Федеральное государственное образовательное бюджетное
учреждение высшего профессионального образования
«Финансовый университет
при Правительстве Российской Федерации»

Кафедра «Математика 1»

Е.В. Маевский, П.В. Ягодовский

КОМПЬЮТЕРНАЯ МАТЕМАТИКА

Высшая математика в СКМ Maxima

Часть I. Введение

Учебное пособие

Москва • 2014

УДК 004.432.42
ББК 22.1, 32.973.26-018
М 13

Рецензенты:
Д.В. Берзин, к.ф.-м.н., доц.
(Финансовый университет)
П.К. Силаев, д.ф.-м.н., проф.
(МГУ им. М.В. Ломоносова)
Н.А. Стахин, к.ф.-м.н., доц.
(Томский государственный педагогический университет)

Маевский Е.В., Ягодовский П.В.

М 13 Компьютерная математика. Высшая математика в СКМ Maxima: учебное пособие. — М.: Финансовый университет, 2014. — 196 с.

ISBN 978-5-7942-1137-5
Ч. 1: Введение. — 196 с.
ISBN 978-5-7942-1178-8

В первой части пособия излагаются основы работы в системе компьютерной математики Maxima. Обсуждаются типы данных общего характера: логический тип, числа, строки, списки, множества и массивы, а также дается описание основных возможностей языка Maxima: условные операторы и циклы, построение пользовательских функций и операторов, работа с выражениями, инструментарий для построения графиков, файловый ввод-вывод и взаимодействие с Lisp.

Пособие ориентировано на студентов факультета «Прикладная математика и информационные технологии» Финансового университета и предназначено для проведения занятий и организации самостоятельной работы студентов, изучающих курс по выбору «Компьютерная математика».

Публикуется в авторской редакции.

УДК 004.432.42
ББК 22.1, 32.973.26-018

ISBN 978-5-7942-1178-8 (Ч. 1)
ISBN 978-5-7942-1137-5

Federal State — Funded Educational Institution
of Higher Professional Education
«Financial University
under the Government of the Russian Federation»

Chair «Mathematics-1»

E.V. Maevskiy, P.V. Jagodowscy

COMPUTER MATHEMATICS

Higher Mathematics in CAS Maxima
Part I. Introduction

Manual

Moscow • 2014

UDC 004.432.42

Reviewers:
D.V. Berzin, c.p.-m.s., doc.
(Financial University)
P.K. Silaev, d.p.-m.s., prof.
(Moscow State University)
N.A. Stakhin, c.p.-m.s., doc.
(Tomsk State Pedagogical University)

Maevskiy E.V., Jagodowscy P.V.

Computer mathematics. Higher mathematics in CAS Maxima: manual. — M.: Financial University, 2014. — 196 p.

ISBN 978-5-7942-1137-5
P. 1: Introduction. — 196 p.
ISBN 978-5-7942-1178-8

In the first part of the book the basics of working in the CAS Maxima are introduces. Discusses the general data types: boolean, numbers, strings, lists, sets, and arrays, as well as describes the main language features of Maxima: conditional statements and loops, construction of custom functions and operators, working with expressions, tools for charting, file input-output and interaction with Lisp.

The book is aimed at students of the Faculty «Applied Mathematics and Information Technology» of Financial University and is intended for training and self study organization of students of the elective course «Computer Mathematics».

Published in author's edition.

UDC 004.432.42

ISBN 978-5-7942-1178-8 (P. 1)
ISBN 978-5-7942-1137-5

Содержание

Contents

Предисловие

Пособие состоит из нескольких частей. Первая часть содержит описание основных команд и синтаксических конструкций системы компьютерной математики Maxima. Вторая часть будет посвящена примерам применения Maxima в решении задач элементарной и линейной алгебры, а также аналитической геометрии. В третьей части будут рассмотрены задачи математического анализа, включая обыкновенные дифференциальные уравнения.

Пособие ориентировано на студентов факультета «Прикладная математика и информационные технологии» и предполагает соответствующий уровень подготовки по математике и программированию. В области математики необходимо свободное владение материалом дисциплин «Линейная алгебра» и «Математический анализ». В области информационных технологий необходимы общие представления о программном коде и работе интерпретатора.

Известно, что научиться решать математические задачи можно только решая их. С другой стороны, для изучения языка программирования необходимо самостоятельно писать программы. Maxima совмещает в себе средство для решения математических задач и язык программирования, поэтому сказанное выше применимо здесь вдвойне. Для лучшего понимания и усвоения материала пособия рекомендуется как минимум самостоятельно набирать и запускать примеры из листингов.

Электронная версия пособия снабжена гиперссылками и примерами анимации. Гиперссылки выделены цветом, например: определение функции `sum` дано на стр. 61, использование функции `sum` показано в листинге 40. Ссылки на ресурсы в Интернете также активны. Для воспроизведения анимации рекомендуется использовать Adobe Acrobat Reader.

1. Системы компьютерной математики

Системы компьютерной математики (СКМ) предназначены для облегчения решения технически сложных математических задач. Они позволяют использовать компьютер для проведения рутинных операций: громоздких символьных или численных преобразований, трудоемких вычислений, перебора вариантов и т.п.

Поскольку процесс решения такой задачи должен быть строго алгоритмизован и, тем самым, задача фактически сведена к алгебраической, — то системы компьютерной математики называют также системами компьютерной алгебры (CAS — Computer Algebra System).

Системы компьютерной математики бывают универсальные и специализированные. Первые позволяют более или менее успешно решать задачи во всех основных разделах математики: алгебра, анализ, геометрия, теория вероятностей, численные методы. Вторые — иногда очень сильные в одном-двух разделах, оказываются вовсе бесполезными в других.

К сожалению, все системы пока далеки от совершенства. Например, решение уравнения 5-й степени

$$(x-3)^5 + 2(x+2)^5 = 0$$

очевидно хорошему школьнику. Тем не менее, авторам не известна СКМ, которая бы решила это уравнение. Впрочем, квалифицированный пользователь может «подсказать» системе путь решения задачи или запрограммировав нужный алгоритм, или проведя необходимые действия в интерактивном режиме.

Решение простейших задач с помощью СКМ не требует от пользователя особой математической квалификации. Однако, в серьёзных случаях, когда нужно подобрать адекватный метод решения, когда в ходе решения может возникнуть неожиданный результат, требующий должного истолкования, — без глубокого знания математики не обойтись.

1.1. Компьютерная математика

Системы компьютерной математики предоставляют исследователю две категории методов: символьные и численные.

Под символьными методами понимается умение системы работать с различными математическими объектами и выражениями, используя их свойства и не подменяя их без указания пользователя на какие-либо упро-

щения. При этом в системе должны быть реализованы необходимые алгоритмы преобразования и проверки тождественности объектов.

Рассмотрим несколько примеров таких математических объектов и действий, которые можно проводить в СКМ.

Целые и рациональные числа. Можно проводить арифметические действия с целыми числами и обыкновенными дробями, не выполняя никаких округлений. При этом результаты вычислений оказываются абсолютно точными. Величина целых чисел, с которыми система может работать, ограничена лишь возможностями компьютера и операционной системы.

Рациональное выражение. Можно поручить системе его упростить, привести к общему знаменателю, разложить на множители или на простейшие дроби. Любое выражение, содержащее иррациональности и переменные, рассматривается системой в первую очередь как рациональное относительно некоторых искусственных переменных. Например, выражение $y = \dfrac{\sqrt{3}\ln x - 5\sin^2 x}{\operatorname{arctg} 2 + \ln^2 x}$ является рациональным относительно переменных: $t_1 = \sqrt{3}$, $t_2 = \sin x$, $t_3 = \ln x$, $t_4 = \operatorname{arctg} 2$; имеем $y = \dfrac{t_1 t_3 - 5t_2^2}{t_4 + t_3^2}$.

Иррациональные числа и выражения. Система работает с иррациональными числами $\sqrt{2}$, $\sin 3$, $\operatorname{arctg} 4$, $\ln 5$ и т.п. также как с переменными a, b, c и т.п. Разница между $a = \sqrt{2}$ и неопределенной переменной a состоит в том, что в первом случае $a^2 = 2$. При этом точная форма числа $\sqrt{2}$ и его приближённое значение 1.4142 — не одно и то же. Можно выполнять арифметические операции и «вычислять» элементарные функции от выражений, содержащих иррациональности. Система умеет преобразовывать выражения с основными элементарными функциями и понимает, например, что $5\sqrt{2}\sqrt{6} = 10\sqrt{3}$ и $\ln(e^5) = 5$. При этом всегда можно поручить СКМ вычислить приближённое значение выражения с иррациональностями и выдать результат в виде конечной десятичной дроби.

Выражения и функции. В системе можно работать с выражениями, содержащими неопределенную переменную, а можно задавать пользовательские функции. Хотя синтаксически это не одно и тоже, но принцип работы одинаков: в любом случае обрабатывается символьное выражение, а не массив числовых значений. Речь может идти о всевозможных подстановках и заменах переменных, об операциях математического анализа (предел, производная, интеграл), либо о более сложных и изощренных процедурах.

С помощью символьных методов можно искать точные решения задачи, а можно — точно находить приближения к решению задачи. Напри-

мер, приближенное решение дифференциального уравнения может быть получено в виде частичной суммы бесконечного ряда, при этом для каждого члена ряда может быть найдено точное выражение.

Численные методы оперируют не математическими выражениями и объектами, а их числовыми значениями. Таким образом, для применения численных методов к функции $f : X \to Y$ необходимо лишь задать массив её значений $y_i = f(x_i)$. Здесь отсутствуют алгоритмы преобразования выражений, поэтому численные методы проще символьных.

В численных алгоритмах используются числа с плавающей точкой, поэтому точно могут быть представлены лишь рациональные числа, сводящиеся к конечным двоичным дробям. Например, операция присвоения $0.2 \to x$ будет выполнена приближенно, ведь число 0.2 не представимо конечной двоичной дробью. Численные методы всегда приближенные, поскольку даже при использовании алгоритма, дающего точное решение, в процессе его применения появятся ошибки округления. Поэтому их целесообразно использовать только при решении задач, для которых или не существует символьных методов вообще или таковые в настоящее время не известны или известные алгоритмы недопустимо трудоёмкие. Лишь в этих случаях можно мириться с недостатками численных методов.

Можно предложить следующую классификацию задач, которые могут быть поставлены перед СКМ.

1. Задача, решаемая формально по известному алгоритму. Например:
   - решение квадратного уравнения,
   - решение системы линейных уравнений методом Гаусса,
   - вычисление производной элементарной функции,
   - статистические вычисления.
2. Задача, для решения которой не известно алгоритма, либо он сильно разветвлен. Например:
   - решение уравнения 5-й степени,
   - решение иррационального уравнения,
   - вычисление предела,
   - вычисление интеграла от элементарной функции,
   - решение дифференциального уравнения I-го порядка.
3. Задача, не имеющая символьного решения. Например:

- корни уравнения $x^5 - 9x - 3 = 0$ не выражаются через радикалы,
- интеграл $\int e^{-x^2}\,dx$ не вычисляется в элементарных функциях,
- дифференциальное уравнение $y' = y^2 + x^2$ не решается в квадратурах.

В 1-м или 2-м случае СКМ решит задачу, если в ней запрограммирован соответствующий алгоритм или необходимая ветка общего алгоритма. В хороших СКМ реализованы алгоритмы для решения большинства самых распространенных задач.

В 3-м случае СКМ сможет решить задачу доступными ей приближенными методами (асимптотическими или численными). Выбор метода, скорее всего, останется за пользователем, для этого ему понадобится математическая подготовка.

1.2. Обзор известных СКМ

Ниже перечислены в основном самые известные и многофункциональные СКМ. Вместе с этим для решения более узких задач существует необозримое число специальных программ. Ссылки, а также информацию о разнообразных СКМ можно найти на страницах сайта
http://www.computeralgebra.nl/.

Данная ниже информация в основном заимствована из Википедии и с официальных сайтов соответствующих программ.

1.2.1. Универсальные СКМ

Maxima
http://maxima.sourceforge.net/
http://maxima-online.org/

Maxima — потомок системы Macsyma, разработанной в начале 60-х в Массачусетском технологическом институте (MIT). С 1982 по 2001 год работу над Macsyma вел Уильям Шелтер. В 1998 году он получил разрешение на преобразование коммерческого продукта Macsyma в свободное программное обеспечение Maxima (существующее под лицензией GNU GPL). После его ухода в 2001 году была сформирована группа пользователей и разработчиков, ставящая своей целью донести Maxima до широкой аудитории.

Axiom
http://axiom-developer.org/

http://www.open-axiom.org/
http://fricas.sourceforge.net/

Axiom — свободная система компьютерной алгебры общего назначения. Разработка системы была начата в 1971 году группой исследователей IBM под руководством Ричарда Дженкса. Изначально система называлась Scratchpad. Проект развивался медленно и в основном рассматривался как исследовательская платформа для разработки новых идей в вычислительной математике. В 90-х система была продана компании Numerical Algorithms Group (NAG), получила название Axiom и стала коммерческим продуктом. Но по ряду причин система не получила коммерческого успеха и была отозвана с рынка в октябре 2001. NAG решила сделать Axiom свободным программным обеспечением и открыла исходные коды под модифицированной лицензией BSD. В 2007 у Axiom появились два ответвления с открытым исходным кодом: OpenAxiom и FriCAS.

Reduce
http://reduce-algebra.com/

Reduce — свободная система для инженерных и научных алгебраических вычислений. Она была создана совместными усилиями многих участников. К основным возможностям системы относятся: преобразования многочленов и рациональных функций, поиск и замена в выражениях, автоматическое и настраиваемое упрощение выражений, матричные вычисления, целые числа и числа с плавающей точкой произвольной точности, операции с пользовательскими функциями, символьное дифференцирование и интегрирование, решение алгебраических уравнений.

Yacas
http://yacas.sourceforge.net/

Yacas (Yet Another Computer Algebra System) — свободная система компьютерной алгебры общего назначения. Насколько известно авторам данного пособия, это единственная серьезная СКМ, реализованная в виде java-апплета и поэтому допускающая встраивание в web-страницу. К сожалению, проект не поддерживается с ноября 2009.

Maple
http://www.maplesoft.com/products/Maple/

Maple — платная система. Является продуктом компании Waterloo Maple Inc., которая с 1984 года выпускает программные продукты, ориентированные на сложные математические вычисления, визуализацию данных и моделирование. Система Maple предназначена для символьных

вычислений, хотя имеет ряд средств и для численного решения дифференциальных уравнений и нахождения интегралов. Обладает развитыми графическими средствами.

Mathematica
http://www.wolfram.com/mathematica/

Mathematica — платная система компьютерной алгебры компании Wolfram Research. Содержит множество функций как для символьных преобразований, так и для численных расчётов. Кроме этого, программа поддерживает работу с графикой и звуком, включая построение двух- и трёхмерных графиков функций, рисование произвольных геометрических фигур, импорт и экспорт изображений и звука.

Scientific WorkPlace
http://www.mackichan.com/

Scientific WorkPlace — платный пакет для набора научных текстов. Представляет собой визуальный текстовый процессор, базирующийся на системе LaTeX, со встроенной системой компьютерной алгебры (использует символьное ядро Maple или MuPad). Позволяет набирать в визуальном режиме математические тексты с формулами, проводить несложные символьные преобразования и строить графики в единой рабочей области. Использовался в учебном процессе некоторыми преподавателями кафедры «Математика» Финансового университета[1].

Свободно-распространяемый аналог: LyX (http://www.lyx.org/)

1.2.2. Алгебра

GAP
http://www.gap-system.org/

GAP (Groups, Algorithms, Programming) — свободно распространяемая на условиях лицензии GNU GPL кроссплатформенная система компьютерной алгебры для вычислительной дискретной алгебры с особым вниманием к вычислительной теории групп. Совместная разработка университетов Сент-Эндрюс (Шотландия), Ахен (с 1986), Брауншвейг (Германия) и университета штата Колорадо (США).

Magma
http://magma.maths.usyd.edu.au/magma/

[1] *Бабайцев В.А.* Материалы к спецкурсу «Компьютерная математика», Москва, ФА, 2001.

Magma — платный программный продукт, разработанный для вычислений в алгебре, теории чисел, алгебраической геометрии и алгебраической комбинаторике. Magma предоставляет математически строгую среду для работы с такими структурами как группы, кольца, поля, модули, алгебры, схемы, кривые, графы и многими другими. Magma разрабатывается Computational Algebra Group Университета Сиднея.

1.2.3. Теория вероятностей

R

http://www.r-project.org/

R — язык программирования для статистической обработки данных и работы с графикой, а также свободная программная среда вычислений с открытым исходным кодом в рамках проекта GNU. Язык создавался как аналогичный языку S, разработанному в Bell Labs и является его альтернативной реализацией. Изначально R был разработан сотрудниками статистического факультета Оклендского университета Россом Айхэкой и Робертом Джентлменом (первая буква их имён — R), на момент 2011 года язык и среда поддерживаются и развиваются организацией R Foundation.

Statistica

http://www.statsoft.com/

Statistica — платный пакет для всестороннего статистического анализа, разработанный компанией StatSoft. В пакете Statistica реализованы процедуры для анализа данных, управления данными, получения данных и визуализации данных. Пакет Statistica имеет модульную структуру. Каждый модуль содержит уникальные процедуры и методы.

MatCalc

http://www.matcalc.ru/

MatCalc — разработанный сотрудником кафедры теории вероятностей Финансового университета профессором А.В. Браиловым интерпретатор языка матричных вычислений, содержащий встроенную библиотеку статистических функций. Применяется в учебном процессе[2] при выполнении лабораторных и курсовых работ. Распространяется свободно.

[2] *Браилов А.В.* Опыт применения «Матричного калькулятора» на практических занятиях по математической статистике. В сб. Компетентностный подход в высшем образовании: материалы межвузовской методической конференции (Москва, декабрь 2009 г.).– М.: Альфа-М, 2010. с. 81-85.

1.2.4. Численные методы

Octave

http://www.gnu.org/software/octave/

Octave — свободная система для математических вычислений, использующая совместимый с MatLab язык высокого уровня. Octave представляет интерактивный командный интерфейс для решения линейных и нелинейных математических задач, а также проведения других численных экспериментов. Язык Octave оперирует арифметикой вещественных и комплексных чисел и матриц, имеет расширения для решения задач линейной алгебры, нахождения корней систем нелинейных алгебраических уравнений, работы с полиномами, решения различных дифференциальных уравнений, интегрирования систем дифференциальных и дифференциально-алгебраических уравнений первого порядка, интегрирования функций на конечных и бесконечных интервалах.

SciLab

http://www.scilab.org/

Scilab — свободный пакет прикладных математических программ, предоставляющий мощное открытое окружение для инженерных и научных расчётов. С 1994 года распространяется вместе с исходным кодом через Интернет. В 2003 году для поддержки Scilab был создан консорциум Scilab Consortium. Сейчас в него входят 25 участников, в том числе Mandriva, INRIA и ENPC (Франция). Scilab содержит сотни математических функций, и есть возможность добавления новых, написанных на различных языках (C, C++, Fortran и т.д.). Также имеются разнообразные структуры данных (списки, полиномы, рациональные функции, линейные системы), интерпретатор и язык высокого уровня.

Matlab

http://www.mathworks.com/products/matlab/

MatLab (Matrix Laboratory) — платный пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете. MatLab как язык программирования был разработан Кливом Моулером в конце 1970-х годов. Инженер Джон Литтл познакомился с этим языком во время визита Клива Моулера в Стэнфордский университет в 1983 году. Поняв, что новый язык обладает большим коммерческим потенциалом, он объединился с Кливом Моулером и Стивом Бангертом. Совместными усилиями они переписали MatLab на C (изначально код был на Фортране) и основали в 1984 компа-

нию The MathWorks для дальнейшего развития. Первоначально MatLab предназначался для проектирования систем управления (основная специальность Джона Литтла), но быстро завоевал популярность во многих других научных и инженерных областях.

MathCad

http://www.ptc.com/product/mathcad/

MathCad — платная система компьютерной алгебры из класса систем автоматизированного проектирования, ориентированная на подготовку интерактивных документов с вычислениями и визуальным сопровождением. Mathcad был задуман и первоначально написан Алленом Раздовом из Массачусетского технологического института. Mathcad имеет простой и интуитивный для использования интерфейс пользователя. Для ввода формул и данных можно использовать как клавиатуру, так и специальные панели инструментов. Работа осуществляется в пределах рабочего листа, на котором уравнения и выражения отображаются графически, в противовес текстовой записи в языках программирования. Символьные вычисления в Mathcad основаны на подмножестве системы компьютерной алгебры Maple.

1.2.5. Библиотеки для C

GiNaC

http://www.ginac.de/

GiNaC (GiNaC is Not a CAS) — это свободная C++ библиотека, позволяющая создавать математические программы, использующие символьные преобразования. Распространяется по лицензии GPL. В пакет включена программа Ginsh, являющаяся примером CAS, построенной на Ginac. Библиотека была специально разработана, чтобы служить заменой движку Xloops, до сих пор поддерживаемому Maple. В отличие от остальных CAS, Ginac не предоставляет возможностей символьных преобразований и простой язык программирования, а расширяет имеющийся язык (C++), снабжая его новыми классами и функциями для алгебраических вычислений.

GMP

http://gmplib.org/

GNU MP — это свободная библиотека, написанная на C, предназначенная для работы с целыми произвольного размера, точными рациональными числами и числами с плавающей точкой произвольной точности.

Корректно обрабатывает числа, содержащие от сотен до миллионов знаков, выбирает для каждого случая подходящий алгоритм.

1.3. *Задания для самостоятельной работы*

1.1 Имеется квадратное уравнение с целыми коэффициентами. Напишите его символьное решение. Предложите какой-либо численный алгоритм для приближенного решения квадратного уравнения на машине, умеющей производить арифметические действия, но не умеющей извлекать корень.

1.2 Пусть $f(x)$ непрерывна на отрезке $[a, b]$. Напишите алгоритм приближенного решения уравнения $f(x) = 0$ на отрезке $[a, b]$ *методом деления отрезка пополам*.

1.3 Имеется система линейных уравнений с целыми коэффициентами. Напишите алгоритм точного решения этой системы *методом Гаусса*. Операцию деления при этом использовать нельзя, поскольку тогда возникнут числа с плавающей точкой и точного решения не получится. Можно использовать лишь сокращение уравнения на общий множитель всех коэффициентов. Рациональные числа должны появится лишь на последнем этапе как результаты формального деления коэффициентов уравнения на разрешающий элемент.

1.4 Напишите алгоритм вычисления определителя матрицы:

- разложением по строке (столбцу);
- методом Гаусса.

Какой метод потребует меньшего числа операций?

1.5 Напишите алгоритм вычисления ранга матрицы методом Гаусса.

1.6 Точным решением системы уравнений

$$\left(\begin{array}{ccc|c} 1 & \frac{1}{2} & \frac{1}{3} & -\frac{2}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & -\frac{2}{15} \end{array}\right)$$

является $x = (-1, 0, 1)$. Решите эту систему методом Гаусса в числах с плавающей точкой, на каждом шаге округляя число до 3-го знака. Получится ли ответ с точностью до 3-го знака? Проверьте, что определитель

матрицы этой системы $\Delta = \frac{1}{2160}$. Используя формулы Крамера, оцените точность вычисления x когда каждый элемент матрицы задан с точностью 10^{-3}.

1.7 Пусть $\vec{a}_1, \ldots, \vec{a}_n$ — базис евклидова пространства. Напишите алгоритм ортогонализации базиса методом Грамма-Шмидта.

1.8 Имеется квадратичная форма от произвольного числа переменных. Напишите алгоритм, приводящий квадратичную форму к нормальному виду методом Лагранжа.

1.9 Напишите алгоритм, реализующий «деление в столбик» одного многочлена на другой (с остатком).

1.10 Напишите алгоритм вычисления наибольшего общего делителя (НОД) двух многочленов.

1.11 Математическое выражение в *польской нотации* записывается по следующему правилу: сначала имя функции или оператора, затем список аргументов (операндов). Например, выражение

$$\frac{\sin(x+5) - 2x}{x^2 + 7} - 5e^x$$

будет записано в виде[3]

```
["+",
  ["/",
    ["+",[sin,["+",x,5]],["*",-2,x]],
    ["+",["^",x,2],7]
  ],
  ["*",-5,["^",e,x]]
]
```

Польская нотация (равно как и *обратная польская нотация*) удобна для внутренней машинной обработки выражений и используется во всех СКМ, реализующих символьные вычисления. Потренируйтесь записывать выражения в польской нотации.

1.12 (Продолжение) Напишите алгоритм символьного дифференцирования элементарной функции, записанной в польской нотации.

1.13 *Метод Остроградского* применяется при символьном интегрировании рациональной функции. Он состоит в выделении рациональной ча-

[3] Отступы и переносы строк добавлены для удобочитаемости.

сти первообразной:

$$\int \frac{P(x)}{Q(x)}\,dx = \frac{P_1(x)}{Q_1(x)} + \int \frac{P_2(x)}{Q_2(x)}\,dx.$$

Здесь все рациональные дроби предполагаются правильными, $Q_1(x)$ есть наибольший общий делитель многочленов $Q(x), Q'(x)$, а многочлен $Q_2(x) = \frac{Q(x)}{Q_1(x)}$ не имеет кратных корней. Изучите[4] метод Остроградского.

1.14 Имеется правильная рациональная дробь

$$\frac{a_m x^m + \ldots + a_1 x + a_0}{(x - x_1)(x - x_2)\ldots(x - x_n)},$$

где $m < n$, а числа $x_1 < x_2 < \ldots < x_n$ попарно различны. Напишите алгоритм разложения такой дроби в сумму *простейших дробей*.

1.15 Пусть $R(u, v)$ — рациональная функция. Напишите алгоритм сведения интеграла $\int R(\sin x, \cos x)\,dx$ к интегралу от рациональной функции $\int Q(t)\,dt$.

1.16 Пусть $R(u, v)$ — рациональная функция, a, b, c — числа. Напишите алгоритм сведения интеграла $\int R(x, \sqrt{ax^2 + bx + c})\,dx$ к интегралу от рациональной функции $\int Q(t)\,dt$.

1.17 Пусть $m, n, p \in \mathbb{Q}$, $ab \neq 0$. Напишите алгоритм сведения интеграла $\int x^m (ax^n + b)^p\,dx$ к интегралу от рациональной функции $\int Q(t)\,dt$.

[4] См., например, *Ильин В.А., Позняк Э.Г.* Основы математического анализа, Часть I. М.: Физматлит, 2005

2. Maxima и wxMaxima

Данное пособие предназначено научить студентов использовать СКМ для решения своих учебных, научных и профессиональных задач. Для иллюстрации возможностей систем компьютерной математики авторы выбрали систему Maxima. Этот выбор основан на следующих соображениях.

Во-первых, это бесплатная система, поэтому не возникнет ситуации, когда часть студентов не сможет выкупить необходимой для работы с СКМ лицензии.

Во-вторых, это достаточно мощная система, обладающая широкими возможностями, достаточными в большинстве случаев. Конечно, существуют и более мощные системы с гораздо более богатыми библиотеками реализованных алгоритмов (например, Maple), но они весьма дорогостоящие.

В-третьих, общие принципы работы этой системы сходны с рядом других систем, так что при необходимости вполне возможно переключиться с Maxima на другую СКМ. Вместе с тем, можно отметить, что Maxima как язык программирования имеет ряд своеобразных особенностей, делающих его крайне любопытным с точки зрения «компьютерной лингвистики».

Наконец, Maxima — это система с открытым кодом.

По некоторым своим возможностям Maxima уступает коммерческим СКМ, таким как Maple и Mathematica. Например, в Maxima не реализован ни один геометрический алгоритм. Но возможности этой системы легко расширяются пользовательскими библиотеками функций, создание которых сильно облегчается открытостью кода.

Maxima распространяется с исходным кодом, написанном на Lisp, поэтому пользователь может открыть любой из библиотечных файлов и изучить алгоритм, по которому работает та или иная функция, а в случае необходимости создавать свои функции, в том числе используя исходный код функций, входящих в комплект поставки системы.

Авторы подчеркивают, что не ставили перед собою цели написать исчерпывающее руководство по Maxima. Поэтому далеко не все функции и даже не все конструкции языка Maxima будут рассмотрены. Наша задача была показать работу с СКМ на примере использования системы Maxima для решения ряда задач высшей математики.

Мы будем рассматривать версию Maxima под Windows. Реализации Maxima как СКМ под разные операционные системы принципиально ничем друг от друга не отличаются. Графическая оболочка wxMaxima, кото-

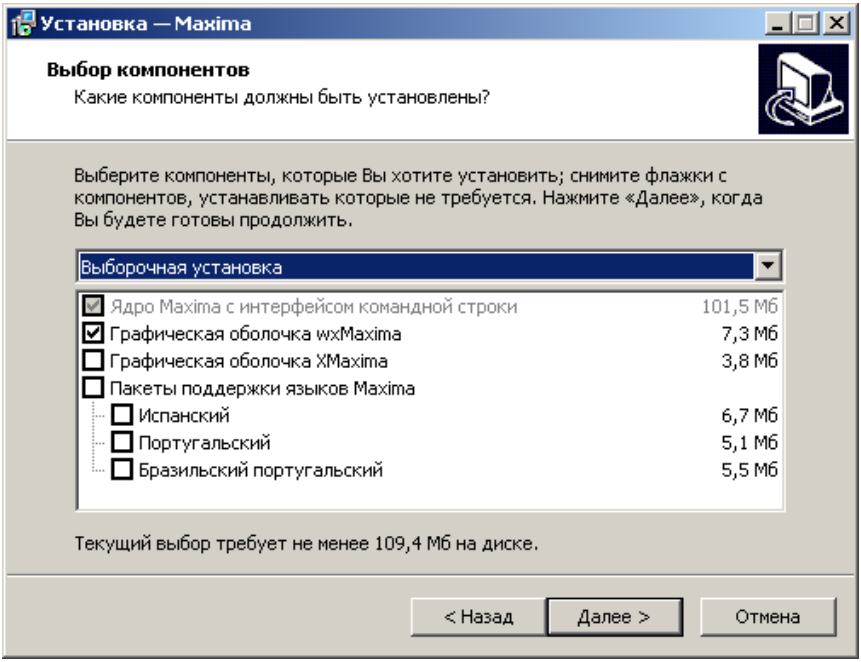
рую мы рекомендуем использовать нашим читателям, имеется для операционных систем: Windows, X11 и Mac OS. Имеется также версия Maxima для операционной системы Android, обладающая рядом особенностей.

В Maxima можно работать online на веб-сайте http://maxima-online.org/. Но по удобству работы веб-интерфейс сильно уступает графической оболочке wxMaxima. Кроме этого, скорость работы существенно зависит от суммарной нагрузки на сервер. Поэтому лучше установить Maxima на локальный компьютер, если есть такая возможность.

2.1. Установка Maxima

Дистрибутив Maxima следует скачать с официального сайта http://maxima.sourceforge.net/. Он представляет собой единственный файл формата `exe`. Дистрибутив той версии Maxima, которой пользовались авторы при написании этого пособия, называется `Maxima-5.26.0.exe`.

Установка состоит из нескольких стандартных шагов. В пункте `Выбор компонентов` следует обязательно выбрать графическую оболочку wxMaxima. Остальные параметры установки могут быть выбраны по усмотрению пользователя.



В результате установки получаем папку с установленной программой (по умолчанию `C:\Program Files\Maxima-5.26.0`), а также ярлыки:

 консольная версия Maxima;

 wxMaxima.

Папка установки содержит следующие подпапки: `bin`, `gnuplot`, `include`, `info`, `lib`, `libexec`, `share`, `uninst`, `wxMaxima`. Остановимся подробнее на некоторых из них:

- `bin` — содержит файл `maxima.bat`, запускающий консольную версию Maxima;
- `gnuplot` — содержит Gnuplot — мощную программу для визуализации данных, существующую независимо от Maxima (http://www.gnuplot.info/) и распространяющуюся также под лицензией GNU GPL;
- `share` — содержит файл справки (`maxima\5.26.0\doc\chm\maxima.chm`), встроенные и дополнительные пакеты функций (`maxima\5.26.0\share`, `maxima\5.26.0\src`), файлы с примерами (`maxima\5.26.0\demo`, `maxima\5.26.0\tests`);
- `wxMaxima` — содержит графическую оболочку wxMaxima. Официальный сайт оболочки: http://andrejv.github.com/wxmaxima/.

Можно сделать Maxima портабельной — для этого надо добавить в текстовый файл `maxima.bat` следующие строки[5]

Листинг 1

```
cd ..
set maxima_prefix=%cd%
```

После этого можно будет просто скопировать папку установки на USB-диск или вообще на диск другого компьютера и запускать систему оттуда.

В папке `share` находятся файлы следующих типов:

- `.lisp`, `.lsp` — исполняемые встроенным в Maxima интерпретатором языка Lisp;

[5] Редактируйте `maxima.bat` только если у вас есть опыт в написании bat-скриптов и вы уверены в том, что делаете.

- `.mac`, `.dem` — написанные на языке Maxima.

Все файлы — текстовые, поэтому их можно открывать и изучать в любом текстовом редакторе. Назначение файлов разнообразно: в некоторых описаны дополнительные функции Maxima, другие предназначены для тестирования работы новых функций или алгоритмов. В любом случае эти файлы могут служить источником прекрасных примеров, поскольку написаны профессиональными математиками и программистами.

Все функции Maxima делятся на две группы: *встроенные* — принадлежащие собственно системе, и *дополнительные*. Дополнительные функции разделены на *пакеты*. Пользователи могут писать и присоединять к системе свои пакеты.

При запуске графической оболочки wxMaxima часть пакетов загружается автоматически, содержащиеся в них функции сразу же доступны пользователю. Для использования дополнительных функций из других пакетов, надо загрузить соответствующий пакет командой `load`(*pack*), где *pack* — название пакета.

В консольной версии нет предварительно загружаемых пакетов — возможность работать с дополнительными функциями без загрузки пакета есть только в wxMaxima.

В версию Maxima 5.26.0 входит 67 пакетов. На страницах этого пособия будут рассмотрены некоторые из них.

Пакеты общего назначения:

- `stringproc` — работа со строками;
- `numericalio` — чтение и запись файлов;
- `functs` — некоторые функции из различных областей математики;
- `finance` — некоторые финансовые вычисления;
- `draw` — рисование в Gnuplot.

Алгебра и геометрия:

- `to_poly_solve` — решение уравнений и систем уравнений;
- `mnewton` — решение системы уравнений методом Ньютона;
- `linearalgebra` — многие функции линейной алгебры;
- `lapack` — некоторые функции из линейной алгебры;

- `eigen` — собственные значения и собственные векторы матриц;
- `diag` — жорданова форма матрицы;
- `simplex` — симплекс-метод
- `graphs` — работа с графами.

Математический анализ:

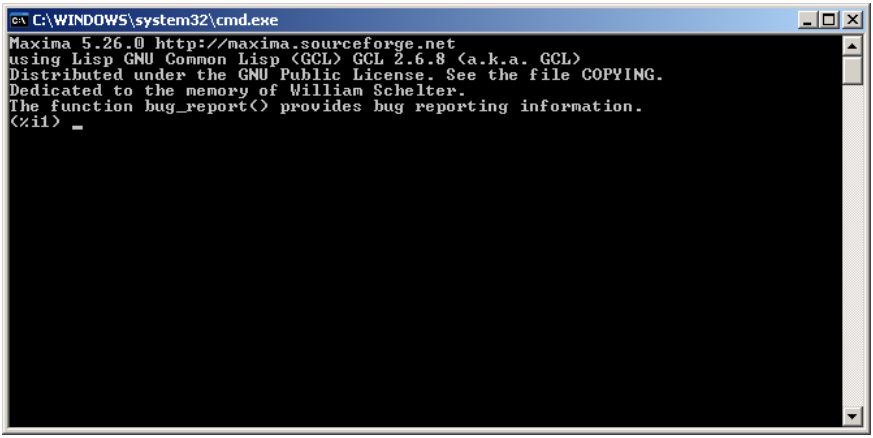
- `impdiff` — производные неявной функции;
- `simplify_sum` — вычисление сумм и рядов;
- `drawdf, plotdf` — поле направлений ОДУ 1-го порядка.

Теория вероятностей и статистика:

- `distrib` — стандартные распределения (дискретные и непрерывные) и их характеристики;
- `stats` — статистические функции.

2.2. Первый запуск

Запустим консольную версию Maxima `maxima.bat`.



Видим окно консоли и 1-ю метку ввода (`%i1`). Maxima нумерует вводимые пользователем выражения (input) *метками ввода* (`%iN`) и возвращаемые

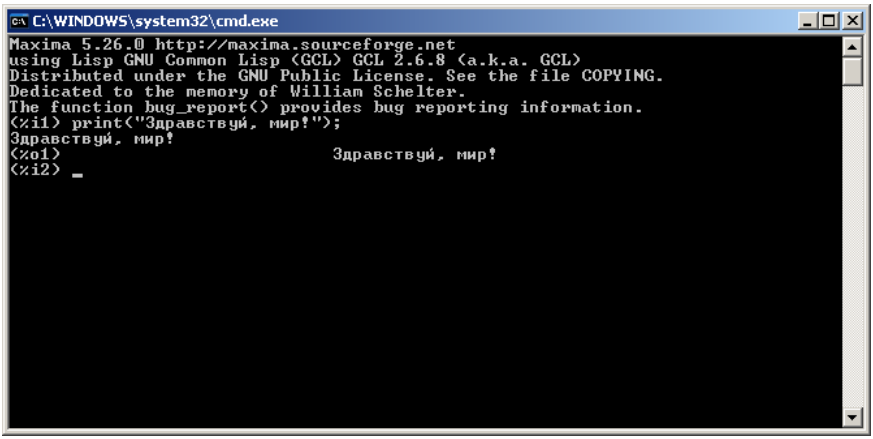
результаты (**o**utput) — *метками вывода* (`%oN`), где N — порядковый номер выражения.

Знакомство с языками программирования, к каковым с некоторыми оговорками можно отнести и Maxima, принято начинать с программы, печатающей строку «Hello, world!». Для этого нам понадобится следующая функция:

♦ `print`(*str*) — печатает строку *str* и ее же возвращает в качестве значения.

Каждый оператор и каждая функция языка Maxima, помимо выполнения своего прямого дела, обязательно *возвращает некоторое значение*[6]. Если выражение заканчивается *терминальным символом* `;` — то метка вывода и возвращаемое значение будут показаны, если терминальным символом `$` — то метка вывода и возвращаемое значение показаны не будут.

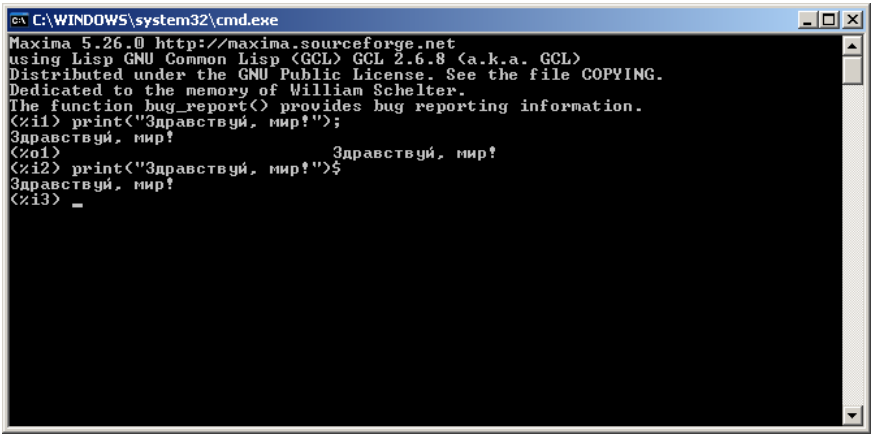
Выполнение выражения вызывается нажатием Enter и срабатывает только после ввода терминального символа, — в противном случае Enter переводит на новую строку. Например так:



В примере команда `print` первым делом напечатала сообщение — мы видим это в строке, следующей за строкой (%i1). А затем, возвратила своё значение, представляющее собою то же самое сообщение. Так как терминальный символ не отменил печать финального значения, то последнее было тоже выведено на экран под меткой (%o1). Сообщение, напечатанное

[6] Другими словами, в Maxima нет чистых процедур, которые нечто делают и при этом ничего не возвращают. Например, даже в результате выполнения цикла возвращается строка `done`.

два раза, выглядит нелепо, поэтому попробуем тоже самое но с другим терминальным символом:



Работа в консоли сопряжена с некоторыми трудностями:

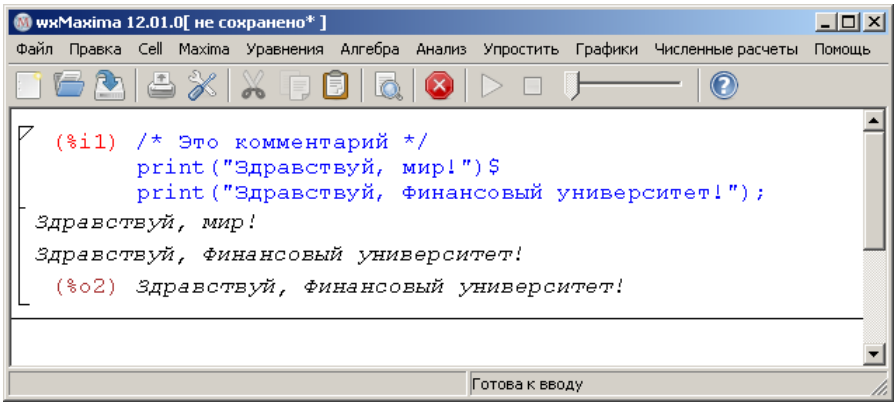
- затруднено копирование и вставка;
- нельзя вернуться назад.

Поэтому закончим на этом знакомство с консольной версией Maxima и запустим графическую оболочку wxMaxima.

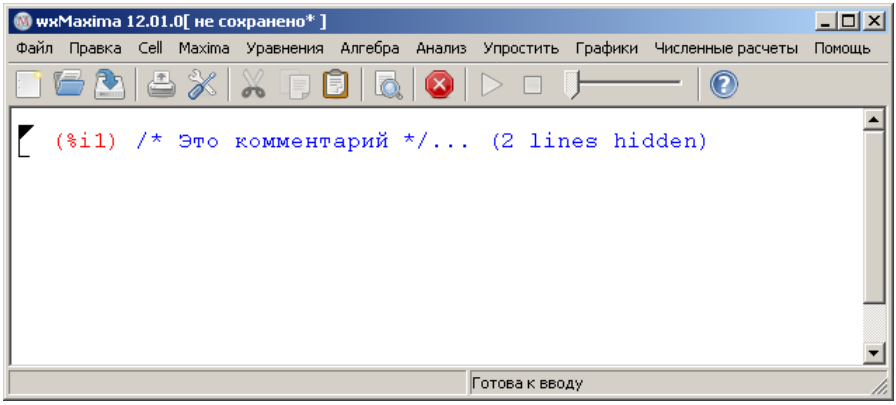
Область ввода пуста, но если набрать с клавиатуры какой-нибудь символ, то появится приглашение `-->` и набранный символ. Набрав одно выражение (с терминальным символом), нажмем Enter — перейдем на новую строку и наберем еще одно выражение.

Интерпретатор вызывается нажатием Shift+Enter. После вызова интерпретатора всем введенным выражениям будут присвоены порядковые номера, но метка ввода будет показана только у первого выражения. Выражения будут выполняться в том порядке, в котором они записаны.

Например, напишем так:



Строки, ограниченные квадратной скобкой слева, образуют *блок*. Блок разделен на две части: *блок ввода* и *блок вывода*. Блок ввода может содержать любое количество выражений. Если блок занимает на экране слишком много места, то его можно свернуть, щелкнув мышью на треугольнике в верхнем углу квадратной скобки.



Текст, набранный в скобках `/*  */`, называется *комментарием* и интерпретатором игнорируется. Расположение комментария произвольно с единственной оговоркой: блок ввода не должен на нем заканчиваться.

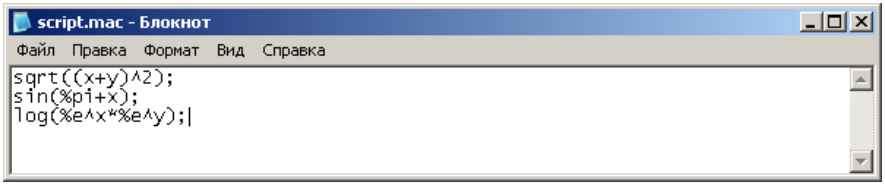
В дальнейшем мы не будем приводить примеры работы скриптов на скриншотах, а будем пользоваться следующим визуально аналогичным форматом записи:

Листинг 2

```
(%i1)  /* Это комментарий */
       print("Здравствуй, мир!")$
       print("Здравствуй, Финансовый университет!");
```

Здравствуй, мир!
Здравствуй, Финансовый университет!
(%o2) *Здравствуй, Финансовый университет!*

Еще один альтернативный способ работы с Maxima состоит в том, чтобы набирать текст скрипта в любом текстовом редакторе (например, в блокноте). Назовем текстовый файл `script.mac` и сохраним его в папке `D:/Compmath`.



После этого в консоли или в окне wxMaxima введем команду `batch`(*file*), где *file* — полное имя файла с путем.

Листинг 3

```
(%i1)  batch("D:/Compmath/script.mac")$
```

read and interpret file: #pD:/Compmath/script.mac

(%i2) $\sqrt{(y+x)^2}$
(%o2) $|y+x|$
(%i3) $\sin(x+\pi)$
(%o3) $-\sin(x)$
(%i4) $\log(\%e^x\,\%e^y)$
(%o4) $y+x$

Оболочка wxMaxima может выводить результат в трех *форматах вывода*: `xml`, `ascii` и `none`. По умолчанию — `xml`.

♦ Функция

`set_display`(*type*)

задает формат вывода результатов. Эта же функция продублирована в меню: `Maxima` → `Change 2d Display`.

Сравните различные форматы вывода:

Листинг 4

```
(%i1)  set_display(none)$
       1234^56;
       sqrt(2);
       (3*x+2)*5^x/(2^x-1);

(%o2) 12991190255487145194103208439623513775465782010127392384379012704624259433055094648925678485362472902010613951564738491094492118652386584905627535906626235291168250476992921 6
(%o3) sqrt(2)
(%o4) (3*x+2)*5^x/(2^x-1)

(%i5)  set_display(ascii)$
       1234^56;
       sqrt(2);
       (3*x+2)*5^x/(2^x-1);

(%o6) 12991190255487145194103208439623513775465782010127392384379\
01270462425943305509464892567848536247290201061395156473849109449\
21186523865849056275359066262352911682504769929216
(%o7)                                  sqrt(2)
(%o8)                                            x
                                      (3 x + 2) 5
                                     ------------
                                         x
                                        2  - 1

(%i9)  set_display(xml)$
       1234^56;
       sqrt(2);
       (3*x+2)*5^x/(2^x-1);

(%o10) 12991190255487145194103[128 digits]6235291168250476992921 6
(%o11) √2

(%o12) (3 x + 2) 5^x / (2^x - 1)
```

Каждый из форматов обладает своими достоинствами. Например, принятый по умолчанию `xml` удобен для вывода не слишком громоздких формул и не слишком больших чисел, в нем формулы выглядят красиво. Но если формула громоздкая, то лучше воспользоваться форматом `none`,

а если мы хотим видеть все цифры числа 1234^{56}, то форматом `ascii`. При выводе большого числа в формате `none` придется воспользоваться горизонтальной прокруткой, чтобы увидеть все его цифры. Громоздкая формула, выведенная в формате `ascii`, выглядит отвратительно.

2.3. Получение справочной информации

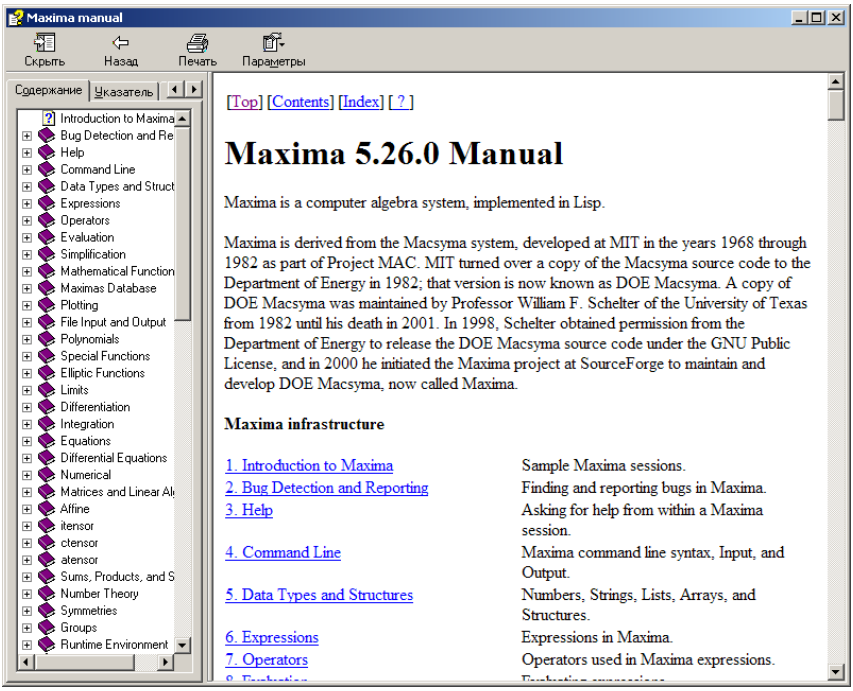
Справочную информацию по языку Maxima можно получить из следующих источников:

- файл справки;
- специальные команды языка Maxima;
- техническая поддержка
  http://blog.gmane.org/gmane.comp.mathematics.maxima.general.

Рассмотрим первые две возможности.

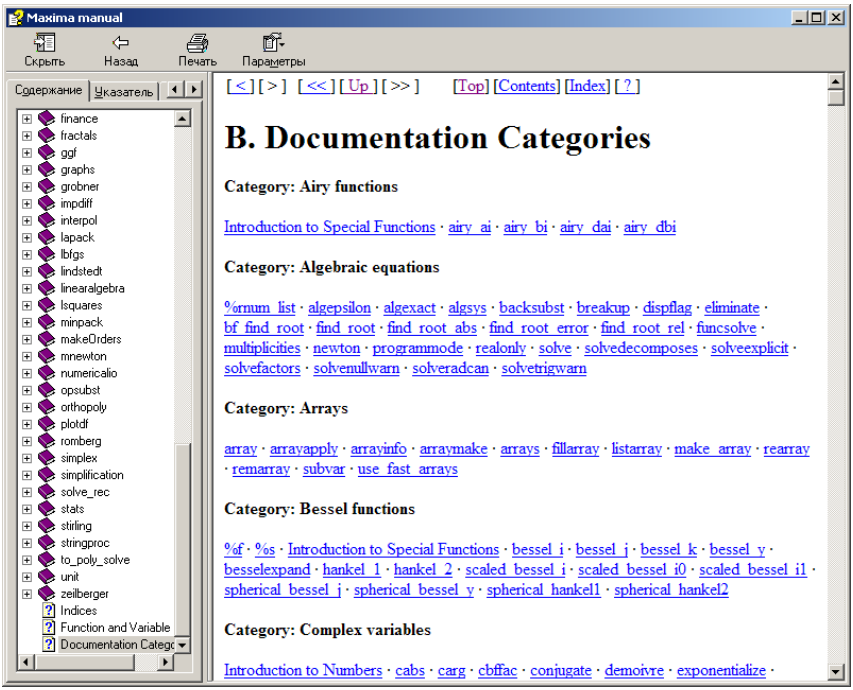
Файл справки можно вызвать из меню графической оболочки: `Помощь` → `Maxima Help`. Он содержит описание всех доступных функций и конструкций языка Maxima. Там также имеется некоторое количество примеров.

В левой панели — навигация: Содержание, Указатель и Поиск. В правой панели — просмотр текущей страницы справки. На главную страницу справки, озаглавленную Maxima 5.26.0 Manual, всегда можно попасть, щелкнув гиперссылку Top. Гиперссылки на главной странице дублируют пункты Содержания.



Справочный материал классифицирован в двух направлениях: по темам и по пакетам. Темы имеют понятные смысловые названия, чего, к сожалению, не всегда скажешь про пакеты. Например: Differential Equations, Numerical, Matrices and Linear Algebra — темы, далее: Affine, itensor, ctensor, atensor — пакеты. Затем: Sums Products and Series, Number Theory — опять темы и так далее. Впрочем, описание дополнительной функции, относящейся к определенному пакету, продублировано в описании пакета и в теме, к которой эта функция относится.

Для быстрого поиска нужной функции удобно обратиться к странице Documentation Categories, где функции упорядочены в соответствии с пунктами Содержания, либо к вкладке Указатель на левой панели (содержимое которой продублировано на странице Function and Variable Index) — там перечислены все операторы и функции в алфавитном порядке, либо собственно ко вкладке Поиск.



В Maxima имеется несколько вспомогательных команд, призванных облегчить поиск необходимой функции и показать ее возможности.

♦ Команда `? word` вызывает поиск в справке по точному ключевому слову *word*.

Листинг 5

```
(%i1)  ? sin$
```

– Function: sin (<x>)
– Sine.
There are also some inexact matches for ‘sin’.
Try ‘?? sin’ to see them.

```
(%o1)  true
```

♦ Команда `??` *word* вызывает поиск в справке по ключевым словам, содержащим *word*. В ответ предлагается список ключевых слов, из которого можно выбрать интересующее, введя его порядковый номер.

Листинг 6

```
(%i1)  ?? cos$

0: acos (Functions and Variables for Trigonometric)
1: acosh (Functions and Variables for Trigonometric)
2: benefit_cost (Functions and Variables for finance)
3: cos (Functions and Variables for Trigonometric)
4: cosh (Functions and Variables for Trigonometric)
5: cosnpiflag (Functions and Variables for Fourier series)
6: fourcos (Functions and Variables for Fourier series)
7: fourintcos (Functions and Variables for Fourier series)
8: great_rhombicosidodecahedron_graph
     (Functions and Variables for graphs)
9: icosahedron_graph (Functions and Variables for graphs)
10: icosidodecahedron_graph (Functions and Variables for graphs)
11: small_rhombicosidodecahedron_graph
     (Functions and Variables for graphs)
12: truncated_icosahedron_graph (Functions and Variables for graphs)
Enter space-separated numbers, 'all' or 'none':
```

♦ Функция

`example`("*func*")

выводит примеры использования функции *func*. Возвращает строку done. Чтобы просмотреть список функций, для которых есть примеры, надо вызвать `example`() с пустым аргументом. В версии Maxima 5.26.0 таких функций 147.

Листинг 7

```
(%i1)  example("is")$
(%i2)  is(x^2 >= 2*x-1)
(%o2)  true
(%i3)  assume(a > 1)
(%o3)  [a>1]
(%i4)  is(log(1+log(1+a)) > 0 and 1+a^2 > 2*a)
(%o4)  true
```

2.4. Один пример

Продемонстрируем возможности СКМ Maxima на примере исследования иррационального уравнения

$$x^2 - x - 1 = \sqrt[3]{-x^2 + x + 3}.$$

К сожалению, СКМ без подсказок плохо решают иррациональные уравнения. Вот и в этом примере применение функции `solve` «в лоб» ничего не даёт.

Листинг 8

```
(%i1)  /*  Сохраняем уравнение в переменной eq  */
       eq:x^2-x-1=(-x^2+x+3)^(1/3);
       /*  Пробуем сразу решить eq с помощью функции solve  */
       solve(eq);
```

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

(%o2) $[x = -\frac{\sqrt{4\left(-x^2 + x + 3\right)^{\frac{1}{3}} + 5} - 1}{2}, x = \frac{\sqrt{4\left(-x^2 + x + 3\right)^{\frac{1}{3}} + 5} + 1}{2}]$

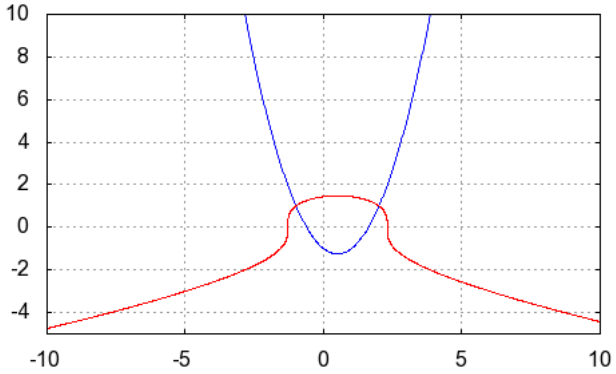
Результат работы (%o2) функции `solve` не удовлетворителен.

Попробуем построить графики левой и правой частей уравнения и визуально найти точки пересечения.

Листинг 9

```
(%i3)  wxdraw2d(
           nticks=200,
           grid=true,
           yrange=[-5,10],
           color=blue,
           /*  Левая часть  */
           explicit(lhs(eq), x,-10,10),
           color=red,
           /*  Правая часть  */
           explicit(rhs(eq), x,-10,10)
       )$
```

(%t3)



На рисунке видны две точки пересечения и похоже, что других нет. Чтобы проверить это изучим производные правой и левой частей уравнения.

Листинг 10

```
(%i4)  /*  Производная левой части уравнения  */
       diff(lhs(eq),x);
       solve(%=0);
       /*  Производная правой части уравнения  */
       diff(rhs(eq),x);
       solve(%=0);
```

(%o4) $2\,x-1$

(%o5) $[x=\frac{1}{2}]$

(%o6) $$\frac{1-2\,x}{3\,(-x^2+x+3)^{\frac{2}{3}}}$$

(%o7) $[x=\frac{1}{2}]$

На промежутке $(-\infty,\frac{1}{2})$ левая часть убывает, а правая — возрастает, значит на этом промежутке будет единственный корень уравнения. Далее, на $(\frac{1}{2},+\infty)$ — наоборот, поэтому здесь также получится единственный корень уравнения. Сами корни несложно угадать: $x=-1$, $x=2$.

Другой подход состоит в том, чтобы заменить переменную. Поскольку x присутствует только в комбинации x^2-x, то введем новую переменную

$t^3 = -x^2 + x + 3$ — чтобы извлекся кубический корень. Относительно t получаем кубическое уравнение, оно легко решается.

Листинг 11

```
(%i8)  /*  Сохраняем замену в переменной cg  */
       cg:t^3=-x^2+x+3;
       /*  Подставляем замену в уравнение eq  */
       ratsubst(lhs(cg),rhs(cg),eq);
       /*  Решаем полученное кубическое уравнение  */
       solve(%);
```

(%o8) $t^3 = -x^2 + x + 3$

(%o9) $2 - t^3 = t$

(%o10) $[t = -\frac{\sqrt{7}\,\%i + 1}{2}, t = \frac{\sqrt{7}\,\%i - 1}{2}, t = 1]$

Из 3-х полученных корней выберем действительный корень и найдем x, подставив $t = 1$ в нашу замену.

Листинг 12

```
(%i11) /*  Подставляем t=1 в замену  */
       subst([t=1],cg);
       /*  Решаем уравнение cg  */
       solve(%);
```

(%o11) $1 = -x^2 + x + 3$

(%o12) $[x = 2, x = -1]$

Еще один способ решения нашего иррационального уравнения состоит в том, чтобы сразу избавится от иррациональности, возведя в куб левую и правую части. Получится уравнение 6-й степени, но СКМ это нисколько не пугает.

Листинг 13

```
(%i13) /*  Возводим в куб обе части eq  */
       expand(lhs(eq)^3)=rhs(eq)^3;
       /*  Решаем  */
       solve(%);
```

(%o13) $x^6 - 3\,x^5 + 5\,x^3 - 3\,x - 1 = -x^2 + x + 3$

(%o14) $[x = 2,\ x = -1,\ x = \frac{1}{2} - \frac{\sqrt{3 - 2\sqrt{7}\,\%i}}{2},\ x = \frac{\sqrt{3 - 2\sqrt{7}\,\%i}}{2} + \frac{1}{2},\ x = \frac{1}{2} - \frac{\sqrt{2\sqrt{7}\,\%i + 3}}{2},\ x = \frac{\sqrt{2\sqrt{7}\,\%i + 3}}{2} + \frac{1}{2}]$

2.5. *Задания для самостоятельной работы*

2.1 Найдите файл с кодом функции `solve`, решающей уравнения и системы уравнений. Откройте файл в блокноте. Так выглядит типичный программный код на языке Lisp.

2.2 Названия многих функций образованы из соответствующих английских слов. Найдите в Справке описание функции, вычисляющей:

- ранг матрицы;
- определитель матрицы;
- предел функции;
- производную функции;
- первообразную функции.

2.3 Выясните общее предназначение пакета `lsquares`. Какие функции входят в этот пакет и каково предназначение каждой из них?

3. Типы данных

3.1. Идентификаторы

Идентификатор — основная синтаксическая единица языка программирования. Идентификаторы — это всевозможные имена: констант, переменных, функций и операторов. За некоторыми идентификаторами в Maxima закреплен определенный смысл, например:

- идентификатор `%e` представляет константу e — эйлерово число;
- идентификатор `+` — оператор сложения;
- идентификатор `sin` — функция sin;
- идентификатор `limit` — функция, вычисляющая предел.

Такие идентификаторы называются *служебными*. При этом для одних служебных идентификаторов смысл фиксирован (`%e`, `+`, `sin`), а другие можно переопределять[7] (`limit`).

Идентификатором в Maxima является любая последовательность букв латинского алфавита (большие и малые буквы различаются), цифр, символов `%`, `_`. Если цифра стоит на первом месте, то перед ней должен быть обратный слэш \.

Идентификаторы `%alpha`, `%beta`, `%gamma`, `%delta`, . . . , `%omega` и `%Alpha`, `%Beta`, `%Gamma`, `%Delta`, . . . , `%Omega` изображаются в wxMaxima соответсвующими буквами греческого алфавита.

Листинг 14

```
(%i1)  /*  Греческие буквы  */
       (%zeta+%Zeta)/(%xi+%Xi);
(%o1)  (Z + ζ)/(Ξ + ξ)
```

3.1.1. Константы

К служебным идентификаторам относятся *константы*.

♦ Константа `%e` — *эйлерово число* $e = 2.718281828459045\ldots$, основание натурального логарифма.

[7] Впрочем, этого лучше не делать — во избежание путаницы.

♦ Константа `%i` представляет *мнимую единицу*.

♦ Константы `true`, `false` представляют одноименные *булевы константы*.

♦ Константа `%pi` — отношение длины окружности к ее диаметру $\pi = 3.141592653589793\ldots$.

Имеются также некоторые специфические константы, относящиеся к математическому анализу. Они будут рассмотрены в соответствующем разделе пособия.

3.1.2. Переменные

♦ Идентификаторы `%i1`, `%i2`, ... и `%o1`, `%o2`, ... ссылаются на соответствующие блоки ввода и результаты.

♦ Идентификатор `%` ссылается на предыдущий результат.

Листинг 15

```
(%i1)  2/3+1/2;
(%o1)  5/6
(%i2)  (3/2)^3;
(%o2)  27/8
(%i3)  /*  Объясните ответ  */
       %i2+%o2*%;
(%o3)  945/64
```

Значение каждого идентификатора по умолчанию равно самому идентификатору. Такие идентификаторы, которым не присвоено значение, будем называть *свободными*. С помощью *оператора присвоения* идентификатору можно присвоить значение некоторого выражения.

♦ В результате выполнения команды *var*`:`*expr* переменной *var* присвоится значение выражения *expr*. Именно значение выражения, а не само выражение. Оператор `:` возвращает присвоенное значение.

♦ Функция

`print`(*expr1, expr2, ...*)

печатает значения выражений *expr1, expr2, ...* и возвращает напечатанную строку. Эту функцию удобно использовать для форматированного

вывода результатов вычислений.

Листинг 16

```
(%i1) a:(b:5);
      print("Выводим значения: a=", a, ", b=", b)$
(%o1) 5
Выводим значения: a = 5, b = 5
(%i3) %alpha:%beta;
      %alpha:3;
      print("Выводим значения: ", '%alpha, "=", %alpha,
         ", ", '%beta, "=", %beta)$
(%o3) β
(%o4) 3
Выводим значения: α = 3, β = β
```

Обратите внимание на вывод греческих букв с помощью функции `print`: если набрать `%alpha` внутри кавычек, то греческая буква выведена не будет, а если набрать `%alpha` вне кавычек, но без штриха, то вместо греческой буквы получим ее значение. Забегая вперед заметим, что оператор `'` предотвращает вычисление (т.е. подстановку значения в нашем случае).

Для некоторых задач необходимо, чтобы используемый идентификатор был *свободен*. Освободить идентификатор от присвоенного ему значения позволяет следующая функция.

- Функция

  `kill`(*var1*, . . . , *varN*)
  `kill`(`all`)

удаляет значения пользовательских переменных *var1*, . . . , *varN* либо значения всех пользовательских переменных. Последнюю процедуру можно также вызвать из меню: `Maxima` → `Clear Memory`.

Листинг 17

```
(%i1) x:sin(3)$ /*  Присвоили идентификатору x значение  */
      x;
      kill(x)$ /*  Сбросили значение x  */
      x;
```

```
(%o2)  sin(3)
(%o4)  x
```

В начале сеанса все идентификаторы свободны и нет необходимости применять функцию `kill`. Тем не менее, мы будем это делать всегда в том случае, если идентификатор должен быть свободен для корректной работы скрипта.

Иногда требуется задать *область изменения переменной*, не придавая переменной конкретного значения.

- Функция

  `assume`(*pred1, pred2, ...*)

добавляет сведения об используемых переменных в текущий *контекст выполнения*. Аргументы (высказывания) *pred1, pred2, ...* должны быть *отношениями сравнения* с операторами:

```
< <= > >= equal notequal
```

Листинг 18

```
(%i1)  kill(n,x)$
       /*  Сообщаем, что n не равно -1  */
       assume(notequal(n,-1))$
       integrate(x^n,x);
(%o3)  x^(n+1)/(n+1)
(%i4)  /*  Теперь пусть n=-1  */
       n:-1$
       integrate(x^n,x);
(%o5)  log(x)
(%i6)  kill(a,b)$
       assume(a>=0)$
       sqrt(a^2)+sqrt(b^2);
(%o8)  |b|+a
```

Обратим внимание, что в блоке ввода (%i1) нет никакой необходимости применять функцию `kill` — все идентификаторы и так ещё свободны.

Однако, если этот программный код окажется продолжением сеанса, использующим идентификаторы `x` или `n`, то предварительная очистка этих переменных будет необходимой для корректной работы скрипта. С целью напомнить читателю об этом обстоятельстве, мы и начали последний листинг с команды `kill`. Аналогично будем поступать и далее.

3.1.3. Функции и операторы

Функция задается следующим шаблоном: *fun*(*x1, . . . , xN*), где *fun* — идентификатор (имя) функции, *x1,. . . ,xN* — аргументы функции.

Оператор отличается от функции лишь оформлением. По существу это тоже самое и во внутреннем представлении Maxima не отличает операторы от функций. Использование операторов удобно и привычно: сравните $a+b$ и $\mathrm{add}(a,b)$ — здесь в первом случае применен оператор +, а во втором — некая гипотетическая функция add. Итак, отдавая дань традиции, Maxima использует следующие типы операторов:

- *префиксный* оператор: *op x1*;
- *постфиксный* оператор: *x1 op*;
- *бинарный инфиксный* оператор: *x1 op x2*;
- *N-арный инфиксный* оператор: *x1 op x2 op . . . op xN*;
- *скобочный* оператор: *lop x1, . . . , xN rop*.

Во всех шаблонах выше: *op* — идентификатор (имя) оператора, *x1, . . . , xN* — аргументы (операнды) оператора. Скобочный оператор имеет два идентификатора: левый *lop* и правый *rop*.

Идентификатор может быть одновременно именем функции и именем переменной. Но не может быть одновременно именем оператора и именем функции (или переменной). Функция `kill`(*f*) уничтожает все объекты, связанные с идентификатором *f*.

Для каждого типа данных в системе Maxima предусмотрены свои функции и операторы. Мы расскажем о них на страницах данного пособия в соответствующих темах. Отметим, что пользователь сам может определять удобные ему (пользовательские) функции и операторы. Более подробно об этом будет рассказано в следующей главе.

♦ *Пользовательская функция* задается так:

fun(*x1, . . . , xN*) `:=` *expr*;

Здесь *expr* может быть любым Maxima-выражением, необязательно математическим.

Листинг 19

```
(%i1)  /*  Аргументы должны быть свободными  */
       kill(x,y)$

(%i2)  /*  Задаем математическую функцию  */
       f(x,y):=x*y/(x^2+y^2);
```

(%o2) $\mathrm{f}(x,y) := \dfrac{x\,y}{x^2+y^2}$

```
(%i3)  f(1,-5);
       f(sin(x),cos(x));
```

(%o3) $-\dfrac{5}{26}$

(%o4) $\dfrac{\cos(x)\,\sin(x)}{\sin(x)^2+\cos(x)^2}$

```
(%i5)  kill(f)$
       f(1,-5);
```

(%o6) $\mathrm{f}(1,-5)$

```
(%i7)  /*  Задаем Maxima-функцию  */
       f(x):=print("x=", x);
```

(%o7) $\mathrm{f}(x) := \mathrm{print}(x =, x)$

```
(%i8)  f(sqrt(y))$
```

$x = \sqrt{y}$

♦ Оператор должен быть сначала зарегистрирован в системе, а затем определен командой, аналогичной определению функции. Например, для задания постфиксного оператора *op* его надо сначала зарегистрировать:

`postfix`(*op*),

а затем определить:

op(*x*) `:=` *expr*.

Листинг 20

```
(%i1) /*  Аргумент должен быть свободным  */
      kill(x)$
      /*  Регистрируем постфиксный оператор  */
      postfix("?");
(%o2) ?
(%i3) /*  Задаем оператор  */
      "?"(x):=x/(x^2+1);
(%o3) ?(x) := x/(x^2+1)
(%i4) /*  Применяем оператор к числу 3!=6  */
      3!?;
(%o4) 6/37
(%i5) sin(x)?;
(%o5) sin(x)/(sin(x)^2+1)
```

3.2. Выражения и блоки

Maxima-выражение (обычно будем говорить просто *выражение*) — последовательность идентификаторов и данных, которая может быть выполнена системой Maxima. Выражение должно оканчиваться терминальным символом ; или $. В первом случае значение выражения будет напечатано, во втором — нет.

Листинг 21

```
(%i1) /*  Значением этого выражения будет строка "текст"  */
      print("текст");
текст
(%o1) текст
(%i2) kill(b,c)$
      /*  Присвоим переменной a значение выражения b+c  */
      a:b+c;
```

```
(%o3)  b + c

(%i4)  kill(x,y)$
       /*  В переменной eq сохраним равенство 2*x+3*y=4  */
       eq:2*x+3*y=4;

(%o5)  2x + 3y = 4

(%i6)  /*  Список и множество  */
       [2,3,5,7,11,13,17];
       {2,3,5,7,11,13,17};

(%o6)  [2, 3, 5, 7, 11, 13, 17]
(%o7)  {2, 3, 5, 7, 11, 13, 17}

(%i8)  /*  Пользовательская функция  */
       f(x,y):=print(x,y);
       f("Значение x=",x:5);

(%o8)  f(x, y) := print(x, y)
Значение x=5
(%o9)  5
```

3.2.1. Основные операции с выражениями

Maxima различает *действующую форму* и *неопределенную форму* выражений. Выражение в действующей форме вычисляется, выражение в неопределенной форме — не вычисляется. По умолчанию все функции имеют действующую форму. Чтобы представить выражение в неопределенной форме надо поставить перед ним штрих ' — оператор неопределенной формы. Наоборот, чтобы привести выражение к действующей форме (вычислить насильно) применяется оператор '' — оператор действующей формы.

Сначала рассмотрим несколько простых примеров, поясняющих суть дела.

Листинг 22

```
(%i1)  a:3;
       b:'a;
```

```
(%o1) 3
(%o2) a

(%i3) b+a;
      /*  Довычисляем полученный результат  */
      ''%;

(%o3) a + 3
(%o4) 6

(%i5) kill(x,y)$
      a:x+y;

(%o6) y + x

(%i7) b+a;
      /*  Довычисляем полученный результат  */
      ''%;

(%o7) y + x + a
(%o8) 2 y + 2 x
```

Неопределенную форму выражения можно использовать в следующей ситуации.

Листинг 23

```
(%i1) kill(x)$
      'diff(x^2*%e^(3*x),x)=diff(x^2*%e^(3*x),x);
      'integrate(1/x, x, 1, 42)=integrate(1/x, x, 1, 42);
```

(%o2) $$\frac{d}{d\,x}\left(x^2\,\%e^{3\,x}\right)=3\,x^2\,\%e^{3\,x}+2\,x\,\%e^{3\,x}$$

(%o3) $$\int_1^{42}\frac{1}{x}\,dx=\log(42)$$

Применим операторы неопределенной и действующей формы для задания и вычисления *функции Дирихле*. Эта функция равна 1 если x — рациональное число и 0 — в противном случае.

Листинг 24

```
(%i1) kill(x)$
      /*  Выражение f не будет вычислено сразу  */
```

```
       f:'(if ratnump(x) then 1 else 0)$

(%i3)  x:sqrt(4)$
       /*  Подставляем x и вычисляем f  */
       print("f(", x, ")=", ''f)$
```

$f(2) = 1$

```
(%i5)  x:sqrt(2)$
       /*  Подставляем x и вычисляем f  */
       print("f(", x, ")=", ''f)$
```

$f(\sqrt{2}) = 0$

Рассмотрим используемый в Maxima механизм *сравнения выражений*. В Maxima имеются два различных типа равенств для выражений общего вида:

- *синтаксическое равенство* выражений: a = b — если совпадают представления этих выражений во внутреннем формате системы;
- *тождественное равенство* выражений: `equal`(a, b) — если выражения могут быть приведены к одинаковому виду с помощью тождественных преобразований.

Отрицанием оператора `=` является оператор `#`, а отрицанием `equal` — функция `notequal`. Числовые выражения могут быть также сравнены с помощью операторов `<`, `<=`, `>`, `>=`.

Все равенства и неравенства представляют собой объекты (называемые *высказываниями*), а не логические значения.

♦ Функция

`equal`(*expr1*, *expr2*)

возвращает высказывание «выражения *expr1*, *expr2* равны тождественно». Для проверки тождественного равенства используется функция `is`. В отличие от `equal`, оператор `=` возвращает высказывание «выражения равны синтаксически».

♦ Функция

`notequal`(*expr1*, *expr2*)

возвращает отрицание высказывания `equal`(*expr1*, *expr2*).

Чтобы получить логическое значение высказывания (`true` или `false`) используется

- функция

  `is`(*expr*)

— вычисляет логическое значение высказывания *expr* в контексте предположений, созданных функцией `assume`. Если высказывание не имеет определённого логического значения, то функция `is` возвратит `unknown`.

Листинг 25

```
(%i1)  1=2;
       is(1=2);
       1<2;
       is(1<2);

(%o1)  1 = 2
(%o2)  false
(%o3)  1 < 2
(%o4)  true

(%i5)  kill(x,y)$
       is(x>0);
       is( (x+y=y+x) and (x^2=x*x) );
       is(x^2+y^2>=2*x*y);

(%o6)  unknown
(%o7)  true
(%o8)  true

(%i9)  is(sqrt(x*y)=sqrt(x)*sqrt(y));
       assume(x>=0,y>=0)$
       is(sqrt(x*y)=sqrt(x)*sqrt(y));

(%o9)  false
(%o11) true

(%i12) is(x^2-1=(x-1)*(x+1));
       is(equal(x^2-1,(x-1)*(x+1)));

(%o12) false
(%o13) true
```

```
(%i14) is(equal(x,sqrt(x^2)));
       is(equal(abs(x),sqrt(x^2)));
       is(notequal(x,sqrt(x^2)));
       is(notequal(abs(x),sqrt(x^2)));

(%o14) unknown
(%o15) true
(%o16) unknown
(%o17) false
```

3.2.2. Объединение выражений в блок

Иногда возникает необходимость объединить несколько выражений в одно (например: при задании функции, при использовании условных операторов или операторов цикла). Это делается с помощью *блока*[8].

- Выполнять по порядку *expr1, expr2, ...* и возвратить результат последнего выражения:

  (*expr1, expr2, ...*);

- То же самое, но со списком *vars локальных переменных*:

  `block`([*vars*], *expr1, expr2, ...*);

Область видимости локальной переменной ограничена блоком, в котором она объявлена. В блоках также можно использовать *внешние переменные*. Имя локальной переменной может совпадать с именем внешней переменной.

Листинг 26

```
(%i1)  x:0$
       y:5$
       (x:1, z:2, x+y+z);
       x;

(%o3)  8
(%o4)  1

(%i5)  a:0$
       b:5$
```

[8] Не следует смешивать описываемую здесь синтаксическую конструкцию *блок* с блоками ввода и вывода оболочки wxMaxima, о которых шла речь на стр. 29.

```
       block([a,c], a:1, c:2, a+b+c);
       a;

(%o7)  8
(%o8)  0
```

♦ Функция

`return`(*expr*)

вызванная в теле `block`, прерывает его выполнение и возвращает в качестве результата блока значение выражения *expr*.

Листинг 27

```
(%i1)  (x:1, y:2*x, return(x+y), x*y);

return: not within 'block'
– an error. To debug this try: debugmode(true);

(%i2)  block(x:1, y:2*x, return(x+y), x*y);

(%o2)  3
```

♦ Идентификатор `%%` ссылается на последний результат в блоке — аналогично тому, как работает `%` вне блока.

Листинг 28

```
(%i1)  block([x:1,y:2], y:x+y, x:%%,
               block([z:10], y:x+z, z:%%, x+z)
       );

(%o1)  16
```

3.3. Булев тип

Рассмотрим операторы и функции, определенные над выражениями следующих двух типов.

- *предикаты* — выражения, принимающие логические значения `true`, `false`;
- *высказывания* — выражения, представляющие собой объекты, сводящиеся к значениям `true`, `false` с помощью функции `is`.

♦ Логические операторы `and`, `or`, `not` имеют обычный смысл:

`and` — *логическое умножение* или конъюнкция $\wedge$,

`or` — *логическое сложение* или дизъюнкция $\vee$,

`not` — *логическое отрицание* $\neg$.

Среди них наивысшим приоритетом обладает унарный оператор `not`, далее `and` и на последнем месте `or`.

Листинг 29

```
(%i1)  true and true and false;
       true and false or true;
       not true or not false;
       not (true or not false);
(%o1)  false
(%o2)  true
(%o3)  true
(%o4)  false
(%i5)  kill(x,y)$
       x and false;
       x or y and false;
(%o6)  false
(%o7)  x
```

В следующем примере задается *булева функция* [9] и вычисляются ее значения.

Листинг 30

```
(%i1)  kill(x1,x2,x3,x4)$

       f(x1,x2,x3,x4):=
       x1 and x2 and not x3 and not x4 or
       x1 and not x2 and x3 and not x4 or
       not x1 and x2 and x3 and not x4 or
       x1 and x2 and x3 and not x4 or
       x1 and not x2 and not x3 and x4 or
```

[9] Функция от логических переменных, принимающая логическое значение.

```
       not x1 and x2 and not x3 and x4 or
       x1 and x2 and not x3 and x4 or
       not x1 and x2 and x3 and x4$

       f(true,true,false,false);
       f(false,false,false,true);
       f(x1,x2,false,false);
(%o3)  true
(%o4)  false
(%o5)  x1 ∧ x2
```

♦ Функция

$$\texttt{charfun}(b)$$

возвращает 1 если b=`true` и 0 если b=`false`

В следующем примере запрограммирована система перевода оценки из 100-балльной шкалы в 5-балльную, принятая в Финансовом университете.

Листинг 31

```
(%i1)  kill(n)$
       f(n):=2+charfun(n>=51)+charfun(n>=70)+charfun(n>=86)$
       f(37);
       f(62);
       f(80);
       f(86);
(%o3)  2
(%o4)  3
(%o5)  4
(%o6)  5
```

3.4. Числа

В Maxima имеются числа трех различных типов:

Листинг 32

```
(%i1)  /*  Целое  */
       123;
       123.;
```

```
/*  Стандартное число с плавающей точкой - float  */
123.0;
123e0;
123.0e0;

/*  Длинное число с плавающей точкой - bigfloat  */
123b0;
123.0b0;
```

Простая дробь никогда не будет «по умолчанию» преобразована в число с плавающей точкой, если только пользователь не сделает этого явно. Обратное преобразование иногда выполняется системой внутри других функций — для уменьшения погрешности вычислений.

Система способна обрабатывать довольно большие *целые числа*, например вычисление и вывод на экран числа 1234^{123456}, содержащего 381642 цифры,[10] занимает на компьютере одного из авторов около 0.07 секунды. Вычисление без вывода на экран числа $1234^{12345678}$ заняло 13.92 секунд. Вычислять $1234^{123456789}$ Maxima отказалась, причем никак это не комментируя.

Рациональное число представляется в системе как упорядоченная пара целых:

[числитель, знаменатель].

Целое число считается частным случаем рационального — со знаменателем, равным 1. В общем случае рациональное число не является числом с точки зрения внутреннего представления СКМ Maxima.

Число с плавающей точкой представляется в виде: $x \cdot 10^n$. Имеется два типа таких чисел: стандартное и длинное.

Стандартное число с плавающей точкой (float) вводится в формате x`e`n, причем мантисса x содержит суммарно в целой и дробной частях не более 16 цифр, а показатель степени n не слишком велик. Например, число $1.0 \cdot 10^n$ при $n \geqslant 309$ вызывает переполнение, а при $n \leqslant -324$ — тождественно равно 0.0.

Длинное число с плавающей точкой (bigfloat) вводится в формате x`b`n. Мантисса в bigfloat сохраняется почти с произвольной точностью (ограниченной лишь возможностями компьютера), которая определяется

[10] При стандартном форматировании это соответствует примерно 80 страницам.

параметром `fpprec` и по умолчанию равна 16. Показатель степени произволен. Конечно, все неприятности, имеющие место для стандартных чисел с плавающей точкой, возникают и здесь, но в более далеких разрядах.

Из-за ошибок округления числа с плавающей точкой обладают некоторыми неожиданными с точки зрения математики свойствами.

Листинг 33

```
(%i1)  /*  Операция сложения не ассоциативна  */
       x:1.000000000000001;
       y:-x;
       z:0.0000000000000001;
       print("1e16*((x+y)+z)=",1e16*((x+y)+z),",   ",
          "1e16*(x+(y+z))=",1e16*(x+(y+z)))$
```

(%o1) 1.000000000000001
(%o2) -1.000000000000001
(%o3) $9.9999999999999998\,10^{-17}$
$1e16 * ((x + y) + z) = 1.0, \quad 1e16 * (x + (y + z)) = 0.0$

```
(%i5)  /*  Нарушается свойство экспоненты  */
       x:100.0;
       y:0.01;
       %e^(x+y)-%e^x*%e^y;
```

(%o5) 100.0
(%o6) 0.01
(%o7) $1.43601044557104\,11\,10^{29}$

```
(%i8)  /*  В цикле ошибка накапливается  */
       x:1e-5;
       y:0.0;
       for i:1 thru 1000000 do y:y+x$
       y-1000000*x;
```

(%o8) $1.0000000000000001\,10^{-5}$
(%o9) 0.0
(%o11) $-2.0940937872637733\,10^{-10}$

♦ Логическая функция

`numberp`(x)

проверяет является ли x (математическим) числом: целым, рациональным, с плавающей точкой (стандартным или длинным) — но не *символическим обозначением* числа.

- Логическая функция

  `ratnump`(x)

проверяет является ли x рациональной дробью (в частности — целым числом).

- Логическая функция

  `integerp`(x)

проверяет, является ли x целым числом.

- Логические функции

  `floatnump`(x),
  `bfloatp`(x)

проверяют, является ли x стандартным (соответственно, длинным) числом с плавающей точкой.

Листинг 34

```
(%i1) numberp(2.3456b78);
      numberp(234/5678);
      numberp(log(2));
      numberp(%e);

(%o1) true
(%o2) true
(%o3) false
(%o4) false

(%i5) ratnump(2.3);
      ratnump(23);
      ratnump(2/3);

(%o5) false
(%o6) true
(%o7) true

(%i8) integerp(1.);
      integerp(1.0);
      integerp(4/2);
```

```
(%o8)  true
(%o9)  false
(%o11) true
```

♦ Над числами определены операции `+ - * / ^` имеющие обычный смысл:

`+` — N-арный инфиксный *оператор сложения*;

`-` — префиксный *оператор арифметического отрицания*, выражение $x - y$ понимается как $x + (-y)$;

`*` — N-арный инфиксный *оператор умножения*;

`/` — бинарный инфиксный *оператор деления*;

`^` — бинарный инфиксный оператор *возведения в степень*.

Преобразование чисел с плавающей точкой в рациональные осуществляется функциями `rationalize`, `rat`. Такое преобразование также выполняется при вызове некоторых других функций (например, `solve`) для уменьшения погрешности вычислений.

♦ Функция

`rationalize`(x)

возвращает число x, представленное рациональной дробью.

♦ Функция

`rat`(x)

возвращает число x, представленное приближенно рациональной дробью. Результат при этом будет отмечен символом `/R/`. Погрешность, с которой работает функция, задается глобальной переменной `ratepsilon` (по умолчанию 2.0e-8).

Обратный перевод рациональных чисел в числа с плавающей точкой выполняется с помощью следующих функций.

♦ Функции

`float`(*expr*),
`bfloat`(*expr*)

переводят все числа в выражении *expr* в стандартные (соответственно, длинные) числа с плавающей точкой.

Обратите внимание, что Maxima по умолчанию не будет переводить рациональные числа в числа с плавающей точкой. Это позволяет проводить вычисления с рациональными числами без потери точности.

Листинг 35

```
(%i1) -3/7^5+2/15*(6/11-13/17)^3;
```

$$(\%o1)\quad -\frac{2610974629}{1648565772315}$$

```
(%i2) fpprec:30$
      bfloat(%);
```

$$(\%o3)\quad -1.58378553822183053031391178426b-3$$

```
(%i4) rationalize(%);
```

$$(\%o4)\quad -\frac{2055871168675525458266368821431}{1298074214633706907132624082305024}$$

```
(%i5) float(%);
```

$$(\%o5)\quad -0.0015837855382218$$

```
(%i6) rat(%);
```

rat: replaced -0.00158378553822
by $-543405/343105166 = -0.00158378553822$

$$(\%o6)/\mathrm{R}/\quad -\frac{543405}{343105166}$$

♦ Чтобы все числовые функции вычислялись в числах с плавающей точкой следует установить глобальную переменную

`numer: true`.

По умолчанию Maxima производит только символьные преобразования, т.е. переменная `numer` имеет значение `false`.

Листинг 36

```
(%i1) /*  Maxima удобно использовать как калькулятор  */
      numer:true$

(%i2) /*  Арифметические действия  */
      (12341-675/15+94*12.3)/(37.61+82*1.9);
```

```
(%o3)  69.55276355927823

(%i4)  /*  Тригонометрические вычисления  */
       sin(%pi/8)*cos(3*%pi/16);

(%o4)  0.31818964514321

(%i5)  /*  Вычисления степеней и логарифмов  */
       log(2^15+%e^25)/log(2);

(%o5)  36.0673766787658

(%i6)  kill(x)$
       /*  Решение уравнений  */
       solve(x^3-5*x+4=0,x);

rat: replaced 4.123105625617661 by 76996132/18674305 = 4.123105625617661
rat: replaced 4.123105625617661 by 76996132/18674305 = 4.123105625617661

(%o7)  [x = -2.56155281280883, x = 1.56155281280883, x = 1]
```

♦ Функция

`compare`(x, y)

сравнивает числа x,y и возвращает либо оператор $<$, $\leq$, $>$, $\geq$, $=$, $\#$ (оператор «не равно»), либо `notcomparable` — если сравниваются комплексные числа, либо `unknown` — если значения переменных невозможно сравнить.

Листинг 37

```
(%i1)  kill(x)$
       compare(%e^%pi, %pi^%e);
       compare(x^2+1,2*x);
       compare(1/x,0);
       compare(x^2,x*x);
       compare(x,1);

(%o2)  >
(%o3)  ≥
(%o4)  #
(%o5)  =
(%o6)  unknown
```

♦ Функции

`max`(*x1*, ..., *xN*),
`min`(*x1*, ..., *xN*)

возвращают, соответственно, максимальное и минимальное число.

Листинг 38

```
(%i1) max(%pi^%e,%e^%pi);
      min(%pi^%e,%e^%pi);
(%o1) %e^%pi
(%o2) %pi^%e
```

♦ Функция

`mod`(*x*, *y*)

возвращает число $z \in [0, y)$ такое, что $\frac{x-z}{y}$ — целое. В случае натуральных x, y число z есть остаток от деления x/y.

Листинг 39

```
(%i1) mod(-345/17,%pi);
      float(%);
(%o1) 7 %pi - 345/17
(%o2) 1.697030928069729
```

♦ Функции

`sum`(*Xi*, *i*, *i0*, *i1*),
`product`(*Xi*, *i*, *i0*, *i1*)

возвращают, соответственно, сумму $\sum_{i=i0}^{i1} Xi$ и произведение $\prod_{i=i0}^{i1} Xi$.

Листинг 40

```
(%i1) sum(i^2,i,1,1000);
      product(i/(i^2+1),i,1,20);
(%o1) 333833500
(%o2) 17200947456/146517391394041314511720578125
```

♦ Функция

`abs`(x)

возвращает абсолютную величину $|x|$ числа x. Функция также корректно работает при комплексных x.

♦ Функция

`signum`(x)

возвращает 0 при $x = 0$ и $\dfrac{x}{|x|}$ в остальных случаях. Функция работает и на комплексных числах.

♦ Функция

`sqrt`(x)

возвращает $\sqrt{x}$. Равносильно x`^(1/2)`. В случае комплексного x функция возвращает одно из значений комплексного корня.

♦ Функции

`sin`(x),
`cos`(x),
`tan`(x),
`cot`(x)

возвращают значения тригонометрических функций: $\sin x$, $\cos x$, $\operatorname{tg} x$, $\operatorname{ctg} x$. Корректно работают также и при комплексных x.

♦ Функции

`asin`(x),
`acos`(x),
`atan`(x),
`acot`(x)

– обратные тригонометрические: $\arcsin x$, $\arccos x$, $\operatorname{arctg} x$, $\operatorname{arcctg} x$. Корректно работают также и при комплексных x.

♦ Функции

`exp`(x),
`log`(x)

возвращают значения экспоненты e^x и натурального логарифма $\ln x$. Запись `exp`(x) равносильна записи `%e^`x. Корректно работают также и при комплексных x.

♦ Функции

`sinh`(x),
`cosh`(x),
`tanh`(x),
`coth`(x)

возвращают значения соответствующих гиперболических функций: $\operatorname{sh} x$, $\operatorname{ch} x$, $\operatorname{th} x$, $\operatorname{cth} x$. Корректно работают также и при комплексных x.

♦ Функции

`asinh`(x),
`acosh`(x),
`atanh`(x),
`acoth`(x)

– обратные гиперболические: $\operatorname{arsh} x$, $\operatorname{arch} x$, $\operatorname{arth} x$, $\operatorname{arcth} x$. Корректно работают также и при комплексных x.

Рассмотрим простой пример вычислений с тригонометрическими функциями.

Листинг 41

```
(%i1)  t:sin(2)*cos(2);
(%o1)  sin(2) cos(2)
(%i2)  float(t);
(%o2)  −0.37840124765396
(%i3)  asin(2*t);
(%o3)  asin(2 cos(2) sin(2))
(%i4)  trigreduce(%);
(%o4)  π − 4
```

♦ Операция $x!$ для натуральных x возвращает точное значение факториала $x!$. Для остальных x операция возвращает приближенное значение $\Gamma(x+1)$ так называемой *гамма-функции Эйлера* $\Gamma(x) = \int_0^{+\infty} t^{x-1} e^{-t}\, dt$, являющейся обобщением факториала.

Листинг 42

```
(%i1)  20!;
```

```
(%o1)  2432902008176640000

(%i2)  1.4!;
       gamma(2.4); /*  Гамма-функция Эйлера  */

(%o2)  1.242169344504305
(%o3)  1.242169344504305

(%i4)  floatnump(1.4!);
       bfloatp(1.4!);

(%o4)  true
(%o5)  false
```

♦ Функция

$$\texttt{random}(x)$$

имитирует *равномерно распределенную дискретную случайную величину*. Для натурального x функция возвращает случайное целое число из промежутка $[0, x-1]$. Для положительного x с плавающей точкой функция возвращает случайное число с плавающей точкой[11] из $(0, x)$.

Листинг 43

```
(%i1)  random(100);
       random(100);

(%o1)  15
(%o2)  91

(%i3)  random(float(%pi));
       random(float(%pi));

(%o3)  2.870817524923704
(%o4)  0.792431801944530
```

3.4.1. Целые числа

♦ Логические функции

[11] В этом случае можно считать, что функция имитирует непрерывную случайную величину. При этом надо понимать, что по сути `random`(x) является дискретной величиной, т.к. имеет конечное число знаков. Вообще, компьютер не может имитировать полноценную непрерывную случайную величину.

`evenp`(n),
`oddp`(n)

проверяют, является ли n четным (соответственно, нечетным) числом.

Листинг 44

```
(%i1) evenp(5);
      oddp(5);
(%o1) false
(%o2) true
```

Следующие функции связаны с разложением числа на простые множители. Время вычисления этих функций определяется величиной максимального простого множителя. Если даже само число n велико, но не содержит в разложении большого простого множителя, то функция будет работать довольно быстро.

♦ Функция

`ifactors`(n)

возвращает список простых делителей $p1, p2, \ldots$ числа n и их степеней $k1, k2, \ldots$ в следующем формате: `[[`$p1, k1$`], [`$p2, k2$`], ...]`.

♦ Функция

`factor`(n)

возвращает разложение числа n в произведение степеней простых чисел.

♦ Функция

`divisors`(n)

возвращает множество делителей числа n, включая 1 и само число.

Листинг 45

```
(%i1) n:319455212624291824716636O$
      factor(n);
      ifactors(n);
(%o2) 2^3 3^5 5 7 11^2 13 17^3 19 23^4 29 31^2 41
(%o3) [[2,3], [3,5], [5,1], [7,1], [11,2], [13,1], [17,3], [19,1], [23,4], [29,1], [31,2],
[41,1]]
```

```
(%i4)  divisors(34);

(%o4)  {1,2,17,34}
```

♦ Функция

`totient`(*n*)

для натурального *n* возвращает значение *функции Эйлера* $\varphi(n)$ — количество натуральных чисел, меньших *n* и взаимно простых с ним.

Листинг 46

```
(%i1)  n:2*3^4*7^2*13^2*23;
       totient(n);
       /*  Формула для функции Эйлера  */
       n*(1-1/2)*(1-1/3)*(1-1/7)*(1-1/13)*(1-1/23);
(%o1)  30855006
(%o2)  7783776
(%o3)  7783776
```

♦ Функция

`lcm`(*n1, n2, ...*)

возвращает *наименьшее общее кратное* (НОК) чисел *n1, n2, ...*.

♦ Функция

`gcd`(*n1, n2*)

возвращает *наибольший общий делитель* (НОД) чисел *n1, n2*.

♦ Функция

`ezgcd`(*n1, n2, ...*)

возвращает список, первый элемент которого — НОД чисел *n1, n2, ...*, а остальные элементы суть частные от деления аргументов *n1, n2, ...* на НОД.

Листинг 47

```
(%i1)  lcm(2,34,12,102);
(%o1)  204
(%i2)  gcd(20790,385);
       gcd(20790,84);
```

```
gcd(gcd(20790,385),84);
ezgcd(20790,385,84);
%[1];

(%o2) 385
(%o3) 42
(%o4) 7
(%o5) [7, 2970, 55, 12]
(%o6) 7
```

- Логическая функция

  `primep`(n)

проверяет, является ли n простым числом.

- Функции

  `next_prime`(n),
  `prev_prime`(n)

возвращают ближайшее простое число, большее (меньшее) n.

Листинг 48

```
(%i1) primep(285449538541191976211657193889899027276549329 3);
(%o1) true
(%i2) next_prime(2^100);
(%o2) 1267650600228229401496703205653
(%i3) 100!-prev_prime(100!);
(%o3) 271
```

- Функция

  `inv_mod`$(a,\ m)$

возвращает число, *обратное a по модулю m*, т.е. такое целое число x в промежутке $[1, m-1]$, что произведение ax дает при делении на m в остатке 1. Функция возвращает `false`, если такого x не существует.

- Функция

  `power_mod`$(a,\ n,\ m)$

возвращает $x = a^n \mod m$, т.е. остаток от деления a^n на m. При $n = -1$ функция идентична предыдущей. Функция возвращает `false`, если такого x не существует.

Листинг 49

```
(%i1) inv_mod(19, 17);
      /* Проверка: */
      integerp( (19*%-1)/17 );
(%o1) 9
(%o2) true

(%i3) power_mod(19, 2, 17);
      /* Проверка: */
      integerp( (19^2-%)/17 );

(%o3) 4
(%o4) true
```

В следующем примере продемонстрирован известный алгоритм шифрования RSA.

Листинг 50

```
(%i1) /* 1. Подготовительный этап */
      set_display(ascii)$
      /* Берем два простых числа */
      p:prev_prime(2^1024);
(%o2) 17976931348623159077293051907890247336179769789423065727343\
00811577326758055009631327084773224075360211201138798713933576587\
89768814416622492847430639474124377767893424865485276302219601246\
09411945308295208500576883815068234246288147391311054082723716335\
05106845862982399472459384797163048353563296242241371 11

(%i3) q:next_prime(2^1024);

(%o3) 17976931348623159077293051907890247336179769789423065727343\
00811577326758055009631327084773224075360211201138798713933576587\
89768814416622492847430639474124377767893424865485276302219601246\
09411945308295208500576883815068234246288147391311054082723716335\
05106845862982399472459384797163048353563296242241378 59
```

```
(%i4)  /*  n=totient(p*q)  */
       n:(p-1)*(q-1);
```

(%o4) 323170060713110073007148766886699519604441026697154840321307\
345427524655138867890893197201411522913463688717960921898019494117\
955915049092109508815238644828312063087736730099609175019775038967\
521067960576383840675682767922186426197561618380943384761704705817\
645852036305042887575891541065808607552399123930385618270685418287\
847507497573301285629521677974210663395435663632915987574042445697\
765532820024376746164240463246433295286970627096672368289555605697\
920435991417897710716560438720317823146807994837771496531004898217\
979003229476300391310310823295241627283645509947586237148546125707\
423016300800875853068462896437337103807

```
(%i5)  /*  Ключ шифрования и ключ расшифрования  */
       a:17$
       b:inv_mod(a,n);
```

(%o6) 285150053570391240888660676664734870239212670615136623812914\
812595805780637069625528210600689908059973723982008134394289653996\
110151390480250777815174543674593801859123244083162501744856379283\
295259332103388831484795225458611350789663277302986554445328661581\
047379162143125669631244698311243109404034679873102388400749604191\
838021173642849663677468312705593714911497021998903591980502734293\
664727391246615506291099794084076738788614710913196666710911062670\
512510862153381415635767510204236541130921513204685337337233441461\
424379446292086249114286025936792250877728197395459952268020308500\
706655164472173143915297444 53

```
(%i7)  /*  2. Шифрование  */
       /*  Исходный текст представлен в виде числа m < p*q  */
       m:12345678910111213141516171819202122232425262728 29$
       /*  Шифротекст:  */
       c:power_mod(m, a, p*q);
```

(%o8) 254444641731990269159186949693610795086168029618143622057993\
681875591782091249860482836700716153166400761680449666185634982774\
052750792743856099675109135437350943626504805270499884121017132664\
461906944332993835859760111263014189443992572725700834604277397716\
428598900801372339858161464237830852499774200665187173423141650803\

```
75616919418748772680518615583559596572855814592347803449970013558\
6
03806294536191181381445931196868033938331431040049266560354717834\
9
11406617139629997253134493832266284484140118653349112215563180051\
3
15708189178091467161222068463649801604527579044130612883084004932\
3
69905631610376348001434873676
```

```
(%i9)  /*  3. Расшифрование  */
       power_mod(c, b, p*q);
```

```
(%o9) 12345678910111213141516171819202122232425262728 29
```

3.4.2. Числа с плавающей точкой

♦ Глобальная переменная `fpprec` — число *значащих цифр* в арифметике длинных чисел с плавающей точкой (по умолчанию 16).

♦ Глобальная переменная `fpprintprec` — сколько цифр числа с плавающей точкой (обычного или длинного) выводить на экран. По умолчанию `fpprintprec:0` — неограничено.

Листинг 51

```
(%i1)  /*  Число e до 200-го знака  */
       set_display(ascii)$
       fpprec:200$
       bfloat(%e);
       fpprintprec:5$
       bfloat(%e);
```

```
(%o3)  2.7182818284590452353602874713526624977572470936999595749669
6762772407663035354759457138217852516642742746639193200305992181\
7413596629043572900334295260595630738132328627943490763233829880 7\
53195251019b0
(%o5)  2.7182b0
```

♦ Функции

`ceiling`(x),
`floor`(x)

округляют x в большую и меньшую сторону.

♦ Функция

`round`(x)

округляет x по правилам округления с одной оговоркой: полуцелые числа округляются до ближайшего четного числа.

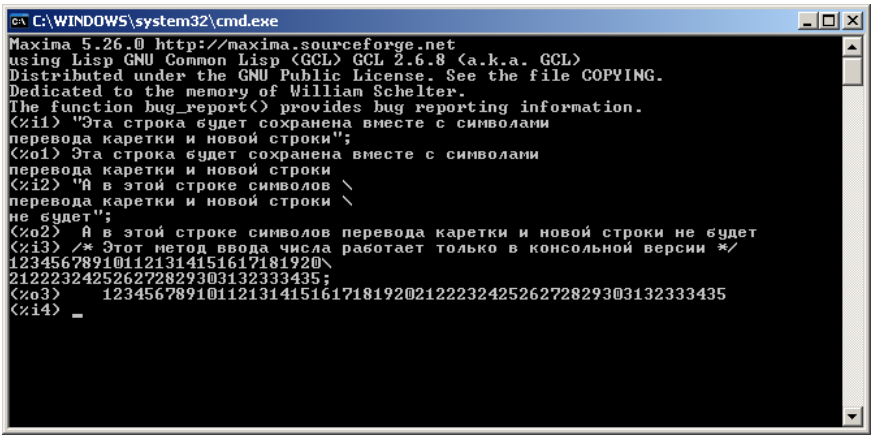
Листинг 52

```
(%i1)  x:10.5$
       ceiling(x);
       floor(x);
       round(x);
(%o2)  11
(%o3)  10
(%o4)  10
(%i5)  x:-12.5$
       ceiling(x);
       floor(x);
       round(x);
(%o6)  −12
(%o7)  −13
(%o8)  −12
```

3.5. Строки

Строка — последовательность символов, заключенная в двойные кавычки. Чтобы использовать символы " (двойная кавычка) и \ (обратный слэш) внутри строки, надо поставить перед ними обратный слэш. Если строка длинная и набрана в несколько экранных строк, то в ней сохранятся символы переноса строки. Чтобы этого избежать также используют обратный слэш. Впрочем, заметить разницу можно только работая в консольной версии Maxima. Оболочка wxMaxima почему-то игнорирует символы переноса строки в любом случае.

Большое число также может быть введено с помощью слэш. Но этот метод, к сожалению, тоже работает только в консольной версии Maxima.



Для использования описанных ниже функций необходимо загрузить библиотеку `stringproc` командой `load("stringproc")$`. В wxMaxima эта библиотека загружается по умолчанию.

♦ Глобальная переменная `stringdisp` — печатать строку в двойных кавычках[12], или без. По умолчанию `false`, т.е. печатает строку без кавычек. Но иногда это неудобно.

Листинг 53

```
(%i1)  s:"Какой-то текст";
       stringdisp:true$
       s;
(%o1)  Какой-то текст
(%o3)  "Какой-то текст"
```

♦ Логическая функция

`stringp`(*str*)

проверяет, является ли *str* строкой.

♦ Логические функции

`digitcharp`(*char*),
`alphacharp`(*char*),
`alphanumericp`(*char*)

проверяют, является ли *char* строкой, состоящей из одного символа:

[12] Обратите внимание, что кавычки только указывают границы строки, но частью строки не являются.

- цифры,
- буквы алфавита (латинского или кириллицы),
- цифры либо буквы алфавита.

Заметим, что:

- `"1"` — строка, состоящая из одной цифры, а `1` — число;
- `"x"` — строка, состоящая из одной буквы, а `x` — идентификатор.

Листинг 54

```
(%i1) stringp(1234);          /*  - это число  */
      stringp(\1234);          /*  - это идентификатор  */
      stringp("1234");
(%o1) false
(%o2) false
(%o3) true
(%i4) digitcharp(1);          /*  так неверно  */
      digitcharp("1");          /*  а так - верно  */
      digitcharp("x");
stringproc: 1 is no Maxima character.
— an error. To debug this try: debugmode(true);
(%o5) true
(%o6) false
```

♦ Функции

`ascii`(*int*),
`cint`(*char*)

возвращают символ *char* по его *ASCII-коду* $int = 0, 1, \ldots, 255$ и обратно. Обратите внимание, что оболочка wxMaxima не умеет выводить так называемые управляющие символы — соответствующие кодам: 1-31, 127, 152. Соответствующие символы можно увидеть, выполнив `ascii`(*int*) в консольной версии Maxima.

Листинг 55

```
(%i1) ascii(49);
      ascii(64);
```

```
       cint("$");
       cint("Б");
(%o1)  1
(%o2)  @
(%o3)  36
(%o4)  193
```

3.5.1. Строка как последовательность символов

- Функция

  `slength`(*str*)

возвращает длину строки *str*.

- Функция

  `charat`(*str*, *n*)

возвращает *n*-ый символ строки *str*.

- Функция

  `sposition`(*char*, *str*)

возвращает позицию первого вхождения символа *char* в строку *str*. Регистрозависимая функция.

Листинг 56

```
(%i1)  s:"Математика - царица наук"$
       slength(s);
       charat(s,14);
       sposition("ц",s);
(%o2)  24
(%o3)  ц
(%o4)  14
```

- Функция

  `charlist`(*str*)

возвращают список символов строки *str*.

- Функция

  `simplode`(*list*, *del*),
  `simplode`(*list*)

преобразует список *list* в строку, используя строку *del* в качестве разделителя. По умолчанию *del*="". Обратное преобразование выполняет функция `split`.

- Функция

  `split`(*str, del, mdel*),
  `split`(*str, del*),
  `split`(*str*)

возвращает список подстрок. Разбиение на подстроки проводится по разделителю *del* (по умолчанию — символ пробела). Если *mdel*=`true` (по умолчанию), то несколько идущих подряд разделителей *del* трактуются как единственный разделитель. В противном случае — между ними подразумевается пустая строка. Обратное преобразование выполняется функцией `simplode`.

Листинг 57

```
(%i1)  s:"Maxima is derived from the Macsyma system."$
       stringdisp:true$
       l:charlist(s);

(%o3)  ["M", "a", "x", "i", "m", "a", " ", "i", "s", " ", "d", "e", "r", "i",
"v", "e", "d", " ", "f", "r", "o", "m", " ", "t", "h", "e", " ", "M", "a", "c",
"s", "y", "m", "a", " ", "s", "y", "s", "t", "e", "m", "."]

(%i4)  strigdisp:false$
       simplode(l,"");

(%o5)  Maxima is derived from the Macsyma system.

(%i6)  split(%);

(%o6)  [Maxima, is, derived, from, the, Macsyma, system.]
```

- Функция

  `substring`(*str, n0, n1*),
  `substring`(*str, n0*)

возвращает часть строки *str*, соответствующую позициям от *n0* до *n1* не включая последнее, либо от *n0* до конца строки.

Листинг 58

```
(%i1)  substring("This is a string",6,8);
       substring("This is a string",11);
(%o1)  is
(%o2)  string
```

Строки иногда удобно использовать при работе с числами. В следующем примере выполняется статистический анализ первых 100000 цифр числа π.

Листинг 59

```
(%i1)  fpprec:100000$
       n:string(bfloat(%pi))$
       c:makelist(0,10)$
       for i:3 thru slength(n)-2 do (
         k:1+eval_string(charat(n,i)),
         c[k]:c[k]+1
       )$
       for k:1 thru 10 do print("Цифра ",k-1,": ",c[k])$
```

```
Цифра 0: 9999
Цифра 1: 10137
Цифра 2: 9908
Цифра 3: 10025
Цифра 4: 9970
Цифра 5: 10027
Цифра 6: 10028
Цифра 7: 10025
Цифра 8: 9978
Цифра 9: 9902
```

Результаты более полного анализа, выполненного в специальной программе на мощном компьютере, приведены на страницах сайта http://www.super-computing.org/.

3.5.2. Строка в целом

- Функция

  `string`(*expr*)

преобразует выражение в строку, предварительно его упростив. Упрощение можно отключить с помощью глобальной переменной `simp: false`.

- Функция

  `eval_string`(*str*)

преобразует строку в выражение и выполняет его. При этом строка не обязательно должна заканчиваться символом `$` или `;`. Если строка соответствует нескольким выражениям, то только первое из них выполняется.

Листинг 60

```
(%i1) x:string(2+3);
      numberp(x); /*  число?  */
      stringp(x);
(%o1) 5
(%o2) false
(%o3) true
(%i4) (eco:1, nom:2, ics:3);
      eval_string("eco-nom-ics");
(%o4) 3
(%o5) −4
(%i6) (a:"sin", b:"(%pi/3)", c:"^log", d:"(%e^2)")$
      eval_string(simplode([a,b,c,d]));
(%o7) 3/4
```

- Логические функции

  `sequal`(*str1, str2*),
  `sequalignore`(*str1, str2*)

проверяют, совпадают ли строки. Вторая функция является регистронезависимым аналогом первой (с кириллицей не работает).

Листинг 61

```
(%i1) /*  Латинский алфавит  */
      is("Mathematics"="mathematics");
      sequal("Mathematics", "mathematics");
      sequalignore("Mathematics", "mathematics");
```

```
(%o1) false
(%o2) false
(%o3) true

(%i4) /*  Кириллический алфавит  */
      is("Математика"="математика");
      sequal("Математика", "математика");
      sequalignore("Математика", "математика");

(%o4) false
(%o5) false
(%o6) false
```

♦ Функция

`concat`(*str1, str2, . . .*)

объединяет строки в одну строку. Напомним, что разбить строку на части можно с помощью функции `split`.

Листинг 62

```
(%i1) s1:"Как "$
      s2:"однажды "$
      s3:"Жан-"$
      s4:"звонарь "$
      s5:"головой "$
      s6:"сломал "$
      s7:"фонарь."$
      concat(s1,s2,s3,s4,s5,s6,s7);
      split(%);
(%o8) Как однажды Жан-звонарь головой сломал фонарь.
(%o9) [Как, однажды, Жан-звонарь, головой, сломал, фонарь.]

(%i10) "красный, оранжевый, желтый, эеленый,
          голубой, синий, фиолетовый"$
       split(%,", ");

(%o11) [красный, оранжевый, желтый, эеленый, голубой, синий, фиолетовый]
```

♦ Функция

`sinsert`(*seq, str, n*)

вставляет в строку *str* подстроку *seq*, начиная с *n*-го символа.

- Функции

  `ssubst`(*new, old, str, test, n0, n1*),
  `ssubst`(*new, old, str, test*),
  `ssubst`(*new, old, str, n0, n1*),
  `ssubst`(*new, old, str*),
  `ssubstfirst`(*new, old, str, test, n0, n1*),
  `ssubstfirst`(*new, old, str, test*),
  `ssubstfirst`(*new, old, str, n0, n1*),
  `ssubstfirst`(*new, old, str*)

заменяют в строке *str* все вхождения (либо только первое вхождение) подстроки *old*, содержащиеся на промежутке от *n*0-го до *n*1-го символа, на подстроку *new*. Параметр *test* управляет методом сравнения строк. По умолчанию *test*=`sequal`. Если требуется регистронезависимая замена (с кириллицей не работает), то *test*=`sequalignore`.

Листинг 63

```
(%i1) sinsert("простой ","Вот пример",5);
      ssubst("три","две","Две ноги, две руки, две головы");
      ssubst("три","две","Две ноги, две руки, две головы",
         sequal,10,20);
      ssubstfirst("три","две","Две ноги, две руки, две головы");
(%o1) Вот простой пример
(%o2) Две ноги, три руки, три головы
(%o3) Две ноги, три руки, две головы
(%o4) Две ноги, три руки, две головы
```

- Функция

  `smismatch`(*str1, str2, test*),
  `smismatch`(*str1, str2*)

возвращает позицию первого символа, различного у строк *str1, str2*, либо `false`. Если требуется регистронезависимое сравнение (с кириллицей не работает), то *test*=`sequalignore`.

Обратите внимание, что некоторые разные с точки зрения компьютера символы на экране и на печати выглядят одинаково. Например, это отно-

сится к одинаковым по начертанию буквам латинского и кириллического алфавитов.

Листинг 64

```
(%i1) smismatch("Почему эти строки различны?",
          "Почему эти строки различны?");
      smismatch("А эти одинаковы?", "А эти одинаковы?");
(%o1) 15
(%o2) false
```

- Функция

  `ssearch`(*seq, str, test, n0, n1*),
  `ssearch`(*seq, str, test*),
  `ssearch`(*seq, str, n0, n1*),
  `ssearch`(*seq, str*)

возвращает позицию первого вхождения подстроки *seq* в строку *str*, содержащееся на промежутке от $n0$-го до $n1$-го символа. Если требуется регистронезависимый поиск (с кириллицей не работает), то *test*=`sequalignore`.

Листинг 65

```
(%i1) ssearch("я","Последняя буква в алфавите");
      ssearch("ax",
          "Жили-были Ox и Ax, друг от друга в двух шагах");
(%o1) 8
(%o2) 44
```

- Функция

  `sremove`(*seq, str, test, n0, n1*),
  `sremove`(*seq, str, test*),
  `sremove`(*seq, str, n0, n1*),
  `sremove`(*seq, str*),
  `sremovefirst`(*seq, str, test, n0, n1*),
  `sremovefirst`(*seq, str, test*),
  `sremovefirst`(*seq, str, n0, n1*),
  `sremovefirst`(*seq, str*)

удаляет из строки *str* все вхождения (либо только первое вхождение) подстроки *seq* на промежутке $[n0, n1]$. Если требуется регистронезависимый поиск (с кириллицей не работает), то *test*=`sequalignore`.

Листинг 66

```
(%i1) /*  Научное название загранпаспорта  */
      s:"Паспорт гражданина Российской Федерации,
         удостоверяющий личность
         гражданина Российской Федерации
         за пределами Российской Федерации"$
      sremove(" Российской Федерации",s);
      sremovefirst(" Российской Федерации",s);
(%o2) Паспорт гражданина, удостоверяющий личность гражданина
за пределами
(%o3) Паспорт гражданина, удостоверяющий личность гражданина
Российской Федерации за пределами Российской Федерации
```

♦ Функция

`sreverse`(*str*)

возвращает строку в обратном порядке.

Листинг 67

```
(%i1) /*  Известные фразы-перевертыши  */
      sreverse("а роза упала на лапу азора");
      sreverse("дорого небо да надобен огород");
      sreverse("гни комсомол лом о смокинг");
      sreverse("на в лоб, болван");
(%o1) ароза упал ан алапу азор а
(%o2) дорого небодан ад обен огород
(%o3) гникомс о мол ломосмок инг
(%o4) навлоб ,бол в ан
```

3.6. Списки, множества и массивы

В Maxima имеется три типа данных, аналогичных массиву в языках программирования. Это списки, множества и собственно массивы. *Множество* принципиально отличается от двух других типов тем, что его элементы не имеют нумерации и не могут повторяться (в полном соответствии с математическим понятием множества). Различие между списками и массивами не столь очевидное. Первое и основное отличие *списка* от

массива заключается в том, что все элементы списка должны быть определены в момент его создания. Т.е. нельзя объявить список длины 10, а задать только 2 его элемента. Во-вторых, к элементам списка нельзя обращаться, не задав предварительно собственно список. Поэтому если возникает необходимость использовать «индексированную переменную» $x[k]$ не определяя полностью x, то Maxima будет трактовать x как массив.

3.6.1. Списки общего вида

Список — очень важный тип данных в Maxima (и в Lisp), поскольку любое Maxima-выражение, за исключением чисел, строк и массивов, во внутреннем представлении изображается списком. Это приводит к тому, что некоторые функции, работающие со списками, работают также и с выражениями достаточно общего вида.

Обратите внимание, что нумерация элементов списка начинается с 1.

♦ Список задается явно конструкцией

`[`*elem1, elem2, ...*`]`,

либо функциями `create_list`, `makelist`.

♦ Функция

`makelist`(*form, i, i0, i1*),
`makelist`(*form, i, i1*),
`makelist`(*form, i1*)

— возвращает список из элементов вида *form*, где *form* зависит от целочисленного параметра *i*, пробегающего значения от *i0* до *i1*. Если *i0*=1, то его можно опустить. Если выражение *form* от *i* не зависит, то указание на переменную *i* также можно опустить.

♦ Функция

`create_list`(*form, x1, L1, x2, L2, ...*)

— возвращает список из элементов вида *form*, где *form* зависит от параметров *x1, x2, ...*, пробегающих независимо списки *L1, L2, ...*.

Листинг 68

```
(%i1)  /*  Задание списка явным перечислением элементов  */
       L:[1,2,3,4,[5,6,7]]$
       L[5];
       L[5][2];
```

```
(%o2)  [5,6,7]
(%o3)  6

(%i4)  /*  Значения переменным можно присваивать списком  */
       [a,b,c]:[1,2,3];
       [a,b,c]:[b-c,c-a,a-b];
       [b-c,c-a,a-b];

(%o4)  [1,2,3]
(%o5)  [-1,2,-1]
(%o6)  [3,0,-3]

(%i7)  /*  Построение списка  */
       create_list(1/(i+j),i,[1,2],j,[10,20,30]);
       makelist( random(10)*x+random(10)*y=random(10) ,3);

(%o7)  [1/11, 1/21, 1/31, 1/12, 1/22, 1/32]

(%o8)  [2y + 2x = 4, 4y + 5x = 1, 5y + 9x = 8]
```

- Логическая функция

  `listp`(var)

проверяет, является ли var списком.

- Функция

  `copylist`(L)

возвращает копию списка L. Назначение этой функции проясняется в следующем примере.

Листинг 69

```
(%i1)  a:[1,2,3]$
       /*  Ссылка на тот же самый список  */
       b:a$
       /*  Меняя элементы b, меняем элементы a и наоборот  */
       b[1]:5$
       a[1];
       a[2]:13$
       b[2];
```

```
(%o4) 5
(%o6) 13

(%i7) a:[1,2,3]$
      /*  Теперь это другой список   */
      b:copylist(a)$
      /*  Элементы списка b меняются независимо от a  */
      b[1]:5$
      a[1];
      a[2]:13$
      b[2];

(%o10) 1
(%o12) 2
```

- Функция

  `length`(L)

возвращает длину списка. Работает также с произвольными выражениями.

Листинг 70

```
(%i1) length([a,b,c]);
      length([a,[b,c]]);

(%o1) 3
(%o2) 2

(%i3) length( sin(a)*sin(b)*sin(c) );
      length( sin(a)*(sin(b)+sin(c)) );

(%o3) 3
(%o4) 2
```

- Функции

  `first`(L),
  `second`(L),
  $\ldots$,
  `tenth`(L),
  `last`(L)

возвращают соответственно: 1-й, 2-й, $\ldots$, 10-й, последний элементы списка L. В отличие от команд L`[1]`, L`[2]`, L`[length(`L`)]`, указанные функции

работают не только со списками, но с любыми выражениями.

♦ Функция

`rest`(L, n)

возвращает список L, в котором удалены первые n элементов, если $n > 0$ и последние $|n|$ элементов — если $n < 0$. Функция работает также и с выражениями.

Листинг 71

```
(%i1)  kill(x,y)$
       /*  Работа со списком  */
       P:makelist([x[i],y[i]],i,5);
(%o2)  [[x1, y1], [x2, y2], [x3, y3], [x4, y4], [x5, y5]]
(%i3)  third(P);
       first(last(P));
(%o3)  [x3, y3]
(%o4)  x5
(%i5)  rest(P,2);
       rest(P,-2);
(%o5)  [[x3, y3], [x4, y4], [x5, y5]]
(%o6)  [[x1, y1], [x2, y2], [x3, y3]]
(%i7)  /*  Работа с выражением  */
       E:cos(y)*cos(2*y)*sin(x)*sin(2*x)$
(%i8)  /*  Объясните результаты  */
       first(E);
       rest(E,2);
(%o8)  sin(x)
(%o9)  cos(y) cos(2y)
```

♦ Функция

`unique`(L)

возвращает новый список, равный списку L без совпадающих элементов.

Листинг 72

```
(%i1) makelist((-1)^n,n,5);
      unique(%);
(%o1) [−1, 1, −1, 1, −1]
(%o2) [−1, 1]
```

♦ Функции

`cons`(*elem, L*),
`endcons`(*elem, L*)

ставят элемент *elem* в начало (конец) списка. Возвращают новый список. Эти функции работают также и с выражениями.

Листинг 73

```
(%i1) kill(all)$
      cons(a,[b,c]);
      endcons(d,%);
(%o2) [a, b, c]
(%o3) [a, b, c, d]
```

♦ Функция

`append`(*L1, L2, ...*)

объединяет списки.

Листинг 74

```
(%i1) append([a],[b,c]);
      append([[1,2]],[[a,b],c],[3,[4,5]]);
(%o1) [a, b, c]
(%o2) [[1, 2], [a, b], c, 3, [4, 5]]
```

♦ Логическая функция

`member`(*elem, L*)

проверяет, является ли *elem* элементом списка *L*. Если да, то *elem* можно удалить из списка с помощью функции `delete`. Функция работает с выражениями.

♦ Функция

`delete`(*elem*, *L*)

удаляет все элементы *elem* из списка. Работает также с выражениями.

Листинг 75

```
(%i1) kill(a,b,c,d)$
      member(a,[a,b]);
      member(a,[[a,b],b]);
(%o2) true
(%o3) false
(%i4) delete(a,[a,b,c,[a,b],a,b,c,[a,d]]);
(%o4) [b, c, [a, b], b, c, [a, d]]
```

- Функции

  `setify`(*L*),
  `fullsetify`(*L*)

преобразуют список во множество. Вторая функция рекурсивно проходит по всем вложенным спискам.

Листинг 76

```
(%i1) setify([1,2,3,2,1]);
      fullsetify([1,2,3,[1,2,3],[2,1,3]]);
(%o1) {1, 2, 3}
(%o2) {1, 2, 3, {1, 2, 3}}
```

- Функции

  `permutations`(*L*),
  `random_permutation`(*L*)

возвращают перестановки списка. Первая возвращает множество всех перестановок, вторая — случайную перестановку.

Листинг 77

```
(%i1) /*  Список натуральных чисел от 1 до 3  */
      L:makelist(i,i,3);
      /*  Все перестановки  */
      permutations(L);
      /*  Случайная перестановка  */
```

```
       random_permutation(L);
(%o1)  [1,2,3]
(%o2)  {[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]}
(%o3)  [3,1,2]
```

В следующем примере рассмотрим шифрование текста *методом полиалфавитной замены*. Каждая буква исходного текста заменяется на соответствующую букву из очередного алфавита замены. Алфавиты замены чередуются.

Листинг 78

```
(%i1)  /*  Исходный алфавит  */
       A:makelist(ascii(i),i,192,223);
(%o1)  [А,Б,В,Г,Д,Е,Ж,З,И,Й,К,Л,М,Н,О,П,Р,С,Т,У,Ф,Х,Ц,Ч,Ш,Щ,
Ъ,Ы,Ь,Э,Ю,Я]
(%i2)  /*  Алфавиты замены  */
       B:makelist(random_permutation(A),3);
(%o2)  [[Р,К,И,А,М,Ш,Ю,Б,О,Я,В,Х,Г,У,Ф,Ы,З,П,Т,Ж,Щ,Д,Ъ,Ь,
С,Ц,Й,Н,Э,Ч,Л,Е],
[Р,О,Х,Л,Й,Н,Ф,У,И,М,П,Ы,Э,Ю,Е,Ь,А,Ч,Д,К,Ж,С,Я,В,Г,Ш,Б,Ъ,
З,Ц,Т,Щ],
[Ф,Х,Н,Г,Ъ,С,Ю,П,Д,А,У,К,Щ,О,И,Р,Е,Ч,Б,Ы,М,Й,Л,В,Ж,Ш,Т,З,
Ь,Ц,Я,Э]]
(%i3)  /*  Исходный текст  */
       m:split("ДАЗДРАВСТВУЕТКРИПТОГРАФИЯ","");
(%o3)  [Д,А,З,Д,Р,А,В,С,Т,В,У,Е,Т,К,Р,И,П,Т,О,Г,Р,А,Ф,И,Я]
(%i4)  /*  Шифрование  */
       c:makelist(B[mod(i-1,3)][cint(m[i])-191],i,1,length(m))$
       simplode(c);
(%o5)  МРПМАФИЧБИКСТПЕОЬБФЛЕРЖДЕ
```

♦ Функция

`reverse`(L)

возвращает список в обратном порядке.

- Функция

  `sort`(L, p)

возвращает список, отсортированный в соответствии с логической функцией p. Логическая функция p есть функция двух переменных $p(x, y)$, возвращающая `true`, если x предшествует y. По умолчанию p=`orderlessp` — естественный порядок возрастания.

Листинг 79

```
(%i1)  /*  Создаем список пар заглавных и малых букв  */
       stringdisp:true$
       L:makelist([ascii(i),ascii(i+32)],i,65,69);
(%o2)  [["A", "a"], ["B", "b"], ["C", "c"], ["D", "d"], ["E", "e"]]
(%i3)  /*  Список в обратном порядке  */
       reverse(L);
(%o3)  [["E", "e"], ["D", "d"], ["C", "c"], ["B", "b"], ["A", "a"]]
(%i4)  /*  Сортируем в соответствии с функцией p  */
       kill(x,y)$
       p(x,y):=sposition(x[1],"DECBA")<=sposition(y[2],"cbdea")$
       sort(L,p);
(%o6)  [["B", "b"], ["C", "c"], ["D", "d"], ["E", "e"], ["A", "a"]]
```

- Функции

  `sublist`(L, p),
  `sublist_indices`(L, p)

выделяют те элементы списка L, для которых логическая функция p=`true`. Первая функция возвращает список самих элементов, а вторая — список их индексов.

Листинг 80

```
(%i1)  /*  Генерируем список чисел  */
       L:makelist(random(10000),20);
(%o1)  [7801, 1673, 4425, 7642, 6209, 5879, 1891, 1326, 2609, 3880, 4788,
4008, 2678, 1923, 1099, 1100, 9586, 2873, 7582, 3983]
```

```
(%i2)  /*  Выбираем простые числа  */
       sublist(L,primep);
       sublist_indices(L,primep);

(%o2)  [5879, 2609]
(%o3)  [6, 9]
```

3.6.2. Числовые списки

♦ Над числовыми списками определены арифметические операции: умножение на число, сложение (+), умножение (*) и деление (/) списков. Эти операции работают поэлементно.

С числовыми списками поэлементно работают также некоторые числовые функции (в общем случае такое поведение достигается с помощью функции map).

♦ Определена также операция скалярного умножения (.) работающая как обычное скалярное умножение векторов.

Листинг 81

```
(%i1)  kill(all)$
       /*  Список чисел  */
       l1:makelist(k,k,4);
       /*  Идентификаторы: 4 буквы латинского алфавита  */
       l2:makelist(eval_string(ascii(k)),k,97,100);

(%o2)  [1, 2, 3, 4]
(%o3)  [a, b, c, d]

(%i4)  %alpha+l1;
       %beta*l1;
       %gamma/l1;
       l1^%delta;

(%o4)  [α + 1, α + 2, α + 3, α + 4]
(%o5)  [β, 2 β, 3 β, 4 β]

(%o6)  [γ, γ/2, γ/3, γ/4]

(%o7)  [1, 2^δ, 3^δ, 4^δ]
```

```
(%i8)  sin(l1);
       log(l1);
       %e^l1;

(%o8)  [sin(1), sin(2), sin(3), sin(4)]
(%o9)  [0, log(2), log(3), log(4)]
(%o10) [%e, %e^2, %e^3, %e^4]

(%i11) l1+l2;
       l1*l2;
       l1/l2;
       l1.l2;

(%o11) [a + 1, b + 2, c + 3, d + 4]
(%o12) [a, 2 b, 3 c, 4 d]

(%o13) [1/a, 2/b, 3/c, 4/d]

(%o14) 4 d + 3 c + 2 b + a
```

- Функции

  `lmax`(L),
  `lmin`(L)

возвращают максимальный и минимальный элементы списка.

Листинг 82

```
(%i1)  /*  Создаем список случайных чисел  */
       L:makelist(random(21)-10,10);
       /*  Находим максимальное и минимальное  */
       lmax(L);
       lmin(L);

(%o1)  [5, 2, -8, 1, -9, -10, 5, -3, 0, -5]
(%o2)  5
(%o3)  -10
```

- Функция

  `lsum`$(Xi,\ i,\ L)$

возвращает сумму $\sum_{i \in L} Xi$.

Листинг 83

```
(%i1) [1,2,3,4]$
      lsum(i^2,i,%);
(%o2) 30
(%i3) kill(a)$
      [2,3,5,7,9,13]$
      s:lsum(1/a[i]^2,i,%);
      a:makelist(random(10),15);
      ''s;
```

(%o5) $\frac{1}{a_{13}^2}+\frac{1}{a_9^2}+\frac{1}{a_7^2}+\frac{1}{a_5^2}+\frac{1}{a_3^2}+\frac{1}{a_2^2}$

(%o6) [6, 5, 4, 7, 2, 8, 3, 0, 6, 2, 6, 6, 7, 4, 6]

(%o7) $\frac{90281}{176400}$

3.6.3. Множества

В математике рассматриваются как конечные, так и бесконечные множества. Система Maxima умеет работать только с конечными множествами. Множество отличается от списка тем, что порядок элементов здесь не важен, а совпадающие элементы присутствуют в единственном экземпляре.

- Множество задается перечислением элементов в фигурных скобках

  {*elem1, elem2, ...*}.

- Функция

  `makeset`(*expr*, [*x1, ..., xN*], [[*a11, ..., aN1*], ..., [*a1M, ..., aNM*]])

возвращает множество, составленное из значений выражения *expr* при вектор-переменной [*x1, ..., xN*], пробегающей векторы [*a11, ..., aN1*], ..., [*a1M, ..., aNM*].

Листинг 84

```
(%i1) kill(x,y)$
      makeset(x+y*sqrt(2),[x,y],[[-1,5],[-3,-7],[2,9],[0,-5]]);
```

(%o2) $\{-7\sqrt{2}-3,-5\sqrt{2},5\sqrt{2}-1,9\sqrt{2}+2\}$

- Логическая функция

  `setp`(S)

проверяет, является ли S множеством.

- Логическая функция

  `setequalp`(*S1, S2*)

проверяет, совпадают ли множества *S1, S2*. Для этой цели можно также использовать функцию `is` с оператором равенства `=`.

Листинг 85

```
(%i1) [setp([1,2]), setp({1,2})];
(%o1) [false, true]
(%i2) setequalp({1,2},{2,1});
      is({1,2}={2,1});
      setequalp({1,1,2},{2,1});
      is({1,1,2}={2,1});
(%o2) true
(%o3) true
(%o4) true
(%o5) true
```

- Функции

  `listify`(S),
  `full_listify`(S)

строят список из множества. Вторая функция рекурсивно проходит по всем вложенным множествам. Следует иметь в виду, что список, возвращаемый этими функциями, зависит от принятого упорядочивания выражений. По умолчанию, выражения упорядочиваются в обратном алфавитном порядке.

Листинг 86

```
(%i1) listify({1,2,3,2,1});
      full_listify({1,2,3,{1,2,3},{2,1,3}});
```

```
(%o1) [1, 2, 3]
(%o2) [1, 2, 3, [1, 2, 3]]
```

♦ Логическая функция

`emptyp`(S)

возвращает `true`, если множество S пустое.

♦ Функция

`cardinality`(S)

возвращает число элементов (кардинальное число) множества S.

Листинг 87

```
(%i1) cardinality({1,2,1,1,2});
(%o1) 2
```

♦ Логическая функция

`elementp`(x, S)

проверяет $x \in S$.

♦ Функции

`adjoin`(x, S),
`disjoin`(x, S)

выполняют, соответственно, добавление (удаление) элемента x.

Листинг 88

```
(%i1) adjoin(1,{2,3});
      adjoin(1,{1,2,3});
      disjoin(1,{2,3});
      disjoin(1,{1,2,3});
(%o1) {1,2,3}
(%o2) {1,2,3}
(%o3) {2,3}
(%o4) {2,3}
```

♦ Функции

`union`($S1$, ..., SN),
`intersection`($S1$, ..., SN)

возвращают объединение (пересечение) множеств *S1, ..., SN*.

♦ Функция

`setdifference`(*S1, S2*)

возвращает разность множеств $S1 \setminus S2$.

♦ Логическая функция

`disjointp`(*S1, S2*)

проверяет, являются ли множества *S1, S2* непересекающимися (дизъюнктными).

♦ Логическая функция

`subsetp`(*S1, S2*)

возвращает `true`, если множество *S1* является подмножеством множества *S2*.

♦ Функция

`powerset`(*S*),
`powerset`(*S, n*)

с необязательным аргументом *n* возвращает множество всех подмножеств множества *S*, либо множество всех подмножеств мощности *n*.

♦ Функция

`cartesian_product`(*S1, ..., SN*)

возвращает декартово произведение множеств *S1, ..., SN*, т.е. множество всех списков вида `[`*x1, ..., xN*`]`, где *x1, ..., xN* независимо пробегают соответствующие множества *S1, ..., SN*.

Листинг 89

```
(%i1)  a:{1,2,3,4}$
       b:{3,4,5,6,7}$
       union(a, b);
       intersection(a, b);
       setdifference(a, b);
(%o3)  {1,2,3,4,5,6,7}
(%o4)  {3,4}
(%o5)  {1,2}
(%i6)  a:{1,2,3}$
       powerset(a,2);
```

```
       powerset(a);

(%o7)  {{1,2},{1,3},{2,3}}
(%o8)  {{},{1},{1,2},{1,2,3},{1,3},{2},{2,3},{3}}

(%i9)  kill(x,y)$
       a:{1,2,3}$
       b:{x,y}$
       cartesian_product(a, b);

(%o12) {[1, x], [1, y], [2, x], [2, y], [3, x], [3, y]}
```

♦ Функция

`partition_set`(*S*, *p*)

возвращает список из двух множеств. Первое составляют те элементы множества *S*, для которых *p*=`false`, второе — все остальные элементы множества *S*.

♦ Функция

`equiv_classes`(*S*, *p*)

разбивает множество *S* на классы эквивалентности по отношению *p*. Второй аргумент *p* — имя логической функции от двух переменных, возвращающей `true`, если переменные эквивалентны.

♦ Функция

`extremal_subset`(*S*, *f*, *ext*)

возвращает подмножество множества *S*, на котором числовая функция *f* достигает *ext*=`max` или *ext*=`min`.

Листинг 90

```
(%i1)  kill(x,y)$
       stringdisp:true$
       "Финансовый университет при Правительстве РФ"$
       FU:setify(charlist(%));

(%o4)  {" ", "П", "Р", "Ф", "а", "в", "е", "и", "й", "л", "н", "о", "п",
"р", "с", "т", "у", "ы", "ь" }

(%i5)  p(x):=member(x,charlist("Кафедра математики"))$
       partition_set(FU,p);
```

```
(%o6) [{"П", "Р", "Ф", "в", "й", "л", "н", "о", "п", "с", "у", "ы", "ь"},
{" ", "а", "е", "и", "р", "т"}]

(%i7) p(x,y):=is(mod(cint(x)-cint(y),3)=0)$
      equiv_classes(FU,p);

(%o8) {{" ", "Ф", "а", "й", "п", "т", "ы"}, {"П", "н", "р", "у", "ь"},
{"Р", "в", "е", "и", "л", "о", "с"}}

(%i9) f(x):=cardinality(divisors(cint(x)))$
      extremal_subset(FU,f,min);
      extremal_subset(FU,f,max);

(%o10) {"е", "й", "п", "с", "ы"}
(%o11) {"р"}
```

3.6.4. Применение функций к спискам и множествам

- Логическая функция

  every(*p*, *L1*, ..., *LN*),
  every(*p*, *S*)

применяется к спискам или к множеству. В первом случае функция возвращает `true`, если *p*=`true` на всех наборах *x1*, ..., *xN*, где *x1*, ..., *xN* — соответствующие элементы списков *L1*, ..., *LN*. Во втором случае функция возвращает `true`, если *p*(*x*)=`true` для каждого элемента *x* множества *S*.

- Логическая функция

  some(*p*, *L1*, ..., *LN*),
  some(*p*, *S*)

применяется к спискам или к множеству. В первом случае функция возвращает `true`, если *p*=`true` хотя бы на одном наборе *x1*, ..., *xN*, где *x1*, ..., *xN* — соответствующие элементы списков *L1*, ..., *LN*. Во втором случае функция возвращает `true`, если *p*(*x*)=`true` хотя бы для одного элемента *x* множества *S*.

Листинг 91

```
(%i1) kill(x)$
      every(numberp, {1b-2, 3/7, 0e0, %e});
      every("<", [x, 2, 3, 4], [2, 4, 5, 5]);
```

```
        some(equal, [abs(x), x^2, sqrt(x)], [sqrt(x^2), x, x]);
        some("#", [1, 2, 3, 4], [(-1)^2, sqrt(4), log(%e^3),
                8*sin(%pi/6)]);
(%o2)  false
(%o3)  unknown
(%o4)  true
(%o5)  false
```

♦ Функция

map(*f*, *L1*, *L2*, ...),
map(*f*, *S*)

в первом случае возвращает список, состоящий из значений $f(x1, x2, \ldots)$, где *x1*, *x2*, ... — соответствующие элементы списков *L1*, *L2*, ... Во втором случае функция возвращает множество, состоящее из значений $f(x)$, где x пробегает множество S.

♦ Функция

apply(*f*, *L*)

— применяет N-арную функцию f к списку L.

Использование этих конструкций иногда позволяет избежать циклов и сделать программу более быстрой.

Листинг 92

```
(%i1)  kill(all)$
       L:[a,b,c,d,e]$
       M:[1,2,3,4,5]$
       map(f,L,M);
(%o4)  [f(a,1), f(b,2), f(c,3), f(d,4), f(e,5)]
(%i5)  L:[1,2,3,4,5]$
       S:{1,2,3,4,5}$
       map(f,L);
       map(f,S);
       apply(f,L);
(%o7)  [f(1), f(2), f(3), f(4), f(5)]
(%o8)  {f(1), f(2), f(3), f(4), f(5)}
(%o9)  f(1, 2, 3, 4, 5)
```

```
(%i10) L:makelist(
               makelist(random(5),2+random(3)),
       3+random(2));
       map(length,L);
       apply(append,L);

(%o10) [[1,4,1],[3,2,0],[2,1,3],[2,4,4,1]]
(%o11) [3,3,3,4]
(%o12) [1,4,1,3,2,0,2,1,3,2,4,4,1]
```

Часто приходится использовать `apply` в следующей ситуации. Пусть *f*(*A*, *B*, *x1*, ..., *xN*) — функция, имеющая два обязательных аргумента *A*, *B* и произвольное число аргументов *x1*, ..., *xN*. Последние удобно передать списком *x*=[*x1*, ..., *xN*]. Добавим к этому списку *A*, *B* и применим *f* через `apply`:

`apply`(*f*, `append`([*A*, *B*], *x*))

В дальнейшем мы будем часто пользоваться этим приемом.

В следующем примере преобразуем строку в список ASCII-кодов ее символов, а затем выполняем обратное преобразование.

Листинг 93

```
(%i1)  /*  Преобразуем строку в список ASCII-кодов  */
       "Вот такая строка."$
       charlist(%)$
       map(cint,%);

(%o3)  [194, 238, 242, 32, 242, 224, 234, 224, 255, 32, 241, 242, 240, 238,
234, 224, 46]

(%i4)  /*  Обратное преобразование  */
       [194, 238, 242, 32, 242, 224, 234, 224, 255, 32,
                          241, 242, 240, 238, 234, 224, 46]$
       map(ascii,%)$
       simplode(%,"");

(%o6)  Вот такая строка.
```

♦ Функции

`rreduce`$(f,\ L)$,
`lreduce`$(f,\ L)$,
`tree_reduce`$(f,\ L)$

— рекурсивно применяют бинарную функцию f к списку, соответственно: справа налево, слева направо и разбивая список на пары.

Листинг 94

```
(%i1) kill(all)$
      lreduce(f,[a,b,c,d,e]);
      rreduce(f,[a,b,c,d,e]);
      tree_reduce(f,[a,b,c,d,e]);
(%o2) f(f(f(f(a,b),c),d),e)
(%o3) f(a,f(b,f(c,f(d,e))))
(%o4) f(f(f(a,b),f(c,d)),e)
(%i5) /*  Объясните результаты  */
      lreduce("-",[5,6,7,8]);
      rreduce("^",[4,3,2,1]);
      tree_reduce("/",[1,2,3,4,5,6]);
(%o5) -16
(%o6) 262144
(%o7) 4/5
(%i8) /*  Вычисление НОД от списка чисел  */
      makelist(product(random(10)+1,k,1,4),4);
      lreduce('gcd,%);
(%o8) [1296, 144, 720, 270]
(%o9) 18
```

3.6.5. Массивы

Массивы обычно используются для хранения данных. Это весьма неповоротливый тип данных, для которого практически отсутствуют функции, выполняющие обработку «в целом». Скажем, если есть два числовых списка, то их можно сложить, скалярно перемножить, можно выбрать элементы одного списка, принадлежащие другому списку и т.п. Ничего подобного с массивами сделать нельзя.

Как уже было отмечено выше, если определять отдельные значения индексированной переменной $x[k]$, то Maxima будет трактовать x как массив. Чтобы объяснить Maxima, что x — это список, можно поступить следующим образом. При этом надо обратить особое внимание на значения индекса k.

Листинг 95

```
(%i1)  kill(x)$
       x:makelist(x[k],k,5);
       listp(x);
(%o2)  [x_1, x_2, x_3, x_4, x_5]
(%o3)  true
```

В следующем примере показана возможная проблема с индексацией при записи элементов массива в список. Обратите внимание, что во 2-й части примера переменная x опознается системой как список и массив одновременно. Такая ситуация может стать источником ошибок.

Листинг 96

```
(%i1)  kill(x,a)$
       x:makelist(x[2*k/(k^2+1)],k,-2,3);
       x[1]:a$
       x;
       x[0];
       x[-4/5];
       listp(x);
       arrayinfo(x);
(%o2)  [x_{-4/5}, x_{-1}, x_0, x_1, x_{4/5}, x_{3/5}]
(%o4)  [a, x_{-1}, x_0, x_1, x_{4/5}, x_{3/5}]
apply: no such "list" element: [0]
– an error. To debug this try: debugmode(true);
apply: subscript must be an integer; found: -4/5
– an error. To debug this try: debugmode(true);
(%o7)  true
arrayinfo: x is not an array.
– an error. To debug this try: debugmode(true);
```

```
(%i9)  kill(x,a)$
       x[1]:a$
       x:makelist(x[2*k/(k^2+1)],k,-2,3);
       x[1];
       x[0];
       x[-4/5];
       listp(x);
       arrayinfo(x);

(%o11) [x_{-4/5}, x_{-1}, x_0, a, x_{4/5}, x_{3/5}]
(%o12) a
(%o13) x_0
(%o14) x_{-4/5}
(%o15) true
(%o16) [hashed,1,[1]]
```

В Maxima имеются 2 типа массивов. *Декларированные массивы* — если идентификатор массива заранее объявляется как идентификатор массива. Элементы декларированного массива нумеруются подобно элементам списка, только нумерация начинается с 0. *Недекларированные массивы* (или хэш-таблицы) — в противном случае. Ключами недекларированных массивов могут быть любые выражения. Недекларированные массивы подразделяются на: *индексированные переменные*, *массив-функции* и *индексированные функции* (см. далее).

♦ Функция

`arrayinfo`(*a*)

возвращает информацию о массиве *a*. Если *a* — декларированный массив размерности *N*, то возвращается список вида

`[declared, N, [d1, ..., dN]]`

Для недекларированного массива функция возвращает список:

`[hashed, N, [key1], [key2], ...]`

♦ Функция

`listarray`(*a*)

возвращает список использованных элементов массива *a*. В случае декларированного массива места неиспользованных элементов отмечаются символом `#####`.

♦ Декларированный массив размерности $N \leq 5$ создается конструктором

`array`(*id, type, d1, ..., dN*)
`array`(*id, d1, ..., dN*)

где *id* — имя массива, *d1, ..., dN* — размеры массива, *type* — тип элементов массива:

- *type*=`fixnum` — массив из целых чисел;
- *type*=`flonum` — массив из чисел с плавающей точкой.

Если тип элементов не указан, то элементы массива могут быть произвольными.

Листинг 97

```
(%i1) kill(a,x,y)$
      array(a,2,2);
      a[0,1]:x*y$
      a[2,0]:x+y$
      arrayinfo(a);
      listarray(a);
(%o2) a
(%o5) [declared, 2, [2,2]]
(%o6) [#####, x y, #####, #####, #####, #####, y + x,
#####, #####]
```

Недекларированный массив (хэш-таблица) задается неявно в конструкциях следующих типов.

♦ Индексированная переменная:

a`[`*key1*`]`: *expr1*;
a`[`*key2*`]`: *expr2*;
...,

где как ключи *key1*, *key2*, ... так и значения *expr1*, *expr2*, ... — произвольные выражения.

Листинг 98

```
(%i1) kill(a,x)$
      a[1]:12/7$
      a[2]:x^2$
```

```
       listp(a);
       arrayinfo(a);
       listarray(a);
(%o4)  false
(%o5)  [hashed, 1, [1], [2]]
(%o6)  [12/7, x^2]
(%i7)  kill(a,x)$
       a[-3.45]:1$
       a[x^2*sin(x)/log(x^2+1)]:-1$
       a["или даже так"]:0$
       arrayinfo(a);
       listarray(a);
(%o11) [hashed, 1, [-3.45], [или даже так], [x^2*sin(x)/log(x^2+1)]]
(%o12) [1, 0, -1]
(%i13) kill(a,t,x)$
       a[1,1]:t$
       a[2,3]:t^2$
       a[sin(x),cos(x)]:tan(x)$
       arrayinfo(a);
       listarray(a);
(%o17) [hashed, 2, [1,1], [2,3], [sin(x),cos(x)]]
(%o18) [t, t^2, tan(x)]
```

♦ Массив-функция (`array function`):

a`[`*x1, . . . , xM*`]:=` *expr*;

где *x1, . . . , xM* — переменные индексы. Основное отличие массив-функции от обычной функции $a(x1, \ldots, xM)$ состоит в том, что результаты вызовов массив-функции сохраняются в памяти (*кэшируются*). Таким образом, при повторном вызове массив-функции с теми же аргументами, ее значение не будет вычисляться заново.

Листинг 99

```
(%i1) kill(a,n,m)$
      a[n,m]:=n+m$
      a[3,5];
(%o3) 8
(%i4) a[n,m]:=n*m$
      /*  Здесь будет новое значение:  */
      a[4,6];
      /*  А здесь - старое:  */
      a[3,5];
(%o5) 24
(%o6) 8
```

♦ Индексированная функция (`subscripted function`):

$$a\texttt{[}x1,\ \ldots,\ xM\texttt{]}(t1,\ \ldots,\ tK) := expr;$$

где $x1, \ldots, xM$ — переменные индексы, $t1, \ldots, tK$ — обычные переменные. Индексированная функция работает как массив-функция от $x1, \ldots, xM$ и как обычная функция от $t1, \ldots, tK$. Другими словами, результаты вызова индексированной функции кэшируются по значениям индексов.

Таким образом, функция двух переменных $f(x, y)$ может быть задана тремя различными способами:

$$f\texttt{[}x,\ y\texttt{]}, \quad f\texttt{[}x\texttt{]}(y), \quad f(x,\ y).$$

Первый способ используется в том случае, когда функция вызывается многократно в одной и той же точке (x, y). Второй способ — если приходится многократно вызывать функцию с одним и тем же x, но значения y повторяются не часто. Третий способ применяется тогда, когда функция каждый раз вызывается в различных точках (x, y).

Листинг 100

```
(%i1) kill(a,n,t,x,y)$
      a[n](t):=t^n$
      a[1/2](x);
      a[-1.3](y);
      listarray(a);
(%o3) sqrt(x)
(%o4)  1/y^1.3
```

```
(%o5)  [lambda([t],1/t^1.3),lambda([t],sqrt(t))]
(%i6)  a[n](t):=n^t$
       a[0.5](x);
       a[1/2](y);
       listarray(a);
(%o6)  0.5^x
(%o7)  sqrt(y)
(%o8)  [lambda([t],1/t^1.3),lambda([t],sqrt(t)),lambda([t],0.5^t)]
```

Обратим внимание на блоки вывода (%o5), (%o8). Система сохраняет элементы индексированной функции в виде анонимных функций (lambda-функций). Как и во многих других языках программирования, в Maxima можно создавать функции «на лету», не присваивая им идентификатора. Мы обсудим этот механизм подробно в следующей главе.

Недекларированные массивы часто используются для рекурсии. Продемонстрируем это на примере вычисления членов последовательности по рекуррентным формулам. Обратите внимакние, что в первом блоке мы строим числовую последовательность (числа Фибоначчи), во втором — последовательность списков (строки треугольника Паскаля), а в третьем — последовательность многочленов (так называемых многочленов Лагерра).

Листинг 101

```
(%i1)  kill(n,x)$
       /*  Числа Фибоначчи  */
       f[n]:=f[n-1]+f[n-2]$
       f[0]:1$
       f[1]:1$
       f[100];
       arrayinfo(f);
(%o5)  573147844013817084101
(%o6)  [hashed, 1, [0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13],
[14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28],
[29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43],
```

```
[44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58],
[59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73],
[74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88],
[89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100]]
```

```
(%i7)  /*  Биномиальные коэффициенты  */
       C[n]:=if n=1 then [1,1]
             else cons(0,C[n-1])+endcons(0,C[n-1])$
       C[100][37];
```

```
(%o8)  197720458214493298944377017 5
```

```
(%i9)  /*  Многочлены Лагерра  */
       L[n]:=if n=0 then 1
             elseif n=1 then 1-x
             else expand((2*n-1-x)*L[n-1]-(n-1)^2*L[n-2])$
       L[10];
```

(%o10) $x^{10}-100x^9+4050x^8-86400x^7+1058400x^6-7620480x^5+31752000x^4-72576000x^3+81648000x^2-36288000x+3628800$

Массивы также применяются для хранения информации. В следующем примере в массиве сохраняются данные о пяти правильных многогранниках: тетраэдр, куб, октаэдр, додекаэдр, икосаэдр. Радиусы сфер, площади и объемы приведены для многогранников с единичным ребром.

Листинг 102

```
(%i1)  kill(P)$
       /*  Число вершин, число ребер, число граней  */
       /*  Число сторон у грани, число ребер у вершины  */
       /*  Радиусы вписанной и описанной сферы  */
       /*  Площадь поверхности и объем  */
       P["Тетраэдр"]:[
         [4,6,4],
         [3,3],
         [1/(2*sqrt(6)),sqrt(3)/(2*sqrt(2))],
         [sqrt(3),sqrt(2)/12]
       ]$
       P["Куб"]:[
         [8,12,6],
```

```
        [4,3],
        [1/2,sqrt(3)/2],
        [6,1]
      ]$
      P["Октаэдр"]:[
        [6,12,8],
        [3,4],
        [1/sqrt(6),1/sqrt(2)],
        [2*sqrt(3),sqrt(2)/3]
      ]$
      P["Додекаэдр"]:[
        [20,30,12],
        [5,3],
        [(sqrt(5)+3)/(2^(3/2)*sqrt(5-sqrt(5))),
          (sqrt(3)*sqrt(5)+sqrt(3))/4],
        [(3*5^(3/2)+15)/(sqrt(2)*sqrt(5-sqrt(5))),
          (5^(3/2)+10)/(sqrt(5)-5)]
      ]$
      P["Икосаэдр"]:[
        [12,30,20],
        [3,5],
        [(sqrt(5)+3)/(4*sqrt(3)),5^(1/4)/sqrt(2)],
        [5*sqrt(3),(5^(3/2)+15)/12]
      ]$

(%i7) /*  Список правильных многогранников  */
      arrayinfo(P)$
      rest(%,2)$
      map(first,%);

(%o9) [Додекаэдр, Икосаэдр, Куб, Октаэдр, Тетраэдр]

(%i10) /*  Число граней  */
       L:listarray(P)$
       map(first,L)$
       map(third,%);

(%o12) [12, 20, 6, 8, 4]
```

```
(%i13) /*  Объем единичного многогранника  */
       map(last,L)$
       map(second,%);
```

(%o14) $[\frac{5^{\frac{3}{2}}+10}{\sqrt{5}-5}, \frac{5^{\frac{3}{2}}+15}{12}, 1, \frac{\sqrt{2}}{3}, \frac{1}{3\,2^{\frac{3}{2}}}]$

В качестве более сложного примера, демонстрирующего применение индексированной переменной, рассмотрим частотный анализ употребления букв в тексте.

Листинг 103

```
(%i1)  kill(i,v,k,a,b,c)$
       /*  Исследуемый текст записываем в список  */
       m:charlist("УЖ НЕБО ОСЕНЬЮ ДЫШАЛО,
           УЖ РЕЖЕ СОЛНЫШКО БЛИСТАЛО,
           КОРОЧЕ СТАНОВИЛСЯ ДЕНЬ,
           ЛЕСОВ ТАИНСТВЕННАЯ СЕНЬ
           С ПЕЧАЛЬНЫМ ШУМОМ ОБНАЖАЛАСЬ.
           ЛОЖИЛСЯ НА ПОЛЯ ТУМАН,
           ГУСЕЙ КРИКЛИВЫЙ КАРАВАН
           ТЯНУЛСЯ К ЮГУ: ПРИБЛИЖАЛАСЬ
           ДОВОЛЬНО СКУЧНАЯ ПОРА;
           СТОЯЛ НОЯБРЬ УЖ У ДВОРА.")$
       /*  Анализируем каждый элемент списка  */
       f(c):=if alphacharp(c) then (
         if integerp(i[c]) then i[c]:i[c]+1
         else i[c]:1
       )$
       map(f,m)$
       /*  Значения массива i - частоты букв  */
       v:listarray(i)$
       /*  Ключи массива i - сами буквы  */
       k:rest(arrayinfo(i),2)$
       /*  Печатаем:  */
       p(a,b):=print(a[1],": ",b)$
       map(p,k,v)$
```

А: 19
Б: 5

В: 7
Г: 2
Д: 4
Е: 11
Ж: 7
И: 8
Й: 2
К: 7
Л: 17
М: 4
Н: 18
О: 21
П: 4
Р: 8
С: 16
Т: 7
У: 10
Ч: 3
Ш: 3
Ы: 4
Ь: 8
Ю: 2
Я: 9

3.7. *Задания для самостоятельной работы*

3.1 Используя `print`, напечатайте по-гречески слово «математика»: $\mu\alpha\theta\eta\mu\alpha\tau\iota\kappa\alpha$.

3.2 Для каждого из следующих операторов:

```
+ * and - not ^ or = # ! []
```

определите тип, к которому он относится (префиксный, инфиксный, постфиксный или скобочный; унарный, бинарный или n-арный).

3.3 Объясните разницу между `a=b` и `equal(a,b)`, а также между `a#b`

и `notequal(a,b)`. Здесь `a`, `b` — любые выражения.

3.4 Используя `charfun` и `sum`, запишите следующую функцию:

$$f(x) = \begin{cases} b_0,\ x \leqslant a_1 \\ b_1,\ x \in (a_1, a_2] \\ \dots \\ b_{n-1},\ x \in (a_{n-1}, a_n] \\ b_n,\ x > a_n \end{cases},$$

где $a_1 < a_2 < \ldots < a_n$, $b_0 < b_1 < \ldots < b_n$ — заданные числа.

3.5 Используя функцию `rationalize`, вычислите ошибку округления, возникающую при присвоении `x:0.2`. Возникает ли ошибка при присвоении `x:0.5`? Почему?

3.6 Какую проблему решает функция `copylist`? Может ли аналогичная проблема возникнуть в случае множества? Проанализируйте сходную проблему, возникающую для массивов.

3.7 Объясните разницу между функциями `cons` (или `endcons`) и `append`. Выразите функцию `append` через функцию `cons` с помощью `lreduce`.

3.8 Напишите скрипт, который принимая на входе номер $i = 1, 2, \ldots, 23$, возвращает список, составленный из малой и заглавной i-й буквы греческого алфавита.

3.9 Возьмите текст первых трех абзацев второй части первого тома эпопеи Л.Н. Толстого «Война и мир» и проведите статистический анализ употребления букв. Какая буква встречается в этом тексте чаще всего?

3.10 Некоторый текст на русском языке зашифрован методом простой замены. Знаки препинания и пробелы при шифровании опускались. Расшифруйте текст, применив частотный анализ:

ПЛХЭЪЯЩСЖТБУХСГНЙСЖЯПКБПХЯЭАБВХГПУЙВДТНВЙНУ
ХИУТБЦНАУЕПНАЕВХЕВОДПАТСЙПУВТТНОАЙБДДПЯПЦБГ
ЯПЬВЯРБВХЖСЯЧБВКВЪВЖНХРБЬВЙЩТНЫСЯСЧВОТПГЯСД
СЖВРБУХИЙУПУВДЬПОГПУГВФБВХЕПХШВРЖНВГПЙЭУПАП
ХПЮУЕПВОНУХЯБЖДСХПРНЬНПХЪВФВТПОРБЪБЕИНЪСДНХ
ЙБОУПЪБЛСУТСЕФНВДСЪЯБЕИХВГВЯЩЬПЭГПЯБЭТВЙЮЪЙ
ЮЕВУТИУЛСЧТБЬТВПХХВГВЙЩЕПТЩЦЯЭРЩЕВУТПОЭЪПЙВ
ТЛЯПЕЩЪЯПДНХЧСЕУХЕБСЬХПУЛПЮУХВУТВТИУСЯПЕПЮР
НЬПОЭЪПЙВВДПЕПЙВТЙЮЪЙЮВВУТВЦБЕГЯНУСХУХЕННЙС

ТИЛБЛЙВЦЛНОЪВЦУБТВОУГПДЯСЦПОЪИУХЯНЕПЙВТЛПЦД
БГПДУПЪПЙВЬУПЦЯВХБНУЕВЖБПТБЕБЬЯСЛСЖЬВХГИЙБЭ
НДЯПЖБ

3.11 Рассмотрим десятичное разложение числа e^2. Верно ли, что среди первых 500 цифр дробной части этого числа встречается комбинация 499? Если да, то найдите ее расположение.

3.12 Вовочку отпустят гулять, если согласна мама, либо хотя бы двое из трех других ближайших родственников: папа, дедушка, бабушка. Напишите соответствующую булеву функцию.

3.13 В некотором учебном заведении принят следующий алгоритм перевода 100-бальной оценки в 10-бальную:

0-50	0
51-55	1
56-60	2
...	...
96-100	10

Запрограммируйте этот алгоритм, используя арифметические операторы и функцию `floor` (или `ceiling`).

3.14 Выразите обратные гиперболические функции через натуральный логарифм.

3.15 Пусть x — число с плавающей точкой. Выразите `random`(x) через `random`(1.0). Сделайте тоже самое для случая натурального x.

3.16 Для дискретной случайной величины `random`(1.0) найдите (конечное) множество ее значений и вероятность каждого значения.

3.17 Любое положительное рациональное число $\frac{p}{q}$ может быть представлено в виде конечной цепной дроби:

$$\frac{p}{q} = r + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \ddots \cfrac{1}{q_{n-1} + \cfrac{1}{q_n}}}},$$

где r, $q_1 > q_2 > \ldots > q_n$ – натуральные числа. Напишите соответствующий скрипт.

3.18 Любое натуральное число n может быть разложено по факториалам:

$$n = n_1 \cdot 1! + n_2 \cdot 2! + \ldots + n_m \cdot m!,$$

где $n_1 \in \{0,1\}$, $n_2 \in \{0,1,2\}$, ..., $n_m \in \{0,1,\ldots,m\}$. Напишите соответствующий скрипт.

3.19 Напишите скрипт, принимающий на входе длинное число с плавающей точкой и возвращающий список его цифр. Цифры должны быть в числовом, а не в строковом формате, т.е. на выходе должен быть числовой список.

3.20 Для простого числа $p \geqslant 7$ число $\frac{1}{p}$ записывается бесконечной периодической десятичной дробью. Период[13] этой дроби имеет длину $p-1$. Напишите скрипт, выводящий период и среднее арифметическое цифр периода для простых чисел из заданного промежутка.

3.21 Натуральное число называется *совершенным*, если оно равно полусумме всех своих делителей. Найдите все совершенные числа $\leqslant 10000$.

3.22 Выразите функции `min`, `max` (от двух аргументов) через функцию `abs` и арифметические операторы.

3.23 Выразите функцию `abs` через функцию `max` (или `min`) и арифметические операторы.

3.24 Используя `sum`, проверьте *формулу Архимеда*

$$1^2 + 2^2 + \ldots + n^2 = \frac{1}{6}n(n+1)(2n+1)$$

для $n = 10000$.

3.25 Найдите приближенно предел разности

$$1 + \frac{1}{2} + \ldots + \frac{1}{n} - \ln n,$$

вычислив ее значения при достаточно больших n. Этот предел называется *числом Эйлера-Маскерони* и обозначается `%gamma`.

3.26 Выразите функции `every`, `some` через функции `map`, `apply` и логические операторы `and`, `or`.

3.27 Объясните, почему определена функция `every` от нескольких списков, но не может быть определена функция `every` от нескольких мно-

[13] Точнее говоря, длина наименьшего периода является делителем числа $p-1$.

жеств. Тоже самое верно для `some` и `map`.

3.28 Выразите функцию `simplode` (со вторым аргументом по умолчанию) через функцию `concat`, используя `apply`.

3.29 Используя `map` и `apply`, вычислите произведение чисел $f(k)$, при k, пробегающем список L.

3.30 Перепишите скрипт из листинга 59, используя `map` вместо циклов.

3.31 Напишите скрипт, выводящий по очереди все перестановки элементов данного множества, снабжая их порядковыми номерами.

3.32 Дан список $L = [L_1, \ldots, L_n]$, где $L_1, \ldots, L_n$ — списки одинаковой длины. Напишите скрипт, выполняющий транспонирование двойного списка L как транспонирование матрицы со строками $L_1, \ldots, L_n$.

3.33 Список натуральных чисел $L = [x_1, \ldots, x_n]$ называется *разбиением натурального числа* x, если $x = x_1 + \ldots + x_n$. В разбиении допускаются одинаковые слагаемые. Пусть $x_1, \ldots, x_m$ попарно различны, а в исходном списке L число x_1 попадается k_1 раз, число x_2 — k_2 раз и т.д. Тогда список $L' = [k_1, \ldots, k_m, \ldots]$, в котором число k_1 попадается x_1 раз, число k_2 — x_2 раз и т.д. также является разбиением числа x. Это разбиение называется *сопряженным* исходному. Напишите скрипт, который по данному разбиению числа x выводит его сопряженное.

3.34 Дан список $[e_1, \ldots, e_{3n}]$, состоящий из выражений произвольного вида. Требуется написать скрипт, возвращающий двойной список

$$[[e_1, e_2, e_3], \ldots, [e_{3n-2}, e_{3n-1}, e_{3n}]].$$

3.35 Перестановка p множества $\{1, 2, \ldots, n\}$ задана в виде списка $[p(1), p(2), \ldots, p(n)]$. Напишите скрипт, вычисляющий обратную перестановку.

3.36 Выразите логическую функцию `emptyp` через функцию `cardinality`.

3.37 Выразите логическую функцию `setequalp` через функции `setdifference` и `emptyp`.

3.38 Выразите логическую функцию `disjointp` через функции `intersection` и `emptyp`.

3.39 Выразите значение `cardinality(union(`*S1*, ..., *SN*`))` через всевоз-

можные `cardinality` (`intersection`(...)). Используйте формулу включения-исключения.

3.40 Чему равны `cardinality`(`powerset`(S)) и `cardinality`(`powerset`(S, n)), если `cardinality`(S) $= N$?

3.41 Имеется список множеств $L = [S_1, S_2, \ldots, S_N]$. Почему

$$\texttt{apply}(\texttt{"*"}, \texttt{map}(\texttt{cardinality}, L)) = \\ = \texttt{cardinality}(\texttt{apply}(\texttt{cartesian\_product}, L))?$$

3.42 Дано множество $S = \{1, 2, \ldots, N\}$. Напишите скрипт, который на входе принимает подмножество $A \subseteq S$ и возвращает список $[k_1, k_2, \ldots, k_N]$, где $k_i = 1$ если $i \in A$ и $k_i = 0$ в противном случае.

3.43 Требуется разбить заданную строку на подстроки длины n (последняя подстрока может содержать меньшее количество символов). На выходе должен получиться список подстрок. Проверьте, что примененная к этому списку функция `simplode` дает исходную строку.

3.44 Напишите скрипт, меняющий в заданной строке регистр символов на противоположный.

3.45 Требуется в каждой фразе заданного текста поменять порядок слов на противоположный.

3.46 Последовательность x_n задана рекуррентной формулой $x_n = (-1)^n x_{n-1} + n x_{n-2}$, $x_1 = 1$, $x_2 = 5$. Вычислите x_{100}.

3.47 Последовательность многочленов $p_n(x)$ задана рекуррентной формулой $p_n(x) = x p'_{n-1}(x) - (x+n) p_{n-1}(x-n)$, $p_1(x) = 1$. Вычислите $p_{100}(x)$.

3.48 Для каждого многогранника из базы данных правильных многогранников (листинг 102) проверьте, что:

- число вершин V, число ребер E и число граней G связаны формулой Эйлера: $V - E + G = 2$;
- число p сторон у грани и число q ребер у вершины связаны с V, E, G равенствами: $pG = 2E = qV$;
- объем v связан с радиусом вписанной сферы r и площадью поверхности s формулой $v = \frac{rs}{3}$.

Используя полученные соотношения, докажите, что других правильных многогранников не существует.

3.49 Постройте базу данных по основным элементарным функциям e^x, $\ln(1+x)$, $\sin x$, $\cos x$, $(1+x)^a$, записав для каждой функции ее разложение по формуле Маклорена до 10-го порядка и оценку ошибки этого разложения на промежутке $[-\frac{1}{2}, \frac{1}{2}]$. Разложение по формуле Маклорена можно получить с помощью функции `taylor`. Для оценки ошибки воспользуйтесь выражением для остаточного члена в форме Лагранжа.

4. Описание языка

4.1. Условные операторы и циклы

Во всех приведенных ниже шаблонах синтаксиса *body* обозначает единственное выражение. Если требуется выполнить несколько выражений, то их следует объединить в блок. Под *cond* понимается некоторое высказывание, например: `x^2+y^2<4`.

4.1.1. Операторы `if`

- Оператор

  `if` *cond* `then` *body*

возвращает значение выражения *body*, если `is`(*cond*)=`true`. В противном случае, если `is`(*cond*)= `false`, то оператор `if` возвращает `false`.

Листинг 104

```
(%i1) kill(x,a)$
      f(x):= if x<2 then 5$
      f(1);
      f(3);
      /*  Пусть a>0  */
      assume(a>0)$
      f(2-a);
      f(2+a);
(%o3) 5
(%o4) false
(%o6) 5
(%o7) false
```

- Оператор

  `if` *cond* `then` *body1* `else` *body0*

выполняет *body1*, если `is`(*cond*)=`true` и *body0* в противном случае.

Листинг 105

```
(%i1) kill(n)$
      f(n):= if alphacharp(ascii(n)) then print("Это буква")
          else print("Это не буква")$
```

```
        f(1)$
        f(65)$
        f(125)$
```

Это не буква
Это буква
Это не буква

- Оператор

```
if cond1 then body1
    elseif cond2 then body2
    ...
    else body0
```

выполняет *bodyK*, если `is`(*condK*)=`true`, а все предыдущие `is`(*condI*)=`false`. Если все `is`(*condI*)= `false`, то выполняется *body0*. Переносы строки и отступы добавлены для удобочитаемости.

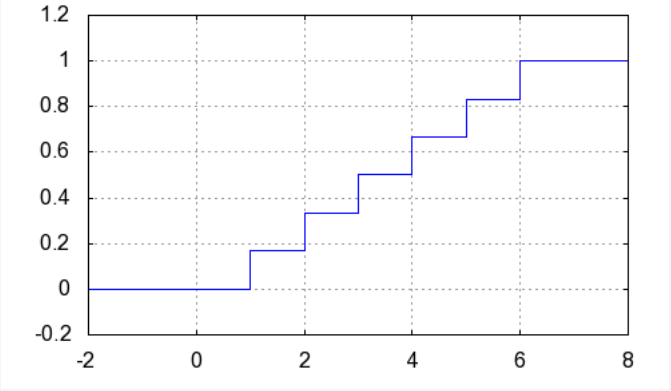
В следующем примере запрограммирована функция распределения дискретной случайной величины[14] — числа очков, выпавших на игральной кости.

Листинг 106

```
(%i1)  kill(x)$
       f(x):= if x<=1 then 0
           elseif x<=2 then 1/6
           elseif x<=3 then 2/6
           elseif x<=4 then 3/6
           elseif x<=5 then 4/6
           elseif x<=6 then 5/6
           else 1$
       /*  Строим график  */
       wxdraw2d(
           yrange=[-0.2,1.2],
           grid=true,
           explicit(f(x), x,-2,8)
       )$
```

[14] Напомним, что функцией распределения случайной величины X называется $F(x) = P(X < x)$. Поскольку в рассматриваемом случае X принимает значения $1, 2, \ldots, 6$, то $F(x) = 0$ при $x \leqslant 1$ и $F(x) = 1$ при $x > 6$.

(%t3)



4.1.2. Операторы `for`

♦ Оператор

`for` *var*:*val* `step` *incr* `thru` *limit* `do` *body*

выполняет *body* пока переменная *var*, увеличиваясь с каждым шагом на *incr*, не превысит значения *limit*. Если *incr*=1, то фрагмент `step` *incr* может быть опущен.

В следующем примере выполняется перевод числа из десятичной системы счисления в двоичную.

Листинг 107

```
(%i1) f(n):=block([r:"",k],
          for k:floor(log(n)/log(2))+1 step -1 thru 1 do (
              n:mod(n,2^k),
              charfun(n/2^k>=1/2),
              r:concat(r,%%)
          ),
          eval_string(r)
      )$
      f(123456789101112);
(%o4) 11100000100100010000110000011110011101000111000
```

♦ Оператор

`for` *var* `in` *list* `do` *body*

выполняет *body* для элементов списка *list*. Причем *list* — не обязательно список, а может быть любым выражением, в этом случае *var* пройдет по всем его подвыражениям. Того же эффекта можно достичь, применив функцию `map`.

В следующем примере для выражения $x^2+\sin x-\frac{x}{\operatorname{tg} x}$ двумя способами выводятся все подвыражения 1-го уровня.

Листинг 108

```
(%i1)  kill(x)$
       for e in x^2+sin(x)-x/tan(x) do print(e)$
```

$$-\frac{x}{\tan(x)}$$

$$\sin(x)$$

$$x^2$$

```
(%i3)  map(print,x^2+sin(x)-x/tan(x))$
```

$$-\frac{x}{\tan(x)}$$

$$\sin(x)$$

$$x^2$$

Вместо цикла всегда можно использовать рекурсию. Продемонстрируем это на примере вычисления арифметико-геометрического среднего двух чисел.

Листинг 109

```
(%i1)  /*  Цикл  */
       kill(a,b)$
       (a[0]:2.0, b[0]:13.0)$
       for i:1 thru 10 do (
          a[i]:(a[i-1]+b[i-1])/2,
          b[i]:(a[i-1]*b[i-1])^(1/2)
       )$
       a[10];
       b[10];
(%o4)  6.241653373328108
(%o5)  6.241653373328108
```

```
(%i6)  /*  Рекурсия  */
       kill(a,b)$
       (a[0]:2.0, b[0]:13.0)$
       a[i]:=(a[i-1]+b[i-1])/2$
       b[i]:=(a[i-1]*b[i-1])^(1/2)$
       a[10];
       b[10];

(%o10) 6.241653373328108
(%o11) 6.241653373328108
```

4.1.3. Операторы `while` / `unless`

♦ Операторы

`while` *cond* `do` *body*
`unless` *cond* `do` *body*

выполняют *body* пока `is`(*cond*) = `true` (соответственно, `false`).

В следующем примере из заданного списка в цикле удаляем его первый элемент.

Листинг 110

```
(%i1)  a:[-2,0,-1,-1,3,-3,-1,3,-1,3]$
       while length(a)>2 do (
           a:delete(a[1],a),
           print(a)
       )$
```

$[0,-1,-1,3,-3,-1,3,-1,3]$
$[-1,-1,3,-3,-1,3,-1,3]$
$[3,-3,3,3]$
$[-3]$

♦ Операторы

`for` *var*:*val* `step` *incr* `while` *cond* `do` *body*
`for` *var*:*val* `step` *incr* `unless` *cond* `do` *body*

выполняют *body* пока `is`(*cond*)=`true` (соответственно, `false`). Переменная *var* с каждым шагом увеличивается на *incr*. Если *incr*=1, то фрагмент `step` *incr* может быть опущен.

♦ Операторы

`for` *var* `in` *list* `while` *cond* `do` *body*
`for` *var* `in` *list* `unless` *cond* `do` *body*

выполняют *body* для элементов списка *list* пока `is`(*cond*) = `true` (соответственно, `false`).

4.1.4. Оператор `go`

♦ Внутри блока `block`(...) можно применять следующую конструкцию:

`block`(..., *label*, ..., `go`(*label*), ...)

— при достижении команды `go`(*label*) выполнение блока продолжается с метки *label*. Меткой может быть любой свободный идентификатор.

В следующем примере выводится список всех простых чисел от 10000 до 10100.

Листинг 111

```
(%i1) block([i:10000,r:[]],
          _1, i:i+1,
          if primep(i) then r:endcons(i,r),
          if i<10100 then go(_1),
          r
      );
(%o1) [10007, 10009, 10037, 10039, 10061, 10067, 10069, 10079, 10091,
10093, 10099]
```

4.2. Пользовательские функции и операторы

4.2.1. Пользовательские функции

Пользовательская функция *fun* задается с помощью оператора `:=`

fun(*x1*, ..., *xN*) := *body*

При этом символу *fun*(*x1*, ..., *xN*) присваивается выражение *body*, а не его значение — в отличие от того, как это происходит для обычного оператора присвоения.

Листинг 112

```
(%i1) kill(x)$
      /*  Случайное целое число -5..5  */
```

```
       r():=random(11)-5$

(%i3)  /*  При таком определении правая часть не вычисляется  */
       f(x):=r()*abs(r()*x)+r()*cos(r()*x);
       f(2);
       f(2);

(%o3)  f(x) := r() |r() x| + r() cos (r() x)
(%o4)  −3 cos(4) − 12
(%o5)  2 cos(6)

(%i6)  /*  А теперь правая часть вычисляется  */
       g(x):=''(r()*abs(r()*x)+r()*cos(r()*x));
       g(2);

(%o6)  g(x) := 2 |x| − 4 cos(2 x)
(%o7)  4 − 4 cos(4)
```

Обратите внимание на функцию $r()$ с пустым аргументом. Такая функция задается и вызывается с пустыми скобками `()`.

Идентификатор может быть одновременно именем функции и именем переменной.

Листинг 113

```
(%i1)  kill(all)$
       f(x):=x^2$
       f(2);
       f;
       f:1;
       f(2);

(%o3)  4
(%o4)  f
(%o5)  1
(%o7)  4
```

Если некоторая вспомогательная функция $g(t)$ определяется в теле другой функции $f()$, то $g(t)$ становится доступна глобально после вызова $f()$. Если необходимо определить $g(t)$ локально, то надо воспользоваться следующей функцией.

♦ Функция

`local`(*f1*, . . . , *fN*)

объявляет пользовательские функции с идентификаторами *f1*, . . . , *fN* как локальные. Возвращает `done`. Может употребляться только в теле блока или `lambda`-функции.

Листинг 114

```
(%i1)  kill(all)$
       f():=block([g],
           g(t):=t^2,
           g:1,
           h(t):=t^3
       )$
       /*  Здесь пока функции g(t), h(t) не определены  */
       g(2);
       g;
       h(2);
       /*  Вызываем f()  */
       f()$
       /*  Функции доступны, но переменная g еще свободна  */
       g(2);
       g;
       h(2);
(%o3)  g(2)
(%o4)  g
(%o5)  h(2)
(%o7)  4
(%o8)  g
(%o9)  8
(%i10) kill(all)$
       f():=block(
           local(g),
           g(t):=t^2,
           h(t):=t^3
       )$
       g(2);
       h(2);
       /*  Вызываем f()  */
```

```
        f()$
        /*  Теперь функция h доступна, а функция g - нет  */
        g(2);
        h(2);
```

(%o12) $g(2)$
(%o13) $h(2)$
(%o15) $g(2)$
(%o16) 8

Математическая функция не обязательно должна задаваться в виде, описанном выше. Во многих ситуациях бывает удобнее вместо математической функции рассматривать определяющее ее математическое выражение, присвоив его некоторой переменной. Такой подход часто применяется и в математике, сравните: $f(x) = x^2$ и $y = x^2$.

Листинг 115

```
(%i1)  kill(x,y)$
       /*  Определим переменную z  */
       z:x^2*sin(x*y)$
       /*  Проинтегрируем z по переменной x  */
       w:integrate(z,x);
```

(%o3) $$\frac{2xy\,\sin(xy) + \left(2 - x^2y^2\right)\cos(xy)}{y^3}$$

```
(%i4)  /*  Подставим в результат значения переменных x, y  */
       subst([x=1/4,y=-%pi],w);
```

(%o4) $$-\frac{\frac{2-\frac{\pi^2}{16}}{\sqrt{2}} + \frac{\pi}{2^{3/2}}}{\pi^3}$$

```
(%i5)  /*  Упрощаем  */
       ratsimp(%);
```

(%o5) $$\frac{\sqrt{2}\,\pi^2 - 2^{7/2}\,\pi - 2^{11/2}}{32\,\pi^3}$$

Рассмотрим задание функции, принимающей произвольное число аргументов. Общий синтаксис таков:

$$f(x1,\ \ldots,\ xN,\ [y]),$$

где *x1, . . . , xN* — обязательные аргументы, [*y*] — необязательные аргументы, передаваемые функции в виде списка *y*.

Листинг 116

```
(%i1) kill(a,b,x)$
      f(a,b,[x]):=a*b+lsum(
        if oddp(k) then a*k else b*k,
      k,x)$
      f(1,2,3,4);
      f(1,2,3,4,5);
(%o3) 13
(%o4) 18
```

Рассмотрим задание функции, принимающей в качестве аргументов списки и множества.

Листинг 117

```
(%i1) kill(x)$
      f(x):=x[1]+x[2]-x[3]$
      a:[1,2,3]$
      b:[4,5,6]$
      f(a+2*b);
(%o4) 6
(%i5) kill(x,t)$
      p(t):=is(mod(t,3)=1)$
      f(x):=sublist(x, p)$
      makelist(i,i,-20,20)$
      f(%);
(%o7) [−20, −17, −14, −11, −8, −5, −2, 1, 4, 7, 10, 13, 16, 19]
```

Рассмотрим примеры рекурсивного построения функции. Построим так называемые многочлены Чебышева.

Листинг 118

```
(%i1) kill(n,x)$
      T(n,x) := if n=0 then 1
        elseif n=1 then x
```

```
        else expand(2*x*T(n-1,x)-T(n-2,x))$
      T(19,x);
```

(%o3) $262144x^{19}-1245184x^{17}+2490368x^{15}-2723840x^{13}+1770496x^{11}-695552x^9+160512x^7-20064x^5+1140x^3-19x$

А теперь построим индексированную функцию. Напомним, что это один из видов недекларированного массива. Многочлены, которые мы будем вычислять, называются многочленами Эрмита.

Листинг 119

```
(%i1) kill(n,x)$
      /*  Без условного оператора  */
      H[n](x):=expand(2*x*H[n-1](x)-2*(n-1)*H[n-2](x))$
      H[0](x):=1$
      H[1](x):=2*x$
      H[15](x);
```

(%o5) $32768x^{15}-1720320x^{13}+33546240x^{11}-307507200x^9+1383782400x^7-2905943040x^5+2421619200x^3-518918400x$

```
(%i5) /*  С условным оператором  */
      H[n](x):=if n=0 then 1
        elseif n=1 then 2*x
        else expand(2*x*H[n-1](x)-2*(n-1)*H[n-2](x))$
      H[15](x);
```

(%o7) $32768x^{15}-1720320x^{13}+33546240x^{11}-307507200x^9+1383782400x^7-2905943040x^5+2421619200x^3-518918400x$

4.2.2. Анонимные функции

Во многих Maxima-функциях, таких как:

`makelist`, `sum`, `map`, `equiv_classes` и т.д.

в одном из аргументов передается некоторая зависимость от переменной. При этом в

`makelist`, `sum` и др.

эта зависимость передается выражением, а в

`map`, `equiv_classes` и др.

— функцией. В последнем случае можно либо заранее построить необходимую пользовательскую функцию, либо воспользоваться анонимной функцией.

♦ Конструкция

`lambda`([*x1, x2, ...*], *expr1, expr2, ..., exprN*)

создает анонимную пользовательскую функцию: *x1, x2, ...* — ее аргументы, *expr1, expr2, ..., exprN* — тело функции. Функция возвращает значение последнего выражения *exprN*.

Листинг 120

```
(%i1)  /*  Сортировка списка  */
       L:makelist(random(10000),10);
       sort(L,lambda([x,y],
         totient(x)<totient(y)
       ));
(%o1)  [1099, 1100, 9586, 2873, 7582, 3983, 1720, 6969, 2059, 8444]
(%o2)  [1100, 1720, 1099, 2059, 2873, 3983, 7582, 8444, 6969, 9586]

(%i3)  /*  Отображение списка  */
       L:makelist(random(10000),10);
       map(lambda([x],
               cardinality(divisors(x))-2
       ),L);

(%o3)  [1612, 9302, 6734, 4585, 4204, 5391, 9429, 3985, 2298, 7403]
(%o4)  [10, 2, 14, 6, 4, 4, 6, 2, 6, 2]

(%i5)  /*  Выбор подсписка  */
       L:makelist(random(10000),20);
       sublist(L,lambda([x],
               cardinality(divisors(x))>10
       ));

(%o5)  [7801, 1673, 4425, 7642, 6209, 5879, 1891, 1326, 2609, 3880, 4788,
4008, 2678, 1923, 1099, 1100, 9586, 2873, 7582, 3983]
(%o6)  [4425, 1326, 3880, 4788, 4008, 1100]
```

```
(%i7)  /*  Подмножества и классы эквивалентности  */
       S:{-6,3,7,-9,5,4,%pi,%e^2}$
       partition_set(S, lambda([x],
               (x<-7) or (x>5)
       ));
       equiv_classes(S, lambda([x,y],
               integerp((x-y)/3)
       ));

(%o8)  [{-6,3,4,5,π},{-9,7,%e^2}]
(%o9)  {{-9,-6,3},{4,7},{5},{%e^2},{π}}

(%i10) /*  Экстремальные подмножества  */
       S:{%pi,%e,%pi^2,%e^2,%pi*%e,%pi+%e,%pi-%e}$
       extremal_subset(a, lambda([x],
               if x<%pi then 1/x else x^2
       ), min);
       extremal_subset(a, lambda([x],
               if x<%pi then 1/x else x^2
       ), max);

(%o11) {%e}
(%o12) {π^2}
```

Работа со списками существенно упрощается благодаря функции `map`, применение которой почти немыслимо без анонимных функций. В следующем примере для множества $S = \{1, 2, \ldots, n\}$ рекурсивно выводятся все его разбиения[15].

Листинг 121

```
(%i1)  kill(n)$
       F(n):=if n=1 then [[{1}]] else block([r:F(n-1),a,i],
         a:map(lambda([t],cons({n},t)),r),
         for i:1 thru length(r) do
           a:append(a,
             map(lambda([t],
               subst(t=adjoin(n,t),r[i])
```

[15] Напомним, что разбиением множества S называется любой способ представить S в виде объединения попарно непересекающихся непустых множеств.

```
            ),r[i])
          ),
        map(sort,a)
      )$
      F(3);
```

(%o3) $[[\{1\},\{2\},\{3\}],[\{1,2\},\{3\}],[\{1,3\},\{2\}],[\{1\},\{2,3\}],[\{1,2,3\}]]$

4.2.3. Пользовательские операторы

Можно определить новый оператор любого типа. Обратите внимание, что идентификатор не может быть одновременно именем оператора и именем переменной (или функции).

- Функции

  `prefix`(*op*),
  `infix`(*op*),
  `postfix`(*op*),
  `matchfix`(*lop*, *rop*),
  `nary`(*op*)

регистрируют соответственно: префиксный, инфиксный, постфиксный, скобочный и n-арный оператор.

Рассмотрим примеры задания пользовательских операторов.

Листинг 122

```
(%i1) kill(a,x)$
      prefix("o")$
      "o"(a):=a^2;
      o x;
      o 3/4;
      o(3/4);
```

(%o3) $\mathrm{o}(a) := a^2$
(%o4) x^2
(%o5) $\frac{9}{4}$
(%o6) $\frac{9}{16}$

```
(%i7)  kill(a,b,x)$
       infix("~")$
       "~"(a,b):=a*b^2;
       x~5;
```

(%o9) $(a \sim b) := a\,b^2$

(%o10) $25\,x$

```
(%i11) kill(x,y)$
       postfix("?")$
       "?"(x):=1/x;
       2?;
       x+y?;
```

(%o13) $?\,(x) := \dfrac{1}{x}$

(%o14) $\dfrac{1}{2}$

(%o15) $\dfrac{1}{y} + x$

```
(%i16) kill(a,n,x,y)$
       matchfix("<<",">>")$
       <<[x]>>:=lsum(a[n],n,x);
       <<1,-4,8>>;
       <<x,x,y,y>>;
```

(%o18) $<<[x]>> := \displaystyle\sum_{n\,\mathrm{in}\,x} a_n$

(%o19) $a_8 + a_1 + a_{-4}$

(%o20) $2\,a_y + 2\,a_x$

```
(%i21) kill(x,a)$
       nary("##")$
       "##"([x]):=lmax(x);
       1##5##(-7)##8;
       a##3##5;
```

(%o23) $\#\#([x]) := \mathrm{lmax}(x)$

(%o24) 8

(%o25) $\max(5, a)$

4.3. Выражения

4.3.1. Типы выражений

Атом или *атомарное выражение* — это идентификатор, число (целое либо с плавающей точкой), или строка.

Алгебраическое выражение содержит числа (в том числе константы `%e`, `%pi`), числовые переменные, операторы `+ - * / ^` и числовые функции (`sin`, `atan`, `log` и т.п.).

Интегро-дифференциальное выражение содержит дополнительно операции дифференцирования и интегрирования (`diff`, `integrate`).

Векторно-матричное выражение содержит числа, числовые матрицы, числовые и матричные переменные, числовые и матричные операторы и функции.

Выражение, относящееся к одному из трех перечисленных выше типов либо к их комбинации, будем называть *математическим выражением.*

Уравнение имеет вид *expr1* `=` *expr2*, где *expr1, expr2* — математические выражения. В частных случаях говорят о полиномиальном, иррациональном, трансцендентном, матричном, дифференциальном и интегральном уравнениях.

Математическое тождество имеет вид `equal`(*expr1, expr2*), где *expr1, expr2* — математические выражения. Тождество отличается от уравнения тем, что равенство должно быть справедливо при всех допустимых значениях входящих в него переменных.

Неравенство имеет вид *expr1* `o` *expr2*, где *expr1, expr2* — алгебраические либо интегро-дифференциальные выражения, `o` — оператор сравнения:

```
< <= > >= #
```

либо вид `notequal`(*expr1, expr2*) — отрицание тождественного равенства.

Логическое выражение содержит булевы константы (`true`, `false`) и переменные, логические операторы (`not`, `and`, `or`) и логические функции. Логическое выражение может включать в себя в качестве подвыражений математические уравнения, тождества и неравенства. Таким образом последние можно считать частным случаем логических выражений.

Самым общим типом выражений является *Maxima-выражение*. Оно содержит любые определенные в Maxima типы данных, любые переменные, операторы и функции, управляющме конструкции (условные операторы и циклы). Оно может включать в себя в качестве подвыражений любые математические и логические выражения.

Каждое Maxima-выражение состоит из атомов: целых чисел и чисел с плавающей точкой, строк и идентификаторов.

Для понимания работы некоторых функций необходимо знать о *внутреннем представлении* выражений в системе Maxima. Внутреннее представление в Maxima — это польская нотация. Любое выражение, не являющееся атомом или массивом, изображается списком. Например:

- `sqrt(2)` → `[sqrt,2]`
- `x[3]^2` → `["^",[x,3],2]`
- `x+y+z` → `["+",x,y,z]`
- `a-b` → `["+",a,["-",b]]`
- `f(x,y,g(x,y))` → `[f,x,y,[g,x,y]]`
- `(x^2-4*x+5)/sin(7*x)` →
```
["/",
  ["+",["^",x,2],["*",["-",4],x],5],
  [sin,["+",7,x]]
]
```
- `[1,2,{1,2}]` → `["[",1,2,["{",1,2]]`
- `if x<1 then y` → `[if,["<",x,1],y,true,false]`
- `if x<1 then y else z` → `[if,["<",x,1],y,true,z]`
- `if x<1 then y elseif x<2 then z else w` →
```
[if,["<",x,1],y,["<",x,2],z,true,w]
```
- `for i:1 thru n do y` → `[mdo,i,1,false,false,n,false,y]`
- `for i:1 step k thru n while x do w` →
```
[mdo,i,1,k,false,n,[not,x],w]
```
- `for i:1 next k(i) thru n unless y do w` →
```
[mdo,i,1,false,[k,i],n,y,w]
```

- `for i in a while x do w` →
  `[mdoin,i,a,false,false,false,[not,x],w]`

Поэтому некоторые функции, работающие со списками, работают также и с произвольными выражениями. Например: `length`, `first`, `second`, `last`, `rest`, `cons`, `endcons`, `member`, `delete`.

Листинг 123

```
(%i1) kill(all)$
      /*  f(x,y) = [f,x,y]  */
      cons(a,f(x,y));
      endcons(a,f(x,y));
(%o2) f(a,x,y)
(%o3) f(x,y,a)
(%i4) /*  a^2+2*b = [+,[^,a,2],[*,2,b]]  */
      member(a,a^2+2*b);
      /*  a^2 = [^,a,2]  */
      member(a^2,a^2+2*b);
(%o4) false
(%o5) true
(%i6) /*  (a+b)*(a-b) = [*,[+,a,b],[+,a,[-,b]]]  */
      delete(a,(a+b)*(a-b));
      delete(a+2*b,(a+b)*(a+2*b)*(a+3*b));
      /*  2*x*sin(x)+sin(x) = [+,[*,2,x,[sin,x]],[sin,x]]  */
      delete(sin(x),2*x*sin(x)+sin(x));
(%o6) (a+b)(a-b)
(%o7) (a+b)(a+3b)
(%o8) 2x sin(x)
```

Каждый подсписок или атом во внутреннем представлении соответствует (полному) *подвыражению*. Элементы основного списка называются подвыражениями 1-го уровня, их элементы — подвыражениями 2-го уровня и т.д. Например, выражение `(x^2-4*x+5)/sin(7*x)` имеет 2 подвыражения 1-го уровня: `x^2-4*x+5`, `sin(7*x)` и 4 подвыражения 2-го уровня: `x^2`, `-4*x`, `5`, `7*x`.

Первый элемент каждого списка или подсписка является именем оператора (или функции) и имеет индекс 0. Каждый подсписок или атом

имеет *адрес*, состоящий из индексов в списках. Это позволяет функциям типа `part` возвращать подвыражение по его адресу.

4.3.2. Преобразование выражений

Рассмотрим функции, позволяющие, в определенном смысле, работать с выражениями как со строками. Например: разбивать выражение на части, заменять одни части на другие и т.п.

- Логическая функция

  `atom`(*expr*)

возвращает `true`, если выражение *expr* атомарно, и `false` в противном случае.

Листинг 124

```
(%i1)  [atom(1234), atom(\1234), atom(%e),
          atom("вот и строка"), atom(sqrt(2))];
(%o1)  [true, true, true, true, false]
```

- Функция

  `concat`(*atom1, atom2, ...*)

объединяет атомарные выражения *atom1, atom2, ...*. Возвращает идентификатор, если *atom1* — идентификатор, или строку — в противном случае.

Листинг 125

```
(%i1)  x:5$
       x1:2$
       concat(x,1);
       concat('x,1);
       ''%;
(%o3)  51
(%o4)  x1
(%o5)  2
```

- Функция

  `sconcat`(*expr1, expr2, ...*)

объединяет выражения *expr1, expr2, ...* в строку.

Листинг 126

```
(%i1) sconcat(sin,"(",%pi,")");
      eval_string(%);
(%o1) sin(%pi)
(%o2) 0
```

♦ Логическая функция

`freeof`(*part1, ..., partN, expr*)

проверяет, отсутствуют ли в выражении *expr* части, равные *part1, ..., partN*.

Листинг 127

```
(%i1) kill(x,y,z)$
      e: 'diff(y^x,x)/(2*y*z[1]^x)+z[2]^(3*y)/integrate(x^y,y);
(%o2) diff(y^x,x)/(2*y*z[1]^x)+z[2]^(3*y)/integrate(x^y,y)
(%i3) freeof(y^x,e);
(%o3) false
(%i4) freeof('diff,e);
      freeof('integrate,e);
(%o4) false
(%o5) true
(%i6) freeof("-",e);
(%o6) true
(%i7) freeof(z,e);
      freeof(z[4],4*y,x^2,e);
(%o7) false
(%o8) true
```

♦ Функция

`op`(*expr*)

возвращает главный оператор выражения *expr*, т.е. оператор (или функцию), который будет выполняться в последнюю очередь. Тождественна `part`(*expr, 0*) — см. ниже.

Листинг 128

```
(%i1)  kill(a,b,c)$
       stringdisp: true$
       op(a*b*c);
       op(a*b+c);
       op('sin(a+b));
(%o3)  "*"
(%o4)  "+"
(%o5)  sin
```

♦ Функция

`part`(*expr, n1, ..., nN*)

возвращает подвыражение N-го уровня выражения *expr*, определяемое индексами *n1, ..., nN*. На 1-м уровне берется *n1*-е подвыражение, на 2-м — *n2*-е и т.д.

Листинг 129

```
(%i1)  kill(x,y)$
       e: (4*x^y-'sum(3*x/(k^2+y^2),k,1,inf))/(5/x+3*atan(y/x));
(%o2)  (4x^y - 3x sum(1/(y^2+k^2), k, 1, inf)) / (3 atan(y/x) + 5/x)
(%i3)  stringdisp:true$
       [part(e,0), part(e,1), part(e,2)];
(%o4)  ["/", 4x^y - 3x sum(1/(y^2+k^2), k, 1, inf), 3 atan(y/x) + 5/x]
(%i5)  part(e,1,2);
(%o5)  -3x sum(1/(y^2+k^2), k, 1, inf)
(%i6)  part(e,1,2,1);
```

```
(%o6)  3x sum(1/(y^2+k^2), k, 1, inf)
(%i7)  [part(e,1,2,1,0), part(e,1,2,1,1), part(e,1,2,1,2),
          part(e,1,2,1,3)];
(%o7)  ["*", 3, x, sum(1/(y^2+k^2), k, 1, inf)]
(%i8)  makelist(part(e,1,2,1,3,n), n, 0,
          length(part(e,1,2,1,3)));
(%o8)  [sum, 1/(y^2+k^2), k, 1, inf]
```

В следующем более сложном примере для данного выражения рекурсивно выводятся все его операторы.

Листинг 130

```
(%i1)  kill(all)$

       /*  Шаг рекурсии  */
       S(E):=block([e],
           r:endcons(op(E),r), /*  добавляем в результат  */
           for e in E do if not atom(e) then S(e)
       )$

       /*  Старт рекурсии  */
       F(f):=(
           r:[], /*  глобальная переменная для результата  */
           S(f),  /*  запускаем рекурсию  */
           r  /*  выдаем результат  */
       )$

       log(y^2-3*sqrt(x))*asin(sqrt(y-1))/
          (cot(2*y-z)+sin(3*z^2)^(-6/8));
       F(%);
(%o4)  asin(sqrt(y-1))*log(y^2-3*sqrt(x))/(1/sin(3*z^2)^(3/4)-cot(z-2*y))
```

(%o5) $[/, *, asin, sqrt, +, log, +, \hat{}, -, *, sqrt, +, /, \hat{}, sin, *, \hat{}, /, -, cot, +, -, *]$

♦ Функции

`lhs`(*expr*),
`rhs`(*expr*)

тождественны `part`(*expr*, 1), `part`(*expr*, 2) в случае, когда главный оператор выражения *expr*, т.е. `part`(*expr*, 0), является одним из следующих:

```
< <= = # equal notequal >= > := :
```

Листинг 131

```
(%i1) kill(x)$
      eq:expand((x-1)*(x+2)*(x+5))=0;
      solve(eq,x);
      map(rhs,%);
```

(%o2) $x^3 + 6\,x^2 + 3\,x - 10 = 0$
(%o3) $[x = 1, x = -2, x = -5]$
(%o4) $[1, -2, -5]$

♦ Функция

`substpart`(*ins, expr, n1, . . . , nN*)

заменяет подвыражение N-го уровня выражения *expr*, находящееся по адресу *n1, . . . , nN*, на выражение *ins*.

♦ Функция

`args`(*expr*)

тождественна `substpart`("[", *expr*, 0). Таким образом, заменяя главный оператор на оператор списка, функция возвращает список подвыражений 1-го уровня (вместо исходного оператора применяется оператор списка).

Листинг 132

```
(%i1) kill(x,y)$
      e: (4*x^y-'sum(3*x/(k^2+y^2),k,1,inf))/(5/x+3*atan(y/x));
      substpart('integrate,e,1,2,1,3,0);
```

(%o2) $$\frac{4x^y - 3x\sum\limits_{k=1}^{\infty}\frac{1}{y^2+k^2}}{3\,\mathrm{atan}\left(\frac{y}{x}\right) + \frac{5}{x}}$$

```
(%o3)  (4x^y - 3x * integral_{k=1}^{inf} dk/(y^2+k^2)) / (3 atan(y/x) + 5/x)
(%i4)  args( part(e,1,2,1,3) );
(%o4)  [1/(y^2+k^2), k, 1, inf]
```

♦ Функция

`subst`([*sch1*=*ins1*, *sch2*=*ins2*, ...], *expr*)
`subst`(*sch1*=*ins1*, *expr*)
`subst`(*ins1*, *sch1*, *expr*)

выполняет последовательные замены в выражении *expr* полных подвыражений *sch1*, *sch2*,... на выражения *ins1*, *ins2*, ...

Листинг 133

```
(%i1)  kill(x,y,z)$
       e: 2*x[1]*y[3]-3*x[2]*y[1]+x[2]*x[1];
       subst([x=y],e);
(%o2)  2x_1 y_3 - 3y_1 x_2 + x_1 x_2
(%o3)  2y_1 y_3 - 2y_1 y_2
(%i4)  e1: (x+y+z)*f(x+y)-(x+y)*g(x+y+z)$
       subst([x+y=w],e1);

       simp:false$   /*  Отключаем упрощение выражений  */
       e2: ((x+y)+z)*f(x+y)-(x+y)*g((x+y)+z)$
       subst([x+y=w],e2);
       simp:true$
(%o5)  (x+y+z)f(w) - wg(x+y+z)
(%o8)  (w+z)f(w) - wg(w+z)
```

♦ Функции

`sublis`([*x1*=*ins1*, *x2*=*ins2*, ...], *expr*),
`psubst`([*sch1*=*ins1*, *sch2*=*ins2*, ...], *expr*)

выполняют параллельные замены в выражении *expr*. Первая заменяет идентификаторы *x1*, *x2*,..., вторая — полные подвыражения *sch1*, *sch2*,... на выражения *ins1*, *ins2*, ...

Листинг 134

```
(%i1) /*  Здесь можно использовать sublis, psubst  */
      kill(x,y,z,f,g)$
      e1: 2*x[1]*y[3]-3*x[2]*y[1]+x[2]*x[1];
      sublis([x=y, y=x],e1);
```

(%o2) $2x_1y_3 - 3y_1x_2 + x_1x_2$
(%o3) $2y_1x_3 - 3x_1y_2 + y_1y_2$

```
(%i4) /*  А здесь работает только psubst  */
      e2: (x+y+z)*f(x+y)-(x+y)*g(x+y+z)$
      psubst([x+y=x+y+z, x+y+z=x+y],e2);
```

(%o5) $(x+y)f(x+y+z) - (x+y+z)g(x+y)$

♦ Функция

`ratsubst`(*ins, sch, expr*)

заменяет все вхождения части *sch* в выражении *expr* на выражение *ins*. Часть *sch* — не обязательно подвыражение, как в случае функции `subst`, которая выполняет чисто синтаксическую подстановку.

Листинг 135

```
(%i1) kill(x,y,z,w,f,g)$
      e: 2*x[1]*y[3]-3*x[2]*y[1]+x[2]*x[1];
      ratsubst(y,x,e); /*  Не работает  */
      ratsubst(y[1],x[1],e);
```

(%o2) $2x_1y_3 - 3y_1x_2 + x_1x_2$
(%o3) $2x_1y_3 + (x_1 - 3y_1)x_2$
(%o4) $2y_1y_3 - 2y_1x_2$

```
(%i5) e: (x+y+z)*f(x+y)-(x+y)*g(x+y+z)$
      ratsubst(w,x+y,e);
```

(%o6) $(x+y+z)f(w) - wg(x+y+z)$

```
(%i7) e: x^4*y^3 + x^4*y^8$
      ratsubst(z,x*y^2,e);
```

(%o8) $x^3yz + z^4$

4.3.3. Упрощение и вычисление

В каждом классе выражений (рациональные, иррациональные, тригонометрические, показательно-логарифмические, дифференциальные) имеются свои специфические функции для упрощения и параметры, управляющие этим процессом. Здесь мы рассмотрим только самые общие.

♦ Логическая переменная `simp` — упрощать ли выражение. Упрощение состоит в следующем:

- приведение подобных слагаемых в суммах и подобных множителей в произведениях;
- подстановка табличных значений основных элементарных функций;
- упорядочение слагаемых и множителей в обратном алфавитном (лексикографическом) порядке.

По умолчанию `simp:true`, выражения упрощаются. Никаких прочих упрощений по умолчанию Maxima не производит.

Листинг 136

```
(%i1)  kill(x,k)$
       simp:true$
       x+2*x;
       x/(x*(x+1));
       sum(1/k,k,1,10);
(%o3)  3 x
(%o4)  1/(x+1)
(%o5)  7381/2520
(%i6)  simp:false$
       x+2*x;
       x/(x*(x+1));
       sum(1/k,k,1,10);
(%o7)  x+2 x
(%o8)  x/(x (x+1))
```

```
(%o9)  sum(1/k, k, 1, 10)
```

$$\sum_{k=1}^{10} \frac{1}{k}$$

♦ Функция

ev(*expr, opt1, ..., optN*)

вычисляет выражение *expr* с опциями *opt1, ..., optN*, которые могут принимать следующие значения:

- `nouns` — вычислять выражения, имеющие неопределенную форму;
- `diff, integrate, sum` — выполнить все дифференцирования, интегрирования, суммирования и т.п.;
- `numer` — функции с численными аргументами вычислять в числах с плавающей точкой;
- `pred` — вычислить все предикаты;
- `eval` — вызывает дополнительное вычисление;
- *x*:*expr0* — переменной *x* будет присвоено значение *expr0*;
- *F*(*x*):=*expr0* — функции *F*(*x*) будет присвоено выражение *expr0*.

Опции применяются в том порядке, в котором они указаны в вызове функции.

Для увеличения быстродействия при подстановке значений, вместо

ev(*expr*, *x1*:*expr1*, *x2*:*expr2*, ...)

рекомендуется использовать

subst([*x1*=*expr1*, *x2*=*expr2*, ...], *expr*).

Вообще, следует по возможности избегать использования функции `ev`, особенно многократного (например, в цикле).

Листинг 137

```
(%i1)  kill(a)$
       s:lsum(a[n],n,[2,3,5,7]);
       a:makelist(random(10),7);
       ''s;
       ev(s,nouns);
```

```
(%o2)  a_7 + a_5 + a_3 + a_2
(%o3)  [2, 2, 4, 5, 4, 1, 9]
(%o4)  19
(%o5)  19

(%i6)  y:5;
       x:('y)^2;
       ''x;
       ev(x,nouns);

(%o6)  5
(%o7)  y^2
(%o8)  25
(%o9)  25

(%i10) kill(x,z)$
       a:x^2+4*x-3;
       /*  Можно так:  */
       ev(a,x:sqrt(z));
       /*  Но лучше так:  */
       subst(x=sqrt(z),a);

(%o11) x^2 + 4 x - 3
(%o12) z + 4 sqrt(z) - 3
(%o13) z + 4 sqrt(z) - 3

(%i14) kill(x,f)$
       b:diff(f(x),x)^2-3*integrate(f(x),x,1,x);
       subst(f(x)=x^3,b);
       ev(b,f(x):=x^3,diff);
       ev(b,f(x):=x^3,diff,integrate);
       expand(%);

(%o15) (d/dx f(x))^2 - 3 integrate(f(x), x, 1, x)

(%o16) (d/dx x^3)^2 - 3 integrate(x^3, x, 1, x)

(%o17) 9 x^4 - 3 integrate(x^3, x, 1, x)
```

(%o18) $9\,x^4 - 3\left(\frac{x^4}{4} - \frac{1}{4}\right)$

(%o19) $\frac{33\,x^4}{4} + \frac{3}{4}$

♦ Функция

`at`(*expr*, [*x1*=*expr1*, ..., *xN*=*exprN*])

— подставляет в выражение *expr* вместо переменных *x1*,...,*xN* выражения *expr1*,...,*exprN*. В отличие от функций типа `subst`, корректно работает с заменами переменных в дифференциальных выражениях.

Листинг 138

```
(%i1) kill(x,y,t,g)$
      f(x,y):=y*diff(g(x,y),x);
      F:at(f(x,y),[x=t^2,y=t]);
      subst(g(x,y)=y^2/x^2,F);
      ev(F,g(x,y):=y^2/x^2,nouns);
```

(%o2) $\mathrm{f}(x,y) := y\,\mathrm{diff}(\mathrm{g}(x,y),x)$

(%o3) $t\left(\frac{d}{d\,x}\,\mathrm{g}(x,y)\Big|_{[x=t^2,y=t]}\right)$

(%o4) $t\left(\frac{d}{d\,x}\,\frac{y^2}{x^2}\Big|_{[x=t^2,y=t]}\right)$

(%o5) $-\frac{2}{t^3}$

♦ Функции

`expand`(*expr*),
`factor`(*expr*)

служат для раскрытия скобок и разложения на множители. Первая функция раскрывает произведения сумм, применяя распределительный закон умножения. Вторая — раскладывает многочлены на множители (над полем рациональных чисел).

Листинг 139

```
(%i1)  kill(x)$
       (x-1)^3*(x+2)^4*(x-3)^5*(x^2+x-1)^6;
       expand(%);
       diff(%,x);
       factor(%);
```

(%o2) $(x-3)^5\,(x-1)^3\,(x+2)^4\,(x^2+x-1)^6$

(%o3) $x^{24}-4\,x^{23}-33\,x^{22}+126\,x^{21}+504\,x^{20}-1722\,x^{19}-4621\,x^{18}+13312\,x^{17}+27261\,x^{16}-64372\,x^{15}-103544\,x^{14}+207204\,x^{13}+245335\,x^{12}-466420\,x^{11}-339783\,x^{10}+736478\,x^9+204904\,x^8-768138\,x^7+97861\,x^6+449528\,x^5-229845\,x^4-76392\,x^3+102168\,x^2-33696\,x+3888$

(%o4) $24\,x^{23}-92\,x^{22}-726\,x^{21}+2646\,x^{20}+10080\,x^{19}-32718\,x^{18}-83178\,x^{17}+226304\,x^{16}+436176\,x^{15}-965580\,x^{14}-1449616\,x^{13}+2693652\,x^{12}+2944020\,x^{11}-5130620\,x^{10}-3397830\,x^9+6628302\,x^8+1639232\,x^7-5376966\,x^6+587166\,x^5+2247640\,x^4-919380\,x^3-229176\,x^2+204336\,x-33696$

(%o5) $2(x-3)^4(x-1)^2(x+2)^3\left(x^2+x-1\right)^5$
$\left(12x^4-10x^3-57x^2+20x+26\right)$

♦ Функция

`rat`(*expr, x1, ..., xN*)

выполняет приведение выражения *expr* к канонической рациональной форме (CRE):

- числа с плавающей точкой приводятся к рациональным числам с точностью `ratepsilon`;
- раскрываются все произведения;
- сумма дробей приводится к наименьшему общему знаменателю;
- переменные упорядочиваются в заданном порядке;
- многочлены группируются по степеням переменных.

Результат отмечается специальным символом `/R/` указывающим на то, что он не обязательно в точности равен исходному выражению.

Листинг 140

```
(%i1)  kill(x,y,z)$
       e:1+y+2*z+2*y*z+3*x*z+3*x*y*z+4*z*z+5*y*z*z;
```

```
rat(e);
rat(e,z,y,x);
```

(%o2) $5\,y\,z^2+4\,z^2+3\,x\,y\,z+2\,y\,z+3\,x\,z+2\,z+y+1$

(%o3)/R/ $(5\,y+4)\,z^2+((3\,x+2)\,y+3\,x+2)\,z+y+1$

(%o4)/R/ $(3\,z\,y+3\,z)\,x+\left(5\,z^2+2\,z+1\right)\,y+4\,z^2+2\,z+1$

```
(%i5) e:(1+(x-2*y)^4/(x^2-4*y^2)^2)*(z+y)*(x+2*y)/(x^2+4*y^2);
rat(e);
rat(e,z,y,x);
```

(%o5) $$\frac{(2\,y+x)\,\left(\frac{(x-2\,y)^4}{(x^2-4\,y^2)^2}+1\right)\,(z+y)}{4\,y^2+x^2}$$

(%o6)/R/ $$\frac{2\,z+2\,y}{2\,y+x}$$

(%o7)/R/ $$\frac{2\,y+2\,z}{x+2\,y}$$

```
(%i8) kill(x)$
(1-3/(x-1))^4*(1+2/(x+1))^5*(1+4/(x+2))^6;
rat(%);
diff(%,x);
factor(%);
```

(%o9) $$\left(1-\frac{3}{x-1}\right)^4\left(\frac{2}{x+1}+1\right)^5\left(\frac{4}{x+2}+1\right)^6$$

(%o10)/R/ $(x^{15}+35\,x^{14}+450\,x^{13}+1850\,x^{12}-12395\,x^{11}-156945\,x^{10}-366840\,x^9+2485620\,x^8+16050960\,x^7+10866960\,x^6-163669248\,x^5-500385600\,x^4+33592320\,x^3+2620200960\,x^2+4837294080\,x+2902376448)/(x^{15}+13\,x^{14}+68\,x^{13}+168\,x^{12}+118\,x^{11}-370\,x^{10}-916\,x^9-396\,x^8+1089\,x^7+1469\,x^6+8\,x^5-1124\,x^4-624\,x^3+176\,x^2+256\,x+64)$

(%o11)/R/ $-(22\,x^{16}+522\,x^{15}+1740\,x^{14}-59300\,x^{13}-703650\,x^{12}-1722990\,x^{11}+17179600\,x^{10}+127387560\,x^9+139352400\,x^8-1558638720\,x^7-6446946816\,x^6-3478951296\,x^5+34796977920\,x^4+97613683200\,x^3+100575406080\,x^2+25960144896\,x-13544423424)/(x^{18}+15\,x^{17}+93\,x^{16}+289\,x^{15}+360\,x^{14}-438\,x^{13}-2110\,x^{12}-2094\,x^{11}+1953\,x^{10}+5875\,x^9+2649\,x^8-4755\,x^7-5818\,x^6+36\,x^5+3480\,x^4+1648\,x^3-480\,x^2-576\,x-128)$

(%o12) $$-\frac{2\,(x-4)^3\,(x+3)^4\,(x+6)^5\,\left(11\,x^4-69\,x^3-558\,x^2-560\,x+168\right)}{(x-1)^5\,(x+1)^6\,(x+2)^7}$$

♦ Функции

`ratsimp`(*expr, x1, . . . , xN*),
`fullratsimp`(*expr*)

упрощают выражение *expr* и все его подвыражения, включая аргументы нерациональных функций. Вторая функция применяет первую в цикле пока выражение меняется.

Листинг 141

```
(%i1) kill(x)$
      ((x-1)^(3/2)-(1+x)*sqrt(x-1))/sqrt(x-1)/sqrt(1+x);
      ratsimp(%);
```

(%o2) $$\frac{(x-1)^{\frac{3}{2}}-\sqrt{x-1}\,(x+1)}{\sqrt{x-1}\,\sqrt{x+1}}$$

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

```
(%i4) kill(x)$
      (x-1)^(1/3)*(x+3)^(2/5)*(x-2)^(-6/7);
      diff(%,x);
      ratsimp(%);
      factor(%);
```

(%o5) $$\frac{(x-1)^{\frac{1}{3}}\,(x+3)^{\frac{2}{5}}}{(x-2)^{\frac{6}{7}}}$$

(%o6) $$-\frac{6\,(x-1)^{\frac{1}{3}}\,(x+3)^{\frac{2}{5}}}{7\,(x-2)^{\frac{13}{7}}}+\frac{(x+3)^{\frac{2}{5}}}{3\,(x-2)^{\frac{6}{7}}\,(x-1)^{\frac{2}{3}}}+\frac{2\,(x-1)^{\frac{1}{3}}}{5\,(x-2)^{\frac{6}{7}}\,(x+3)^{\frac{3}{5}}}$$

(%o7) $$-\frac{13\,x^2+271\,x-144}{(x-2)^{\frac{6}{7}}\,(x-1)^{\frac{2}{3}}\,(x+3)^{\frac{3}{5}}\,(105\,x-210)}$$

(%o8) $$-\frac{13\,x^2+271\,x-144}{105\,(x-2)^{\frac{13}{7}}\,(x-1)^{\frac{2}{3}}\,(x+3)^{\frac{3}{5}}}$$

```
(%i9) kill(x,a)$
      e:(x^(a/2) + 1)^2*(x^(a/2) - 1)^2/(x^a - 1);
```

```
/* Можно несколько раз применить ratsimp */
ratsimp(e);
ratsimp(%);

/* Или один раз fullratsimp */
fullratsimp(e);
```

(%o10) $$\frac{\left(x^{\frac{a}{2}}-1\right)^2\left(x^{\frac{a}{2}}+1\right)^2}{x^a-1}$$

(%o11) $$\frac{x^{2\,a}-2\,x^a+1}{x^a-1}$$

(%o12) x^a-1

(%o13) x^a-1

4.4. Файловый ввод-вывод

4.4.1. Текстовый ввод-вывод

Для записи результатов вычислений, оформленных в виде списка, массива или матрицы, можно воспользоваться следующей функцией.

♦ Функция

`write_data`(*obj*, *fn*, *sep*)

пишет объект *obj* по адресу *fn* с разделителем, отделяющим один фрагмент данных от другого:

sep =

`comma` (`,`),
`pipe` (`|`),
`semicolon` (`:`),
`space` (пробел),
`tab` (символ табуляции).

Объект *obj* может быть списком, массивом или матрицей.

Листинг 142

```
(%i1) A:[2-3*%i, f(1/2,sin(1)), "и еще строка"]$
      /*  Папка D:/Compmath должна существовать  */
      write_data(A,"D:/Compmath/temp_list",pipe);
```

```
(%o2)  done

(%i3)  kill(x,y,g)$
       A:matrix([2,-3/2,g(x,y)],[0.5,x^2,sqrt(2)]);
       /*  Папка D:/Compmath должна существовать  */
       write_data(A,"D:/Compmath/temp_matrix",comma);
```

$$(\%o4)\quad \begin{bmatrix} 2 & -\frac{3}{2} & g(x,y) \\ 0.5 & x^2 & \sqrt{2} \end{bmatrix}$$

```
(%o5)  done
```

- Функция

  `file_search`(*fn*)

ищет файл *fn* и возвращает его полный путь *fn* в случае успеха, либо `false` в случае неудачи.

- Функция

  `printfile`(*fn*)

выводит на экран содержимое текстового файла. Возвращает полный путь к файлу *fn*, либо пишет сообщение об ошибке.

Листинг 143

```
(%i1)  fn:"D:/Compmath/temp_list"$
       if (file_search(fn)=fn) then printfile(fn)$

       fn:"D:/Compmath/temp_matrix"$
       if (file_search(fn)=fn) then printfile(fn)$
2-3*%i|f(1/2,sin(1))|"и еще строка"
2,-3/2,g(x,y)
0.5,x^2,sqrt(2)
```

Для записи в текстовый файл произвольных данных можно воспользоваться следующими функциями.

- Функции

  `openw`(*fn*),
  `opena`(*fn*)

возвращают поток для записи текстового файла *fn*. Если файл существует, то в первом случае он будет перезаписан, а во втором — запись будет

добавлена в конец файла.

♦ Функция

`close`(*stream*)

закрывает открытый поток записи (или поток чтения) текстового файла.

♦ Функция

`printf`(*stream, control, expr1, ..., exprN*)

пишет в поток *stream* выражения *expr1, ..., exprN* в соответствии с управляющей строкой[16] *control*:

`~%` перенос строки;

`~t` символ табуляции;

`~d` десятичное целое;

`~@r` натуральное число в римской системе счисления (от 1 до 3999);

`~f` число с плавающей точкой;

`~e` число с плавающей точкой в экспоненциальной форме записи;

`~g` есть `~f` или `~e` в зависимости от величины числа;

`~h` длинное число с плавающей точкой;

`~a` использовать функцию `string`;

`~s` то же, что `~a`, но `stringdisp`=`true`.

Функция возвращает `false`. Для вывода результата не в файл, а на экран надо передать *stream*=`false`. В этом случае функция возвратит строку.

Листинг 144

```
(%i1) makelist(printf(false,"~@r",k!),k,1,7);
(%o1) [I, II, VI, XXIV, CXX, DCCXX, 5040]
(%i2) /*  Папка D:/Compmath должна существовать  */
      f:openw("D:/Compmath/какие-то данные.txt")$
      printf(f,"~a~%",
         "Вот некоторое числовое выражение и его значение:")$
```

[16] Функция `printf` вызывает Lisp-функцию `format`, поэтому полную (и весьма общирную) спецификацию управляющей строки *control* можно посмотреть в справочниках по Common Lisp: http://www.lispworks.com/documentation/HyperSpec/Body/22_c.htm

```
        x:%e^(-20)*sin(10*%pi/9);
        fpprec:30$
        printf(f,"~a~a~h~%",x,"=",bfloat(x))$
        printf(f,"~a~%","Оно же в форматах f, e:")$
        printf(f,"~f~a~e",float(x),"  и  ",float(x))$
        close(f)$
        printfile("D:/Compmath/какие-то данные.txt")$
```

(%o4) $\%e^{-20}\sin\left(\frac{10\,\pi}{9}\right)$

```
Вот некоторое числовое выражение и его значение:
%e^-20*sin(10*%pi/9)=-0.000000000704956057362656845863501906682
Оно же в форматах f, e:
-0.00000000070495605736265667 и -7.0495605736265667E-10
```

♦ Для той же цели можно использовать функцию

`with_stdout`(*fn*, *expr1*, . . . , *exprN*),

направляющую поток печати в файл *fn*. Поток печати создается функцией `print`.

Напечатаем в текстовый файл решения уравнения $x^2 + y^2 = z^2$ в натуральных числах. Каждое такое решение можно интерпретировать как прямоугольный треугольник с целочисленными сторонами или как целую точку на конусе.

Листинг 145

```
(%i1)  /*  Устанавливаем текстовый формат вывода  */
       set_display(none)$
       /*  Папка D:/Compmath должна существовать  */
       with_stdout("D:/Compmath/output.txt",
           for k:2 thru 10000 do
               if evenp(k) then print([k^2-1,2*k,k^2+1])
               else print([(k^2-1)/2,k,(k^2+1)/2])
       )$
```

Для чтения из файла числовой матрицы или числового списка предназначены следующие функции.

♦ Функции

`read_matrix`(*fn, sep*),
`read_list`(*fn, sep*)

читают числовую матрицу или список из текстового файла *fn*. К сожалению, `read_list`, `read_matrix` не читают список и матрицу произвольного вида. Проблема возникает даже с числовыми дробями.

Листинг 146

```
(%i1) /*  Папка D:/Compmath должна существовать  */
      write_data(
              [2-3*%i, f(1/2,sin(1)), "и еще строка"],
              "D:/Compmath/temp_list", pipe
      );
      write_data(
              matrix([2/5,3/7,-1/2],[0.5,-1.247,12.57]),
              "D:/Compmath/temp_matrix", comma
      );

      read_list("D:/Compmath/temp_list",pipe);
      read_matrix("D:/Compmath/temp_matrix", comma);
(%o1) done
(%o2) done
(%o3) [2, −3, *, %i, f, (, 1, /, 2, , , sin, (, 1, ), ), и еще строка]
(%o4)
```

$$\begin{bmatrix} [2,/,5] & [3,/,7] & [-1,/,2] \\ 0.5 & -1.247 & 12.57 \end{bmatrix}$$

♦ Функция

`openr`(*fn*)

возвращает поток для чтения текстового файла *fn*. После окончания работы с потоком его следует закрыть с помощью функции `close`.

♦ Функция

`readline`(*stream*)

читает очередную строку из потока *stream*.

Используя эти функции несложно написать пользовательскую функцию, читающую произвольную матрицу.

Листинг 147

```
(%i1) my_read_matrix(fn):=block([f,A,s],
        if (file_search(fn)=fn) then (
          f:openr(fn),
          A:matrix(), /*  пустая матрица  */
          /*  Читаем строку из файла и добавляем в A  */
          while (s:readline(f))#false do
            A:addrow(A,eval_string(concat("[",s,"]"))),
          close(f)
        ),
        A
      )$

      /*  Запишем в файл матрицу со сложной структурой  */
      /*  Папка D:/Compmath должна существовать  */
      write_data(
        matrix(
          [matrix(
            [sqrt(23),%e^(-2*x)],
            [sin(3*x),x/(x+3)]
          ), g(x,y)],
          [-1.247, -13/59]
        ), "D:/Compmath/temp_matrix", comma
      )$

      /*  Читаем файл  */
      my_read_matrix("D:/Compmath/temp_matrix");
```

(%o4)
$$\begin{bmatrix} \begin{bmatrix} \sqrt{23} & e^{-2\,x} \\ \sin(3\,x) & \frac{x}{x+3} \end{bmatrix} & \mathrm{g}(x,y) \\ -1.247 & -\frac{13}{59} \end{bmatrix}$$

4.4.2. Бинарный ввод-вывод

♦ Функция

`write_binary_data`(*obj*, *fn*)

записывает объект *obj* в бинарный файл *fn*. Объект *obj* может быть чис-

ловым списком, числовым массивом или числовой матрицей. Числовые дроби будут переведены в числа с плавающей точкой.

- Функция

  `read_binary_list`(*fn*)

читает список из файла.

- Функция

  `read_binary_matrix`(*fn*, *M*)

читает матрицу из файла в переменную *M*. Переменная *M* должна быть заданной матрицей (например, нулевой `zeromatrix`) определенного размера.

Листинг 148

```
(%i1) /*  Папка D:/Compmath должна существовать  */
      L:[2, -3/2, 4/7];
      write_binary_data(L,"D:/Compmath/temp_list");
      A:matrix([2, -3/2], [0.5e-3, -0.00435]);
      write_binary_data(A,"D:/Compmath/temp_matrix");
```

(%o1) $[2, -\frac{3}{2}, \frac{4}{7}]$

(%o2) done

(%o3) $\begin{bmatrix} 2 & -\frac{3}{2} \\ 5.0000000000000001\,10^{-4} & -0.00435 \end{bmatrix}$

(%o4) done

```
(%i5) read_binary_list("D:/Compmath/temp_list");

      /*  Размер матрицы должен быть известен  */
      M1:zeromatrix(2,2)$
      read_binary_matrix("D:/Compmath/temp_matrix",M1);
      M2:zeromatrix(2,3)$
      read_binary_matrix("D:/Compmath/temp_matrix",M2);
```

(%o5) $[2.0, -1.5, 0.57142857142857]$

(%o7) $\begin{bmatrix} 2.0 & -1.5 \\ 5.0000000000000001\,10^{-4} & -0.00435 \end{bmatrix}$

(%o9)
$$\begin{bmatrix} 2.0 & -1.5 & 5.000000000000000\,1\,10^{-4} \\ -0.00435 & false & false \end{bmatrix}$$

4.5. Maxima и Lisp

Здесь мы рассмотрим пример использования в Maxima кода, написанного на Lisp. Желающим изучить этот язык программирования рекомендуем начать со статьи в Википедии. В версии Maxima под Windows используется диалект, называемый GNU Common Lisp (GCL). Под Linux можно собрать Maxima и с другими диалектами Lisp, но по скорости работы GCL имеет существенное преимущество.

Если некоторую последовательность операций необходимо применить многократно (например, в цикле), то вычисления можно ускорить, написав функцию на Lisp.

Создадим в директории `D:/Compmath` текстовый файл `func.lisp` следующего содержания:

Листинг 149

```
(defun $f (x &optional (a '$%e))
  (add x (power a (mul -1 x))))
```

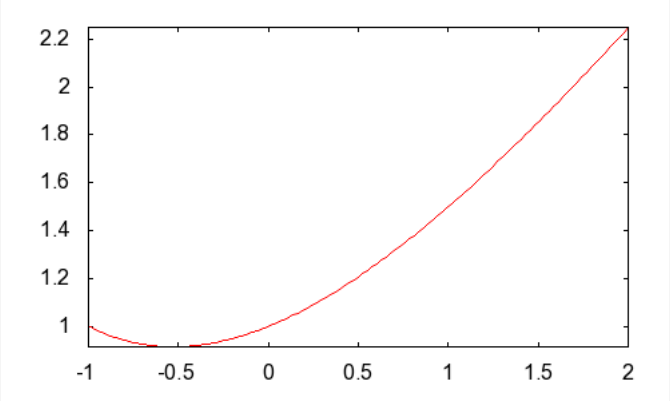
Функция `f` вычисляет значение выражения $x+a^{-x}$, где по умолчанию a есть эйлерово число $e = 2.71828\ldots$. Значок $ указывает Lisp'у на то, что это обозначение из Maxima.

Листинг 150

```
(%i1)  /*  Загружаем нашу lisp-функцию  */
       load("D:/Compmath/func.lisp")$

       /*  Проверяем ее работу на различных примерах  */
       [f(-2), float(f(-1)+1), f(x), f(x^2,3), f(x,1/y)];

       /*  Строим график функции  */
       wxdraw2d(
         color=red,
         explicit(f(x,2), x,-1,2)
       )$
```

(%o2) $[\%e^2 - 2, 2.718281828459045, \%e^{-x} + x, \frac{1}{3^{x^2}} + x^2, y^x + x]$

(%t3)



Аналогично можно действовать с помощью специальной команды[17] `:lisp`. Обратите внимание на прямые скобки и на то, что определяемая функция будет записана в память заглавными буквами.

Листинг 151

```
(%i1)  /*  Создаем lisp-функцию  */
       :lisp (defun |$f| (x &optional (a '$%e))
         (add x (power a (mul -1 x))))
```

$f

```
(%i1)  /*  Проверяем работу функции  */
       F(x);
```

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

4.6. Рисование в Maxima

Для использования этих функций надо загрузить пакет `draw` командой `load("draw")$`. В wxMaxima этот пакет загружен по умолчанию.

Функции `draw2d, draw3d`, строящие графические объекты, вызывают для этого соответствующие команды графического пакета Gnuplot. Ре-

[17] Команды, начинающиеся с двоеточия ":", не распознаются системой Maxima как выражения. Эти команды служат для трансляции кода из Maxima в Lisp и обратно, а также для управления Lisp-интерпретатором.

зультат выводится в окне Gnuplot. При этом выполнение скрипта Maxima прерывается и продолжается только после закрытия окна Gnuplot. Это поведение может оказаться неудобным.

В wxMaxima можно пользоваться wx-аналогами этих функций — функциями `wxdraw2d, wxdraw3d`. Единственное их отличие от оригинальных функций в том, что они выводят результат прямо в окно Maxima. Щелкнув правой кнопкой мыши на рамке с картинкой, можно сохранить ее в формате PNG.

Также wxMaxima предоставляет возможность создания анимации.

♦ Функция

`set_draw_defaults`(*opt1, opt2, ...*)

устанавливает пользовательские опции по умолчанию. Вызов этой функции без аргументов сбрасывает пользовательские установки. Обратите внимание: значения опций указываются с помощью оператора `=`.

Листинг 152

```
(%i1) set_draw_defaults(
        dimensions=[500,500],
        grid=true,
        proportional_axes=xy,
        color=red
      )$
```

4.6.1. Двумерные объекты

♦ Функция

`draw2d`(*opt1, opt2, ..., obj*)

строит двумерный графический объект *obj* с опциями *opt1, opt2, ...*.

В wxMaxima удобнее пользоваться wx-аналогом этой функции `wxdraw2d`.

Список опций, управляющих построением графиков, довольно внушителен. Рассмотрим несколько наиболее употребительных:

- `dimensions` — размеры окна с графиком;
- `grid` — показывать сетку координат: `true, false`;
- `proportional_axes=xy` — одинаковый масштаб по осям;

- `color` — цвет линии графика: `white`, `black` и т.д. или в системе RGB в формате `"#rrggbb"`[18].

Кроме общих опций, некоторые из которых приведены выше, для каждого графического объекта имеются свои специфические опции.

♦ Объект

`explicit`(*f, x, x0, x1*)

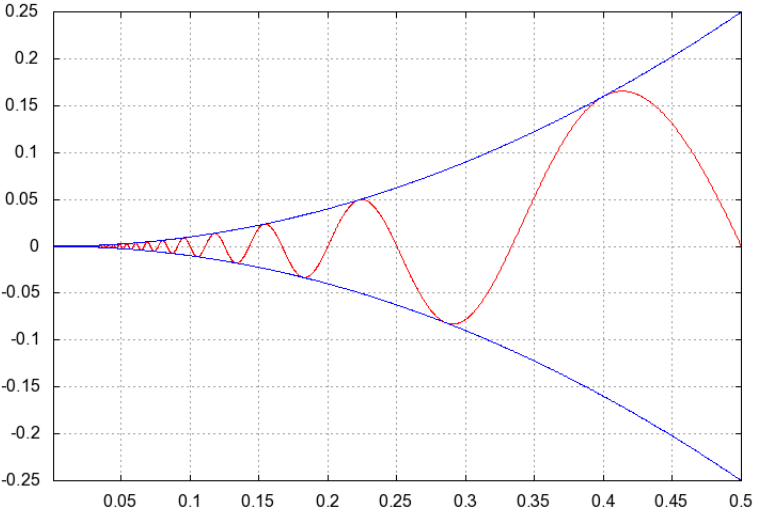
— график функции $y = f(x)$, где f — функция или выражение. Параметр `nticks` представляет число звеньев ломанной. Если кривая выглядит угловато, то надо увеличить значение этого параметра (по умолчанию 29).

Листинг 153

```
(%i1)  kill(x)$
       wxdraw2d(
         dimensions=[750,500],
         color="#ff0000",
         grid=true,
         nticks=200,
         explicit(x^2*sin(%pi/x), x,0.001,0.5),
         color="#0000ff",
         explicit(x^2, x,0.001,0.5),
         explicit(-x^2, x,0.001,0.5)
       )$
```

[18] В системе RGB цвет задается тремя 2-значными шестнадцатиричными словами. Первое слово отвечает за красную компоненту, второе — за зеленую, третье — за синюю. Чем больше числовое значение слова, тем светлее оттенок, в частности: `#000000` — черный, `#ffffff` — белый. Смесь компонентов в равной пропорции соответствует тону серого.

(%t2)



♦ Объект

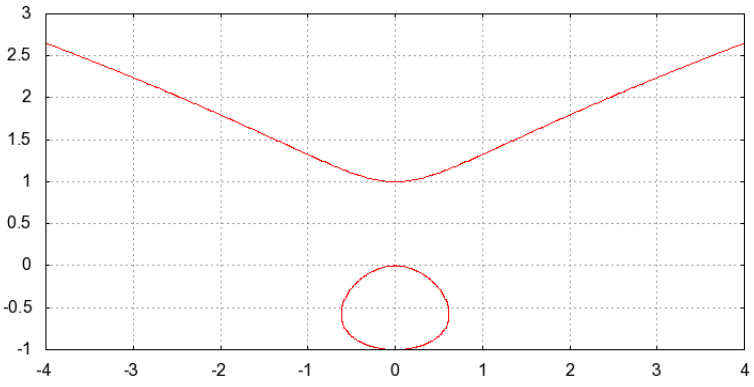
`implicit`(*eq, x, x0, x1, y, y0, y1*)

— кривая, неявно заданная уравнением *eq*. Будет построена часть кривой, лежащая в прямоугольнике $[x0, x1] \times [y0, y1]$. Если кривая выглядит угловато, то следует увеличить значения параметров `ip_grid`, управляющих частотой дискретизации (по умолчанию `[50, 50]`).

Листинг 154

```
(%i1)  kill(x,y)$
       wxdraw2d(
         dimensions=[750,375],
         color=red,
         grid=true,
         ip_grid=[100,100],
         implicit(x^2=y*(y^2-1), x,-4,4, y,-1,3)
       )$
```

(%o2)



♦ Объект

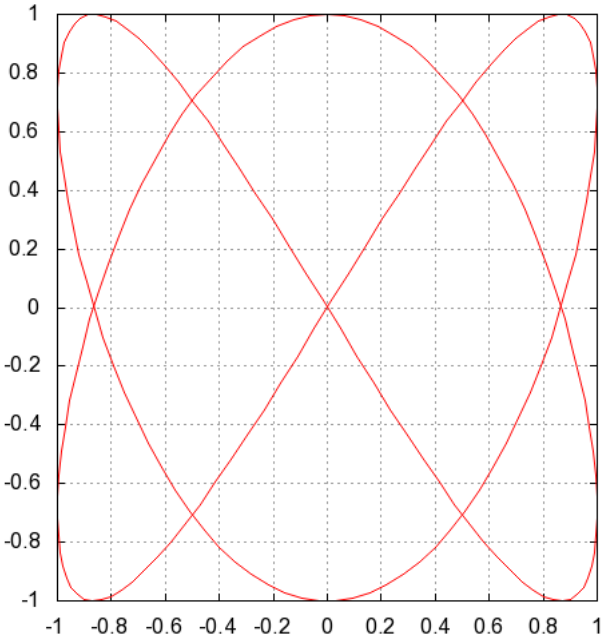
`parametric`(*x, y, t, t0, t1*)

— параметрически заданная кривая $(x(t), y(t))$. Частота дискретизации задается параметром `nticks` (по умолчанию 29).

Листинг 155

```
(%i1)  kill(t)$
       wxdraw2d(
         dimensions=[500,500],
         color=red,
         grid=true,
         nticks=200,
         parametric(sin(2*t),cos(3*t), t,0,2*%pi)
       )$
```

(%o2)



♦ Объект

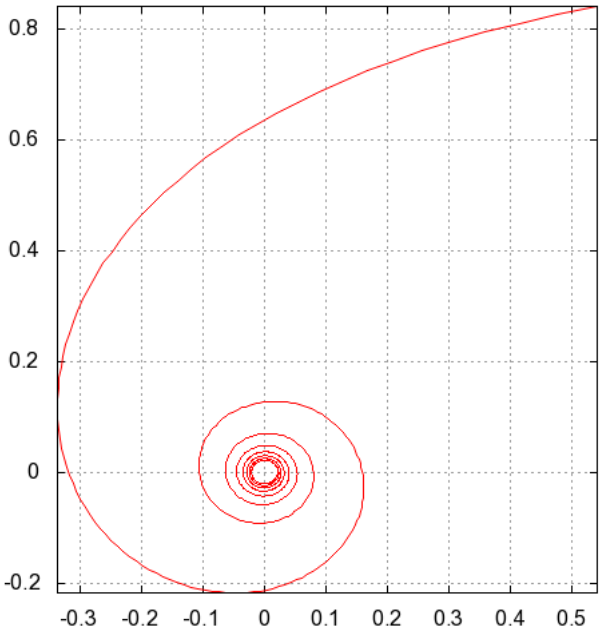
`polar`(*r, t, t0, t1-t0*)

— кривая в полярной системе координат $(r(t), t)$. Частота дискретизации задается параметром `nticks` (по умолчанию 29).

Листинг 156

```
(%i1)  kill(t)$
       wxdraw2d(
         dimensions=[500,500],
         color=red,
         grid=true,
         nticks=1000,
         polar(1/t, t,1,50)
       )$
```

(%o2)



♦ Объект

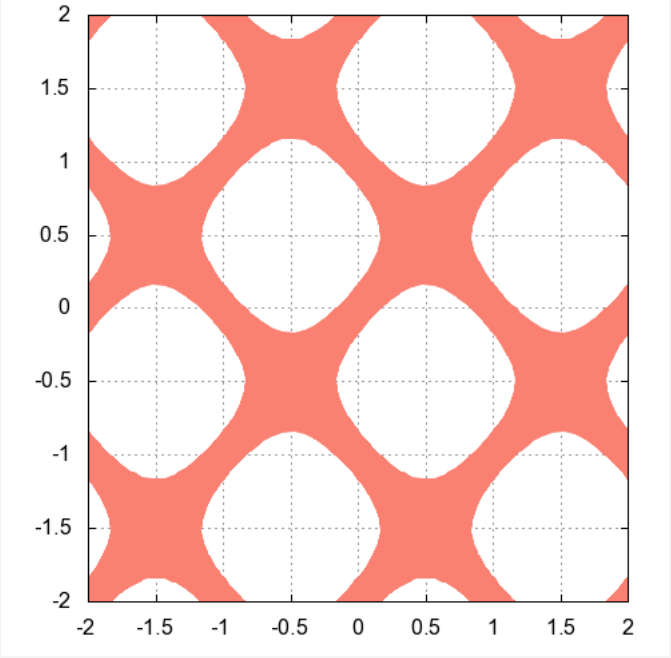
`region`(*p, x, x0, x1, y, y0, y1*)

— область в декартовой системе координат, определяемая условием $p(x, y) =$ `true`. Частоты дискретизации задаются параметрами `x_voxel, y_voxel` (по умолчанию 10). Параметр `fill_color` определяет цвет заливки.

Листинг 157

```
(%i1)  kill(x,y)$
       wxdraw2d(
         dimensions=[500,500],
         x_voxel=50,
         y_voxel=50,
         fill_color=salmon,
         grid=true,
         region(abs(sin(%pi*x)-sin(%pi*y))<1/2, x,-2,2, y,-2,2)
       )$
```

(%o2)



Функции `draw2d` можно передать единовременно довольно много графических объектов. В следующем примере строится $3^7 = 2187$ треугольников.

♦ Объект

`triangle([`*x1,y1*`], [`*x2,y2*`], [`*x3,y3*`])`

— треугольник. Параметр `fill_color` определяет цвет внутренней заливки, `color` — цвет границы.

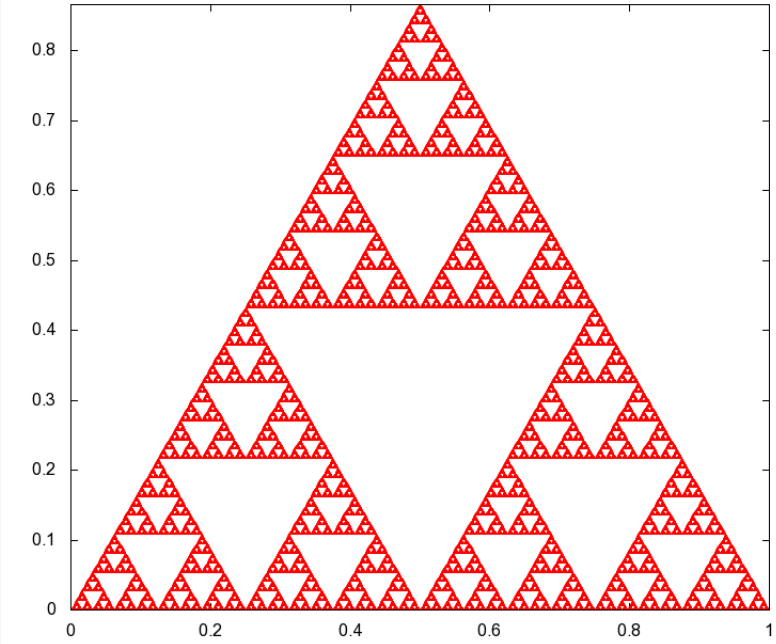
Треугольник Серпинского получается предельным переходом в следующем процессе. Изначально имеется некоторый треугольник. На очередном шаге каждый из имеющихся треугольников делится средними линиями на 4 подобных треугольника и центральный треугольник выкидывается.

Листинг 158

```
(%i1)  /*  Заранее установим необходиые опции  */
       set_draw_defaults(
         proportional_axes=xy,
         dimensions=[750,680],
```

```
        fill_color="#ff0000",
        color="#ff0000"
      )$

(%i2) kill(T)$

      /*  Один шаг преобразования списка треугольников T  */
      f(T):=block([r:[]],
        for t in T do r:append(r,[
           [t[1],(t[1]+t[2])/2,(t[1]+t[3])/2],
           [t[2],(t[2]+t[1])/2,(t[2]+t[3])/2],
           [t[3],(t[3]+t[1])/2,(t[3]+t[2])/2]
        ]),
        r
      )$

      /*  Начальный список  */
      T:[[[0,0],[1,0],[1/2,sqrt(3)/2]]]$
      /*  7 шагов преобразования  */
      for i:1 thru 7 do T:f(T)$
      /*  Действуем на каждый треугольник функцией triangle  */
      map(lambda([t],apply(triangle,t)),T)$

      apply(wxdraw2d,%)$
```

(%o7)



4.6.2. Трехмерные объекты

♦ Функция

`draw3d`(*opt1, opt2, . . . , obj*)

строит трехмерный графический объект *obj* с опциями *opt1, opt2, . . .*.

Пользоваться wx-аналогом этой функции `wxdraw3d` неудобно, поскольку при этом получается статическая картинка, которую невозможно покрутить.

Поверхности по умолчанию (когда `enhanced3d=false`) изображаются *координатной сеткой*. Сетка раскрашивается цветом `color`.

Рассмотрим некоторые опции:

- `dimensions` — размеры окна с графиком;
- `proportional_axes=xyz` — одинаковый масштаб по осям;
- `color` — цвет линии графика;

- `enhanced3d` — параметр, управляющий раскраской поверхностей;
- `wired_surface` — показать или скрыть координатную сетку при включенной раскраске поверхностей `enhanced3d=true`;
- `view` — пара углов *сферической системы координат* $[\varphi, \theta]$ (в градусах), задающая ориентацию координатных осей (x, y, z) относительно экрана;
- `surface_hide` — скрыть невидимую часть поверхности: `true, false`.

Кроме общих опций, для каждого графического объекта имеются свои специфические опции.

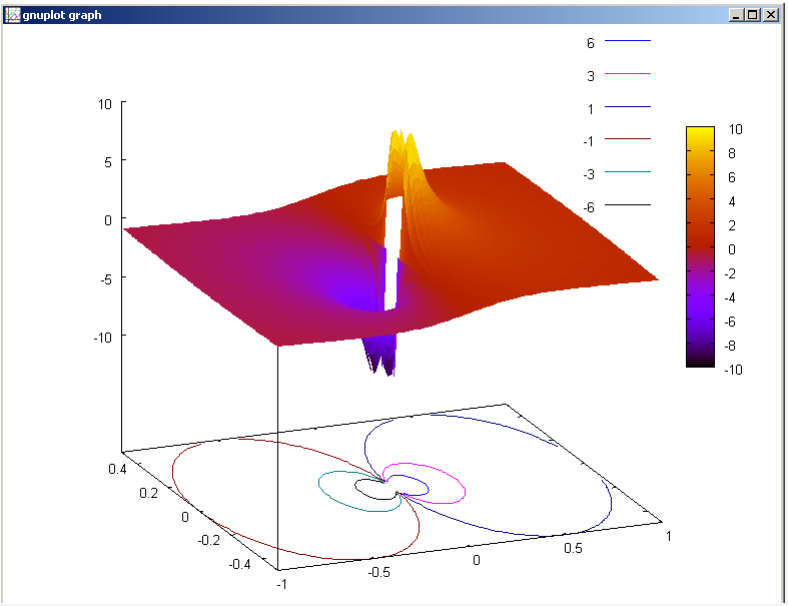
♦ Объект

`explicit`(*f, x, x0, x1, y, y0, y1*)

— поверхность $z = f(x, y)$, где f — функция или выражение. Частотой дискретизации управляют параметры `xu_grid, yv_grid`. Параметры `contour, contour_levels` позволяют изобразить линии уровня $f(x, y) = C$. Первый из них `contour` задает расположение линий уровня (по умолчанию `none` — линии уровня нарисованы не будут), а второй `contour_levels` — множество значений C.

Листинг 159

```
(%i1)  kill(x,y)$
       draw3d(
         enhanced3d=true,
         contour=base,
         contour_levels={-6,-3,-1,1,3,6},
         surface_hide=true,
         xu_grid=100,
         yv_grid=100,
         zrange=[-10,10],
         explicit(x/(x^2+y^2), x,-1,1, y,-0.5,0.5)
       )$
```



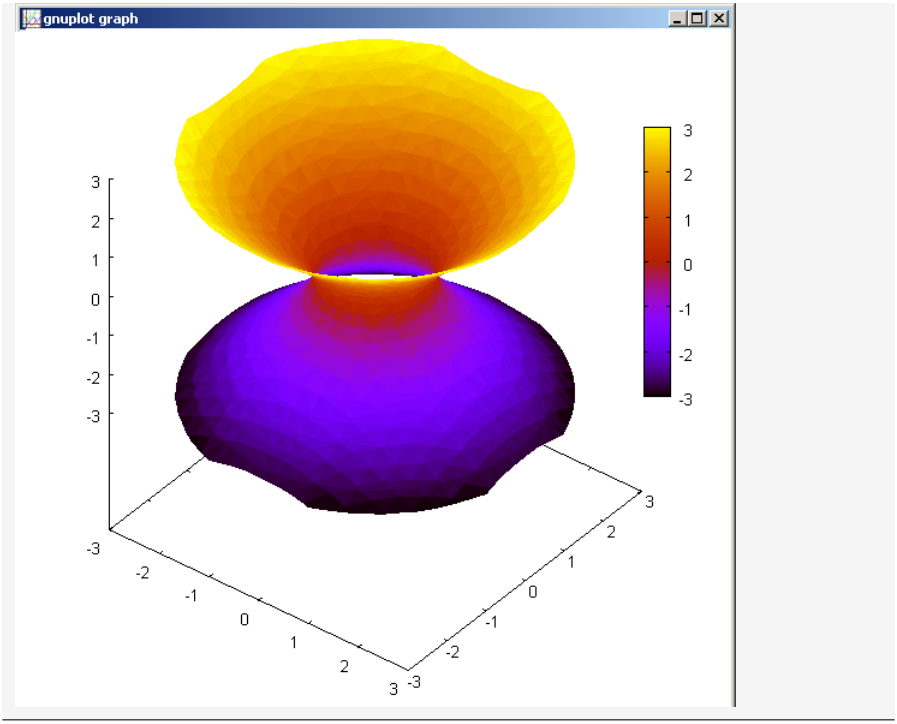
♦ Объект

`implicit`(*eq, x, x0, x1, y, y0, y1, z, z0, z1*)

— поверхность, неявно заданная уравнением *eq*. Будет построена ее часть, лежащая в прямоугольном параллелепипеде $[x0, x1] \times [y0, y1] \times [z0, z1]$. Частотой дискретизации управляют параметры `x_voxel, y_voxel, z_voxel` (по умолчанию 10).

Листинг 160

```
(%i1)  kill(x,y,z)$
       draw3d(
         enhanced3d=[z,x,y,z],
         wired_surface=true,
         surface_hide=true,
         x_voxel=20,
         y_voxel=20,
         z_voxel=20,
         implicit(x^2+y^2-z^2=1, x,-3,3, y,-3,3, z,-3,3)
       )$
```



4.6.3. Анимация

Это специфическая возможность графической оболочки wxMaxima. Программа строит серию картинок, а потом показывает их в режиме слайд-шоу. Процесс показа управляется из меню wxMaxima:

Чтобы активизировать эти элементы управления надо щелкнуть мышью на рамке с анимацией. Можно также управлять из контекстного меню (правая кнопка мыши на рамке с анимацией).

Анимацию можно сохранить в анимированный GIF (Graphics Interchange Format), выбрав соответствующий пункт контекстного меню. Однако чтобы эта функция работала, необходимо предварительно установить программу Image Magick (http://www.imagemagick.org/).

Анимация в примерах ниже сохранена из wxMaxima и встроена в этот pdf-файл. Под рамкой с анимацией имеется меню управления, созданное

при встраивании. К сожалению, далеко не всякая программа для просмотра pdf воспроизводит анимацию. Лучше всего это делает Adobe Acrobat Reader, несколько хуже PDF XChange Viewer, а Foxit и Sumatra — никак.

♦ Функция

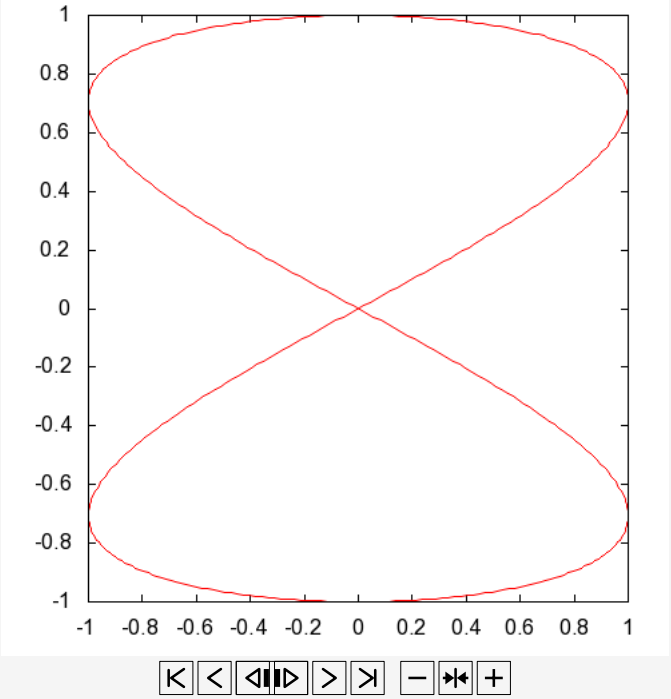
`wxanimate_draw`(*k, list, opt1, opt2, . . . , obj*)

— строит последовательность 2D-графических объектов *obj*, зависящих от параметра *k*, пробегающего список *list*. Функции можно передать несколько графических объектов, но их количество не может зависеть от параметра *k*. Опции — те же, что и для `draw2d`.

Листинг 161

```
(%i1)  kill(k,t)$

       wxanimate_draw(
         k, [1,3,5,7,9],
         dimensions=[500,500],
         color=red,
         nticks=500,
         parametric(sin(2*t),cos(k*t), t,0,2*%pi)
       )$
```

(%o2)



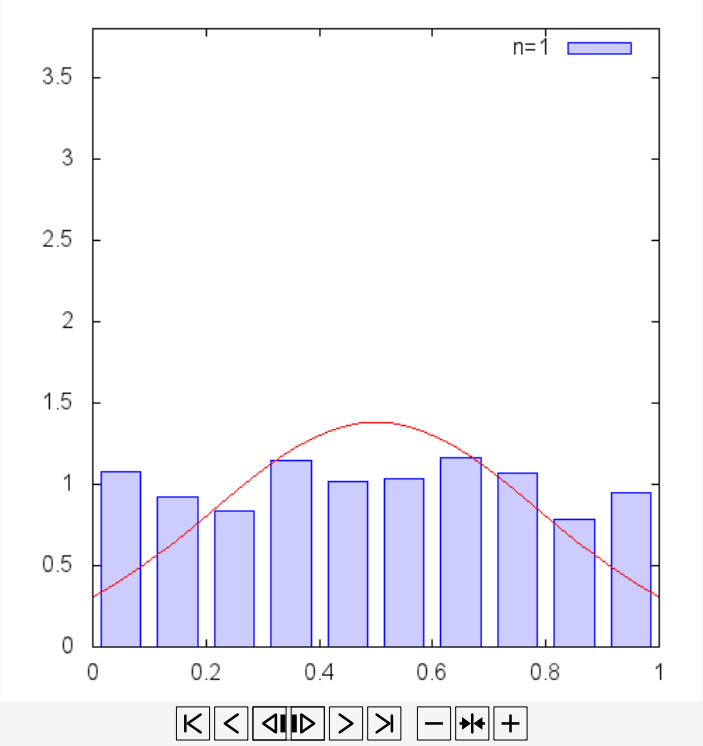
Следующий пример иллюстрирует центральную предельную теорему. Пусть X — среднее арифметическое n равномерных распределений. По этой теореме, распределение случайной величины X при больших n будет близко к нормальному. Возьмем выборку из $N = 1000$ значений X и построим гистограмму. В тех же осях построим соответствующее предельное нормальное распределение.

Листинг 162

```
(%i1)  kill(all)$

       b(n):=block([N:1000,r:makelist(0,10)],
         makelist(sum(random(1.0),k,1,n)/n,N),
         map(lambda([t],
           k:floor(10*t)+1,
           r[k]:r[k]+1
         ),%%),
         makelist([0.1*k-0.05,r[k]/(0.1*N),0.07],k,1,10)
```

```
)$

wxanimate_draw(
  n, [1,2,3,4,5,6,7],
  dimensions=[500,500],
  yrange=[0,3.8],
  fill_color=blue,
  fill_density=0.2,
  key=concat("n=",n),
  apply(bars,b(n)),
  key=false,
  color=red,
  explicit(exp(-6*n*(x-0.5)^2)/sqrt(%pi/(6*n)), x,0,1)
)$
```

(%o3)



В качестве более сложного примера рассмотрим первые 5 шагов построения варианта так называемой кривой Коха. На очередном шаге по-

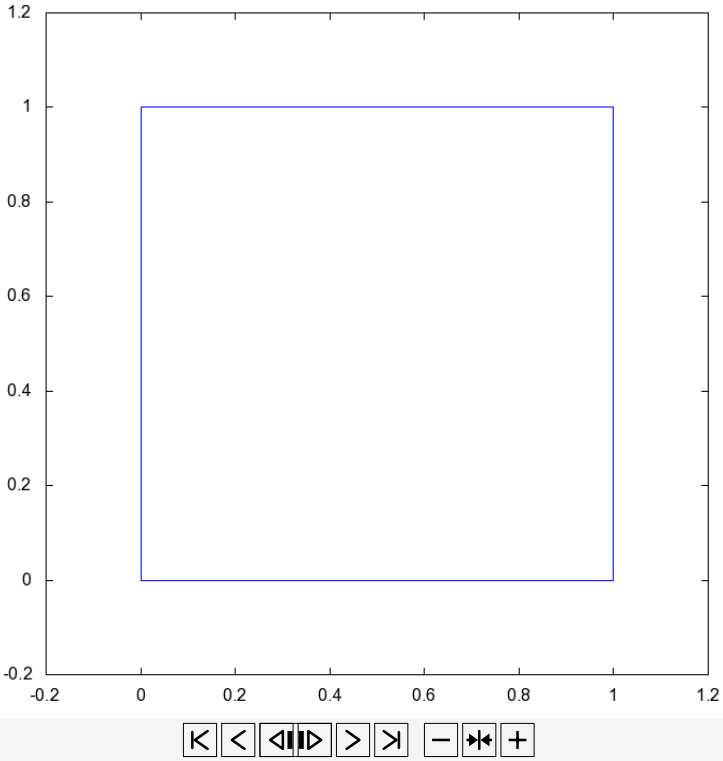
строения каждый отрезок ломанной делится на 3 части и средняя часть заменяется на боковые стороны равностороннего треугольника, для которого она является основанием.

Листинг 163

```
(%i1) kill(x,k,P)$

      /*  Функция возвращает вектор, ортогональный данному  */
      o(x):=[-x[2],x[1]]$

      /*  Один шаг построения ломанной, P - список вершин  */
      f(P):=block([v,r:[P[1]]],
        for i:1 thru length(P)-1 do (
          v:P[i+1]-P[i],
          r:append(r,[
            P[i]+1/3*v,
            P[i]+1/2*v+sqrt(3)/6*o(v),
            P[i]+2/3*v,
            P[i+1]
          ])
        ),
        r
      )$

      /*  k шагов пострения ломанной  */
      F(P,k):=block([r:P],
        for i:1 thru k do r:f(r),
        r
      )$

      /*  Начальная ломанная - квадрат  */
      P:[[0,0],[1,0],[1,1],[0,1],[0,0]]$

      wxanimate_draw(
        k,[0,1,2,3,4,5],
        transparent=true,
        proportional_axes=xy,
        dimensions=[750,680],
```

```
    xrange=[-0.2,1.2],
    yrange=[-0.2,1.2],
    polygon(F(P,k))
)$
```

(%o6)



♦ Функция

`with_slider_draw3d`(*k, list, opt1, opt2, . . . , obj*)

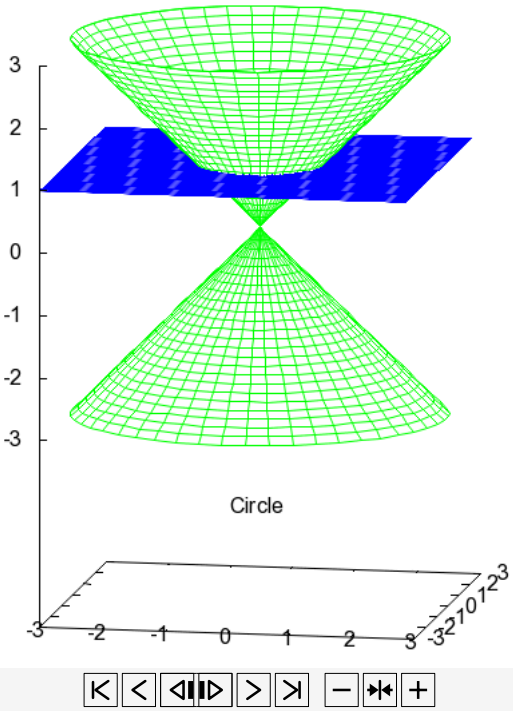
— строит последовательность 3D-графических объектов *obj*, зависящих от параметра *k*, пробегающего список *list*. Функции `with_slider_draw3d` можно передать несколько графических объектов, но их количество не может зависеть от параметра *k*.

Конические сечения. Меняя коэффициент наклона плоскости, получаем последовательно: окружность, эллипс, параболу (когда плоскость параллельна одной из образующих конуса) и гиперболу.

Листинг 164

```
(%i1)  kill(k)$

       /*  Подписи к рисункам  */
       s[0]:"Circle"$
       s[-1/4]:"Ellipse"$
       s[-1/2]:"Ellipse"$
       s[-1]:"Parabola"$
       s[-4]:"Hyperbola"$
       s[-8]:"Hyperbola"$

       with_slider_draw3d(
         k,[0,-1/4,-1/2,-1,-4,-8],
         dimensions=[550,650],
         proportional_axes=xyz,
         surface_hide=true,
         view=[80,10],
         xu_grid=50,
         yv_grid=50,
         zrange=[-3,3],
         color=blue,
         /*  Плоскость сечения  */
         explicit(1+k*(x-1), x,-3,3, y,-3,3),
         color=green,
         /*  Конус  */
         cylindrical(z, z,-3,3, t,0,2*%pi),
         color=black,
         /*  Подпись  */
         label([s[k], 0,0,-4.5])
       )$
```

(%o8)



4.7. *Задания для самостоятельной работы*

4.1 Не используя интерпретатора Maxima, найдите значения выражения

```
if not (if x then not x) then x
```

при x=`true, false`. Проверьте свой ответ с помощью интерпретатора.

4.2 Выразите оператор `for-in-do` через оператор `for-thru-do`.

4.3 Выразите оператор `go` через оператор `while-do` (или `unless-do`).

4.4 Функция принимает на входе показатель степени n и произвольное количество положительных чисел $x_1, x_2, \ldots, x_k$. Функция возвращает среднее n-степенное[19] чисел $x_1, x_2, \ldots, x_k$. Запрограммируйте такую

[19] Средним n-степенным чисел $x_1, x_2, \ldots, x_k$ называется число $\overline{x} = \left(\frac{x_1^n + x_2^n + \ldots + x_k^n}{k}\right)^{1/n}$. При $n = 1$ — среднее арифметическое, $n = 2$ — среднее квадратичное, $n = -1$ — среднее гармоническое.

функцию.

4.5 Постройте функцию, принимающую на входе числовое множество и возвращающую сумму тех его элементов, на которых достигает максимума функция `sin`.

4.6 Постройте аналог функции `apply`, принимающую в качестве аргумента множество вместо списка.

4.7 Символы $0123456789ABCDEFGHIJKLMNOPQRSTUV$ примем за цифры 32-чной системы счисления (т.е. $A = 10$, $B = 11$, ..., $V = 31$). Запрограммируйте функции, выполняющие преобразование числа из 10-чной системы счисления в 32-чную и обратно.

4.8 Булева функция $f(x_1, x_2, x_3, x_4)$ может быть задана списком своих значений $[y_0, y_1, \ldots, y_{15}]$, где $y_0 = f(0,0,0,0)$, $y_1 = f(0,0,0,1)$, ..., $y_{15} = f(1,1,1,1)$. Запрограммируйте функцию, принимающую на входе список значений булевой функции f и возвращающую *совершенную дизъюнктную нормальную форму* (СДНФ) функции f.

4.9 Рассмотрим рекурсивное задание последовательности многочленов в листингах 118, 119. Который из предложенных способов можно было бы назвать правильным?

4.10 Определите префиксный оператор, который будучи примененным к математическому выражению, вызывает вычисление этого выражения в числах с плавающей точкой.

4.11 Определите скобочный оператор, возвращающий множество переданных ему аргументов (операндов).

4.12 Перечислите функции и операторы, которые работают с выражениями как со списками.

4.13 Проанализируйте сходства и различия между функциями, выполняющими замену подвыражений: `subst`, `sublis`, `psubst` и `ratsubst`.

4.14 Напишите скрипт, который в произвольном математическом выражении меняет все операторы `*` на операторы `+` и наоборот. Объясните, почему выражение $x - y$ будет преобразовано в $x(y - 1)$.

4.15 Замените в данном алгебраическом выражении все квадратные корни на кубические, а кубические — на квадратные.

4.16 Напишите скрипт, принимающий на вход математическое выражение и возвращающий список всех имеющихся в этом выражении зна-

менателей.

4.17 Запрограммируйте функцию, возвращающую внутреннее представление выражения в соответствии с примерами, приведенными на стр. 133.

4.18 Как сделать, чтобы Maxima:

- вывела выражение $\sin(\pi)$, не вычисляя его значения;
- вывела выражение $y + x + z$, не переупорядочивая входящие в него идентификаторы?

4.19 Список натуральных чисел $[x_1, \ldots, x_n]$ называется *разбиением числа* x, если $x = x_1 + \ldots + x_n$. В разбиении допускаются одинаковые слагаемые. Для заданного натурального x выведите список всех его разбиений. Порядок элементов списка учитывать не надо.

4.20 Постройте исправленную функцию `read_list`, которая будет корректно читать из файла произвольный список. Запишите в файл список из 1000 случайных рациональных дробей. Проверьте работу функции, прочитав этот список из файла.

4.21 Потренируйтесь программировать математические функции на языке Lisp. Проверяйте работу вычислениями и построением графика.

4.22 Кривая провисания проводов между столбами называется *цепной линией*. Найдите информацию об этой кривой и постройте ее, воспользовавшись объектом `explicit`.

4.23 Муха села на колесо движущегося велосипеда. Постройте траекторию движения мухи. Задайте координаты (x, y) как функции времени t и воспользуйтесь графическим объектом `parametric`.

4.24 Нарисуйте выпуклый многоугольник, заданный системой неравенств

$$\begin{cases} a_1 x + b_1 y \geqslant c_1 \\ \ldots \\ a_n x + b_n y \geqslant c_n \end{cases}$$

4.25 В произвольный эллипс вписан шестиугольник. Покажите на графике, что точки пересечения продолжений противоположных сторон шестиугольника лежат на одной прямой.

4.26 Положим $S_0 = 0$. В каждый момент времени $t = 1, 2, \ldots$ случайная величина $S_t = S_{t-1} + X_t$, где X_t принимает значения $-1, 1$ с равными вероятностями. Функция (последовательность) S_t является примером

случайного процесса и называется *случайным блужданием*. Постройте график случайного блуждания. Воспользуйтесь для этого графическим объектом `points`.

4.27 Скачайте с http://finance.yahoo.com/ данные о каком-либо финансовом индексе[20] и постройте график зависимости цены закрытия от времени.

4.28 Запрограммируйте *игру «Жизнь»*[21]. Для изображения клеток воспользуйтесь графическим объектом `rectangle`.

4.29 Поверхность невесомой мыльной пленки, натянутой на две окружности, плоскости которых перпендикулярны прямой, соединяющей их центры, называется *катеноидом*. Найдите информацию об этой поверхности и постройте ее.

4.30 Поверхность $x^2+y^2-z^2=1$ называется *однополостным гиперболоидом*. Поскольку уравнение поверхности можно переписать двумя способами:

$$(x-z)(x+z)=(1-y)(1+y) \quad \text{и} \quad (y-z)(y+z)=(1-x)(1+x),$$

то через каждую ее точку (x_0,y_0,z_0) проходит две прямые:

$$\begin{cases} (x_0-z_0)(x+z)=(1-y_0)(1+y) \\ (x-z)(x_0+z_0)=(1-y)(1+y_0) \end{cases}$$

и

$$\begin{cases} (y_0-z_0)(y+z)=(1-x_0)(1+x) \\ (y-z)(y_0+z_0)=(1-x)(1+x_0) \end{cases},$$

полностью лежащие на поверхности. Изобразите это на графике.

[20] Например, RTS.RS: http://finance.yahoo.com/q/hp?s=RTS.RS+Historical+Prices
[21] http://ru.wikipedia.org/wiki/Жизнь_(игра)

Список литературы

1. Maxima Reference Manual.
http://maxima.sourceforge.net/docs/manual/en/maxima.html — 1048 p.

Регулярно обновляемая официальная документация системы Maxima. На сайте — в виде pdf-файла, в составе системы — в более удобном формате chm. Несмотря на впечатляющий объем, документация во многих случаях не дает исчерпывающей информации.

2. *Стахин Н.А.* Основы работы с системой символьных вычислений Maxima. М.: Федеральное агентство по образованию, 2008 — 86 с.

В пособии подробно обсуждается большое количество наглядных примеров. Можно рекомендовать для первого знакомства с системой Maxima.

3. *Чичкарёв Е.А.* Компьютерная математика с Maxima. Руководство для школьников и студентов. М.: ALT Linux, 2009 — 384 с.

Единственная книга по Maxima, вышедшая на русском языке в широкой печати. Несмотря на название, книга не подходит для первоначального знакомства с системой.

4. *Dodier R.* Minimal Maxima. 2005 — 20 p.

Опубликовано на странице
http://maxima.sourceforge.net/documentation.html
Очень краткое пособие, написанное всемирно известным специалистом по символьным вычислениям и одним из разработчиков системы Maxima. Для новичков не подходит.

5. *Leydold J., Petry M.* Introduction to Maxima for Economics. Institute for Statistics and Mathematics, WU Wien, 2011 — 119 p.

Опубликовано на странице автора
http://statmath.wu.ac.at/~leydold/maxima
Материал в пособии удобно организован в цветные таблицы. Разобрано много примеров. В конце рассмотрено несколько задач с экономическим содержанием. На наш взгляд, это самое удачное пособие по Maxima для новичков.

Указатель служебных имен Maxima

Номер страницы с описанием соответствующего объекта языка Maxima выделен жирным.

Перечень листингов

Листинг 110 (121). Из данного списка рекурсивно удаляем элементы (`delete`), совпадающие с очередным элементом списка.

Листинг 109 (120). Сравнение цикла и рекурсии. Вычисление арифметико-геометрического среднего.

Листинг 111 (122). Создание цикла внутри блока с помощью функции `go`. Выводим список всех простых чисел, лежащих на данном отрезке.

Листинг 112 (122). Задание пользовательской функции: оператор присвоения `:=`.

Листинг 113 (123). Идентификатор может быть одновременно именем функции и именем переменной.

Листинг 114 (124). Область видимости пользовательской функции внутри блока, функция `local`.

Листинг 115 (125). Математическую функцию в Maxima необязательно задавать как пользовательскую функцию.

Листинг 116 (126). Задание пользовательской функции, принимающей произвольное число аргументов.

Листинг 117 (126). Задание функции, принимающей в качестве аргументов списки и множества.

Листинг 118 (126). Рекурсивное построение функции.

Листинг 119 (127). Построение индексированной функции.

Листинг 120 (128). Примеры применения анонимных `lambda`-функций: сортировка списка, отображение списка, выбор подсписка, разбиение на подмножества и на классы эквивалентности, экстремальные подмножества.

Листинг 121 (129). Для множества $\{1, 2, \ldots, n\}$ выводятся все его разбиения.

Листинг 122 (130). Примеры задания пользовательских операторов различных типов.

Листинг 123 (134). Функции `length`, `cons`, `endcons`, `member`, `delete` работают с произвольными выражениями как со списками.

Учебное издание

Евгений Валерьевич Маевский,
кандидат физико-математических наук
Пётр Владимирович Ягодовский,
кандидат физико-математических наук, доцент

КОМПЬЮТЕРНАЯ МАТЕМАТИКА

Высшая математика в СКМ Maxima
Часть I. Введение

Публикуется в авторской редакции.

Учебное пособие

Оформление обложки и титульного листа В.А. Селин
Формат 60 × 90/16. Гарнитура Computer Modern
Подписано в печать 17.03.2014
Усл.п.л. 12,25. Уч.изд.л. 8,50
Тираж 16 экз. Заказ № 33

Финансовый университет
Ленинградский проспект, 49, Москва, 125993 (ГСП-3)
Отпечатано в ООП (Настасьинский пер., д.3, стр.1)
Издательства Финансового университета