; y + \beta_{10} \phi_0);$$

$$\phi_2 = hf(x + \alpha_2 h; y + \beta_{20} \phi_0 + \beta_{21} \phi_1);$$

...

Метод четвертого порядка для $q = 3$, являющийся аналогом широко известной в литературе четырехточечной квадратурной формулы "трех восьмых", имеет вид

$$\Delta y \approx \frac{1}{8}(\phi_0 + 3\phi_1 + 3\phi_2 + \phi_3),$$

где

$$\phi_0 = hf(x_n, y_n);$$

$$\phi_1 = hf\left(x_n + \frac{h}{3}, y_n + \frac{\phi_0}{3}\right);$$

$$\phi_2 = hf\left(x_n + \frac{2}{3}h, y_n - \frac{\phi_0}{3} - \phi_1\right);$$

$$\phi_3 = hf(x_n + h, y_n + \phi_0 - \phi_1 + \phi_2).$$

Особо широко известно другое вычислительное правило типа Рунге-Кутты четвертого порядка точности:

$$\Delta y = \frac{1}{6}(\phi_0 + 2\phi_1 + 2\phi_2 + \phi_3),$$

где

$$\phi_0 = hf(x_n, y_n),$$

$$\phi_1 = hf\left(x_n + \frac{h}{2}, y_n + \frac{\phi_0}{2}\right),$$

$$\phi_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{\phi_1}{2}\right),$$

$$\phi_3 = hf(x_n + h, y_n + \phi_2).$$

Метод Рунге-Кутты имеет погрешность четвертого порядка ($\sim h^4$).

Функция **Maxima**, реализующая метод Рунге-Кутты 4-го порядка, приведена в следующем примере (с печатью промежуточных результатов):

```
(%i1) rk4(rp, fun, y0, x0, xend, h) := block(
      [OK, n, h1, _x, _y, _k1, _k2, _k3, _k4, rez],
```

```

_x:x0, _y:y0, rez:[_y], OK:-1, h1:h,
n:length(_y),
while OK<0 do (
  if (_x+h1>=xend) then (h1:xend-_x, OK:1),
  _k1:makelist(float(h1*subst([fun[i]=
float(_y[i]),x=float(_x)],rp[i])),i,1,n),
  _k2:makelist(float(h1*subst([fun[i]=
float(_y[i]+_k1[i]/2),x=float(_x+h1/2)],
rp[i])),i,1,n),
  _k3:makelist(float(h1*subst([fun[i]=
float(_y[i]+_k2[i]/2),x=float(_x+h1/2)],
rp[i])),i,1,n),
  _k4:makelist(float(h1*subst([fun[i]=
float(_y[i]+_k3[i]),x=float(_x+h1)],rp[i])),
i,1,n),
  _y1:makelist(float(_y[i]+
(_k1[i]+2*_k2[i]+2*_k3[i]+_k4[i])/6),i,1,n),
  rez:append(rez,[_y1]),
  print("x= ",_x, " y= ",_y),
  _x:_x+h1,
  _y:_y1
), rez
) $

```

Пример обращения к функции *rk4* представлен следующей последовательностью команд (решалась та же система, что и при тестировании метода Эйлера):

```

(%i2) rk4([-2*y, -5*v, 3*x], [y, v, z], [1, 1, 1], 0, 1, 0.1);
x= 0 y= [1, 1, 1]
x= 0.1 y= [0.8187333333333333, 0.6067708333333333, 1.015]
x= 0.2 y= [0.6703242711111111, 0.36817084418403, 1.06]
x= 0.3 y= [0.54881682490104, 0.22339532993458, 1.135]
x= 0.4 y= [0.44933462844064, 0.13554977050718, 1.24]
x= 0.5 y= [0.3678852381253, 0.082247647208783, 1.375]
x= 0.6 y= [0.30119990729446, 0.04990547343658, 1.54]
x= 0.7 y= [0.24660240409888, 0.030281185705008, 1.735]
x= 0.8 y= [0.20190160831589, 0.018373740284549, 1.96]
x= 0.9 y= [0.16530357678183, 0.011148649703906, 2.215]
x= 1.0 y= [0.13533954843051, 0.0067646754713805, 2.5]

```

9. Справочный материал

9.1 Сокращённый список основных функций Maxima

Функция или переменная	Краткое описание
' , '' , , %	простейшие команды
addcol	Функция добавляет столбец к матрице
addrow	Функция добавляет строку к матрице
algsys	Функция решает полиномиальные системы уравнений. Допускаются системы из одного уравнения с одной неизвестной. Кроме того, допускаются не до определенные системы
allroots	Функция, которая находит и печатает все (в том числе и комплексные) корни полиномиального уравнения с действительными либо комплексными коэффициентами
antidiff	Функция выполняет интегрирование выражений с произвольными функциями, перед ее первым вызовом следует загрузить пакет <code>antid</code>
append	Функция позволяет склеивать два списка
arrayinfo	Функция печатает информацию о массиве — его вид, число индексов, размер
arrays	Переменная содержит список имен массивов первого и второго видов, определенных на данный момент
array	Функция определяет массив с данным именем, определенным количеством индексов и заданным размером
assume	Функция вводит информацию о переменной в базу данных
atom	Функция возвращает <i>true</i> , если аргумент не имеет структуры, т.е. составных частей (например, число или переменная не имеют структуры).
atvalue	Функция позволяет задать значение функции и ее производных при некоторых значениях аргументов
at	Функция вычисляет значение выражения в заданной точке с учетом свойства

Функция или переменная	Краткое описание
augmented_lagrangian_method	Функция осуществляет минимизацию ФНП с ограничениями
batch	Функция запускает файл с программой. Операторы выполняются один за другим либо до конца файла, либо до синтаксической ошибки, либо до некорректной операции
bc2	Функция позволяет учесть краевые условия в решениях дифференциальных уравнений второго порядка
cabs	Функция возвращает модуль комплексного выражения
carg	Функция возвращает фазу комплексного выражения
cfdisrep	Функция преобразует список (как правило результат выполнения функции cf) в собственно цепную дробь
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
cfdirep	Функция преобразует список в собственно цепную дробь
changevar	реализует замену переменных в интеграле
charpoly	Функция является до некоторой степени избыточной — она вычисляет характеристический полином матрицы (корни этого полинома — собственные значения матрицы)
closefile	Функция прекращает вывод в файл
col	Функция выделяет заданный столбец матрицы
combine	Функция объединяет слагаемые с идентичным знаменателем
compile	Функция сначала транслирует функцию Maxima на язык LISP, а затем компилирует эту функцию LISP'a до двоичных кодов и загружает их в память

Функция или переменная	Краткое описание
conjugate	Функция для вычисления комплексно-сопряжённых выражений
cons	Функция позволяет добавлять элемент в начало списка
contrib_ode	Функция решает дифференциальные уравнения (больше возможностей, чем у ode2)
copylist	Функция создаёт копию списка
create_list	Функция создаёт список
copymatrix	Функция Создает копию матрицы
cspline	Функция строит сплайн-интерполяцию
define	Функция позволяет преобразовать выражение в функцию
demoivre	Функция заменяет все экспоненты с мнимыми показателями на соответствующие тригонометрические функции
denom	Функция выделяет знаменатель
depends	Функция позволяет декларировать, что переменная зависит от одной или нескольких других переменных
desolve	Функция решает дифференциальные уравнения и системы дифференциальных уравнений методом преобразования Лапласа
determinant	Функция вычисляет детерминант матрицы
diff	Функция выполняет дифференцирование
display2d	Переменная включает или выключает "двумерное" рисование дробей, степеней, и т.п. Изначально установлено значение <i>true</i>
display	Функция печатает значения своих аргументов вместе с их именем, каждое в отдельной строке
disp	Функция печатает значения своих аргументов, причем каждое значение печатается в отдельной строке
divide	Функция позволяет вычислить частное и остаток от деления одного многочлена на другой
draw2d	строит двумерные графики
draw3d	строит трёхмерные графики

Функция или переменная	Краткое описание
echelon	Функция преобразует матрицу к верхней треугольной
eigenvalues	Функция аналитически вычисляет собственные значения матрицы
eigenvectors	Функция аналитически вычисляет собственные значения и собственные вектора матрицы, если это возможно
eliminate	Функция исключает из системы уравнений указанные переменные. Оставшиеся уравнения приводятся к виду с нулевой правой частью, которая опускается
endcons	Функция позволяет добавлять элемент в конец списка
ev	Функция является основной функцией, обрабатывающей выражения
expand	Функция раскрывает скобки
exponentialize	Функция приводит комплексное выражение к экспоненциальной форме
express	Функция преобразует дифференциальные операторы в выражения
factor	Функция представляет в виде произведения некоторых сомножителей заданное выражение
factorsum	Функция факторизует отдельные слагаемые в выражении
fillarray	Функция позволяет заполнять одноиндексные массивы третьего вида из списка
find_root	Функция находит корень уравнения на заданном интервале методом деления отрезка пополам
first	Функция выделяет первый элемент списка
float	Функция конвертирует любые числа в выражениях в числа машинной точности
fourier	Функция позволяет вычислить коэффициенты ряда Фурье
foursimp	Функция позволяет упростить коэффициенты ряда Фурье
fullratsimp	Функция вызывает функцию <i>ratsimp</i> до тех пор, пока выражение не перестанет меняться

Функция или переменная	Краткое описание
genmatrix	Функция возвращает матрицу заданной размерности, составленную из элементов индексного массива
gfactorsum	Функция представляет в виде сомножителей слагаемые выражения с комплексными числами
gfactor	Функция представляет в виде сомножителей выражение с комплексными числами
gradef	Функция определяет результат дифференцирования функции по своим аргументам
gramschmidt	Функция вычисляет ортонормированную систему векторов
ic1	Функция позволяет учесть начальное условие в решениях дифференциальных уравнений первого порядка
ic2	Функция позволяет учесть начальные условия в решениях дифференциальных уравнений второго порядка
ident	Функция возвращает единичную матрицу заданной размерности
ilt	Функция реализует обратное преобразование Лапласа
imagpart	Функция возвращает действительную часть выражения
integrate	Функция выполняет интегрирование заданного выражения по указанной переменной (неопределенная константа не добавляется). Можно также указать пределы интегрирования — в этом случае вычисляется определенный интеграл
invert	функция выполняет обращение матрицы
join	функция выполняет компоновку списков
kill	Функция уничтожает всю информацию (как свойства, так и присвоенное значение) об объекте или нескольких объектах
lagrange	Функция строит интерполяцию полиномом Лагранжа
lambda	создает лямбда-выражение (безымянную

Функция или переменная	Краткое описание
	функцию). Лямбда-выражение может использоваться в некоторых случаях как обычная функция
laplace	Функция реализует прямое преобразование Лапласа
last	Функция выделяет последний элемент списка
lbfgs	Функция осуществляет минимизацию ФНП
ldisplay	Функция печатает значения своих аргументов вместе с их именем и метками " ",
ldisp	Функция печатает значения своих аргументов вместе с метками " "
length	Функция возвращает длину списка
lhs	Функция выделяет левую часть уравнения
limit	функция осуществляет вычисление пределов
linearinterpol	Функция строит линейную интерполяцию
linsolve	Функция решает системы линейных и полиномиальных уравнений. Допускаются недоопределенные системы
listarray	Функция печатает содержимое массивов первого и второго видов
load	Функция загружает тот или иной файл: <i>load(some file)</i> ; Тип загрузки зависит от типа файла (макрос Maxima, программа на Lisp, бинарный файл)
logcontract	Функция компактифицирует логарифмы в данном выражении
make_array	Функция создает массивы третьего вида, содержимое которых печатается автоматически
makelist	Функция позволяет создавать списки
map	Функция применяет заданную функцию к каждому элементу списка
matrix	Функция возвращает матрицу, заданную поэлементно
matrixmap	Функция для заполнения матрицы значениями некоторой функции
mattrace	Функция вычисляет след матрицы (сумму ее

Функция или переменная	Краткое описание
	диагональных элементов)
max	перебирает свои аргументы и находит максимальное число
member	Функция возвращает <i>true</i> , если ее первый аргумент является элементом заданного списка, и <i>false</i> в противном случае
min	перебирает свои аргументы и находит минимальное число
minor	вычисляет миноры матрицы
mnewton	Функция находит корень системы уравнений многомерным методом Ньютона. Для использования функции необходимо сначала загрузить пакет mnewton
multthru	Функция умножает каждое слагаемое в сумме на множитель, причем при умножении скобки в выражении не раскрываются
newton	Функция находит корень указанной функции методом Ньютона
nroots	Функция, которая возвращает количество действительных корней полиномиального уравнения с действительными коэффициентами, которые локализованы в указанном интервале
num	Функция выделяет числитель
ode2	Функция решает дифференциальные уравнения первого и второго порядков
odelin	Функция решает однородные линейные уравнения первого и второго порядка, и возвращает фундаментальное решение ОДУ
pade	Функция аппроксимирует отрезок ряда Тейлора дробно-рациональной функцией
part	Функция позволяет выделить тот или иной элемент часть списка
plog	представляет основную ветвь комплексного логарифма
plot2d, wxplot2d	строит двумерные графики
plot3d, wxplot3d	строит трёхмерные графики
polarform	Функция приводит комплексное выражение к

Функция или переменная	Краткое описание
	тригонометрической форме
polyfactor	Переменная определяет форму выдачи функции <i>allroots</i>
powerseries	Функция строит разложение в степенной ряд
print	печатает значения всех своих аргументов в одну строку
product	Функция реализует цикл умножения
properties	Функция печатает свойства переменной
radcan	Функция упрощает выражения со вложенными степенями и логарифмами
ratepsilon	Переменная задает точность преобразования действительного числа в рациональное
ratexpand	Функция раскрывает скобки в выражении. Отличается от функции <i>expand</i> тем, что приводит выражение к канонической форме
ratfac	Переменная включает или выключает частичную факторизацию выражений при сведении их к CRE. Изначально установлено значение <i>false</i>
ratsimpexpons	Переменная управляет упрощением показателей степени в выражениях
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 <i>rat</i> . Повторный вызов функции может изменить результат, т.е. упрощение не идет до конца
ratsubst	Функция Реализует синтаксическую подстановку для рациональных выражений
ratvars	Функция позволяет изменить алфавитный порядок "главности" переменных, принятый по умолчанию
rat	Функция приводит выражение к каноническому представлению и снабжает его меткой <i>/R/</i> . Она упрощает любое выражение, рассматривая его как дробно-рациональную функцию, т.е.

Функция или переменная	Краткое описание
	работает с арифметическими операциями и с возведением в целую степень
realpart	Функция возвращает действительную часть комплексного выражения
read	основная функция для считывания вводимых пользователем выражений
read_matrix, read_list	функция для ввода массивов чисел
realroots	Функция выдает действительные корни полиномиального уравнения с действительными коэффициентами
rectform	Функция Приводит комплексное выражение к алгебраической форме
remarray	Функция уничтожает массив или массивы
remove	Функция удаляет свойство переменной
residue	Функция позволяет вычислять вычеты на комплексной плоскости
rest	Функция выделяет остаток после удаления первого элемента списка
reverse	Функция меняет порядок элементов в списке на обратный
rhs	Функция выделяет правую часть уравнения
romberg	Функция численно находит определенный интеграл функции на заданном отрезке. При этом используется алгоритм Ромберга
rk	Функция реализует метод Рунге-Кутты решения ОДУ
row	Функция выделяет заданную строку матрицы
save	сохраняет текущие значения рабочей области в файл
solve	Функция решает уравнения и системы уравнений
sort	Функция упорядочивает элементы списка
sublist	Функция составляет список из тех элементов исходного списка, для которых заданная логическая функция возвращает значение <i>true</i> .
submatrix	Функция выделяет из матрицы подматрицу

Функция или переменная	Краткое описание
subst	Функция Реализует синтаксическую подстановку
Sum	Функция реализует цикл суммирования
taylor	Функция Возвращает разложение функции в ряд Тейлора
tlimit	Функция отличается от функции <i>limit</i> только алгоритмом — она использует разложение выражения в ряд Тейлора
totalfourier	Функция позволяет вычислить построить ряд Фурье
translate	Функция транслирует функцию Maxima на язык LISP
transpose	Функция транспонирует матрицу
trigexpand	Переменная управляет работой функции <i>trigexpand</i>
trigexpand	Функция раскладывает все тригонометрические функции от сумм в суммы произведений тригонометрических функций
trigreduce	Функция свертывает все произведения тригонометрических функций в тригонометрические функции от сумм
trigsimp	Функция только применяет к выражению правило $\sin^2(x) + \cos^2(x) = 1$
trirat	Функция пытается свести выражение с тригонометрическими функциями к некому универсальному каноническому виду (в общем, пытается упростить выражение)
uniteigenvectors	Функция отличается от функции <i>eigenvectors</i> тем, что возвращает нормированные на единицу собственные вектора
writefile	Функция начинает запись выходных данных Maxima в указанный файл
write_matrix, write_list	функция для вывода массивов чисел
xthru	Функция приводит выражение к общему знаменателю, не раскрывая скобок и не

Функция или переменная	Краткое описание
	факторизуя слагаемые
zeromatrix	Функция возвращает матрицу заданной размерности, составленную из нулей
' '	Две одиночные кавычки "a" вызывают дополнительное вычисление в момент обработки a
'	Одиночная кавычка ' предотвращает вычисление

9.2 Перечень основных пакетов расширения Maxima

Наименование пакета	Краткое описание функций пакета
augmented_lagrangian	Минимизация функции нескольких переменных с ограничениями методом неопределённых множителей Лагранжа (используется совместно с lbfgs)
bode	Построение диаграмм Боде (узкоспециальный пакет)
contrib_ode	Дополнительные функции для аналитического решения обыкновенных дифференциальных уравнений
descriptive	Описательная статистика, оценка параметров распределения (генеральной совокупности) по выборке
diag	Пакет для операций с некоторыми видами диагональных матриц
distrib	Пакет, содержащий функции для расчёта различных распределений вероятностей и их параметров (нормальное распределение, распределение Стьюдента и т.п.)
draw	Интерфейс Maxima-Gnuplot. Предназначен для подготовки иллюстраций полиграфического качества
Dynamics	Различные функции, в т.ч. графические, относящиеся к моделированию динамических систем и фракталов
f90	Экспорт кода Maxima в код на Фортран90

Наименование пакета	Краткое описание функций пакета
ggf	Пакет включает единственную функцию, позволяющую оперировать с производящими функциями последовательностей (узкоспециальный пакет)
graphs	Пакет, включающий функции для работы с графами
grobner	Функции для того, чтобы работать с базисом Грёбнера (Groebner)
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, описывающих колебания
lsquares	Функции для оценки параметров различных зависимостей методом наименьших квадратов
makeOrders	Пакет включает одну функцию для операций с полиномами
mnewton	Метод Ньютона для решения систем нелинейных уравнений
numericalio	Чтение и запись файлов (преимущественно с матричными числовыми данными)
opsubst	Пакет содержит одну функцию <i>opsubst</i> , позволяющую выполнять замену в выражениях (по возможности мало отличается от <i>subst</i>)
orthopoly	Пакет, содержащий функции для операций с ортогональными полиномами (Лежандра, Чебышева и др.)
plotdf	Пакет, позволяющий строить поле направлений для решения автономных систем

Наименование пакета	Краткое описание функций пакета
	(интересный, но довольно узкоспециальный пакет)
romberg	Пакет, включающий ряд функций для численного интегрирования
simplex	Пакет, предназначенный для решения задач линейного программирования
solve_rec	Пакет, содержащий функции для упрощения рекуррентных выражений
stats	Пакет, включающий функции для статистической проверки гипотез (о равенстве математических ожиданий или дисперсий выборок и т.п.)
stirling	Расчёт гамма-функции
stringproc	Пакет, включающий функции для обработки строк
unit	Пакет, включающий функции для операций с единицами измерения
zeilberger	Функции для гипергеометрического суммирования

9.3 Список основных математических констант, доступных в Maxima

Обозначение в Maxima	Математическое содержание
%e	основание натуральных логарифмов
%i	мнимая единица ($\sqrt{-1}$)
inf	отрицательная бесконечность (на действительной оси)
infinite	бесконечность (на комплексной плоскости)
%phi	Золотое сечение (ϕ)
%pi	Постоянная π — отношение длины окружности к её диаметру
%gamma	Постоянная Эйлера (γ)
false, true	логические (булевы) величины

9.4 Список основных математических функций, доступных в Maxima

Обозначение в Maxima	Математическое содержание
abs	абсолютная величина
acos	арккосинус
acosh	обратный гиперболический косинус
acot	арккотангенс
acsc	арккосеканс
asec	арксеканс
asin	арксинус
asinh	обратный гиперболический синус
atan	арктангенс
atanh	обратный гиперболический тангенс
ceiling	округление до целого с избытком
cos	косинус
cosh	гиперболический косинус
cot	котангенс
csc	косеканс
exp	экспонента
fix	целая часть
float	преобразование к формату с плавающей точкой
floor	округление до целого с недостатком
log	натуральный логарифм
sec	секанс
sin	синус
sinh	гиперболический синус
sqrt	квадратный корень
tan	тангенс
tanh	гиперболический тангенс

Литература

- 1 Компьютерная математика с Maxima -
<http://www.intuit.ru/studies/curriculums/17584/courses/726/info>
- 2 Документация по текущей версии пакета -
<http://maxima.sourceforge.net/docs/manual/en/maxima.html>
- 3 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>
- 4 Статьи Тихона Тарнавского -
<http://maxima.sourceforge.net/ru/maxima-tarnavsky-1.html>
- 5 Методическое пособие по изучению математического пакета Maxima. Математический практикум с применением пакета Maxima.
<http://www.pmtf.msiu.ru/chair31/students/spichkov/maxima2.pdf>
- 6 Стахин Н. А. Основы работы с системой аналитических (символьных) вычислений МАХИМА (ПО для решения задач аналитических (символьных) вычислений). Москва: Федеральное агентство по образованию, 2008 — 86 с.
- 7 Книги по Maxima (электронное руководство)
<http://maxima.sourceforge.net/ru/documentation.html>
- 8 Gilberto E. Urroz
<http://www.neng.usu.edu/cee/faculty/gurro/Maxima.html>
- 9 Аладьев В. З. Системы компьютерной алгебры: Maple: искусство программирования М.: Лаборатория базовых знаний, 2006. — 792 с.
- 10 Васильев А. Н. Mathcad 13 на примерах СПб.: БХВ-Петербург, 2006. — 528 с.
- 11 Воробьев Е. М. Введение в систему символьных, графических и численных вычислений Mathematica 5 М.: ДИАЛОГ-МИФИ, 2005. — 368 с.
- 12 Цибулин В. Г., Говорухин В. Н. Введение в Maple. Математический пакет для всех М.: Мир, 1997. — 208 с
- 13 Гурский Вычисления в MathCAD Мн.: Новое знание, 2003. — 814 с.
- 14 Гурский Д. А., Турбина Е. Mathcad для студентов и школьников. Популярный самоучитель СПб.: Питер, 2005. — 400 с.
- 15 Дьяконов В. П. Maple 9 в математике, физике и образовании М.: СОЛОН-Пресс, 2004. — 688 с.
- 16 Дьяконов В. П. Справочник по MATHCAD PLUS 7.0. PRO М.: СК Пресс, 1998. — 352 с.
- 17 Очков В. Ф. Mathcad 12 для студентов и инженеров СПб.: БХВ-Петербург, 2005. — 464 с.
- 18 Советов Б. Я., Яковлев С. А. Моделирование систем М.: Высш. шк. — 2001. — 343 с
- 19 Морозов В. К., Рогачёв Г. Н. Моделирование информационных и динамических систем М.: ИЦ «Академия», 2011. — 384 с.