

В. П. Дьяконов

Mathematica 5.1/5.2/6

Программирование и математические вычисления



Москва, 2008

УДК 32.973.26-018.2

ББК 004.438

Д93

Д93 Дьяконов В. П.

Mathematica 5.1/5.2/6. Программирование и математические вычисления.
– М.: ДМК-Пресс, 2008. – 576 с.: ил.

ISBN 5-94074-405-2

В книге впервые описаны основы программирования и применения трех последних версий системы Mathematica 5.1, 5.2 и 6.0. Все они – мировые лидеры среди универсальных систем компьютерной математики. Особое внимание уделено описанию новейшей версии Mathematica 6.0, в ядро которой добавлено свыше тысячи новых функций и команд, введены уникальные средства динамической оценки переменных, визуализации любых видов вычислений и динамического графического интерфейса ноутбуков (документов). Описаны сотни примеров применения систем. Для всех пользователей ПК, применяющих математические методы в образовании, в инженерной практике и в научных расчетах и, прежде всего, желающих освоить программирование в системах Mathematica.

УДК 519.6

ББК В162я73

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 5-94074-405-2

© Дьяконов В. П., 2008

© Оформление, издание, ДМК-Пресс, 2008

Краткое содержание

Введение	29
-----------------------	-----------

Глава 1	
ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ	
И РАБОТА С МАТНЕМАТИСА 5/6	33

Глава 2	
ТИПОВЫЕ СРЕДСТВА	
ПРОГРАММИРОВАНИЯ	97

Глава 3	
ТИПЫ ДАННЫХ, ОПЕРАТОРЫ	
И ФУНКЦИИ	157

Глава 4	
ФУНКЦИИ РАБОТЫ СО СЛОЖНЫМИ	
ТИПАМИ ДАННЫХ	203

Глава 5	
ФУНКЦИИ МАТЕМАТИЧЕСКОГО	
АНАЛИЗА	231

Глава 6	
ФУНКЦИИ ОБРАБОТКИ ДАННЫХ,	
ФУНКЦИЙ И СИГНАЛОВ	285

Глава 7	
ФУНКЦИИ СИМВОЛЬНЫХ	
ПРЕОБРАЗОВАНИЙ	369
 Глава 8	
СРЕДСТВА ПРОГРАММИРОВАНИЯ	
ГРАФИКИ	399
 Глава 9	
СПЕЦИАЛЬНЫЕ СРЕДСТВА	
ПРОГРАММИРОВАНИЯ	495
 Список литературы	570
 Алфавитный указатель	574

Содержание

Введение	29
-----------------------	-----------

Глава 1

Интерфейс пользователя

и работа с Mathematica 5/6	33
---	-----------

1.1. Пуск системы и начало работы с ней	34
---	----

1.1.1. История появления системы Mathematica и ее место	34
--	----

1.1.2. Инсталляция и запуск системы Mathematica 5	35
---	----

1.1.3. Главное меню и окно редактирования документов	36
---	----

1.1.4. Палитры математических операторов и функций	37
---	----

1.1.5. Первые навыки работы и понятие о ноутбуках (документах)	38
---	----

1.2. Работа с файлами (File)	40
------------------------------------	----

1.2.1. Основные виды файлов и пакеты расширения	40
---	----

1.2.2. Команды позиции File меню	41
--	----

1.2.3. Работа с файлами документов	42
--	----

1.2.4. Операции с файлами со специальным форматом	42
--	----

1.2.5. Преобразование документов в палитру и наоборот	43
--	----

1.2.6. Печать ноутбуков	43
-------------------------------	----

1.2.7. Команда завершения работы с системой – Exit	43
--	----

1.3. Редактирование документа (Edit)	44
--	----

1.3.1. Основные понятия о документах и их стилях	44
--	----

1.3.2. Выделения в документах и использование мыши	44
---	----

1.3.3. Подготовка текстовых комментариев	46
--	----

1.3.4. Команды позиции Edit главного меню	47
1.3.5. Операции с буфером промежуточного хранения	48
1.3.6. Специальные команды правки	48
1.3.7. Установка предпочтений	49
1.4. Работа с ячейками (Cell)	50
1.4.1. Понятие о ячейках документов	50
1.4.2. Команды позиции Cell главного меню	50
1.4.3. Манипуляции с ячейками	51
1.4.4. Работа с графическими и звуковыми возможностями	54
1.5. Операции форматирования ячеек (Format)	55
1.5.1. Команды позиции Format главного меню	55
1.5.2. Изменение стиля документов	55
1.5.3. Опции стилей и программ и их изменение	56
1.5.4. Уточненное управление стилем документов	57
1.5.5. Установка стиля интерфейса	58
1.6. Ввод элементов документов (Input)	58
1.6.1. Ввод координат двумерных графиков	58
1.6.2. Работа с селектором обзора трехмерных графиков	59
1.6.3. Изменение цветовой гаммы	61
1.6.4. Работа с фонографом	61
1.6.5. Вставка файла	61
1.6.6. Ввод таблиц, матриц и палитр	61
1.6.7. Ввод и редактирование кнопок	62
1.6.8. Вставка гиперссылки	62
1.6.9. Создание и ввод специальных объектов	63
1.6.10. Вставки, связанные с ячейками	64
1.6.11. Вставки имен функций и списков их параметров ...	65

1.7. Управление работой ядра системы (Kernel)	65
1.7.1. Команды позиции Kernel главного меню	65
1.7.2. Управление процессом вычислений	65
1.7.3. Выбор ядра системы	67
1.7.4. Управление показом номеров ячеек	67
1.7.5. Удаление всех ячеек вывода	68
1.8. Операции поиска и замены	68
1.8.1. Обзор подменю Find	68
1.8.2. Команды поиска и замены	68
1.8.3. Обнаружение и открытие выделенных строк	68
1.8.4. Работа с этикетками	69
1.9. Управление окнами (Windows)	69
1.9.1. Команды позиции Windows главного меню	69
1.9.2. Управление расположением и вывод специальных окон	69
1.10. Работа с информационными ресурсами системы Mathematica	70
1.10.1. Справка по системе Mathematica 5	70
1.10.2. Открытие справочной базы данных Mathematica 5.2	70
1.10.3. Работа со справкой Mathematica 5.1/5.2	71
1.10.4. Другие команды меню Help	73
1.11. Возможности системы Mathematica 5.2	74
1.11.1. Увеличение функциональности системы	74
1.11.2. Поддержка многоядерных микропроцессоров	74
1.11.3. Увеличение скорости вычисления математических функций	76
1.11.4. Поддержка 64-разрядных микропроцессоров	77

1.11.5. Повышение производительности в обычных условиях	78
1.12. Интерфейс пользователя системы Mathematica 6	78
1.12.1. Запуск Mathematica 6 и изменения в меню системы	78
1.12.2. Справочная система Mathematica 6	80
1.13. Особенности системы Mathematica 6	83
1.13.1. Основные новинки системы Mathematica 6	83
1.13.2. Скорость работы Mathematica 6	85
1.13.3. Ориентация в изучении системы на примеры ее применения	86
1.13.4. Динамическая интерактивность при символьных вычислениях	87
1.13.5. Управление графиками мышью	89
1.13.6. Динамическая интерактивность при графической визуализации	90
1.13.7. Комплексное тестирование Mathematica 6 на скорость вычислений	93

Глава 2

Типовые средства программирования

97

2.1. Mathematica как система программирования ...	98
2.1.1. Понятие о входном языке системы и языке реализации	98
2.1.2. Возможности языка программирования системы Mathematica	98
2.1.3. Структура систем Mathematica	100
2.1.4. Идеология систем Mathematica	101
2.1.5. Пакеты расширения Add-On	101

2.1.6. Полная и частичная загрузка пакетов расширения Add-On	102
2.1.7. Применение пакетов Add-On системы Mathematica 6	102
2.1.8. Концепция динамического изменения переменных в Mathematica 6	104
2.2. Функции символьных вычислений	106
2.2.1. Понятие о символьных (аналитических) вычислениях	106
2.2.2. Диагностика ошибок	107
2.2.3. Простые примеры из математического анализа	108
2.2.4. Точная арифметика	109
2.2.5. Проблемы символьных вычислений	110
2.2.6. Проверка результатов вычислений	113
2.2.7. Удаление введенных в ходе сессии определений ..	113
2.3. Применение образцов	114
2.3.1. Понятие об образцах	114
2.3.2. Задание свойств функций с помощью образцов	114
2.3.3. Задание в образцах типов данных	115
2.3.4. Типы образцов	115
2.4. Основы функционального программирования в среде Mathematica	116
2.4.1. Суть функционального программирования	116
2.4.2. Функции пользователя	117
2.4.3. Задание чистых функций	118
2.4.4. Анонимные функции	119
2.4.5. Суперпозиция функций	120
2.4.6. Функции FixedPoint и Cath	120
2.4.7. Реализация рекурсивных и рекуррентных алгоритмов	121

2.5. Основы процедурного программирования	122
2.5.1. Однострочные процедуры и их задание	122
2.5.2. Блоки для задания процедур	123
2.6. Организация циклов	123
2.6.1. Для чего нужны циклы	123
2.6.2. Циклы типа Do	124
2.6.3. Циклы типа For	126
2.6.4. Циклы типа While	126
2.6.5. Директивы-функции прерывания и продолжения циклов	127
2.7. Условные выражения и безусловные переходы	128
2.7.1. Функция If	129
2.7.2. Функции-переключатели	130
2.7.3. Безусловные переходы	131
2.8. Механизм контекстов	132
2.8.1. Старые проблемы	132
2.8.2. Что такое контекст?	133
2.8.3. Работа с контекстами	134
2.8.4. Получение списков определений с контекстами	135
2.9. Программирование ввода-вывода	136
2.9.1. Осуществление интерактивного диалога	136
2.9.2. Задание формата вывода	137
2.10. Функции задания объектов GUI ноутбуков	140
2.10.1. Слайдеры однокоординатные	140
2.10.2. Слайдеры двухкоординатные	141

2.10.3. Элементы установки опций CheckBox	141
2.10.4. Локаторы	142
2.10.5. Функции управления и контроля мышью	142
2.10.6. Кнопка с надписью	144
2.10.7. Манипулятор	144
2.10.8. Задатчик угла поворота радиус-вектора	144
2.10.9. Выпадающее меню акций	145
2.10.10. Панель ввода выражений	146
2.10.11. Радиокнопки и меню установок	147
2.10.12. Слайдер изменения цвета	149
2.10.13. Спусковой «механизм»	150
2.10.14. Функции указания места на объекте	151
2.10.15. Вывод сообщения при активизации объекта мышью	152
2.10.16. Вывод меню и выбор его позиций	154
2.10.17. Вывод меню с вкладками и их переключение	154
2.10.18. Вывод слайд-меню	155
2.10.19. Конструирование отдельных окон с GUI	155

Глава 3

Типы данных, операторы и функции

3.1. Работа с простыми типами данных	158
3.1.1. Типы данных системы	158
3.1.2. Работа с целыми числами	158
3.1.3. Работа с числами вещественного типа	160
3.1.4. Работа с комплексными числами	162
3.2. Работа со сложными типами данных	163
3.2.1. Символьные данные и строки	163
3.2.2. Выражения	163

3.3. Работа с объектами и функциями	164
3.3.1. Объекты и идентификаторы	164
3.3.2. Функции, опции, атрибуты и директивы	165
3.4. Применение констант и размерных величин...	167
3.4.1. Применение констант	167
3.4.2. Физические константы и размерные величины	168
3.5. Работа с переменными	168
3.5.1. Расширенное понятие о переменных	168
3.5.2. Назначение переменным идентификаторов (имен)	169
3.5.3. Особенности применения переменных	169
3.5.4. Эволюция значений переменных и операции присваивания	170
3.5.5. Предполагаемые переменные	171
3.6. Применение подстановок	172
3.6.1. Назначение подстановок	172
3.6.2. Подстановки с помощью оператора /.	172
3.6.3. Подстановки с помощью операторов -> и :>	173
3.7. Задание и применение функций пользователя	173
3.7.1. Задание функций пользователя	173
3.7.2. Сохранение на диске и считывание функций пользователя	174
3.7.3. Задание функций пользователя с синтаксисом языков программирования	174
3.8. Средства арифметических вычислений	175
3.8.1. Арифметические операторы	175

3.8.2. Особенности выполнения арифметических операций	176
3.8.3. Рационализация чисел	177
3.8.4. Укороченная форма записи арифметических операций	178
3.9. Функции арифметических операций	179
3.9.1. Встроенные функции	179
3.9.2. Основные арифметические функции	179
3.9.3. Функции генерации случайных чисел	181
3.9.4. Функции выявления погрешностей и анализа структуры чисел	183
3.10. Логические операторы и функции	183
3.10.1. Логические операции	183
3.10.2. Основные логические функции	184
3.10.3. Дополнительные логические функции	186
3.11. Работа с математическими функциями	187
3.11.1. Функции комплексного аргумента	187
3.11.2. Элементарные функции	188
3.11.3. Ортогональные многочлены	189
3.11.4. Интегральные показательные и родственные им функции	190
3.11.5. Гамма- и полигамма-функции	191
3.11.6. Функции Бесселя	192
3.11.7. Гипергеометрические функции	193
3.11.8. Эллиптические интегралы и интегральные функции	193
3.11.9. Функции Эйри	194
3.11.10. Бета-функция и относящиеся к ней функции	195
3.11.11. Специальные числа и полиномы	195

3.11.12. Другие специальные функции	196
3.11.13. Новые специальные функции в Mathematica 6	198
3.12. Расширенные возможности работы с объектами	199
3.12.1. Оперативная помощь	199
3.12.2. Средства диагностики и сообщения об ошибках	199
3.12.3. Включение и выключение сообщений об ошибках	200
3.12.4. Защита от модификации и ее отмена	201
 Глава 4	
Функции работы со сложными типами данных	203
4.1. Создание списков и выделение элементов списков	204
4.1.1. Создание списков	204
4.1.2. Генерация списков	205
4.1.3. Выделение элементов списков	206
4.1.4. Вывод элементов списков	208
4.2. Выявление структуры списков	209
4.2.1. Функции выявления структуры списков	209
4.2.2. Примеры выявления структуры списков	211
4.3. Работа со списком в стеке	211
4.3.1. Понятие о стеке	211
4.3.2. Работа со стеком	212
4.4. Манипуляции с элементами списков	212

4.4.1. Включение в список новых элементов	212
4.4.2. Удаление элементов из списка	213
4.4.3. Изменение порядка элементов в списке	214
4.4.4. Комбинирование списков и работа с множествами	215
4.4.5. Другие функции для работы со списками	216
4.5. Базовые средства линейной алгебры	217
4.5.1. Задание массивов	217
4.5.2. Векторные функции	218
4.5.3. Функции для операций линейной алгебры	219
4.5.4. Функции декомпозиции матриц	221
4.5.5. Решение систем линейных уравнений	222
4.6. Новые средства работы со списками в Mathematica 6	223
4.6.1. Работа с оператором ;; для списков	223
4.6.2. Новые функции для работы со списками	224
4.6.3. Новые функции для массивов, векторов и матриц	225
4.7. Работа со строками	227
4.7.1. Функции работы со строками	227
4.7.2. Примеры работы со строковыми функциями	228
4.7.3. Дополнительные функции работы со строками	228

Глава 5

Функции математического анализа

231

5.1. Функции вычисления сумм и произведений рядов	232
5.1.1. Функция вычисления сумм	232

5.1.2. Функция вычисления сумм в численном виде	233
5.1.3. Функция вычисления произведений	234
5.1.4. Функция вычисления произведений в численном виде	235
5.2. Функции вычисления производных	236
5.2.1. Основные функции для вычисления производных ...	236
5.2.2. Примеры вычисления производных	237
5.2.3. Примеры вычисления обобщенных производных ..	239
5.3. Вычисление первообразных и определенных интегралов	240
5.3.1. Вычисление интегралов в символьном виде	240
5.3.2. Примеры на вычисление определенных интегралов	242
5.3.3. Примеры на вычисление кратных интегралов	243
5.3.4. Численное интегрирование в Mathematica 5.1/5.2	245
5.3.5. Численное интегрирование в Mathematica 6	247
5.4. Вычисление пределов функций	248
5.4.1. Функция для вычисления пределов Limit	248
5.4.2. Опции функции вычисления пределов	249
5.5. Функции решения алгебраических и нелинейных уравнений	250
5.5.1. Функция Solve для решения уравнений	250
5.5.2. Решение систем нелинейных уравнений в символьном виде	250
5.5.3. Опции функции Solve	251
5.5.4. Функции численного решения уравнений	253
5.5.5. Функции вычисления корней уравнений	255
5.5.6. Дополнительные функции для решения уравнений	256

5.5.7. Графическая иллюстрация и выбор метода решения уравнений	258
5.5.8. Получение одновременно нескольких корней	261
5.5.9. Получение неизвестных в явном виде	262
5.5.10. Решение рекуррентных уравнений	263
5.5.11. Решение уравнения Фробениуса в Mathematica 6	264
5.6. Решение дифференциальных уравнений	265
5.6.1. Решение дифференциальных уравнений в символьном виде	265
5.6.2. Решение дифференциальных уравнений в частных производных	267
5.6.3. Решение дифференциальных уравнений в численном виде	268
5.7. Функции минимизации и максимизации	269
5.7.1. Поиск максимального и минимального чисел в списке	270
5.7.2. Поиск локального минимума и максимума аналитической функции	271
5.7.3. Поиск глобального максимума и минимума аналитической функции	272
5.7.4. Функции оптимизации в Mathematica 5/5.1/5.2	273
5.7.5. Функции оптимизации в Mathematica 6	274
5.7.6. Визуализация оптимизации в Mathematica 6	275
5.8. Функции интегральных преобразований	276
5.8.1. Функции преобразований Лапласа	277
5.8.2. Функции Фурье-преобразований	279
5.8.3. Функции косинусного и синусного преобразований Фурье	282
5.8.4. Функции z-преобразований	283

Глава 6

Функции обработки данных, функций

и сигналов 285

6.1. Разложение функций в степенные ряды 286

6.1.1. Разложения в ряды Тейлора и Маклорена 286

6.1.2. Примеры разложения в ряды Тейлора и Маклорена 287

6.1.3. Удаление члена с остаточной погрешностью ряда 288

6.1.4. Графическая визуализация разложения в ряд 288

6.1.5. О разложении в ряд при большом числе членов 289

6.2. Средства синтеза сигналов 291

6.2.1. Синтез сигналов на основе встроенных функций ... 291

6.2.2. Гармонический синтез сигналов 292

6.3. Функции полиномиальной интерполяции и аппроксимации 294

6.3.1. Функции полиномиальной интерполяция 294

6.3.2. Пример полиномиальной аппроксимации 296

6.3.3. Погрешность полиномиальной аппроксимации 297

6.3.4. Полиномиальная аппроксимация специальных функций 297

6.3.5. Полиномиальная аппроксимация при большом числе узлов 299

6.3.6. Рациональная интерполяция и аппроксимация 301

6.3.7. Функции рациональной Паде-аппроксимация 305

6.3.8. Оптимизация аппроксимации 308

6.3.9. Методика минимаксной аппроксимации 310

6.3.10. Сплайновая интерполяция и аппроксимация 314

6.4. Регрессия и метод наименьших квадратов	315
6.4.1. Регрессия и визуализация ее результатов	315
6.4.2. Функции линейной регрессии	317
6.4.3. Функции нелинейной регрессии	318
6.4.4. Функции полиномиальной регрессии	320
6.4.5. Функции тригонометрической регрессии	322
6.5. Функции дискретного преобразования Фурье	323
6.5.1. Прямое и обратное дискретное преобразование Фурье	323
6.5.2. Спектральный анализ на основе прямого преобразования Фурье	325
6.5.3. Применение преобразования Фурье для получения спектра сигналов	326
6.5.4. Фильтрация сигналов с помощью преобразований Фурье	327
6.5.5. Расширенные функции для преобразования Фурье	329
6.6. Кусочные функции Piecewise	331
6.6.1. Задание кусочных функций	331
6.6.2. Работа с кусочными функциями	332
6.7. Новые средства Mathematica 6	333
6.7.1. Функции полиномиальной интерполяции	333
6.7.2. Пример трехмерной полиномиальной интерполяции	333
6.7.3. Полиномиальная интерполяция с заданием значений производной в узлах	333
6.7.4. Функция нелинейной регрессии FindFit	334

6.8. Функции для работы со звуковыми сигналами	335
6.8.1. Роль синтеза звука	335
6.8.2. Функции для работы со звуком	336
6.8.3. Примеры синтеза звуков в Mathematica 5.1/5.2	337
6.8.4. Работа со звуком в Mathematica 6	337
6.9. Функции для работы с потоками и файлами	341
6.9.1. Потоки и файлы	341
6.9.2. Упрощенная работа с файлами	341
6.9.3. Обычные средства для работы с файлами	342
6.9.4. Использование файлов других языков программирования	343
6.9.5. Запись в файл определений	344
6.9.6. Другие функции для работы с файлами	344
6.10. Системные функции	346
6.10.1. Функции времени и даты	346
6.10.2. Общесистемные функции	347
6.10.3. Общесистемные функции в Mathematica 6	349
6.11. Функции статистической обработки данных и массивов Statistics	350
6.11.1. Назначение пакета Statistics в Mathematica 5.1/5.2	350
6.11.2. Манипуляции с данными – DataManipulation	350
6.11.3. Стандартная обработка массива данных	352
6.11.4. Линейное сглаживание данных и их фильтрация	354
6.11.5. Экспоненциальное сглаживание	356

6.11.6. Функции непрерывного распределения вероятностей	357
6.11.7. Функции дискретного распределения	359
6.11.8. Графика пакета Statistica	360
6.11.9. Другие функции статистики	361
6.12. Статистические вычисления в Mathematica 6	363
6.12.1. О пакете расширения Statistics в системе Mathematica 6	363
6.12.2. Аналитические статистические расчеты	363
6.12.3. Численные статистические расчеты в Mathematica 6	365
6.12.4. Статистические расчеты с графической визуализацией	365

Глава 7

Функции символьных преобразований 369

7.1. Работа с выражениями	370
7.1.1. Полная форма выражений	370
7.1.2. Основные формы выражений	371
7.1.3. Части выражений и работа с ними	371
7.1.4. Удаление элементов выражения	373
7.1.5. Другие манипуляции с выражениями	373
7.1.6. Контроль выражений	375
7.2. Работа с функциями	375
7.2.1. Приложение имени функции к выражению или его части	375
7.2.2. Укороченная форма функций	376
7.2.3. Выделение заданного аргумента в функциях	376

7.2.4. Подстановки в функциях	377
7.2.5. Рекурсивные функции	377
7.2.6. Дополнительные примеры на работу с функциями	378
7.2.7. Инверсные функции	379
7.3. Задание математических отношений	379
7.3.1. Для чего нужно задание новых отношений	379
7.3.2. Примеры задания математических отношений	380
7.4. Функции упрощения выражений	381
7.4.1. Роль упрощения выражений	381
7.4.2. Основная функция Simplify	382
7.4.3. Примеры упрощения выражений функцией Simplify	383
7.4.4. Функция полного упрощения FullSimplify	383
7.5. Раскрытие и расширение выражений	384
7.5.1. Функции раскрытия и расширения выражений	384
7.5.2. Примеры расширения и раскрытия выражений	385
7.5.3. Функция Collect	386
7.5.4. Функции преобразования тригонометрических выражений	387
7.6. Функции и директивы для работы с полиномами	389
7.6.1. Определение полинома (степенного многочлена)	389
7.6.2. Основные операции над полиномами	390
7.6.3. Разложение полиномов – функции класса Factor ...	390
7.6.4. Функции для работы с полиномами	392
7.6.5. Примеры работы с полиномами	392

7.7. Расширенные операции с выражениями	394
7.7.1. Функции для расширенных операций с выражениями	394
7.7.2. Примеры расширенной работы с выражениями	396
7.7.3. Средства работы с выражениями в Mathematica 6	397

Глава 8

Средства программирования графики 399

8.1. Построение графиков функций одной переменной	400
8.1.1. Графическая функция Plot	400
8.1.2. Опции функции Plot	400
8.1.3. Применение опций функции Plot	402
8.1.4. Директивы двумерной графики и их применение ..	405
8.1.5. Построение графика по точкам – функция ListPlot	407
8.1.6. Получение информации о графических объектах	408
8.2. Перестройка и комбинирование графиков	409
8.2.1. Директива Show	409
8.2.2. Примеры применения функции Show	409
8.3. Примитивы двумерной графики	410
8.4. Построение графиков в полярной системе координат	412
8.4.1. Задание функции в параметрической форме	412
8.4.2. Функции для построения параметрически заданных графиков	413

8.4.3. Примеры построения графиков в полярной системе координат	414
8.5. Построение контурных графиков	415
8.5.1. Функции для построения контурных графиков	415
8.5.2. Опции для функций контурной графики	416
8.5.3. Примеры построения контурных графиков	417
8.6. Построение графиков плотности	419
8.6.1. Функции графиков плотности	419
8.6.2. Примеры построения графиков плотности	420
8.7. Построение графиков поверхностей	421
8.7.1. Принципы построения поверхностей и фигур	421
8.7.2. Основные функции для построения 3D графиков ..	421
8.7.3. Опции 3D графики	421
8.7.4. Директивы трехмерной графики	423
8.7.5. Примеры модификации 3D графиков с помощью опций	424
8.7.6. Графическая функция ListPlot3D	429
8.7.7. Параметрическая 3D графика	430
8.7.8. Построение фигур, пересекающихся в пространстве	432
8.8. Примитивы трехмерной графики и их применение	434
8.8.1. Функция Graphics3D и ее опции и примитивы	434
8.8.2. Примеры применения функции Graphics3D с примитивами	436
8.9. Дополнительные средства графики Mathematica 5.1/5.2	438
8.9.1. Импорт графических изображений	438

8.9.2. Экспорт графических изображений	439
8.9.3. Вставка графических и иных объектов	440
8.10. Новые средства графики в Mathematica 6	442
8.10.1. Позиция Graphics меню и графический редактор	442
8.10.2. Расширение возможностей функции Plot	445
8.10.3. Использование опций закрашки областей двумерных графиков	445
8.10.4. Графические динамические модули в Mathematica 6	448
8.10.5. Визуализация данных из списков	450
8.10.6. Рельефная графика	453
8.10.7. Трехмерные объекты, полученные вращением кривых	454
8.10.8. Визуализация работы клеточных автоматов	458
8.10.9. Графы, деревья и прочее	461
8.11. Функции пакета расширения Graphics	461
8.11.1. Функции анимационной графики	461
8.11.2. Управление цветом графиков	465
8.11.3. Построение стрелок	467
8.11.4. Задание картографических систем	467
8.11.5. Построение объемных контурных графиков – ContourPlot3D	468
8.11.6. Построение графиков с окраской внутренних областей	470
8.11.7. Графики логарифмические и полулогарифмические	472
8.11.8. Графики в полярной системе координат	473
8.11.9. Построение столбиковых и круговых диаграмм	473
8.11.10. Объединение графиков различного типа	475

8.11.11. Трехмерные столбиковые диаграммы	476
8.11.12. Построение точек и кривых в пространстве	477
8.11.13. Построение графиков поверхности и ее проекций	477
8.11.14. Построение графиков неявных функций	478
8.11.15. Вывод обозначений кривых – легенд	479
8.11.16. Построение графиков с примитивами	480
8.11.17. Построение трехмерных заданных параметрически графиков	481
8.11.18. Трехмерные графики в сферической и цилиндрической системах координат	481
8.11.19. Построение графиков полей	481
8.11.20. Построение пространственных фигур стереометрии	482
8.11.21. Создание графических форм	485
8.11.22. Построение фигур, пересекающихся в пространстве	485
8.11.23. Применение сплайнов	486
8.11.24. Функции построения фигур вращения	486
 8.12. Идеология применения пакета Graphics в Mathematica 6	 488
8.12.1. Роль пакета Graphics в Mathematica 6	488
8.12.2. Представление точек графиков произвольными объектами	488
8.12.3. Функция PolyhedronData	488
8.12.4. Функция GraphData	490
8.12.5. Функция GraphicsGrid	491
8.12.6. Директива вставки Inset	492
8.12.7. Директива непрозрачности Opacity	493

Глава 9

Специальные средства

программирования 495

9.1. Функциональное программирование специальной графики 496

9.1.1. Пример программирования графической задачи .. 496

9.1.2. Задание функции для построения фрактала Манделброта 497

9.1.3. Задание функции для построения модели деления клеток 498

9.2. Подготовка пакетов расширений системы Mathematica 499

9.2.1. Типовая структура пакетов расширения 499

9.2.2. Средства создания пакетов расширений 501

9.2.3. Текстовые сообщения и комментарии 501

9.2.4. Примеры подготовки пакетов расширений 502

9.2.5. Подготовка пакетов применений 504

9.3. Отладка и трассировка программ 505

9.3.1. Некоторые правила культурного программирования 506

9.3.2. Трассировка программных модулей 507

9.3.3. Основные функции трассировки и отладки 508

9.4. Новые средства программирования в Mathematica 6 510

9.4.1. Динамическое изменение переменных и функция Dynamic 510

9.4.2. Динамический модуль DynamicModule 511

9.4.3. Функция сброса интерактивных изменений Deploy	512
9.4.4. Модуль манипуляций Manipulate	512
9.4.5. Средства отладки программ и ноутбуков	513
9.5. Обзор пакетов расширения Add-On	518
9.5.1. Состав пакетов расширения Add-On систем Mathematica 5.1/5.2	518
9.5.2. Пакет алгебраических функций Algebra	520
9.5.3. Пакет вычислительных функций Calculus	523
9.5.4. Функции дискретной математики – пакет DiscreteMath	527
9.5.5. Функции вычислительной геометрии	529
9.5.6. Функции геометрических расчетов – пакет Geometry	532
9.5.7. Расширение в теории чисел – пакет NumberTheory	535
9.5.8. Функции численных расчетов – расширение NumberMath	540
9.5.9. Функции работы со звуком пакета Miscellaneous ...	545
9.5.10. Функции для работы с географическими объектами	551
9.5.11. Физические и химические данные	556
9.5.12. Задание данных только вещественного типа – RealOnly	561
9.5.13. Пакет расширения с утилитами – Utilities	562
9.6. Данные о других средствах расширения	566
Список литературы	570
Алфавитный указатель	574

Введение

Первые попытки создания компьютерных программ для аналитических вычислений были предприняты еще на первых ламповых ЭВМ. Но их технические возможности (в частности, ничтожно малый по нынешним временам объем памяти и низкая скорость вычислений) не позволили этому направлению развиваться всерьез и дать практические, а не только абстрактные, результаты. Поэтому говорить о появлении реальных систем компьютерной алгебры стало возможным только после появления ЭВМ третьего поколения класса «Мир-2» и «Мир-3», созданных научной школой академика В. М. Глушкова. Эти ЭВМ, увы, тоже с довольно скромными техническими характеристиками, имели язык программирования Аналитик, поддерживающий аналитические вычисления [1].

Однако массовое развитие компьютерная алгебра получила после появления программируемых микрокалькуляторов и персональных компьютеров, выполненных на интегральных схемах большой степени интеграции (БИС) [2]. Это случилось уже в 80–90-х годах прошлого века. Тогда уже появились первые графические калькуляторы с ориентацией на научные расчеты и с системами символьных вычислений [7, 8] и первые книги по средствам персональных массовых вычислений и компьютерной алгебре [2–7, 9, 10]. Появилось и множество разработок коммерческих систем компьютерной математики (СКМ), описанных в первой обобщающей монографии по этому направлению [8] и во множестве книг [8–21, 26–33].

Побудительным мотивом к подготовке этой книги стало появление новейшей версии системы Mathematica – шестой и отсутствие в нашей литературе книг по ней и даже по двум предшествующим реализациям этой системы (Mathematica 5.1 и Mathematica 5.2). Система Mathematica изначально была лидером среди систем компьютерной алгебры. Версии Mathematica 5.1/5.2, ныне самые известные и популярные среди пользователей системами этого класса, были направлены на сохранение этого лидерства.

Однако новейшая версия Mathematica 6 – это уже не просто лидер. Это поистине революционный программный продукт с огромным числом новаций, выдвигающих систему в особое положение, и придающий ей возможности, которые, скорее всего, покажутся фантастическими для многих пользователей системами компьютерной математики! Это касается новаций интерфейса документов системы, высочайшей скорости вычислений и огромного набора полезных функций.

К сожалению, современные версии системы Mathematica в русскоязычной литературе описаны очень мало. Так, из ряда книг [23–32] только три [29–30] содержат описание уже изрядно устаревшей версии Mathematica 5. Книг по версиям Mathematica 5.1/5.2/6 до настоящего времени не было, хотя каждая очередная версия вела к существенному обновлению системы. Особенно существенным было отсутствие книг по основам программирования в системах класса Mathematica.

Данная книга впервые описывает основы программирования и применения трех последних версий систем компьютерной алгебры и компьютерной математики – Mathematica 5.1/5.2/6. Они созданы фирмой Wolfram Research, Inc. (США). В ней автор книги прошел научную стажировку (рис. 0.1). Многие материалы, полученные во время этой стажировки, включены в данную книгу.



Рис. 0.1. Автор книги В. Дьяконов (в центре) с главой фирмы С. Вольфрамом (справа) и одним из ведущих менеджеров Р. Гермундсоном (слева) на международной конференции по MathML (США)

Особенностью этой книги, в отличие от ближайшей к ней по тематике книги автора [29], является то, что она в первую очередь ориентирована на программистов и описывает систему Mathematica как мощный профессионально-ориентированный на математические и научно-технические расчеты язык программирования сверхвысокого уровня. Огромное число встроенных средств вычислений позволяет решать подавляющее большинство задач на языке Mathematica, предельно приближенном к обычному языку математических вычислений. Пользователь начинает программировать в среде Mathematica, фактически, начиная с первых страниц книги. Но книга полезна и обычному пользователю, даже не знакомому с программированием.

Это далеко не первая книга автора по СКМ, объединяющим средства компьютерной алгебры и численных расчетов. За многие годы работы в области компьютерной математики автором опубликовано около полусотни книг практически по всем версиям систем компьютерной математики. В списке литературы данной

книги указаны только первые и последние (на момент завершения рукописи этого труда) книги по таким широко известным системам, как Mathcad, Maple, MATLAB, Derive и др. [10-21].

Серьезное знакомство автора со всеми СКМ позволило сделать вывод, что две системы – Mathematica и Maple стали бесспорными лидерами среди систем, ориентированных на аналитические расчеты. Они как пара лидирующих спортсменов, попеременно вырывающихся вперед. В наше время есть множество серьезных оснований считать, что Mathematica 6 в настоящее время явно вышла на первое место. И серьезные пользователи СКМ должны учитывать это новое важное обстоятельство.

За рубежом система Mathematica широко применяется не только в научно-технических и математических расчетах, но и в преподавании многих дисциплин в университетах, вузах и даже школах [58-76]. Новые потрясающие возможности визуализации вычислений (в том числе графической и динамической) резко расширяют возможности последних версий Mathematica в сфере образования.

К сожалению, что отмечалось, в нашей литературе системы Mathematica 5.1/5.2/6 не были описаны и их новые возможности нашим читателям неизвестны. Соответственно, место этих систем в нашей науке (в том числе технической) и в нашем образовании объективно не оценено и неадекватно мировой значимости этих систем. Этот серьезный пробел и призвана восполнить эта книга.

Материал данной книги дан так, что основная его часть может использоваться пользователями любой версии системы Mathematica, начиная с объявленных в заголовке книги и даже более ранних версий. Новые возможности Mathematica 6, которые отсутствуют в предшествующих версиях, выделены в отдельные разделы. Это позволяет избежать недоразумений, связанных с существенно разным числом функций, операторов и команд в системах Mathematica 5.1/5.2 и Mathematica 6 (в Mathematica 6 число функций и команд только в ядре удвоилось) и различной организацией доступа к ним.

В книге умеренного объема, такой как эта, просто физически невозможно описать все около 3000 функций системы Mathematica 6. Если отвести каждой функции и примерам по ней только по страничке, то даже краткий справочник только по функциям системы имел бы свыше 3000–4000 страниц! Именно столько страниц (и даже больше) имеют в совокупности книги Михаила Трота – одного из разработчиков системы Mathematica [73–76].

На роль подобных детальных справочников, для большинства пользователей характерных большой информационной избыточностью, данная книга не претендует. Поэтому главным в ее написании было изучение основных функций системы и тщательный подбор тех из них, которые соответствуют тематике книги (математические и научно-технические расчеты). Ответственность за удачу или неудачу такого подбора лежит на авторе книги.

Разумеется, автором использовался материал по его предшествующим книгам по системам Mathematica [26–29], который накапливался многие годы, материал справок (увы, англоязычных) и полученная автором во время стажировки в Wolfram Research, Inc. обширная документация (в том числе электронная) по системе.

Однако во всех случаях речь идет не о формальном переводе справки или указанной документации, ввиду их огромного объема просто невозможного, а об их творческой переработке. В список литературы данной книги включены только книги по Mathematica 3 и более поздним версиям. Описание более ранних версий в наше время уже неактуально.

Данная книга продолжает крупную серию книг автора, посвященных применению современных СКМ в массовых научно-технических и учебных расчетах. К их освоению и изучению автор пришел после многих лет работы с микрокалькуляторами и персональными компьютерами (ПК), что нашло отражение в первых книгах автора [6,7] и в последующих. При написании этой книги использовался опыт автора в подготовке этих книг и участие автора в подготовке ряда научных конференций, например 8 конференций «Системы компьютерной математики и их приложения», проведенных в Смоленском государственном университете. Это позволило подготовить данную книгу достаточно быстро – до того, как могла бы появиться новая версия системы Mathematica.

Автор благодарит корпорацию Wolfram Research, Inc., ее создателя и главного разработчика систем класса Mathematica С. Вольфрама (S. Wolfram) и одного из ведущих сотрудников фирмы Wolfram О. Маричева за интерес, проявленный к работе автора, и прекрасные условия, созданные для стажировки автора на этой фирме, что позволило собрать ряд бесценных сведений, многие из которых вошли в эту книгу и до этого не публиковались. Автор благодарит также менеджера фирмы Анну Форейман за помощь в работе и ряд других сотрудников Wolfram Research, Inc., охотно продемонстрировавших автору замечательные возможности систем класса Mathematica. Некоторые из них описаны в данной книге.

С фирмой Wolfram Research, Inc., разработчиком математических систем Mathematica, вы можете связаться по адресу:

Wolfram Research, Inc

100 Trade Center Drive, Champaign, IL 61820, USA.

<http://www.wolfram.com>

E-mail: info@wri.com

Тел.: 217-398-0700. Факс: 217-398-0747

С автором можно связаться по электронной почте vpdyak@keytown.com.

Интерфейс пользователя и работа с Mathematica 5/6

1.1. Пуск системы и начало работы с ней	34
1.2. Работа с файлами (File)	40
1.3. Редактирование документа (Edit)	44
1.4. Работа с ячейками (Cell)	50
1.5. Операции форматирования ячеек (Format)	55
1.6. Ввод элементов документов (Input)	58
1.7. Управление работой ядра системы (Kernel)	65
1.8. Операции поиска и замены	68
1.9. Управление окнами (Window)	69
1.10. Работа с информационными ресурсами системы Mathematica	70
1.11. Возможности системы Mathematica 5.2	74
1.12. Интерфейс пользователя системы Mathematica 6	78
1.13. Особенности системы Mathematica 6	83

1.1. Пуск системы и начало работы с ней

1.1.1. История появления системы *Mathematica* и ее место

Система Mathematica 1, появившаяся в 1988 г., стала первой серьезной системой компьютерной алгебры [22]. Очередная версия Mathematica 2 могла уже работать под операционные системы MS-DOS и Windows 3.0 [23–26]. Системы вызвали живейший интерес со стороны учащихся, преподавателей вузов и университетов, аспирантов, инженеров и научных работников во всем мире. В том числе и в России, хотя у нас этот период совпал с распадом СССР и разрушением основ советской науки.

Версии Mathematica 3/4 [26, 27] обеспечивали системе лидирующее место среди систем компьютерной математики конца XX века – начала XXI века. Уже они обеспечивали практическое решение огромного числа математических, физических и научно-технических задач [34–57]. Число операторов и функций в ядре систем было доведено до более чем 1000, в пакетах расширения до 800.

Однако в начале XXI века Mathematica стала испытывать острую конкуренцию со стороны других систем компьютерной математики. В области компьютерной алгебры в лидеры пробилась система Maple [20, 21], созданная изначально в университетских кругах и быстро развивающаяся (последняя версия этой системы – Maple 11). Кстати, за подготовку книги [21] по Maple 9.5/10 автор стал победителем всероссийского конкурса «Лучшая научная книга 2006», проведенного фондом развития общественного образования, в номинации «Информационные технологии».

Среди систем для численных расчетов и моделирования ведущее место заняла мощная матричная система MATLAB с пакетом блочного математического моделирования Simulink. Для более или менее полного ее описания пришлось подготовить серию из пяти книг [13–17]. Большую известность получила система Mathcad с ее бесподобным математически ориентированным интерфейсом и тщательным отбором входящих в ее ядро функций [10, 11]. Определенную часть рынка заняли малые системы Derive [19, 20] и MuPAD [8], созданные для целей образования.

В связи с этим лидирующее место системы Mathematica в России и в странах СНГ стало менее определенным. Нередко начинающие пользователи отказывались от Mathematica просто потому, что им был плохо понятен язык функционального программирования этой системы, их пугали необычность фиксации ввода клавишами **Shift+Enter** (обычно ввод фиксируется клавишей **Enter**), необычные имена функций, например **Sin[x]**, с указанием параметров в квадратных скобках, и прочие неожиданные для них «мелочи». Но главное – первые версии Mathematica имели малое число практических примеров применения системы. Кроме того, многим казалось, что система слабо и долго модернизируется – смена

основного номера версии у разработчиков системы (фирма Wolfram Research, Inc.) занимала несколько лет, тогда как другие системы обновлялись практически ежегодно. Это видно по номерам их последних версий, например, Maple 11 и Mathcad 14.

Свою роль в неверной оценке возможностей и популярности системы Mathematica сыграли не слишком объективные, а то и явно некомпетентные Интернет-форумы по СКМ. Например, судя по форумам на сайте Exponenta.Ru, система Mathematica имеет худший рейтинг популярности среди систем MATLAB, Mathcad, Maple и Mathematica. Некоторые такие «форумы», например, по системе Maple на Exponenta.Ru, были просто захвачены некомпетентными и неизменно анонимными участниками. Их работа сводится к подсказкам от одних студентов другим в решении задач курсовых или дипломных работ и проектов. Разумеется, что по такому «общественному» обсуждению судить о популярности той или иной системы компьютерной математики нельзя. И просто глупо!

Между тем, кажушиеся недостатки системы Mathematica нередко оборачиваются ее достоинствами. Так, большое время между разработками версий Mathematica 3, 4, 5 и 6 на деле свидетельствует об их тщательной проработке. Так, показанные автору еще в 2000 г. элементы динамического изменения переменных, интерактивного динамического интерфейса и другие очень полезные и интересные возможности, впервые описанные в этой книге, по настоящему вошли только в шестую версию системы. И неслучайно ее разработчики считают эту версию по значимости сравнимой с Mathematica 1 – системой, в свое время приведшей к появлению систем компьютерной алгебры и компьютерной математики на персональных компьютерах.

Но еще важнее то, что новые реализации Mathematica 5.1/5.2/6 – это действительно универсальные математические системы, одинаково быстро, эффективно и надежно выполняющие как аналитические (символьные), так и численные вычисления. Даже загружается система быстрее других систем. Есть все основания считать, что Mathematica действительно выполняет вычисления быстрее своих конкурентов, в частности, благодаря впервые реализованной поддержке возможностей современных сверхскоростных микропроцессоров – в том числе многоядерных.

1.1.2. Установка и запуск системы Mathematica 5

Установка системы Mathematica 5 (в том числе версий 5.1 и 5.2) ничем не отличается от установки любого приложения под операционную систему Windows. После установки на рабочем столе появляется ярлык, активизация которого приводит к загрузке системы в память компьютера и появлению окон системы (рис. 1.1).

Первоначально после запуска Mathematica появляются панель меню и большое окно рабочего документа (ноутбука). На рис. 1.2 показаны также окна с информацией о системе Mathematica. Их можно убрать активизацией кнопки со знаком «X».

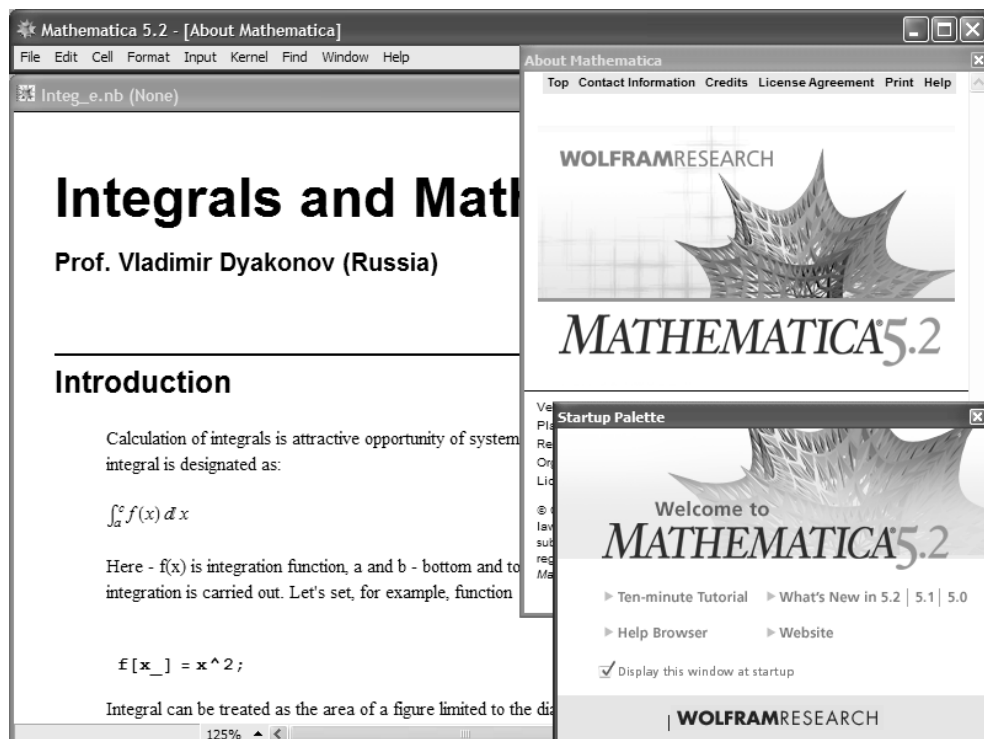


Рис. 1.1. Обзор интерфейса системы Mathematica 5.2

1.1.3. Главное меню и окно редактирования документов

Панель главного меню имеет всего две строки:

- с названиями системы и загруженного файла;
- позициями главного меню.


Справа и снизу окна редактирования находятся линейки прокрутки с характерными ползунками, управляемыми мышью. В самом низу в начале линейки прокрутки имеется так называемая статусная строка с информацией о текущем режиме работы (Status bar). Эта информация (если она есть в данный момент) полезна для оперативного контроля в ходе работы с системой.

Висящее главное меню системы (рис. 1.1 сверху) содержит следующие позиции:

- **File** – работа с файлами: задание нового файла, выбор файла из каталога, закрытие файла, запись текущего файла, запись файла с изменением имени, печать документа и выход в Windows;

- **Edit** – основные операции редактирования (отмена операции, копирование выделенных участков документа в буфер с их удалением и без удаления, перенос выделенных участков, их стирание);
- **Cell** – работа с ячейками (объединение и разъединение ячеек, установка статуса ячейки, открытие и закрытие);
- **Format** – установка форматов документов;
- **Input** – задание элементов ввода (графиков, матриц, гиперссылок и др.);
- **Kernel** – управление ядром системы;
- **Find** – поиск заданных данных;
- **Window** – операции с окнами и их расположением;
- **Help** – управление справочной системой.

Каждая позиция меню, будучи активной, порождает выпадающее подменю, содержащее относящиеся к ней команды. Названия выполняемых команд выделяются четким, а не выполняемых в данное время – характерным серым расплывчатым шрифтом.

Элементы интерфейса, в частности окно редактирования, можно перетаскивать мышью (зацепившись курсором мыши за титульную строку и удерживая нажатой левую клавишу) или растягивать в разные стороны. Курсор мыши обычно имеет вид , но меняется при установке на определенные детали элементов интерфейса. Например, при установке на вертикальную границу окна он приобретает вид двухсторонних стрелок \leftrightarrow , расположенных по горизонтали. Они указывают на возможность перемещения этой линии по горизонтали. Аналогично можно растягивать или сжимать окно перемещением по вертикали или диагонали.

В начале титульных строк главного меню и окна редактирования имеется кнопка с логотипом системы, открывающая подменю со следующими командами:

- **Восстановить** – восстановить размеры элемента интерфейса;
- **Переместить** – переместить элемент интерфейса;
- **Размер** – задать размеры элемента интерфейса;
- **Свернуть** – свернуть элемент в бирку в панели задач Windows;
- **Развернуть** – развернуть элемент интерфейса;
- **Закрыть** – закрыть элемент интерфейса.

Это подменю создается средствами операционной системы Windows. Если используется локализованная (русифицированная) версия последней, то она имеет надписи на русском языке. Кроме того, в конце этих строк есть характерные кнопки, повторяющие три последние команды. Они служат для управления окнами соответствующих элементов интерфейса (пока панели главного меню и окна редактирования). Эти кнопки хорошо знакомы пользователям приложениями под Windows.

1.1.4. Палитры математических операторов и функций

Для облегчения ввода математических выражений Mathematica имеет выводимые пользователем и перемещаемые по экрану в любое место инструментальные панели с множеством пиктограмм ввода математических символов, функций и

команд управления системой. Они выводятся с помощью подменю **Palettes** (Палитры) в позиции **File** главного меню системы (рис. 1.2).

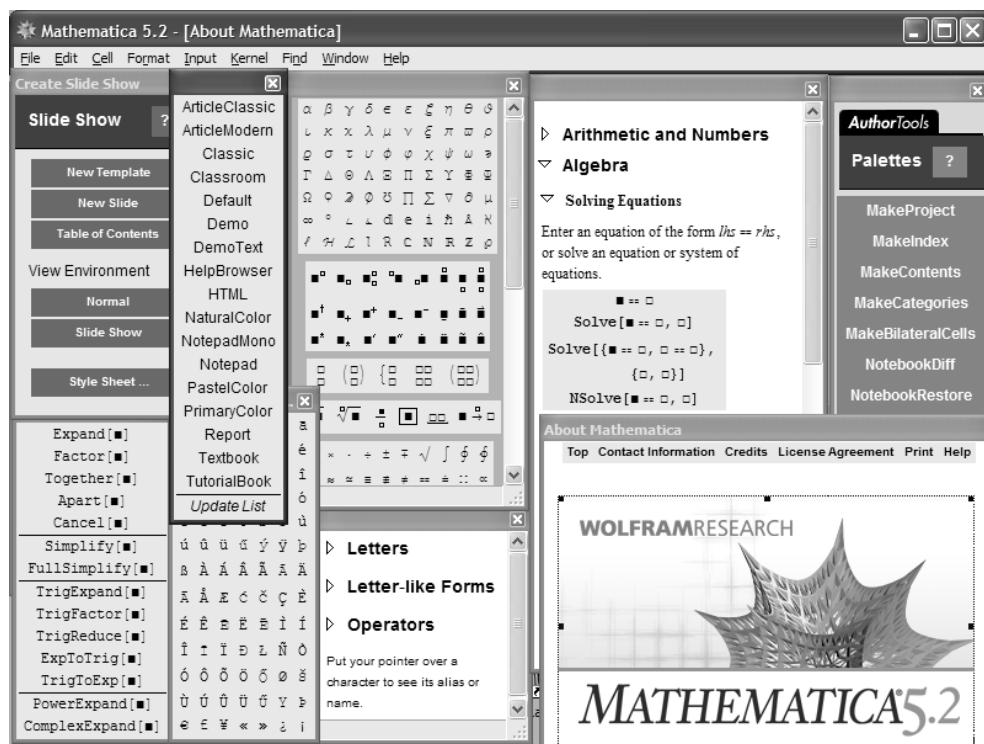


Рис. 1.2. Окно системы Mathematica 5.2 со всеми палитрами

Общее число специальных математических знаков (греческих и латинских букв, операторов, функций и команд), вводимых с помощью палитр, около 700. Многие знаки имеют альтернативные варианты ввода с применением комбинаций клавиш – их можно найти в справочной базе данных системы. Целесообразно пользоваться не более чем 2–3 панелями одновременно. Для удаления ненужных панелей в правом верхнем углу каждой панели расположены маленькие кнопки со знаком «x». Все панели максимально компактны и могут перетаскиваться мышью в наиболее удобное место экрана.

1.1.5. Первые навыки работы и понятие о ноутбуках (документах)

Работа с документами сводится к набору в ячейках ввода выражений (например, математических) и их исполнению. Для исполнения выражений достаточно нажать клавиши **Shift** и **Enter** одновременно (сама по себе клавиша **Enter** использу-

ется только для задания перевода строки внутри текущей строки ввода). К примеру, чтобы вычислить $2+3$, необходимо вначале ввести это выражение в строку ввода. В Mathematica строка ввода формируется по мере ввода объектов выражений 2, + и 3. После нажатия клавиш **Shift** и **Enter** получим:

In[1]:= 2+3

Out[1]= 5

Любопытно, что при первом вычислении Mathematica выполняет его с заметной задержкой. Это связано с загрузкой ядра системы (см. главу 2). В дальнейшем подобные вычисления происходят практически мгновенно.

Документ системы Mathematica строками (ячейками) ввода и вывода, текстовыми комментариями, рисунками и т.д. очень напоминает страницу блокнота учебного или инженера (рис. 1.3). Потому он и называется ноутбуком (от английского notebook).

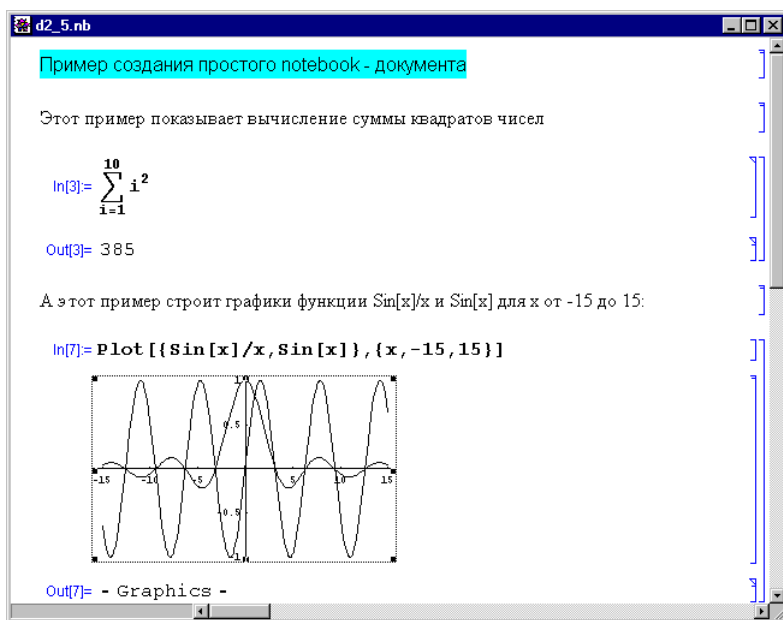


Рис. 1.3. Простейший блокнот (ноутбук)

Отдельные ячейки с математическими выражениями и результатами их вычислений отмечаются в правой части главного окна редактирования характерными тонкими квадратными скобками синего цвета. Это позволяет отслеживать то, к чему относятся математические выражения – к исходным данным или результатам. Кроме того, ячейки могут иметь различный статус, который отмечается соответствующими значками над квадратными скобками (об этом более подробно будет изложено ниже).

Для того, чтобы документ имел наглядный вид блокнота (ноутбука), необходимо предпринять определенные операции по форматированию документа и приданию ему нужного вида. Прежде всего, каждый шаг вычислений следует снабжать поясняющими надписями. Их можно прямо вводить в строки ввода, но затем отформатировать под текстовый формат подходящего стиля. Для этого выделяется строка ввода (установкой маркера ввода на ее скобку и щелчком левой клавиши мыши) с текстовой надписью. Пространство внутри скобки при этом затеняется (делается черным). Затем выполняется команда **Format – Style – Text** (Alt+7). Она задает текстовый формат надписи, который является неисполняемым.

С помощью других команд позиции **Format** главного меню, которые мы рассмотрим в дальнейшем, можно задать надпись разным шрифтом, разным цветом с выделением фона и т.д. Как уже отмечалось, для ввода математического выражения по шаблону и для представления его в естественной математической форме используется стандартный формат StandardForm ячеек ввода.

В блокнотах желательно, чтобы форма представления математических выражений хотя бы напоминала общепринятую. Mathematica позволяет задавать формы представления документов, принятые в таких мощных языках программирования, как Fortran, C и даже TeX (язык для программирования типографского набора сложных научных текстов).

Каждая надпись, математическое выражение или график занимают отдельную ячейку – Cell. Ячейка может занимать одну или несколько строк и всегда выделена своей квадратной скобкой. Важным свойством ячеек систем Mathematica является возможность их эволюции (изменения) по всему документу. Этим осуществляется динамический обмен данными в ходе символьных преобразований.

1.2. Работа с файлами (File)

1.2.1. Основные виды файлов и пакеты расширения

В новых версиях Mathematica 5/6 основным типом документов стали блокноты – notebooks. Им соответствуют файлы текстового формата с расширением .nb. Эти файлы могут редактироваться любым текстовым редактором, поддерживающим формат ASCII. Файлы содержат подробное описание документа с указаниями типов шрифтов, деталей оформления и местоположения различных объектов.

Кроме того, система имеет ряд пакетов расширения (в оригинале дополнения – AddOn) системы, расположенных в каталоге ADDON. Пакеты содержат множество (полторы сотни) библиотечных файлов с расширениями .m., в каждом из которых определен ряд новых функций системы. С их помощью можно реализовать новые алгоритмы решения математических задач и постоянно расширять возможности системы.

1.2.2. Команды позиции **File** меню

Для работы с файлами служит позиция **File** меню. Она позволяет задавать следующие команды:

- **New... Ctrl+N** – вывод окна нового документа;
- **Open... Ctrl+O** – вывод окна загрузки документа;
- **Close Ctrl+F4** – закрытие текущего окна;
- **Save Ctrl+S** – запись документа с текущим именем;
- **Save As... Shift+Ctrl+S** – запись документа с изменением имени;
- **Save As Special...** – запись в специальных форматах;
- **Open Special...** – открытие файлов в специальных форматах;
- **Open Selection...** – открытие выделенных файлов;
- **Import...** – импорт файлов (аналогично **Open...**);
- **Send To...** – отправка ноутбука по электронной почте;
- **Send Selection...** – отправка выделенной части ноутбука по электронной почте;
- **Palettes...** – вывод палитр математических спецзнаков, операторов и функций (см. выше);
- **Notebook...** – вывод списка документов, которые загружались ранее;
- **Generate Palette from Selection** – преобразует выделенные ячейки документа в палитру (окно с единственной кнопкой закрытия);
- **Generate Notebook from Palette** – преобразует палитру в документ со своим именем;
- **Printing Settings** – установка параметров печати;
- **Print... Ctrl+P** – вывод окна печати текущего документа;
- **Print Selection** – печать выделенных ячеек;
- **Exit Alt+F4** – завершение работы с системой.

Выбор любой команды в этой и в других позициях главного меню возможен любым из четырех способов:

- Выбор позиции подменю с помощью клавиш перемещения курсора и активизация этой позиции нажатием клавиши **Enter**.
- Выбор позиции нажатием клавиши выбора (она отмечена в позициях подменю горизонтальной чертой снизу) и нажатием затем клавиши **Enter**.
- Использование комбинации клавиш прямого доступа к команде (не требует активизации главного меню).
- Выбор позиции подменю с помощью мышки перемещением ее курсора и быстрым двойным нажатием левой клавиши мышки в момент, когда курсор находится на нужной позиции.

Следует отметить, что хотя библиотечные файлы расширений можно загружать, как и файлы с расширением .ma, в окно редактирования, как правило это делается только при их подготовке и редактировании. Указанные файлы обычно подгружаются в текущий документ без отображения их текстов с помощью специальных команд. Они будут рассмотрены в дальнейшем.

1.2.3. Работа с файлами документов

Команда **New** используется, когда нужно начать работу с новым документом. Эта команда полностью очищает экран с запросом о том, нужно ли записать текущий документ, если он есть и модернизировался. Документ получает имя *Untitled-N*, где *N* – текущий номер документа. Важно отметить, что эта команда не отменяет определений, сделанных в предшествующих исполненных документах и в ранее загруженных файлах пакетов расширений. Лишь полная перезагрузка системы отменяет эти определения.

Загрузка файлов ранее созданных документов – одна из самых распространенных операций. Она реализуется командой **Open...**, которая служит для загрузки ранее созданного документа с его поиском в файловой системе компьютера. Эта команда выводит стандартное диалоговое окно, типичное для Windows-приложений и предназначенное для удобного поиска файлов. Работа с этим окном пользователю Mathematica хорошо знакома и в особом описании не нуждается.

Для совместимости по интерфейсу пользователя с другими программами введена команда импорта файла – **Import...**. Эта команда аналогична команде **Open...**, и поэтому тоже детально не рассматривается. Обе команды позволяют загружать файлы как основного формата *notebook* с расширением *.nb*, так и файлы ряда других форматов.

Если документ создан после введения команды **New** или ввода с помощью команд **Open...** или **Import...**, то он обычно подвергается модификации и редактированию. После отладки документа иногда возникает необходимость записать его измененный вариант на магнитный диск – гибкий или жесткий. Для этого служат команды **Save** и **Save As...**. Три точки после имени этой (и других) команд означают, что команда исполняется не сразу, а после выполнения определенных действий (например, установок в диалоговом окне). Команда **Save** выполняет запись текущего документа без изменения его имени. Поэтому она выполняется быстро и без каких-либо дополнительных действий. Запись идет в формате *notebook*.

Команда **Save As...** позволяет изменить имя файла и поместить его в любую директорию любого диска. Эта команда вызывает появление стандартного диалогового окна. В этом окне помимо установок диска и нужной директории следует задать имя записываемого файла или подтвердить предлагаемое имя. Запись идет в формате *notebook*.

1.2.4. Операции с файлами со специальным форматом

Mathematica может записывать и считывать файлы, представленные в специальных форматах. Список этих форматов можно найти в подменю команды **Save As Special...**. Из специальных форматов файлов особо следует выделить три формата – TeX, HTML и XML. Формат TeX используется у весьма популярных у математиков редакторах математических текстов, насыщенных математическими зна-

ками и формулами. Форматы HTML и XML используются для подготовки страниц в Интернете [93]. Эти языки гиперссылок позволяют быстро переходить от одного документа к другому путем активизации гиперссылок. В последнее время формат HTML становится стандартным для подготовки электронных документов и книг, а также для создания высококачественных (в том числе обновляемых через Интернет) справочных систем. Вместе с XML может использоваться формат стандарта MathML, который используется для передачи через Интернет математической информации со сложными формулами.

Для загрузки файлов в этих форматах служит команда **Open Special...**, а для открытия выделенных файлов – команда **Open Selection....** Эти команды открывают довольно простые окна, с помощью которых устанавливаются данные, необходимые для открытия файлов.

1.2.5. Преобразование документов в палитру и наоборот

Любую часть документа после выделения можно преобразовать в палитру. Для этого используется команда **Generate Palette from Selection**.

Палитра – это уменьшенное окно, похожее на окно документа, но имеющее свое имя в титульной строке и только одну кнопку закрытия (у обычного окна их три). Палитру можно, как и документ, записывать на магнитные диски. Для преобразования палитры в документ со своим именем используется команда **Generate Notebook from Palette**.

1.2.6. Печать ноутбуков

Подготовленный документ обычно нуждается в печати. Mathematica под Windows не имеет своей системы печати и использует стандартную систему печати операционных систем Windows 95/98/NT/2000/XP. При этом окна настройки печати задаются драйверами печати, установленными для применяемых принтеров. К примеру, Windows 98/XP поддерживает сотни типов принтеров десятков фирм. Ввиду известности операций печати подробное описание их опущено. Для настройки печати служит команда **Printing Setting**, для печати – команда **Print**, а для печати выделенных ячеек ноутбука служит команда **Print Selection....** Печать идет по известному принципу WYSIWYG: что видишь, то и будет напечатано.

1.2.7. Команда завершения работы с системой – Exit

Команда **Exit** используется для окончания работы с системой. Если все документы, с которыми пользователь работал (их может быть много), были записаны на диск, то при исполнении этой команды можно наблюдать закрытие одного за дру-

гим окон с текстами документов. Если какой-то из документов не был записан после модернизации, то команда **Exit** сделает запрос о необходимости записи. Исполнение команды **Exit** завершается выходом в оболочку Windows.

В подменю **Notebooks** позиции **File** главного меню виден также перечень файлов, с которыми в последнее время работал пользователь. Указание любого из этих файлов ведет к его загрузке в текущее окно редактирования. Это делает систематическую работу с системой более удобной, так как избавляет пользователя от поиска наиболее нужных файлов по дискам и директориям.

1.3. Редактирование документа (Edit)

1.3.1. Основные понятия о документах и их стилях

Редактированием документа является всякое изменение текста комментариев, исходных данных и математических формул с целью придания документу более подходящего вида (стиля) или получения новых результатов. К редактированию относится и изменение формата графиков.

В общем случае документы характеризуются *стилем* оформления. От выбора стиля документа во многом зависит его наглядность и эстетичность восприятия. Поэтому в Mathematica предусмотрены обширные возможности по изменению стиля документов и их частей. Они сосредоточены в подменю команды **Format** главного меню. Однако обилие средств установки стиля порождает проблему совместимости стилей, поскольку нередко стили бывают несовместимыми. Для ее решения используются специальные средства преобразования стилей. Для ячеек они сосредоточены в подменю позиции **Cell** главного меню.

1.3.2. Выделения в документах и использование мыши

При редактировании документа курсор мыши приходится перемещать из одной ячейки в другую и обращаться к командам главного меню для выполнения тех или иных операций, например копирования содержания ячейки в буфер, изменения шрифта и т.д. Следует учитывать, что вид курсора при этом меняется и позволяет оценивать его местонахождение. Рекомендуется понаблюдать за изменением формы курсора мыши при его перемещении в различных областях документов.

Важным моментом в работе с документами является *выделение* их элементов – ячеек ввода и вывода, их содержимого и т.д. Для *выделения ячейки* достаточно установить курсор мыши на ее скобку и нажать левую клавишу – скобка заполнится

темным цветом. Если при этом нажать правую клавишу мыши, появится *контекстно-зависимое меню правой клавиши* с подменю позиции **Copy As...**, позволяющей скопировать содержимое выделенной ячейки в *буфер промежуточного хранения* Windows – clipboard (далее он будет именоваться просто *буфер*).

Возможно также выделение выражений внутри ячеек, а также выделение рисунков (рис. 1.4). Для выделения рисунка (графика) достаточно ввести курсор мыши в область рисунка и щелкнуть левой клавишей мыши. Рисунок будет обведен рамкой с характерными прямоугольниками. Цепляясь за них курсором мыши и нажав и удерживая левую клавишу мыши, можно растягивать график в разные стороны и менять его размер. В этом случае также можно вывести контекстно-зависимое меню правой клавиши мыши.

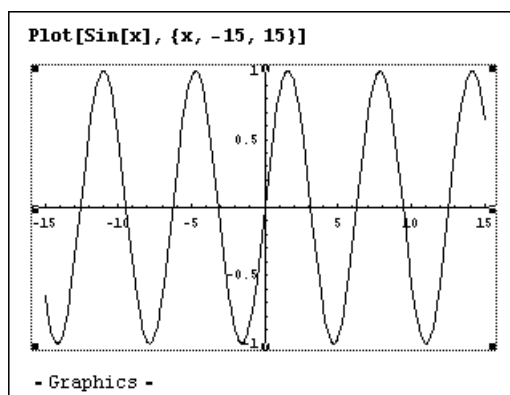


Рис. 1.4. Пример выделения графика в ячейке вывода

Контекстно-зависимые меню правой клавиши мыши очень удобны при профессиональной работе с системой Mathematica. Они дают полный перечень команд, которые можно использовать для выделенного объекта, не обращаясь к главному меню; там они также есть, но разбросаны по многочисленным позициям главного меню.

Когда курсор находится в пределах ячейки ввода или вывода, нажатие левой клавиши мыши вызывает выделение некоторой части этой ячейки. Выделение при этом можно расширять повторным нажатием левой клавиши мыши. Можно расширять область выделения и просто перемещением маркера ввода при постоянно нажатой левой клавише мыши.

Если курсор находится в ячейке ввода, то он используется для точного указания места, в котором начинается редактирование. Для фиксации курсора на этом месте нужно нажать левую клавишу мыши – курсор при этом превращается в жирную вертикальную черту. За пределами ячеек (т.е. в строках главного меню, линейки и т.д.) курсор мышки имеет обычный вид наклонной жирной стрелки.

1.3.3. Подготовка текстовых комментариев

Важной частью профессионально составленного документа являются текстовые комментарии. Без них документ через некоторое время становится непонятным даже его разработчику. Поэтому правилом хорошего тона является применение достаточно подробных текстовых комментариев.

Тестовые комментарии вводятся прямо в текущую строку ввода. Однако не следует завершать ввод нажатием комбинации клавиш **Shift+Enter**, так как это приведет к выводу комментария в строку вывода с возможными сообщениями об ошибке (рис. 1.5). Они обусловлены тем, что в текстовых комментариях обычно не придерживаются синтаксиса входного языка системы Mathematica, что и чревато появлением ошибок.

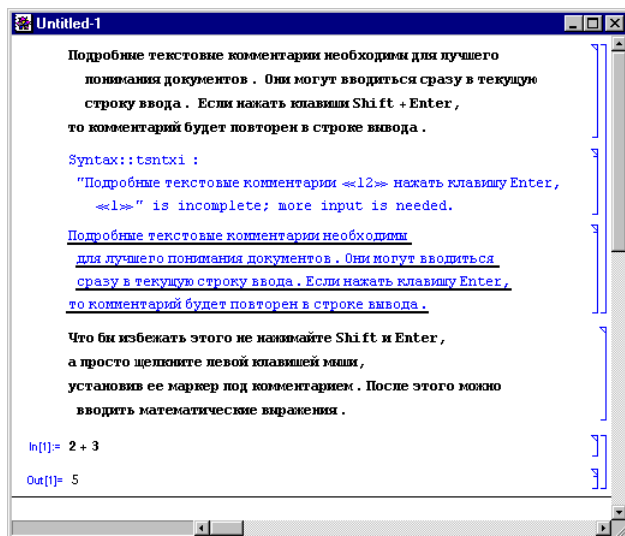


Рис. 1.5. Примеры ввода текстовых комментариев

Чтобы отмеченная ситуация не повторялась, просто установите маркер мыши под строку ввода с комментарием, а затем щелкните левой клавишей мыши; в новой строке ввода можно размещать новый комментарий или математические выражения для вычислений.

Часто в ходе редактирования приходится изменять текстовые комментарии, например заглавные надписи в документах. Для этого достаточно выделить ту ячейку, в которой находится надпись. Для этого подведите курсор мышки в квадратной скобке в правом конце ячейки – курсор при этом превращается в стрелку с вертикальной линией. Указав стрелкой нужную ячейку, нажмите левую клавишу мышки. Скобка выделенной ячейки заполняется черным цветом.

Далее можно выбрать тип оформления ячейки. Для установки стиля ячеек используется ряд команд, которые имеются в позиции **Style** подменю **Format**. Эти команды более подробно будут рассмотрены ниже.

К важной операции редактирования ячеек с текстами комментариев относится выравнивание текстов в пределах строки ввода. Для оперативного осуществления этой операции целесообразно вывести панель форматирования **ToolBar** и мерную линейку **Ruller**. Соответствующие команды есть в позиции **Format** главного меню. На рис. 1.6 представлено окно документа с этими средствами и примерами форматирования строки ввода с текстовыми комментариями. Для быстрого форматирования используются четыре кнопки с изображением соответствующего отформатированного текста.

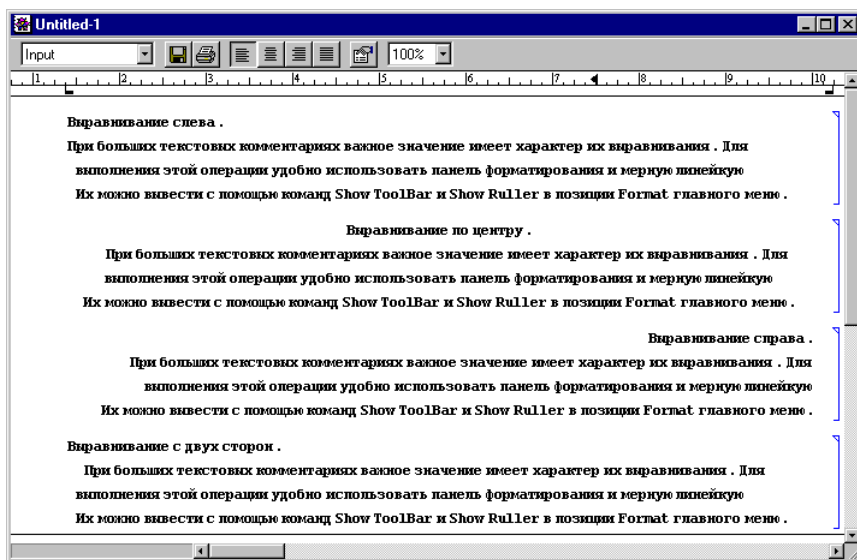


Рис. 1.6. Различные типы выравнивания текстовых надписей

Ряд расширенных возможностей редактирования представляют команды подменю, относящегося к позиции **Edit** (редактор) главного меню. В основном эти операции связаны с обменом информацией между выделенной (отмеченной) ячейкой или группой ячеек и специальным буфером.

1.3.4. Команды позиции **Edit** главного меню

Как и в других приложениях под Windows, средства правки (редактирования) ноутбуков сосредоточены в позиции **Edit** меню:

- **Undo Ctrl+Z** – отмена операции;
- **Cut Ctrl+X** – перенос в буфер содержимого ячейки;

- **Copy Ctrl+C** – копирование в буфер;
- **Paste Ctrl+V** – вызов из буфера без его очистки;
- **Clear Del** – уничтожение выделенной ячейки;
- **Paste and Discard** – вызов из буфера с его очисткой;
- **Save Selection As** – запись выделенных ячеек в специальных форматах;
- **Select All Ctrl+A** – выделение всех ячеек;
- **Insert Object** – включение объектов;
- **Motion** – различные перемещения в ячейках;
- **Expression Input** – ввод выражений в разных форматах;
- **Make 2D Shift+Ctrl+Y** – создание входных выражений в 2D формате;
- **Check Balance Shift+Ctrl+B** – включение проверки баланса скобок в выражениях;
- **Check Spelling...** – включение проверки орфографии;
- **Preferences** – вызов окна селектора опций предпочтений.

1.3.5. Операции с буфером промежуточного хранения

Как известно, операционные системы класса Windows имеют так называемый *буфер промежуточного хранения* (в дальнейшем просто *буфер*). Это специально организованная динамическая область памяти, в которую можно помещать информацию различного формата, например текстовую или графическую. Буфер используется как для редактирования, так и для обмена информацией между различными приложениями.

Команда **Cut** удаляет выделенную ячейку и помещает ее содержимое в буфер. Команда **Copy** делает то же самое, что и **Cut**, но без удаления выделенной ячейки. Команда **Paste** копирует содержимое буфера в место вставки, намеченное графическим курсором. При этом содержимое буфера сохраняется. Команда **Paste and Discard** переносит содержимое буфера на место, намеченное курсором, но при этом сам буфер очищается. Таким образом, в этом случае возможна только одна операция переноса. Ее применение разумно при последнем переносе, с тем чтобы высвободить память, занимаемую буфером. Команда **Clear** уничтожает выделенную ячейку без ее сохранения в буфере.

Хотя до сих пор речь шла о манипуляциях с одной ячейкой, они вполне возможны и с несколькими одновременно выделенными ячейками. При этом содержимое ячеек может быть любым: тексты, математические формулы или графики.

1.3.6. Специальные команды правки

Команда **Insert объект** открывает окно вставки объектов. Это стандартное окно, присущее всем приложениям операционной системы Windows. В окне есть перечень приложений, экспортирующих в Mathematica порожденные ими объекты. Это могут быть тексты, рисунки, документы различных программных систем и т.д. Такие объекты внедряются в ячейки Mathematica и могут редактироваться

теми программами, которые их породили. Позже мы рассмотрим технологию вставки объектов более подробно.

Для редактирования внутри ячеек блоков служат команды подменю **Motion**. Это подменю характерно именно для программы Mathematica и содержит команды перемещения маркера ввода по отдельным символам, словам и т.д. Следует отметить, что аккуратное их исполнение в случае текстов гарантируется только для англоязычных текстов.

Команда **Expression Input** открывает подменю с рядом команд, задающих вид ячеек. Это также специфическая команда, присущая системам Mathematica. Если необходимо представлять и редактировать ячейки ввода как двумерные объекты, то следует использовать команду **Make 2D**. Практика, однако, показывает, что гораздо проще вводить содержимое ячеек в обычном текстовом формате, чем в формате 2D. В этот формат легко перейти средствами изменения формата.

Меню Edit содержит также команды, управляющие проверкой правильной расстановки скобок (**Check Balance...**) и орфографии (**Check Spelling...**). Это редко используемые команды. Так команда проверки орфографии пригодна только для англоязычных текстов.

1.3.7. Установка предпочтений

Последняя команда **Preferences** в меню **Edit** имеет особое значение. Она открывает окно *установки предпочтений*, показанное на рис. 1.7. В нем сосредоточено

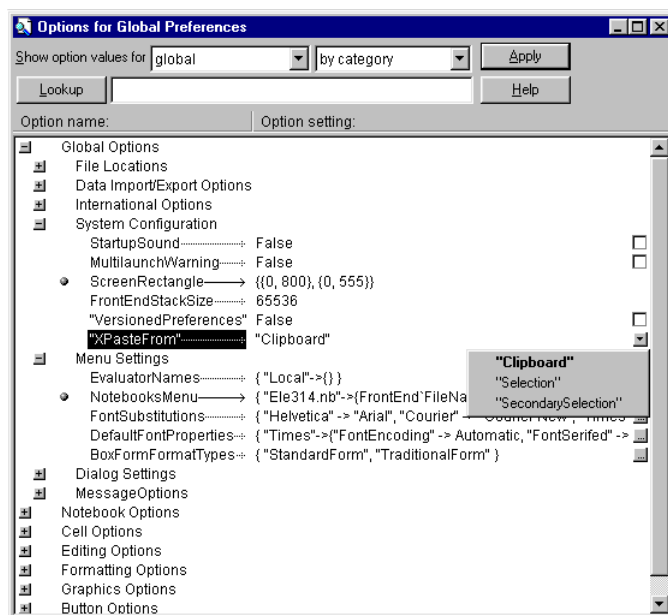


Рис. 1.7. Окно установки предпочтений

большое число опций, задающих все основные параметры системы. Эти опции определяют глобальные настройки системы и ноутбуков, в частности, выбор шрифтов, цветов, типов данных и т.п.

Настройки в окне предпочтений вряд ли стоит рекомендовать неопытному пользователю, поскольку в них легко запутаться из-за обилия опций. Однако опытные пользователи могут настроить работу с системой в соответствии со своими предпочтениями. Рекомендуется записывать сделанные установки и те, которые были перед их изменениями.

1.4. Работа с ячейками (Cell)

1.4.1. Понятие о ячейках документов

Итак, *ячейки* (Cells) являются основными объектами документов. Ячейки отличаются их *статусом*, т.е. совокупностью свойств, определяющих поведение ячейки в различных ситуациях и ее тип. Важным понятием, относящимся к ячейкам и отражающим особенности работы систем символьной математики, являются понятия изменения (эволюции) и модификации содержимого ячеек.

К примеру, ячейки вывода содержат текстовые надписи – комментарии не эволюционируют, т.е. не меняются в ходе исполнения документа. Ячейки ввода, напротив, эволюционируют (и их содержание меняется) и порождают ячейки вывода с разным содержимым; например ячейка, выражение которой содержит функцию $f[x]$, будет меняться в соответствии с изменением $f[x]$. Ячейки могут быть заблокированными от модификации и разблокированными и т.д. Статус ячеек постоянно проверяется с помощью операции *оценивания* в ходе исполнения документа.

Статус ячеек можно распознать и без исполнения документа по ряду характерных признаков. Один из них – вид курсора мышки при его размещении в области ячеек – был описан выше. Другой признак – малозаметный опознавательный знак в верхней части квадратной скобки, обрамляющей ячейку. Отсутствие знака означает, что это обычная ячейка ввода. Знак «-» (короткая горизонтальная черточка) отмечает ячейку вывода со статусом *Inactive*. Заблокированная (закрытая) ячейка (*Locked*) помечается знаком «x», а инициализированная ячейка (*Initialization*) – знаком «t». Кроме того, эволюционирующие ячейки отмечаются маленьким треугольником. О типе ячейки можно также судить по их стилю, в частности по шрифту используемых в ней символов.

Для переключения на нужный тип ячейки достаточно поместить курсор мышки на нужную надпись и нажать левую клавишу мышки. Соответствующая надпись о типе ячейки появится в окошке селектора типов ячеек.

1.4.2. Команды позиции *Cell* главного меню

Ячейки (cells) – это наиболее характерная деталь ноутбуков системы Mathematica. В позиции **Cell** главного меню предусмотрены команды для работы с ячейками.

При активной позиции **Cell** главного меню появляется подменю, имеющее ряд команд. Вначале рассмотрим команды общего характера:

- **Convert To** – преобразование формата ячеек;
- **Display As** – установка формата отображения ячеек;
- **Default Input Format Type** – установка формата по умолчанию для ячеек ввода;
- **Default Output Format Type** – установка формата по умолчанию для ячеек вывода;
- **Default Inline Format Type** – установка формата по умолчанию для ячеек Inline;
- **Cell Properties** – установка свойств ячеек;
- **Cell Grouping** – группировка ячеек;
- **Divide Cell Shift+Ctrl+D** – разделение сгруппированных ячеек;
- **Merge Cell Shift+Ctrl_M** – объединение ячеек.

Ниже эти команды описаны более подробно.

1.4.3. Манипуляции с ячейками

При вводе данных в ячейки ввода данные представляются в одном из форматов, заданных командой **Default Input Format Type**. Соответственно в ячейках вывода результаты представляются в формате, установленном командой **Default Output Format Type**. Однако есть возможность изменить формат данных в ячейках с помощью команды **Convert To** – преобразование формата ячеек. Эта команда открывает подменю с перечнем всех возможных форматов. Формат данной ячейки помечен знаком птички. Для задания другого формата надо установить его наименование, активизировав ячейку.

Возможна установка следующего формата ячеек:

- **InputForm Shift+Ctrl+I** – формат входа;
- **OutputForm** – формат выхода;
- **StandardForm Shift+Ctrl+N** – стандартный формат;
- **TradicionalForm Shift+Ctrl+T** – традиционный формат;
- **PostScript** – формат «Пост-Скрипт» черно-белой печати;
- **Bitmap** – побитовый формат изображений;
- **Metafile** – формат метафайлов.

Из этих форматов (стоит их просмотреть, поскольку форматов множество) особо следует отметить стандартный формат, который позволяет отображать формулы в ячейках ввода в наиболее приближенном к обычному виде, т.е. с применением стандартных математических знаков для интегралов, сумм, произведений и т.д. Указанные форматы фигурируют и в других подменю позиции **Cell** главного меню.

Позиция **Cell Properties** служит для установки свойств – статуса ячеек. Ее подменю содержит следующие позиции:

- **Cell Open** – делает ячейку открытой или закрытой;
- **Cell Editable** – делает ячейку редактируемой или нередатируемой;
- **Cell Edit Duplicate** – делает двойную ячейку редактируемой или нередатируемой;

- **Cell Evaluatable** – делает ячейку эволюционирующей или нет;
- **Cell Active** – делает ячейку активной или нет;
- **Initialization Cell** – делает ячейку инициализированной или нет.

Установка свойств выделенной ячейки осуществляется установкой (как обычно мышкой) символа птички против той или иной позиции подменю **Cell Properties** или исполнением команды в этой позиции. Одновременно может быть установлено несколько свойств. Для удаления свойства необходимо снять знак птички.

Ячейка ввода и соответствующая ей ячейка вывода обрамляются не только своими удлинненными квадратными скобками справа, но и общей скобкой. Активизируя эту скобку быстрым двойным нажатием левой клавиши мыши (при наведенном на скобку курсоре мыши), можно скрывать и выводить на экран выходную ячейку. Скрывать последнюю полезно, если содержащийся в ней результат слишком громоздкий.

Интересно отметить, что редактировать можно не только входные, но и выходные ячейки, например, вручную задавая более приемлемый вид результата. Однако для этого надо выходную ячейку сделать редактируемой – свойство **CellEditable**. Редактируемая ячейка имеет символ «X» у своей обрамляющей скобки.

Ячейки могут быть эволюционирующими (иначе говоря, исполняемыми) или нет, что задается командой **Cell Evaluatable**. Только эволюционирующие ячейки исполняются ядром системы и порождают выход. Такие ячейки помечаются знаком «-» в обрамляющей их правой скобке.

Эволюция начинается, как только происходит оценивание статуса какой-либо ячейки. От пользователя зависит, какие ячейки и в каких сочетаниях оцениваются при исполнении документа и начинают эволюцию. Можно выполнить, например, выделение ячеек так, чтобы они эволюционировали только совместно, но не индивидуально.

Ячейки также могут быть активными и неактивными. Изменение активности достигается командой **Cell Active**. Активная ячейка помечается в скобке знаком «A» и обычно управляется кнопкой.

Наконец, ячейки могут быть инициализированными и нет – команда **Initialization Cell**. Инициализированная ячейка помечается в скобке знаком «|» и автоматически исполняется при загрузке документа, содержащего такую ячейку (или ряд ячеек).

Команда **Group Cells** используется для объединения ряда ячеек в одну группу. Объединяемые ячейки нужно вначале выделить (рис. 1.8), а затем использовать команду объединения.

При этом выбранные ячейки обрамляются общей для них длинной квадратной скобкой (рис. 1.9). Активизация этой черты позволяет управлять просмотром ячеек.

Активизируя черту для всех ячеек, можно получить скрытый блок ячеек, в качестве названия которого выступает первая ячейка (рис. 1.10). Таким образом, можно поочередно то открывать, то закрывать блок ячеек. Заметим, что закрытые ячейки выполняются в соответствии с их статусом (свойствами).

Команда **Ungroup Cells** разъединяет объединенные в группу ячейки. Если при этом в группе есть ячейки, объединенные в более мелкие подгруппы, то они сохра-

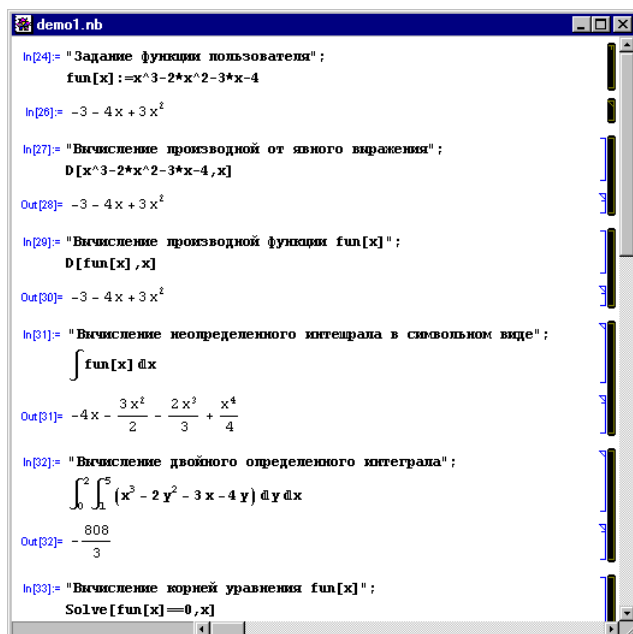


Рис. 1.8. Выделение ячеек документа перед их объединением

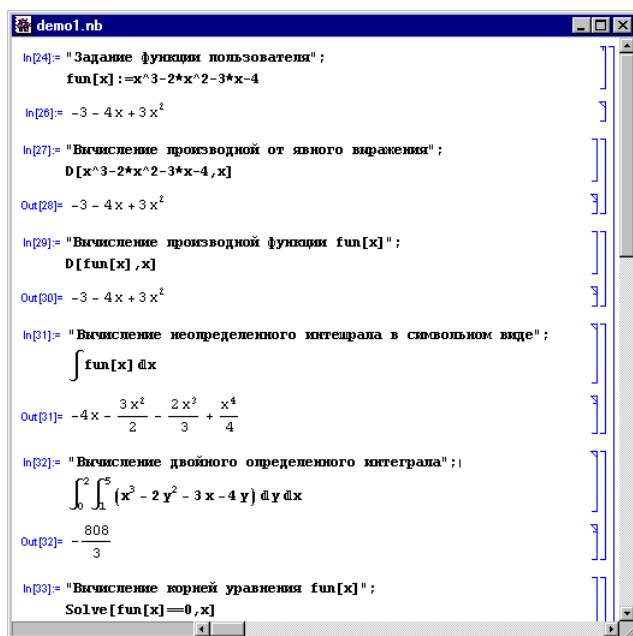


Рис. 1.9. Документ после объединения ячеек в группу

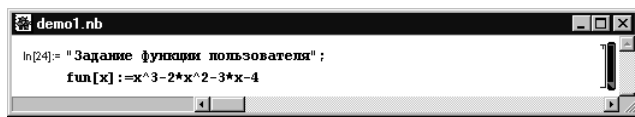


Рис. 1.10. Документ со скрытым блоком ячеек

няются. Для деления ячейки на части используется команда **Divide Cell**, а для объединения двух ячеек используется команда **Merge Cells**.

Команда **Open All Subgroups** открывает все отмеченные группы и подгруппы ячеек, а команда **Close All Subgroups** закрывает все выделенные группы и подгруппы ячеек.

Действие команды **Open/Close Group** уже описывалось – она сокращает число ячеек в группе так, что видимой остается только первая ячейка, как правило, имеющая титульную надпись.

Таким образом, команды управления статусом и объединением ячеек позволяют создавать довольно сложные структуры электронных документов с многочисленными открывающимися и закрывающимися вложениями. Такие документы удобны для создания полноценных электронных уроков и даже электронных книг. С системой поставляется множество примеров таких уроков, с которыми стоит познакомиться.

1.4.4. Работа с графическими и звуковыми возможностями

Система Mathematica обладает превосходными графическими возможностями – от построения двумерных и трехмерных графиков до синтеза сложных изображений (например, цветных карт) и динамически изменяющихся математических поверхностей. Эти возможности задаются встроенными в ядро графическими функциями и расширяются средствами пакетов AddOn.

Каждая графическая функция в ответ на обращение к ней возвращает *графический объект* – тот или иной рисунок. Именно поэтому в системе Mathematica для построения графиков используются функции, а не операторы, как в большинстве языков программирования. Это говорит о том, что понятие функции в данной системе существенно расширено.

Возвращаемый графической функцией объект представлен ячейкой с соответствующим графиком. Ряд параметров такого объекта, например: размеры графика, используемые цвета, толщина линий и т.д. – задаются по умолчанию. Помимо указанных параметров в их список могут включаться специальные опции и директивы, расширяющие возможности графики. С их помощью можно управлять выводом координатных осей и текстовых надписей, менять размеры графика, строить графики типовых геометрических фигур и т.д. Эти возможности мы рассмотрим позднее.

В подменю позиции **Cell** главного меню можно найти ряд команд, относящихся только к ячейкам вывода с графическими и звуковыми объектами. Укажем эти команды:

- **Animate Selected Graphics Ctrl+Y** – анимация выделенной ячейки с графиком;
- **Play Sound** – воспроизведение синтезированного звука;
- **Rerender Graphics** – перепостроение графиков;
- **Rerender And Save Graphics** – перепостроение графиков и их запись;
- **Make Standard Size** – установка стандартного размера ячейки;
- **Align Selected Graphics...** – объединение выделенных графиков;
- **Cell Size Statistics...** – вывод статистики о размерах ячеек.

В общих чертах назначение этих команд очевидно. Команды, связанные с анимацией графиков и воспроизведением звука, мы рассмотрим более детально после описания графических и звуковых возможностей систем Mathematica.

1.5. Операции форматирования ячеек (Format)

Mathematica обладает обширными возможностями в форматировании ячеек ввода и вывода: изменении размеров и цвета символов, выбора набора шрифта, задании цвета фона, выделении ячеек и т.д.

1.5.1. Команды позиции *Format* главного меню

Средства форматирования сосредоточены в позиции **Format** главного меню. Это меню содержит множество позиций, дающих практически неограниченные средства по форматированию документов. Большинство из них обычному пользователю может никогда и не понадобиться – вполне достаточно установок, введенных по умолчанию. Однако, при решении специфических задач, например, при подготовке документов к полиграфическому изданию, наличие обширных средств форматирования становится далеко не лишним.

1.5.2. Изменение стиля документов

Позиция **Style** открывает меню стандартных стилей ячеек. Под стилем ячеек по существу называют совокупность их параметров, задающих вид ячеек. Прежде всего, это используемые наборы шрифтов, размеры символов, различные виды выделений и т.д.

На рис. 1.11 представлена серия ячеек ввода, отформатированная под все возможные стандартные стили. Все стили существенно отличаются друг от друга, что позволяет легко распознавать стиль различных ячеек. Самые распространенные из них – это текстовые ячейки разного стиля и ячейки ввода и вывода.

Рис. 1.11. Ячейки входа, отформатированные разным стилем

Следующие две команды позиции **Style** – это **ScreenStyleEnvironment** и **PrintStyleEnvironment**, которые служат для изменения текущего формата ячеек документа при его наблюдении на экране дисплея и при печати на принтере. Возможны следующие установки:

- **Working** – рабочий стиль (типичный);
- **Presentation** – презентационный (увеличенные размеры символов);
- **Condensed** – сжатый (уменьшенный размер символов);
- **Printout** – принтерный (оптимальный при печати).

Эти установки вполне очевидны, и пользователь может проверить их на том или ином ноутбуке.

1.5.3. Опции стилей и программ и их изменение

Команда **ShowExpression** служит для управления показом выражений в стандартном и развернутом виде. Например, при отключенной этой команде введем и исполним простое выражение:

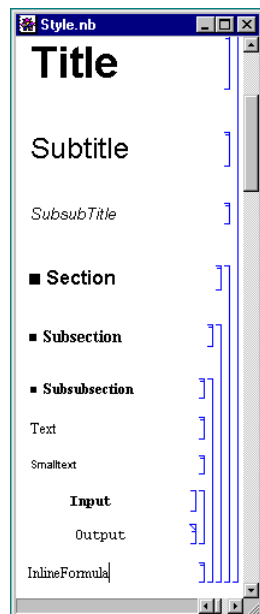
2*Log[3]/Exp[5]

$$\frac{2 \operatorname{Log}[3]}{e^5}$$

Здесь вид ячеек стандартный. А теперь, выделив эти ячейки и исполнив команду **ShowExpression (Shift+Ctrl+E)**, получим представление в развернутом формате:

```
Cell[BoxData[
  StyleBox[
    RowBox[{"2", "*",
      RowBox[{
        RowBox[{"Log", "[", "3", "]"}], "/",
        RowBox[{"Exp", "[", "5", "]"}]}],
    FormatType->StandardForm,
    FontFamily->"Courier New",
    FontSize->10]], "Input",
  CellLabel->"In[21]:="]
```

```
Cell[BoxData[
  FractionBox[
    RowBox[{"2", " ",
      RowBox[{"Log", "[", "3", "]"}]}],
```




```
SuperscriptBox["\[ExponentialE]", "5"], "Output",  
CellLabel->"Out[21]="]
```

Такой формат является внутренним в том смысле, что он характерен для внутреннего представления вывода на экран дисплея, принятого в языке программирования системы Mathematica. Словом, это типичная программа для вывода указанных выражений. Чем сложнее выражение, тем длиннее и непонятнее для непосвященных выглядит развернутое представление во внутреннем формате.

Читатель, вероятно, догадался, что наглядность представления информации на экране дисплея и при печати на принтере в системе Mathematica достигается дорогой ценой – каждую «приятную мелочь» приходится программировать, используя при этом функции и команды встроенного языка программирования системы. При этом часто используются опции – специальные указания, задающие объектам системы особые свойства. Опции обычно записываются в виде **Имя_Опции->Значение_Опции**. Даже в приведенном простом примере программы используются две опции.

Поспешим успокоить рядового пользователя системы Mathematica: опции задаются по умолчанию настолько удачно, что можно чаще всего вообще не вспоминать о них, работая с системой без программирования. Тем не менее, система позволяет контролировать и изменять опции, используемые в программах. Для этого служит специальный инспектор опций, запускаемый командой **Option Inspector (Shift+Ctrl+O)**.

Фактически инспектор опций обеспечивает визуально-ориентированное изменение программ в части, касающейся установок опций. Окно инспектора не только дает представление о многочисленных опциях в программах, но и обеспечивает легкое их изменение с целью решения особых задач представления информации. Еще раз отметим, что это нужно достаточно опытным пользователям и может не учитываться в начале работы с системой.

Команда **RemoveOptions...** убирает все опции, введенные пользователем, и восстанавливает исходное состояние системы. То, с которым и целесообразно работать в большинстве случаев.

1.5.4. Уточненное управление стилем документов

Целый ряд последующих команд служит для уточненного управления стилем документов:

- **Style Sheet** – выбор стиля документа из обширного стандартного набора стилей;
- **Edit Style Sheet** – вывод окна выбора и редактирование стиля документов;
- **Font** – установка типа шрифта из полного списка наборов шрифтов;
- **Face** – установка стиля символов (наклонный, жирный, подчеркнутый снизу);
- **Size** – установка размера символов;
- **Text Color** – установка цвета текста;

- **Background Color** – установка цвета фона;
- **Choose Font** – вывод окна для выбора и замены набора шрифта;
- **Text Alignment** – установка типа выравнивания (справа, слева, по центру) и полей;
- **Text Justification** – установка выключки строк;
- **Word Wrapping** – установка разворота слов;
- **Cell Dingbat** – установка меток ячеек из обширного списка (метки сохраняются при отключении номеров строк);
- **Horizontal Lines** – установка типа горизонтальной линии.

Каждая из этих команд выводит подменю с обширным списком установок для выбора соответствующего параметра. Поскольку эти параметры достаточно очевидны, мы не будем их обсуждать более подробно. Единственным исключением является команда **Choose Font**, которая вместо списка параметров выводит стандартное окно установки шрифтов. Работа с этим окном понятна любому пользователю, который имеет хотя бы минимальный опыт работы с программами класса Microsoft Office 95/97/NT/2000/XP.

1.5.5. Установка стиля интерфейса

В Mathematica 4/5 установка стиля интерфейса также отнесена в позицию **Format** главного меню. Для этого имеются четыре команды:

- **Show Ruller** – установка мерной линейки;
- **Show ToolBar** – установка панели инструментов;
- **Show Page Break** – установка линии обрыва страницы;
- **Magnitification** – установка (в процентах) размера элементов документа.

Все эти команды влияют на вид окна документа, а не панели главного меню.

1.6. Ввод элементов документов (Input)

Позиция главного меню **Input** открывает подменю с рядом описанных ниже команд. Все они объединены под термином **Input** (Ввод), хотя принятым для ряда этих команд является термин **Insert** (Вставка). Ряд команд подменю **Input** создан не без претензий на новизну, что на практике оборачивается лишь усложнением их применения. За исключением вставок координат двумерных графиков, использования инспектора обзора трехмерных графиков и применения гиперссылок, остальные виды ввода обычный пользователь может и не использовать.

1.6.1. Ввод координат двумерных графиков

Иногда бывает нужно знать координаты точек двумерных графиков. Например, это полезно для решения нелинейных уравнений с целью уточнения корней функции, график которой был построен. Mathematica имеет довольно своеобразную возможность определения координат произвольной точки графика и даже ряда точек. Они выводятся окном, которое появляется при исполнении команды **Get**

Graphics Coordinates.... Для получения координат двумерных графиков прежде всего необходимо выделить их. Затем следует нажать и удерживать клавишу **Ctrl** и поместить маркер мыши вблизи нужной точки графика. При этом в левой нижней части окна появится список с координатами точки.

Можно повторить определение координат для ряда точек. Затем следует воспользоваться командой **Copy** для переноса координат точек в буфер и, исполнив команду **Paste**, можно перенести список с координатами точек в текущую строку ввода. Удобно это делать, используя команды контекстно-зависимого меню правой кнопкой мыши.

1.6.2. Работа с селектором обзора трехмерных графиков

Команда **3D View Point Selector ... (Shift+Ctrl+V)** служит для вывода селектора обзора трехмерных графиков (рис. 1.11). Это следует делать при наличии в документе трехмерного графика.

В этом окне имеется изображение куба в пространстве, который можно вращать с помощью мыши или перемещением ползунков в линейках прокрутки, задающих параметры просмотра и перспективы объекта (увы, сам объект при этом не виден). Для задания поворота рекомендуется окно с самой фигурой разместить рядом (на рис. 1.12 оно показано справа).

В левой части окна вращения имеется ряд кнопок:

- **Close Dialog** – завершение диалога;
- **Cancel** – прекращение работы;
- **Paste** – перенос указания о вращении в окно с фигурой;
- **Default** – задание исходного значения параметров вращения и перспективы;
- **Help** – вызов справки.

Действие всех кнопок вполне очевидно. Поэтому остановимся на главном. Нажатие кнопки **Paste** создает строку с опцией **ViewPoint[{x,y,z}]**, которая помещается на место расположения маркера ввода в окне редактирования документа. В нашем случае маркер ввода необходимо расположить в строке функции **Show[g1,g2]** после запятой, предварительно установленной вслед за g1. Если теперь исполнить модифицированную функцию **Show**, то рисунок будет перестроен (см. рис. 1.13).

Разумеется, такая процедура поворота фигуры в пространстве не очень удобна. При очередном повороте необходимо будет отредактировать строку с функцией **Show**, убрав из нее старую запись опции **ViewPoint** и вставив новую запись.

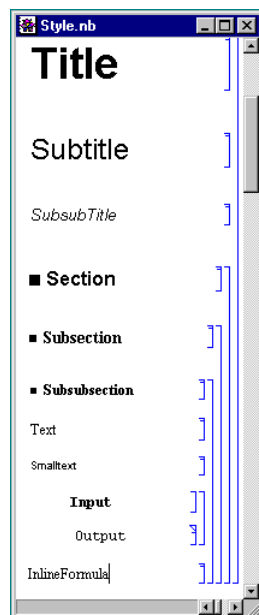


Рис. 1.11. Селектор обзора 3D графиков (слева)

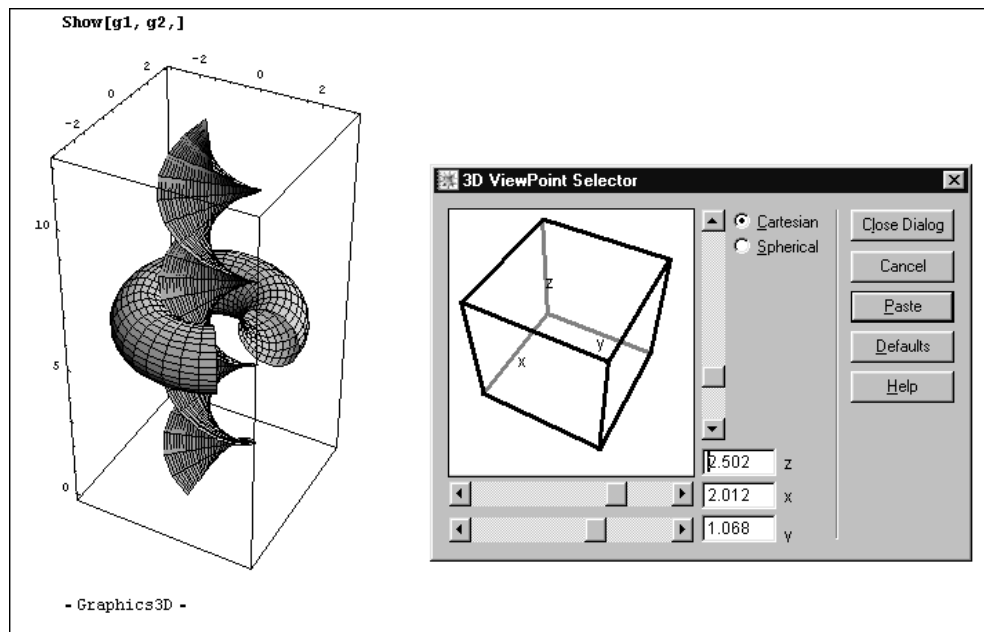


Рис. 1.12. Пример разворота трехмерной фигуры

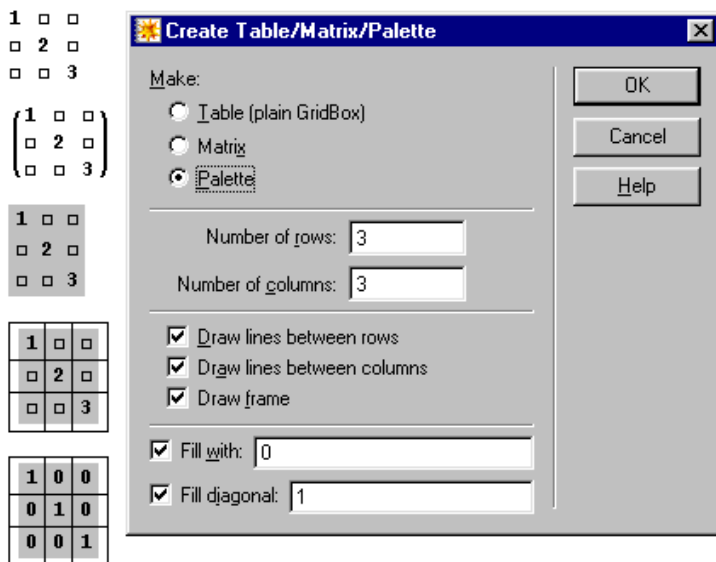


Рис. 1.13. Работа с окном Create Table/Matrix/Palette

Между тем, в ряде систем компьютерной математики, например в Maple и Mathcad, уже давно появилось новое мощное средство редактирования изображений 3D-объектов – их вращение мышью в реальном масштабе времени. Подобное средство введено и в системы Mathematica 4/5, но реализуется пакетом расширения RealTime3D.

1.6.3. Изменение цветовой гаммы

Команда **Color Selector...** выводит стандартное окно изменения цветовой гаммы, используемой при функциональной окраске графиков. Это типовое окно системы Windows. С его помощью можно создать дополнительные цвета и изменить гамму цветов линий рисунков и заливки.

Не рекомендуется пользоваться селектором цветов без особой на то надобности. Практика показывает, что исходные установки цветов выбраны очень тщательно с учетом особенностей нашего зрения. Обычно после изменения установок цвета пользователь быстро осознает, что его цветовые «новации» лишь испортили цветовую гамму рисунков, и возвращается к исходной установке цветов.

1.6.4. Работа с фонографом

Команда **Record Sound...** выводит окно **Фонографа**, входящего в состав операционной системы Windows. Фонограф – это специальное приложение, позволяющее записывать звуки с микрофона и воспроизводить их с помощью звуковой карты ПК. Органы управления фонографа по виду подобны применяемым у обычных магнитофонов, а потому не нуждаются в особом описании.

Особенности работы со звуком будут описаны в дальнейшем. Отметим лишь, что Mathematica имеет возможность работы как математическим синтезом звуковых сигналов, так и с реальными звуковыми сигналами речи и музыки, записываемыми с помощью фонографа в виде файлов с расширением .wav. Для этого и служит данная команда.

1.6.5. Вставка файла

В документ могут быть вставлены различные данные, хранящиеся в файлах различных форматах, например текстовых или графических. Команда **Get File Path...** открывает окно загрузки файлов. Это то же окно, что и у команды **Open...**, которое уже описывалось. К этому описанию нечего добавить.

1.6.6. Ввод таблиц, матриц и палитр

Задание таблиц и матриц в системе Mathematica легко выполняется с помощью соответствующих функций. Однако, команда **Create Table/Matrix/Palette...** дает такую возможность и с главного меню. Она выводит окно задания таблиц, матриц

и палитр, показанное на рис. 1.13 справа. В левой части документа показаны примеры работы с данным окном. Эта работа вполне очевидна.

1.6.7. Ввод и редактирование кнопок

При создании сложных документов для диалогового режима работы с системой Mathematica иногда полезно создание кнопок. Оно выполняется командой **Create Button**. Она выводит подменю, содержащее небольшое число типов кнопок. Например, кнопка типа **Evaluate Cell** служит для создания исполняющей вычисления ячейки.

Редактирование кнопок осуществляется по команде **EditButton**. Она выводит окно редактирования кнопок. В этом окне содержится перечень кнопок и окно с программой, создающей кнопку с нужными свойствами.

1.6.8. Вставка гиперссылки

Гиперссылка является объектом класса **ButtonBox** (кнопка), связанным с некоторым другим объектом, представленным файлом, например, какого-либо документа или рисунка. При активизации гиперссылки мышью связанный с ней объект загружается.

Гиперссылка создается следующим образом. В строке ввода готовится некоторый текст, например, фраза: «Просмотр документа d1.nb». Какое-либо слово или фраза выделяются с помощью мыши, и выполняется команда **Create Hyperlink** (Создать Гиперссылку). Открывается окно, показанное на рис. 1.14 в правой части экрана.

Следующий этап заключается в установке связи гиперссылки с нужным файлом. Его полное имя можно прямо указать в верхнем поле над кнопкой **Browse....** Однако чаще всего пользователь не помнит полного имени файла. Тогда он может воспользоваться кнопкой обзора файловой системы **Browse...**, которая выводит стандартное окно поиска файлов, показанное на рис. 1.14 слева. В этом окне необходимо найти нужный файл (в нашем случае это файл документа d1.nb) и нажать клавишу **Открыть**. Имя файла появится в поле окна создания гиперссылки, и для ее получения достаточно нажать кнопку **ОК**.

Выделенное слово (фраза) превратится в кнопку, подчеркнутую чертой. Это и есть гиперссылка. Не следует путать ее с гипертекстовой ссылкой, которая представляет собой подчеркнутое слово или фразу. В Mathematica гиперссылка – объект класса **ButtonBox**, что уже отмечалось. Активизация гиперссылки вызовет немедленное появление документа, представленного в нашем примере файлом d1.nb (рис. 1.15).

Гиперссылки обычно применяются для создания сложных документов с многочисленными связями друг с другом. На основе гиперссылок создаются документы, широко используемые в мировой сети Internet.

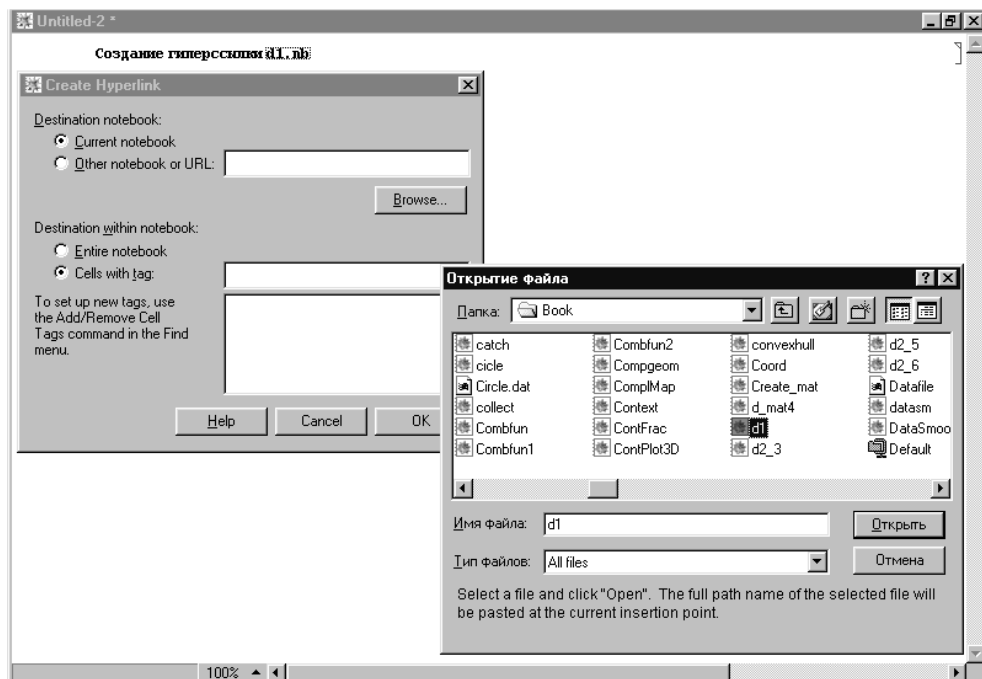


Рис. 1.14. Создание гиперссылки

1.6.9. Создание и ввод специальных объектов

Еще одна редко используемая возможность – создание объекта, которому присвоен номер. При этом номер генерируется автоматически. Для этого служит команда **Create Automatic Numbering Object....** Она создает простое окно, которое содержит переключатель, выводящий обширный перечень объектов, которые создаются.

В дальнейшем мы не будем пользоваться объектами данного типа, так что ограничимся приведенным выше описанием и предоставим пользователю самому поэкспериментировать с такими объектами. Предоставим читателю самостоятельно разобраться и с еще одной редко используемой возможностью – вставкой объектов с помощью команды **Create Value Display Object....** Она также выводит окно для задания свойств таких объектов.

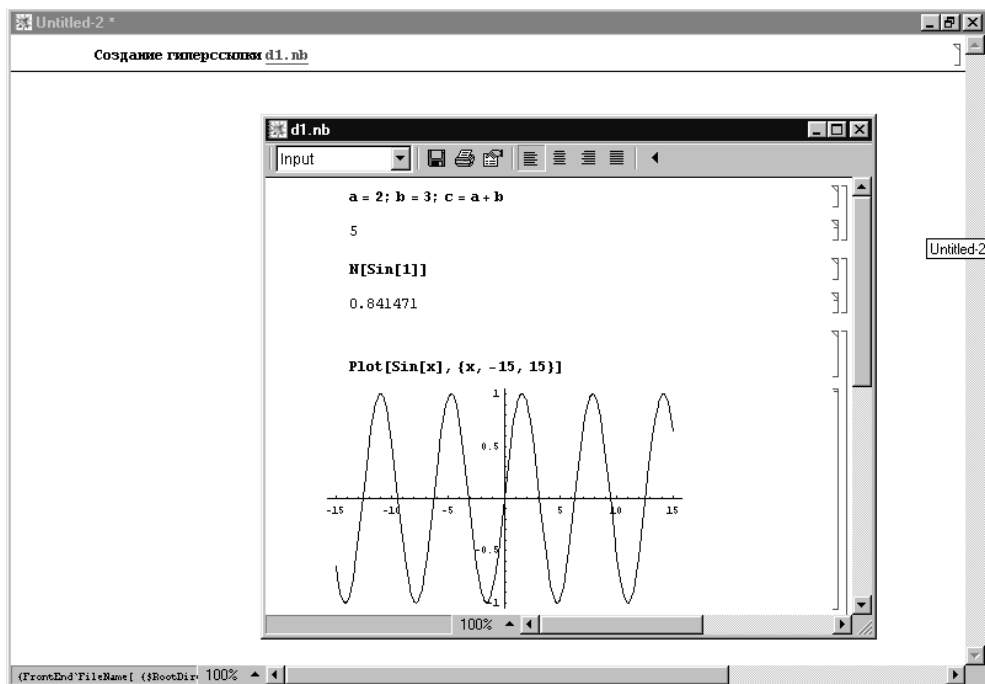


Рис. 1.15. Пример использования гиперссылки

1.6.10. Вставки, связанные с ячейками

Для вставки содержимого предшествующих ячеек ввода и вывода служат команды **Copy Input from Above** и **Copy Output from Above**. Поясним это на примерах. Введем в ячейку ввода выражение

1+2

Нажав клавиши **Shift+Enter**, получим строку вывода:

3

Теперь, исполнив команду **Copy Input from Above**, получим в новой строке ввода:

1+2

Ее исполнение даст

3

А исполнение команды **Copy Output from Above** даст

3

Еще одна команда **Start New Cell Bellow** служит для вставки новых пустых ячеек ввода между уже имеющимися. Ячейка вставляется ниже положения маркера мыши, указывающего место вставки.

1.6.11. Вставки имен функций и списков их параметров

Запомнить около тысячи функций, входящих в ядро систем Mathematica, довольно сложно, также как и правила их записи. Для облегчения этого служат две заключительные команды подменю **Input**.

Первая из них работает, если маркер мыши находится на каком-либо ключевом слове в строке ввода. Тогда исполнение команды **Complete Selection** (**Ctrl+K**) выводит список имен всех функций, которые содержат данное слово. Рисунок 1.16 поясняет это на примере ввода слова **Plot**.

Соответственно команда **Make Template** в указанной маркером функции выдаст список ее параметров. Например, если введено слово

Plot|

и маркер ввода стоит после него, то команда **Make Template** приведет к следующему изменению строки ввода:

Plot[f, x, xmin, xmax].

Теперь ясно, какие параметры имеет эта функция, и редактированием строки ввода можно ввести нужные конкретные значения этих параметров.

Plot

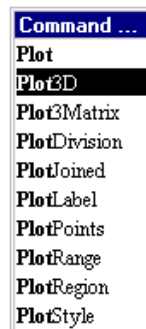


Рис. 1.16. Пример исполнения команды **Complete Selection**

1.7. Управление работой ядра системы (Kernel)

1.7.1. Команды позиции Kernel главного меню

Позиция главного меню **Kernel** служит для управления акциями (действиями) системы, проводимыми ядром системы над ячейками загруженного документа. Рассмотрим команды этого подменю более подробно.

1.7.2. Управление процессом вычислений

Основные команды по управлению процессом вычислений сосредоточены в подменю **Evaluation** меню **Kernel**:

- **Evaluate Cells** **Shift+Enter** – исполнение выбранной ячейки;
- **Evaluate Place** **Shift+Ctrl+Enter** – исполнение выделенного выражения по месту;
- **Evaluate Next Input** **Ctrl+Enter** – исполнение следующей строки ввода;

- **Evaluate Subsection** – исполнение субсекции документа;
- **Evaluate Notebook** – исполнение всего документа;
- **Evaluate Initialization** – исполнение инициализированных ячеек;
- **Enter Subsection** – ввод субсекции;
- **Exit Subsection** – удаление субсекции.

Данная группа команд управляет эволюцией (исполнением) ячеек. Перед исполнением каждая ячейка оценивается по своим признакам. Команда **Evaluation Cells** оценивает все выделенные ячейки, вызывает их эволюцию и помещает результат эволюции каждой ячейки сразу после нее. Это одна из наиболее распространенных команд. Следует помнить, что, казалось бы, естественное нажатие клавиши **Enter** вызывает лишь прерывание строки, а не эволюцию выделенных ячеек ввода. При управлении с клавишного пульта эволюция выбранных ячеек происходит при одновременном нажатии клавиш **Shift** и **Enter**.

Эта команда может быть исполнена также при нажатии клавиши 5 блока цифровых клавиш в правой части клавишного пульта (но не клавиши 5 в главном блоке клавиш). Перед этой командой должно быть загружено ядро системы. Если этого не было, может пройти определенное время для загрузки ядра, прежде чем данная команда будет исполнена.

Особое внимание следует обратить на команду **Evaluate Place**. Допустим, вы ввели в ячейку ввода выражение:

(2+3) / 7

Выделите мышью выражение (2+3). Теперь, исполнив эту команду нажатием клавиш **Ctrl+Shift+Enter** в строке ввода, получите:

5 / 7

Таким образом, выражение (2+3) было вычислено прямо в строке ввода, и не его месте появился результат – 5. Если теперь исполнить команду **Evaluate Cells**, то появится строка вывода с результатом:

$\frac{5}{7}$

Поскольку результат представлен дробно-рациональным числом, он повторяет выражение в строке ввода, но в ином формате – вывода.

Команда **Evaluate Next Input** позволяет последовательно исполнить ряд ячеек, расположенных под выделенной ячейкой. После исполнения выделенной ячейки она перемещает выделение на следующую ячейку и оценивает ее (но вначале не исполняет). Последующее использование команды ведет к эволюции этой ячейки и так далее. Таким образом, можно последовательно вызывать оценивание и эволюцию каждой ячейки документа, используя дважды эту команду для каждой следующей ячейки.

Команда **Evaluate Notebook** оценивает все ячейки введенного документа сверху вниз. Это особенно полезно, если результаты эволюции нижестоящих ячеек зависят от результатов эволюции вышестоящих ячеек. При этом все ячейки переоцениваются, т.е. выполняются заново с учетом всех возможных изменений их содержимого. Это напоминает работу с электронными таблицами, когда смена

численного значения в одной ячейке автоматически меняет содержание всех других ячеек, использующих данные из этой ячейки.

Команда **Evaluate Initialization** выбирает и оценивает все ячейки, имеющие признак инициализации – символ «т» над квадратной скобкой, обрамляющей ячейку. О задании такого признака говорилось выше. Ячейки с указанным признаком выполняются этой командой без их выделения.

Следующие две команды меню **Kernel** управляют процессом текущих вычислений:

- **Interrupt** – **Alt+**, – прерывание вычислений;
- **Abort** – **Alt+.** – полное прекращение вычислений.

Их действие вполне очевидно. Команда **Interrupt** служит для прерывания текущих вычислений. Эта команда при исполнении задает запрос о том, каким образом вы хотите прервать вычисления и сколько шагов вычислений необходимо еще сделать. Разумеется, можно и отменить прерывание. Команда **Abort** дает полное прекращение вычислений, и их можно возобновить лишь с самого начала. Вместо результата выдается сообщение \$Abort.

Полезно запомнить исполнение этих команд набором клавиш, поскольку «зависание» системы из-за чрезмерно большого времени исполнения неудачного алгоритма (например, глубокой рекурсии) нередкость. Кстати, в процессе таких вычислений доступны команды прерывания и из меню.

1.7.3. Выбор ядра системы

Новые версии Mathematica приобрели возможность работы не только с установленным локальным ядром, но и с другими ядрами, ориентированными на какие-либо специфические классы вычислений. Это привело к появлению ряда новых команд:

- **Start Kernel** – задает старт выбранного ядра;
- **Quit Kernel** – задает выход из выбранного ядра;
- **Default Kernel** – задает ядро, используемое по умолчанию;
- **Notebook's Kernel** – задает ядро данного документа;
- **Kernel Configuration Options** – выводит окно установки свойств ядер.

Пока, однако, система поступает с одним ядром (Local), поэтому выбора при работе с этими командами нет. Они введены с расчетом на будущее расширение системы.

1.7.4. Управление показом номеров ячеек

Номера строк ввода и вывода – причуда системы, унаследованная от доброго старого Бейсика. В принципе, нумерация строк при культурном программировании в системе Mathematica не нужна и даже вредна. В частности, нумерация не является строго последовательной и нарушается при изменении ячеек ввода в начале документа и их исполнении после редактирования. В общем случае она не совпадает для записываемого и считываемого документов.

Поэтому предусмотрена команда **Show In/Out Number**. Она управляет показом или скрытием номеров строк. Если против команды установить знак птички, то номера строк будут показаны в виде *In[n]* и *Out[n]*. При отсутствии знака птички номера строк вместе с их опознавателями ввода и вывода не показываются.

1.7.5. Удаление всех ячеек вывода

Иногда желательно удалить все ячейки вывода, например, для того, чтобы при объединении в группы они не превратились в элементы ячеек ввода. Для удаления всех ячеек вывода служит команда-переключатель **Delete All Output**. Если на ней установить знак птички, то в текущем документе будут удалены все ячейки вывода.

1.8. Операции поиска и замены

1.8.1. Обзор подменю *Find*

Позиции **Find** главного меню выводит подменю с операциями поиска и замены фрагментов текстов и выражений. Эти операции характерны для любого текстового процессора, например Microsoft Word, и потому знакомы даже начинающим пользователям. Поэтому ограничимся их кратким описанием.

1.8.2. Команды поиска и замены

Первая группа команд в позиции **Find** реализует типичные операции поиска и замены:

- **Find Ctrl+F** – поиск заданных строк вперед и назад (но без замены);
- **Enter Selection** – ввод в окно поиска выделенной строки;
- **Find Next F3** – поиск по документу вперед (вниз);
- **Find Previous** – поиск по документу назад (вверх);
- **Find in Select Tags** – поиск в выделенных ячейках этикеток;
- **Replace** – замена одной строки на заданную другую;
- **Replace and Find Again** – выполнение замены с продолжением поиска;
- **Replace All** – выполнение замены по всему документу.

Эти операции выполняются с помощью обычного окна поиска и замены. Оно имеет поля для задания искомой строки и строки замены. Работа с этим окном очевидна.

1.8.3. Обнаружение и открытие выделенных строк

Следующие две команды в подменю **Find** служат для работы с выделенными строками:

- **Open Selected** – открытие выделенных строк;
- **Scroll to Selection** – прокрутка документа до выделенной строки.

Их действие также очевидно.

1.8.4. Работа с этикетками

Особым признаком ячеек ввода могут быть их **этикетки** (tags) – короткие сообщения, характеризующие суть выполняемых ячейками действий и размещаемые сверху строки ввода. Этикетки вводятся для того, чтобы можно было одновременно вызвать на просмотр те ячейки, которые объединены какими-либо общими свойствами.

Признаком наличия у данного документа этикеток является их список, который появляется у позиции **Cell Tags** (ячейки с этикетками) подменю **Find**. Команда **Add/Remove Cell Tags... (Ctrl+J)** позволяет вставить этикетку в строку ввода, в которой ее нет, или удалить этикетку из строки, где она есть. Эта команда вызывает появление окна редактирования этикеток. Работа с этим окном вполне очевидна – кнопка **Add** добавляет этикетку, а кнопка **Remove** удаляет ее. Команда **Cell Tag from In/Out Name** позволяет добавить в список этикеток заголовок выделенной строки ввода или вывода.

Последняя команда подменю **Find – Make Index** помещает в буфер все этикетки текущего документа. Перед этим она выводит окно, в котором можно указать признаки этикеток. Нажатие клавиши **OK** помещает список этикеток в буфер, откуда его можно извлечь с помощью команды **Paste**.

1.9. Управление окнами (Windows)

1.9.1. Команды позиции *Windows* главного меню

Система Mathematica многооконная и может сразу работать с рядом документов. По мере загрузки файлов их список появляется внизу подменю позиции **Window** (Окно). Само подменю содержит следующие команды для работы с окнами:

- **Cascade Windows** – каскадное расположение окон;
- **Tile Windows Wide** – фронтальное расположение с равной шириной окон;
- **The Windows Tall** – фронтальное расположение с равной высотой окон;
- **Messages** – управление выводом окна сообщений об ошибках.

За каждым загруженным документом закрепляется свое окошко и своя пиктограмма. Обычно в главном окне редактирования виден лишь последний документ. Однако позиция главного меню **Window** позволяет наблюдать за различным расположением окон, наиболее приемлемым для пользователя.

1.9.2. Управление расположением и вывод специальных окон

Если задать команду **Cascade Windows**, то окна будут расположены каскадно, т.е. одно за другим. При каскадном расположении окон на переднем плане находится окно с текущим, загруженным последним, документом. Оно заслоняет другие окна, но так, что их первая строка с титульной надписью видна. Две другие ко-

манды **Tile Windows Wide** и **Tile Windows Tall** обеспечивают фронтальное расположение окон по горизонтали и по вертикали.

Команда **Messages** выводит отдельное окно с сообщениями, а команда **Startup Pallete** управляет выводом окна с этим же названием (в правом нижнем углу рис. 1.1). Под этими командами располагается список ноутбуков, загруженных в систему Mathematica. Активизируя мышью то или иное имя ноутбука, можно сделать соответствующий ноутбук работающим в данное время, т.е. текущим.

1.10. Работа с информационными ресурсами системы Mathematica

1.10.1. Справка по системе Mathematica 5

В Mathematica встроена довольно мощная справочная база данных. Справка данных позволяет уточнить назначение любой функции, оператора или служебного слова системы и постепенно знакомиться с ее возможностями. Однако она не претендует на роль обучающей системы и неудобна для знакомства с системой. Для этого удобнее пользоваться обычными книгами.

В то же время, если вы уже сидите за компьютером, то пользоваться книгами не очень удобно. Куда проще тут же посмотреть нужные сведения по встроенной в Mathematica справочной системе. К тому же, в отличие от книг, она содержит «живые» примеры, которые можно быстро переименовать на свой лад.

Таким образом, можно сделать вывод, что справочная база данных систем Mathematica ориентирована, прежде всего, на получение оперативной справки по той или иной функции или команде, либо по некоторому элементу интерфейса. В общем, пока книги и электронная справочная система прекрасно уживаются друг с другом.

1.10.2. Открытие справочной базы данных Mathematica 5.2

Окно справки системы Mathematica 5.2, показанное на рис. 1.17, похоже на окно справки системы Mathematica 4. Более удобным стал просмотр окон справки: для этого сверху размещены кнопки переключения страниц справки. Нижняя граница области контекстного меню может перемещаться мышью, уменьшая или увеличивая высоту этой области.

Но главное отличие заключается в изменении и некотором расширении состава разделов окна справки:

- **Build-in Functions** – встроенные функции;
- **AddOn&Links** – пакеты расширений и установка связи с ними;
- **Front End** – интерфейс пользователя;
- **The Mathematica Book** – математический справочник (книга С. Вольфрама);

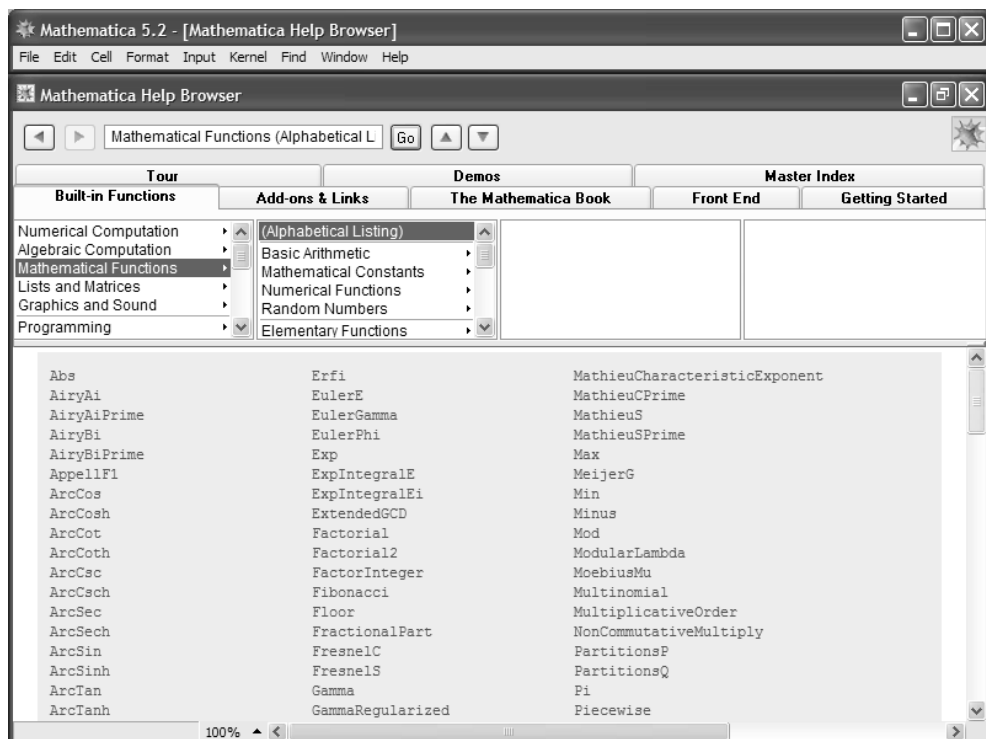


Рис. 1.17. Окно справки Mathematica 5.2

- **Getting Started** – начало работы с системой;
- **Demos** – демонстрационные примеры;
- **Tour** – обучающий раздел;
- **Master Index** – справка по индексу (алфавитный каталог).

1.10.3. Работа со справкой Mathematica 5.1/5.2

Работа со справкой Mathematica 5.1/5.2 вполне очевидна. Как недостаток можно указать некоторую сложность поиска нужного раздела справки, поскольку перечень разделов справки узок, и пользователь должен знать, что он ищет и в каком разделе справки находятся нужные ему сведения. На рис. 1.17 в качестве примера приведена страница справки со списком всех функций системы. Активизируя название нужной функции, можно получить детальную справку по ней и по ее применению.

В разделе **Demos** справки Mathematica 5.2 имеется множество интересных и поучительных примеров применения этой системы. Для демонстрации возможностей системы служит электронный учебник Tour. Электронный учебник содер-

жит множество полезных применений системы Mathematica. Однако в целом он рассчитан на начальный уровень знакомства с системой. Учебник представляет материал по контексту.

Mathematica 5.2 имеет еще один довольно эффектный учебник «Ten Minute Tutorial – знакомство с системой за десять минут». Это рекламное название не стоит принимать за «чистую монету». Но несомненно, что, просмотрев этот учебник, можно получить общее впечатление о возможностях системы Mathematica 5.1/5/1.

В систему Mathematica встроен ряд дополнительных пакетов расширения (в оригинале дополнения) AddOn, содержащих массу полезных новых функций. Они служат для расширения функциональных возможностей системы в таких областях, как алгебра, геометрия, приближенные вычисления, дискретная математика, теория чисел, математическая статистика, линейная алгебра и др.

Доступ к ним возможен объявлением соответствующего пакета. На рис.1.18 показан раздел справки по применению одного из таких пакетов для прямого и обратного Фурье-преобразований.

Доступ к справке по пакетам расширения обеспечивается разделом AddOn справочной системы. Любой пример применения из пакета расширения также можно перенести в документ. Для этого нужно выделить соответствующую ячей-

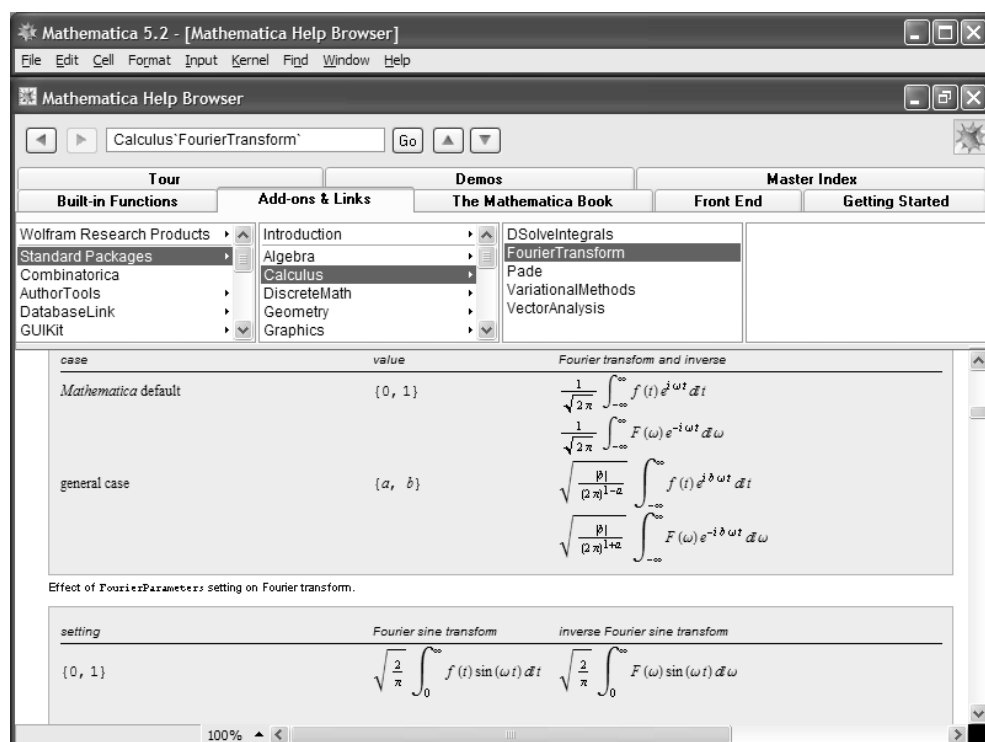


Рис. 1.18. Пример справки по преобразованию Фурье

ку примера, и с помощью команды **Copy** перенести в буфер промежуточного хранения. Затем с помощью команды **Paste** можно перенести содержимое буфера в документ. Для этого можно использовать и меню, которое создается при нажатии правой клавиши мышки.

Электронная книга «The Mathematica Book» – это электронный вариант книги S. Wolframa по соответствующей версии системы, который входит в справку системы Mathematica. С первого взгляда трудно уловить отличие электронной книги «The Mathematica Book» (Математическая книга) от пакетов расширения системы. Однако эти различия есть и заключаются в следующем:

- книга содержит большой объем чисто справочной информации (формулы, графики, примеры вычислений и т.д.);
- книга является систематическим руководством по применению системы;
- книга использует как встроенные, так и дополнительные функции из пакетов применений.

Электронная книга системы Mathematica являет собой наглядный пример развития электронных книг. Они характеризуются рядом новых качеств:

- красотой и наглядностью оформления;
- простотой поиска нужных сведений;
- возможностью применения действующих примеров;
- возможностью применения гипертекстовых ссылок;
- применением наглядных средств анимации изображений;
- объединением с другими программными средствами;
- легкостью модификации.

При всех этих очевидных достоинствах совершенно ясно, что в ближайшие годы подобные электронные книги не заменят обычные, поскольку их стоимость неизмеримо выше, чем стоимость обычных книг, а удобства работы с электронными книгами нивелируются необходимостью многочасового просиживания за экраном дисплея ПК. К тому же обычные книги написаны на исконно русском и понятном нам языке, тогда как подавляющее большинство электронных книг остаются англоязычными.

Последний раздел справочной системы – алфавитный (или индексный) указатель Master Index. Работа с ним вполне очевидна. Алфавитный указатель ценен тем, что в него входят все команды и функции, опции и примитивы, причем не только встроенные, но и входящие в состав пакетов расширений.

1.10.4. Другие команды меню Help

Помимо выше описанных команд меню **Help** система Mathematica 5 имеет еще ряд дополнительных команд:

- **About Mathematica...** – вывод окна с краткими данными о системе Mathematica 5 и фирме Wolfram Research Inc.;
- **Find Selected Function** – поиск указанных разделов в справочной системе;
- **Master Index...** – открытие окна справки с индексным (алфавитным) указателем;

- **Tutorial...** – открытие окна справки с учебным курсом;
- **Online Registration...** – онлайн-регистрация системы через Internet;
- **Information Center on the Web...** – доступ к Интернет-страницам информационного центра по системе Mathematica;
- **Wolfram Research on the Web...** – доступ к Интернет-сайту корпорации Wolfram Research, Inc.;
- **Rebuild Help Index** – создание индексного указателя (обычно требуется сразу после первого запуска системы);
- **Startup Palette ...** – управление выводом окна Startup Palette;
- **Why the Beep?...** – зарезервирована под выдачу информации о звуковых возможностях (обычно не задействована).

Действие этих команд очевидно и для многих команд оно уже было описано. Остается лишь отметить, что уверенное владение системой основано не только на освоении пользовательского интерфейса системы, но и многих ее команд и функций, а также при знании языка программирования системы.

1.11. Возможности системы Mathematica 5.2

1.11.1. Увеличение функциональности системы

В систему Mathematica 5.2 было введено более 60 новых функций и опций, а также модулей для быстрой и мощной обработки изображения. Введено также более 200 встроенных функций для обработки цвета и шкалы яркости изображений или других n -размерных сигналов. Эти новые функции обеспечивают преимущества новой версии Mathematica 5.2 в обработке реальных изображений.

Более важное значение имеет существенное повышение скорости вычислений. Вычислительным ядром Mathematica 5.2 теперь можно управлять на компьютере, который независим от пользователя и интерфейсной части. Это выгодно в случаях, когда доступен более мощный отдаленный компьютер. Повышена степень безопасности при работе с Mathematica 5.2.

1.11.2. Поддержка многоядерных микропроцессоров

В настоящее время ведущие производители микропроцессоров (корпорации Intel, AMD, IBM и др.) переходят на выпуск новых многоядерных микропроцессоров (multicore processors), реализующих методы параллельных вычислений, до этого применяемых лишь в супер-ЭВМ (в частности кластерных). Mathematica 5.2 стала первой СКМ, в некоторых вариантах которой впервые обеспечена как

технология уплотнения потоков Hyper Threading, так и поддержка возможностей новейших многоядерных процессоров. Эти преимущества реализованы в первую очередь при решении задач по линейной алгебре. В этой области проще всего достигается разбиение процессов вычислений на отдельные части, исполняемые отдельными ядрами процессора.

Интерфейсная часть Mathematica реализована как отдельный процесс, отделенный от процессов ее вычислительного ядра. Это создает диалоговый интерфейс даже тогда, когда многоядерный процессор находится под предельной нагрузкой. Даже при использовании двухъядерного процессора Mathematica 5.2 обеспечивает работу на разных ядрах интерфейсного модуля и вычислительного ядра (kernel). В итоге время «обдумывания», традиционно значительное для прежних версий Mathematica, сокращено примерно в 1000 раз.

На рис. 1.19 показано сравнение времен вычислений при решении четырех наиболее характерных задач линейной алгебры в СКМ Mathematica 5.2 на компьюте-

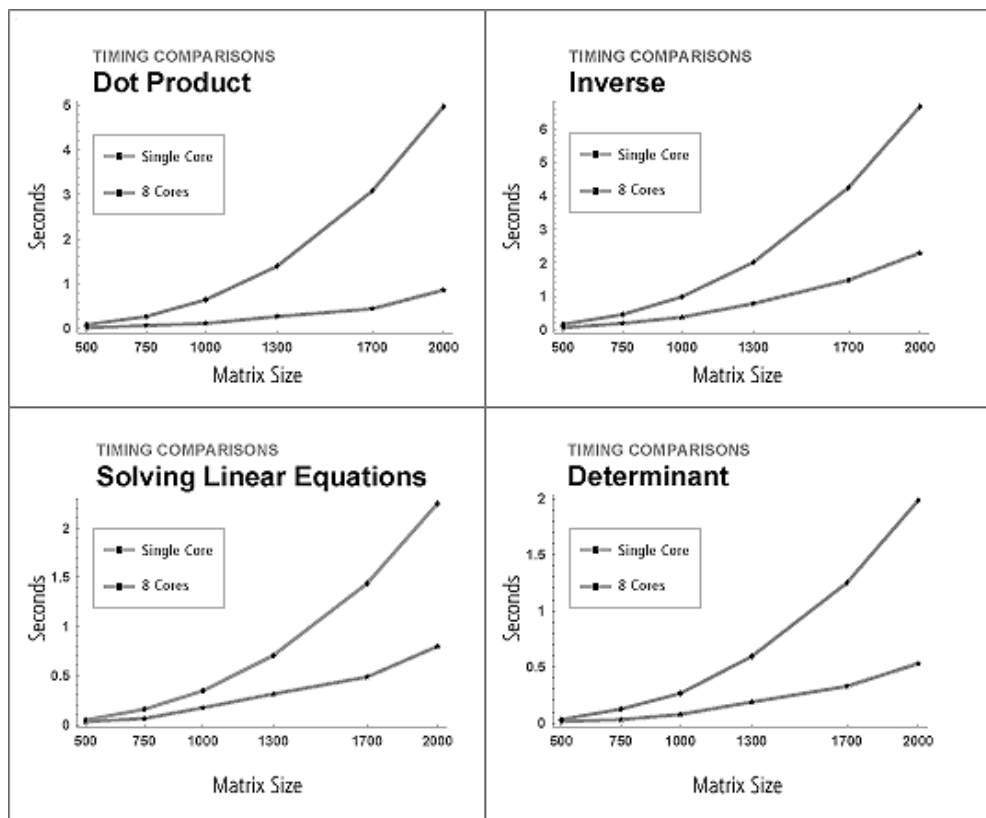


Рис. 1.19. Сравнительные результаты матричных вычислений в среде Mathematica 5.2 для ПК с одноядерным и 8-миядерным микропроцессорами

рах с одноядерным и 8-моядерным процессорами. Это вычисление точечного произведения, инвертирование матриц, решение систем линейных уравнений и вычисление определителя (детерминанта) матриц. Ускорение вычислений при работе с многоядерными процессорами достигает нескольких раз при решении задач с большой размерностью матриц.

1.11.3. Увеличение скорости вычисления математических функций

Алгоритмически параллельные вычисления в ряде СКМ реализованы в виде операции векторизации. При этом аргументами функций могут быть векторы и матрицы. Mathematica уже давно (начиная с версии 4.0) реализует специальную технологию упакованных массивов. Для этого были созданы специальные математические библиотеки, оптимизированные для каждого типа микропроцессора. Теперь на некоторых платформах эти библиотеки используют многоядерную технологию.

Это позволило резко повысить скорость вычисления большинства встроенных в ядро Mathematica 5.2 функций. Это особенно заметно при вычислениях, использующих операцию векторизации. На рис. 1.20 представлены данные о скорости вычисления ряда часто используемых функций. Здесь каждая функция применя-

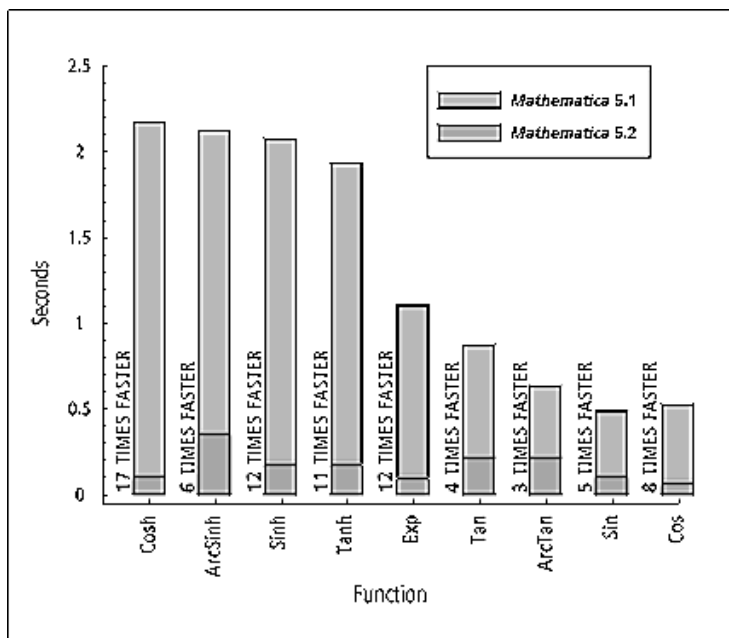


Рис. 1.20. Сравнение времен вычисления функций при использовании операции векторизации для версий Mathematica 5.2 и Mathematica 5.1

ется к вектору, имеющему 107 чисел в формате с плавающей точкой. Показано время вычисления для разных версий СКМ Mathematica и относительное ускорение вычислений.

1.11.4. Поддержка 64-разрядных микропроцессоров

Помимо всех основных 32-разрядных платформ Mathematica 5.2 поддерживает 64-разрядные платформы. Традиционно, 32-разрядные платформы были способны обеспечить однозначные адреса для менее чем 32 байтов (4.3 Гбайта). Как известно, уровень сложности и скорость выполнения аналитических вычислений сильно зависят от используемого объема памяти, и его повышение весьма актуально.

Новые операционные системы, например Linux и новейшие версии операционных систем от корпораций Microsoft и Apple, поддерживают уже 64-разрядную адресацию памяти. Это теоретически позволяет использовать до 64 байта, или примерно 18 000 000 000 Гбайт, хотя текущие аппаратные средства ЭВМ поддерживают более низкий предел. При установке Mathematica 5.2 автоматически определяется, версия какой разрядности должна быть установлена на данном ПК.

Высокая разрядность данных и адресов особенно важна при обработке сложных изображений. На рис. 1.21 показаны примеры обработки снимков прохождения цунами по подводным горам. Левое изображение (образ) было вычислено с почти максимальным использованием памяти на 32-разрядной системе памяти. Правое же изображение получено при использовании более высокого разрешения при 64-разрядном процессоре, что позволило избежать артефактов, которые (в виде «дыр») отчетливо заметны на левом изображении.

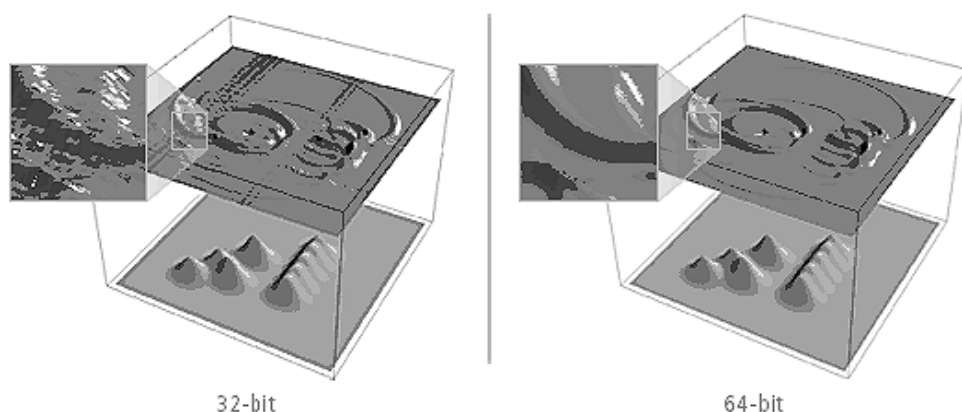


Рис. 1.21. Сравнение изображений (прохождение цунами по подводным горам) при их обработке на 32- и 64-разрядных ПК

1.11.5. Повышение производительности в обычных условиях

Вычислительным ядром Mathematica 5.2 теперь можно управлять на компьютере, который независим от пользователя и интерфейсной части. Это выгодно в случаях, когда доступен более мощный отдаленный компьютер. Повышена степень безопасности при работе с Mathematica 5.2.

К сожалению 64-разрядные ПК и ПК с многоядерными процессорами пока не являются массовыми. Однако существенно улучшенные алгоритмы вычислений СКМ Mathematica 5.2 дают заметное ускорение вычислений даже на обычных ПК. Это демонстрируют тесты, представленные на рис. 1.22.

Для Mathematica 4 время исполнения этих тестов составляло 0,2; 5,867 и 2,017 с. Таким образом, для вполне «обычных» вычислений также обеспечено существенное повышение скорости вычислений, причем, как за счет лучших алгоритмов вычислений, так и благодаря применению вполне современных ПК среднего класса. И эта тенденция сохраняется, подчеркивая уникальные возможности систем класса Mathematica.

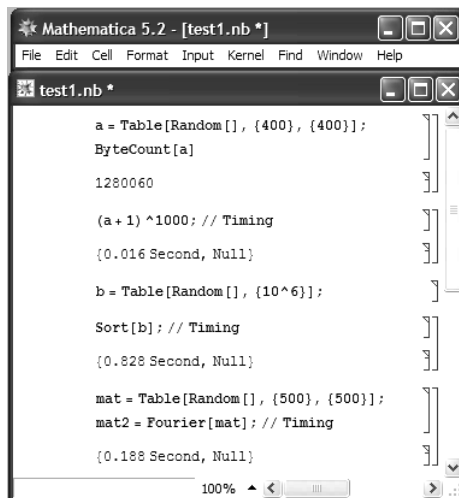


Рис. 1.22. Результаты тестирования СКМ Mathematica 5.2 на ПК с процессором Pentium 4 HT 2,6 ГГц

1.12. Интерфейс пользователя системы Mathematica 6

1.12.1. Запуск Mathematica 6 и изменения в меню системы

Инсталляция системы Mathematica 6 не имеет существенных моментов, достойных подробного описания. После нее на рабочем столе создается ярлык системы с надписью «Mathematica 6». Его активизация запускает систему и выводит ее окна, показанные на рис. 1.23. Среди них показано окно с информацией о системе, которое появляется при исполнении команды About Wolfram Mathematica в позиции **Help** меню (это вполне очевидно с первого взгляда на рисунок, тем более, что окно имеет титульную строку с именем «About Wolfram Mathematica»).

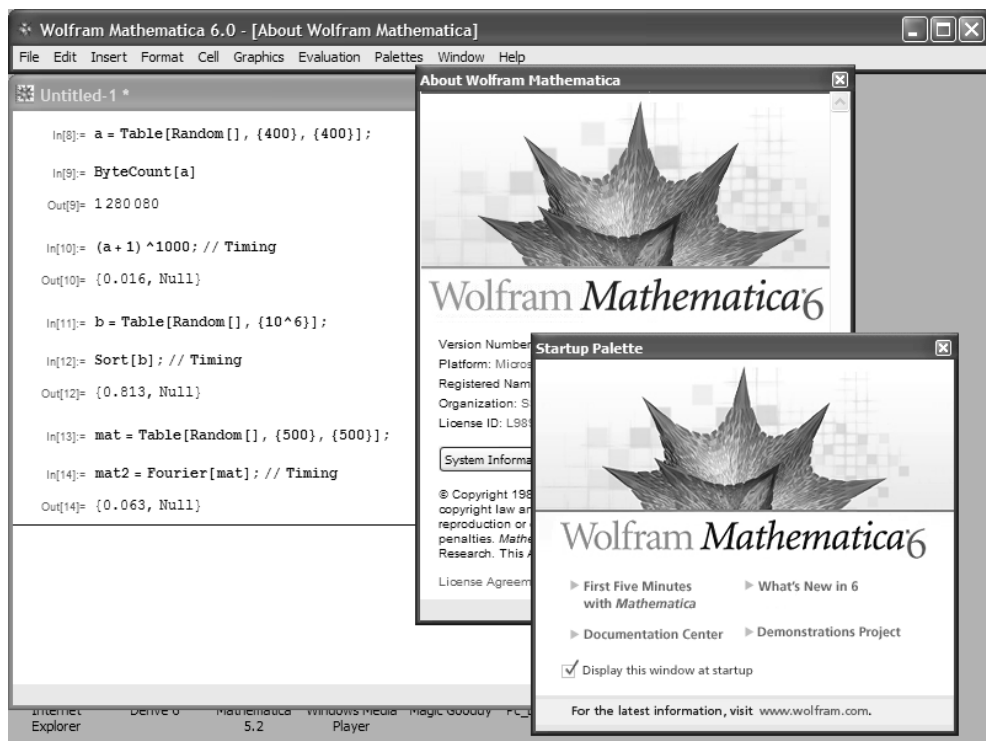


Рис. 1.23. Окно системы Mathematica 6

Меню системы Mathematica 6 серьезно переработано. Можно сразу заметить, что в меню Mathematica 6 исчезла позиция **Find**. Операции поиска перенесены в позицию **Edit** меню, как это сделано в большинстве приложений под операционные системы класса Windows. Исчезла также позиция **Kernel** с командами выбора ядра системы и управления им. Эти команды перекочевали в новую позицию **Evaluation** (Оценка) меню. Появились также новые позиции **Graphics** (вывод окна для создания рисунков с помощью простого графического редактора) и **Palletes** (для вывода палитр с различными математическими символами).

В позиции **Palletes** имеются команды для вызова шести палитр (рис. 1.20). Хотя число палитр в Mathematica 6 уменьшено, число вводимых ими знаков, операторов и функций даже увеличено за счет улучшения организации палитр. Названия команд в позиции **Palletes** соответствуют названию палитр в их титульных строках. Из рис. 1.24 очевидно, что и в Mathematica 6 все палитры настолько загромождают экран дисплея, что реально можно (и нужно) пользоваться одновременно 1–2 палитрами или переходить к работе с дисплеем, с более высоким, чем 800×600 пикселей, разрешением экрана.

Обычно после использования палитр они удаляются, активизацией кнопки со знаком «X» в правом верхнем углу каждой палитры. Типичный вид минимально-

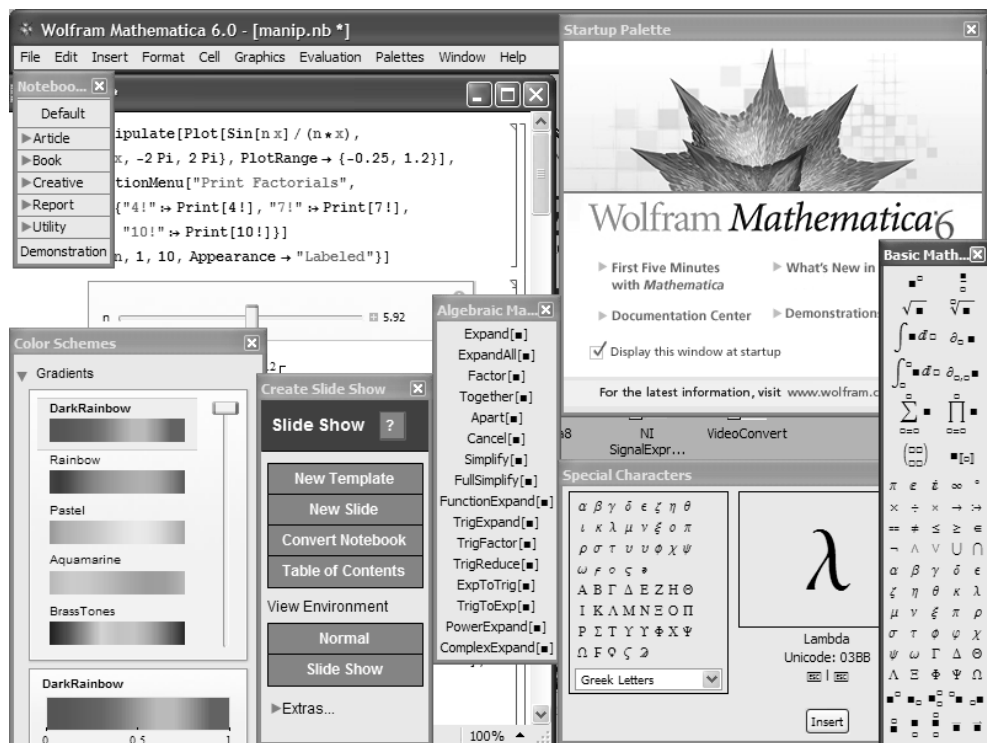


Рис. 1.24. Окна системы Mathematica 6 со всеми ее палитрами

го набора средств графического интерфейса пользователя системы Mathematica 6 показан на рис. 1.25. В минимальный набор окон входит плавающее окно (панель) с меню и хотя бы одно окно ноутбука (документа). Такой вид интерфейса Mathematica 6 обеспечивает максимальный обзор документа, который можно при желании растянуть на весь экран.

1.12.2. Справочная система Mathematica 6

Справочная система Mathematica 6 переработана кардинально. Сразу отметим, что она включает в себя множество команд, вводимых в строках ввода ноутбуков, и новый **Центр документации Documentation Center**. Доступ к последнему из позиций **Help** меню показан на рис. 1.21. Окно Центра документации (попросту справки) показано на рис. 1.26.

Сравнив окно рис. 1.26 с окном справки системы Mathematica 5.2 (рис. 1.17), нетрудно уловить в их принципиальной разнице. В окне рис. 1.25 выставлено как бы на показ все оглавление справки в виде множества кратких гиперссылок. Нет многоступенчатого контекстного меню. В каждом разделе справки есть ги-

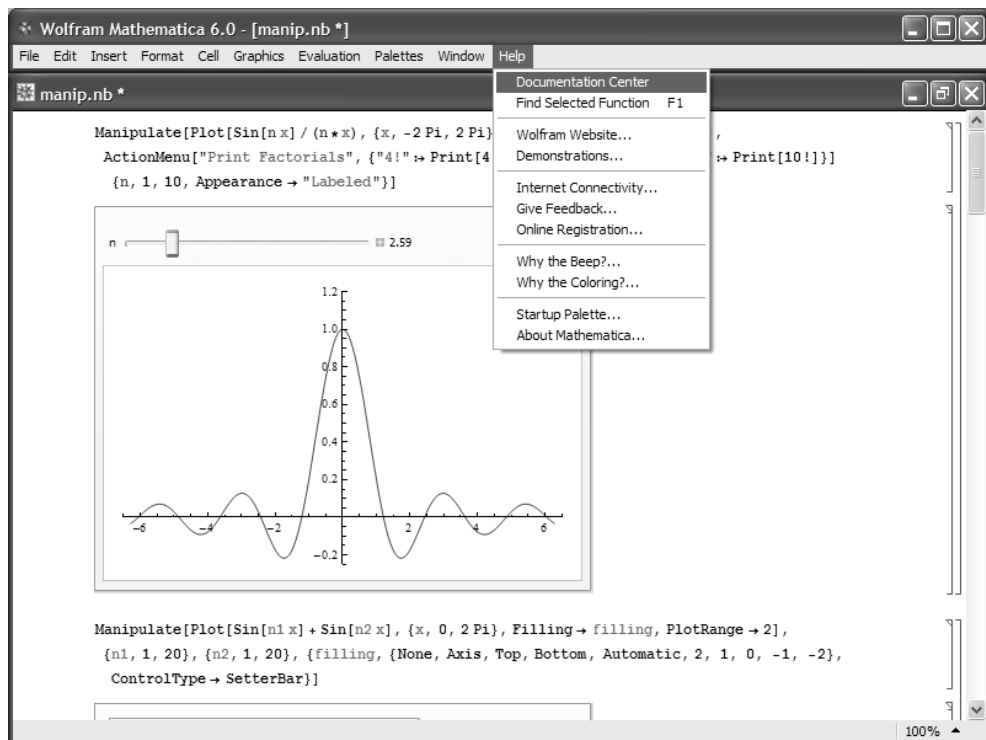


Рис. 1.25. Вид минимального набора окон Mathematica 6

перссылка **New in 6**. Использование этой гиперссылки позволяет отдельно изучить новые возможности системы Mathematica 6.

Здесь уместно отметить, что гиперссылки в справке не имеют привычного вида в виде подчеркнутых снизу надписей. Они представлены обычными надписями, но (как и обычные гиперссылки) активизируются при наведении курсора мыши на них и щелчке левой клавишей мыши. На рис. 1.27 для примера показан раздел справки по динамической интерактивности, которая была описана в конце главы 1 и является важной новинкой системы Mathematica 6.

Справка по большинству функций предельно проста. Для примера на рис. 1.28 показано окно справки по функции символьного интегрирования **Integrate**. Упор сделан на изучение функции по простым и наглядным примерам. Впрочем, в нормально скрытом виде есть масса дополнительных данных по каждой функции. Например, активизировав треугольник с надписью **MORE INFORMATION**, можно открыть дополнительные материалы по заданной функции, треугольник с надписью **EXAMPLES** (на рис. 1.28 он активизирован) открывает доступ к описанию примеров применения функции и т.д.

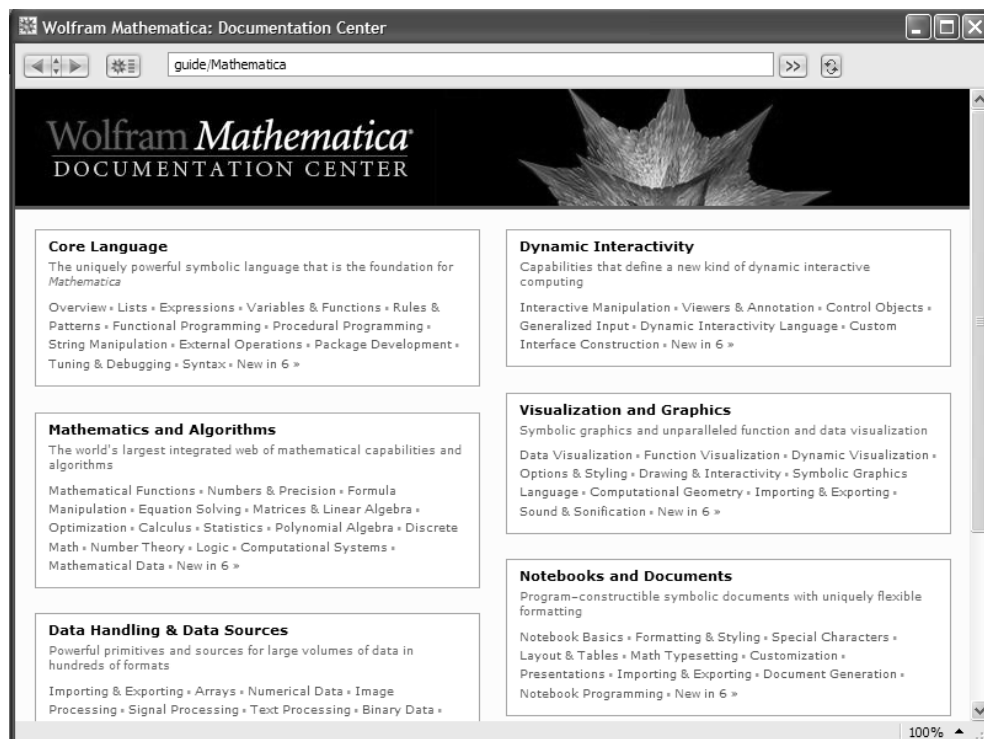


Рис. 1.26. Окно Центра документации (справки)

Интересно отметить, что справка реализована как набор ноутбуков, написанных на языке программирования системы Mathematica. Это означает, что все примеры в справке могут модифицироваться пользователем, и можно немедленно запускать измененные примеры, наблюдая результаты их работы. В принципе копирования примеров в ноутбуки пользоваться вполне возможно, но просто для изучения примеров оно не требуется. Похоже, что по числу примеров новейшая Mathematica 6 ничуть не уступает новым версиям системы Maple и, пожалуй, даже их превосходит.

Окно справки имеет минимум деталей интерфейса. В начале титульной строки имеются три объединенные кнопки: средняя выводит список (историю) обращения к справке, крайние задают переходы на предыдущие и последующие страницы. За ними следует кнопка перехода к начальной странице справки. Затем идет панель навигации по справке. Ее можно использовать и для поиска разделов справки по заданному запросу. Завершается титульная строка кнопкой перезагрузки текущей страницы.

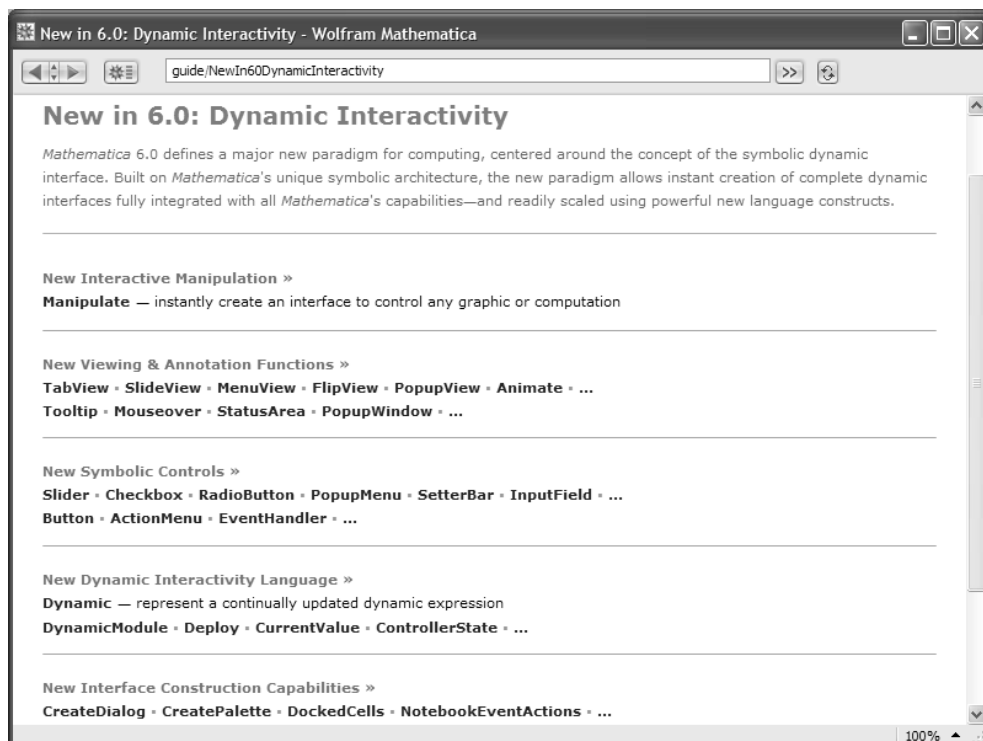


Рис. 1.27. Окно с разделом справки по динамической интерактивности

1.13. Особенности системы Mathematica 6

1.13.1. Основные новинки системы Mathematica 6

Вышедшая в 2007 г. новейшая версия Mathematica 6 представляет собой не просто кардинально переработанную систему, а поистине революционный программный продукт, справедливо приравняемый его разработчиками по последствиям применения к первым версиям системы Mathematica 1/2. Последние в свое время (конец 80-х годов) открыли новое научное направление – системы компьютерной математики для персональных компьютеров. Справедливости ради стоит отметить, что задолго до этого в СССР под руководством академика

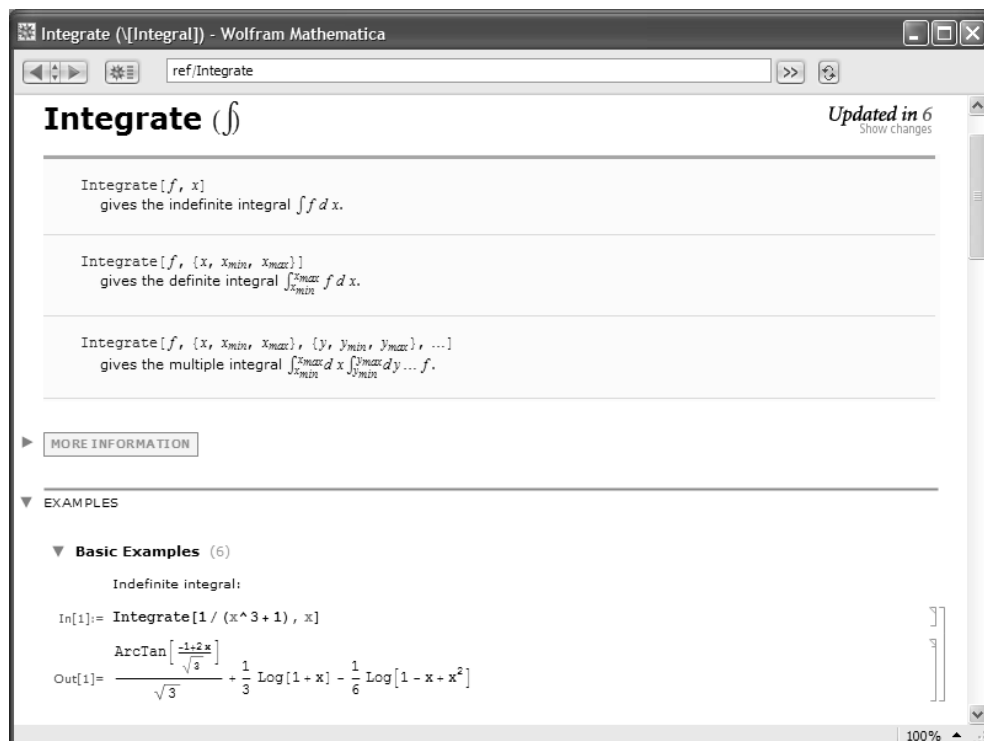


Рис. 1.28. Окно справки по функции символьного интегрирования *Integrate*

Глушкова были созданы первые малые ЭВМ «Мир», выполняющие аналитические вычисления, и язык «Аналитик» для них.

Перечень нововведений в Mathematica 6 настолько обширен, что приводить его весь не имеет смысла. В этой книге новинки системы описаны в ряде глав и дают достаточно полное (но не исчерпывающее) представление об обширных новых возможностях системы. Информацию о текущей версии Mathematica дает системная переменная:

\$Version

6.0 for Microsoft Windows (32-bit) (April 28, 2007)

Отметим основные наиболее крупные и значимые нововведения в системе Mathematica 6:

- включение в ядро системы более 1000 новых функций различного рода, операторов и команд интерфейса;
- дальнейшее увеличение скорости вычислений в несколько раз, иногда и больше;
- новая концепция интерактивного динамического интерфейса, в корне меняющая и резко упрощающая создание ноутбуков с новейшими деталями

интерфейса (кнопками, переключателями, слайдерами и т.д.), подобными маплетам в системе Maple и окнам GUI в MATLAB;

- новая позиция Graph в меню, позволяющая вводить в документы графические окна и с помощью встроенного графического редактора рисовать в них рисунки из графических объектов;
- введение ряда пакетов расширений с сохранением пакетов Add-On, имеющихся в прежних версиях системы;
- поддержка документов, созданных в прежних версиях;
- многочисленные новые функции построения графиков самого различного вида со средствами управления мышью и интерактивными динамическими средствами;
- существенное расширение типов файлов, которые поддерживает система;
- полностью переработанная и легкая в применении справочная система, удобная как начинающим пользователям, так и профессионалам;
- огромное число самых разнообразных примеров буквально на каждую функцию;
- уменьшенное время загрузки системы;
- заметно расширенные средства обращения к обширным Интернет-ресурсам.

При этом Mathematica 6 избежала присущей другим системам (например, Mathcad 14 или Maple 11) и раздражающей опытных пользователей пестроты интерфейса и, уже переходящего рамки разумного, обилия панелей ввода различных символов и других объектов ввода. Тем не менее, такие панели есть и в Mathematica 6, но их немного и их можно вводить по мере надобности.

1.13.2. Скорость работы Mathematica 6

Многие системы компьютерной математики обнаруживают в новых версиях явные признаки замедления работы интерфейса пользователя. Это вызвано не меньшей скоростью вычислений, а существенным усложнением интерфейса и увеличением числа его деталей. Mathematica 6 лишена этого недостатка – при работе с ее интерфейсом никакого замедления в сравнении с прежними, более простыми, версиями не ощущается – разумеется при условии работы на одном и том же достаточно современном компьютере.

Загрузка системы с жесткого диска в оперативную память компьютера также происходит достаточно быстро. Вот данные о загрузке ряда СКМ на компьютере автора (процессор Pentium 4 HT 2,6 ГГц, ОЗУ 1 Гбайт). Данные не требуют комментариев! Заметим лишь, что повторная загрузка происходит быстрее у всех СКМ, поскольку начинает работать кэш-память компьютера.

*Время загрузки (в секундах) различных СКМ
(первая загрузка | повторная загрузка)*

Mathematica 6	Mathematica 5.2	Mathcad 14	Maple 11	MATLAB R2007a
4 2 с	4 2 с	12 8 с	13 6 с	15 6 с

Выполним на Mathematica 6 простые тесты на скорость вычислений, которые были представлены на рис. 1.28 для предшествующей версии системы. Результаты их представлены на рис. 1.29. Нетрудно заметить, что при переходе с работы с Mathematica 5.2 к работе с Mathematica 6, время вычислений в первых двух тестах практически не изменилось, но в тесте на быстрое преобразование Фурье оно уменьшилось втрое.

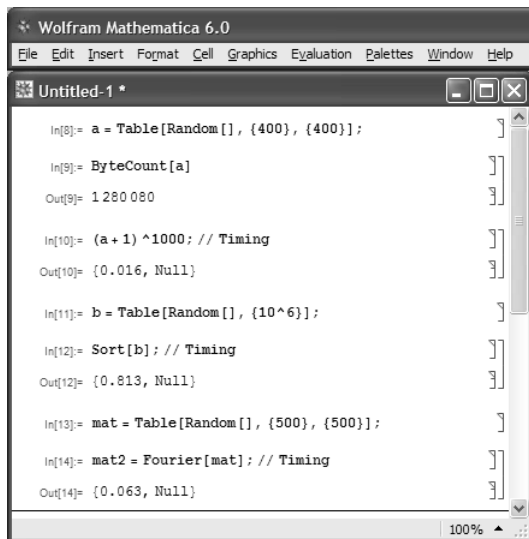


Рис. 1.29. Результаты тестирования СКМ Mathematica 6 на ПК с процессором Pentium 4 HT 2,6 ГГц

Можно сказать, что если разработка Mathematica 5.2 шла под эгидой повышения скорости вычислений, то разработка новой Mathematica 6 пошла по иному пути – расширению функциональности системы и резкому увеличению ее функций. Но и повышение скорости вычислений не осталось без внимания. Мы еще вернемся к детальному тестированию системы Mathematica 6.

1.13.3. Ориентация в изучении системы на примеры ее применения

Первые версии Mathematica страдали отсутствием представительного числа примеров применения функций. Это быстро поняли конкуренты – разработчики систем класса Maple, насытившие последние массой примеров, число которых достигало (по весьма приближенным оценкам) с десятков тысяч. Эта цифра включает все примеры – как простые, так и достаточно сложные.

Создатели Mathematica быстро поняли свой просчет, и теперь в Mathematica 6 включены примеры по каждой из почти трех тысяч функций. Число примеров по некоторым наиболее важным функциям достигает полусотни и больше. Есть основания полагать, что по числу примеров Mathematica 6 вышла в лидеры среди систем компьютерной алгебры.

Уместно отметить и простоту навигации по огромному числу функций этой системы. Подавляющее большинство функций имеет составные названия. Хотя они англоязычные, многие из них вполне понятны даже начинающим пользователям, например **ArcSin**, **Sin** или даже **ParametricPlot3D** либо **ColorFunction**. Это заметно облегчает навигацию по функциям.

В справке есть как полный алфавитный перечень всех функций, так и перечень всех новых функций, введенных в Mathematica 6. Это облегчает разбор ноутбуков, которые создавались в ранее созданных версиях системы. К сожалению, при такой серьезной переработке системы, которая была сделана в Mathematica 6, созданные в прежних версиях ноутбуки могут требовать заметной доработки. Система помечает неработающие фрагменты красным цветом и выдает сообщения о такой нестыковке.

При открытии справки по каждой функции материал справки описан предельно просто и с небольшим числом базовых примеров (Basic Examles). В нормально закрытом разделе MORE INFORMATION (Дополнительная информация) можно найти теоретические сведения и описание алгоритмов, положенных в основу той или иной функции. Разумеется, нередко такое описание есть ноу-хау разработчиков Mathematica 6, и потому детальностью «не страдает». Понятно, что описание таких алгоритмов читатель не найдет и в книгах — всему свое время! Зато у пользователя системой Mathematica есть прекрасные возможности для изобретения и отладки своих алгоритмов и сравнения их с алгоритмами, предложенными в Mathematica. Кроме того, освоивший программирование в Mathematica любой версии может раздобыть m-файлы пакетов расширения Add-On, которые являются неисчерпаемым источником идей и детальнейшим описанием многих алгоритмов на языке программирования системы Mathematica.

1.13.4. Динамическая интерактивность при символьных вычислениях

Динамическая интерактивность — новое качество ноутбуков системы Mathematica 6. Оно заключается в применении довольно простых средств, позволяющих превращать ноутбуки (документы) системы в диалоговые (интерактивные) окна с элементами, позволяющими динамически управлять параметрами исходных данных для вычислений (в том числе символьных) и построения графиков.

Одним из таких средств является управляющая функция **Manipulate**. Это типичный программный модуль, который будет детально описан в дальнейшем. Пока же рассмотрим пример его применения для вычисления интеграла с подынтегральной функцией $\sin(a^m)$ при разных целых m от 0 до 10, показан на

рис. 1.30. Функция Manipulate в ячейке вывода строит интерактивное окно со слайдером, движок которого можно перемещать мышью и окном вывода результатов аналитического интегрирования. При этом m меняется и меняется символьное значение интеграла. На рис. 1.30 показан случай, когда слайдером задано значение $m=1$. Оно выведено в конце слайдера.

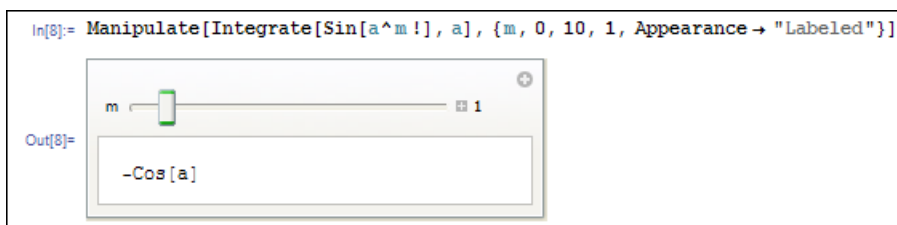


Рис. 1.30. Вычисление интеграла с подынтегральной функцией $\sin(a^m!)$ – случай $m=1$

Для демонстрации динамической интерактивности на рис. 1.31 и 1.32 показаны примеры для $m=2$ и $m=3$. Эти значения m получены передвиганием движка слайдера. Хорошо видно, что при изменении m меняется формула, представляющая символьное значение интеграла.

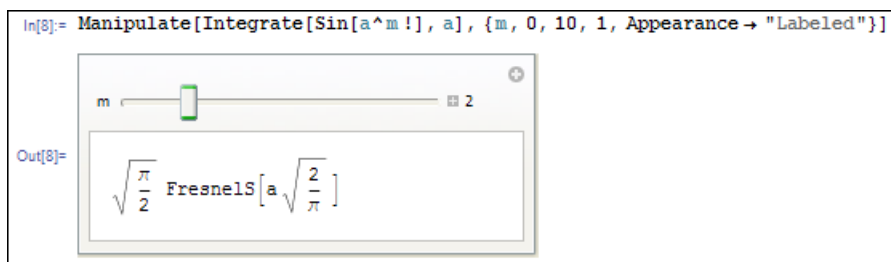


Рис. 1.31. Вычисление интеграла с подынтегральной функцией $\sin(a^m!)$ – случай $m=2$

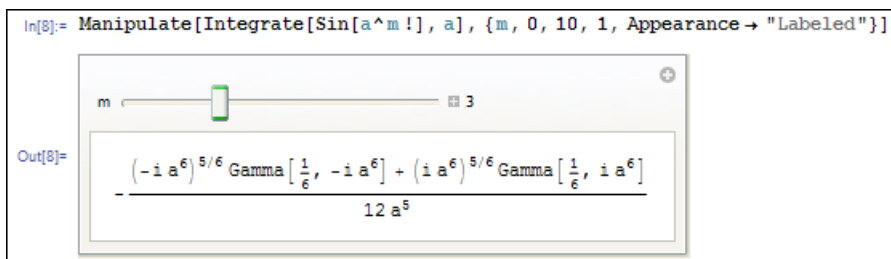


Рис. 1.32. Вычисление интеграла с подынтегральной функцией $\sin(a^m!)$ – случай $m=3$

1.13.5. Управление графиками мышью

Графика системы Mathematica долгое время имела явный недостаток – отсутствие возможности поворота трехмерных графиков фигур мышью. Эта возможность уже давно была реализована в других системах компьютерной математики – Mathcad, Maple и MATLAB. Правда, в Mathematica 4/5 можно было загрузить ноутбук с процедурой, дающей такую возможность. Однако только в Mathematica 6 она присутствует уже как свойство трехмерных графиков, построение которых задано в ядре системы.

На рис. 1.33 дан классический пример на построение поверхности, описываемой функцией (выражением) двух переменных – $x^2 + y^2$. Поверхность (объемная парабола) строится с помощью функции **Plot3D** и представлена при построении по умолчанию – выпуклостью вниз.

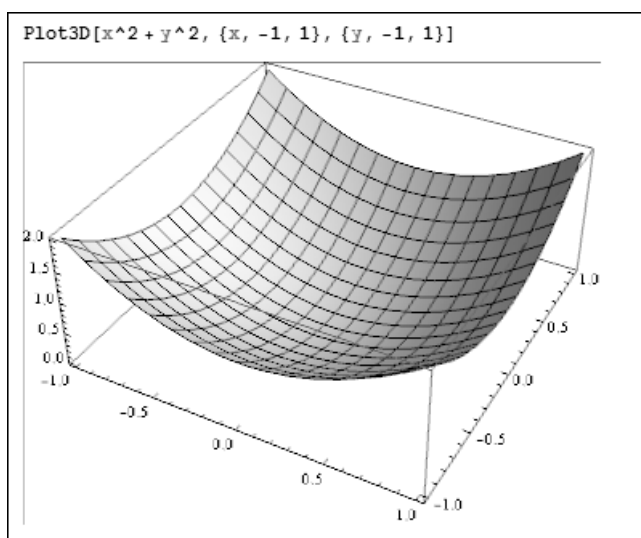


Рис. 1.33. Построение выпуклой параболы

Если ввести курсор мыши внутрь рисунка, он изменится на изображение стрелки, указывающей на вращение. Нажав левую клавишу мыши и удерживая ее нажатой, можно начать поворачивать фигуру и наблюдать ее вращение в пространстве. На рис. 1.34 показан результат таких манипуляций, в результате которой объемная парабола оказалась построенной выпуклостью вверх.

Возможность вращения трехмерных графических объектов мышью позволяет рассматривать их с разных сторон и выявлять изначально невидимые особенности их. Если нажать клавишу **Shift**, то при нажатой левой клавише мыши можно перемещать график по полю графического окна, а при нажатой клавише **Ctrl** можно плавно приближать или удалять график (операция **Zoom**).

```
Plot3D[x^2 + y^2, {x, -1, 1}, {y, -1, 1}]
```

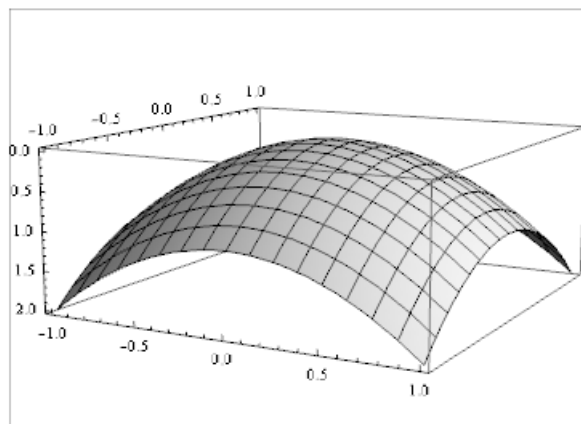


Рис. 1.34. Построение выпуклой параболы

1.13.6. Динамическая интерактивность при графической визуализации

Возможность вращать трехмерные фигуры (графики) мышью – это лишь один из компонентов графической динамической визуализации. Разработчики Mathematica 6 вошли дальше и ввели возможности наблюдения за поведением графиков различного типа (в том числе трехмерных) при изменении любого из их параметров. Эти возможности также обеспечивает функция **Manipulate**.

На рис. 1.35 показано построение графика по выражению $\sin(nx)/(nx)$ с помощью функций **Plot** и **Manipulate**. При этом задано изменение параметра n от 1 до 10. График на рис. 1.35 построен для $n=1$.

Теперь, перемещая движок слайдера, можно наблюдать за изменениями графика, что существенно повышает его наглядность. На рис. 1.36 показан тот же ноутбук, но при положении движка слайдера, соответствующего $n=2,52$. Характер изменения графика очевиден.

Число слайдеров и изменяемых параметров нетрудно увеличить, просто задав изменения нескольким переменным в списке параметров функции **Manipulate**. На рис. 1.37 показан пример построения параболической поверхности, у которой меняются два параметра – коэффициенты n_1 и n_2 при переменных x и y . Поверхность построена при значениях $n_1=n_2=1$ и слегка повернута мышью для получения более наглядного изображения.

На рис. 1.38 показана та же поверхность после перемещения движков слайдеров с установкой значений $n_1=-0,445$ $n_2=0.62$. Хорошо видно, как меняется вид поверхности. Можно просмотреть плавное изменение этой поверхности при воз-

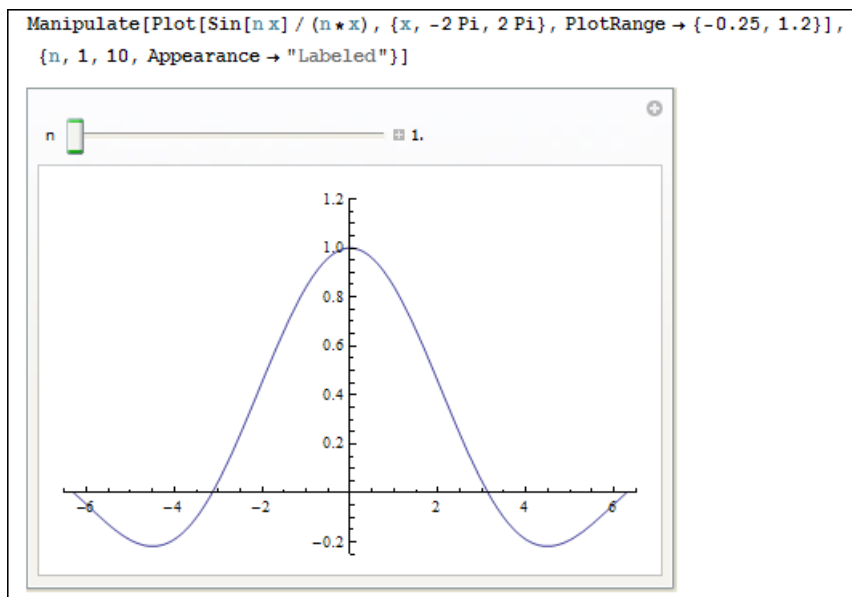


Рис. 1.35. Построение графика выражения $\sin(nx)/(nx)$ со слайдером для изменения параметра n для случая $n=1$

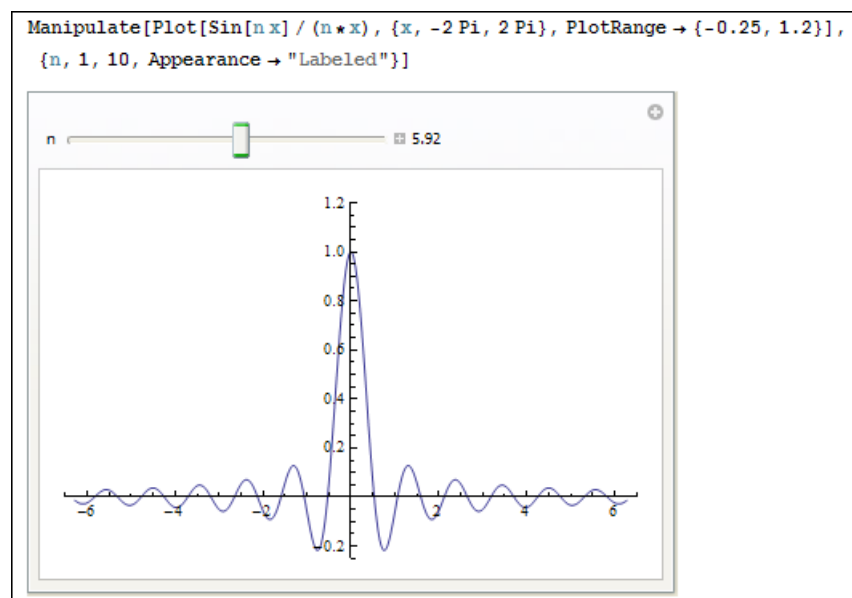


Рис. 1.36. Построение графика выражения $\sin(nx)/(nx)$ со слайдером для изменения параметра n для случая $n=2,52$

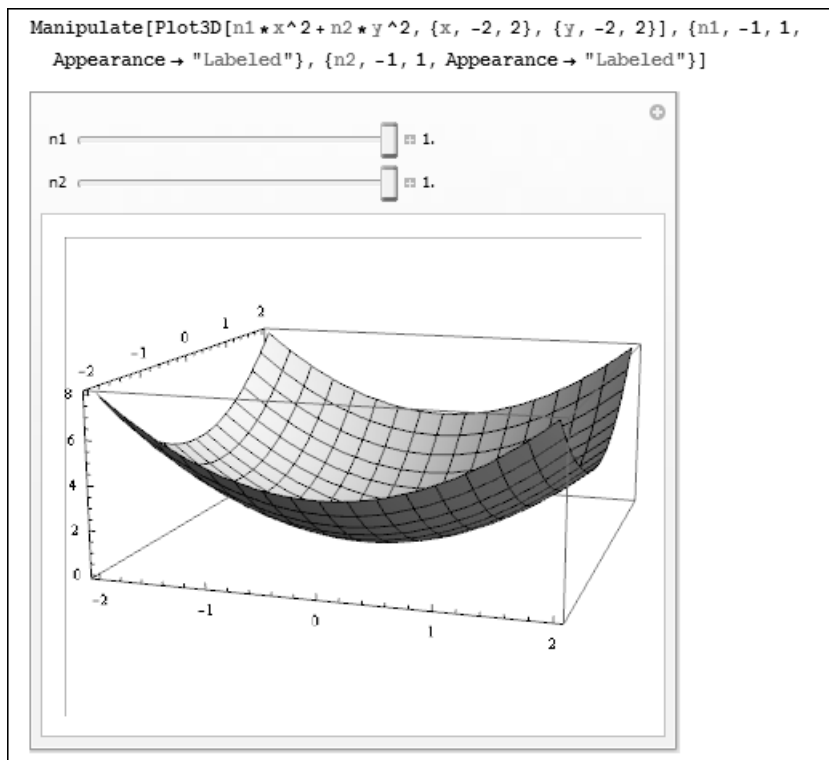


Рис. 1.37. Построение параболической поверхности при двух изменяемых параметрах (случай установки $n1=n2=1$)

можном перемещении движков слайдеров. При этом затраты на их вывод минимальны – задаются параметрами функции **Manipulate**. В других системах для реализации таких интерактивных возможностей пришлось бы воспользоваться процедурным или визуально-ориентированным программированием, предварительно освоив его.

Однако и на этом возможности Mathematica 6 не заканчиваются. Можно реализовать анимацию построенных графических объектов. На рис. 1.38 показано меню операций, которое появляется при активизации мышью кружка со знаком «+» в верхнем правом углу ноутбука. Внизу этого меню есть позиция **Autorun**, выбор которой вызывает появление сверху окна панели анимационного проигрывателя Autorun (рис. 1.39).

По умолчанию проигрыватель Autorun обеспечивает просмотр анимации поверхности при плавном автоматическом перемещении вначале одного движка, а затем второго. Можно вмешаться в работу проигрывателя, например, остановив его, изменив скорость и направление перемещения движков, и, наконец, заурить его. Соответствующие кнопки с очевидным мнемоническим изображением имеются на панели проигрывателя.

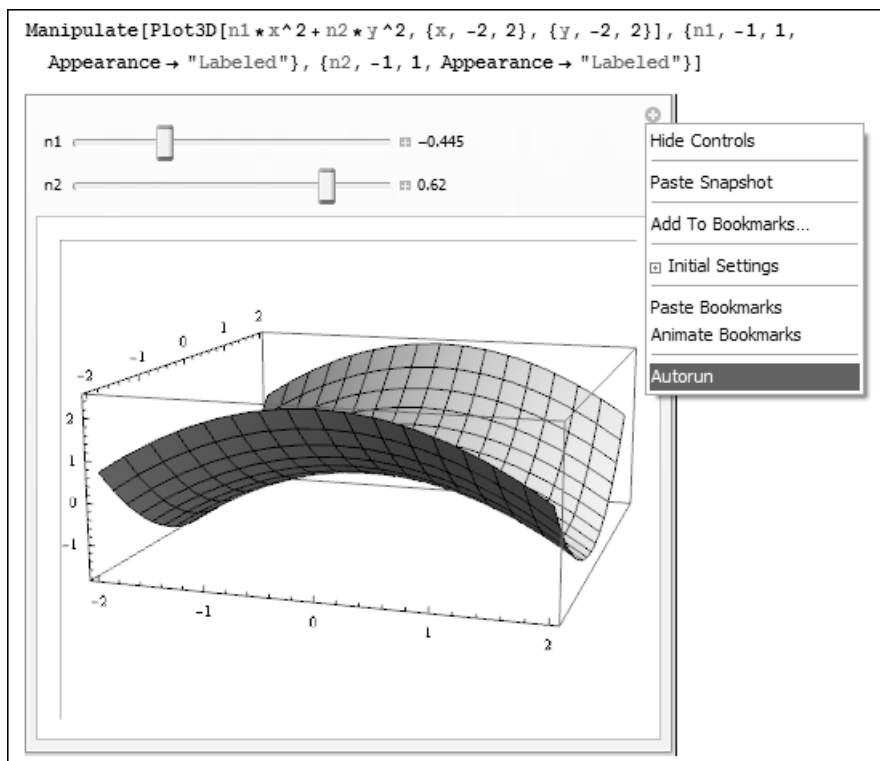


Рис. 1.38. Построение параболической поверхности при двух изменяемых параметрах (случай установки $n1=-0,445$ $n2=0.62$)

1.13.7. Комплексное тестирование Mathematica 6 на скорость вычислений

При работе с любой системой компьютерной математики пользователя в первую очередь интересует производительность его компьютера при работе системы Mathematica и выполнении в ней тех или иных операций. В Mathematica 6 для этого включен специальный тестирующий пакет, который вызывается на исполнение следующей командой:

Needs[«Benchmarking`»];

При этом начинается выполнение 15 тестов, которое занимает от менее минуты до нескольких минут в зависимости от производительности ПК. Этот тест можно также вызвать из окна с информацией о системе Mathematica 6 About Mathematica. По окончании тестирования появляется окно с отчетом о нем, показанное на рис. 1.40.

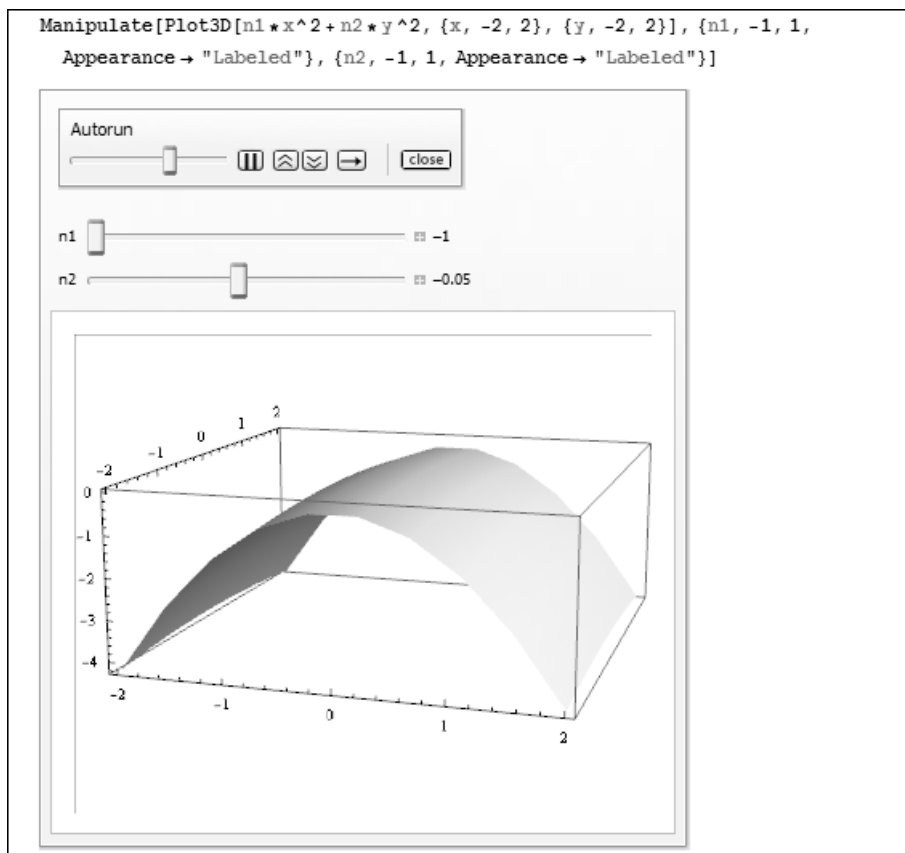


Рис. 1.39. Построение параболической поверхности и ее анимация с помощью проигрывателя Autorun

В этом окне перечислены все 15 тестов, которые используются для проверки системы Mathematica 6. Можно открыть соответствующую нужному тесту ячейку и просмотреть выражение, обеспечивающее тестирование. Например, на рис. 1.40 открыта ячейка, соответствующая тестированию на скорость вычисления 380 000 точных знаков числа π .

Результаты комплексного тестирования в виде гистограмм приведены на рис. 1.41. Нетрудно заметить, что компьютер автора с именем `dvr` занял «почетное» место со значением скорости вычислений, близкой к условному балу в 1. Для сравнения: компьютер на двухъядерном процессоре Intel Xeon 5160 имеет бал 2,84, т.е. выполняет вычисления примерно в три раза быстрее.

Другая страница отчета (рис. 1.42) дает полные данные о сравнительном времени вычислений для всех 15 тестов для тестируемого и образцовых компьютеров. Анализ этих данных позволит оценить возможности тестируемого компьютера при выполнении того или иного теста.

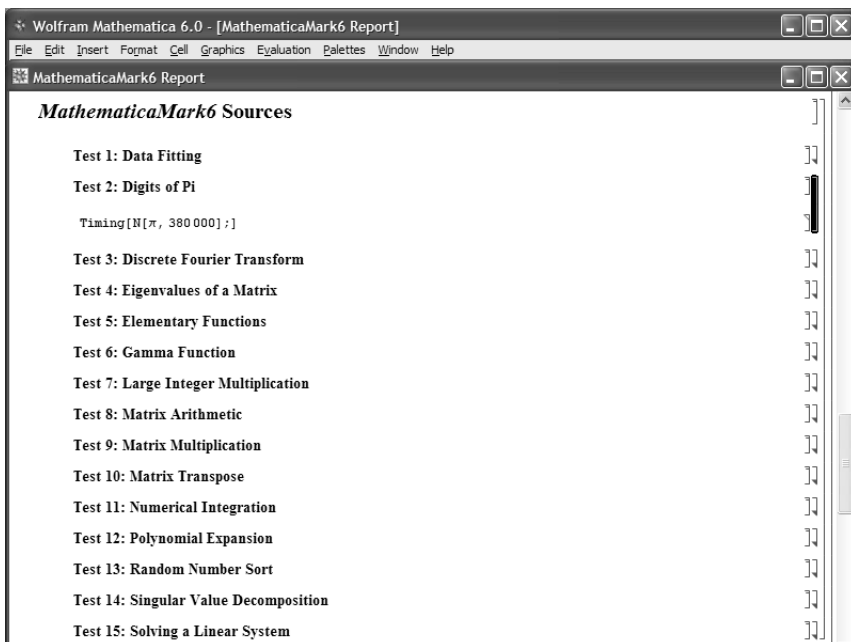


Рис. 1.40. Окно с отчетом о тестировании Mathematica 6

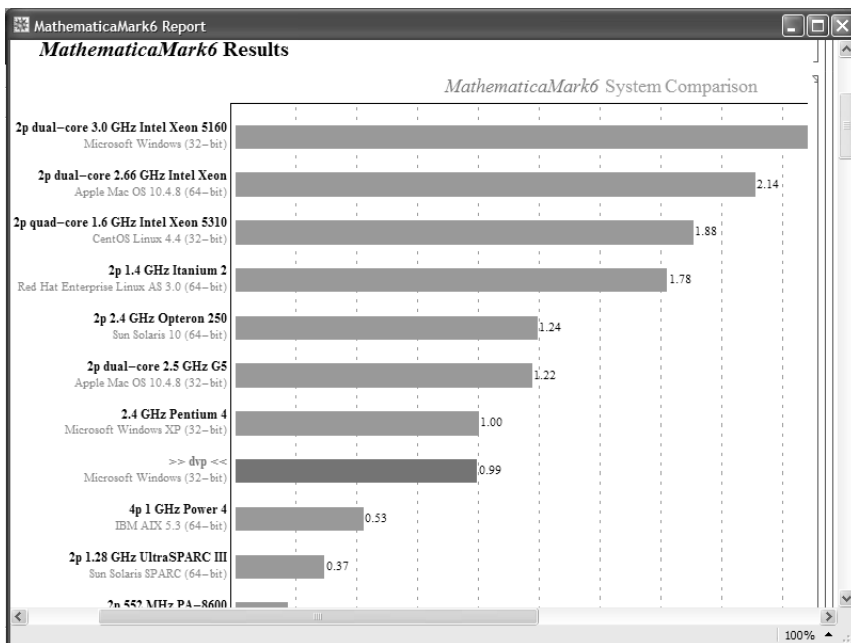


Рис. 1.41. Гистограммы тестирования для различных компьютеров

MathematicaMark6 Report

MathematicaMark6 Detailed Timings

	Total	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
2p dual-core 3.0 GHz Intel Xeon 5160 Microsoft Windows (32-bit)	30.39	2.08	0.67	1.09	2.70	4.03	0.50	0.89	3.17	2.00	2.16
2p dual-core 2.66 GHz Intel Xeon Apple Mac OS 10.4.8 (64-bit)	40.32	1.75	0.44	1.85	3.38	11.75	0.30	0.52	3.62	2.07	2.39
2p quad-core 1.6 GHz Intel Xeon 5310 CentOS Linux 4.4 (32-bit)	45.78	2.01	1.20	0.93	4.42	8.32	0.88	1.54	5.12	1.61	3.14
2p 1.4 GHz Itanium 2 Red Hat Enterprise Linux AS 3.0 (64-bit)	48.55	3.82	0.83	1.39	1.92	2.80	0.33	0.76	7.44	3.29	4.84
2p 2.4 GHz Opteron 250 Sun Solaris 10 (64-bit)	69.34	3.92	0.60	3.68	5.79	15.35	0.41	0.90	7.62	4.32	3.31
2p dual-core 2.5 GHz G5 Apple Mac OS 10.4.8 (32-bit)	70.59	3.92	1.25	3.88	4.53	18.06	0.81	1.43	6.39	2.29	4.96
2.4 GHz Pentium 4 Microsoft Windows XP (32-bit)	86.19	5.05	1.23	3.00	8.84	7.50	0.92	1.63	9.17	8.78	5.39
>> dvp << Microsoft Windows (32-bit)	86.69	4.64	1.38	2.67	8.41	7.88	0.94	1.67	9.05	9.61	5.05
4p 1 GHz Power 4 IBM AIX 5.3 (64-bit)	163.75	7.34	1.56	5.90	9.20	60.13	0.94	1.64	10.15	14.40	3.30
2p 1.28 GHz UltraSPARC III Sun Solaris SPARC (64-bit)	235.18	12.01	1.97	9.66	12.69	45.34	1.13	2.43	26.31	32.34	7.40
2p 552 MHz PA-8600 HP HP-UX 11.11 (64-bit)	405.53	25.27	3.92	30.48	9.76	64.92	33.08	3.41	49.54	22.65	28.23

Timings are CPU time in seconds

100%

Рис. 1.42. Полные данные о тестировании для различных компьютеров

Типовые средства программирования

2.1. Mathematica как система программирования	98
2.2. Функции символьных вычислений	106
2.3. Применение образцов	114
2.4. Основы функционального программирования в среде Mathematica	116
2.5. Основы процедурного программирования	122
2.6. Организация циклов	123
2.7. Условные выражения и безусловные переходы	128
2.8. Механизм контекстов	132
2.9. Программирование ввода-вывода	136
2.10. Функции задания объектов GUI ноутбуков	140

2.1. Mathematica как система программирования

2.1.1. Понятие о входном языке системы и языке реализации

Система Mathematica способна без общепринятого программирования решать огромное число математических и научно-технических задач. Однако все средства системы (включая алфавит ее входного языка, его буквы, цифры, операторы и многочисленные специальные знаки) в сущности являются частью *проблемно-ориентированного языка программирования сверхвысокого уровня* [34,45,46]. По своим возможностям в выполнении математических и научно-технических вычислений этот язык намного превосходит обычные универсальные языки программирования, такие как Фортран, Бейсик, Паскаль или Си.

Важно подчеркнуть, что здесь речь идет о языке программирования системы Mathematica, а не о языке реализации самой системы. *Языком реализации* является универсальный язык программирования C++, показавший свою высокую эффективность в качестве языка системного программирования.

2.1.2. Возможности языка программирования системы Mathematica

Мощь системы Mathematica, как средства программирования решения математических задач, обусловлена необычно большим (в сравнении с обычными языками программирования) набором функций, среди которых немало таких, которые реализуют сложные и практически полезные математические преобразования и современные вычислительные методы (как численные, так и аналитические). Число функций в Mathematica 6 достигает 3000.

Язык программирования системы Mathematica является типичным интерпретатором и не предназначен для создания исполняемых файлов. Впрочем, для отдельных выражений этот язык может осуществлять компиляцию с помощью функции **Compile**, что полезно при необходимости увеличения скорости счета в старых реализациях системы. В новых версиях алгоритмы улучшены настолько, что особой необходимости в применении функции **Compile** просто нет.

Язык систем Mathematica вобрал в себя лучшие средства ряда поколений языков программирования, таких как Бейсик, Фортран, Паскаль и Си. Благодаря этому он позволяет легко реализовать все известные типы (концепции) программирования: функциональное, структурное, объектно-ориентированное, математическое, логическое, рекурсивное и т.д. В него включены и средства визуально-ориентированного программирования на основе применения шаблонов математических символов, таких как знаки интеграла, суммирования, произведения и т.д.

Внутреннее представление всех вычислений базируется на применении полных форм выражений, представленных функциями. И вообще, функциям в системе Mathematica принадлежит решающая роль в организации вычислений любого вида. Таким образом, Mathematica фактически изначально реализует как главный *функциональный* метод программирования – один из самых эффективных и надежных. А обилие логических операторов и функций позволяет полноценно реализовать и *логический* метод программирования. Множество операций преобразования выражений и функций позволяют осуществлять программирование на основе *правил преобразования*.

Необходимо также отметить, что язык системы позволяет разбивать программы на отдельные модули (блоки) и хранить эти модули в тексте документа или на магнитном диске. Возможно создание полностью самостоятельных блоков – поименованных процедур и функций с локальными переменными. Все это, наряду с типовыми управляющими структурами, позволяет реализовать *структурное* и *модульное* программирование.

Столь же естественно язык системы реализует ставшее модным в последнее время *объектно-ориентированное программирование*. Оно, прежде всего, базируется на обобщенном понятии объекта и возможности задания множества связанных друг с другом объектов. В системе Mathematica каждая ячейка документа является объектом и порождается другими, предшествующими объектами. При этом содержанием объектов могут быть математические выражения, входные и выходные данные, графики и рисунки, звуки и т.д.

С понятием объекта тесно связаны три основных свойства: инкапсуляция, наследование и полиформизм. Все они органично присущи объектам системы Mathematica и не требуют для своей реализации каких-либо специальных средств.

Инкапсуляция означает объединение в одном объекте как данных, так и методов их обработки.

Наследование означает, что каждый объект, производный от других объектов, наследует их свойства.

Полиформизм – свойство, позволяющее передать ряду объектов сообщение, которое будет обрабатываться каждым объектом в соответствии с его индивидуальными особенностями.

Приведенный ниже пример объектно-ориентированного программирования дает три определения, ассоциированные с объектом h:

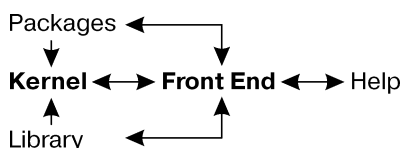
```
h/: h[x_] + h[y_] := hplus[x, y]
h/: p[h[x_] , x] := hp[x]
h/: f_[h[x_] ] := fh[f, x]
```

В принципе, язык программирования системы Mathematica специально создан для реализации любого из перечисленных подходов к программированию, а также ряда других, например *рекуррентного программирования*, при котором очередной шаг вычислений базируется на данных, полученных на предыдущих шагах. Возможно и *рекурсивное программирование* – это когда функция в общем случае неоднократно обращается к себе самой. Наглядным примером этому может служить вычисление факториала рекурсивным методом: $N! = N * (N-1)!$.

Средства языка Mathematica позволяют осуществить и элементы *визуально-ориентированного* программирования. Его смысл заключается в автоматической генерации программных модулей путем указания визуально понятного объекта – чаще всего кнопки. Mathematica позволяет создавать палитры и панели с различными кнопками, позволяющими управлять программой, или вводить новые программные объекты. Однако, визуально-ориентированное программирование пока не является основным и находится в зачаточном состоянии. Примеры его реализации мы рассмотрим в дальнейшем.

2.1.3. Структура систем Mathematica

Чтобы понять, как выполняются вычисления в системе Mathematica, нужно хотя бы в общих чертах знать структуру систем Mathematica. Ее можно представить в следующем виде:



Для ориентации системы на конкретную машинную платформу служит интерфейсный процессор **Front End**. Именно он определяет, какой вид имеет пользовательский интерфейс системы. Так, в Главе 1 данной книги был описан интерфейсный процессор для ПК с массовыми операционными системами класса Windows. Разумеется, интерфейсные процессоры систем Mathematica для других платформ могут иметь свои нюансы, но особых различий с описанным интерфейсным процессором у этих версий систем нет.

Именно через интерфейсный процессор осуществляется связь пользователя с системой. Пользователь вводит вычисляемые выражения с помощью клавиатуры и следит за вводом с помощью дисплея. Интерфейсный процессор запрашивает у ядра нужные операторы, функции и правила их преобразования, вычисляет с их помощью выражение, и результат вычислений выводит на экран дисплея.

Центральное место в системах класса Mathematica занимает машинно-независимое ядро математических операций – **Kernel**. Оно содержит набор операторов и функций, правил вычислений и преобразований математических выражений. Ядро сделано достаточно компактным с тем, чтобы любая функция из него вызывалась достаточно быстро. Для расширения набора функций служит библиотека Library и набор пакетов расширения Packages. Пакеты расширений готовятся на собственном языке программирования систем Mathematica и являются главным средством расширения возможностей системы и их адаптации к решению конкретных классов задач пользователя. Кроме того, системы имеют встроенную справочную систему – **Help**. Она содержит ряд электронных книг с «живыми» примерами.

2.1.4. Идеология систем Mathematica

Идеология систем Mathematica базируется на двух, казалось бы, взаимно исключающих друг друга, положениях:

- решение большинства математических задач в системе может производиться в диалоговом режиме без традиционного программирования;
- входной язык общения системы является одним из самых мощных языков функционального программирования, ориентированных на решение различных задач (в том числе математических).

Противоречивость этих положений чисто кажущаяся. На самом деле Mathematica – типичная система программирования с проблемно-ориентированным языком программирования сверхвысокого уровня. Его можно отнести к классу *интерпретаторов*. Как известно, языки такого типа последовательно анализируют (интерпретируют) каждое выражение и тут же исполняют его. Таким образом, работа с системой происходит в диалоговом (интерактивном) режиме: пользователь задает системе задание, а она тут же выполняет его. Разумеется, Mathematica содержит достаточный набор управляющих структур процедурного программирования для создания условных выражений, ветвления в программах, циклов и т.д., для полной автоматизации вычислений и легкого создания пользователем своих процедур и функций.

В новых реализациях Mathematica 5/6 к указанным двум принципам добавлен ряд новых принципов:

- за счет устранения ограничений по скорости реализации численных методов системы Mathematica 5/6 стали поистине универсальными СКМ;
- системы стали существенно расширяемыми за счет применения встроенных (Add-On) и внешних пакетов расширения, а также пакетов расширения, создаваемых пользователем;
- системы реализуют высочайшее качество подготовки ноутбуков – документов, содержащих одновременно текстовые записи, аналитические выражения, таблицы, графики, рисунки и другие компоненты;
- системы позволяют создавать полностью завершенные высококачественные электронные уроки, статьи и книги с высоким уровнем визуализации всех видов вычислений;
- системы стали интеллектуальными системами предоставления знаний в области фундаментальной и прикладной математики.

Все это существенно расширяет возможности СКМ Mathematica 5/6 и делает их применение более привлекательным.

2.1.5. Пакеты расширения Add-On

Ядро системы Mathematica 5 содержит около 1000 функций [58]. Оно сделано достаточно компактным, для того чтобы вызов нужной функции осуществлялся достаточно быстро. При этом никаких указаний на место функции не требуется.

Еще около 800 функций размещено в так называемых пакетах расширения Add-On [59]. В состав Mathematica 4/5 включены следующие пакеты расширения:

Algebra – алгебра

DiscreteMath – дискретная математика

Graphics – графика

Miscellaneous – всячина

NumericalMath – численные вычисления

Utilities – утилиты

Calculus – вычисления

Geometry – геометрия

LinearAlgebra – линейная алгебра

NumberTheory – теория чисел

Statistics – статистика

Более детально назначение пакетов Add-On рассмотрено в Главе 9.

2.1.6. Полная и частичная загрузка пакетов расширения Add-On

Для применения той или иной функции из пакетов расширения, необходимо весь пакет или часть его (вплоть до отдельной функции) загрузить в память компьютера. Это делается с помощью следующих команд:

- `<<Dir`` – загрузка всех пакетов из заданной папки с пакетами;
- `<<Dir`Package`` – загрузка заданного пакета из заданной папки с пакетами.

Например, для вызова функций решения алгебраических неравенств из пакета средств алгебры необходимо выполнить следующую команду:

```
<<Algebra`AlgebraicInequalities`
```

Для загрузки пакетов расширения используется и функция:

```
Needs["context`"]      Needs["context`", "file"]
```

Полный список пакетов расширения Mathematica 6 можно найти в папке Add-On\Packages. С помощью функции `Names["Name`*"]`, где Name – имя пакета, после загрузки пакета можно вывести список вошедших в него функций. Например, ниже это дано для пакета расширения по сплайнам:

```
<<Splines`
```

```
Names["Splines`*"]
```

```
{Bezier, CompositeBezier, Cubic, RenderSpline, Spline, SplineDivision, SplineDots, SplineFit, SplineFunction, SplinePoints}
```

Функции пакетов расширения заданы в виде файлов с расширением .m, написанных на языке программирования системы Mathematica. Их можно прочесть и модифицировать с помощью любого текстового редактора. Просмотр этих файлов дает богатейший материал для программиста, желающего создавать свои пакеты расширения.

2.1.7. Применение пакетов Add-On системы Mathematica 6

Пакеты Add-On системы Mathematica 6 размещены в папке Add-On\Packages. В ней расположено 48 пакетов расширения, существенно расширяющих и без того мощные средства системы в решении самых разнообразных задач. Некото-

рые пакеты расширения подобны таковым для систем Mathematica 5.1/5.2. Однако полное совпадение средств пакетов и наборов функций не гарантируется. Поэтому рекомендуется внимательно ознакомиться с теми пакетами расширения системы Mathematica 6, которые относятся к сфере профессиональных интересов читателя.

Функции пакетов становятся доступными, как и в Mathematica 5, после исполнения команды:

```
Needs["Имя_пакета`"]
```

Например, для построения круговой диаграммы можно воспользоваться пакетом расширения PieCharts, вызвав его командой:

```
Needs["PieCharts`"]
```

После этого можно воспользоваться функцией **PieChart[{list}]** (см. пример на рис. 2.1). Функция строит круговую диаграмму, у которой сумма всех элементов листа принимается за 100% площади. Каждый сегмент диаграммы нумеруется в соответствии с положением его данных в списке и выделяется своим цветом.

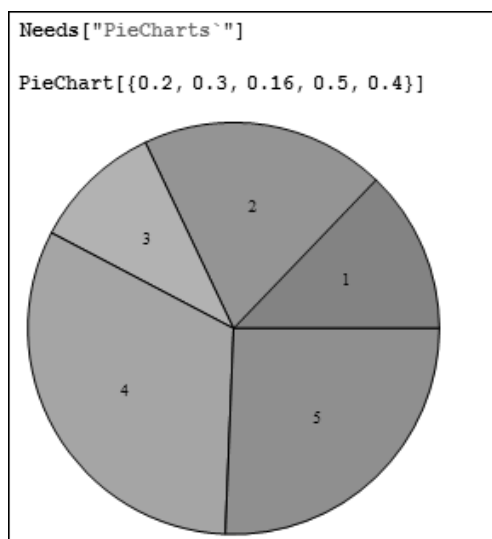


Рис. 2.1. Пример построения круговой диаграммы с применением пакета расширения PieChart

Следует отметить, что примеров на применение функций пакетов расширения, вошедших в Mathematica 6, намного меньше, чем у функций, вошедших в ядро системы. Это, пожалуй, единственный и, вероятно временный, недостаток новейшей версии системы.

2.1.8. Концепция динамического изменения переменных в Mathematica 6

Язык программирования Mathematica имеет такие хорошо известные объекты, как *переменные*. Мы рассмотрим их подробно в дальнейшем, хотя уже не раз применяли эти объекты ввиду их интуитивной понятности. Пока же отметим, что переменные имеют имена-идентификаторы и им можно присваивать те или иные численные, строковые или символьные значения. При этом переменные делятся на глобальные (их значения действуют во всем ноутбуке) и локальные (их область действия ограничена заданными программно процедурами или функциями со списками параметров – формальными переменными).

Пока ограничимся обсуждением *глобальных переменных*. Таким переменным можно присвоить значение в любом месте ноутбука, и это значение сохранится в последующих частях ноутбука вплоть до очередного присваивания. Однако часто возникает необходимость изменить значение переменной так, чтобы новое значение сохранялось как после момента присваивания, так и до него. Особенно ценным была бы такая возможность с применением средств динамического изме-

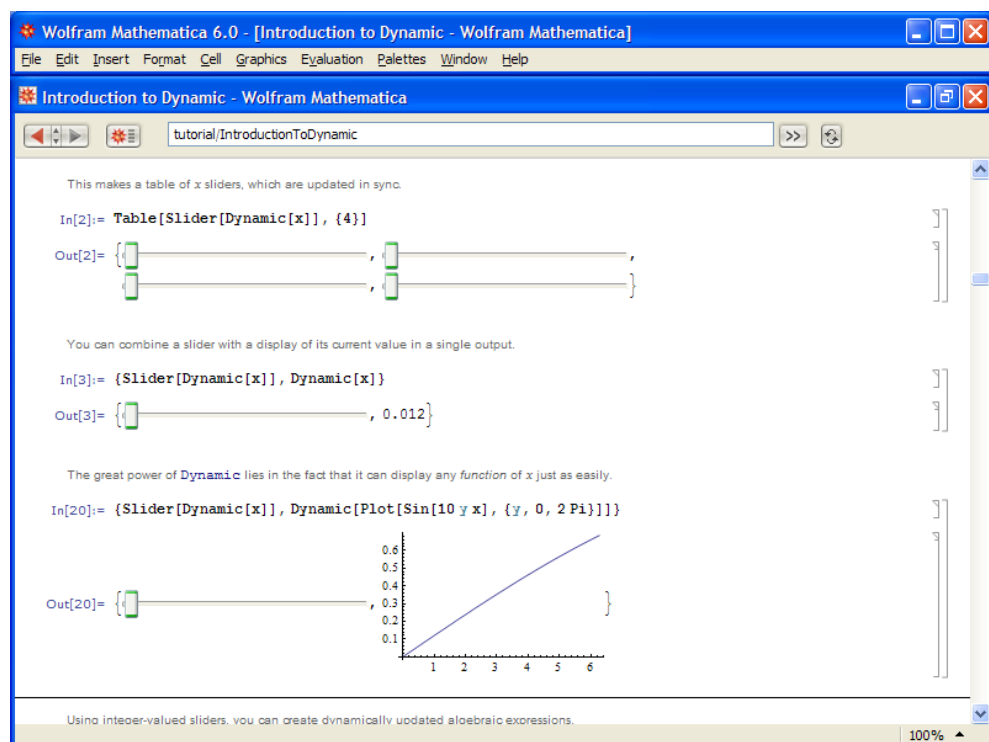


Рис. 2.2. Динамическое изменение значения переменной x (случай 1)

нения значений глобальных переменных. Именно эти средства и введены в систему Mathematica 6. Они реализованы функцией-признаком **Dynamic**.

Применение функции **Dynamic** иллюстрирует ноутбук (рис. 2.2). На нем представлены: построение таблицы с четырьмя слайдерами, одиночный слайдер и слайдер с графиком синусоидальной функции. Везде переменная x указана как параметр функции **Dynamic**. Это означает, что изменение положения движка любого слайдера (или просто изменение значений переменной) автоматически ведет к изменению ее значения выше и ниже точки присваивания переменной заданного значения. Другими словами, это означает, что положение движков всех слайдеров задается изменением движка любого слайдера. На это указывает идентичность положения движков всех слайдеров.

На рис. 2.3 показан второй случай – перемещение движка нижнего слайдера, вызывающего перестроение графика синусоидальной функции под ним. Нетрудно заметить, что заданное положение слайдера (и значение переменной x) автоматически переносится на вышестоящие ячейки ноутбука.

Возможности новой концепции динамического и согласованного изменения переменных еще предстоит осмыслить и научиться ее применять. Владеющим ан-

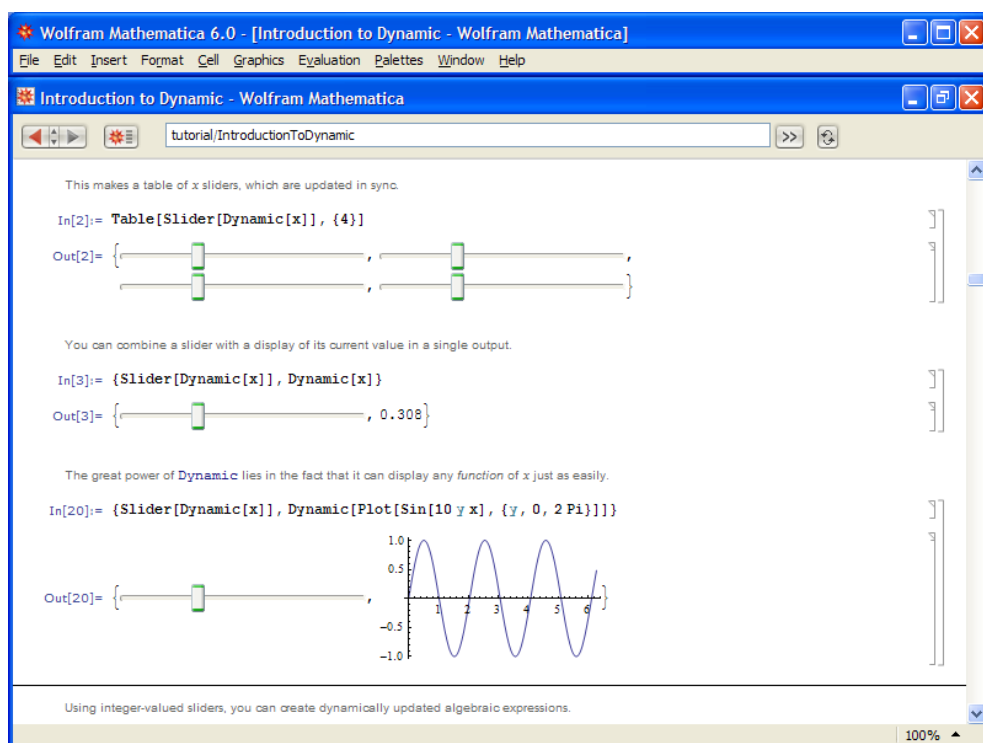


Рис. 2.3. Динамическое изменение значения переменной x (случай 2)

глийским языком рекомендуется изучить введение в эту концепцию (Introduction to Dynamic), которое есть в справке.

2.2. Функции символьных вычислений

2.2.1. Понятие о символьных (аналитических) вычислениях

Символьные операции – это то, что кардинально отличает систему Mathematica (и подобные ей системы компьютерной математики) от систем для выполнения численных расчетов, например таких, как табличные процессоры Excel. Системы с символьными операциями часто именуются также *системами компьютерной алгебры*, что, однако, не вполне отражает куда большие возможности систем класса Mathematica.

Ниже рассмотрены некоторые простые примеры применения символьной математики, которые реализуются в системе Mathematica. При символьных операциях, называемых также аналитическими, задания на вычисление подаются в виде символьных (формульных) выражений, и результаты вычислений также получаются в символьном виде. Численные результаты при этом являются частными случаями результатов символьных вычислений.

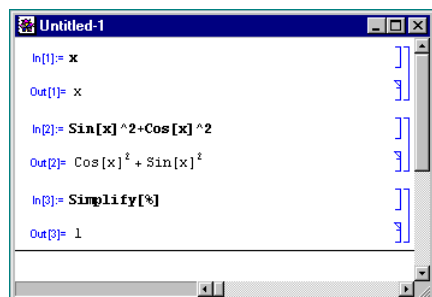


Рис. 2.4. Система Mathematica вычисляет значение $\sin(x)^2 + \cos(x)^2$

К примеру, попытка вычислить в общем виде выражение $\sin(x)^2 + \cos(x)^2 = 1$ с помощью численных математических систем или программ на обычных языках программирования к успеху не приведет. Вместо ожидаемого результата появится сообщение об ошибке вида: «Переменная x не определена!». Компьютер будет ждать ввода конкретного значения для x . Так будет независимо от того, запрограммировали вы вычисления на простеньком Бейсике или языке профессионалов-программистов C++. И лишь системы символьной математики при вычислениях дадут долгожданное и абсолютно точное значение 1 (рис. 2.4).

При рассмотрении даже простейшего примера, показанного на рис. 2.2, можно сделать несколько важных выводов. Прежде всего, видно, что вывод неопределенной переменной x дает саму переменную. Функции $\sin(x)$ и $\cos(x)$ в системе Mathematica обозначаются как **Sin[x]** и **Cos[x]**. Это соответствует двум важным особенностям системы: имена функций обычно начинаются с заглавной буквы, а параметры функции задаются в квадратных, а не круглых скобках. Последние используются для конструирования выражений и задания приоритета операций, например, $1 + (2 + 3 * 4)$.

Само по себе выражение $\sin(x)^2 + \cos(x)^2$ после ввода просто повторяется, а для его вычисления используется функция **Simplify** (упростить), аргументом которой является знак %, означающий подставку предшествующего выражения. Два знака % можно использовать для подстановки предшествующего предшествующему выражению и т.д.

Обратите внимание на то, что система выделяет области ввода определителем In[N], а *области вывода* – определителем Out[N], где N – автоматически представляемый номер строки. Кроме того, в левой части отображаются квадратные скобки с особыми признаками, которые будут описаны позже. Далее мы, как правило, будем опускать определители и представлять документы в упрощенной и более компактной форме. Например, представленный на рис. 2.1 документ может быть записан в виде:

```
x
x
Sin[x]^2+Cos[x]^2
Cos[x]2+Sin[x]2
Simplify[%]
1
```

Здесь входные выражения задаются жирным прямым шрифтом, а выходные не жирным прямым шрифтом. При этом выходные выражения имеют обычный (в терминах системы Mathematica стандартный) вид, присущий математическим формулам. Все такие выражения в книге представлены путем копирования областей ввода и вывода в текст с помощью *буфера*. Технология такого копирования общеизвестна.

Ячейки нумеруются по мере их использования. При этом можно с конца документа вернуться к его началу или середине и, изменив содержимое ранее использованных ячеек, снова выполнить вычисления. При этом ячейки меняют номера. При загрузке файла ячейки перенумеровываются в строго последовательном порядке. Таким образом, номера ячеек не являются жестко фиксированными и являются сугубо техническим средством. Это говорит в пользу отказа от вывода определителей ячеек.

2.2.2. Диагностика ошибок

При обычной работе с системой действуют средства диагностики синтаксических ошибок. Если даже вводится неверное выражение, но с правильным синтаксисом, то система не вычисляет его, а просто повторяет с выводом предупреждающего сообщения. Например:

```
2sin(Pi)
General ::spell1 :
Possible spelling error : new symbol name "sin" is
similar to existing symbol "Sin". More...
2 π sin
```

Здесь вместо 2 Sin[Pi] введено 2sin(Pi). Система поняла sin как имя переменной, но подсказала, что правильное имя функции Sin. Отсутствие пробела между

2 и sin исправлено, а (Pi) выведено без круглых скобок. Правильное задание выражения дает корректное вычисление:

2*Sin[Pi]

0

Вот пример реакции на очевидную синтаксическую ошибку – отсутствие в выражении при его вводе закрывающей круглой скобки:

1+(2+3

Syntax ::bktmcp :

Expression "(2+3" has no closing ")". More...

1+(2+3

А вот характерный пример математической ошибки – деления числа 1 на 0:

1/0

Power ::infty : Infinite expression $\frac{1}{0}$ encountered. More...

ComplexInfinity

Сообщения об ошибках выводятся красным цветом. Поскольку описать все возможные ошибки при вычислениях практически невозможно, можно дать лишь пару рекомендаций: необходимо внимательно следить за сообщениями об ошибках и, хотя бы по минимуму, владеть чтением фраз на английском языке.

2.2.3. Простые примеры из математического анализа

Разумеется, роль систем символьной математики далеко не исчерпывается подтверждением указанного общеизвестного тождества. Эти системы способны преобразовывать сложнейшие алгебраические выражения, находить аналитические решения сложных систем линейных, нелинейных и дифференциальных уравнений, манипулировать со степенными многочленами, вычислять производные и интегралы, анализировать функции и находить их пределы и т.д. [34-46]. Это видно из следующих примеров для системы Mathematica:

D[x^n,x]

$n x^{-1+n}$

Integrate[x^n,x]

x^{1+n}

$1+n$

D[%,x]

x^n

Solve[a*x^2+b*x+c==0,x]

$\left\{ \left\{ x \rightarrow \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \left\{ x \rightarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right\} \right\}$

Series[Sin[x],{x,0,7}]

$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + O[x]^8$

В этих примерах функция **D** (как исключение из правил, обозначенная одной буквой) вычисляет производную, функция **Integrate** – интеграл, функция **Solve** решает нелинейное уравнение (в данном случае квадратное), а функция **Series** разлагает выражение в ряд относительно заданной переменной и при заданных начальном значении переменной и максимальной степени ряда. В фигурных скобках задаются списки некоторых входных и выходных параметров (аргументов).

Системы символьной математики по существу являются справочниками по многим специальным и иным функциям. При этом они способны давать результаты вычислений в виде специальных функций, что демонстрируют следующие примеры:

```
Sum[1/k^9, {k, 1, n}]
HarmonicNumber[n, 9] + PolyGamma[8, 1] + Zeta[9]
                        40320
Integrate[Log[x]*Exp[-x^4], {x, 0, Infinity}]
- 1/32 Gamma[1/4] (2 EulerGamma + pi + Log[64])
DSolve[y''[t] + y'[t] + y[t]/t == 0, y[t], t]
{{y[t] -> e^-t t C[1] - e^-t C[2] (e^t - t ExpIntegralEi[t])}}
```

Здесь специальные функции получаются в результате символьного вычисления суммы, символьного интегрирования и решения в аналитическом виде дифференциального уравнения. Соответствующие функции будут более подробно описаны в дальнейшем.

Впрочем, уже из этих примеров видны определенные тонкости записи входных выражений, которые определяются совокупностью правил их ввода, т.е. *синтаксисом входного языка* системы или (более строго) *языка программирования* системы.

Даже интуитивно ясно, что осуществление символьных операций – процесс более тонкий и сложный, чем реализация численных расчетов. К тому же известно, что таблицы только производных, интегралов и формул преобразований занимают многие тома объемных книг. Все это означает, что высокая эффективность символьных операций реальна только при их реализации на современных высокопроизводительных ПК. Неслучайно системы символьной математики получили серьезное развитие лишь в последний десяток лет и относятся к средствам искусственного интеллекта.

2.2.4. Точная арифметика

Точная арифметика (или арифметика произвольной точности) – одна из важных областей применения систем символьной математики. Такие вычисления часто используются в теоретико-числовых расчетах, криптографии и в других специальных разделах математики.

Примеры точных вычислений с разумным для их демонстрации числом верных цифр результатов приведены ниже:

К сожалению, нередко в математике результирующие выражения быстро нарастают по сложности при, казалось бы, незначительном усложнении или просто из-

менении решаемых задач. Покажем это на примере решения одной из самых часто встречаемых задач – поиска в аналитическом виде корней алгебраического уравнения с целыми степенями членов.

К примеру, многие из нас прекрасно помнят формулы для корней квадратного уравнения, которые даются еще в школе (их в точности воспроизвела Mathematica в одном из примеров, приведенных ранее). Однако едва ли кто вспомнит по памяти формулы аналитического решения кубического уравнения общего вида. Зато система Mathematica играючи справляется с этой задачей:

Solve[a*x^3+b*x^2+c*x+d==0,x]

$$\left\{ \left\{ x \rightarrow -\frac{b}{3a} - \frac{2^{1/3} (-b^2 + 3ac)}{3a \left(-2b^3 + 9abc - 27a^2d + \sqrt{4(-b^2 + 3ac)^3 + (-2b^3 + 9abc - 27a^2d)^2} \right)^{1/3}} + \left(-2b^3 + 9abc - 27a^2d + \sqrt{4(-b^2 + 3ac)^3 + (-2b^3 + 9abc - 27a^2d)^2} \right)^{1/3} \right\}^{1/3} \right\},$$

$$\left\{ x \rightarrow -\frac{b}{3a} + \frac{(1 + i\sqrt{3})(-b^2 + 3ac)}{3 \cdot 2^{2/3} a \left(-2b^3 + 9abc - 27a^2d + \sqrt{4(-b^2 + 3ac)^3 + (-2b^3 + 9abc - 27a^2d)^2} \right)^{1/3}} - \frac{1}{6 \cdot 2^{1/3} a} \left((1 - i\sqrt{3}) \left(-2b^3 + 9abc - 27a^2d + \sqrt{4(-b^2 + 3ac)^3 + (-2b^3 + 9abc - 27a^2d)^2} \right)^{1/3} \right) \right\},$$

$$\left\{ x \rightarrow -\frac{b}{3a} + \frac{(1 - i\sqrt{3})(-b^2 + 3ac)}{3 \cdot 2^{2/3} a \left(-2b^3 + 9abc - 27a^2d + \sqrt{4(-b^2 + 3ac)^3 + (-2b^3 + 9abc - 27a^2d)^2} \right)^{1/3}} - \frac{1}{6 \cdot 2^{1/3} a} \left((1 + i\sqrt{3}) \left(-2b^3 + 9abc - 27a^2d + \sqrt{4(-b^2 + 3ac)^3 + (-2b^3 + 9abc - 27a^2d)^2} \right)^{1/3} \right) \right\} \right\}$$

Заметим, что знак = в Mathematica используется для оператора присваивания переменным заданных значений, например **x=2** или **a=b=c=0**. Поэтому знак равенства левой и правой частей уравнения задается как **==**.

Полученное выше выражение для корней кубического уравнения впечатляет даже студентов университетов, уже изучивших курс математики в полном объеме. Это блестящий пример эффективного представления справочной информации. Можно пойти чуть дальше и убедиться в том, что Mathematica решает даже подобное уравнение и четвертого порядка:

Solve[a*x^4+b*x^3+c*x^2+d*x+e==0,x];

Получаемое при этом громоздкое решение (просмотрите его сами, убрав точку с запятой) заставит в задумчивости почесать затылок многих любителей математики. Но можно ли продолжать эти вычисления? Увы, классическая математика говорит, что нет! Подобные уравнения порядка выше четвертого современная ма-

тематика в аналитическом виде не решает. Тем не менее, попытаемся вычислить корни алгебраического уравнения пятой степени:

```
Solve[a*x^5+b*x^4+c*x^3+d*x^2+e*x+f==0,x]
{{x -> Root[f + e #1 + d #1^2 + c #1^3 + b #1^4 + a #1^5 &, 1]},
 {x -> Root[f + e #1 + d #1^2 + c #1^3 + b #1^4 + a #1^5 &, 2]},
 {x -> Root[f + e #1 + d #1^2 + c #1^3 + b #1^4 + a #1^5 &, 3]},
 {x -> Root[f + e #1 + d #1^2 + c #1^3 + b #1^4 + a #1^5 &, 4]},
 {x -> Root[f + e #1 + d #1^2 + c #1^3 + b #1^4 + a #1^5 &, 5]}}
```

А вот и сюрприз: Mathematica не только не отказалась решать эту задачу, но даже подсказала путь ее решения с помощью функции вычисления корней **Root** степенных многочленов пониженной степени.

В приведенных выше примерах мы сталкиваемся с одной из самых серьезных проблем символьной математики – *разбуханием результатов аналитических преобразований* при порой незначительном усложнении решаемых задач. В приведенных выше примерах это никоим образом не является недостатком систем компьютерной математики, как таковых: просто так нарастает сложность решения данной математической задачи в соответствии с канонами абстрактной математики.

Однако нередко разбухание результатов кроется в сложности алгоритмов, особенно рекурсивных. Современные системы символьной математики способны осуществлять весьма глубокую рекурсию и порою трудно даже предположить, к сколь громоздкому результату и какого вида это, в конечном счете, приведет.

Научные работники и инженеры настолько привыкли к упрощению (порой весьма грубому с точки зрения математиков), что громоздкие решения, получаемые с помощью систем символьной математики, способны их раздражать. Это в определенной мере препятствует применению систем символьной математики на практике.

Однако для частных случаев нередко можно получить вполне сносные по виду решения. Вот пример на решение неполного алгебраического уравнения десятой степени, при котором Mathematica благополучно возвращает все десять корней довольно простого вида:

```
Solve[a*x^10+x^2==0,x]
{{x -> 0}, {x -> 0}, {x -> -((-1)^(1/8)/a^(1/8))}, {x -> (-1)^(1/8)/a^(1/8)}, {x -> -((-1)^(3/8)/a^(1/8))},
 {x -> (-1)^(3/8)/a^(1/8)}, {x -> -((-1)^(5/8)/a^(1/8))}, {x -> (-1)^(5/8)/a^(1/8)}, {x -> -((-1)^(7/8)/a^(1/8))}, {x -> (-1)^(7/8)/a^(1/8)}}
```

Примеры такого рода можно встретить повсеместно. Есть множество нелинейных алгебраических или дифференциальных уравнений обманчиво простого вида, но имеющих сложнейшие решения или не имеющих их вовсе. Так что, если система символьной математики не находит ответ, это зачастую не является признаком ее слабости – может быть решения не существует вообще? Порой даже такой отрицательный результат избавляет пользователя от трудоемкого поиска

несуществующих решений «в лоб» и направляет его на поиск обходных, порою весьма ценных и полезных, методов решения.

2.2.6. Проверка результатов вычислений

Если вы получили результат, который не ожидали, не спешите считать его окончательным и тем более новым. Очень редко, но СКМ может выдать и ошибочный результат. Всегда желательно проверить результат ее работы. Покажем, как это делается.

Пусть мы решили некоторое уравнение

```
eqns=x^3-3*x+2==0
```

$$x^3 - 3x + 2 = 0$$

и получили его корни

```
r=Solve[eqns,x]
```

```
{{x→-2},{x→1},{x→1}}
```

Для проверки решения можно использовать операцию подстановки в eqns списка корней r. Эта операция реализуется оператором /., что иллюстрирует следующий пример:

```
eqns/.r
```

```
{True,True,True}
```

Результат этой операции – список из трех символьных констант True (Истинно). Он значит, что решение верно. Кстати, с помощью этой подстановки можно получить истинный список корней:

```
x/.r
```

```
{-2,1,1}
```

В данном случае результат верен. В особо каверзных случаях необходимо ориентироваться на свою интуицию, решение схожей тестовой задачи или решение с помощью других математических систем с иным ядром, например, Maple, Derive или MuPAD.

2.2.7. Удаление введенных в ходе сессии определений

В ряде случаев необходимо удалить введенные в ходе сессии определения, например переменных. К примеру, символьные вычисления возможны только при использовании неопределенных переменных. Для удаления определений используются функции:

- **Clear[symbol1, symbol2,...]** — стирает значения и определения для указанных символов (идентификаторов);
- **Clear["pattern1","pattern2",...]** — стирает значения и определения для всех символов, чьи имена подходят под любой из указанных строковых шаблонов;

- **ClearAll[symbol1, symbol2,...]** — стирает все значения, определения, атрибуты, сообщения и значения, принятые по умолчанию, связанные с указанными символами;
- **ClearAll["pattern1", "pattern2",...]** — стирает все символы, чьи имена буквально подходят к одному из указанных строковых образцов;
- **ClearAttributes[s, attr]** — удаляет attr из списка атрибутов символа s.

Определения переменных можно удалить также с помощью конструкции вида **x=.**, что эквивалентно **Clear[x]**.

Применение большинства из этих функций полезно разработчику серьезных приложений для систем Mathematica, например новых пакетов расширений и применений системы. В то же время для большинства пользователей вполне достаточны возможности, предоставляемые системой по умолчанию — средства диалога с ее оболочкой и функции **Input** и **Print**.

2.3. Применение образцов

2.3.1. Понятие об образцах

Образцы в системе Mathematica служат для задания выражений различных классов и придания переменным особых свойств, необходимых для создания специальных программных конструкций, таких как функции пользователя и процедуры. Это необычайно гибкое и мощное средство обобщенного представления математических выражений, используемое при любом подходе к программированию.

Признаком образца являются знаки подчеркивания « » (от одного до трех). Они обычно выглядят слитно, так что необходимо внимательно следить за общей длиной символов образцов. Наиболее распространенное применение образцов — указание на локальный характер переменных при задании функций пользователя. Например, функция

```
fsc[x_,y_] := x * Sin[y] + y * Cos[x] + z
```

в списке параметров содержит два образца **x_** и **y_**. В правой части этого выражения переменные **x** и **y**, связанные с образцами **x_** и **y_**, становятся локальными переменными, тогда как переменная **z** будет глобальной переменной. Обратите особое внимание на то, что символы образцов используются только в списках параметров, в правой части выражений они уже не используются.

2.3.2. Задание свойств функций с помощью образцов

Образцами можно задавать некоторые общие свойства функций. Например,

```
f[x_,x_] := p[x]
```

означает, что функция **f** двух идентичных аргументов становится тождественной функции **p[x]**. Следовательно:

```
f[a,a] + f[a,b]
```

даст выход в виде:

```
f[a,b] + p[a],
```

a

```
f[a^2 - 1, a^2 - 1]
```

даст выход

```
p[-1 + a^2]
```

Вообще говоря, вычисление таких функций, как факториал в Mathematica, можно запрограммировать более чем десятком способов. Вот несколько самых интересных реализаций этой функции:

```
f[n_] := n!
```

```
f[n_] := Gamma[n+1]
```

```
f[n_] := n*f[n-1]; f[0]=1; f[1]=1;
```

```
f[n_] := Product[i, {i, 1, n}]
```

```
f[n_] := Module[{t=1}, Do[t=t*i, {i, 1, n}]; t]
```

```
f[n_] := Module[{t=1}, For[i=1, i<=n, i++, t*=i]; t]
```

```
f[n_] := Fold[Times, 1, Range[n]]
```

Все их можно проверить по примеру:

```
{f[0], f[1], f[5], f[10]}
```

```
{1, 1, 120, 3628800}
```

2.3.3. Задание в образцах типов данных

С образцом можно указывать его тип данных:

- **x_Integer** - образец целочисленный,
- **x_Real** - образец с действительным значением,
- **x_Complex** - образец с комплексным значением,
- **x_h** - образец с заголовком h (от слова head – голова).

Задание с помощью образцов типов данных делает программы более строгими и наглядными и позволяет избежать ошибок, связанных с несоответствием типов данных.

2.3.4. Типы образцов

В системе Mathematica используются следующие типы образцов:

Обозначение	Назначение образца
_	Любое выражение
x_	Любое выражение, представленное именем x
x:pattern	Образец, представленный именем x
pattern ? test	Возвращает True, когда test применен к значению образца
_h	Любое выражение с заголовком h

Обозначение	Назначение образца
<code>x_h</code>	Любое выражение с заголовком <code>h</code> , представленное именем <code>h</code>
<code>--</code>	Любая последовательность с одним и более выражениями
<code>----</code>	Любая последовательность с нулем или более выражений
<code>x_</code> или <code>x__</code>	Последовательности выражений, представленные именем <code>x</code>
<code>_h</code> или <code>h__</code>	Последовательности выражений, каждое с заголовком <code>h</code>
<code>x_h</code> или <code>x__h</code>	Последовательности выражений с заголовком <code>h</code> , представленные именем <code>x</code>
<code>x_:v</code>	Выражение с определенным значением <code>v</code>
<code>x_h:v</code>	Выражение с заголовком <code>h</code> и определенным значением <code>v</code>
<code>x_.</code>	Выражение с глобально заданным определенным значением <code>v</code>
<code>Optional[x_h]</code>	Выражение с заголовком <code>h</code> и с глобально заданным значением
<code>pattern..</code>	Образец, повторяемый один или более раз
<code>pattern...</code>	Образец, повторяемый ноль или более раз

Еще раз отметим, что символ «`_`» в образцах может иметь одинарную, двойную или тройную длину. Надо следить за правильностью его применения, поскольку эти применения различаются по смыслу. Образцы широко применяются при задании функций пользователя и в пакетах расширения системы.

2.4. Основы функционального программирования в среде Mathematica

2.4.1. Суть функционального программирования

Суть функционального программирования заключается в использовании в ходе решения задач только функций. При этом возможно неоднократное вложение функций друг в друга и применение функций различного вида – в Mathematica это именуют понятием *функции в функции*. В ряде случаев, особенно в процессе символьных преобразований, происходит взаимная рекурсия множества функций, сопровождаемая почти неограниченным углублением рекурсии и нарастанием сложности обрабатываемых системой выражений.

Понятие функции ассоциируется с обязательным возвратом некоторого значения в ответ на обращение к функции по ее имени с указанием в квадратных скобках аргументов (параметров). Возврат функциями некоторых значений позволяет применять их наряду с операторами для составления математических выражений.

Функции подразделяются на встроенные в ядро системы внутренние функции и функции, заданные пользователем. Примером первых могут быть **Sin[x]**, **BesselI[n,x]** и т.д. Mathematica содержит множество таких функций, охватывающих практически все широко распространенные элементарные и специальные

математические функции. Есть возможность и создания функций со специальными свойствами – *чистых* (pure functions) и *анонимных функций*.

2.4.2. Функции пользователя

Хотя в системах Mathematica около тысячи встроенных функций, любому пользователю рано или поздно может потребоваться создание какой-либо своей функции. Кажется естественным задать ее по правилам, принятым во многих языках программирования. Например, функцию для возведения x в степень n можно было бы определить так:

```
powerxn[x,n] := x^n
```

Однако оказывается, что такая функция работать отказывается:

```
powerxn[a,b]
powerxn[a,b]
```

Причина этого в том, что в системе Mathematica символы x и n являются обычными символами, не наделенными особыми свойствами. Будучи использованными в качестве параметров функции, они не способны воспринимать *формальные параметры* (в нашем случае a и b).

Для того чтобы функция пользователя была полноценной, в списке параметров необходимо использовать образцы в виде переменных, но имеющие после своих имен символы подчеркивания. Образцы способны быть формальными параметрами функций и воспринимать значения *фактических параметров*. Таким образом, правильной будет запись функции пользователя в виде:

```
powerxn[x_,n_] := x^n
```

Теперь вычисление по заданной функции пользователя пройдет гладко, причем как в численном, так и в символьном виде:

```
powerxn[2,3]
8
powerxn[a,b]
a^b
```

Рассмотрим еще один простой пример, в котором задана функция **scn**[x,n], вычисляющая сумму синуса в степени n и косинуса в степени n :

```
scn[x_,n_] := Sin[x]^n + Cos[x]^n
scn[1,2]
Cos[1]^2 + Sin[1]^2
scn[x,2]
Cos[x]^2 + Sin[x]^2
scn[x,n]
Cos[x]^n + Sin[x]^n
```

В этом простейшем примере результат вычислений – есть возвращаемое функцией **scn** символьное значение. Можно также задать функцию пользователя, содержащую несколько выражений, заключив их в круглые скобки:

```
f[x_] := (t = (1+x)^2; t = Expand[t])
```

Переменные списка параметров, после имени которых стоит знак «_», являются локальными в теле функции или процедуры с параметрами. На их место подставляется фактическое значение соответствующего параметра, например:

f[a+b]

$1 + 2a + a^2 + 2b + 2ab + b^2$

t

$1 + 2a + a^2 + 2b + 2ab + b^2$

Обратите внимание на то, что переменная *t* в функции *f* является глобальной. Это поясняет результат последней операции. Применение глобальных переменных в теле функции вполне возможно, но создает так называемый *побочный эффект*, в данном случае меняет значение глобальной переменной *t*. Для устранения побочных эффектов необходимо использовать образцы и другие специальные способы задания функций, описанные ниже.

Итак, можно сформулировать ряд правил для задания функций пользователя:

- такая функция имеет идентификатор-имя, который должен быть уникальным и достаточно понятным;
- в списке параметров функции, размещенном в квадратных скобках после идентификатора, должны использоваться образцы переменных, а не просто имена переменных;
- переменные в теле функции, указанные в списке переменных-образцов являются локальными;
- может использоваться отложенное := или неотложенное = присвоение;
- тело функции может содержать несколько выражений, заключенных в круглые скобки, при этом возвращается значение последнего выражения;
- переменные образцов в списке параметров являются локальными и действуют в пределах только тела функции;
- в теле функции могут использоваться глобальные переменные, но при этом возможны побочные эффекты.

Параметрами функций могут быть списки, при условии допустимости их комбинации. Например, допустимо задать *x* списком, а *n* – переменной или числом:

powerxn[{1,2,3,4,5},z]

{1, 2^z, 3^z, 4^z, 5^z}

powerxn[{1,2,3,4,5},2]

{1, 4, 9, 16, 25}

После своего задания функции пользователя могут использоваться по тем же правилам, что и встроенные функции (см. раздел 9.2).

2.4.3. Задание чистых функций

Иногда может потребоваться задание некоторой функции только в момент ее создания. Эта функция представляется только выражением без имени, отсюда и ее название *чистая функция*. Для создания такого объекта служит встроенная функция **Function**, используемая в виде:

- **Function[body]** – создает чистую функцию с телом body;
- **Function[{x},body]** – создает чистую функцию параметра x с телом body;
- **Function[{x1,x2,...},body]** – создает чистую функцию ряда параметров x1, x2,... с телом body.

Для вычисления созданной таким образом функции после нее задается список параметров в квадратных скобках. Например, для взятой в качестве примера функции ее можно задать и использовать следующим образом:

```
Function[{x,n},x^n]
Function[{x, n}, x^n]
% [2,3]
8
```

Чистую функцию можно легко превратить в обычную функцию пользователя, что показывает следующий пример:

```
fun=Function[{x,n},x^n]
Function[{x, n}, x^n]
fun[2,3]
8
```

2.4.4. Анонимные функции

Предельно компактную форму задания функций имеют так называемые *анонимные функции*. Они не имеют ни названия, ни обычного определения и задаются только выражениями специального вида. В этом выражении вместо переменных используют обозначения # (для одной переменной), #1, #2,... для ряда переменных. Завершается тело функции символом &. Если необходимо вычислить функцию, то после ее записи в квадратных скобках указывается список фактических параметров.

Для нашего примера такая функция выглядит так:

```
#1^#2&[2,3]
8
#1^#2&[y,z]
yz
```

С помощью анонимных функций нетрудно создать обычные функции пользователя:

```
f[x_,y_]=#1^#2&[x,y]
xy
f[2,3]
8
```

Несмотря на то, что применение анонимных функций открывает возможности к компактному заданию многих функций, эта форма задания функций едва ли интересна для большинства читателей; они наверняка предпочтут пусть немного более длинное, но совершенно очевидное задание функций в других формах.

2.4.5. Суперпозиция функций

При функциональном программировании часто используется суперпозиция функций. Для ее реализации используются функции:

- **Nest[expr,x,n]** – применяет выражение (функцию) к заданному аргументу x n раз.
- **NestList[f,x,n]** – возвращает список приложений функции f к заданному аргументу x $n+1$ раз.
- **Fold[f,x,list]** – дает следующий элемент в **FoldList[f,x,list]**.
- **FoldList[f,x,{a,b,...}]** – дает $\{x, f[x, a], f[f[x, a], b], j\}$.
- **ComposeList[{f,f₂,...},x]** – генерирует список в форме $\{x, a[x], a[a[x]], \dots\}$.

Примеры, иллюстрирующие действие этих функций, представлены ниже:

```
Clear[f]
Nest[f,x,5]
f[f[f[f[f[f[x]]]]]]
Nest[Exp[x],x,5]
ex[ex[ex[ex[ex[x]]]]]
NestList[f,x,3]
{x, 1+2 x+x2, 4+8 x+8 x2+4 x3+x4,
 25+80 x+144 x2+168 x3+138 x4+80 x5+32 x6+8 x7+x8}
Fold[f,x,{1,2,3}]
x6
FoldList[f,x,{1,2,3}]
{x, x, x2, x6}
ComposeList[{Exp,Ln,Sin},x]
{x, ex, Ln[ex], Sin[Ln[ex]]}
```

2.4.6. Функции FixedPoint и Cath

В функциональном программировании вместо циклов, описанных далее, может использоваться следующая функция:

- **FixedPoint[f,expr]** – вычисляет expr и прикладывает к нему f , пока результат не повторится.
- **FixedPoint[f,expr,SameTest->comp]** – вычисляет expr и прикладывает к нему f , пока два последующих результата не дадут True в тесте.

Пример применения функции FixedPoint:

```
FixedPoint[Function[t, Print[t]; Floor[t/2]], 27]
27
13
6
3
1
0
0
```


Последний результат 0 выводится в отдельной (нумерованной) ячейке вывода и означает завершение процесса итераций – деления t на 2.

Следующий пример показывает, как можно создать цепную дробь с помощью функции **Nest**:

```
Nest[ Function[t, 1/(1+t)], y, 3 ]
```

$$1 + \frac{1}{1 + \frac{1}{1 + y}}$$

Еще одна функция такого рода – это **Catch**:

- **Catch[expr]** – вычисляет expr, пока не встретится Throw[value], затем возвращает value.
- **Catch[expr, form]** – вычисляет expr, пока не встретится Throw[value,tag], затем возвращает value.
- **Catch[expr, form, f]** – возвращает f[value, tag] вместо value.

Ниже представлены некоторые конструкции циклов с оператором **Catch**:

```
Catch[ Throw[ x, y ], y, fun ]
fun[x,y]
Catch[ NestList[1/(# + 1)&, -3, 5] ]
{-3, -1/2, 2/3, 3/4, 4/7}
Catch[ NestList[1/(# + 1)&, -3., 5] ]
{-3., -0.5, 2., 0.333333, 0.75, 0.571429}
```

2.4.7. Реализация рекурсивных и рекуррентных алгоритмов

Важное место в решении многих математических задач занимают реализации *рекурсивных* и *рекуррентных* алгоритмов. Рассмотрим, как это делается с помощью описанных выше функций.

Вначале рассмотрим типичный пример реализации итерационного рекуррентного алгоритма вычисления квадратного корня из выражения $f(x)$ при начальном значении $x_0=a$, по следующим формулам метода Ньютона:

$$x_0=a \quad \text{и} \quad x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1}).$$

Эту функцию можно записать следующим образом:

```
newtoniter[f_,x0_,n_] := Nest[ (#-f[#]/f' [#])&,N[x0],n]
```

Тогда вычисления корня из выражения e^x-2 с начальным приближением $x_0=0.5$ и количеством итераций n можно организовать с помощью функций **Nest** и **NestList**:

```
newtoniter[Function[{x},Exp[x]-2.0],0.5,5]
0.693147
newtoniter[Function[{x},Exp[x]-2.0],0.5,#]&/@Range[5]
```

```
{0.713061,0.693344,0.693147,0.693147,0.693147}
newtoniter1[f_,x0_,n_]:=NestList[(#-f[#]/f'[#])&,N[x0],n]
newtoniter1[Function[{x},Exp[x]-2.0],0.5,5]
{0.5,0.713061,0.693344,0.693147,0.693147,0.693147}
```

В первом случае возвращается последний результат, а в других – все промежуточные. Функция **FixedPoint** позволяет осуществлять итерацию до тех пор, пока результат не перестанет изменяться (с машинной точностью). Это иллюстрирует следующий пример:

```
newtonfp[f_, init_, options___] := FixedPoint[#1 -  $\frac{f[\#1]}{f'[\#1]}$  &, init, options];
newtonfp[Function[{x}, Exp[x]-2.0], 0.5, 5]
0.693147
```

Рекурсивные алгоритмы основаны на том, что при вычислении некоторой функции в ее теле происходит обращение к ней же самой. Mathematica допускает такую возможность. Типичный пример этого – вычисление факториала по формуле $N! = N \cdot (N-1)!$. Некоторые алгоритмы символьных вычислений основаны на глубокой рекурсии, что может в некоторых случаях вызвать переполнение памяти.

2.5. Основы процедурного программирования

2.5.1. Однострочные процедуры и их задание

В основе *процедурного программирования* лежит понятие процедуры как законченного программного модуля и типовых средств управления: циклов, условных и безусловных выражений и т.д. Процедурный подход самый распространенный в программировании, и разработчики Mathematica были вынуждены обеспечить его полную поддержку. Хотя, программирование систем Mathematica и в этом случае остается функциональным, поскольку элементы процедурного программирования заданы в виде функций. Однако появляется возможность применения традиционных средств программирования – условных выражений, циклов, процедур и т.д.

Процедуры являются полностью самостоятельными программными модулями, задаются своим именем и отождествляются с выполнением некоторой последовательности операций. Они могут быть заданы в одной строке с использованием в качестве разделителя символа «;» (точка с запятой). Вот пример задания однострочной процедуры, отождествленной с именем g :

```
r=(1+x)^2;r=Expand[r];r-1
2 x + x2
```

Обратите внимание, что в теле процедуры символ g используется как вспомогательная переменная. Эта процедура возвращает символьное выражение **Expand**[(1+x)²] - 1.

В общем случае в теле процедуры могут быть произвольные выражения, разумеется, с синтаксисом, присущим языку программирования системы. Процедура может не возвращать никаких значений, а просто выполнять определенный комплекс операций. Область записи подобных элементарных процедур ограничена ячейкой (строкой) ввода.

Для задания процедуры со списком локальных переменных $\{a, b, \dots\}$ и телом `proc` может использоваться функция

Module[$\{a, b, \dots\}, \text{proc}$]

С применением этой функции мы столкнемся ниже.

2.5.2. Блоки для задания процедур

Для создания полноценных процедур и функций, которые могут располагаться в любом числе строк, может использоваться базовая структура-блок:

- **Block**[$\{x, y, \dots\}, \text{procedure}$] – задание процедуры с декларацией списка локальных переменных x, y, \dots
- **Block**[$\{x = x0, y = y0, \dots\}, \text{procedure}$] – задание процедуры с декларацией списка переменных x, y, \dots с заданными начальными значениями.

Пример использования базовой структуры:

```
g[x_] := Block[{u}, u = (1+x)^2; u=Expand[u] ]
g[a+b]
1 + 2 a + a^2 + 2 b + 2 a b + b^2
u
u
u=123456;g[2]
9
u
123456
```

Обратите внимание, что последние действия указывают на то, что переменная u , введенная в тело базовой структуры, является действительно локальной переменной, и присвоение ей символьного выражения $(1+x)^2$ в теле блока игнорируется вне блока. Если переменная u до применения в функции была не определена, то она так и остается неопределенной. А если она имела до этого некоторое значение (например, 123456 в нашем случае), то и по выходе из процедуры она будет иметь это значение.

2.6. Организация циклов

2.6.1. Для чего нужны циклы

Многие задачи в системе Mathematica решаются с использованием *линейных алгоритмов и программ*. Они могут быть представлены непрерывной цепочкой выражений, выполняемых последовательно от начала до конца.

Однако в большинстве случаев серьезные вычисления базируются на использовании *циклических и разветвленных алгоритмов и программ*. При этом, в зависимости от промежуточных или исходных данных, вычисления могут идти по разным ветвям программы, циклически повторяться и т.д. Для реализации разветвленных программ язык программирования должен содержать управляющие структуры, т.е. специальные конструкции языка, реализующие в программах ветвление. Они используются при различных методах программирования, в том числе при процедурном и функциональном программировании.

2.6.2. Циклы типа *Do*

К важнейшим управляющим структурам в языках программирования относят циклы. С их помощью осуществляется циклическое исполнение некоторого выражения *expr* заданное число раз. Это число нередко определяется значением некоторой управляющей переменной (например, *i*, *j* и т.д.), меняющейся либо с шагом +1, либо от начального значения *imin* до конечного значения *imax* с шагом *di*. Циклы могут быть одинарными или множественными – вложенными друг в друга. Последние используют ряд управляющих переменных.

Такого рода циклы организуются с помощью функции *Do*:

- **Do[expr, {imax}]** – выполняет *imax* раз вычисление *expr*.
- **Do[expr, {i, imax}]** – вычисляет *expr* с переменной *i*, последовательно принимающей значения от 1 до *imax* (с шагом 1).
- **Do[expr, {i, imin, imax}]** – вычисляет *expr* с переменной *i*, последовательно принимающей значения от *imin* до *imax* с шагом 1.
- **Do[expr, {i, imin, imax, di}]** – вычисляет *expr* с переменной *i*, последовательно принимающей значения от 1 до *imax* с шагом *di*.
- **Do[expr, {i, imin, imax}, {j, jmin, jmax}, ...]** – вычисляет *expr*, организуя ряд вложенных циклов с управляющими переменными *j*, *i* и т.д.

Примеры организации цикла **Do** и его исполнения представлены ниже:

```
Do[Print[«hello»], {5}]
hello
hello
hello
hello
hello
Do[Print[i], {i, 3}]
1
2
3
Do[Print[i], {i, 5, 8}]
5
6
7
8
Do[Print[i], {i, 0, 1, 0.25}]
```

```
0
0.25
0.5
0.75
1.
```

Нетрудно убедиться в том, что переменная *i* в теле цикла – итератор – является локальной, и по выходе из цикла ее значение остается тем же, что и до входа:

```
i=2
2
Do[Print[i], i, 1, 5]
1
2
3
4
5
i
2
```

Вся программа с циклом является содержанием одной ячейки, и ее листинг охвачен квадратной скобкой. Для иллюстрации вывода здесь использована команда **Print** в теле цикла. Нетрудно заметить, что управляющая переменная цикла может иметь как целочисленные, так и вещественные значения. Возможность организации цикла в цикле иллюстрируется следующим примером:

```
Do[Do[Print[i, «    », j, «    «, i+j], {j, 1, 3}], {i, 1, 3}];
1      1      2
1      2      3
1      3      4
2      1      3
2      2      4
2      3      5
3      1      4
3      2      5
3      3      6
```

Здесь используются два цикла с управляющими переменными *i* и *j*. Командой **Print** выводятся значения переменных *i* и *j*, а также их суммы *i+j*.

Следующий пример показывает задание процедуры с циклом **Do** для задания функции, вычисляющей *n*-ое число Фибоначчи:

```
fibonacci[(n_Integer)?Positive] :=
Module[{fn1 = 1, fn2 = 0},
  Do[{fn1, fn2} = {fn1 + fn2, fn1}, {n - 1}]; fn1]
fibonacci[10]
55
fibonacci[100]
354224848179261915075
fibonacci[-10]
fibonacci[-10]
```

Обратите внимание на применение в этом примере функции **Module**. Она создает программный модуль с локальными переменными (в нашем случае `fn1` и `fn2`), в котором организовано рекуррентное вычисление чисел Фибоначчи.

Наконец, последний пример показывает применение цикла **Do** для создания цепной дроби:

```
x = y; Do[x = 1/(1 + k x) , k, 2, 8, 2]; x
```

$$1 + \frac{1}{1 + \frac{8}{1 + \frac{6}{1 + \frac{4}{1 + 2y}}}}$$

2.6.3. Циклы типа *For*

Другой вид цикла **For** реализуется функцией:

```
For[start, test, incr, body]
```

В ней внутренняя управляющая переменная вначале приобретает значение `start`, затем циклически меняется от этого значения до значения `body` с шагом изменения `incr`; и так до тех пор, пока условие `test` не перестанет давать логическое значение `True`. Когда это случится, т.е. `test` даст `False`, цикл заканчивается.

Следующий пример показывает задание простой программы с циклом **For** и результат ее выполнения:

```
Print[<<i    x>>]
For [x=0;i=0,i<4,i++
          [x+=5*i,Print[i,»    <<x]]]

i        x
1        5
2        15
3        30
4        50
Return[x]
Return[50]
```

Программа, приведенная выше, позволяет наблюдать за изменением значений управляющей переменной цикла `i` и переменной `x`, получающей за каждый цикл приращение, равное `5*i`. В конце документа показан пример на использование функции возврата значений **Return**[**x**]. В цикле **For** не предусмотрено задание локальных переменных, так что необходимо следить за назначением переменных, т.к. при использовании глобальных переменных неизбежны побочные эффекты.

2.6.4. Циклы типа *While*

Итак, функция **For** позволяет создавать циклы, которые завершаются при выполнении (эволюции) какого-либо условия. Такие циклы можно организовать и с помощью функции **While**:

- **While[test, expr]** – выполняет *expr* до тех пор, пока *test* дает логическое значение True.

Ниже рассмотрен практический пример организации и использования цикла **While**.

```
i:=1;x:=1;Print[«i    x»];
While[i<5,i+=1;x+=2*i;Print[i,»    «,N[x]]]
i      x
2      5.
3      11.
4      19.
5      29.
Return[x]
Return[50]
```

Циклы типа **While**, в принципе, могут заменить другие, рассмотренные выше, типы циклов. Однако это усложняет запись и понимание программ. Аппарат локальных переменных в этом типе циклов не используется.

2.6.5. Директивы-функции прерывания и продолжения циклов

В указанных типах циклов и в иных управляющих структурах можно использовать следующие директивы-функции:

- **Abort[]** – вызывает прекращение вычислений с сообщением \$Aborted;
- **Break[]** – выполняет выход из тела цикла или уровня вложенности программы, содержащего данный оператор (циклы типа Do, For и While или тело оператора – переключателя Shith). Оператор возвращает Null – значение (без генерации секции выхода).
- **Continue[]** – задает переход на следующий шаг текущего цикла Do, For или While.
- **Interrupt[]** – прерывает вычисления с возможностью их возобновления.
- **Return[]** – прерывает выполнение с возвратом Null.
- **Return[expr]** – прерывает выполнение с выводом значения выражения *expr*.

На рис. 2.5 представлено применение директив **Abort[]** и **Interrupt[]** в середине набора команд. Нетрудно заметить, что директива **Abort[]** просто прерывает выполнение цепочки команд и выводит сообщение **\$Aborted**. А вот директива **Interrupt[]** выводит диалоговое окно, с помощью которого можно либо прервать вычисления, либо продолжить их.

Если продолжить вычисления (нажав кнопку **Continue Evaluation**), то вывод выражений командами **Print** будет продолжен.

Для создания управляющих структур служат также функции, появившиеся в Mathematica 5 и представленные ниже:

Sow[e]	Sow[e, tag]	Sow[e, {tag₁, tag₂, j}]
Reap[expr]	Reap[expr, patt]	Reap[expr, {patt₁, patt₂, j}]
Reap[expr, patt, f]		

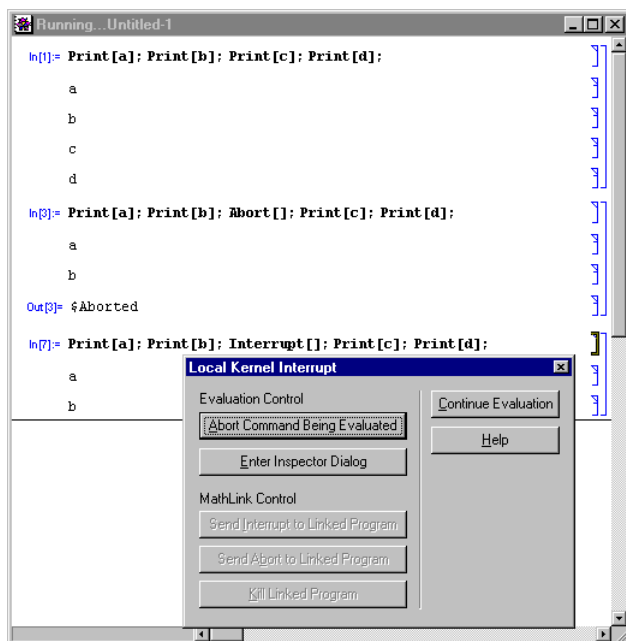


Рис. 2.5 Действие директив **Abort[]** и **Interrupt[]**

Функция **Sow** прикрепляет к выражению метку, которую распознает функция **Reap**, и возвращает соответствующее выражение:

```
Reap[Sow[1];2;Sow[3];Sow[4];5;6;7]
{7,{{1,3,4}}}
Reap[Sow[1,y];Sow[2,x];Sow[3,y],_,Rule]
{3,{y->{1,3},x->{2}}}
```

Эти функции относятся к числу используемых довольно редко. Множество примеров их применения можно найти в справке.

2.7. Условные выражения и безусловные переходы

Для подготовки полноценных программ помимо средств организации циклов необходимы и средства для создания разветвляющихся программ произвольной структуры. Обычно они реализуются с помощью условных выражений, позволяющих в зависимости от выполнения или невыполнения некоторого условия (condition) выполнять те или иные фрагменты программ.

2.7.1. Функция *If*

Как у большинства языков программирования, условные выражения задаются с помощью оператора или функции **If** (Если). Система Mathematica имеет функцию **If**, формы которой представлены ниже:

- **If**[condition, t, f] – возвращает t, если результатом вычисления condition будет True, и даст f, если результат False.
- **If**[condition, t, f, u] – даст u, если в результате вычисления condition не будет получено ни True, ни False.

Следующий пример поясняет задание программной процедуры с циклом **Do**, выход из которой задается с помощью функции **If** и директивы прерывания **Aborted** []:

```
x:=1;Print["i   x"];
Do[{If [i==5,Abort[],None],
i+=1;x+=2*i;Print[i,"   ",N[x]]},
{i,1,100}]
i       x
2       5.
3       11.
4       19.
5       29.
$Aborted
Return[x]
Return[29]
```

Тот же пример, но с применением директивы выхода из цикла **Break**[] в функции **If**, показан ниже:

```
x:=1;Print["i   x"];
Do[{If [i==5,Break[],None],
i+=1;x+=2*i;Print[i,"   ",N[x]]},
{i,1,100}]
i       x
2       5.
3       11.
4       19.
5       29.
Return[x]
Return[29]
```

В данном случае никаких специальных сообщений о выходе из цикла не дается.

Функция **If** обеспечивает ветвление максимум по двум ветвям программы. Для ветвления по многим направлениям можно использовать древовидные структуры программ с множеством функций **If**. Однако это усложняет программы.

2.7.2. Функции-переключатели

Для организации ветвления по многим направлениям в современных языках программирования используются операторы-переключатели. В системе *Mathematica* множественное ветвление организовано с помощью функций **Which** и **Switch**:

- **Which**[*test1*, *value1*, *test2*, *value2*, ...] – вычисляет в порядке следования каждый из *testi*, сразу возвращая именно ту величину из *valuei*, которая относится к первому *testi*, давшему True.
- **Switch**[*expr*, *form1*, *value1*, *form2*, *value2*, ...] – вычисляет *expr*, затем сравнивает его последовательно с каждым *formi*, вычисляя и возвращая то *valuei*, которое соответствует первому совпадению.

Приведем примеры работы функции **Which**:

```
Which[1==2,1,2==2,2,3==3,3]
2
Which[1==2,x,2==2,y,3==3,z]
y
```

Следующие примеры иллюстрируют работу функции **Switch**:

```
Switch[2,1,a,2,b,3,c]
b
Switch[3,1,a,2,b,3,c]
c
Switch[8,1,a,2,b,3,c]
Switch[8,
1,a,
2,b,
3,c]
```

Обратите внимание на последний пример: при неверном задании первого параметра (выбора) просто повторяется запись функции.

Следующий пример показывает возможность выбора с применением вещественных значений параметра выбора и меток:

```
Switch[8.,1.5,a,2.5,b,8.,c]
c
Switch[1.5,1.5,a,2.5,b,8.,c]
a
Switch[8,1.5,a,2.5,b,8.,c]
Switch[8.2,
1.5,a,
2.5,b,
8.,c]
```

Обратите опять-таки внимание на последний пример: параметр выбора здесь выбран как целочисленное число 8, тогда как метка выбора – вещественное число 8.

Выбор при этом не происходит, поскольку целочисленное значение 8 не является тождественным вещественной восьмерке.

2.7.3. Безусловные переходы

В целом, условные выражения в языке программирования системы Mathematica позволяют реализовать любой вид ветвления в программах. Однако иногда бывает полезно без лишних раздумий указать в программе явный переход к какой-либо ее части. Для этого используется оператор безусловного перехода **Goto[tag]**, который дает переход к тому месту программы, которое отмечено меткой **Label[tag]**. Возможны также формы **Goto[expr]** и **Label[expr]**, где *expr* – вычисляемое выражение.

Применение оператора **Goto** иллюстрирует следующий пример:

```
(q = 2; Label[start]; Print[q]; q += 2;  
      If[q < 7, Goto[start]])
```

2
4
6

Здесь с помощью оператора **Goto[start]** организован цикл с возвратом на метку **Label[start]**, действующий до тех пор, пока значение *q* меньше 7. При этом *q* меняется от начального значения 2 с шагом 2, причем добавление 2 к текущему значению *q* осуществляется укороченным оператором сложения *q* += 2.

Интересной особенностью языка программирования Mathematica является возможность создания переходов по значению вычисляемого выражения. Например, **Goto[2+3]** дает переход к метке **Label[5]** или даже **Label[1+4]**, что видно из следующего примера:

```
Goto[2+3];  
Print["aaaaa"];  
Label[1+4];  
Print["bbbbbb"]  
bbbbbb
```

Переходы, задаваемые выражениями, и метки, меняющие свой идентификатор, редко встречаются в обычных языках программирования, хотя они обеспечивают новые обширные и довольно необычные возможности по созданию программ с различными ветвлениями.

Для языка программирования системы Mathematica, ориентированного на безупречное и строгое структурное программирование, введение оператора **Goto** может оцениваться как отступничество от основополагающих идей структурного программирования. Поэтому на применение этого оператора наложено табу в методах структурного программирования. Тем не менее, этот оператор есть, а применять его или нет – дело пользователя.

2.8. Механизм контекстов

2.8.1. Старые проблемы

Читатель, применяющий систему Mathematica на практике, знает, что при создании документов нередко конфликты между переменными, назначаемыми пользователем, и переменными, входящими в ядро, между функциями пользователя и встроенными функциями, между их заголовками и т.д. Ситуация усложняется при использовании пакетов расширения, поскольку в них широко используются переменные и различные функции, причем нередко обозначенные точно так же, как и встроенные функции.

Особенно коварны побочные эффекты в конструкциях, содержащих вспомогательные переменные – например, в итерационных циклах, функциях вычисления суммы и произведения и др. Они содержат *переменные-итераторы* *i*, *j*, *k* и т.д. Обычно избежать конфликтов можно с помощью механизма локализации итераторов. Вернемся к уже обсуждавшимся примерам. Возьмем пример с вычислением суммы:

```
i=2
2
Sum[i,i,1,4]
10
i
2
```

Ясно, что сумма вычисляется с применением цикла с заданным числом повторений. В конце его итератор *i* получает значение 4. Но глобальная переменная с тем же именем имеет значение *i*=2, которое она получила до вычисления суммы с помощью функции Sum. В данном случае это достигнуто за счет того, что в теле функции переменная-итератор является локальной.

Нетрудно убедиться, что проблемы со статусом переменных возможны и в, казалось бы, хорошо изученных функциях суммирования и перемножения. На это явно указывает следующий пример:

```
func[x_]:=Sum[x^i,{i,4}]
{func[y],func[i]}
{y+y2+y3+y4, 288}
i
1
```

Результат вычисления **func[y]** вполне понятен, тогда как вычисление **func[i]** носит явно обескураживающий характер. Причина его в том, что вместо символического значения *i* в данном случае оказались использованными численные значения итератора *i*. А в этом случае функция **Sum** просто вычисляет численные значения. Говорят, что она работает по контексту!

А теперь возьмем пример с циклом **For**:

```
For [i=1,i<=4,i++,Print[i]]
```

```
1
2
3
4
i
5
```

На этот раз переменная *i* изменила свое значение в конце цикла с 2 на 5. Это говорит о том, что пользователю-программисту следует очень внимательно относиться к статусу переменных во всех итерационных, да и других, программах.

Разумеется, Mathematica содержит средства для избегания подобного смешения ролей переменных. Одно из них – применение конструкции **Module**:

```
i=2
2
Module[{i},For[i=1,i<=4,i++,Print[i]]]
1
2
3
4
i
2
```

На этот раз захвата итератором глобальной переменной *i* удалось избежать. Однако этот пример носит не более чем частный характер. Вообще говоря, если переменная-итератор задается в теле функции, то она будет локальной, а если она задается за пределами функций, то – глобальной.

2.8.2. Что такое контекст?

Для разрешения подобных противоречий в системе Mathematica введен особый механизм *контекстов*. Напомним, что под контекстом подразумевается некоторое разъяснение характера связанных с контекстом объектов. Другими словами, это означает, что с каждым объектом системы Mathematica (например, с переменными или функциями) связан некоторый контекст. Чисто внешне контекст задается в виде **Имя_контекста`** (обратный апостроф в конце имени и есть признак контекста).

Итак, контекст фактически является некоторым признаком объекта. Каждый объект системы Mathematica имеет свой контекст, который записывается перед именем объекта (знак ` при этом является разделителем). Обычно он не виден, но существует. Одни и те же по имени объекты могут иметь разные контексты и действовать по-разному, т.е. по контексту. Пользователям ПК полезно усвоить такую аналогию: контексты – это как бы разные папки (директории) со своими именами, куда могут помещаться одноименные файлы-объекты. Таким образом, могут быть разные объекты с одинаковыми именами.

С другой стороны, один и тот же контекст может принадлежать разным объектам. Например, все системные переменные и встроенные функции имеют кон-

текст `System``, т.е. они относятся к системным объектам, а все символы, вводимые в начале работы с системой, имеют контекст `Global`` (глобальные).

2.8.3. Работа с контекстами

В системе Mathematica есть средства для визуализации контекстов. Прежде всего, это функция **Context**:

```
Context[Tan]
System`
Context[E]
System`
Context/({Cos,Pi,Abort})
{System`,System`,System`}
```

Текущее значение контекста определяет системная переменная **\$Context** или функция **Context[]**:

```
{ $Context, Context[] }
{ Global`, Global` }
```

В начале сессии по умолчанию действует контекст `Global``, что означает глобальный статус вводимых символов:

```
Context/({q,p,w})
{Global`,Global`,Global`}
```

Однако контекст может быть заменен на любой нужный пользователю просто указанием его перед соответствующим символом или словом:

```
{ new`q,new`w,Global`p}
{ new`q, new`w, p}
Context/({new`q,new`w,Global` p})
{new`,new`,Global`}
```

Обратите внимание на то, что символы `new`q` и `new`w` имеют свой новый контекст `new`` и отображаются вместе с ним (но контекст указан перед символом). А вот символ `Global`p` отображается лишь кратким именем, а полностью не отображается. Причина этого в том, что текущий контекст есть `Global``, а контекст `new`` отсутствуют на так называемой контекстной дорожке, проще говоря, списке контекстов. Что касается символов `q`, `r` и `z`, то сами по себе (без новой контекстной приставки) они по-прежнему имеют контекст ``Global``:

```
Context/({q,p,w})
{Global`,Global`,Global`}
```

Для вывода контекстной дорожки используется переменная **\$ContextPath**:

```
$ContextPath
{Global`,System`}
```

С помощью функции **Prepend** можно добавить в контекстную дорожку новый контекст, например `new``:

```
$ContextPath=Prepend[$ContextPath,"new`"]
{new`,Global`,System`}
```

Теперь функция **Context** возвращает только контексты символов `new`q`, `new`w` и `Global`r`:

```
Context/({new`q,new`w,Global`p}
{new`,new`,Global`})
```

С помощью функции **Begin** можно изменить текущий контекст на заданный, например `Global`` на `new``:

```
Begin["new`"]
new`
q=5;
{ q,Context[q] }
{5,new`}
```

Теперь легко разобраться в том, как интерпретируются символы с разными контекстами. Любой символ, вводимый без контекстной приставки, т.е. своим коротким именем, интерпретируется и выводится с этим именем, если его контекст является текущим. Если символ вводится полным именем, то проверяется, есть ли его контекст на контекстной дорожке. Если он есть, то к символу добавляется самый левый контекст из имеющихся на контекстной дорожке. Таким образом, по мере ввода новых контекстов, имена которых совпадают со старыми, происходит вытеснение новыми контекстами старых. Другими словами, это позволяет обновить уже имеющиеся определения, сохранив их на случай отмены старых контекстов.

Этот принципиально важный механизм модификации объектов играет решающую роль в создании пакетов расширений. В них часто уже имеющиеся функции (со старыми контекстами) заменяются новыми одноименными с ними, но имеющими иные контексты.

2.8.4. Получение списков определений с контекстами

Для получения списка всех определений с заданным контекстом можно использовать функции **Name["Context`S"]**, где *S* – символ или строка, определяющие имена определений. Например, для получения всех определений с контекстом `System`` можно использовать функцию **Name["System`*"]**. Поскольку этот список довольно большой, ограничимся примером вывода всех определений с контекстом `System`` в системе *Mathematica* 6, начинающихся с буквы *U*:

```
Names["System`U*"]
{UnAlias,Uncompress,UnderBar,Underflow,Underlined,Underoverscript,
UnderoverscriptBox,UnderoverscriptBoxOptions,Underscript,UnderscriptBox,
UnderscriptBoxOptions,UndocumentedTestFEParsePacket,
UndocumentedTestGetSelectionPacket,Unequal,Unevaluated,UniformDistribution,
Uninstall,Union,UnionPlus,Unique,Unitize,UnitStep,UnitVector,Unprotect,UnsameQ,
UnsavedVariables,Unset, UntrackedVariables,Up,UpArrow,UpArrowBar,
UpArrowDownArrow, Update,UpdateDynamicObjects,
UpdateDynamicObjectsSynchronous,UpdateInterval,UpDownArrow,UpEquilibrium,
```

UpperCaseQ, UpperLeftArrow, UpperRightArrow, UpSet, UpSetDelayed, UpTee, UpTeeArrow, UpValues, URL, UseGraphicsRange, Using}

Функция **Name[]** без параметра выводит **полный список** всех определений как ядра, так и всех определений в пакетах расширений с указанием их контекстов. Таким образом, данная функция дает самую полную информацию об определениях (функциях, константах и т.д.), которые имеет текущая версия системы Mathematica.

2.9. Программирование ввода-вывода

2.9.1. Осуществление интерактивного диалога

Ввод-вывод в системе Mathematica организован с помощью интерфейсного процессора FrontEnd настолько естественно, что у большинства пользователей едва ли появится искушение изменять формы ввода-вывода по сравнению с установленными по умолчанию. Тем не менее, это возможно с помощью функций ввода-вывода:

- **Input[]** — останавливает работу системы и возвращает значение выражения, которое будет введено в появившемся диалоговом окне (служит для организации диалогового ввода);
- **Input[«prompt»]** — то же, что и предыдущая функция, но с выводом в диалоговое окно комментария **prompt**;
- **InputString[]** — выполняет интерактивное чтение в символьную строку;
- **InputString["prompt"]** — то же, но с выводом в диалоговое окно комментария **prompt**;
- **StylePrint[expr,]** — создает в текущем документе новую ячейку со стилем по умолчанию и заносит в нее выражение **expr**;
- **StylePrint[expr, "style"]** — создает в текущем документе новую ячейку со стилем **style** и заносит в нее выражение **expr**;
- **Print[expr]** — выводит на экран дисплея значение выражения **expr**; совместно с **Input** может использоваться для организации диалога;
- **Print["prompt", expr]** — выводит на экран дисплея текстовый комментарий, указанный в кавычках, и следом — значение выражения **expr**.

Этих функций достаточно для организации простейшего диалога с программой.

На рис. 2.6 показан простейший пример организации диалога в стиле, принятом в языке Бейсик. В данном случае вычисляется длина окружности с запросом радиуса R .

При исполнении документа, приведенного на рис. 2.6, вначале выполняется функция **Input**. Это ведет к появлению диалогового окна в центре экрана — на рис. 2.6 оно несколько смещено вниз, чтобы не загромождать содержимое ячейки документа. В окне виден запрос, который указан в кавычках как параметр функции **Input**. После ввода нужного значения (в нашем примере это радиус окружности) и нажатия клавиши **Enter** или щелчка на кнопке **ОК** диалогового окна функ-

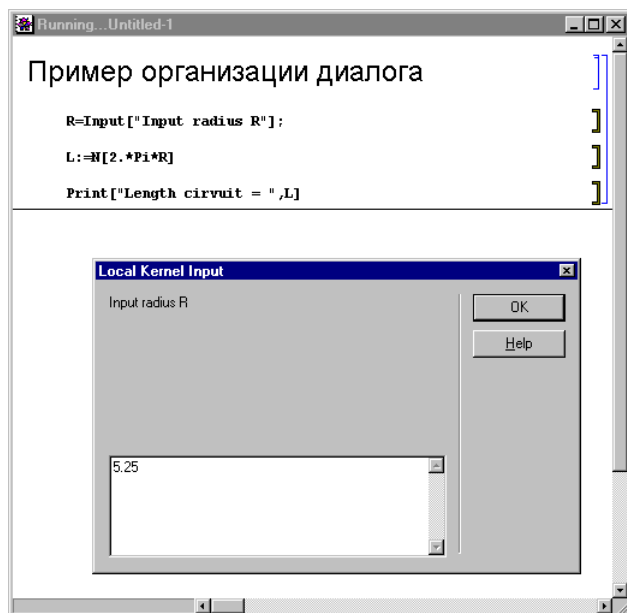


Рис. 2.6. Пример организации диалога

ция **Input** возвращает введенное значение, и оно присваивается переменной **R**. После этого функция **Print** выводит на экран вычисленное значение длины окружности с кратким комментарием (рис. 2.7).

Разумеется, для данного примера нет никакого смысла организовывать диалог в такой форме, поскольку однократное вычисление длины окружности проще задать прямо в тексте документа без запроса радиуса, просто указав $R=10$. Однако при составлении сложных программ, например, ориентированных на многократные вычисления с различными данными по скрытым формулам, такая возможность организации диалога очень полезна. Ее можно использовать и при составлении обучающих программ на базе системы Mathematica.

К сожалению, комментарий, отображаемый в окне функции ввода **Input**, возможен только на английском языке; при вводе символов кириллицы вместо обычных надписей выводятся непонятные символы (в то же время функция **Print** исправно выводит комментарии на русском языке). Это связано с выбором для данного окна шрифта, не содержащего символов кириллицы.

2.9.2. Задание формата вывода

Далее отметим функции, меняющие *формат* представления выражений. Все они имеют в своем названии слово **Form** (форма). Таких функций довольно много, и их полный список вы найдете в справке. Отметим лишь несколько наиболее часто используемых функций этого рода:

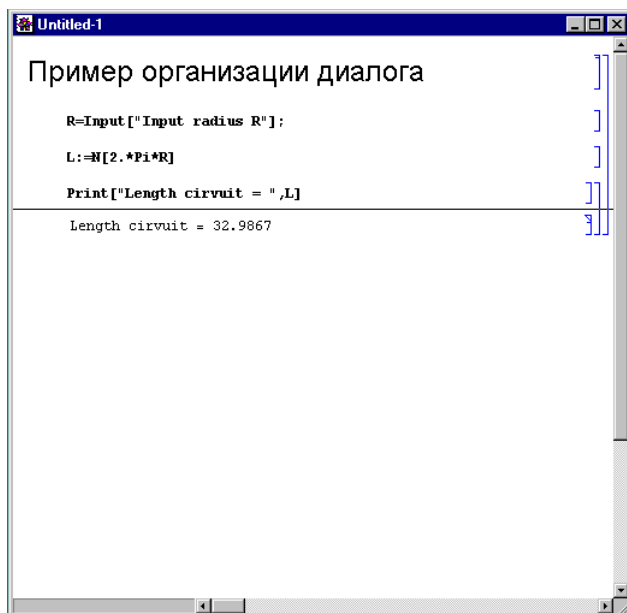


Рис. 2.7. Документ, показанный на рис. 2.6, по окончании диалога

- **AccountingForm[expr]** — выполняет вывод всех чисел, содержащихся в выражении **expr**, в бухгалтерской форме представления;
- **CForm[expr]** — выполняет вывод **expr** в форме, принятой для языка С;
- **EngineeringForm[expr]** — дает вывод, представляя все вещественные числа в выражении **expr** в инженерной форме (это означает, что порядок чисел равен нулю или кратен трем);
- **FortranForm[expr]** — выводит **expr** в форме, принятой для языка Фортран.
- **FullForm[expr]** — выводит полную форму выражения **expr** без использования специального синтаксиса;
- **InputForm[expr]** — выводит **expr** во входной форме;
- **NumberForm[expr, n]** — выполняет вывод **expr** с вещественными числами, представленными с точностью до **n** цифр;
- **OutputForm[expr]** — выполняет вывод **expr** в стандартной выходной форме системы Mathematica;
- **ScientificForm[expr]** — выполняет вывод, представляя все вещественные числа в выражении **expr** в научном формате;
- **TeXForm[expr]** — выводит **expr** в форме, принятой для языка TeX, ориентированного на верстку текстов с математическими формулами;
- **TextForm[expr]** — выполняет вывод **expr** в обычном текстовом формате;
- **TreeForm[expr]** — выполняет вывод **expr** с показом разных уровней выражения.

В большинстве своем действие этих функций вполне очевидно. Если это не так, то смело экспериментируйте с ними. Следующие примеры дают представление об использовании различных форм вывода в системе.

Ввод (In)	Вывод (Out)
AccountingForm [30*10 ¹⁵]	30000000000000000
BaseForm [55434,16]	d88a ₁₆
CForm [x ² +3*x+x]	4*x + Power(x,2)
ColumnForm [{a,b,c}]	a b c
EngineeringForm [N[12*10 ²⁹]]	1.2 10 ³⁰
Format [Exp[x ²]/a]	$\frac{x^2}{a}$
FortranForm [Exp[x] ² /a]	E**(2*x)/a
HoldForm [Exp[x] ² /a]	$(e^x)^2$ $\frac{\quad}{a}$
NumberForm [N[Exp[2]],15]	7.38905609893065
OutputForm [Exp[x] ² /a]	$\frac{E^{2x}}{a}$
TeXForm [Exp[x] ² /a]	$\frac{e^{2x}}{a}$
ScientificForm [12*10 ⁵]	1200000

Приведем еще несколько примеров использования различных форм вывода (здесь содержимое ячеек вывода дано под содержимым ячеек ввода):

FullForm[Exp[x]²/a]
Times[Power[a, -1], Power[E, Times[2, x]]]
TreeForm[Exp[x]²/a]
Times[
Power[a, -1] Power[E,
Times[2, x]
]

PaddedForm[(x³+2*x²+3*x-1)/(x-1),3]

$$\frac{-1 + 3x + 2x^2 + x^3}{-1 + x}$$

PrecedenceForm[12*b/c,5]

$$a + \frac{12b}{c}$$

SequenceForm[Exp[x]²/a]

$$\frac{E^{2x}}{a}$$

```
TableForm[{{"x", "y"}, {1, 2}, {3, 4}, {5, 6}}]
```

```
x    y
1    2
3    4
5    6
```

```
Prefix[f[x^2]]
```

```
2
```

```
f@ (x )
```

```
Unevaluated[Exp[x^(a/b)]/x/a]
```

```
a/b
```

```
Exp[x    ]
```

```
Unevaluated[----]
```

```
x a
```

2.10. Функции задания объектов GUI ноутбуков

2.10.1. Слайдеры однокоординатные

Средства Mathematica 6 позволяют создавать объекты *графического интерфейса пользователя* GUI (Graphic User Interface) ноутбуков, что делает последние намного более наглядными и удобными в работе. Это особенно важно в образовании, поскольку позволяет наглядно представлять самые различные математические выражения и графические зависимости во многих вариантах и при плавном изменении тех или иных их параметров. По простоте задания и числу элементов GUI Mathematica 6 не имеет равных среди других СКМ.

Одними из таких объектов являются *слайдеры*, имитирующие реостаты с движком или переключаемые потенциометры, перемещение которых позволяет плавно или дискретно менять выходное напряжение – возвращаемое слайдером значение.

На рис. 2.8 показано задание трех слайдеров. Слайдеры задаются функцией **Slider** с различными вариантами задания параметров:

Slider[x]

Slider[x, {x_{min}, x_{max}}]

Slider[x, {x_{min}, x_{max}, dx}]

Slider[x, {{e₁, e₂, ...}}]

Slider[x, {{e₁, w₁}, {e₂, w₂}, ...}}]

В первом случае (верхний слайдер) параметр (число 0,8) задает построение слайдера, возвращающего значение 0,8, определяющее исходное положение движка. При перемещении его будет возвращаться значение в диапазоне от 0 до 1.

Второй слайдер (в центре рис. 2.8) позволяет задавать значение переменной *x*. Чтобы сделать ее значение динамически доступной во всем ноутбуке, переменная объявляется динамически изменяемой функцией **Dynamic[x]**. По умолчанию диапазон изменения переменной составляет от 0 до 1. Повтор **Dynamic[x]** обеспечивает вывод возвращаемого слайдером значения справа от него.

Третий слайдер (снизу рис. 2.8) позволяет задавать выход в диапазоне от 0 до 100 с шагом 1. Выход в этом случае является целочисленным, что важно в использовании слайдера в ноутбуках с символьными вычислениями.

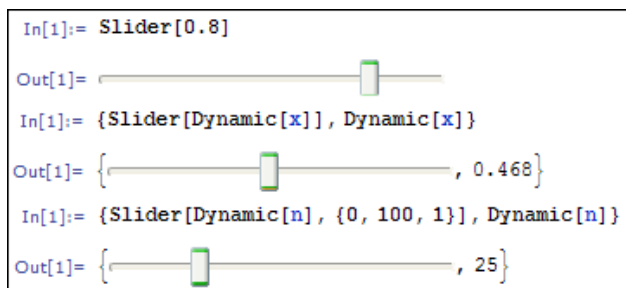


Рис. 2.8. Задание трех слайдеров

2.10.2. Слайдеры двухкоординатные

Иногда, например, при построении поверхностей, описываемых функциями двух переменных, удобны и необходимы двухкоординатные слайдеры. Они создаются функцией **Slider2D**.

Slider2D[{x,y}]

Slider2D[pt,{min,max}]

Slider2D[pt,{{x_{min},y_{min}},{x_{max},y_{max}}}]

Slider2D[pt,{{x_{min},y_{min}},{x_{max},y_{max}},{dx,dy}}]

Slider2D[Dynamic[pt]]

Slider2D[pt,{min,max,d}]

Пример задания такого слайдера представлен на рис. 2.9. Движок такого слайдера может перемещаться мышью в любом направлении, и слайдер возвращает два числовых значения, определяемых положением движка. Эти значения можно использовать для вычисления функций двух переменных и построения графиков поверхностей, трехмерных фигур, параметрически заданных графиков и т.д.

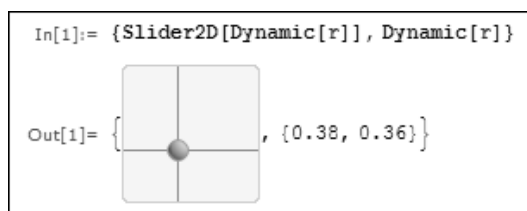


Рис. 2.9. Двухкоординатный слайдер

2.10.3. Элементы установки опций CheckBox

Нередко вычисления необходимо проводить при установке некоторых опций, например, представленных флагом, который задается в маленьком квадратном окошке, именуемом **CheckBox**. Он создается функцией **CheckBox**, записываемой в виде:

Checkbox[x]**Checkbox[x,{val₁,val₂}]****Checkbox[Dynamic[x]]****Checkbox[x,{val₁,val₂,val₃,...}]**

Примеры применения этой функции показаны на рис. 2.10.

```
In[1]:= {Checkbox[False], Checkbox[True]}
Out[1]= {☐, ☒
```

```
In[1]:= {Checkbox[1, {1, 2, 3}], Checkbox[2, {1, 2, 3}], Checkbox[3, {1, 2, 3}]}
Out[1]= {☐, ☒, ☒
```

Рис. 2.10. Примеры построения объекта CheckBox

2.10.4. Локаторы

Локатор – объект, представляющий точку в графическом окне и возвращающий координаты точки. Этот объект представлен закрашенным кружком в середине перекрестия, имеющем светлую точку в середине. Локатор может перемещаться мышью и задается функцией:

Locator[{x,y}]**Locator[Dynamic[pos]]****Locator[{x,y,obj}]****Locator[{x,y},None]**

Пример построения локатора с помощью функции **Locator** показан на рис. 2.11.


```
In[1]:= DynamicModule[{p = {0.5, 0.5}},
  {Graphics[Locator[Dynamic[p]], PlotRange -> 2], Dynamic[p]}]
Out[1]= {, {0.126667, 1.03333}}
```

Рис. 2.11. Пример построения локатора

Локаторы часто применяются для построения графиков по точкам. При этом точки хорошо выделяются и перемещаются по кривой мышью.

2.10.5. Функции управления и контроля мышью

Иногда возникает необходимость в определении координат курсора мыши. Это позволяет функция **MousePosition**. Пример применения ее показан на рис. 2.12. Координаты выводятся в целых числах, которые меняются при изменении поло-

```
In[1]:= Dynamic[MousePosition[]]
Out[1]= {229, 521}
```

Рис. 2.12. Пример определения координат курсора мыши

жения курсора мыши. Включение функции **MousePosition** в список параметров функции **Dynamic** позволяет вывести координаты курсора мыши в виде списка текущих координат.

Функция **Mouseover[expr, over]** контролирует событие, когда курсор мыши устанавливается на объект ее вывода. Она выводит expr, например:

Mouseover[Иван, Петр]

Иван

Однако если навести курсор мыши на выводимое слово «Иван», то оно тут же изменится на второе слово – «Петр». Тут любопытно, что Mathematica 6 благополучно воспринимает параметры как слова, записанные кириллицей, и воспроизводит их без искажений.

Функция **Opener[x]** или **Opener[Dynamic[x]]** реагирует на каждый клик мыши по черному треугольнику. Например, команда

{Opener[Dynamic[x]], Dynamic[x]}

выводит в строку вывода список:

{►, False}

Однако если кликнуть по нему мышью, то появится список:

{▼, True}

Функция

Toggler[x]	Toggler[Dynamic[x]]	Toggler[x, {val1, val2, ...}]
Toggler[x, {val1->pict1, val2->pict2, ...}]		Toggler[x, vlist, dpict]

обеспечивает возможность реакции на неоднократные клики мышью. Например, в следующем примере

Toggler[1, {a, b, c, d}]
1

изначально выводится 1. Однако, щелкая по ней 4 раза, можно получить на ее месте вывод символов a, b, c и d.

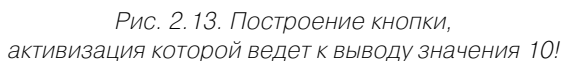
Функция

PasteButton[expr] **PasteButton[label, expr]**

выводит параметры в строку выхода. Если активизировать вывод expr (или label во второй форме записи функций), то expr будет скопировано в буфер и выведено в строку буфера.

Следует отметить, что возможности указанных функций гораздо больше описанных. Они достигаются применением как различных форм записи функций, так и различных опций. Это открывает почти неисчерпаемые возможности построения интерактивных элементов GUI в ноутбуках системы Mathematica 6.

Нередко некоторое действие должно выполняться при нажатии кнопки с надписью. Для создания такой действующей кнопки служит функция **Button[label,action]**. В списке ее параметров указываются строка с именем кнопки и выражение, которое исполняется при активизации кнопки мышью. Пример применения кнопки, при нажатии которой вычисляется факториал числа 10, показан на рис. 2.13. Для вывода значения факториала используется функция **Print[10!]**.



Манипулятор – особый объект, напоминающий слайдер, но имеющий более обширные функциональные и визуальные возможности. Манипулятор создается с помощью функции:

Отличительной особенностью манипулятора является характерная кнопочка в виде серого прямоугольника со знаком «+» в ней. При активизации мышью этой кнопки под слайдером манипулятора появляется панель с органами управления слайдером (см. рис. 2.14, пример сверху).

С помощью органов (кнопок) управления манипулятора можно пустить его и обеспечить автоматическое перемещение движка слайдера, можно изменить направление и скорость перемещения слайдера, осуществить его остановку (паузу). При приближении слайдера к начальной и конечной точкам появляется характерная тень у движка.

В некоторых случаях, например, при построении графиков функций в полярной системе координат, нужен объект, задающий угол поворота радиус-вектора. Такой объект – *задатчик угла поворота* – задается функцией **angularSlider**. На рис. 2.15 представлен весьма наглядный пример применения этой функции из справки по ней.

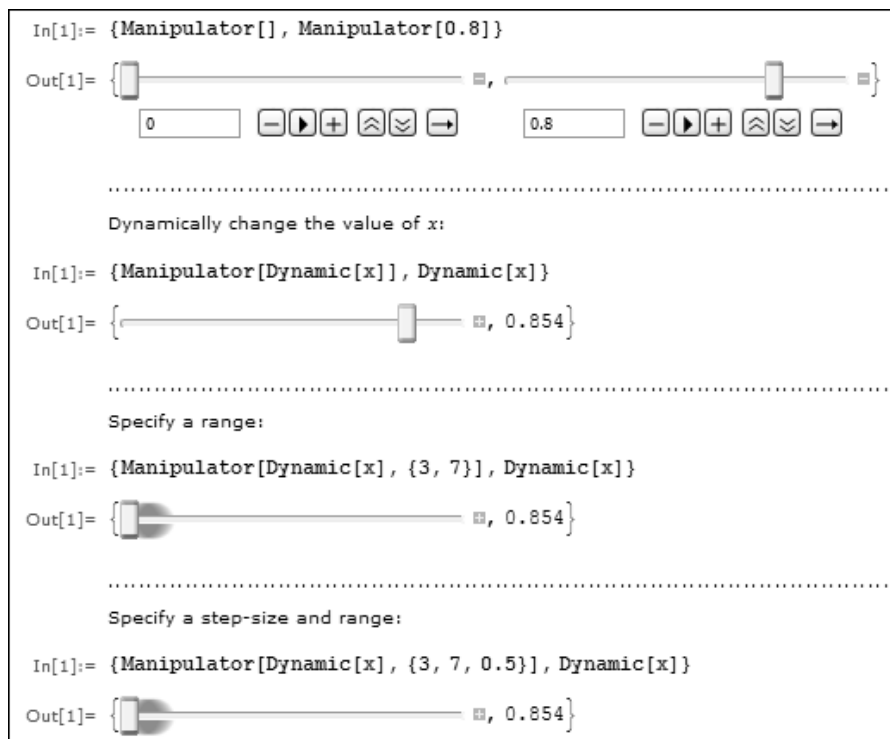


Рис. 2.14. Примеры применения манипулятора

Задатчик является графическим объектом в виде окружности, внутри которой помещен радиус-вектор. Он может вращаться в ту или иную сторону с помощью мыши, что ведет к изменению задаваемого задатчиком угла поворота. В примере, показанном на рис. 2.15, построены проекции конца радиус-вектора на координатные оси. Эти проекции, как известно, есть синусоидальные функции. Угол, соответствующий текущему положению радиус-вектора, помечен точками на графиках синусоидальных функций.

2.10.9. Выпадающее меню акций

Выпадающее меню акций – еще один широко применяемый объект для построения GUI. Он создается функцией

ActionMenu[name,{lbl₁:>act₁,lbl₂:>act₂,...}]

Параметрами функции являются строка с надписью на кнопке **И** и список названий позиций меню и действий, выполняемых при активизации соответствующих позиций меню. В примере, показанном на рис. 2.16, выполняются вычисления 4!, 7! и 10!.

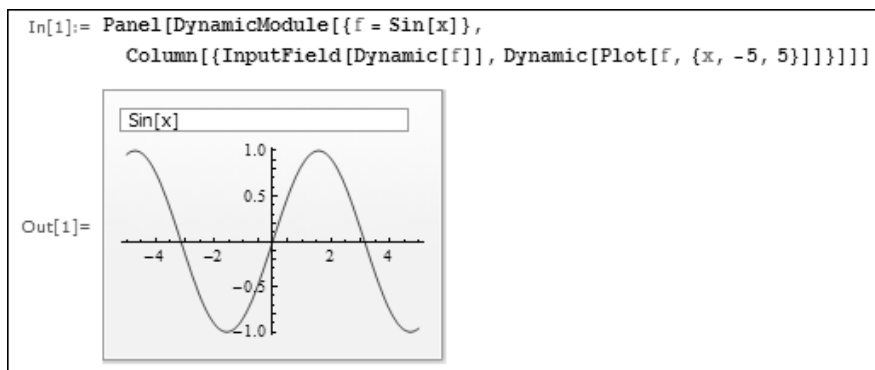


Рис. 2.17. Пример задания панели ввода выражения и построения его графика

Обратите внимание на то, что в панели ввода по умолчанию появляется выражение (в нашем случае $\text{Sin}[x]$), которое как параметр функции **DynamicModule** внутри списка параметров функции **Panel**. Это выражение и задает вид графика, который строится в области вывода. Однако, изменив выражение в панели ввода на другое, можно наблюдать построение нового графика – уже по этому выражению. Пример такого построения дан на рис. 2.18.

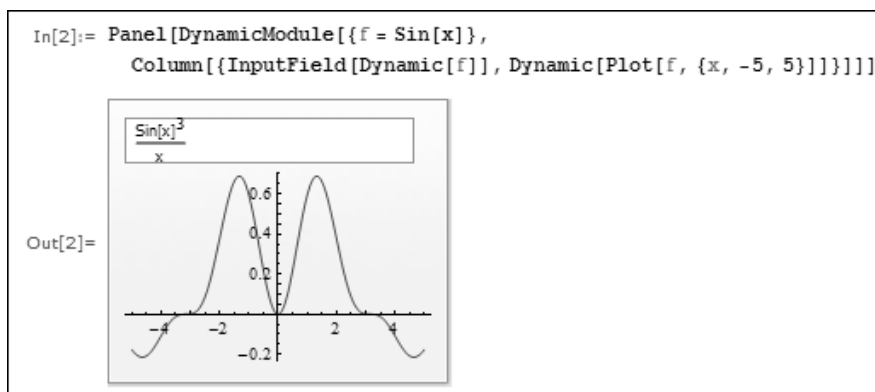


Рис. 2.18. Пример задания в панели ввода нового выражения и построения его графика

При реализации такой возможности в более ранних версиях Mathematica пришлось бы немало потрудиться над соответствующей программой.

2.10.11. Радиокнопки и меню установок

Функция **RadioButton[x, val]** создает так называемую радиокнопку в виде кружка того или иного размера, которую можно активизировать наведением на нее

мышью и нажатием (или только наведением курсора) левой клавиши мыши. Возвращаемое радиокнопкой число можно использовать для программного ввода того или иного действия. Примеры задания и применения радиокнопки даны на рис. 2.19.

```
Table[RadioButton[False, True, Appearance → a],
  {a, {Tiny, Small, Medium, Large}}]

{○, ○, ○, ●}

Table[RadioButton[Dynamic[x], a, Appearance → Dynamic[x]],
  {a, {Tiny, Small, Medium, Large}}]

{○, ○, ○, ●}

{RadioButton[Dynamic[x], 1, AutoAction → True],
  RadioButton[Dynamic[x], 2, AutoAction → True], Dynamic[x]}

{○, ●, 2}
```

Рис. 2.19. Примеры задания и применения радиокнопки

Еще одна функция в ряде форматов записи задает построение меню (бара) установок:

SetterBar[x, {val₁, val₂, ...}] **SetterBar[Dynamic[x], {val₁, val₂, ...}]**
SetterBar[x, {val₁ -> lbl₁, val₂ -> lbl₂, ...}]

Ее применение также довольно очевидно и представлено на рис. 2.20. В этом примере, активизируя ту или иную позицию меню с цифрой, можно наблюдать прямоугольную волну с заданной частотой.

На рис. 2.21 показано сравнение применений радиокнопки и меню установок. Демонстрируется возврат значения, соответствующего номеру кнопки или позиции меню выбора. Очевидно, что действие этих элементов GUI практически равноценно и сводится к выбору значения параметра n .

Еще один вариант построения меню установок задает функция

Setter[x, val] Setter[Dynamic[x], val]
 Setter[x, val, label] Setter[x, {val₁, val₂, ...}, label]

На рис. 2.22 показан пример, в котором кнопки меню установок с надписями Small (Малый) и Medium (Средний) обеспечивают установку соответствующего размера рисунка, представляющего график функции $\sin(x)^3$.

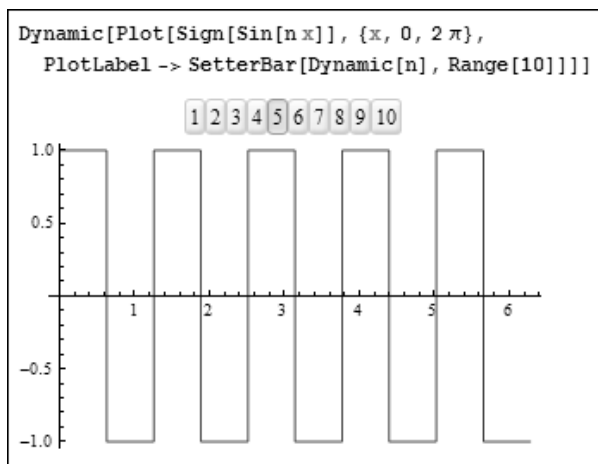


Рис. 2.20. Пример задания и применения меню установок

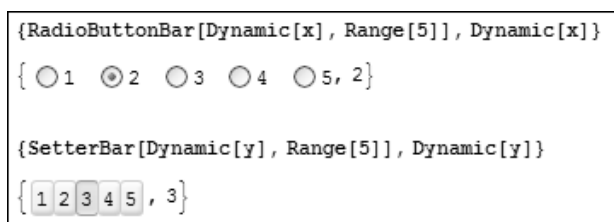


Рис. 2.21. Сравнение радиокнопок с меню установок

2.10.12. Слайдер изменения цвета

Для изменения цвета объектов GUI удобно использовать цветовой слайдер, который задается следующей функцией:

ColorSlider[*color*] ColorSlider[Dynamic[*color*]] ColorSlider[]

Примеры применения этой функции даны на рис. 2.23. Первый квадрат представляет возвращаемый цвет. За ним следует панель выбора цвета. Для выбора цвета достаточно установить курсор мыши (точнее его острие) на нужный цвет панели и щелкнуть левой клавишей мыши. Цвет контрольного квадрата станет аналогичным указанному мышью цвета.

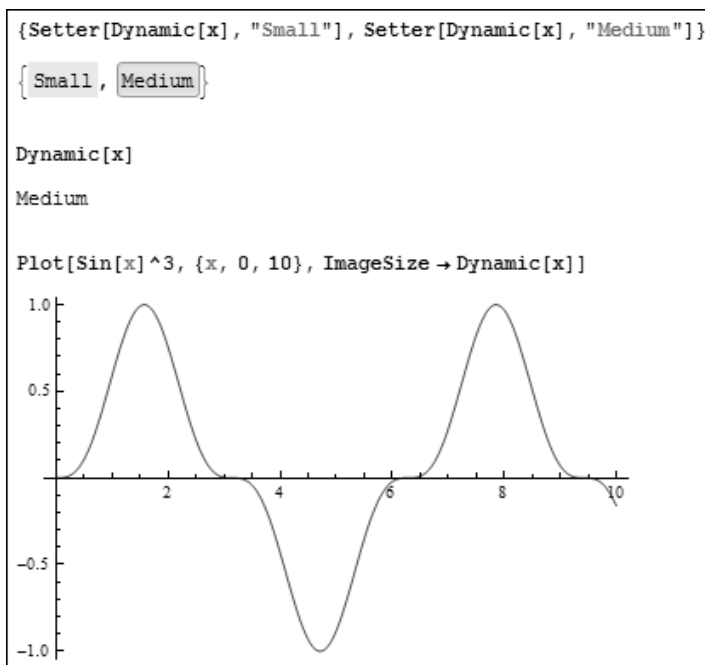


Рис. 2.22. Пример применения кнопок функции Setter для изменения размера рисунков

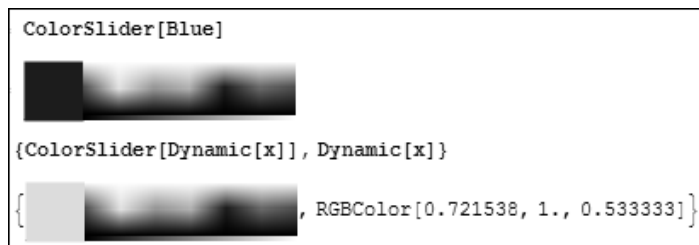


Рис. 2.23. Примеры применения цветового слайдера

2.10.13. Спусковой «механизм»

Функция **Trigger** имитирует работу спускового механизма. Она имеет несколько вполне очевидных форм записи:

```
Trigger[Dynamic[u]]
Trigger[Dynamic[u], {umin, umax}]
Trigger[Dynamic[u], {umin, umax, du}]
Trigger[Dynamic[u], {umin, umax}, ups]
```

В простейшем случае (рис. 2.24) кнопка в строке вывода выводит панель управления спусковым «механизмом». При нажатии на кнопку пуска (большой тре-

угольник) спусковой «механизм запускается» и динамическая переменная x начинает меняться от 0 до 1 в течение нескольких секунд. Затем это изменение прекращается. На панели есть также кнопки остановки изменения x и возврата к 0 (**Reset**).

```
{Trigger[Dynamic[x]], Dynamic[x]}
{
  {▶ || ⏏, 0.45}
```

Рис. 2.24. Пример применения спускового «механизма»

2.10.14. Функции указания места на объекте

Функция

ClickPane[image, func] **ClickPane[image, {{x_{min}, y_{min}}, {x_{max}, y_{max}}}, func]**

служит для указания места на рисунке image и применяет func к этой области. Чаще всего функция **ClickPane** используется для указания курсором мыши на некоторое место рисунка, в которое необходимо щелчком мыши поместить какой-либо графический объект, например точку, острие стрелки, кружок и т.д.

На рис. 2.25 показан пример применения функции для указания на экстремум синусоидальной функции с помощью стрелки, исходящей от надписи «Экстремум».

Функция

Tooltip[expr, label]

выводит в строку вывода expr и заменяет его label, если курсор мыши установить на вывод expr.

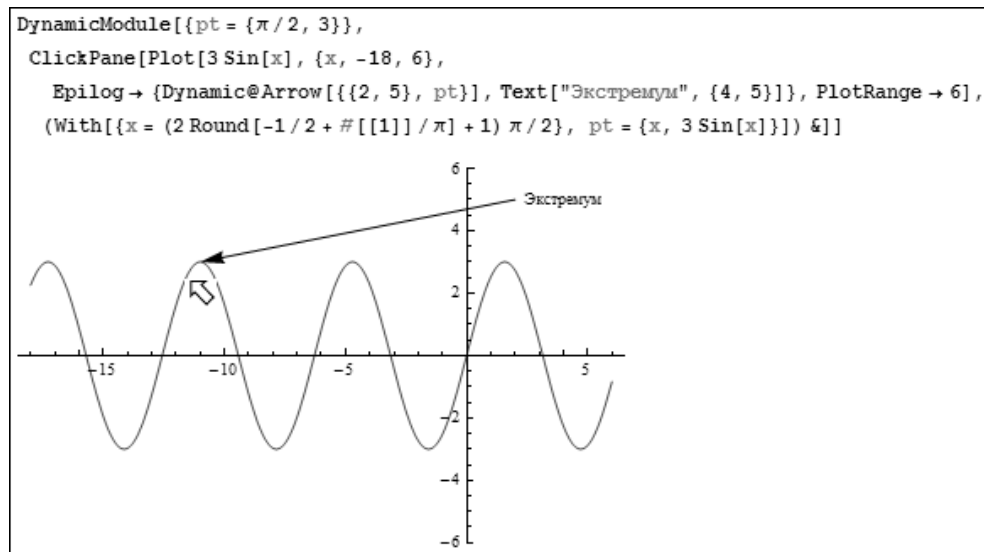


Рис. 2.25. Пример указания острием стрелки на экстремум синусоиды

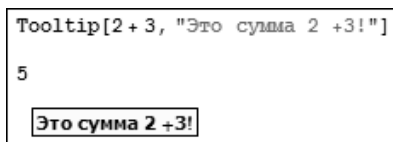


Рис. 2.26. Пример организации всплывающей текстовой надписи, поясняющей результат вычислений

Рисунок 2.26 показывает простейшую реализацию этой функции. Здесь она вычисляет выражение $2+3$ и выводит в строку вывода результат в виде числа 5. Если на это число навести курсор мыши (он виден только на экране дисплея), то появится всплывающая текстовая надпись «Это сумма $2+3!$ », которая была задана как параметр label.

Другой пример (рис. 2.27) показывает применение этой функции для распознавания одной из двух кривых, построенных одним цветом. Стоит курсор мыши (он виден как жирная стрелка) навести кончиком на одну из кривых, как под курсором и правее его появляется окошко с помещенным в нем выражением для соответствующей функции, по которой построен график.

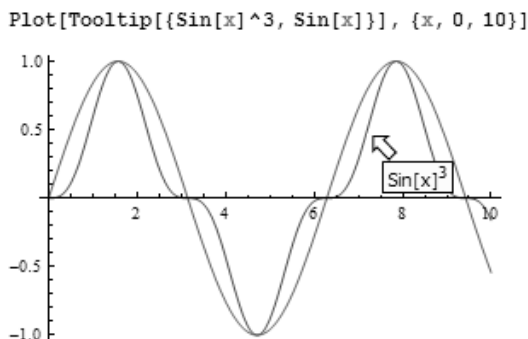


Рис. 2.27. Пример указания на один из двух графиков с выводом сообщения о функции этого графика

Рисунок 2.28 показывает еще один полезный прием применения функции Tooltip – вывод точного значения ординаты точечного графика синусоиды при наведении на нужную точку острия стрелки курсора мыши.

Наконец, на рис. 2.29 показан пример создания вывода пяти выражений вида $\text{Sin}[i*x]^i$, где i от 1 до 5. Если установить курсор мыши на одно из выражений, то снизу и справа от него появится всплывающее окно с графиком соответствующей функции. На рис. 2.29 показан случай, когда курсор мыши был установлен на выражение $\text{Sin}[3*x]^3$.

2.10.15. Вывод сообщения при активизации объекта мышью

Функция **PopupWindow** обеспечивает контроль над наведением курсора мыши на заданный в ней объект. Если курсор наведен на объект, то он модифицируется

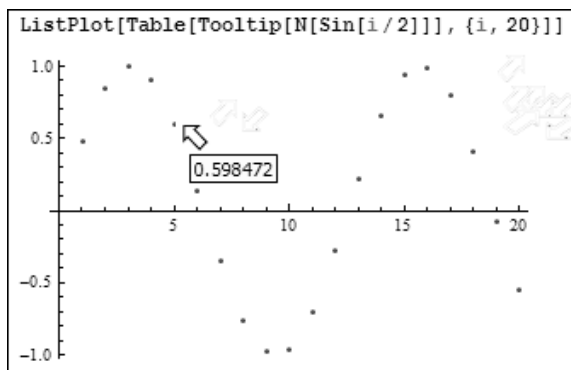


Рис. 2.28. Пример вывода значения ординаты точки точечного графика синусоиды

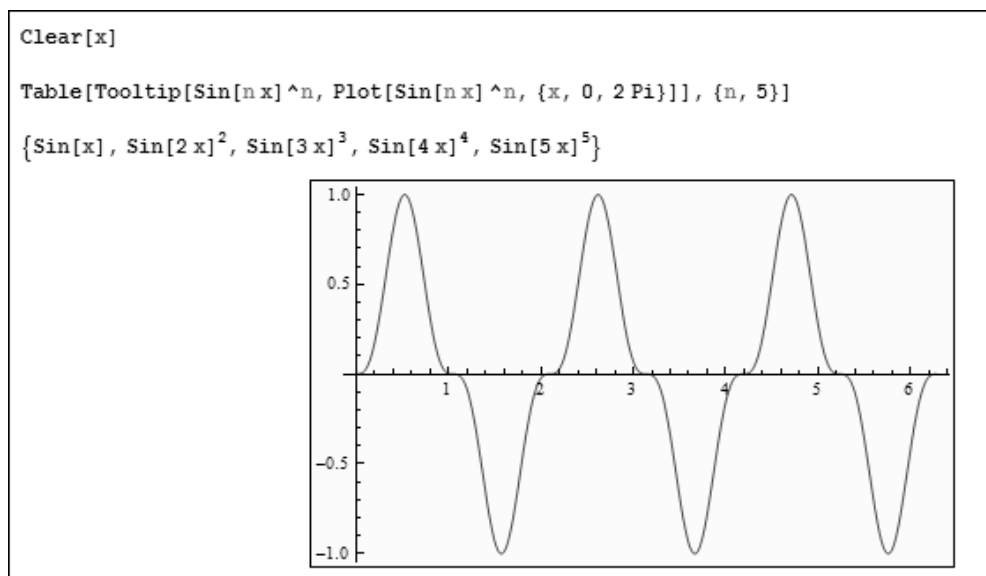


Рис. 2.29. Пример вывода всплывающего графика указанного курсором мыши математического выражения

и появляется панель с заданным сообщением. Например, на рис. 2.30 объектом является круг, который в момент наведения на него курсора мыши теряет окраску и становится окружностью. Затем после появления панели с надписью «Это диск» круг восстанавливает окраску.

Здесь стоит отметить примечательный факт: надписи для панели можно делать на русском языке, т.е. символами кириллицы, и эти надписи вводятся в панель и выводятся в окне вывода панели без искажений.

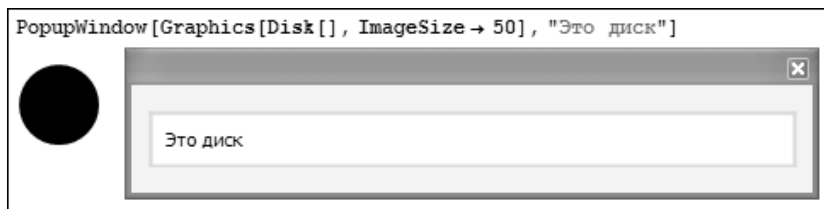


Рис. 2.30. Пример вывода панели с сообщением при наведении курсора мыши на графический объект (круг)

2.10.16. Вывод меню и выбор его позиций

Функция **MenuView** обеспечивает задание меню с выпадающими позициями под номерами, соответствующими значению переменной n (в нашем случае от 1 до 5). В примере рис. 2.31 задается вычисление функции $\tan(nx)$ для разных n .

2.10.17. Вывод меню с вкладками и их переключение

Функция **TabView** выводит меню с вкладками, которые можно переключать мышью. Пример применения этой функции дан на рис. 2.32. В данном случае в окне объекта строится рисунок, создаваемый клеточным автоматом с помощью функции **CellularAutomaton** с различным числом переменной r (tuple). Возможные значения r выбираются из списка активизацией соответствующей вкладки мышью и ведут к построению различных рисунков.

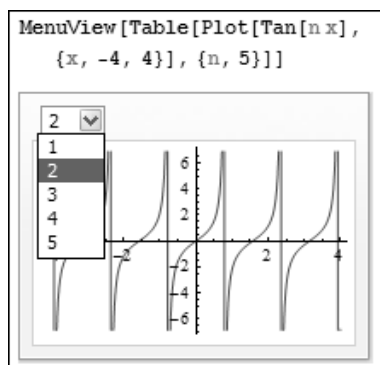


Рис. 2.31. Пример построения и использования меню с выпадающими позициями

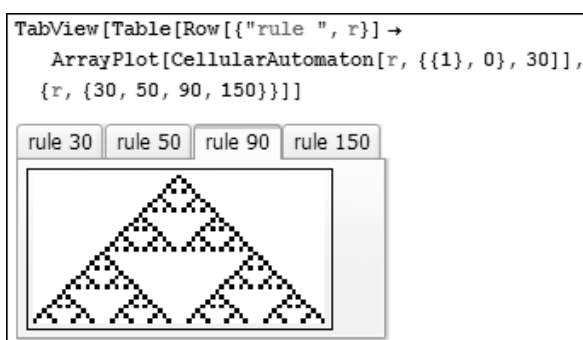


Рис. 2.32. Пример применения меню с переключаемыми вкладками, задающими различные рисунки

2.10.18. Вывод слайд-меню

Еще один вид меню – слайд-меню, управляемого проигрывателем (по типу переключения слайдов), реализует функция **SlaidMenu**. Работа с ней поясняется рис. 2.33. При активизации кнопок с изображениями треугольника можно выбирать символ из списка. Переключение может идти в любую сторону, а также сразу в начало или в конец списка.

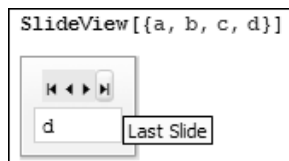


Рис. 2.33. Пример применения слайд-меню

2.10.19. Конструирование отдельных окон с GUI

В Mathematica 6 предусмотрена возможность создания отдельных окон GUI. Для этого служит специальный пакет расширения GUIKit. На рис. 2.34 показан вызов этого пакета и затем демонстрационного пакета – окна для демонстрации арифметических операций с двумя вводимыми числами.

Для создания даже такого простого примера требуется программа, содержащая около шести десятков строк. Она представлена файлом Calculator.m, с листингом которого легко познакомиться, загрузив файл в редактор Mathematica 6 или в Notepad Windows XP. Заинтересованный читатель может познакомиться с основами создания таких окон с GUI с помощью имеющегося в справке самоучителя GUIKit.

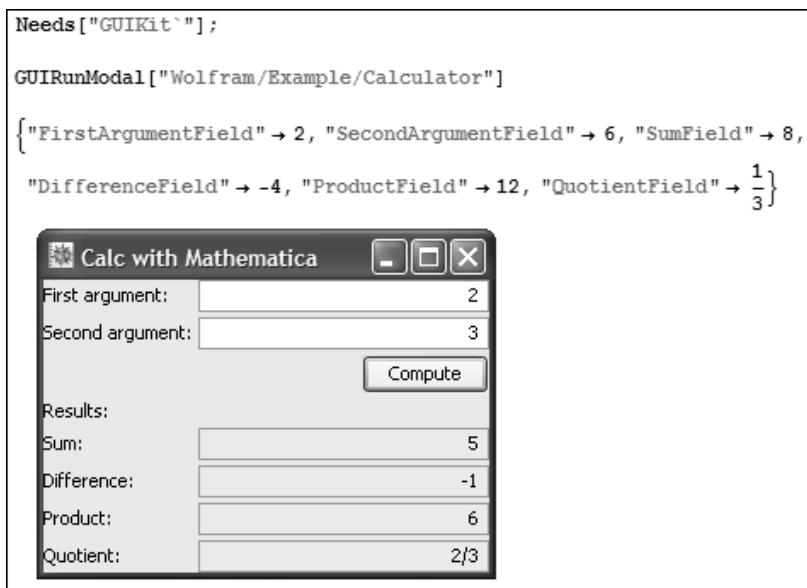


Рис. 2.34. Пример окна с GUI, демонстрирующий сложение двух чисел

Типы данных, операторы и функции

3.1. Работа с простыми типами данных	158
3.2. Работа со сложными типами данных	163
3.3. Работа с объектами и функциями	164
3.4. Применение констант и размерных величин	167
3.5. Работа с переменными	168
3.6. Применение подстановок	172
3.7. Задание и применение функций пользователя	173
3.8. Средства арифметических вычислений	175
3.9. Функции арифметических операций	179
3.10. Логические операторы и функции	183
3.11. Работа с математическими функциями	187
3.12. Расширенные возможности работы с объектами	199

3.1. Работа с простыми типами данных

3.1.1. Типы данных системы

Как и всякая программная система, Mathematica предназначена для автоматической обработки информации – в нашем случае математической. Чаще всего эта обработка сводится к вычислениям и их визуализации, например, графической. Информация может быть представлена в самом различном виде: числами, математическими формулами, текстовыми символами и иными элементами информации [4, 5]. Полное описание типов данных системы можно найти в поставляемой с ней технической документации [26–28]. В Mathematica 5.1/5.2 она в электронном виде включена в справку.

На уровне своего входного языка программирования и общения с пользователем система Mathematica оперирует с тремя *основными классами данных*:

- численные, представляющие числа различного вида;
- символьные, представляющие символы, тексты и математические выражения (формулы);
- списки – данные в виде множества однотипных или разнотипных данных.

Каждый из этих классов данных, в свою очередь, имеет ряд специальных, более частных типов данных. Остановимся на работе с простыми типами данных – числами.

3.1.2. Работа с целыми числами

В Mathematica используются целые числа с различным основанием и десятичные числа с плавающей точкой (их часто именуют числами с плавающей запятой), представленные в различной нотации. Из целых чисел широко используются двоичные числа с основанием 2, восьмеричные с основанием 8, десятичные с основанием 10 и шестнадцатеричные числа с основанием 16. Самыми распространенными являются *десятичные числа* (DECIMAL). Каждый разряд таких чисел имеет представление, заданное одной из арабских цифр: 0, 1, 2, ... 9. Весовой коэффициент старшего разряда относительно предшествующего равен 10.

Для вычисления чисел с произвольным *основанием* используется конструкция

Основание^{Число}

Число должно быть записано по правилам записи чисел с соответствующим основанием. Для оснований более 10 для обозначений значений чисел используются буквы от a до z. Наиболее известными из чисел с разрядностью более 10 являются *шестнадцатеричные числа* (HEX – от слова hexagonal). Разряд таких чисел может иметь значения:

HEX	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Старший разряд имеет весовой коэффициент относительно заданного разряда, равный 16.

Примеры задания шестнадцатеричного и двоичного чисел:

```
16^^123abcde
```

```
305839326
```

```
2^^1010111
```

```
87
```

Для представления чисел с произвольным основанием n (до 32) используется функция **BaseForm[expr, n]** – возвращает выражение *expr* в форме числа с основанием n , которое указывается как подстрочный индекс.

Примеры с использованием функции **BaseForm**:

```
BaseForm[87, 2]
```

```
10101112
```

```
BaseForm[305839326, 16]
```

```
123abcde16
```

Для получения списков цифр различных целых чисел служит функция

```
IntegerDigits[n, b, len]
```

где n – число, b – основание и len – длина числовой последовательности, дополняемой слева нулями. Параметры b и len могут отсутствовать. Примеры применения этой функции представлены ниже:

```
IntegerDigits[1234]
```

```
{1, 2, 3, 4}
```

```
IntegerDigits[1234, 2]
```

```
{1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0}
```

```
IntegerDigits[10!, 8]
```

```
{1, 5, 6, 5, 7, 4, 0, 0}
```

```
IntegerDigits[10!, 10, 12]
```

```
{0, 0, 0, 0, 0, 3, 6, 2, 8, 8, 0, 0}
```

В действительности, применяемые в системах символьной математики способы представления чисел более компактны, чем приведенный простейший, но это не меняет сути главного: чем больше количество цифр числа, тем больше памяти отводится на его хранение. Особенно много внимания уделено компактному хранению чисел в системе Mathematica, что позволило в несколько раз снизить затраты памяти на хранение больших числовых массивов и уменьшить время работы с ними.

Характерным примером работы с целыми числами большой разрядности является вычисление факториала $n! = 1 * 2 * 3 * \dots * n$ (при $0! = 1$):

```
1000!/950!
```

```
287731343120013000702468807306664495322316807860141258\  
460384342116480177434914877476604012266378453263443734\  
57983357703768094811080359936000000000000
```

```
(20!+5!)/22!
```

```
20274183401472001
```

```
9366672731480064000
```

Обратите внимание, что знак «\
» в конце строки вывода первого примера означает перевод последующих символов на новую строку.

$$1. \times 10^{-100}$$

Как принято в большинстве языков программирования, целая часть мантиссы отделяется от дробной части точкой, а не запятой. Знак «\» означает перевод строки (части числа). Его, кстати, можно использовать и в ячейках ввода.

Mathematica производит операции с числами, изначально как с целыми. Однако установка значка разделительной точки означает, что число рассматривается как вещественное. Например, число 1 – целое число, но 1. – уже вещественное число. Для представления выражения `expr` в форме вещественного числа используется функция

N[expr] или **N[expr, число_цифр_результата]**

Примеры:

1/3

$$\frac{1}{3}$$

1./3

0.333333

N[1/3]

0.333333

N[2*Pi, 50]

6.283185307179586476925286766559005768394338

N[2*Pi, 500]

6.283185307179586476925286766559005768394338798750211641949889184615\
63281257241799725606965068423413596429617302656461329418768921910116\
44634507188162569622349005682054038770422111192892458979098607639288\
57621951331866892256951296467573566330542403818291297133846920697220\
90865329642678721452049828254744917401321263117634976304184192565850\
81834307287357851807200226610610976409330427682939038830232188661145\
40731519183906184372234763865223586210237096148924759925499134703771\
5054497824558763660238983

Вещественные числа всегда имеют некоторую погрешность представления результатов из-за неизбежного округления их и существования так называемого «машинного нуля» – наименьшего числа, которое воспринимается как ноль.

Mathematica имеет две системные переменные, позволяющие вывести значения максимально и минимально возможных значений чисел, с которыми оперирует система:

\$MaxMachineNumber

1.79769×10^{308}

\$MinMachineNumber

2.22507×10^{-308}

Обратите внимание на то, что функция **N[expr, m]** позволяет получить число с практическим любым числом цифр результата m. Разработчики последней версии Mathematica 4 утверждают, что это верно при количестве цифр результата до 1 миллиона, что с лихвой удовлетворяет требования подавляющего большинства расчетов и вычислений.

Функции **IntegerPart[x]** и **FractionalPart[x]** обеспечивают возврат целой и дробной частей вещественного числа x:


```

N[Pi]
3.14159
IntegerPart[Pi]
3
FractionalPart[Pi]
-3+π
N[FractionalPart[Pi]]
0.141593

```

Еще одна функция **RealDigits[x]** возвращает список реальных цифр результата и число цифр целой части x :

```

RealDigits[N[2*Pi]]
{{6, 2, 8, 3, 1, 8, 5, 3, 0, 7, 1, 7, 9, 5, 8, 6}, 1}

```

Есть множество и других функций для работы с вещественными числами. Они будут рассмотрены в дальнейшем. В Mathematica 4/5 функция **RealDigits** имеет расширенные формы, например **RealDigits[x,b,len,n]**. Для получения цифр мантиссы введены функции **MantissaExponent[x]** и **MantissaExponent[x,b]**.

3.1.4. Работа с комплексными числами

Многие математические операции базируются на понятии *комплексных чисел*. Они задаются в форме:

$$z = \text{Re}(z) + I * \text{Im}(z) \text{ или } z = \text{Re}(z) + i \text{Im}(z),$$

где знак I – мнимая единица (квадратный корень из -1), $\text{Re}(z)$ – действительная часть комплексного числа, а $\text{Im}(z)$ – мнимая часть комплексного числа. Пример задания комплексного числа:

$$2 + I3 \text{ или } 2 + 3*I$$

Мнимая часть задается умножением ее значения на символ мнимой единицы I . При этом знак умножения « $*$ » можно указывать явно или заменить его пробелом, в последнем случае комплексное число выглядит более естественным. Функции **Re[z]** и **Im[z]** выделяют соответственно действительную и мнимую части комплексного числа z . Это иллюстрируют следующие примеры:

```

Re[3+2*I]
3
Im[3+2 I]
2

```

Большинство операторов и функций системы Mathematica работают с комплексными числами. Разумеется, это расширяет сферу применения системы и позволяет решать на ней различные специальные задачи – например, относящиеся к теории функций комплексного аргумента.

3.2. Работа со сложными типами данных

3.2.1. Символьные данные и строки

Символьные данные в общем случае могут быть отдельными символами (например, а, b,...,z), строками (strings) и математическими выражениями `expr` (от expression – выражение), представленными в символьном виде.

Символьные строки задаются цепочкой символов в кавычках, например «sssss». В них могут использоваться следующие управляющие символы для строчных объектов:

`\n` новая линия (line feed),
`\t` табуляция.

Это иллюстрируется следующими примерами:

«Hello my friend!»

Hello my friend

«Hello\nmy\friend!»

Hello

my

friend

«Hello\tmy\tfriend!»

Hello my friend

Следует помнить, что управляющие символы не печатаются принтером и не отображаются дисплеем, а лишь заставляют их выполнять назначенные ими действия. Mathematica имеет множество функций для работы со строками, которые будут описаны в дальнейшем.

3.2.2. Выражения

Выражения в системе Mathematica обычно ассоциируются с математическими формулами, например:

Запись на языке Mathematica	Обычная математическая запись
<code>2*Sin[x]</code>	$2 \cdot \sin(x)$
<code>2 Sin[x]</code>	$2 \sin(x)$
<code>(a + b^2 + c^3) / (3*d - 4*e)</code>	$(a+b^2+c^3)/(3d-4e)$
<code>sqrt(2)</code>	$\sqrt{2}$
<code>Integrate[Sin[x],x]</code>	$\int \sin(x) dx$

Для записи математических выражений используются как операторы, так и функции. Их особенности будут рассмотрены несколько позднее. А пока сразу отметим некоторые тонкости синтаксиса системы, используемого при записи арифметических операций:

- знак умножения может быть заменен пробелом;
- встроенные функции начинаются с большой буквы и обычно повторяют свое общепринятое математическое обозначение (за исключением тех, в названии которых есть греческие буквы – они воспроизводятся латинскими буквами по звучанию соответствующих греческих букв);
- круглые скобки (...) используются для выделения частей выражений и задания приоритета их выполнения;
- параметры функций задаются в квадратных скобках [...];
- фигурные скобки используются при задании списков {...}.

3.3. Работа с объектами и функциями

3.3.1. Объекты и идентификаторы

В общем случае система Mathematica оперирует с *объектами*. Под ними подразумеваются математические выражения (expr), символы (symbols), строки из символов (strings), упомянутые выше числа различного типа, константы, переменные, графические и звуковые объекты и т.д.

Каждый объект характеризуется своим именем-*идентификатором*. Это имя должно быть уникальным, т.е. единственным. Существуют следующие правила задания имен:

s s s s s имя объекта, заданного пользователем,
S s s s s имя объекта, входящего в ядро системы,
\$S s s s s имя системного объекта.

Итак, все объекты (например, функции), включенные в ядро, имеют имя, начинающееся с большой буквы (например, **Plus**, **Sin** или **Cos**). Относящиеся к системе объекты начинаются со знака \$. Заданные пользователем объекты следует именовать прописными (малыми) буквами. Разумеется под символами s...s подразумеваются любые буквы и цифры (но не специальные символы, такие как +, -, * и т.д.).

Объекты (чаще всего это функции), встроенные в систему, принято называть внутренними или встроенными. Объекты, которые создает пользователь (в том числе используя внутренние объекты), называют внешними объектами. К ним, в частности, относятся процедуры и функции, составляемые пользователем, которые детально рассматриваются в дальнейшем.

Различные объекты системы будут более подробно описаны в дальнейшем по мере знакомства с системой. Полный список объектов, заданных в ядре системы, легко получить, используя команду ?* (ниже дано лишь начало и конец этого списка):

?*

```
Abort
AbortProtect
Above
Abs
```

```
AbsoluteDashing
.....
$Version
$VersionNumber
```

Можно также получить список всех определений на заданную букву, используя команду **?S***, где S – любая буква латинского алфавита. Аналогичные возможности представляет функция **Name["S"]**, например **Names["A*"]** дает список всех ключевых слов начинающихся с символа A. Наконец, командой **?Name** можно вывести справку по любому определению с именем Name. Например, из следующего

?Abs

```
Abs[z] gives the absolute value
of the real or complex number z. More...
```

ясно, что идентификатор Abs задает функцию **Abs[z]** для вычисления абсолютного значения комплексного числа.

С помощью выражения **?Name** можно проверить, является ли имя объекта name уникальным, или оно уже использовано в системе:

?sin

```
Information::notfound : Symbol sin not found. More...
```

?Sin

```
Sin[z] gives the sine of z. More...
```

В первом случае ясно, что имя sin не использовано, а вот имя Sin уже зарезервировано – это функция вычисления синуса. В задачу этой книги не входит описание всех без исключения определений ядра системы, так что указанные выше приемы весьма полезны, если вы обнаружили функцию, по которой нет информации. Гиперссылка More..., введенная в Mathematica 5, позволяет получить дополнительную информацию, переходом к нужному окну справки.

Всякий объект перед использованием должен быть определен (задан). Внутренние объекты уже заданы в ядре. Объекты пользователя последний задает в текстах своих документов (Notebooks). Объединенные в определенные группы документы принято называть пакетами применений. Они представлены файлами с расширением .ma, записываемыми в текстовом формате ASCII.

Кроме того, некоторая совокупность новых внешних объектов может храниться в пакетах расширения системы, большой набор которых включен в поставку системы. Пользователь может и сам готовить пакеты расширений, обеспечивающие адаптацию системы к решению интересующего его класса задач. Пакеты расширений представлены файлами с расширением .m.

3.3.2. Функции, опции, атрибуты и директивы

К важному типу объектов принадлежат *функции*-объекты, имеющие имя и список параметров, возвращающие некоторое значение в ответ на обращение к ним по имени с указанием в списке конкретных (фактических) значений параметров. В Mathematica функции задаются в виде

Идентификатор_Функции[o1, o2, o3, ...],

где o1, o2, o3, ... – объекты-параметры, опции, математические выражения и так далее. Список входных параметров задается необычно – в **квадратных скобках**. В числе входных параметров могут быть специальные объекты-**опции**. Они задаются в виде:

Имя_опции->Значение_опции

Значением опции обычно является то или иное слово. Например, в функции построения графиков

```
Plot[sin[x], {x, 0, 20}, Axes->None]
```

опция **Axes->None** указывает на то, что отменяется вывод координатных осей (Axes). Функция **Options[name]** выводит для функции с идентификатором name список всех возможных для нее опций. Некоторые функции, например **Sin**, могут вообще не иметь опций, другие, например **Solve**, могут иметь целый «букет» опций:

```
Options[Sin]
```

```
{}
```

```
Options[Solve]
```

```
{InverseFunctions->Automatic, MakeRules->False, Method->3,
Mode->Generic, Sort->True, VerifySolutions->Automatic,
WorkingPrecision->∞}
```

В последнем случае характер возвращаемого ими результата может сильно зависеть от значений опций. Назначение каждой опции мы рассмотрим в дальнейшем. В этой главе они нам пока не понадобятся.

Каждый объект может характеризоваться некоторой совокупностью своих свойств и признаков, называемых **атрибутами**. Функция **Attributes[Sin]** возвращает список всех атрибутов функции с именем name, например:

```
Attributes[Sin]
```

```
{Listable, NumericFunction, Protected}
```

```
Attributes[Solve]
```

```
{Protected}
```

Так, для функции синуса характерны три атрибута:

- **Listable** – указывает на применимость в списках и таблицах;
- **NumericFunction** – указывает на отношение к числовым функциям;
- **Protected** – указывает на то, что слово Sin защищено от какой-либо модификации.

Кроме того, в Mathematica имеется понятие функций-**директив**. Эти функции не возвращают значений, а указывают, как будут выполняться в дальнейшем функции, работа которых зависит от директив. Синтаксис директив-функций тот же, что и у обычных функций.

Применение опций и директив делает аппарат функций более гибким и мощным, поскольку позволяет задавать те или иные свойства функций и условия их выполнения. Это особенно важно при использовании функций в задачах графики и символьной математики.

3.4. Применение констант и размерных величин

3.4.1. Применение констант

Константы являются типовыми объектами системы, несущими заранее предопределенное численное или символьное значение. Это значение не должно меняться по ходу выполнения документа. К численным константам относятся любые числа, прямо встречаемые в математических выражениях или в программных объектах, например процедурах и функциях. Так, числа 1 и 2 в выражении **2*Sin[1]** являются численными константами. Константы-числа не имеют своего имени. Им, в сущности, является само число. Его представление и хранится в памяти.

Имеется ряд поименованных констант, которые можно рассматривать как функции без аргумента, возвращающие заранее заданное значение. Имена констант (и других объектов, например функций и переменных) представляются их идентификаторами – непрерывной строкой символов, отождествляемой с именем. В системе Mathematica большинство идентификаторов имеют естественный математический смысл и начинаются с большой буквы. Например, *E* – это основание натурального логарифма.

Используются следующие поименованные константы:

- **ComplexInfinity** – комплексная бесконечность, которая представляет величину с бесконечным модулем и неопределенной комплексной фазой.
- **Degree** – число радиан в одном градусе, которое имеет числовое значение $\text{Pi}/180$.
- **E** – основание натурального логарифма с приближенным числовым значением 2.71828....
- **EulerGamma** – постоянная Эйлера с числовым значением 0.577216....
- **GoldenRatio** – константа со значением $(1+\text{Sqrt}[5])/2$, определяющая деление отрезка по правилу золотого сечения.
- **I** – представляет мнимую единицу $\text{Sqrt}[-1]$.
- **Infinity** – «положительная» бесконечность (со знаком минус – «отрицательная» бесконечность).
- **Catalan** – константа Каталана 0.915966....
- **Pi** – число, имеющее значение 3.14159... и дающее отношение длины окружности к ее диаметру.

Имеющие значение константы дают их в виде вещественных чисел:

```
{N[Degree], N[E], N[Pi]}  
{0.0174533, 2.71828, 3.14159}  
{N[EulerGamma], N[GoldenRatio], N[Catalan]}  
{0.577216, 1.61803, 0.915966}
```

Константы в описываемой системе используются вполне естественно, так что от дальнейшего их описания можно воздержаться.

3.4.2. Физические константы и размерные величины

Mathematica позволяет оперировать с *размерными величинами*, которые широко используются в физических и химических расчетах. Размерные величины характеризуются не только численными значениями, но и единицами измерения величин, например **Meter** (метр), **Second** (секунда) и т.д.:

1 Meter

Meter

5 Meter

5 Meter

0.5 Second

0.5 Second

Между значением размерной величины и единицей измерения знак умножения можно не ставить. Это видно из приведенных выше примеров.

Следует отметить, что без острой необходимости применять размерные величины не следует, поскольку они усложняют математические выражения и зачастую не позволяют выполнять с ними символьные преобразования. Рекомендуется нормировать выражения (формулы) так, чтобы результаты их вычисления имели безразмерный вид.

3.5. Работа с переменными

3.5.1. Расширенное понятие о переменных

Под *переменными* в математике принято называть объекты, которые могут принимать различные значения, находящиеся в определенном допустимом множестве значений. Переменные обычно снабжают именами-идентификаторами.

Подобно этому, переменными в системе Mathematica являются поименованные объекты, могущие в ходе выполнения документа неоднократно принимать различные значения – как численные, так и символьные. При этом символьные значения переменных, в отличие от обычных языков программирования, могут представлять собой как исполняемые математические выражения `expr`, так и некоторые обобщенные классы функций и объектов. Например, переменная может представлять графический объект, такой как изображение трехмерной поверхности, или звуковой объект, при активизации которого исполняется звук. Значением переменных могут быть также и множественные объекты-списки.

По характеру их видимости в ноутбуке переменные подразделяются на глобальные и локальные. К *глобальным переменным* доступ возможен в любой части ноутбука. Присвоенное глобальной переменной значение сохраняется до нового присваивания. *Локальные переменные* обычно применяются в локальных объек-

тах, например, в функциях пользователя и процедурах. За пределами этих объектов значения переменных могут быть другими или неопределенными.

Как уже отмечалось в конце главы 1, в Mathematica 6 возможно динамическое изменение глобальных переменных по всему ноутбуку, в частности, с применением средств GUI, например слайдеров, манипуляторов и т.д. Такие переменные должны входить в список параметров функции Dynamic.

3.5.2. Назначение переменным идентификаторов (имен)

Имена переменных называют их *идентификаторами*. Они должны быть уникальными, т.е. не совпадать с именами директив, атрибутов, опций и функций в ядре системы. Имена переменных должны начинаться с буквы. Общеприняты, скажем, имена *x* и *y* для функциональной зависимости *y(x)* или представления графиков, *f* – для функций. Желательно назначать именам переменных смысловые значения, например *xcoordinate* или *ycoordinate* для координат точки. Все сказанное об идентификаторах объектов справедливо и для идентификаторов переменных, поскольку переменные – наиболее распространенные виды объектов.

3.5.3. Особенности применения переменных

В отличие от переменных в математике, каждая переменная в системе Mathematica, как и в любой системе программирования, всегда отождествляется с некоторой физической областью памяти, в которой и хранится значение переменной. Для уменьшения объема памяти применяются различные способы компактного размещения чисел в памяти. Необходимо помнить, что и имя переменной занимает определенную область памяти. Распределение памяти под переменные – динамическое. Это означает, что местоположение ячеек памяти в ходе выполнения задачи и объем памяти под ту или иную переменную не фиксированы, а меняются в ходе выполнения задачи.

Заранее объявлять типы переменных не требуется. Он определяется операцией присваивания переменной некоторого значения. Такой подход упрощает построение программ и естественен при использовании переменных в обычной математической литературе.

Без особых на то указаний переменные в системе Mathematica являются *глобальными*. Это означает, что после определения переменной ее значение можно изменить в любом месте документа или программы. Переменная появляется как действующий объект только после ее первого определения или задания. Определения переменных выполняются с помощью *операции присваивания*, вводимой знаком равенства:

var = value

Здесь var – имя переменной, value – ее значение. Ниже представлены основные операции по присваиванию переменным их значений:

x = value переменной x присваивается значение value,
x = y = value значение value присваивается переменным x и y,
x:=value отложенное присваивание переменной x значения value,
x = . с переменной x снимается определение.

Примеры (комментарий In[...] при строке ввода опущен):

- **g = Plot[Sin[x],{x,0,20}]** – переменной g присваивается значение в виде графического объекта;
- **y = 1 + x^2** – переменной y присваивается символьное значение в виде алгебраического выражения $(1 + x^2)$;
- **z = {1, 2, x, a + b}** – переменной z присваивается значение в виде списка, содержащего четыре элемента.

Различие в присваивании переменным значений с помощью знаков **=** и **:=** иллюстрируют следующие примеры:

```
a=12
12
b:=15
b
15
a=.
a
a
```

Особо обратите внимание на то, что возможно снятие с переменной определения с помощью символа «=.». В символьной математике это очень полезная возможность, поскольку нередко переменные с одним и тем же именем в разных частях программы могут иметь разный смысл и представлять собой объекты, требующие значительных затрат памяти.

Более того, эти объекты сохраняются даже при использовании команды **New** при переходе к подготовке нового документа. Поэтому рекомендуется всякий раз снимать определения переменных, как только использование их завершается. Это предотвращает возникновение конфликтов между одноименными переменными и освобождает память.

3.5.4. Эволюция значений переменных и операции присваивания

Специфику математических выражений в системе Mathematica составляет возможность их изменения – *эволюции*, в соответствии с заложенными в ядро системы правилами математических преобразований. В итоге после эволюции значение выражения, которое присваивается переменной, может быть совсем иным, чем до эволюции. Поэтому в целом для определения переменных используют описанные ниже конструкции.

Основная функция **Set[lhs,rhs]** имеет аналогичные по действию упрощенные операторы:

- **lhs = rhs** – вычисляет правую часть rhs и присваивает ее значение левой части lhs. С этого момента lhs замещается на rhs всюду, где бы ни появилось;
- **{l1, l2, ...} = {r1, r2, ...}** – вычисляет ri и назначает полученные результаты соответствующим li.

Функция задержанного присваивания **SetDelayed[lhs, rhs]** может быть заменена аналогичными по действию операторами:

lhs := rhs – назначает правой части rhs роль отложенного значения левой части lhs. При этом rhs содержится в невычисленной форме. Когда появляется lhs, оно заменяется на rhs, вычисляемое каждый раз заново.

При отложенном присваивании вывода нет, тогда как при обычном немедленном присваивании **lhs=rhs** значение rhs вычисляется немедленно, и результат выводится в строку вывода.

Функция присваивания верхнего уровня **UpSet[lhs, rhs]** применяется в виде:

lhs^=rhs – левой части lhs присваивает значение правой части rhs и связывает это назначение с символами, которые появляются на первом уровне в lhs.

И, наконец, задержанную функцию присваивания **UpSetDelayed[lhs, rhs]** может заменить оператор:

lhs^:=rhs – величина rhs выполняет роль отложенного значения lhs, и связывает это присвоение с символами, которые появляются на первом уровне в lhs.

Отметим еще одну важную конструкцию:

SetOptions[s, name1->value1, name2->value2, ...] – устанавливает для символа s указанные опции, определяемые по умолчанию.

Применение различных типов операций присваивания способствует большей гибкости системы. Различия между этими операциями с первого взгляда несущественны, но они принципиальны, и это станет понятно после более детального знакомства с символьными преобразованиями и практики работы с системой.

В ходе эволюции математических выражений могут появляться весьма сложные результаты, занимающие в памяти много места. Если они приписаны к какой-либо переменной, то и она занимает в памяти много места. Есть много примеров символьных преобразований, способных привести к эффекту разбухания промежуточных или конечных результатов вычислений. Поэтому в системе предусмотрен оперативный контроль над свободной памятью. Чем она больше, тем более сложные задачи в состоянии решать система.

Имеются также системные переменные, значениями которых являются данные о системе и ее работе, например, версия применяемой операционной системы, текущая дата, время в данный момент, машинная точность вычислений и т.д. Многие из таких переменных имеют отличительный знак \$ перед своим именем. Такие переменные более подробно будут рассматриваться в дальнейшем.

3.5.5. Предполагаемые переменные

Иногда вычисления невозможны, если не предполагать наличие у переменных определенных свойств. Например, при упрощении корня квадратного из x в квадрате не слишком сведущего в математике пользователя ждет «странный» результат:

Simplify[Sqrt[x^2]]

$$\sqrt{x^2}$$

Повторение исходного выражения связано с тем, что ожидаемый результат – значение x возможен только при положительных или вещественных x . Таким образом, надо сделать x *предполагаемой переменной*, которая должна иметь только положительные значения. Это можно сделать, например, так:

Simplify[Sqrt[x^2], x>=0]
x

Переменную или несколько переменных из списка можно сделать предполагаемыми с помощью функции-директивы:

Element[x, dom] или **Element[{x1, x2, x3,...}, dom]**,

где опция **dom** задает область определения переменной или переменных в списке. Для Mathematica 6 **dom** задает следующие типы переменных: **Algebraics** – алгебраическая, **Boolean** – логическая, **Complexes** – комплексная, **Integers** – целочисленная, **Primes** – простое число, **Racional** – рациональная, **Reals** – вещественная. Пример:

FullSimplify[{Re[Sin[x]], Re[ArcSin[x]], Sqrt[x^2]}, Element[x, Reals]]
{Sin[x], Re[ArcSin[x]], Abs[x]}

3.6. Применение подстановок

3.6.1. Назначение подстановок

Важное значение в числовых и символьных преобразованиях имеют операции *подстановки*. Их смысл заключается в замене символьного значения заданной переменной на другое (числовое или символьное) значение. Например, часто возникает необходимость вычислить значение некоторого математического выражения при замене некоторой переменной ее конкретным численным значением. Для этого достаточно вместо этой переменной подставить нужное численное значение.

Куда менее тривиальной является замена переменной ее символьным значением в виде математического выражения. При этом исходное выражение может в ходе решения задачи превратиться в совершенно новое выражение, поскольку система может провести над исходным выражением после подстановки достаточно сложные математические преобразования. Говорят, что в этом случае ячейка, содержащая выражение (а точнее – само выражение), эволюционирует, т.е. изменяется по ходу решения задачи.

3.6.2. Подстановки с помощью оператора /.

Операции подстановки обычно вводятся с помощью комбинированного символа «/.»:

- **expr /. x -> value** – в выражение **expr** вместо переменной **x** подставляется ее значение **value**;

- **expr /. {x -> xvalue, y -> yvalue}** – в выражение expr вместо переменных x и y подставляются их значения xvalue и yvalue.

Примеры:

```
1+x^3/.x->1+z
1+(1+z)^3
x^2+2*x+3/.x->2
11
```

Обратите внимание на то, что в результате вместо переменной x оказалось математическое выражение $(1 + z)$. Второй пример иллюстрирует подстановку на место переменной x ее численного значения.

3.6.3. Подстановки с помощью операторов **->** и **:**

В целом, для операций подстановок (rule) используют следующие обозначения:

- **lhs -> rhs** – прямая подстановка lhs в rhs;
- **lhs :> rhs** – задержанная подстановка (RuleDelayed), которая преобразует lhs в rhs, вычисляя rhs только в том случае, когда используется правило подстановки.

Ниже приведено еще два примера на использование операций подстановки.

```
p:=1+x^2+3*x^3
p/.x->1+y
1+(1+y)^2+3(1+y)^3
f[n_]:=n^2
f[4]+f[y]+f[x+y]
16+y^2+(x+y)^2
```

В первом примере подстановка произведена в математическое выражение, а в другом – в список.

Подстановки – мощный и необычайно гибкий инструмент в системе Mathematica. С их помощью можно задать даже новые математические закономерности и произвольные соотношения (к примеру, можно задать абсурдное правило, что $2 + 2 = 5$). Эти необычные возможности мы рассмотрим в дальнейшем.

3.7. Задание и применение функций пользователя

3.7.1. Задание функций пользователя

Хотя в систему входят сотни встроенных функций (начиная от элементарных и кончая специальными математическими функциями и системными функциями), нередко требуется расширить ее вводом новых функций, действие которых задается пользователем. Такие функции принято называть *функциями пользователя*.

(см. Главу 2). Для задания, опознавания и уничтожения функций пользователя используются следующие конструкции:

f(x_) := x^3 отложенное задание функции пользователя с именем f,
f(x_) = x^3 немедленное задание функции пользователя с именем f,
?f вывод информации о функции f,
Clear[f] уничтожение определения функции f.

В обозначениях вида `x_` знак `_` применяется для создания так называемых *образцов*, задающих локальные переменные в теле функции, в нашем примере это `x^3`. При этом в самом теле функции переменные обозначаются, как обычно, без знака образца. Он лишь указывает на особый статус переменных в ограниченном пространстве программы – в теле функции. Так, если вместо `x_` будет подставлено число 2, то `f(2)` будет возвращать `2^3`. Вне тела функции значение переменной `x` не изменяется. Переменная `x` может быть и не определенной: `x_` определяет переменную `x` только для тела функции. Более подробно создание образцов описано в дальнейшем.

3.7.2. Сохранение на диске и считывание функций пользователя

Mathematica позволяет *записать* введенные пользователем функции с их определениями на магнитный диск с помощью оператора:

Save["filename", f1, f2, ...]

После этого функция пользователя становится внешней функцией. При этом для ввода таких функций в текущий документ (notebook) достаточно вызвать файл с именем `filename`:

<<filename

Рекомендуется создавать файлы с типовым расширением `.m`. Такие файлы входят в пакеты расширений системы. Имя файла нужно задавать по общепринятым для MS-DOS правилам, т.е. при необходимости указывать логическое имя дисководов и путь к файлу, например:

<<D:\MAT\myfunc.m

Создание внешних функций по существу означает возможность расширения системы и ее адаптации к решению типовых задач конкретного пользователя. Как уже отмечалось, в систему входит мощная библиотека внешних расширений, и каждый пользователь может пополнить ее своими библиотеками расширений.

3.7.3. Задание функций пользователя с синтаксисом языков программирования

Функции пользователя можно задавать и выводить на печать как на языке системы, так и на некоторых общепринятых языках программирования, например Fortran, C или TeX. Для этого существует ряд функций преобразования, в имена

которых входит слово **Form** (форма) и название языка для записи функций. Среди основных **CForm[expr]**, **FortranForm[expr]** и **TeXForm[expr]**. С их помощью выражения можно преобразовать в форму, принятую для языков программирования C, Fortran и TeX. При преобразовании в форму языка TeX греческие буквы заменяются их латинскими именами, например *alfa*, *APLHA*, *beta*, *BETA*, *gamma* и т.д. К сожалению, в отличие от систем класса MathCAD и Maple V R3, вывод математических формул в их полностью естественном виде не предусмотрен.

Для подсоединения созданных на этих языках программ, содержащих формулы и данные, в вывод системы Mathematica, используются функции:

- **Splice["file.mx"]** – включает вывод системы во внешний файл;
- **Splice["infile", "outfile"]** – включает вывод системы в файл *infile* и отправляет результат в файл *outfile*.

Таким образом, система Mathematica может общаться с другими программами, написанными на языках программирования, получивших распространение в практике реализации математических расчетов. Этому во многом способствует возможность преобразования форматов данных и результатов вычислений в различную форму, характерную для используемой внешней системы. К примеру, если вы работаете с программами на языке Фортран, то следует использовать Фортран-формат представления данных и результатов вычислений.

Система может общаться также с иными системами, например текстовыми редакторами. К примеру, для передачи в текстовый редактор Write, входящий в оболочку Windows, содержимого каких-либо ячеек достаточно выделить эти ячейки и поместить их в буфер Clipboard, используя операцию **Copy** в позиции **Edit** главного меню системы Mathematica. После этого необходимо в многозадачном режиме запустить текстовый редактор, и с помощью команды **Paste** в позиции **Edit** главного меню редактора поместить в окно редактирования содержимое ячеек. Если оно символьное, то с помощью редактора записать полученный документ с расширением *.txt*, т.е. в стандартном текстовом формате, с которым работает большинство DOS-приложений.

3.8. Средства арифметических вычислений

3.8.1. Арифметические операторы

Математические выражения в системе Mathematica записываются с помощью операторов и функций. *Операторы* (от слова *operator* – исполнитель) являются элементами записи математических выражений, указывающими на то, какие действия производятся над символьными или числовыми данными. Когда эти данные используются совместно с операторами, их называют *операндами*.

Выражения, составленные из операторов, операндов и функций, способны возвращать результат вычисления выражений. К примеру, если вычисляется сумма $2 + 3$, то знак $+$ является оператором, числа 2 и 3 – операндами, а $2 + 3$ – выражением. Сами по себе операторы не возвращают какого-либо значения.

Существуют общепринятые *приоритеты* исполнения операций, например: в первую очередь сложение и вычитание, затем умножение и деление, и далее другие операции. С помощью круглых скобок можно изменять приоритет операций, например, в выражении $(2+3)*4$ вначале будет вычислено $2+3$, а затем уже результат будет умножен на число 4.

Ниже перечислены основные операторы для выполнения арифметических операций (x , y и z – операнды, задающие данные, над которыми выполняются вычисления):

$x+y+z$	сложение,
$x-y-z$	вычитание,
$x*y*z$ или $x\ y\ z$	перемножение,
x/y	деление,
x^y	возведение x в степень y ,
<code>expr //N</code>	дает приближенное (с установленной точностью и формой) значение выражения <code>expr</code> .

Полезно отметить, что знак пробела является арифметическим оператором умножения, если по обе стороны от него стоят операнды.

3.8.2. Особенности выполнения арифметических операций

Как уже отмечалось, при выполнении вычислений особая роль принадлежит символам `%`. Символы `%`, как сами по себе, так и в качестве аргументов функций, используются для указания на применение результата предшествующих операций:

- `%` – возвращает результат последней предшествующей операции;
- `%%` – возвращает результат операции, выполняемой перед последней предшествующей операцией;
- `%..%` – возвращает результат операции, выполненной в линии, отстоящей от конца на число повторений символа `%`;
- `%n` – возвращает результат операции в строке n .

Нетрудно заметить, что применение этих символов облегчает выполнение последовательных вычислений.

Как уже отмечалось, для представления арифметических выражений `expr` в виде вещественного результата используется функция `N[expr,m]`. Можно также задать вычисление любого выражения в численном виде, используя выражение:

`expr //N`

Примеры:

`1/3+2/7`

$\frac{13}{21}$

`1/3+2/7 //N`

0.619048

Таким образом, используя функцию **N[expr,m]** или вывод с помощью комбинированного символа **//N**, можно организовать вычисления в режиме калькулятора, находясь в среде оболочки системы.

Если x имеет вещественное значение, то функция

MantissaExponent[x]

возвращает список, содержащий мантиссу и порядок приближенного вещественного числа x . Примеры:

123.456 10^10

1.23456×10^{12}

MantissaExponent[%]

{0.123456, 13}

3.8.3. Рационализация чисел

Следующие две функции:

Rationalize[x] и **Rationalize[x,dx]**

дают приближение для числа x в виде рациональных чисел. Вторая из этих функций задает приближение с заданной точностью.

Mathematica может работать с большими цифрами и выполнять определенные операции с очень высокой точностью. Примеры, приведенные ниже, иллюстрируют эти возможности.

2+3^100

515377520732011331036461129765621272702107522003

Rationalize[N[Pi], 10^-3]

$\frac{355}{113}$

Rationalize[N[Pi], 10^-9]

$\frac{104348}{33215}$

Rationalize[N[Pi], 10^-12]

$\frac{5419351}{1725033}$

(100!+1/2)-100!

$\frac{1}{2}$

N[100!]

9.33262×10^{157}

Как видно из примеров представления рациональных чисел, результат приближения зависит от заданной погрешности. Чем она меньше, тем большие значения целых чисел в числителе и знаменателе результата разыскивает система. Функция **Rationalize** открывает широкие возможности для разработки целочисленных алгоритмов вычислений, позволяя легко получать рациональные приближения для наиболее распространенных числовых констант (выше примеры этого даны для числа π).

Обратите внимание на последние два примера – вычисление факториала достаточно большого числа. Первый результат целочисленный: хотя он занял три строки – он точен. С помощью функции **N[expr]** результат всегда можно представить в виде большого вещественного числа, но приближенного. При этом может использоваться научная форма представления чисел – с мантиссой и порядком.

3.8.4. Укороченная форма записи арифметических операций

Спецификой систем Mathematica являются арифметические операторы с *укороченной формой записи*, объединяющие операцию присваивания с арифметической операцией. Эти довольно специфические операторы, хорошо известные пользователям языка C, представлены ниже вместе с соответствующими им функциями:

Функция	Оператор	Назначение
Increment[i]	i++	увеличивает значение i на 1 до использования i в выражении
Decrement[i]	i--	уменьшает значение i на 1 до использования i в выражении
PreIncrement[i]	++i	увеличивает значение i на 1 после использования i в выражении
PreDecrement[i]	--i	уменьшает значение i на 1 после использования i в выражении
AddTo[x,d]	x += dx	прибавляет dx к x и возвращает новое значение x
SubtractFrom[x,dx]	x -= dx	отнимает dx от x и возвращает новое значение x
TimesBy[x,c]	x *= c	умножает x на c и возвращает новое значение x
DivideBy[x,c]	x /= c	делит x на c и возвращает новое значение x

Применение укороченных операторов делает более короткой запись математических выражений, хотя наглядность их при этом несколько снижается. Примеры выполнения укороченных арифметических операций:

Ввод (In)	Вывод (Out)
i=0	0
++i; ++i; ++i	3
i=0; i++; i++; i++	2
i=5	5
-i	4
i=5	5
i-	5
i-	4
x=5	5
x+=0.5	5.5
x-=0.5	5.
x*=2	10.
x/=5	2.

3.9. Функции арифметических операций

3.9.1. Встроенные функции

Важнейшим объектом любой математической системы является *функция*. Она отражает зависимость некоторой величины от одного или нескольких аргументов. Например, функция $\sin(x)$ дает зависимость синуса x от величины аргумента x при изменении последнего от $-\infty$ до $+\infty$.

Признаком функции является возврат результата выполняемого ею действия. Характер результата будет зависеть от статуса функции, который нередко явно указывается ее именем-идентификатором. Например, функция **DigitsInteger[n]** возвращает число десятичных цифр десятичного целого числа. Это ясно из прямого перевода имени функции. Подобные смысловые имена задаются для большинства функций системы Mathematica и облегчают их запоминание.

Понятие функции в системе Mathematica существенно расширено: функции могут возвращать графические и даже звуковые объекты. Здесь мы, однако, остановимся на общепринятом в программировании понятии функций, возвращающих численные или символьные значения в ответ на обращения к ним.

Функции могут входить в состав математических выражений. Обычно они имеют один или несколько параметров, указываемых в квадратных скобках. Если параметров несколько, то в квадратных скобках указывается список параметров, разделенных запятыми. В общем случае параметрами могут быть списки. Наконец, в состав функций могут входить опции, указанные своим именем и (после знака \rightarrow) значением. Для обозначения положительной бесконечности используется символ Infinity. Целочисленные функции имеют в своем имени слово Integer.

3.9.2. Основные арифметические функции

Для выполнения арифметических действий определены следующие *арифметические функции*:

- **Divide[x, y]** – возвращает результат деления x на y , эквивалентно выражению x/y .
- **Plus[x, y, ...]** – возвращает сумму элементов списка.
- **PowerMod[a, b, n]** – возвращает **Mod[a^b, n]**. Для $b < 0$ возвращает инверсию остатка.
- **Times[x, y, ...]** – возвращает произведение аргументов $x*y*...$
- **Mod[m, n]** – возвращает остаток от деления m на n . Результат имеет такой же знак, как n .

Примеры на применение арифметических функций:

Ввод (In)	Вывод (Out)
Divide[1., 3]	0.333333
Mod[123, 20]	3
Mod[123, -20]	-17

Ввод (<i>In</i>)	Вывод (<i>Out</i>)
Mod [-123, 20]	17
Plus [2, 3, 4]	9
Times [2, 3, 4]	24

Для обмена значениями переменных x и y можно использовать выражение:

{x, y}={y, x}

Пример на обмен переменных значениями:

a=1; b=2;

{a, b}={b, a};

{a, b}

{2, 1}

Следующие функции служат для приведения вещественных чисел к ближайшим целым по определенным правилам:

- **Ceiling[x]** – возвращает значение наименьшего целого числа, большего или равного x .
- **Floor[x]** – возвращает наибольшее целое число, не превышающее данного x .
- **Quotient[n, m]** – возвращает целое значение n/m , определяемое как **Floor[n/m]**.
- **Round[x]** – округляет x до ближайшего целого.

Хотя аргументами этих функций указано значение x , под ним можно понимать список вещественных чисел. Следующие примеры поясняют это и наглядно иллюстрируют правила приведения к целым числам:

Ввод (<i>In</i>)	Вывод (<i>Out</i>)
Ceiling [{- .5.9, -5.1, 5.5.1, 5.9}]	{-5, -5, 5, 6, 6}
Floor [{- .5.9, -5.1, 5.5.1, 5.9}]	{-6, -6, 5, 5, 5}
Round [{- .5.9, -5.1, 5.5.1, 5.9}]	{-6, -5, 5, 5, 6}

Ряд функций обеспечивает нахождение делителей целых чисел и наименьшее общее кратное:

- **Divisors[n]** – возвращает список целочисленных делителей числа n .
- **DivisorSigma[k, n]** – возвращает сумму k -тых степеней положительных делителей числа n .
- **ExtendedGCD[n, m]** – возвращает расширенный наибольший общий делитель целых чисел n и m .
- **GCD[n1, n2, ...]** – возвращает наибольший общий делитель целых чисел ni .
- **LCM[n1, n2, ...]** – возвращает наименьшее общее кратное целых чисел ni .

Примеры на применение этих функций:

Ввод (<i>In</i>)	Вывод (<i>Out</i>)
Divisors [123]	{1, 3, 41, 123}
DivisorSigma [17, 3]	129140164
ExtendedGCD [144, 12]	{12, {0, 1}}
GCD [144, 12, 6]	6
LCM [124, 12, 6]	372

К целочисленным функциям можно отнести функции вычисления факториала и двойного факториала:

- **Factorial[n]** или **n!** – возвращает значение факториала числа n ($n! = n \cdot (n-1) \cdot \dots \cdot 3 \cdot 2 \cdot 1$, причем $0! = 1$ и $1! = 1$).
- **Factorial2[n]** или **n!!** – возвращает значение двойного факториала числа n , равное $n \cdot (n-2) \cdot (n-4) \cdot \dots$

Примеры на вычисление факториалов:

Ввод (In)	Вывод (Out)
Factorial[10]	3628800
20!	2432902008176640000
10!!	3840
20!//N	2.4329×10^{18}

Mathematica способна вычислять факториалы больших чисел. Практически мгновенно вычисляются значения до $1000!$, хотя результат при этом занимает несколько страниц на экране дисплея. Обратите внимание на то, что управляющий символ **//N** за выражением дает вывод (аппроксимацию) в форме научной нотации.

Следующие функции служат для получения простых чисел и некоторых их характеристик:

- **Prime[n]** – возвращает n -ое простое число. Пример: **Prime[5]** возвращает пятое простое число – 11. Всего несколько секунд необходимо системе для вычисления миллиардного простого числа: **Prime[10^9]** – 22801763489.
- **PrimePi[x]** – возвращает количество простых чисел, не превышающих x . Пример: **PrimePi[10]** возвращает 4.
- **PartitionsP[n]** – возвращает число $p(n)$ неупорядоченных разбиений целого числа n . Пример: **PartitionsP[10]** возвращает 42.
- **PartitionsQ[n]** – возвращает $q(n)$ – число разбиений с неравными частями для целого числа n . Пример: **PartitionsQ[15]** возвращает 27.

Эти функции полезны при решении задач теории чисел. Число таких функций в Mathematica 6 заметно расширено. Например, для генерации номера следующего за заданным числом простого числа можно воспользоваться новой функцией **NextPrime[n,k]**. Какое по номеру следующее число следует за миллиардом? Ответ:

```
NextPrime[10^9]
10000000007
```

А какое будет предшествующее число? Ответ:

```
NextPrime[10^9,-1]
9999999937
```

3.9.3. Функции генерации случайных чисел

Для реализации статистических методов моделирования используются случайные числа. Система имеет генератор псевдослучайных чисел, доступ к которому обеспечивают следующие функции:

- **Random[]** – возвращает равномерно распределенное псевдослучайное число типа Real в интервале от 0 до 1.
- **Random[type, range]** – дает псевдослучайное число указанного типа type, лежащее в указанном интервале range. К возможным типам относятся: Integer, Real и Complex. По умолчанию принят интервал от 0 до 1. Можно задать интервал явно {min, max}; спецификация интервала от max эквивалентна {0, max}.
- **SeedRandom[n]** – сбрасывает (восстанавливает, устанавливает в начальное состояние – resets) генератор случайных чисел, используя целое n как начальное число.
- **SeedRandom[]** – устанавливает генератор, используя в качестве начального числа время дня (the time of day).

Хотя генерируемые числа не являются строго случайными, их количество в повторяющейся последовательности очень велико. Использование специальной установки начальной базы генератора, например, по времени дня, делает повторение последовательности практически невозможным.

Для проверки равномерности распределения большого массива случайных чисел можно задать с их помощью случайные координаты, и затем построить точки, соответствующие координатам (x,y). Рисунок 3.1 наглядно показывает, как это делается для массива из 10 000 случайных точек. О равномерности распределения случайных точек говорит равномерность распределения плотности точек на графике.

В Mathematica 6 есть и функция генерации случайных простых чисел **RandomPrime**.



Рис. 3.1. Графическая иллюстрация распределения точек со случайными координатами (x,y)

3.9.4. Функции выявления погрешностей и анализа структуры чисел

Следующие функции, опции и директивы используются, в основном, для выявления погрешностей вычислений и уточнения структуры чисел:

- **Accuracy[x]** – возвращает количество десятичных цифр справа от десятичной точки числа x .
- **EvenQ[expr]** – возвращает значение True, если expr есть четное число, и False – в противном случае.
- **IntegerDigits[n]** – возвращает список десятичных цифр целого числа n .
- **IntegerDigits[n, b]** – возвращает список цифр целого n в записи по основанию b .
- **IntegerDigits[n, b, k]** – возвращает список длиной k , содержащий самые младшие (наименьшие) значащие цифры в n .
- **Precision[x]** – возвращает количество точных знаков в числе x .

Поясним применение этих функций следующими примерами:

Ввод (In)	Вывод (Out)
Accuracy[123.456]	14
EvenQ[2*3+2]	True
EvenQ[2*3+3]	False
IntegerDigits[12345]	{1, 2, 3, 4, 5}
IntegerDigits[12345,16]	{3, 0, 3, 15}
IntegerDigits[12352,16]	{3, 0, 4, 0}
IntegerDigits[12352,2]	{1,1,0,0,0,0,0,0,1,0,0,0,0,0,0}
Precision[123.452]	16

Функциями **Accuracy** и **Precision** возвращаются значения, установленные в последний раз или по умолчанию при первой загрузке системы.

В целом, Mathematica имеет обширный набор арифметических операторов и функций, достаточный для решения задач теории чисел и выполнения практически любых арифметических вычислений. Многие более специальные целочисленные функции будут рассмотрены в дальнейшем, по мере описания системы.

3.10. Логические операторы и функции

3.10.1. Логические операции

Логическими принято называть операции, отражающие чисто логическое соответствие между данными. В математике (да и в информатике) принято характеризовать логическое соответствие утверждениями True (Верно или Да) и False (Неверно или Нет). Слова True и False являются символьными константами, отражающими результаты логических операций и в системе Mathematica.

Для осуществления логических операций используются следующие *логические операторы*:

== равенство (например, $a == b$),
!= неравенство,
> больше (например, $b > a$),
>= больше или равно,
< меньше,
<= меньше или равно.

Возможны следующие формы применения операторов сравнения:

a == b == c

a != b != c

x < y < z

и так далее.

Результатом исполнения этих выражений является выдача логических значений True или False. Это демонстрируют следующие примеры:

Ввод (In)	Вывод (Out)
2==2	True
a==a	True
a==b	$a == b$
2==3	False
2<3	True
2>3	False
2!=3	True
2+1==3==4-1	True

3.10.2. Основные логические функции

Основные логические функции над логическими данными p, q и т.д. задаются следующим образом:

Not[p] или **!p** логическое отрицание;
And[p,q,...] или **p && q && ...** логическое сложение – операция «И»;
Or[p,q,...] или **p || q || ...** логическое умножение – операция «Или».
 Приведем примеры применения логических операторов и функций.

Ввод (In)	Вывод (Out)
And[True, True, True]	True
True && True && False	False
Not[True]	False
Not[False]	True
Or[True, True, False]	False
2==2 && 3==3	True
True && True	True
And[1, 1, 0]	1 1 0
And[1, 1, 0]	1 && 1 && 0

Эти примеры показывают, что аргументами логических функций и операндами логических операций должны быть только логические константы True и False или выражения, значения которых представлены ими. Недопустимо использовать численные значения 1 и 0, отождествляя их с логической единицей и логическим нулем. Результатом задания операций с ними будет повтор задания, возможно, в укороченной форме.

К основным логическим средствам относятся и следующие операторы и функции:

Equal [lhs, rhs]	возвращает True, если lhs и rhs тождественны;
Greater[x,y] или x > y	возвращает True, если x=y оказывается больше y, иначе возвращает False;
Greater[x1,x2,x3] или x1 > x2 > x3	возвращает True, если xi образуют строго убывающую последовательность, иначе возвращает False;
GreaterEqual[x,y] или x >= y	вырабатывает True, если x больше или равно y, иначе возвращает False;
GreaterEqual[x1,x2,x3] или x1>= x2 >= x3	вырабатывает True, если xi образуют невозрастающую последовательность, иначе возвращает False;
Negative[x]	возвращает True, если x оказывается отрицательным числом, иначе возвращает False;
NonNegative[x]	возвращает True, если x неотрицательное число, иначе возвращает False;
Positive[x]	возвращает True, если x положительное число, иначе возвращает False;
SameQ[lhs,rhs]	вырабатывает значение True, если выражение lhs или lhs === rhs тождественно rhs, иначе False;
Xor[e1, e2, ...]	является логической функцией XOR (исключающее OR). Возвращает True, если нечетное количество из ei имеют значение True, а остальные False. И возвращает False, если четное количество ei имеют значение True, а остальные False.

Приведем примеры на применение этих функций.

Ввод (In)	Вывод (Out)
Positive[2-3]	False
Equal[1+2,4-1]	True
Equal[1+2,2]	False
Greater[5,4]	True
Greater[5,4,3]	True
Greater[5,4,9]	False
Less[3,2+3]	True
Positive[2]	True
Negative[-2]	True
Negative[2]	False
NonNegative[-2]	False
NonNegative[2]	True
Xor[True,True]	False
Xor[False,False]	False
Xor[True,False]	True

3.10.3. Дополнительные логические функции

Для более полного использования возможностей логических операций в системе Mathematica имеются следующие дополнительные логические функции:

- **DigitQ[string]** – вырабатывает значение True, если все символы строки string являются цифрами от 0 до 9, иначе возвращает False.
- **Identity[expr]** – возвращает expr (операция тождественности).
- **Implies[p, q]** – представляет логическую импликацию $p \Rightarrow q$.
- **IntegerQ[expr]** – возвращает True, если expr является целым числом, иначе False.
- **LetterQ[string]** – вырабатывает True, если все символы строки string являются буквами, иначе False.
- **ListQ[expr]** – возвращает True, если expr является списком, иначе False.
- **LowerCaseQ[string]** – вырабатывает значение True, если все символы в строке являются строчными буквами (буквами нижнего регистра), иначе False.
- **MachineNumberQ[x]** – возвращает True, если x является числом в машинном формате с плавающей точкой, иначе возвращает False.
- **MatchQ[expr, form]** – возвращает True, если модель (образец) form соответствует expr, и возвращает False в противном случае.
- **NumberQ[expr]** – возвращает True, если expr является числом, иначе False.
- **OddQ[expr]** – возвращает True, если expr нечетное целое, иначе False.
- **OptionQ[e]** – возвращает True, если e может считаться опцией или списком опций, иначе False.
- **PrimeQ[expr]** – дает True, если expr является простым числом, иначе дает False.
- **TrueQ[expr]** – возвращает True, если expr имеет значение True, иначе возвращает False.
- **UnsameQ** – возвращает True, если выражение lhs применяется в виде: не-тождественно (неидентично) rhs, $lhs \neq rhs$ в противном случае возвращает False.
- **ValueQ[expr]** – возвращает True, если было определено значение (a value) для expr, иначе возвращает False.
- **VectorQ[expr]** – возвращает True, если expr является списком, но ни один из его элементов в то же время сам не является списком, иначе возвращает False.
- **VectorQ[expr, test]** – возвращает True только, если test, будучи применен к каждому элементу expr, дает True.

Вполне вероятно, что читатель уже готов самостоятельно проверить вполне очевидное действие этих функций: так, как это было сделано выше для многих, более распространенных логических функций.

В систему Mathematica, помимо указанных выше функций, дополнительно включены побитовые логические функции: **BitAnd[n1,n2,...]**, **BitOr[n1,n2,...]**, **BitXor[n1,n2,...]** и **BitNot[n]**. Их действие вполне очевидно.

3.11. Работа с математическими функциями

Mathematica способна выполнять вычисления с *элементарными функциями* и специальными математическими функциями. *Специальные математические функции* являются решениями линейных дифференциальных уравнений специального вида или представлениями особых интегралов, которые не могут быть выражены через элементарные функции. Здесь не приводятся определения специальных математических функций, ввиду их общеизвестности [17, 18] и наличия таких определений в справочной базе данных систем Mathematica.

3.11.1. Функции комплексного аргумента

Элементарные функции в системе Mathematica могут иметь аргумент в виде действительного числа x или комплексного z . Аргументы указываются как параметры функций в квадратных скобках.

Прежде всего, отметим функции для работы с комплексными числами z .

- **Abs[z]** – возвращает модуль комплексного числа.
- **Arg[z]** – возвращает аргумент комплексного числа z .
- **Conjugate[z]** – возвращает комплексно сопряженное с z число.
- **DirectedInfinity[]** – представляет бесконечную числовую величину с неопределенным направлением на комплексной плоскости.
- **DirectedInfinity[z]** – представляет бесконечную числовую величину, являющуюся положительной вещественной кратной комплексному числу z .
- **Im[z]** – возвращает мнимую часть комплексного числа z .
- **Re[z]** – возвращает вещественную часть числа z .

Ниже приведены примеры операций в непосредственном режиме с комплексными числами:

Ввод (In)	Вывод (Out)
z1:=2+I*3	
z2:=4+I*5	
N[z1+z2]	6. + 8.1 I
Re[2+I*3]	2
N[Im[z2]]	5.
N[z1/z2]	0.560976 + 0.0487805 I
N[Abs[z1*z2]]	23.0868
Conjugate[z1]	2 - 3 I

Если ввести **N[z1/0]**, то система выдаст следующее сообщение:

N[z1/0]

Power::infy : Infinite expression $\frac{1}{0}$ encountered. [More...](#)

ComplexInfinity

Итак, в этом случае система выдает сообщение об ошибке, но после него возвращает константу **ComplexInfinity**, означающую комплексную бесконечность.

3.11.2. Элементарные функции

Свойства *элементарных функций* хорошо известны [17, 18]. Полный список встроенных в ядро системы Mathematica элементарных функций, кстати, имеющих несколько отличные от математических обозначений имена, представлен ниже.

Abs[z] – возвращает абсолютное значение для действительного числа и модуль для комплексного z .

ArcCos[z] – возвращает арккосинус комплексного числа z .

ArcCosh[z] – возвращает значение обратного гиперболического косинуса комплексного аргумента z .

ArcCot[z] – возвращает значение арккотангенса комплексного аргумента z .

ArcCoth[z] – возвращает обратный гиперболический котангенс комплексного аргумента z .

ArcCsc[z] – возвращает арккосеканс комплексного аргумента z .

ArcCsch[z] – возвращает обратный гиперболический косеканс комплексного аргумента z .

ArcSec[z] – возвращает арксеканс комплексного аргумента z .

ArcSech[z] – возвращает обратный гиперболический секанс комплексного аргумента z .

ArcSin[z] – возвращает арксинус комплексного аргумента z .

ArcSinh[z] – возвращает обратный гиперболический синус комплексного аргумента z .

ArcTan[z] – возвращает арктангенс аргумента z .

ArcTan[x, y] – возвращает арктангенс от y/x вещественных x и y для квадранта, в котором лежит точка (x, y) .

ArcTanh[z] – возвращает обратный гиперболический тангенс комплексного аргумента z .

Cos[z] – возвращает косинус аргумента z .

Cosh[z] – возвращает гиперболический косинус аргумента z .

Cot[z] – возвращает значение котангенса аргумента z .

Coth[z] – возвращает гиперболический котангенс аргумента z .

Csc[z] – возвращает значение косеканса z .

Csch[z] – возвращает гиперболический косеканс z .

Exp[z] – возвращает значение $\exp(z)$.

Log[z] – возвращает натуральный логарифм аргумента z (логарифм по основанию E).

Log[b, z] – возвращает логарифм по основанию b .

Sec[z] – возвращает секанс аргумента z .

Sech[z] – возвращает гиперболический секанс z .

Sign[x] – возвращает -1 , 0 или 1 , если аргумент x соответственно отрицательный, ноль или положительный.

Sign[z] – возвращает признак комплексного числа z .

Sin[z] – возвращает синус аргумента z .

Sinh[z] – возвращает гиперболический синус z .

Sqrt[z] – возвращает корень квадратный из аргумента z .

Tan[z] – возвращает тангенс аргумента z .

Tanh[z] – возвращает гиперболический тангенс z .

В связи с общеизвестностью элементарных функций ограничимся приведением лишь нескольких примеров на их использование:

Ввод (<i>In</i>)	Вывод (<i>Out</i>)
Sqrt[2]	Sqrt[2]
Sqrt[2.]	1.41421
2*Sin[1]	2 Sin[1]
N[2*Sin[1]]	1.68294
Log[Exp[1]]	1
Simplify[Sin[x]/Cos[x]]	Tan[x]
ComplexExpand[Sin[a+b*I]]	Cos[b] Sin[a] + I Cos[a] Sinh[b]

Из трех последних примеров видно, что система Mathematica знает и использует основные соотношения между элементарными функциями. В двух последних примерах используются символьные преобразования с применением функций **Simplify** (упрощение выражений) и **ComplexExpand** (расширение выражений с комплексным аргументом). Более подробно эти важные для символьных операций функции будут рассмотрены в дальнейшем.

Mathematica вычисляет все элементарные функции как с действительным, так и с комплексным аргументом. Набор функций функционально полный, т.е. некоторые из отсутствующих функций всегда можно вычислить через представленные в ядре системы.

3.11.3. Ортогональные многочлены

Одними из широко распространенных специальных функций являются ортогональные многочлены (полиномы). Mathematica имеет следующие функции, возвращающие значения ортогональных многочленов:

- **ChebyshevT[n, x]** – Чебышева n -й степени первого рода.
- **ChebyshevU[n, x]** – Чебышева n -й степени второго рода.
- **HermiteH[n, x]** – Эрмита n -й степени.
- **JacobiP[n, a, b, x]** – Якоби n -й степени.
- **GegenbauerC[n, m, x]** – Гегенбауэра.
- **LaguerreL[n, x]** – Лагерра n -й степени.
- **LaguerreL[n, a, x]** – обобщенного Лагерра n -й степени.
- **LegendreP[n, x]** – n -го порядка Лежандра.
- **LegendreP[n, m, x]** – присоединенного полинома Лежандра.
- **LegendreQ[n, z]** – n -го порядка функция Лежандра второго рода.
- **LegendreQ[n, m, z]** – присоединенная функция Лежандра второго рода.

- **LegendreType** – опция для **LegendreP** и **LegendreQ**; она указывает выборы разрывов кривой для функций Лежандра на комплексной плоскости.

Все ортогональные полиномы имеют простые рекуррентные представления. Поэтому приведенные выше функции вычисляются по ним довольно быстро и точно. Они находят широкое применение в технике интерполяции и аппроксимации функций.

Следующие примеры иллюстрируют работу с ортогональными многочленами:

Ввод (In)	Вывод (Out)
	2 4 6 8
ChebyshevT [8,x]	$1 - 32x + 160x^2 - 256x^4 + 128x^6$
ChebyshevT [5,0.2]	0.84512
ChebyshevU [3,0.15]	-0.573
HermiteH [4,3]	876
JacobiP [3,1,2,0.2]	-0.256
	3
GegenbauerC [3,1,x]	$-4x + 8x^3$
	2 3
N [LaguerreL [3,x]]	$0.166667 (6. - 18. x + 9. x^2 - 1. x^3)$
	3 5
	$15x - 70x^3 + 63x^5$
LegendreP [5,x]	-----
	8
LegendreQ [2,0.2]	-0.389202

Важно отметить, что при указании конкретного значения параметра n и символьном значении параметра x функции этой группы возвращают присущие им представления через степенные многочлены с соответствующими коэффициентами.

3.11.4. Интегральные показательные и родственные им функции

К другой известной группе специальных функций относятся интегральные показательные и родственные им функции:

- **CoshIntegral**[x] – гиперболический интегральный косинус.
- **CosIntegral**[x] – интегральный косинус $\text{Ci}(x)$.
- **Erf**[z] – функция ошибок (интеграл вероятности).
- **Erf**[z0, z1] – обобщенная функция ошибок $\text{erf}(z1) - \text{erf}(z0)$.
- **Erfc**[z] – дополняющая функция ошибок $1 - \text{erf}(z)$.
- **Erfi**[z] – мнимое значение функции ошибок $-i \text{erf}(iz)$.
- **ExpIntegralE**[n, z] – интегральная показательная функция $E(n, z)$.
- **ExpIntegralEi**[z] – интегральная показательная функция $Ei(z)$.
- **LogIntegral**[z] – интегральный логарифм $\text{li}(z)$.
- **SinhIntegral**[x] – интегральный гиперболический синус.
- **SinIntegral**[x] – интегральный синус $\text{Si}(x)$.

Примеры на применение этих функций:

Ввод (In)	Вывод (Out)
<code>CoshIntegral[1.]</code>	0.837867
<code>CosIntegral[1.]</code>	0.337404
<code>Erf[1.]</code>	0.842701
<code>Erf[2.+I*3.]</code>	-20.8295 + 8.68732 I
<code>Erf[2.,3.]</code>	0.00465564
<code>Erfc[1.]</code>	0.157299
<code>Erfi[1.]</code>	1.65043
<code>ExpIntegralE[3,1.]</code>	0.109692
<code>ExpIntegralEi[1.]</code>	1.89512
<code>LogIntegral[2.+3.*I]</code>	2.3374 + 2.51301 I
<code>SinhIntegral[1.]</code>	1.05725
<code>SinIntegral[1.]</code>	0.946083

Следует обратить внимание на то, что большая часть из этих функций может иметь комплексный аргумент. Для получения численных значений функций необходимо задавать аргумент в форме вещественного числа или комплексного числа с вещественными действительной и мнимой частями.

3.11.5. Гамма- и полигамма-функции

Широко используется гамма-функция и относящиеся к ней родственные функции:

- **Gamma[a]** – Эйлерова гамма-функция.
- **Gamma[a, z]** – неполная гамма-функция.
- **Gamma[a, z0, z1]** – обобщенная неполная гамма-функция $\Gamma(a, z0) - \Gamma(a, z1)$.
- **GammaRegularized[a, z]** – регуляризованная неполная гамма-функция $Q(a, z) = \Gamma(a, z) / \Gamma(a)$.
- **GammaRegularized[a, z0, z1]** – обобщенная неполная гамма-функция $Q(a, z0) - Q(a, z1)$;
- **LogGamma[z]** – логарифм Эйлеровой гамма-функции.
- **PolyGamma[z]** – дигамма-функция $\psi(z)$.
- **PolyGamma[n, z]** – n-ая производная от дигамма-функции.

Приведем примеры на вычисление этих функций.

Ввод (In)	Вывод (Out)
<code>Gamma[0.5]</code>	1.77245
<code>Gamma[1,2.+3.*I]</code>	-0.133981 - 0.0190985 I
<code>Gamma[1,2.,3.]</code>	0.0855482
<code>GammaRegularized[1,2.+3.I,4.+6.*I]</code>	-0.139176 - 0.0366618 I
<code>LogGamma[0.5]</code>	0.572365
<code>LogGamma[2.+3.*I]</code>	-2.09285 + 2.3024 I
<code>PolyGamma[1]</code>	-EulerGamma
<code>PolyGamma[1.]</code>	-0.577216
<code>PolyGamma[2.+3.*I]</code>	1.20798 + 1.10413 I

Как видно из этих примеров, данный класс функций (как и многие другие) определен в общем случае для комплексного значения аргумента.

О поведении специальных функций многое говорит их график. К примеру, на рис. 3.2 показан график гамма-функции. Читатель может легко построить графики других функций системы Mathematica.

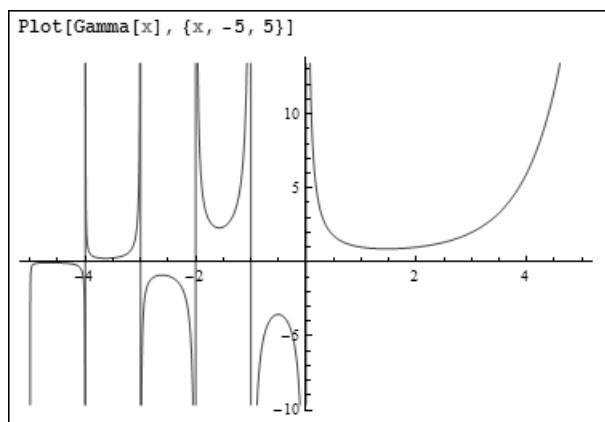


Рис. 3.2. Пример построения графика гамма-функции

3.11.6. Функции Бесселя

Функции Бесселя, являющиеся решениями линейных дифференциальных уравнений вида $z^2 y'' + zy' + (z^2 - n^2)y = 0$, широко используются в анализе и моделировании волновых процессов. В системе Mathematica к этому классу относятся следующие функции:

BesselI[n, z] – модифицированная функция Бесселя первого рода $I(n, z)$.

BesselJ[n, z] – функция Бесселя первого рода $J(n, z)$.

BesselK[n, z] – модифицированная функция Бесселя второго рода $K(n, z)$.

BesselY[n, z] – функция Бесселя второго рода $Y(n, z)$.

Соотношения между этими функциями хорошо известны (см. [84] и справочную базу данных системы Mathematica). Следующие примеры показывают вычисление функций Бесселя:

Ввод (In)	Вывод (Out)
BesselI[0,1.]	1.26607
BesselI[3,1.]	0.0221684
BesselI[1,2.+3.*I]	-1.26098 + 0.780149 I
BesselJ[2,2.+3.*I]	1.25767 + 2.31877 I
BesselK[2,2.+3.*I]	-0.0915555 + 0.0798916 I
BesselY[2,2.+3.*I]	-2.3443 + 1.27581 I
N[BesselJ[1,0.5]]	0.242268
N[BesselJ[1,2+I*3]]	3.78068 - 0.812781 I

3.11.7. Гипергеометрические функции

Класс гипергеометрических функций представлен в системе Mathematica следующими встроенными в ядро функциями:

- **HypergeometricU[a, b, z]** – конфлюэнтная (вырожденная) гипергеометрической функции $U(a, b, z)$.
- **Hypergeometric0F1[a, z]** – гипергеометрическая функция $F_0(a, z)$.
- **Hypergeometric1F1[a, b, z]** – вырожденная гипергеометрической функции Куммера $F_1(a; b; z)$.
- **Hypergeometric2F1[a, b, c, z]** – гипергеометрическая функция $F_2(a, b; c; z)$.

Следующие примеры показывают вычисления гипергеометрических функций:

Ввод (In)	Вывод (Out)
Hypergeometric0F1[2., 1.]	1.59064
Hypergeometric0F1[2., 2.+3.*I]	1.22457+2.51372 I
Hypergeometric1F1[1., 2., 2.+3.*I]	-1.03861+2.07929 I
Hypergeometric2F1[1., 2., 3., 2.+3.*I]	0.0291956+0.513051 I

Следует отметить, что число этих функций в ядре новых версий даже несколько сокращено по сравнению с предшествующими версиями. Убраны довольно редко используемые функции, в имени которых имеется слово Regularized.

3.11.8. Эллиптические интегралы и интегральные функции

В ядро системы Mathematica входят эллиптические функции и функции вычисления эллиптических интегралов:

- **EllipticE[m]** – полный эллиптический интеграл $E(m)$.
- **EllipticE[phi, m]** – эллиптический интеграл второго рода $E(\phi|m)$.
- **EllipticExp[u, {a, b}]** – обобщенный экспоненциал, связанный с эллиптической кривой $y^2 = x^3 + a x^2 + b x$.
- **EllipticExpPrime[u, {a, b}]** – производная по первому аргументу **EllipticExp[u, {a, b}]**.
- **EllipticF[phi, m]** – эллиптический интеграл первого рода $F(\phi|m)$.
- **EllipticK[m]** – полный эллиптический интеграл первого рода $K(m)$.
- **EllipticLog[{x, y}, {a, b}]** – обобщенный логарифм, связанный с эллиптической кривой $y^2 = x^3 + a x^2 + b x$.
- **EllipticNomeQ[m]** – возвращает значение $q = \text{Exp}[-\text{PiEllipticK}[1-m]/\text{EllipticK}[m]]$.
- **EllipticPi[n, phi, m]** – эллиптический интеграл третьего рода $\text{Pi}(n; \phi|m)$.
- **EllipticPi[n, m]** – полный эллиптический интеграл $\text{Pi}(n|m)$.
- **EllipticTheta[i, z, q]** – эллиптическая дзета-функции: $\theta_i(z, q)$, где $i = 1, 2, 3$, или 4 .
- **EllipticThetaC[u, m]** – эллиптическая дзета-функции Невилла $\theta_c(u, m)$.

- **EllipticThetaD[u, m]** – эллиптическая дзета-функции Невилла $\theta_d(u, m)$.
- **EllipticThetaN[u, m]** – эллиптическая дзета-функции Невилла $\theta_n(u, m)$.
- **EllipticThetaPrime[i, z, q]** – производная по второму аргументу эллиптической дзета-функции $\theta_i(z, q)$, где $i=1,2,3$, или 4.
- **EllipticThetaS[u, m]** – эллиптическая дзета-функции Невилла $\theta_s(u, m)$.
- **FresnelC[x]** – интеграл Френеля $C(x)$.
- **FresnelS[x]** – интеграл Френеля $S(x)$.
- **InverseJacobi**[v, m]** – обратная эллиптическая функция Якоби с обобщенным названием **. Возможны следующие наименования для **: **CD**, **CN**, **CS**, **DC**, **DN**, **DS**, **NC**, **ND**, **NS**, **SC**, **SD** и **SN**.
- **JacobiAmplitude[u, m]** – амплитуда для эллиптических функций Якоби.
- **Jacobian** – опция для FindRoot; может применяться для указания якобиана системы функций, для которых ищется корень.
- **Jacobi**[u, m]** – эллиптическая функция Якоби с обобщенным именем **, которое может принимать значения **CD**, **CN**, **CS**, **DC**, **DN**, **DS**, **NC**, **ND**, **NS**, **SC**, **SD** и **SN**.
- **JacobiSymbol[n, m]** – символ Якоби от n и m .
- **JacobiZeta[phi, m]** – дзета-функции Якоби $Z(\phi|m)$.
- **WeierstrassP[u, g2, g3]** – эллиптическая функция Вейерштрасса P .
- **WeierstrassPPrime[u, g2, g3]** – производная эллиптической функции Вейерштрасса P' по переменной u .

Приведем примеры на использование некоторых из этих функций:

Ввод (In)	Выход (Out)
EllipticE[0.1]	1.53076
EllipticE[Pi,0.1]	3.06152
EllipticF[Pi/2,0.1]	1.61244
EllipticPi[Pi,0.1]	-0.0266412 - 1.09088 I
EllipticK[0.1]	1.61244
FresnelC[1.0]	0.779893
FresnelS[1.0]	0.438259
JacobiCD[1,0.2]	0.605887
JacobiZeta[Pi,0.5]	0
WeierstrassPPrime[1.,2.,3.]	-1.31741

Эллиптические функции (интегралы) широко используются в практике выполнения оптических расчетов и в астрофизике.

3.11.9. Функции Эйри

Функции Эйри представляют независимые решения линейного дифференциального уравнения $w'' - zw = 0$. В Mathematica эти функции представлены следующим набором:

- **AiryAi[z]** – возвращает значение функции Эйри $Ai(z)$.
- **AiryAiPrime[z]** – возвращает значение производной функции Эйри $Ai'(z)$.

- **AiryBi[z]** – возвращает значение функции Эйри $Bi(z)$.
- **AiryBiPrime[z]** – возвращает производную функции Эйри $Bi'(z)$.

Ниже представлены примеры на вычисление функций Эйри:

Ввод (In)	Вывод (Out)
AiryAi[2.+3.*I]	0.00810446 + 0.131178 I
AiryAi[1.]	0.135292
AiryBi[2.+3.*I]	-0.396368 - 0.569731 I
AiryBiPrime[2.+3.*I]	0.349458 - 1.10533 I

3.11.10. Бета-функция и относящиеся к ней функции

Класс бета-функций, имеющих специальное интегральное представление (см. [84]), в Mathematica представлен следующим набором:

- **Beta[a, b]** – Эйлерова бета-функция $B(a, b)$.
- **Beta[z, a, b]** – неполная бета-функция.
- **Beta[z0, z1, a, b]** – обобщенная неполная бета-функция $Beta[z1, a, b] - Beta[z0, a, b]$.
- **BetaRegularized[z,a,b]** – регуляризованная неполная бета-функция $I(z,a,b) = Beta[z, a, b]/Beta[a, b]$.
- **BetaRegularized[z0, z1, a,b]** – регуляризованная обобщенная неполная бета-функция $I(z1,a,b) - I(z0, a, b)$.

Примеры на вычисление этих функций представлены ниже:

Ввод (In)	Вывод (Out)
Beta[1., 2.]	0.5
Beta[1., 2., 3.]	0.0833333
Beta[2.+3.*I, 4.+6.*I, 1, 2]	4. - 12. I
BetaRegularized[0.1, 1, 2]	0.19

3.11.11. Специальные числа и полиномы

Для вычисления специальных чисел и полиномов служит следующая группа функций:

- **BernoulliB[n]** – n-ое число Бернулли.
- **BernoulliB[n, x]** – полином Бернулли n-й степени.
- **Binomial[n, m]** – биномиальный коэффициент.
- **Cyclotomic[n, x]** – циклотомический полином порядка n по x.
- **EulerE[n]** – n-ое число Эйлера.
- **EulerE[n, x]** – n-й полином Эйлера.
- **EulerPhi[n]** – Эйлерова функция сумм $\phi(n)$ – количество положительных целых чисел, не превосходящих n и взаимно простых с n.

- **Fibonacci[n]** – n-ое число Фибоначчи.
- **Fibonacci[n, x]** – полином Фибоначчи $F_n(x)$.
- **Multinomial[n1, n2, ...]** – мультиномиальный коэффициент $(n1+n2+...)!/(n1! n2! ...)$.
- **NBernoulliB[n]** – численное значение n-го числа Бернулли.
- **NBernoulliB[n, d]** – n-ое число Бернулли до d-цифровой точности представления.
- **Pochhammer[a, n]** – символ Похгамера.
- **StirlingS1[n, m]** – число Стирлинга первого рода.
- **StirlingS2[n, m]** – число Стирлинга второго рода.

Ниже представлены примеры на вычисление данных функций:

Ввод (In)	Вывод (Out)
N[BernoulliB[2]]	0.166667
BernoulliB[2, 0.1]	0.0766667
Binomial[6, 4]	15
Cyclotomic[5, x]	$1 + x + x^2 + x^3 + x^4$
Cyclotomic[5, 0.2]	1.2496
EulerE[2]	-1
EulerE[2, 0.1]	-0.09
EulerPhi[2]	1
Fibonacci[10]	55
Fibonacci[6, x]	$3x + 4x^3 + x^5$
Pochhammer[1, 3]	6
StirlingS1[8, 4]	6769

3.11.12. Другие специальные функции

Помимо описанных выше функций в ядро системы входит также ряд других, менее распространенных, функций:

- **ArithmeticGeometricMean[a, b]** – арифметико-геометрическое среднее значение аргументов a и b.
- **IncludeSingularTerm** – опция для LerchPhi и Zeta, определяющая, следует ли включать члены вида $(k + a)^{-s}$ при $k + a == 0$.
- **InverseErf[s]** – инверсная функция ошибок.
- **InverseErfc[s]** – инверсная дополнительная функция ошибок.
- **InverseGammaRegularized[a, s]** – инверсная регуляризованная неполная гамма-функция.
- **InverseBetaRegularized[s, a, b]** – инверсная регуляризованная неполная бета-функция.
- **InverseSeries[s]** – берет ряд s, порожаемый директивой **Series**, и возвращает ряд для функции, обратной по отношению к функции, представленной рядом s.
- **InverseSeries[s, y]** – обратный ряд по переменной y.

- **InverseWeierstrassP**[{P, P'}, g2, g3] – возвращает величину u такую, что $P = \text{WeierstrassP}[u, g2, g3]$ и $P' = \text{WeierstrassPPrime}[u, g2, g3]$. Следует заметить, что P и P' не являются независимыми.
- **JordanForm**[A] – возвращает { S, J}, где $A = S \cdot J \cdot \text{Inverse}[S]$ и J является канонической формой Жордана A .
- **LerchPhi**[z, s, a] – трансцендентная функция Лерча $\text{Phi}(z, s, a)$.
- **MathieuC**[a, q, z] и **MathieuS**[a, q, z] – функции Матье.
- **MathieuCPrime**[a, q, z] и **MathieuSPrime**[a, q, z] – производны от функций Матье.
- **MathieuCharacteristic****[r, q] – характеристическая функция Матье (** имеет значения A, B и Exponent).
- **MeijerG**[{{a₁,...,a_n},{a_{n+1},...,a_p}},{{b₁,...,b_m},{b_{m+1},...,b_q}},z] – Мейджера G-функция.
- **MoebiusMu**[n] – значение функции Мебиуса $\mu(n)$.
- **PolyLog**[n, z] – n -ая полилогарифмическая функции от z .
- **RiemannSiegelTheta**[t] – аналитическая функция $\theta(t)$, удовлетворяющей уравнению $\text{RiemannSiegelZ}[t] = \text{Exp}[i \cdot \text{RiemannSiegelTheta}[t]] \cdot \text{Zeta}[1/2 + i t]$. Аргумент t не обязательно должен быть вещественным, но если является таковым, тогда $\text{RiemannSiegelTheta}[t] = \text{Im}[\text{LogGamma}[1/4 + i t/2]] - t \cdot \text{Log}[\text{Pi}]/2$.
- **RiemannSiegelZ**[t] – возвращает значение $\text{Exp}[i \cdot \text{RiemannSiegelTheta}[t]] \cdot \text{Zeta}[1/2 + i t]$.
- **SphericalHarmonicY**[l, m, theta, phi] – сферическая гармоника $Y_{lm}(\theta, \phi)$.
- **Zeta**[s] – дзета-функции Римана $\zeta(s)$.
- **Zeta**[s, a] – возвращает значение обобщенной дзета-функции Римана.

Ниже даны примеры на использование ряда из этих функций:

Ввод (In)	Вывод (Out)
LerchPhi [2.+3.*I,1,2]	0.0145978 + 0.256525 I
InverseErf [0.1]	0.088856
InverseErfc [0.1]	1.16309
InverseGammaRegularized [1, 0.5]	0.693147
InverseBetaRegularized [0.5, 1, 2]	0.292893
MathieuC [1,2,0.1]	0.196600+0.879889 I
MathieuS [1,2,0.1]	0.133005 - 0.0297195 I
MathieuCharacteristicA [1.5,2.]	2.85238
MeijerG [{{1,1},{}},{{1},{0}},x]	Log[1+x]
MoebiusMu [3]	-1
NBernoulliB [2]	0.166667
NBernoulliB [1,5]	-0.5
PolyLog [2,2.+3.*I]	-0.280988 + 3.01725 I
RiemannSiegelTheta [1.]	-1.76755
RiemannSiegelZ [1.]	-0.736305
SphericalHarmonicY [0.1,0.5,Pi/3,Pi/2]	0.195671 + 0.195671 I
Zeta [0.1]	-0.603038
Zeta [0.1,0.5]	-0.0432821

В Mathematica 4 добавлены новые встроенные функции **StruveH[n,z]** и **StruveL[n,z]** вычисляющие функции Струве порядка n для комплексного аргумента z .

По числу встроенных специальных математических функций системы Mathematica заметно превосходят другие системы компьютерной математики. При этом все эти функции могут участвовать в символьных преобразованиях. Это делает математические системы Mathematica предпочтительными при решении задач, в которых часто встречаются специальные математические функции. В тоже время необходимо отметить, что многие специальные функции системами Mathematica вычисляются только для целого порядка.

3.11.13. Новые специальные функции в Mathematica 6

И без того обширный набор специальных математических функций в Mathematica 6 пополнился рядом новых функций:

HankelH1[n,z]	KelvinKer[z]	KelvinKer[n,z]
SphericalBesselJ[n,z]	ParabolicCylinderD[r,z]	WhittakerM[k,m,z]
SpheroidalS1[n,m, r ,z]	BesselJZero[n,k]	BesselJZero[n,k,x0]
AiryAiZero[k]	AiryAiZero[k,x0]	ZernikeR[n,m,r]
ZetaZero[k]	ZetaZero[k,t]	

В подробном описании они не нуждаются в силу трех причин:

- название функции очевидно из ее имени;
- эти функции используются крайне редко;
- в справке даны многочисленные простые примеры их применения.

Для задания функций, обрезанных сверху и снизу, служит функция **Clip**. Ограничимся примером ее применения на рис. 3.3, на котором эта функция обеспечивает несимметричное ограничение синусоиды на уровнях $-0,5$ снизу и $0,8$ сверху.

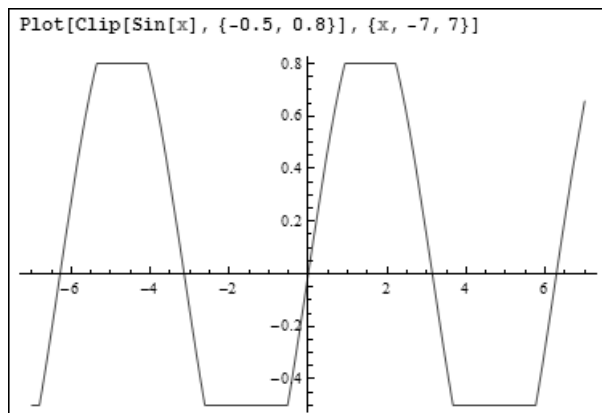


Рис. 3.3. Пример построения ограниченной снизу и сверху синусоиды

3.12. Расширенные возможности работы с объектами

3.12.1. Оперативная помощь

Оперативную помощь в ходе работы с системой о назначении какой-либо функции можно получить, используя следующие обращения:

- **? Name** или **Names["Name"]** – помощь по заданному слову Name.
- **?? Name** – расширенная помощь по заданному слову Name.
- **?Abc*** – перечень всех слов, начинающихся с Abc.
- **Options[name]** – получение информации об опциях объекта Name.

Нетрудно заметить, что есть два уровня комментариев оперативной помощи. Так, при обращении **?Sin** будет получено лишь сообщение о назначении функции. Обращение **??Sin** даст дополнительную информацию о признаках функции, именуемых ее атрибутами.

Многие встроенные функции защищены от модификации атрибутом Protected. К примеру, нельзя попытаться определить новую функцию $\text{Sin}[x]=x^2$, кстати, не потому, что это определение абсурдно (далее мы покажем, что можно снять защиту и переопределить любую функцию даже самым абсурдным образом), а потому, что имя функции защищено указанием ее атрибута Protected (Защищенное). Позднее мы ознакомимся и с иными атрибутами – под ними подразумеваются определенные свойства объектов.

3.12.2. Средства диагностики и сообщения об ошибках

Средства диагностики органично входят во все программные модули системы Mathematica, созданные профессионально. Благодаря этому система отслеживает неточные действия пользователя, например, синтаксические ошибки при вводе идентификаторов функций и команд, неправильное использование типов данных, применение недопустимых операций (вроде злосчастного деления на ноль) и т.д. и т.п. Всякий раз, когда ошибочное действие обнаружено, система выдает сообщение об ошибке в следующем виде:

Тип::Метка%Диагностика:Сообщение

Эти сообщения появляются в отдельных неактивных ячейках. **Тип** указывает на тип ошибки, например General – ошибка общего вида, Syntax – синтаксическая ошибка, Arg – ошибка задания аргумента и т.д. **Метка** указывает место ошибки в списке ошибок данного типа, а **Диагностика** указывает (увы, не всегда) на ошибочное выражение. **Сообщение** обычно раскрывает суть ошибки.

Допустим, мы пытаемся вычислить значение экспоненциальной функции, указав ошибочно аргумент 2,2 с разделительной запятой, а не точкой. Вот как Mathematica отреагирует на такой «пустяк»:

Exp[2,2]

```
Exp::argx : Exp called with 2
arguments; 1 argument is expected. More...
Exp[2,2]
```

Итак, ясно, что произошла ошибка задания аргумента: функция **Exp** должна иметь только один аргумент, а число 2,2 система воспринимает как два аргумента, разделенных запятой. Вот еще один пример: ошибочно заданы имя функции (с малой буквы) и круглые скобки:

N(exp(2))

```
General::spell1 :
Possible spelling error: new symbol name "exp" is
similar to existing symbol "Exp". More...
2 exp N
```

А вот следующий пример более коварный:

N(Exp[2])

$e^2 N$

Тут Mathematica не поняла пользователя: правильное выражение (**Exp[2]**) она представила в иной форме, а вот символ **N** восприняла как переменную, а не как встроенную функцию. В итоге никакого сообщения об ошибке не выдано. Причина казуса здесь в пробеле между **N** и открывающей круглой скобкой. Уберите пробел и введите **Exp[2]** в квадратных скобках, и все будет вычислено «по понятиям»:

N[Exp[2]]

7.38906

Здесь важно отметить, что обычно сообщения об ошибках системы Mathematica дают не только указания о самом по себе факте наличия ошибки, но и сообщают о том, что необходимо сделать для предотвращения ошибок. Позже из сведений о подготовке пакетов расширения мы увидим, что сообщения об ошибках и иные сообщения (в том числе информационные) заданы в их структуре. Насколько эти сообщения точны и как они предугадывают возможные ошибки – это уже зависит от опыта программиста, готовящего программные модули.

3.12.3. Включение и выключение сообщений об ошибках

Опытный пользователь нередко способен опознать ошибки и без слишком назойливых сообщений о них. Например, он может судить о своей промашке просто по отказу системы выполнить вычисление и по повтору выражения в строке вывода (см. примеры выше). Кроме того, часть сообщений носит предупреждающий характер и на первых порах может игнорироваться.

Для отключения сообщений об ошибках служит ключ:

Off[Function::tag]

Например, отключим сообщение об ошибках у функции **Exp**:

```
Off[Exp::argx]
```

```
Exp[2,2]
```

```
Exp[2,2]
```

```
Exp[2]
```

```
E2
```

```
N[Exp[2]]
```

```
7.38906
```

Для включения сообщения об ошибках используется ключ:

On[Function::tag]

Например, для возобновления выдачи ошибок у функции **Exp** следует выполнить команду

```
On[Exp::argx]
```

К сожалению, диагностика ошибок не способна опознать ошибки, имеющие верный синтаксис. Чаще всего это ошибки неверного описания алгоритма вычислений. Например, если пользователь в математическом выражении вместо **Sin[x]** записал **Cos[x]**, то эта грубая ошибка никак не будет распознана системой, поскольку синтаксически выражение **Cos[x]** записано безупречно. Часто пользователи путают идентификаторы переменных. Естественно, что ответственность за такие ситуации целиком лежит на пользователе-программисте.

3.12.4. Защита от модификации и ее отмена

Объекты Mathematica имеют средства защиты от модификации и ее отмены. Для этого используются следующие функции-директивы:

- **Protect[s1, s2, ...]** – устанавливает атрибут защиты от модификации **Protected** для перечисленных символов **si**.
- **Protect["form1", "form2", ...]** – устанавливает атрибут защиты от модификации для всех символов, имена которых сопоставимы с любым из указанных строковых шаблонов **formi**.
- **Unprotect[s1, s2, ...]** – удаляет атрибут защиты от модификации **Protected** для символов **si**, что делает возможной их модификацию.
- **Unprotect["form1", "form2", ...]** – снимает защиту всех символов, имена которых текстуально (по буквам) сопоставимы с любым из указанных **formi**.

Приведем наглядный пример на модификацию встроенной функции логарифма:

```
Log[7]=2
```

```
Set::write : Tag Log in Log[7] is Protected. More...
```

```
2
```

Итак, здесь предпринята попытка приписать логарифму числа 7 вовсе несвойственное ему значение 2. В ответ система выдала сообщение, что символ **Log** записан с атрибутом **Protected**, т.е. защищен от модификации. С помощью директивы **Unprotect** снимем защиту:

Unprotect[Log]

{Log}

Теперь выражение **Log[7]** можно модифицировать

Log[7] = 2

2

и использовать его уже в новом значении:

Log[7]+Log[3]

2+Log[3]

Для удаления модификации и защиты символа Log от модификации используйте следующие действия:

Log[7]=.

Protect[Log]

{Log}

Теперь можно проверить, что присвоение **Log[7]=2** не действует и функция Log работает, как положено, возвращая значение $\ln(7)=1.9451$:

Log[7]

Log[7]

N[Log[7]]

1.94591

Защита идентификаторов объектов от модификации является мощным средством контроля правильности вычислений. Вряд ли стоит устранять ее подавляющему большинству пользователей. Тем не менее, возможность устранения защиты позволяет переименовать объект, например, при использовании с ним новых алгоритмов вычислений или при задании системе Mathematica каких-то новых свойств, не свойственных обычным.

Функции для работы со сложными типами данных

4.1. Создание списков и выделение элементов списков	204
4.2. Выявление структуры списков	209
4.3. Работа со списком в стеке	211
4.4. Манипуляции с элементами списков	212
4.5. Базовые средства линейной алгебры	217
4.6. Новые средства работы со списками в Mathematica 6 ...	223
4.7. Работа со строками	227

4.1. Создание списков и выделение элементов списков

4.1.1. Создание списков

Наиболее общим видом сложных данных в системе являются *списки* (lists). Их также относят к данным *множественного типа* [13, 26, 28]. Списки представляют совокупность однотипных или разнотипных данных, сгруппированных с помощью фигурных скобок. Например:

<code>{1, 2, 3}</code>	– список из трех целых чисел;
<code>{a, b, c}</code>	– список из трех символьных данных;
<code>{1, a, x^2}</code>	– список из разнотипных данных;
<code>{{a,b},{c,d}}</code>	– список, эквивалентный матрице $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$;
<code>{x^2+y^2, 2*Sin[x]}</code>	– список из двух математических выражений.

С помощью списков можно задавать более привычные типы сложных данных, например, векторы – одномерные массивы данных, матрицы – двумерные массивы и многомерные массивы [30]. При этом каждая группа элементов многомерных списков выделяется своей парой фигурных скобок, а в целом список выделяется общими скобками. Уже в системах Mathematica 4/5 плотность упаковки больших списков (массивов) существенно повышена, как и скорость операций с ними.

Возможно задание списков в списке, например:

```
{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
```

Такой список представляет матрицу

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Однако, чтобы вывести список в такой матричной форме, необходимо использовать после списка выражение `//MatrixForm` (см. примеры на рис. 4.1).

На рис. 4.1 показан еще один способ задания списка или матрицы с помощью функции `List`:

- `List[a, b, c, ...]` – создает список {a, b, c,...};
- `List[{a,b,c,...},{d,e,f,...},{i,k,l,...}]` – создает список – матрицу {{a,b,c,...}, {d,e,f,...},{i,k,l,...}}.

Списки можно составлять напрямую, задавая объекты в соответствии с описанным синтаксисом. Однако можно и генерировать некоторые виды списков, таких как таблицы. Списки могут быть объектами присваивания переменным, например

```
v:={1 ,2, 3, 4, 5}
```

```

In[1]:= matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
Out[1]= {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}

In[2]:= matrix // MatrixForm
Out[2]/MatrixForm=

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$


In[3]:= List[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}]
Out[3]= {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}

In[4]:= % // MatrixForm
Out[4]/MatrixForm=

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$


```

Рис. 4.1. Примеры задания и вывода матрицы

Списки характеризуются *размером*, который представляет собой произведение числа элементов списков по каждому направлению. Число направлений называют *размерностью*. Например, одномерный список является вектором и характеризуется числом элементов по единственному направлению. При этом вектор может быть вектором-строкой и вектором-столбцом. Двумерный список представляет матрицу, имеющую m строк и n столбцов. Ее размер $m \times n$. Если $m=n$, матрица называется квадратной. Имена векторов и матриц в данной книге обозначены обычно жирными символами, например, **V** для вектора и **M** для матрицы.

4.1.2. Генерация списков

Для генерации списков с элементами – вещественными и целыми числами или даже целыми выражениями – особенно часто используется функция **Table**, создающая таблицу-список:

- **Table[expr, {imax}]** – генерирует список, содержащий imax экземпляров выражения expr .
- **Table[expr, {i, imax}]** – генерирует список значений expr при i , изменяющемся от 1 до imax .
- **Table[expr, {i, imin, imax}]** – начинается со значения $i = \text{imin}$.
- **Table[expr, {i, imin, imax, di}]** – использует шаги di .
- **Table[expr, {i, imin, imax}, {j, jmin, jmax}, ...]** – возвращает вложенный список. Самым внешним является список по переменной i .

Ниже представлены примеры на использование функции **Table** (первая строка каждого примера – ввод, следующая – вывод).

Пример	Комментарий
Table[Exp[i], {5}] <div style="margin-left: 40px;">i i i i i</div> {E, E, E, E, E}	Генерация пяти значений E ⁱ
Table[Exp[i], {i, 1, 5}] <div style="margin-left: 40px;">2 3 4 5</div> {E, E, E, E, E}	Генерация пяти значений E ⁱ (i=1,2,3,4 и 5)
Table[N[Exp[i]], {i, 0, 2, 0.5}] {1., 1.64872, 2.71828, 4.48169, 7.38906}	Генерация пяти значений E ⁱ в численном виде
Table[i*j, {i, 1, 3}, {j, 1, 3}] {{1, 2, 3}, {2, 4, 6}, {3, 6, 9}}	Генерация матрицы размером 3 r 3

Применяется также функция **Range** для создания так называемых ранжированных числовых элементов, значения которых лежат в некотором диапазоне числовых значений:

- **Range[imax]** – генерирует список числовых элементов {1, 2, ..., imax}.
- **Range[imin,imax]** – генерирует список числовых элементов {imin,...,imax}.
- **Range[imin,imax, di]** – генерирует список числовых элементов с шагом di.

Примеры на использование функции **Range**.

Пример	Комментарий
Range[5] {1, 2, 3, 4, 5}	Генерация пяти целых чисел
Range[0,2,0.5] {0, 0.5, 1., 1.5, 2.}	Генерация пяти вещественных чисел

4.1.3. Выделение элементов списков

Для выделения элементов списка list используются двойные квадратные скобки:

list[[i]] – выделяет i-ый элемент списка с его начала (если i<0 – с конца).

list[[{i,j,...}]] – выделяет i-ый, j-ый и т.д. элементы списка.

Ниже приведены примеры на выделение элементов списков.

Пример	Комментарий
l1:={1,2,3,4,5}	Задание исходного списка l1
l1[[3]] 3	Выделение третьего элемента с начала
l1[[-2]] 4	Выделение второго элемента с конца
l1[[{1,2,5}]] {1, 2, 5}	Выделение первого, второго и пятого элементов
l2:={1,2,3},{4,5,6} {1, 2, 3}, {4, 5, 6}	Задание сдвоенного (двумерного) списка

Пример	Комментарий
TableForm[12] 1 2 3 4 5 6	Вывод сдвоенного списка в табличной форме
12[[2,3]] 6	Выделение элемента сдвоенного списка

Для выделения заданного i -го элемента списка `list` используется также функция **Part[list,i]**. При $i > 0$ отсчет номеров элементов идет с начала списка, а при $i < 0$ – с его конца. Это правило поясняют примеры, показанные ниже:

```
L:={1,2,3,a,b,c}
{Part[L,2],Part[L,5],Part[L,6]}
{2,b,c}
{Part[L,-2],Part[L,-5],Part[L,2]}
{b,2,2}
```

Функция **Part** может использоваться для выбора заданного элемента выражения из списка. В этом случае вместо i надо указать три числа: номер выражения как элемента списка, уровень выражения и порядковый номер извлекаемого из выражения объекта. Показанные на рис. 4.2 примеры иллюстрируют работу со списком, в качестве последнего (четвертого) элемента которого является математическое выражение.

```
Part[{a, b, c, Tan[x + y]}, 2]
b

Part[{a, b, c, Tan[x + y]}, 4]
Tan[x + y]

Part[{a, b, c, Tan[x + y]}, 4, 1, 1]
x

Part[{a, b, c, Tan[x + y]}, 4, 1, 2]
y

Part[{a, b, c, Tan[x + y]}, 4, 2, 2]
Part::partw : Part 2 of Tan[x + y] does not exist.
{a, b, c, Tan[x + y]}[[4, 2, 2]]
```

Рис. 4.2. Примеры выделения элементов выражения

Обратите внимание на то, что в последнем примере неверно задан уровень выражения: использованное выражение имеет только один (первый) уровень. Поэтому задание второго уровня ведет к появлению сообщения об ошибке.

Функция **Select** используется для выделения элементов списка, удовлетворяющих заданному критерию:

- **Select**[list, crit] – выбирает все элементы e_i списка list, для которых crit[e_i] имеет значение True.
- **Select**[list, crit, n] – выбирает из первых n элементов, для которых crit[e_i] есть True.

Ниже представлены примеры на применение этой функции:

```
Select[{1,a,2,b,3,c},NumberQ]
{1,2,3}
Select[{1,a,2,b,3,c},NumberQ,2]
{1,2}
Select[{1,a,2,b,3,c},PrimeQ]
{2,3}
```

4.1.4. Вывод элементов списков

Для вывода элементов списка используются следующие функции:

- **MatrixForm**[list] – выводит список в форме массива – матрицы.
- **TableForm**[list] – выполняет вывод с элементами списка list, расположенными в массиве прямоугольных элементов.

С ними используются следующие опции:

- **TableAlignments** – устанавливает, как должно выравниваться содержимое списка в каждой размерности (слева, по центру или справа).
- **TableDepth** – устанавливает максимальное количество уровней, выводимых в табличном или матричном форматах.
- **TableDirections** – указывает, следует ли последовательные (соседние) размерности располагать в виде строк или столбцов.
- **TableHeadings** – задает метки (labels) для каждой размерности таблицы или матрицы.
- **TableSpacing** – устанавливает количество пробелов, которые следует оставлять между соседними строками или столбцами.

Обратите внимание, что эти же опции используются как для функции **TableForm**, так и для функции **MatrixForm**, используемой для вывода матриц. Вообще, векторы и матрицы являются разновидностью списков. В приведенных на рис. 4.3 примерах поясняется использование функций **MatrixForm** и **TableForm** на примере вывода списка.

Дополнительные возможности функции демонстрирует рис. 4.4. Здесь особенно полезно отметить возможность выравнивания данных в таблицах по левому и правому краям, а также посередине.

В большинстве случаев опции для функций **MatrixForm** и **TableForm** не используются. Точнее, они установлены по умолчанию. Проверить, какие опции использованы, можно, например, следующим образом:

```
Options[MatrixForm]
{TableAlignments->Automatic, TableDepth->Infinity, TableDirections->Column,
TableHeadings->None, TableSpacing->Automatic}
```

```

In[28]:= l = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
Out[28]= {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}

In[30]:= MatrixForm[l]
Out[30]/MatrixForm=

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$


In[31]:= TableForm[l]
Out[31]/TableForm=


|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |



In[33]:= TableForm[l, TableSpacing -> {3, 6}]
Out[33]/TableForm=


|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |


```

Рис. 4.3. Примеры задания и вывода списка в табличной и матричной формах

Options[TableForm]

{TableAlignments→Automatic, TableDepth→∞, TableDirections→Column, TableHeadings→None, TableSpacing→Automatic}

Итак, Mathematica обладает обширными возможностями в части выделения элементов списков и представления списков на экране дисплея и в распечатках документов.

4.2. Выявление структуры списков

4.2.1. Функции выявления структуры списков

Списки относятся к данным сложной структуры. Поэтому при работе с ними возникает необходимость *контроля* над структурой списков, без чего их применение может привести к грубым ошибкам – как явным, сопровождаемым выдачей сообщения об ошибке, так и неявным.

Для выявления структуры списков используется ряд функций:

- **Count[list, pattern]** – возвращает количество элементов в списке **list**, которые соответствуют образцу **pattern**.
- **Dimensions[list]** – возвращает список размеров списка по каждой размерности.


```

TableForm[{1, 22222, 2222222}, TableAlignments -> Left]

1
22222
2222222

TableForm[{1, 22222, 2222222}, TableAlignments -> Center]

  1
 22222
2222222

TableForm[{1, 22222, 2222222}, TableAlignments -> Right]

      1
     22222
    2222222

TableForm[{{1, 2, {{a, b}, {c, d}}, 3, 4}}]

1      2      a  b      3      4
      c  d

TableForm[{{1, 2, {{a, b}, {c, d}}, 3, 4}}, TableDepth -> 1]

{1, 2, {{a, b}, {c, d}}, 3, 4}

TableForm[{{1, 2, {{a, b}, {c, d}}, 3, 4}}, TableDirections -> {Row, Column}]

1
2
None a  b
    c  d
3
4

```

Рис. 4.4. Примеры вывода списка в табличной форме

- **FreeQ[list,form]** – проверяет, свободен ли список от **form**: если да – возвращает True, иначе возвращает False.
- **Length[list]** – возвращает число элементов одномерного списка list или число размерностей в случае многомерного списка.
- **MatrixQ[list]** – проверяет, является ли список матрицей, и дает True, если это так, и False в противном случае.
- **MemberQ[list,form]** – проверяет, есть ли **form** в списке, и возвращает True, если это так и False в противном случае.
- **Position[list,form]** – возвращает номер позиции **form** в списке.
- **TensorRank[list]** – находит ранг списка, если он тензор.
- **VectorQ[list]** – проверяет, является ли список вектором, и дает True, если это так, и False в противном случае.

Функции с буквой Q в конце имени являются тестирующими и возвращают логические значения True или False. Остальные функции возвращают численные значения соответствующего параметра списка.

4.2.2. Примеры выявления структуры списков

Ниже даны примеры на использование этих функций:

Ввод (In)	Вывод (Out)
<code>l1={1,2,3,4,1};</code>	
<code>Length[l1]</code>	5
<code>Dimensions[l1]</code>	{5}
<code>MatrixQ[l1]</code>	False
<code>TensorRank[l1]</code>	1
<code>MemberQ[l1,1]</code>	True
<code>Count[l1,1]</code>	2
<code>FreeQ[l1,5]</code>	True
<code>Position[l1,1]</code>	{{1}, {5}}
<code>VectorQ[l1]</code>	True
<code>M={{1,2,3},{4,5,6}}</code>	
<code>Lenght[M]</code>	2
<code>Dimensions[M]</code>	{2,3}

Система оставляет за пользователем свободу действий по результатам анализа структуры списков.

4.3. Работа со списком в стеке

4.3.1. Понятие о стеке

Списки можно представить в виде особой структуры данных – *стека*. Это особая структура данных, чисто умозрительно напоминающая стопку тарелок в шкафу. При этом роль данных играют тарелки. Очередную тарелку можно положить только сверху – на вершину стека. На дне стека лежит первая, введенная в него тарелка. Стек подчиняется правилу: последнее введенное данное извлекается первым, а первое введенное данное извлекается последним. Стек относится к системам хранения данных динамического типа – его размеры непрерывно меняются по ходу вычислений. Стек может быть пустым, если из него были извлечены все данные.

Стек в явной или неявной форме имеется при реализации многих языков программирования. Особенно велико его значение в языке программирования Форт (Forth) и в языках общения и программирования микрокалькуляторов известной фирмы Hewlett Packard [6], а также большинства отечественных программируемых микрокалькуляторов [2, 3]. Причина этого заключается в том, что стек упро-

щает выполнение математических операций: он позволяет избавиться от скобок и использовать обратную бесскобочную (польскую) запись математических выражений.

4.3.2. Работа со стеком

Система Mathematica дает обширные возможности для операций со стеком:

- **Drop[list, n]** – возвращает список list, из которого удалены первые n элементов.
- **Drop[list, -n]** – возвращает список list с отброшенными последними n элементами.
- **Drop[list, {n}]** – возвращает список list без n-го элемента.
- **Drop[list, {m, n}]** – возвращает список list, отбросив элементы от m до n.
- **Fold[f,x,list]** – возвращает последний элемент из FoldList[f,x,list].
- **Last[list]** – возвращает последний элемент списка list.
- **Rest[list]** – возвращает список с уничтоженным первым элементом.
- **Take[list, n]** – возвращает первые n элементов списка list.
- **Take[list, -n]** – возвращает последние n элементов списка list.
- **Take[list, {m, n}]** – возвращает элементы списка с порядковыми номерами от m до n.

Следующие примеры поясняют работу со стеком:

Ввод (In)	Вывод (Out)
Drop[{1, 2, 3, 4, 5}, 2]	{3, 4, 5}
Drop[{1, 2, 3, 4, 5}, -2]	{1, 2, 3}
Drop[{a, b, c, d, e}, {2, 4}]	{a, e}
Last[{1, 2, 3, 4, 5}]	5
Rest[{1, 2, 3, 4, 5}]	{2, 3, 4, 5}
Take[{1, 2, 3, 4, 5}, 2]	{1, 2}
Take[{1, 2, a, b, c}, -2]	{b, c}
Take[{1, 2, 3, 4, 5}, {2, 4}]	{2, 3, 4}

В обычной практике вычислений явная работа со стеком обычно не используется. Функции для работы со стеком могут использоваться для создания алгоритмов вычисления сложных выражений и моделирования алгоритмов, присущих языкам, использующим стек.

4.4. Манипуляции с элементами списков

4.4.1. Включение в список новых элементов

Тривиальная процедура общения со стеком (типа ввести данные, вывести их) ограничивает возможности стековых операций. Из бытового опыта мы знаем, что, проявив настойчивость, можно вставить тарелку и в середину стопки тарелок.

Аналогично этому Mathematica дает ряд расширенных возможностей для работы со списками, выходящих за рамки обычных стековых операций.

Так, для расширения списка путем включения в него новых элементов, используются следующие функции:

- **Append[list,element]** – добавляет элемент в конец списка.
- **Prepend[list,element]** – добавляет элемент в начало списка.

Insert[list,element,n] – вставляет элемент в позицию n (отсчет позиции ведется с начала листа, а если задано -n, то с конца). Можно данной функцией включать элемент в несколько позиций, указав каждую в фигурных скобках и оформив это указание также в фигурных скобках: вместо n. При таком включении необходимо учитывать позицию данного включаемого элемента с учетом расширения списка включением в него предшествующих элементов.

Следующие примеры иллюстрируют применение этих функций:

Ввод (In)	Вывод (Out)
l={1,2,3}	{1, 2, 3}
Append[l,e]	{1, 2, 3, e}
Prepend[l,e]	{e, 1, 2, 3}
Insert[l,e,2]	{1, e, 2, 3}
L={1, 2, 3, 4, 5}	{1, 2, 3, 4, 5}
Insert[L, e, -5]	{1, e, 2, 3, 4, 5}
Insert[L, e, {{1},{5}}]	{1, e, 2, 3, 4, e, 5}

Обратите внимание, что в данном случае элементы списка – числа, тогда как вставляемый элемент имеет символьное значение e.

4.4.2. Удаление элементов из списка

Обратите внимание, что описанные для стека функции **Drop** и **Rest** позволяют удалить из списка последний или первый элемент. Функция **Delete[list,i]** позволяет удалить из списка произвольный i-ый элемент. Если i>0, отсчет удаленного элемента идет с начала списка, а если i<0, то с конца:

```
L:={1,2,3,4,5}
{Delete[L,1],Delete[L,3],Delete[L,5]}
{{2,3,4,5},{1,2,4,5},{1,2,3,4}}
{Delete[L,-2],Delete[L,-5]}
{{1,2,3,5},{2,3,4,5}}
Delete[{1,2,3,{a,b,c},4,5},2]
{1,3,{a,b,c},4,5}
```

Если элементом списка является список, то он фигурирует как один элемент. Можно, однако, удалять избранный элемент из элемента-списка, указав в фигурных скобках вместо k номер элемента-списка в общем списке и номер удаляемого элемента во внутреннем списке. Это иллюстрируют следующие примеры:

```
Delete[{1,2,3,{a,b,c},4,5},{4,3}]
```

```
{1, 2, 3, {a, b}, 4, 5}
Delete[{1, 2, 3, {a, b, c}, 4, 5}, {4, 1}]
{1, 2, 3, {b, c}, 4, 5}
```

Наконец, можно функцией **Delete** удалить несколько элементов списка, указав их каждый в фигурных скобках и оформив это указание также в фигурных скобках:

```
Delete[{1, 2, 3, {a, b, c}, 4, 5}, {{2}, {4}, {5}}]
{1, 3, 5}
```

Следует учитывать, что есть некоторые функции, которые удаляют в списках определенные элементы. Они будут рассмотрены ниже.

4.4.3. Изменение порядка элементов в списке

Помимо добавления в список новых данных имеется возможность изменения порядка расположения элементов в списке. Она реализуется следующими операциями:

- **Flatten[list]** – выравшивает (превращает в одномерный) список по всем его уровням.
- **Flatten[list, n]** – выравшивает список по его n-уровням.
- **Flatten[list, n, h]** – выравшивает с заголовком h по его n-уровням.
- **FlattenAt[list, n]** – выравшивает подсписок, если он оказывается n-ым элементом списка list. Если n отрицательно, позиция отсчитывается с конца.
- **Sort[list]** – сортирует элементы списка list в каноническом порядке.
- **Sort[list, p]** – сортирует согласно функции упорядочения p.
- **Reverse[list]** – возвращает список с обратным порядком расположения элементов.
- **RotateLeft[list]** – возвращает список после однократного поворота влево.
- **RotateLeft[list, n]** – возвращает список после n-кратного поворота влево.
- **RotateRight[list]** – возвращает список после однократного поворота вправо.
- **RotateRight[list, n]** – возвращает список после n-кратного поворота вправо.
- **Transpose[list]** – осуществляет транспозицию (смену строк и столбцов) для двумерного списка.
- **Transpose[list, n]** – осуществляет транспозицию n-мерного списка.

Таким образом, имеются обширные возможности изменения структуры списков. Ниже приведен ряд примеров на использование этих функций:

Ввод (<i>In</i>)	Вывод (<i>Out</i>)
<code>l3={1, 2, 3}, {4, 5, 6}, {7, 8, 9};</code>	
<code>Flatten[l3]</code>	{1, 2, 3, 4, 5, 6, 7, 8, 9}
<code>FlattenAt[l3, 1]</code>	{1, 2, 3, {4, 5, 6}, {7, 8, 9}}
<code>Sort[{1, 5, 3, 4, 2}]</code>	{1, 2, 3, 4, 5}
<code>Reverse[{1, 2, 3, 4}]</code>	{4, 3, 2, 1}

Ввод (In)	Вывод (Out)
RotateLeft [{1,2,3,4,5},2]	{3, 4, 5, 1, 2}
RotateRight [{1,2,3,4,5},2]	{4, 5, 1, 2, 3}
l2 ={a,b},{c,d};	
TableForm [l2]	a b c d
TableForm [Transpose [l2]]	a c b d

Изменение порядка расположения элементов в списке полезно при осуществлении некоторых алгоритмов. К примеру, сортировка списка ускоряет выполнение статистических расчетов и уменьшает их погрешности.

4.4.4. Комбинирование списков и работа с множествами

Иногда возникает необходимость комбинирования нескольких списков. Для этого используются следующие функции:

- **Complement**[list,list1,list2,...] – возвращает список list с элементами, которые не содержатся ни в одном из списков list1, list2,....
- **Intersection**[list1, list2, ...] – (пересечение множеств) возвращает упорядоченный список элементов, общих для всех списков listi.
- **Join**[list1, list2, ...] – объединяет списки в единую цепочку (выполняет конкатенацию). **Join** может применяться на любом множестве выражений, имеющих один заголовок.
- **Union**[list1,list2, ...] – удаляет повторяющиеся элементы списков и возвращает отсортированный список всех различающихся между собой элементов, принадлежащих любому из данных списков listi. Функция обеспечивает теоретико-множественное объединение списков.
- **Union**[list] – возвращает отсортированный вариант списка list, в котором опущены все повторяющиеся элементы.

Следующая пара примеров иллюстрирует применение функций комбинирования списков:

Ввод (In)	Вывод (Out)
Complement [{1,2,3,4,5},{1,a,2},{b,c,5}]	{3, 4}
l1 ={1,2,3,4,5}; l2 ={a,b,3,4,c};	
Intersection [l1,l2]	{3, 4}
Join [l1,l2]	{1,2,3,4,5,a,b,3,4,c}
Union [{1,2,4,3,2,7,3,5}]	{1, 2, 3, 4, 5, 7}
Union [{3,2},{1,4}]	{1, 2, 3, 4}
Union [{a,b,c,a},{1,d,3}]	{1, 3, a, b, c, d}

Комбинирование списков позволяет создавать сложные структуры данных из более простых структур. Это может быть полезно при построении очередей, деревьев и иных структурных построений. Кроме того, приведенные функции обеспечивают основные операции с множествами.

4.4.5. Другие функции для работы со списками

Для работы со списками используются также следующие, менее распространенные функции, представленные в Приложении (см. раздел «Некоторые функции для работы со списками»). В Mathematica 4/5 появились новые функции расширения списков нулями:

PadLeft[list]	PadLeft[list,n]	PadLeft[list,f,n]
PadRight[list]	PadRight[list,n]	PadRightLeft[list,n]

Примеры их применения:

PadLeft[{a,b,c},6]	возвращает список {0,0,0,a,b,c}
PadRight[{a,b,c},6]	возвращает список {a,b,c,0,0,0}

Некоторые другие функции для работы со списками представлены ниже.

- **Accumulate[f,g[e1, e2, ...]]** – возвращает $g[e1, f[e1, e2], f[f[e1, e2], e3], \dots]$.
- **Cases[{e1, e2, ...}, pattern]** – возвращает список тех e_i , которые соответствуют заданному шаблону (pattern).
- **Cases[{e1, ...}, pattern -> rhs]** или **Cases[{e1, ...}, pattern -> rhs]** – возвращает список значений rhs, соответствующих тем e_i , которые подходят под шаблон pattern.
- **CoefficientList[poly, var]** – возвращает список коэффициентов перед степенями переменной var в полиноме poly, начиная со степени 0.
- **CoefficientList[poly, {var1, var2, ...}]** – возвращает матрицу коэффициентов var_i .
- **\$CommandLine** – список строк, возвращающий элементы исходной командной строки операционной системы, которой была вызвана Mathematica.
- **Compose[a,b,c,d]** – возвращает $a[b[c[d]]]$.
- **ComposeList[{f1, f2, ...}, x]** – формирует список формы $\{x, f1[x], f2[f1[x]], \dots\}$.
- **ComposeSeries[s,t,u,...]** – формирует степенные ряды s,t,u, и т.д. Ряды (исключение для первого элемента) должны начинаться положительной степенью переменной.
- **Composition[f1, f2, f3, ...]** – представляет композицию функций $f1, f2, f3, \dots$.
- **FoldList[f, x, {a, b, ...}]** – возвращает $\{x, f[x, a], f[f[x, a], b], \dots\}$.
- **HeadCompose[a, b, c, d]** – возвращает $a[b][c][d]$.
- **Listable** – атрибут, который может назначаться символу f для указания того, что функция f автоматически будет связной для списков, которые выполняют роль ее аргументов.
- **MemberQ[list, form, levelspec]** – тестирует все части списка list, определяемые спецификацией уровня levelspec.

- **Partition[list,n]** – разбивает список list на неперекрывающиеся части с длиной n. Если количество элементов в списке не делится нацело на n, то последние k ($k < n$) элементов удаляются.
- **Partition[list,n,d]** – как предшествующая функция дает разбиение списка, но с отступом d. При $d < n$ подсписки перекрываются.
- **Permutations[list]** – генерирует список всех возможных перестановок элементов в списке list.
- **Position[expr,pattern]** возвращает список позиций в expr, в которых размещаются объекты, сопоставимые с указанным шаблоном pattern.
- **Position[expr,pattern,levspec]** – выполняет поиск только объектов, находящихся на уровнях, указываемых levspec.
- **RealDigits[x]** – возвращает список цифр в приближенном вещественном числе x вместе с количеством цифр слева от десятичной точки, присутствующих в научной записи числа.
- **RealDigits[x,b]** – возвращает список цифр числа x по основанию b.
- **Signature[list]** – дает сигнатуру перестановки, необходимой для размещения элементов списка list в каноническом порядке.
- **SingularValues[m]** – возвращает особое значение декомпозиции для числовой матрицы m. Результатом будет список {u, w, v}, где w – список ненулевых особых значений, а m может быть записана как **Conjugate[Transpose[u]].DiagonalMatrix w].v**.
- **SequenceLimit[list]** – возвращает по эpsilon-алгоритму Винна аппроксимацию предела последовательности, первые несколько членов которой заданы в виде списка list. Этот алгоритм может давать конечные значения для расходящихся последовательностей.
- **SubValues[f]** – возвращает список правил преобразования, относящихся ко всем по значениям (значениям для f[x,...][...] и т.д.), определенным для символа f.
- **\$SuppressInputFormHeads** – представляет собой список заголовков тех выражений, чьи InputForm не должны автоматически пересылаться в программный препроцессор.

В целом, можно сделать вывод, что обилие функций по работе со списками позволяет решать практически любые задачи, в основе работы с которыми лежат манипуляции со списками, стеками и другими родственными типами данных.

4.5. Базовые средства линейной алгебры

4.5.1. Задание массивов

Разновидностью множественных данных являются *массивы* (Arrays) с индексированными переменными. Массивы могут быть одномерными (один список), двумерными и многомерными (два и более списка). Одномерные массивы в матема-

тике называют векторами, двумерные – матрицами. В общем случае массив характеризуется размерностью (числом направлений) и размером – произведением числа элементов по каждой размерности. Mathematica позволяет создавать многомерные массивы – число элементов в них ограничено лишь объемом памяти компьютера.

Для задания массивов используются следующие функции:

- **Array[f, n]** – генерирует список длиной n с элементами f[i].
- **Array[f, {n1, n2, ...}]** – генерирует массив с размером n1n2r... в виде вложенных списков с элементами f[i1, i2, ...].
- **Array[f, dims, origin]** – генерирует список с размерностью dims, используя спецификацию индекса origin.
- **Array[f, dims, origin, h]** – использует заголовок h, а не List, для каждого уровня массива.

Примеры задания массивов и их вывода:

Ввод (In)	Вывод (Out)
Y:=Array[Exp, 4]	
Y	$\begin{matrix} & 2 & 3 & 4 \\ \{E, & E & , & E & , & E \} \end{matrix}$
N[Y]	{2.71828, 7.38906, 20.0855, 54.5982}
Array[f, {3, 3}]	$\begin{matrix} \{ \{ f[1, 1], f[1, 2], f[1, 3] \}, \\ \{ f[2, 1], f[2, 2], f[2, 3] \}, \\ \{ f[3, 1], f[3, 2], f[3, 3] \} \} \end{matrix}$
Array[Sin, 3, 0]	{0, Sin[1], Sin[2]}
Array[Sin, 4, 1, Plus]	Sin[1] + Sin[2] + Sin[3] + Sin[4]
Array[f, 5, 2, 2]	2[f[2], f[3], f[4], f[5], f[6]]

4.5.2. Векторные функции

К часто применяемым векторным функциям относятся функции точечного и кросс-произведения:

- **a.b.c** или **Dot[a, b, c]** – возвращает точечное произведение для векторов, матриц или тензоров.
- **Cross[a, b]** – возвращает кросс-произведение векторов **a** и **b**.

В системе Mathematica 5 к ним добавились следующие новые функции:

- **Total[list]** – дает общую сумму элементов листа.
- **Total[list, n]** – дает общую сумму элементов листа на уровне **n**.
- **Norm[expr]** – дает норму числа или массива.
- **Norm[expr, p]** – дает **p**-норму.

Применение этих функций достаточно очевидно, так что ограничимся следующими наглядными примерами:

```
Dot[a, b, x]
a.b.x
{a1, a2, a3}.{b1, b2, b3}
```

```

 $a_1 b_1 + a_2 b_2 + a_3 b_3$ 
u={1,2,3};
v={4,5,6};
w=u%%v
{-3,6,-3}
Total[{a,b,c}]
a+b+c
{Total[u],Total[v]}
{6,15}
vc={1,2+3*i,4+5*i}
{1,2+3 i,4+5 i}
Total[vc]
7+8 i
Norm[2+3*d]
 $\sqrt{13}$ 
Norm[{x1,x2,x3}]
 $\sqrt{\text{Abs}[x1]^2 + \text{Abs}[x2]^2 + \text{Abs}[x3]^2}$ 
Norm[{1,2,3}]
 $\sqrt{14}$ 

```

4.5.3. Функции для операций линейной алгебры

Следующая группа функций системы Mathematica позволяет осуществить основные операции над векторами и матрицами, используемыми в *линейной алгебре*.

- **Cross[v1,v2,v3,...]** – кросс-произведение векторов (может задаваться в виде $v1*v2*v3*...$).
- **Det[m]** – возвращает детерминант (определитель) квадратной матрицы **m**.
- **DiagonalMatrix[list]** – возвращает диагональную матрицу с главной диагональю, сформированной из элементов списка **list**, и нулевыми остальными элементами матрицы.
- **Dot[a, b, c]** – возвращает произведения векторов, матриц и тензоров. Операцию произведения можно задавать также и в виде **a.b.c**.
- **Eigensystem[m]** – возвращает список {values,vectors} собственных значений и собственных векторов данной квадратной матрицы **m**.
- **Eigenvalues[m]** – возвращает список собственных значений квадратной матрицы **m**.
- **Eigenvectors[m]** – возвращает список собственных векторов квадратной матрицы **m**.
- **IdentityMatrix[n]** – возвращает единичную матрицу с размером **n** (у нее диагональные элементы имеют значения 1, остальные 0).
- **Inverse[m]** – возвращает обратную матрицу для квадратной матрицы **m**, т.е. матрицу m^{-1} , которая, будучи умноженной на исходную матрицу, дает единичную матрицу.

- **MatrixExp[m]** – возвращает экспоненциал матрицы **m**.
- **MatrixPower[m, n]** – возвращает **n**-ую степень матрицы **m**.
- **MatrixQ[expr]** – возвращает True, если expr является списком списков, который может представлять матрицу, иначе возвращает False.
- **MatrixQ[expr, test]** – возвращает True, только если **test** дает True в применении к каждому элементу матрицы в **expr**.
- **Minors[m, k]** – возвращает матрицу, составленную из определителей всех **krk** субматриц **m**.
- **NullSpace[m]** – возвращает список векторов, которые формируют базис для нулевого пространства матрицы **m**.
- **Pivoting** – опция, относящаяся к функциям декомпозиции матрицы; указывает, что должен выполняться поворот столбца. Результат имеет форму $\{Q, R, P\}$, где **P** – матрица перестановок такая, что имеет место $M.P = \text{Conjugate}[\text{Transpose}[Q]].R$, где **M** – начальная (исходная) матрица.
- **PseudoInverse[m]** – ищет псевдообратную квадратной матрице **m**.
- **Tr[list]** – возвращает след матрицы или тензора (функция только у Mathematica 4).
- **Transpose[m]** – возвращает транспонированную матрицу, у которой столбцы и строки меняются местами, в сравнении с матрицей **m**.
- **RowReduce[m]** – возвращает приведенную к строке форму матрицы **m**.
- **ZeroTest** – опция для **LinearSolve** и других линейных алгебраических функций; дает функцию для применения ее к сочетаниям (комбинациям) из матричных элементов с целью определения, следует или нет полагать их равными нулю.

Следующие примеры иллюстрируют применение основных из этих функций (вывод в текстовом формате):

Ввод (In)	Вывод (Out)
A:=IdentityMatrix[3]	
A	{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
MatrixExp[A]	{{E, 0, 0}, {0, E, 0}, {0, 0, E}}
MatrixQ[A]	True
MatrixPower[MatrixExp[A], -1.5]	{{0.22313, 0, 0}, {0, 0.22313, 0}, {0, 0, 0.22313}}
A+{{1,2,3},{4,5,6},{7,8,9}}	{{2, 2, 3}, {4, 6, 6}, {7, 8, 10}}
m:={{1,2},{3,7}}	
MatrixForm[m]	1 2 3 7
Inverse[m]	{{7, -2}, {-3, 1}}
MatrixQ[m]	True
RowReduce[m]	{{1, 0}, {0, 1}}

Несколько примеров с выводом в стандартном формате представлены на рис. 4.5.

Все эти функции были и в системе Mathematica 4. Многие матричные функции в системе Mathematica 5/5.1/5.2 существенно доработаны, алгоритмы их работы улучшены. Введена также новая функция:

```

m := {{1, 2}, {3, 7}}

MatrixForm[m]


$$\begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix}$$


Det[m]

1

Eigensystem[m]


$$\left\{ \left\{ 4 - \sqrt{15}, 4 + \sqrt{15} \right\}, \left\{ \left\{ \frac{1}{3} (-3 - \sqrt{15}), 1 \right\}, \left\{ \frac{1}{3} (-3 + \sqrt{15}), 1 \right\} \right\} \right\}$$


Eigenvalues[m]


$$\{4 - \sqrt{15}, 4 + \sqrt{15}\}$$


Eigenvectors[m]


$$\left\{ \left\{ \frac{1}{3} (-3 - \sqrt{15}), 1 \right\}, \left\{ \frac{1}{3} (-3 + \sqrt{15}), 1 \right\} \right\}$$


```

Рис. 4.5. Примеры матричных операций с выводом в стандартном формате

CharacteristicPolynomial[m, x] – возвращает характеристический полином матрицы **m** с независимой переменной **x**.

Пример применения этой функции представлен ниже:

```

m =  $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ 
{{1, 2}, {3, 4}}
CharacteristicPolynomial[m, x]
-2 - 5x + x2

```

4.5.4. Функции декомпозиции матриц

При реализации ряда матричных методов используются различные виды декомпозиции (разложения матриц). Для числовой матрицы **m** они реализуются следующими функциями:

- **QRDecomposition[m]** – возвращает QR-разложение (декомпозицию).
- **LUDecomposition[m]** – возвращает результат LU-декомпозиции матрицы **m**.
- **CholeskyDecomposition[m]** – возвращает результат декомпозиции Холецки.
- **SchurDecomposition[m]** – возвращает результат декомпозиции Холецки.
- **SchurDecomposition[m, a]** – возвращает результат декомпозиции Холецки.
- **JordanDecomposition[m]** – возвращает результат декомпозиции Жордана для матрицы **m**.

В данной книге эти функции не используются, поскольку представляют интерес для специалистов в области матричных вычислений. Они могут разобраться с ними самостоятельно, воспользовавшись разъяснениями и примерами, приведенными в справке.

4.5.5. Решение систем линейных уравнений

Приведем также типовые примеры на решение *систем линейных уравнений* матричными методами. В первом из них решение выполняется в символьном виде на основании формулы $\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}$, где \mathbf{A} – матрица коэффициентов системы линейных уравнений, \mathbf{B} – вектор свободных членов. Для перемножения используется функция **Dot**, а для инвертирования матрицы – функция **Inverse**.

Пример решения системы линейных уравнений в символьном виде

```
A:={ {a,b} , {c,d} }
```

```
B:={e,f}
```

```
X:=Dot[Inverse[A],B]
```

```
X
```

$$\left\{ \frac{de}{-bc+ad} - \frac{bf}{-bc+ad}, -\frac{ce}{-bc+ad} + \frac{af}{-bc+ad} \right\}$$

Для решения систем линейных уравнений существует большое число самых различных методов, в том числе использующих различные типы декомпозиции матрицы коэффициентов системы. Возможно применение функций **Solve** и **RowReduce**. Однако на практике обычно применяется специальная функция для решения таких систем уравнений:

LinearSolve[m, b] – возвращает вектор \mathbf{x} – решение матричного уравнения $\mathbf{m}.\mathbf{x}==\mathbf{b}$, где \mathbf{m} – матрица коэффициентов левой части системы линейных уравнений, \mathbf{x} – вектор неизвестных и \mathbf{b} – вектор свободных членов в правой части системы.

Ниже представлен пример (второй) на ее применение для решения системы из двух уравнений в численном виде:

```
LinearSolve[{ {1,2} , {3,4} } , {7,9} ]
```

```
{-5, 6}
```

Нередко, например, в электротехнических расчетах, встречается необходимость решения *систем линейных уравнений с комплексными элементами*. Все описанные выше функции обеспечивают работу с комплексными числами. Следующий пример иллюстрирует решение системы линейных уравнений с комплексными данными с помощью функции **LinearSolve**:

```
A={ {1+2I,2+3I} , {3+4I,4+5I} }
```

```
{ { (1+2 i) , (2+3 i) } , { (3+4 i) , (4+5 i) } }
```

```
B={2I,3}
```

```
{2 i , 3 }
```

```
X=LinearSolve[A,B]
```

$$\left\{ \left(\frac{1}{4} - 4i \right) , \frac{11i}{4} \right\}$$

Число матричных функций в системе Mathematica 5/5.1/5.2 ограничено разумным минимумом, позволяющим реализовать множество других, более сложных матричных функций и преобразований. Их можно найти в пакетах расширения системы, посвященных линейной алгебре.

В Mathematica 6 введена полезная функция **LeastSquares**[*m*,*b*] решения противоречивой системы линейных уравнений методом наименьших квадратов. Так, в следующем примере система уравнений противоречива и не решается функцией **LinearSolve**:

```
LinearSolve[{{1,2},{3,4},{5,6}},{-1,0,2}]
```

```
LinearSolve::nosol : Linear equation encountered that has no solution. ⚡
```

```
LinearSolve[{{1,2},{3,4},{5,6}},{-1,0,2}]
```

Однако приближенное решение возможно:

```
LeastSquares[{{1,2},{3,4},{5,6}},{2,3,4}]
```

```
{-1,3/2}
```

В Mathematica 6 матричные функции и функции решения систем линейных уравнений существенно доработаны и улучшены. Введено и несколько редких в применении функций, интересных специалистам в области линейной алгебры. Существенно повышена скорость выполнения операций линейной алгебры. Оценим это на приведенном ниже примере.

Создадим матрицу размером 500×500 случайных чисел

```
A=Table[Random[],{i,1, 500},{j,1,500}];
```

```
Do[A[[i,i]]=2,{i,1,500}];
```

и вектор из 500 значений косинуса

```
B=Table[Cos[i*0.01],{i,1,500}];
```

Теперь решим систему из 500 линейных уравнений с оценкой времени решения:

```
X=Timing[LinearSolve[A,B]]; 
```

```
X[[1]]
```

```
0.062
```

Итак, система из 500 уравнений решена за время, менее одной сотой секунды. Оценим погрешность решения и убедимся в ее малости:

```
Max[Abs[A.X[[2]]-B]]
```

```
1.69966×10-12
```

Результаты говорят сами за себя.

4.6. Новые средства работы со списками в Mathematica 6

4.6.1. Работа с оператором ;; для списков

В Mathematica 6 для работы со списками может использоваться специальный оператор **;;**. Он используется для выделения группы элементов списков. Например, в форме **i;;j** он осуществляет выделение от *i*-го до *j*-го элементов, например:

```
{a1, a2, a3, a4, a5, a6, a7, a8}[[2;;6]]
{a2, a3, a4, a5, a6}
```

```
{1, 2, 3, 4, 5, 6, 7, 8}[[2;;5]]
{2, 3, 4, 5}
```

Этот оператор можно использовать для выделения заданного элемента из списка и его замены на другой элемент:

```
t={a, b, c, d, e, f, g, h}
{a, b, c, d, e, f, g, h}
```

```
t[[4]]
d
```

```
t[[4;;4]]
{d}
```

```
t[[2;;5]]={2, 3, 4, 5}
{2, 3, 4, 5}
```

```
t
{a, 2, 3, 4, 5, f, g, h}
```

В последнем примере ранее символьные элементы от второго по пятого заменены элементами—числами.

4.6.2. Новые функции для работы со списками

В Mathematica 6 добавлено несколько функций для операций с элементами списков. Функция **Accumulate** служит для создания из списка нового списка с аккумуляцией (последовательным сложением) элементов:

```
Accumulate[{a, b, c, d}]
{a, a+b, a+b+c, a+b+c+d}
```

```
Accumulate[{1, 2, 3, 4}]
{1, 3, 6, 10}
```

Функция **Differences** обеспечивает последовательное вычитание. Функция **Tally** возвращает список имеющихся в исходном списке объектов и их числа. Например:

```
Tally[{a, a, b, a, c, b, a, d, d, e}]
{{a, 4},{b, 2},{c, 1},{d, 2},{e, 1}}
```

Функция **Riffle** создает из заданного списка и отдельно заданного элемента новый список, в котором между каждым элементом заданного списка вставляется отдельно заданный элемент:

```
Riffle[{1, 2, 3, 4, 5}, a]
{1, a, 2, a, 3, a, 4, a, 5}
```

Возможно и такое применение функции **Riffle**, когда между элементами заданного списка вставляются поочередно элементы второго списка:

```
Riffle[{1, 2, 3, 4, 5, 6, 7},{x, y}]
{1, x, 2, y, 3, x, 4, y, 5, x, 6, y, 7, x, Null}
```

В следующем примере *x* вставляется после каждого четвертого элемента исходного списка:

```
Riffle[{1, 2, 3, 4, 5, 6, 7, 8, 9}, x, 4]
{1, 2, 3, x, 4, 5, 6, x, 7, 8, 9}
```

Функция **TakeWhile**[*list*,*crit*] возвращает элементы **Subscript**[*e*, *i*] с начала листа *list* до тех пор, пока соблюдается критерий, при котором **[Subscript**[*e*, *i*]] есть **True**. Например, в следующем примере выводятся элементы с начала списка, пока они четные:

```
TakeWhile[{2, 4, 8, 10, 2, 3, 4, 6, 8}, EvenQ]
{2, 4, 8, 10, 2}
```

Очень интересной является функция поиска кластеров – некоторых совокупностей данных, объединенных вокруг центров кластеризации. Для поиска кластеров используется следующая функция:

FindClusters [{ <i>e</i> ₁ , <i>e</i> ₂ ,...}]	FindClusters [{ <i>e</i> ₁ -> <i>v</i> ₁ , <i>e</i> ₂ -> <i>v</i> ₂ ,...}]
FindClusters [{ <i>e</i> ₁ , <i>e</i> ₂ ,...}->{ <i>v</i> ₁ , <i>v</i> ₂ ,...}]	FindClusters [{ <i>e</i> ₁ , <i>e</i> ₂ ,...}, <i>n</i>]

Примеры применения этой функции представлены ниже:

```
FindClusters [{1,3,3,11,12,3,1,13,15}]
{{1,1},{3,3,3},{11,12,13,15}}
FindClusters [{1,3,3,11,12,3,1,13,15},2]
{{1,3,3,3,1},{11,12,13,15}}
FindClusters [{1,3,3,11,12,3,1,13,15},4]
{{1,1},{3,3,3},{11},{12,13,15}}
FindClusters [{2□a,3□b,11□c,12□d,3□e,1□f,13□g,14□h}]
{{a,b,e,f},{c,d,g,h}}
```

На рис. 4.6 показан интересный пример на поиск 20 кластеров с центрами, начинающимися со слова «bit», в англоязычном словаре системы Mathematica 6.

4.6.3. Новые функции для массивов, векторов и матриц

Следующая функция возвращает список из элементов, чаще всего повторяющихся в списке:

Commonest[*list*]

Например, в данном случае выделяются элементы *b* и *c*, которые повторяются в списке трижды:

```
Commonest [{b,a,c,2,a,b,1,2,3,b,c,c}]
{b,c}
```

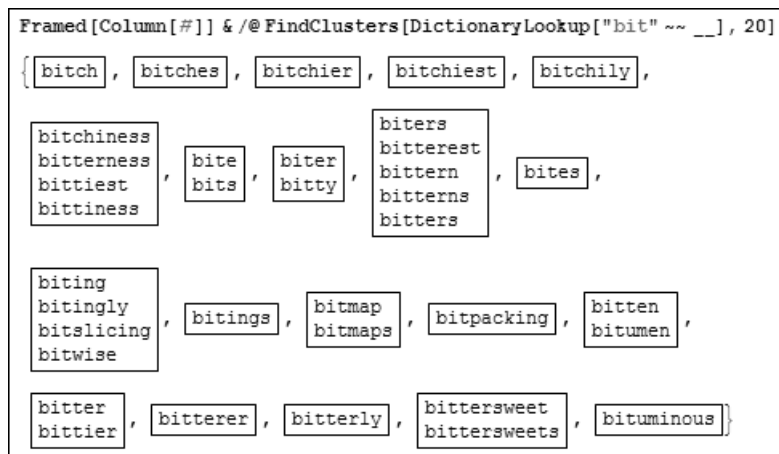



Рис. 4.6. Поиск 20 кластеров в словаре
с центрами, начинающимися со слова «бит»

Эта же функция возвращает n элементов, которые повторяются чаще всего:

Commonest[*list*,*n*]

Пример:

```
Commonest[{b,a,c,2,a,b,1,2,3,b,c,c},3]
{b,a,c}
```

Для создания одномерного или многомерного массивов из заданной константы с служит функция:

ConstantArray[*c*,*n*] **ConstantArray**[*c*,{*n*₁,*n*₂,...}]

Ее применение демонстрируют следующие примеры:

```
ConstantArray[c,5]
{c,c,c,c,c}
ConstantArray[c,{2,3}]/MatrixForm
```

$$\begin{pmatrix} c & c & c \\ c & c & c \end{pmatrix}$$

Стоит отметить и ряд новых функций для создания массивов, векторов и матриц и операций с ними:

- **ArrayFlatten** – конструирование матриц;
- **Band** – создание полос (например, диагональных) в многомерных массивах;
- **Normalize** – нормализация векторов и матриц;
- **Orthogonalize** – ортогонализация векторов и матриц;
- **Projection** – проекция одного вектора на другой;
- **KroneckerProduct** – вычисление произведения Кронекера;
- **LeastSquares** – решение матричной проблемы наименьших квадратов;

- **HermiteDecomposition** – выполнение декомпозиции Эрмита;
- **RotationMatrix** – создание матрицы вращения;
- **RandomInteger** – создает случайные числа, векторы и матрицы с ними;
- **RandomChoice** – создает полосы в массивах с элементами – случайными числами;
- **RandomSample** – создает списки с заданным числом случайных чисел в заданных их пределах;
- **SortBy** – сортирует элементы в списках.

С этими и некоторыми другими более узко специализированными числами можно познакомиться по справке.

4.7. Работа со строками

4.7.1. Функции работы со строками

Хотя Mathematica ориентирована на математические приложения, в ней достаточно полно представлены функции для работы со строками (strings). Они могут потребоваться как для организации вывода текстовых сообщений (например, надписей на графиках), так и для организации текстового диалога при разработке пакетов расширений и приложений системы.

Ввод строк осуществляется чаще всего с помощью клавиатуры посимвольно, а также с помощью файлов одновременно. Многие функции для работы со строками выполняют общепринятые преобразования, имеющиеся в большинстве языков программирования высокого уровня. *Строкой* является произвольная цепочка символов, заключенная в кавычки, например «**String**». Ниже представлены некоторые функции для работы со строками:

- **StringByteCount["string"]** – возвращает полное число байтов, используемых для хранения символов в строке "string";
- **StringDrop["string", {m, n}]** – возвращает строку "string", удалив в ней символы от m до n;
- **StringJoin["s1", "s2", ...]** или **StringJoin[{"s1", "s2", ...}]** – формирует строку, содержащую конкатенацию (объединение) указанных строк "si";
- **StringInsert["string1", "string2", M]** – вставляет строку "string2" в строку "string1", начиная с позиции M от начала этой строки (при –M вставка идет от конца указанной строки);
- **StringLength["string"]** – возвращает число символов в строке;
- **StringReplace["string", "s1" -> "sp1"]** или **StringReplace["string", {"s1" -> "sp1", "s2" -> "sp2", ...}]** – замещает "si" на "spi" всякий раз, когда они появляются как подстроки "string";
- **StringReverse["string"]** – меняет порядок символов в строке "string" на противоположный;
- **StringPosition["string", "sub"]** – возвращает лист с позициями строки "sub" в строке "string" (дополнительные формы см. в справочной системе);

- **StringTake["string", n]** — возвращает строку, состоящую из первых **n** символов строки "string";
- **StringTake["string", -n]** — возвращает последние **n** символов из строки "string";
- **StringTake["string", {n}]** — возвращает **n**-й символ в строке "string";
- **StringTake["string", {m, n}]** — возвращает строку из символов, расположенных в позициях от **m** до **n** строки "string".

Эти функции хорошо известны программистам, работающим с современными языками программирования. Большое число дополнительных функций для работы со строками можно найти в справке. Обилие таких функций в языке программирования системы Mathematica указывает на его универсальный характер и обширные возможности в решении даже на первый взгляд далеких от математики задач. Например — создание баз данных.

4.7.2. Примеры работы со строковыми функциями

Ниже приведены примеры действия ряда функций работы со строками, взятыми в кавычки наборами символов.

Ввод (In)	Вывод (Out)
<code>StringByteCount["Hello!"]</code>	6
<code>StringDrop["Hello my friend!", 6]</code>	my friend!
<code>StringDrop["Hello my friend!", -10]</code>	Hello
<code>StringDrop["Hello my friend!", {7}]</code>	Hello y friend!
<code>StringDrop["Hello my friend!", {6, 8}]</code>	Hello friend!
<code>StringInsert ["Hello friend!", " my", 6]</code>	Hello my friend!
<code>StringJoin["Hello", " my "]<>"friend!"</code>	Hello my friend!
<code>StringLength["Hello"]</code>	5
<code>StringPosition ["Hello my friend!", "e"]</code>	{{2, 2}, {13, 13}}
<code>StringReplace["Hilo", "i" -> "el"]</code>	Hello
<code>StringReverse["Hello!"]</code>	!olleH
<code>StringTake["Hello my friend!", 6]</code>	Hello
<code>StringTake["Hello my friend!", -8]</code>	friend!
<code>StringTake["Hello my friend!", {7, 9}]</code>	my

4.7.3. Дополнительные функции работы со строками

Отметим еще ряд дополнительных функций, относящихся к работе с символами и строками:

- **FromCharCode[n]** — возвращает строку, состоящую из одного символа с кодом **n**;

- **FromCharCode[{n1, n2,...}]** — возвращает строку, состоящую из последовательности символов с кодами **ni**;
 - **Characters["string"]** — возвращает список целочисленных кодов, соответствующих символам строки **"string"**;
 - **ToLowerCase["string"]** — производит строку, в которой все буквы преобразованы в нижний регистр;
 - **ToString[expr]** — возвращает строку, соответствующую форме вывода выражения **expr**. Опции устанавливают ширину линии, тип формата и т.д.;
 - **ToUpperCase["string"]** — вырабатывает строку, в которой все буквы преобразованы в верхний регистр;
 - **Unique[]** — создает новый символ с именем в форме **\$nnn** (**nnn** — уникальный порядковый номер);
 - **Unique[x]** — создает новый символ с именем в форме **x\$nnn** (**nnn** — уникальный порядковый номер);
 - **Unique[{x, y,...}]** — создает список новых символов с уникальными именами;
 - **Unique["xxx"]** — создает новый символ с именем в форме **xxxnnn** (**nnn** — уникальный порядковый номер);
 - **Unique[name, {attr1, attr2,...}]** — создает символ с указанными атрибутами **attri**;
 - **UpperCaseQ[string]** — возвращает **True**, если все символы строки **string** являются прописными буквами (верхнего регистра), иначе возвращает **False**.
- Примеры, приведенные ниже, показывают работу с этими функциями.

Ввод (In)	Вывод (Out)
<code>ToCharCode["Hello!"]</code>	<code>{72,101,108,108,111,33}</code>
<code>FromCharCode[72,101,108, 108,111,33]</code>	<code>Hello!</code>
<code>ToExpression["2+3*4"]</code>	<code>14</code>
<code>ToLowerCase["HeLlO!"]</code>	<code>hello!</code>
<code>ToUpperCase["Hello"]</code>	<code>HELLO</code>
<code>x:=ToString[2+3*4]</code>	
<code>x</code>	<code>14</code>
<code>Unique[]</code>	<code>\$1</code>
<code>Unique[xyz]</code>	<code>xyz\$2</code>
<code>Unique[xyz]</code>	<code>xyz\$3</code>
<code>UpperCaseQ["Hello"]</code>	<code>False</code>
<code>UpperCaseQ["HELLO"]</code>	<code>True</code>

Функции математического анализа

5.1. Функции вычисления сумм и произведений рядов	232
5.2. Функции вычисления производных	236
5.3. Вычисление первообразных и определенных интегралов	240
5.4. Вычисление пределов функций	248
5.5. Функции решения алгебраических и нелинейных уравнений	250
5.6. Решение дифференциальных уравнений	265
5.7. Функции минимизации и максимизации	269
5.8. Функции интегральных преобразований	276

5.1. Функции вычисления сумм и произведений рядов

В этой главе описаны основные операции математического анализа [88], детали которых можно найти в любом справочнике по высшей математике. В их числе, прежде всего, необходимо отметить суммы и произведения рядов, записанные в форме операторов:

$$\sum_{i=\text{imin}}^{\text{imax}} f_i \quad \text{и} \quad \prod_{i=\text{imin}}^{\text{imax}} f_i.$$

В этих операциях индекс i принимает целочисленные значения от минимального (начального) imin до максимального (конечного) imax с шагом, равным $+1$. Иной порядок задания изменений i недопустим.

Суммы и произведения рядов легко вычисляются численными математическими системами, и такие вычисления просто программируются на всех языках программирования. Однако, важным достоинством систем символьной математики, включая Mathematica, является вычисление сумм и произведений в аналитическом виде (если он есть) и при большом числе членов – вплоть до i , стремящегося к бесконечности.

5.1.1. Функция вычисления сумм

Для вычисления сумм в системе Mathematica предусмотрена функция **Sum**, используемая в ряде форм:

- **Sum[f, {i, imax}]** – вычисляет сумму значений f при изменении индекса i от значения от 1 до imax с шагом $+1$.
- **Sum[f, {i, imin, imax}]** – вычисляет сумму значений f при изменении индекса i от минимального значения $i=\text{imin}$ до максимального $i=\text{imax}$ с шагом $+1$.
- **Sum[f, {i, imin, imax, di}]** – вычисляет сумму значений f при изменении управляющей переменной вещественного типа от минимального значения $i=\text{imin}$ до максимального $i=\text{imax}$ с шагом di .
- **Sum[f, {i, imin, imax}, {j, jmin, jmax}, ...]** – вычисляет многократную сумму значений f при изменении индексов i от imin до imax с шагом $+1$, j от $jmin$ до $jmax$ с шагом $+1$ и т.д. (число индексов не ограничено).

Таким образом, эта функция обеспечивает расширенные возможности вычисления сумм, как при целочисленных, так и при вещественных значениях управляющих переменных, задающих циклы вычислений. Примеры использования функций суммирования:

```
Sum[i^2, {i, 10}]
```

```
385
```

```
Sum[i^2, {i, 1, 10}]
```

```
385
```

```
Sum[i^2, {i, 1, 2, 0.25}]
```

```
11.875
Sum[i*j, {i, 1, 10}, {j, 2, 5}]
770

$$\sum_{i=1}^{10} \sum_{j=2}^5 i*j$$

770
```

В последнем примере использована стандартная форма вывода: при ней суммирование представляется в виде оператора. Приведем еще ряд примеров выполнения операции суммирования:

```
Sum[ $\frac{x^n}{n!}$ , {n, 1, 9, 2}]

$$x + \frac{x^3}{6} + \frac{x^5}{120} + \frac{x^7}{5040} + \frac{x^9}{362880}$$

Sum[xi yj, {i, 1, 4}, {j, 1, i}]

$$x y + x^2 y + x^3 y + x^4 y + x^2 y^2 + x^3 y^2 + x^4 y^2 + x^3 y^3 + x^4 y^3 + x^4 y^4$$

Sum[ $\frac{1}{n*n}$ , {n, 1, ∞}]

$$\frac{\pi^2}{6}$$

Sum[i4, {i, 1, n}]

$$\frac{1}{30} n (1+n) (1+2 n) (-1+3 n+3 n^2)$$

Sum[ $\frac{i^3}{i+1}$ , {i, 1, n}]

$$\frac{1}{3} (3 - 3 \text{EulerGamma} + 2 n + n^3) - \text{PolyGamma}[0, 2 + n]$$

```

Из этих примеров видно, что Mathematica обеспечивает возможность символьного вычисления сумм, в том числе с бесконечным пределом суммирования. Вычисляются даже суммы (см. последний пример), выраженные через специальные математические функции.

5.1.2. Функция вычисления сумм в численном виде

Для вычисления сумм в численном виде используются следующие функции:

- **NSum[f, {i, imin, imax}]** – возвращает численное значение суммы f[i] при i, изменяющемся от imin до imax с шагом +1.
- **NSum[f, {i, imin, imax, di}]** – возвращает сумму численных значений функции f[i] при i, изменяющемся от imin до imax с шагом di.
- **NSum[f, {i, imin, imax}, {j, jmin, jmax}, ...]** – выполняет многомерное суммирование. Функция **NSum[...]** эквивалентна выражению **N[Sum[...]]**.

- **NSumTerms** – опция для NSum, задающая число членов, которые явно должны быть включены в сумму перед экстраполяцией.

Особенностью этой функции является возможность использования ряда опций, управляющих вычислительным процессом. Их можно (если это нужно) просмотреть с помощью команды **Options[NSum]**. Пример применения функции **Nsum** представлен ниже:

```
NSum[ $\frac{1}{i^3}$ , {i, 1, ∞}]
1.20206
```

Пример точного вычисления суммы (для сравнения) с помощью функции **Sum**:

```
truesum = Sum[ $\frac{1+k}{2^k} - \frac{k}{3^k}$ , {k, 1, 50}]
1818632874295681087853745424762603034467
808281277464764060643139600456536293376
N[%]
2.25
```

Пример той же суммы с помощью функции **Nsum** с опциями:

```
NSum[ $\frac{1+k}{2^k} - \frac{k}{3^k}$ , {k, 1, 50}, Method -> SequenceLimit,
NSumTerms -> 2, NSumExtraTerms -> 4] - truesum
0.0530365
```

При следующем наборе опций результат еще лучше (меньше погрешность):

```
NSum[ $\frac{1+k}{2^k} - \frac{k}{3^k}$ , {k, 1, 50}, Method -> SequenceLimit,
WorkingPrecision -> 30, NSumTerms -> 2, NSumExtraTerms -> 10,
WynnDegree -> 4] - truesum
-0. × 10-24
```

Функция вычисления суммы **Nsum** выполняется заметно быстрее, чем функция **Sum**, хотя на практике заметить это трудно (приведенные выше примеры все выполняются за доли секунды). Возвращаемый функцией **Nsum** результат вещественный.

5.1.3. Функция вычисления произведений

Операции вычисления произведений представлены следующими функциями:

- **Product[f, {i, imax}]** – возвращает произведения значений $f[i]$ для значений i , изменяющихся от 1 до $imax$.
- **Product[f, {i, imin, imax}]** – возвращает произведение значений $f[i]$ при изменении i от $imin$ до $imax$ с шагом +1.

- **Product[f, {i, imin, imax, di}]** – возвращает произведение f[i] при i, меняющемся от значения imin до значения imax с шагом di.
- **Product[f, {i, imin, imax}, {j, jmin, jmax}, ...]** – вычисляет многократное произведение (произведение по нескольким переменным).

Сказанное о характере изменения индекса i для сумм справедливо и для произведений. Примеры использования функций вычисления произведения:

Ввод (In)	Вывод (Out)
Product[i, {i, 10}]	3628800
NProduct[k^2, {k, 1, 5}]	14400.
NProduct[i^2, {i, 1, 2, 0.2}]	93.6405
Product[Log[i], {i, 2, 5, 0.5}]	4.23201 Log[2]

Следующий пример иллюстрирует вычисление произведения в символьном виде:

$$\prod_{i=1}^5 (x + i^2)$$

(1+x) (4+x) (9+x) (16+x) (25+x)

5.1.4. Функция вычисления произведений в численном виде

Для вычисления численных значений произведения используются следующие функции:

- **NProduct[f, {i, imax}]** – возвращает численное значение произведения значений f[i] для значений i, изменяющихся от 1 до imax.
- **NProduct[f, {i, imin, imax}]** – возвращает численное значение произведения значений f[i] при изменении i от imin до imax с шагом +1.
- **NProduct[f, {i, imin, imax, di}]** – возвращает численное значение произведения f[i] при i, меняющемся от значения imin до значения imax с шагом di.
- **NProduct[f, {i, imin, imax}, {j, jmin, jmax}, ...]** – вычисляет численное значение многократного произведения (произведение по нескольким переменным).

Эти функции применяются с опциями, которые можно вывести, используя команду **Options[NProduct]**. Ниже представлены примеры на использование функции **NProduct**:

Ввод (In)	Вывод (Out)
Product[i, {i, 10}]	3628800
NProduct[k^2, {k, 1, 5}]	14400.
NProduct[i^2, {i, 1, 2, 0.2}]	93.6405
Product[Log[i], {i, 2, 5, 0.5}]	4.23201 Log[2]

Пример точного вычисления с помощью функции **Product** (для сравнения):

```
trueproduct = Product[ $\frac{j}{1+j}$ , {j, 1, 50}]
```

$\frac{1}{51}$

В следующем примере вычисления с помощью функции **NProduct** дают результат с большой погрешностью:

```
NProduct[ $\frac{j}{1+j}$ , {j, 1, 50}, Method → SequenceLimit,  
NProductFactors → 2, NProductExtraFactors → 4] - trueproduct  
0.188235
```

Она резко снижается при лучшем подборе опций:

```
NProduct[ $\frac{j}{1+j}$ , {j, 1, 50}, Method → SequenceLimit,  
NProductFactors → 50, NProductExtraFactors → 4] - trueproduct  
 $-1.38778 \times 10^{-17}$ 
```

Применение функции **NProduct** оправдано высокой скоростью ее вычислений. Однако, как показывают приведенные примеры, к такому применению следует относиться с осторожностью, из-за возможности возникновения больших вычислительных погрешностей.

5.2. Функции вычисления производных

5.2.1. Основные функции для вычисления производных

К числу наиболее часто используемых математических операций принадлежит вычисление производных функций $f(x) - f \odot (x) = df(x)/dx$, как в аналитической, так и в символьной форме. Для этого используются следующие функции:

- **D[f, x]** – возвращает частную производную функции f по переменной x .
- **D[f, {x, n}]** – возвращает частную производную n -го порядка по x .
- **D[f, x1, x2, ...]** – возвращает смешанную производную.
- **Dt[f, x]** – возвращает обобщенную производную функции f по переменной x .
- **Dt[f]** – возвращает полный дифференциал f .

Название из одной буквы – это явно исключение из правил. Оно сделано осознанно в силу массовости этой операции.

Для функции **D** существует опция **NonConstants**, которая возвращает список объектов, находящихся в неявной зависимости от переменных дифференцирования. По умолчанию список пустой. Аналогично, для функции **Dt** имеется опция **Constant** (по умолчанию возвращает пустой список). На практике применение данных опций встречается редко.

Существует еще одна функция:

Derivative[n1, n2, ...][f] – основная (общая) форма представления функции, полученной в результате дифференцирования f $n1$ раз по первому аргументу, $n2$ раз по второму аргументу, и т.д.

К примеру, **Derivative[2][x*y]** возвращает $(x\ y)''$, а **Derivative[2,3][x*y]** соответственно $(x\ y)^{(2,3)}$.

5.2.2. Примеры вычисления производных

Следующие примеры показывают применение функции **D** для вычисления производной в аналитическом виде.

Производная тригонометрической функции

D [x Sin[x], x]
 $x \cos[x] + \sin[x]$

Производная экспоненциальной функции

D[Exp[x/b], x]
 $\frac{e^{\frac{x}{b}}}{b}$

Производная логарифмической функции

D[Log[3*x/4], x]
 $\frac{1}{x}$

Производная степенного многочлена

D[a*x^2+b*x+c, x]
 $b + 2ax$

Четвертая производная от x^n

D[x^n, {x, 4}]
 $(-3+n)(-2+n)(-1+n)x^{-4+n}$

Производная функции двух переменных

D[(x^m)*y^n, x, y]
 $mnx^{1+m}y^{1+n}$

Производная функции Бесселя

D[BesselJ[2, x], x]
 $\frac{1}{2} (\text{BesselJ}[1, x] - \text{BesselJ}[3, x])$

Производная ортогонального полинома Чебышева

D[ChebyshevT[4, x], x]
 $-16x + 32x^3$

Следующие примеры иллюстрируют вычисление производных от первого до третьего порядка включительно для функции $f[x]$, заданной пользователем:

f[x] := x / (1+x^2)

D[f[x], {x, 1}]

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

D[%, x]

$$\frac{8x^3}{(1+x^2)^3} - \frac{6x}{(1+x^2)^2}$$

D[f[x], {x, 2}]

$$\left(-\frac{4x}{(1+x^2)^2} + x \left(\frac{8x^2}{(1+x^2)^3} - \frac{2}{(1+x^2)^2} \right) \right)$$

D[D[f[x], x], x]

$$-\frac{48x^4}{(1+x^2)^4} + \frac{48x^2}{(1+x^2)^3} - \frac{6}{(1+x^2)^2}$$

D[f[x], {x, 3}]

$$\left(x \left(-\frac{48x^3}{(1+x^2)^4} + \frac{24x}{(1+x^2)^3} \right) + 3 \left(\frac{8x^2}{(1+x^2)^3} - \frac{2}{(1+x^2)^2} \right) \right)$$

Из предпоследнего примера видно, что для вычисления высших производных возможно последовательное применение функции D.

На рис. 5.1 показано построение графика функции $\sin[x]/x$, заданной как функция пользователя, и ее производной с помощью функции Plot.

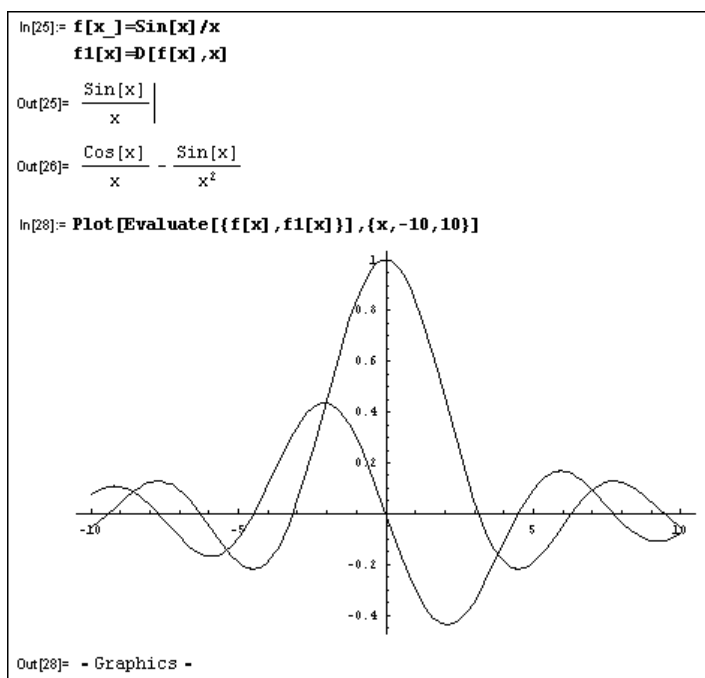


Рис. 5.1. График функции $\sin[x]/x$ и ее производной

В палитре Basic Input можно найти шаблоны для вычисления частных производных. Примеры их применения для экспоненциальной и логарифмической функций представлены ниже:

$\partial_{x,y} \text{Exp}[xy]$

$$e^{xy} + e^{xy} xy$$

$\partial_{x,x,y} \text{Exp}[xy]$

$$2 e^{xy} y + e^{xy} xy^2$$

$\partial_{x,y} \text{Ln}[xy]$

$$\text{Ln}'[xy] + xy \text{Ln}''[xy]$$

$\partial_{x,x,y} \text{Ln}[xy]$

$$2 y \text{Ln}''[xy] + xy^2 \text{Ln}^{(3)}[xy]$$

В целом, средства для символьного вычисления производных, имеющиеся в ядре системы Mathematica (особенно шестой версии), охватывают практически все важные типы математических выражений. Они могут включать в себя как элементарные, так и специальные математические функции, что выгодно отличает систему Mathematica от некоторых простых систем символьной математики, таких как Derive.

5.2.3. Примеры вычисления обобщенных производных

Использование функции Dt демонстрируют примеры, приведенные ниже:

$D[f[x], \{x, 2\}]$

$$\left(-\frac{4x}{(1+x^2)^2} + x \left(\frac{8x^2}{(1+x^2)^3} - \frac{2}{(1+x^2)^2} \right) \right)$$

$D[D[D[f[x], x], x], x]$

$$-\frac{48x^4}{(1+x^2)^4} + \frac{48x^2}{(1+x^2)^3} - \frac{6}{(1+x^2)^2}$$

$D[f[x], \{x, 3\}]$

$$\left(x \left(-\frac{48x^3}{(1+x^2)^4} + \frac{24x}{(1+x^2)^3} \right) + 3 \left(\frac{8x^2}{(1+x^2)^3} - \frac{2}{(1+x^2)^2} \right) \right)$$

$Dt[x^n, x]$

$$x^n \left(\frac{n}{x} + Dt[n, x] \text{Log}[x] \right)$$

$Dt[x \text{Sin}[x], x]$

$$(x \text{Cos}[x] + \text{Sin}[x])$$

$Dt[\text{Exp}[x/5], x]$

$$\frac{1}{5} e^{x/5}$$

Dt[x^a*y^b, x]

$$x^a y^b \left(\frac{a}{x} + \text{Dt}[a, x] \text{Log}[x] \right) + x^a y^b \left(\frac{b \text{Dt}[y, x]}{y} + \text{Dt}[b, x] \text{Log}[y] \right)$$

Dt[a*x^2+b*x+c, x]

$$(b + 2 a x + x^2 \text{Dt}[a, x] + x \text{Dt}[b, x] + \text{Dt}[c, x])$$

Dt[x^n, {x, 2}]

$$\left(x^n \left(\frac{n}{x} + \text{Dt}[n, x] \text{Log}[x] \right)^2 + x^n \left(-\frac{n}{x^2} + \frac{2 \text{Dt}[n, x]}{x} + \text{Dt}[n, \{x, 2\}] \text{Log}[x] \right) \right)$$

Dt[BesselJ[2, x], x]

$$\frac{1}{2} (\text{BesselJ}[1, x] - \text{BesselJ}[3, x])$$

Dt[ChebyshevT[4, x], x]

$$(-16 x + 32 x^3)$$

Dt[x^2*y^3, x, y]

$$6 x y^2 + 2 y^3 \text{Dt}[x, y] + 6 x^2 y \text{Dt}[y, x] + 6 x y^2 \text{Dt}[x, y] \text{Dt}[y, x]$$

Обратите внимание на то, что порой результаты для одного и того же дифференцируемого выражения у функций **D** и **Dt** заметно отличаются. Это вполне закономерно вытекает из различных определений данных функций.

5.3. Вычисление первообразных и определенных интегралов

5.3.1. Вычисление интегралов в символьном виде

Одна из важнейших операций – вычисление первообразных и определенных интегралов в символьном виде. Первообразная – это функция $F(x)$, удовлетворяющая уравнению:

$$\int f(x) dx = F(x) + C,$$

где C – постоянная интегрирования. А вычисление определенного интеграла с пределами – верхним b и нижним a – заключается в вычислении интеграла

$$\int_a^b f(x) dx = F(b) - F(a).$$

Заметим, что и определенный интеграл может быть представлен как аналитическим, так и численным значением. Для вычисления численных значений определенных интегралов создан ряд приближенных методов: от простых (прямоугольников и трапеций) до сложных методов, автоматически адаптирующихся к характеру изменения подынтегральной функции $f(x)$.

Для интегрирования в системе Mathematica используются следующие функции:

- **Integrate[f, x]** – возвращает первообразную (неопределенный интеграл) подынтегральной функции f по переменной x .

- **Integrate[f,{x, xmin, xmax}]** – возвращает значение определенного интеграла с пределами a=xmin до b=xmax.
- **Integrate[f,{x, xmin, xmax},{y, ymin, ymax},...]** – возвращает значение кратного интеграла с пределами от xmin до xmax по переменной x, от ymin до ymax по переменной y и т.д. (кратность реально не ограничена).

Обычно функция **Integrate** применяется прямо, но она имеет три характерные опции:

Options[Integrate]

```
{Assumptions->$Assumptions, GenerateConditions->Automatic,  
PrincipalValue->False}
```

Для обозначения бесконечных пределов используется константа **Infinity**. Эта константа означает положительную бесконечность, для задания отрицательной бесконечности используется эта константа со знаком минус перед ней. Пределы могут задаваться как константами, так и функциями.

Integrate[a*x^b,x]

$$\frac{a x^{1+b}}{1+b}$$

$$\int a x^b dx$$

$$\frac{a x^{1+b}}{1+b}$$

$$\int x \sqrt{x} dx$$

$$\frac{2 x^{5/2}}{5}$$

$$\int \{\text{Log}[x], \text{Exp}[x], \text{Sin}[x]\} dx$$

$$\{-x + x \text{Log}[x], e^x, -\text{Cos}[x]\}$$

D[%, x]

$$\{\text{Log}[x], e^x, \text{Sin}[x]\}$$

$$\int (a x^2 + b x + c) dx$$

$$c x + \frac{b x^2}{2} + \frac{a x^3}{3}$$

D[%, x]

$$c + b x + a x^2$$

Рис. 5.2. Примеры вычисления неопределенных интегралов с алгебраическими подынтегральными функциями (начало)

$$\int \frac{1}{a^2 + x^2} dx$$

$$\frac{\text{ArcTan}\left[\frac{x}{a}\right]}{a}$$

$$\int \frac{1}{\sqrt{9 x^2 + 4}} dx$$

$$\frac{1}{3} \text{ArcSinh}\left[\frac{3 x}{2}\right]$$

$$\int \frac{1}{2 - \sqrt{3} x} dx$$

$$-\frac{2 \sqrt{x}}{\sqrt{3}} - \frac{4}{3} \text{Log}\left[-2 + \sqrt{3} \sqrt{x}\right]$$

$$\int y e^{a y} dy$$

$$e^y \left(-\frac{e^a}{\text{Log}[e]^2} + \frac{e^a y}{\text{Log}[e]} \right)$$

Integrate[(x^3+5*x^2+2*x-4)/(x*(x^2+4)^2),x]

$$\frac{-12 - x}{4 (4 + x^2)} + \frac{3}{8} \text{ArcTan}\left[\frac{x}{2}\right] - \frac{\text{Log}[x]}{4} + \frac{1}{8} \text{Log}[4 + x^2]$$

Рис. 5.3. Примеры вычисления неопределенных интегралов с алгебраическими подынтегральными функциями (конец)

Особый интерес, естественно, вызывает применение функции **Integrate** для вычисления в символьном виде заданных пользователем неопределенных интегралов. Это иллюстрируют примеры на вычисление неопределенных интегралов с алгебраическими подынтегральными функциями, представленные на рис. 5.2 и 5.3.

Другая группа примеров (рис. 5.4) показывает нахождение интегралов с тригонометрическими и гиперболическими подынтегральными функциями.

$$\int \frac{2 + \sin[x] + 2 \cos[x]}{1 + \cos[x]} dx$$

$$2x - 2 \operatorname{Log}\left[\cos\left[\frac{x}{2}\right]\right]$$

$$\int \frac{1}{1 - \cos[x]} dx$$

$$-\cot\left[\frac{x}{2}\right]$$

$$\int \left(x^a + \frac{\sin[x]}{x}\right) dx$$

$$\frac{x^{1+a}}{1+a} + \operatorname{SinIntegral}[x]$$

$$\int \cos[x] x^3 dx$$

$$-6 \cos[x] + 3x^2 \cos[x] - 6x \sin[x] + x^3 \sin[x]$$

$$\int \cosh[x]^2 x dx$$

$$\frac{1}{8} (2x^2 - \cosh[2x] + 2x \sinh[2x])$$

$$\operatorname{Integrate}[\{\sin[x], \tan[x], \cosh[x], \operatorname{ArcSin}[x]\}, x]$$

$$\{-\cos[x], -\operatorname{Log}[\cos[x]], \sinh[x], \sqrt{1-x^2} + x \operatorname{ArcSin}[x]\}$$

Рис. 5.4. Примеры вычисления неопределенных интегралов с тригонометрическими подынтегральными функциями

Последний пример показывает, что возможно вычисление списка определенных интегралов, чьи подынтегральные функции представлены списком.

5.3.2. Примеры на вычисление определенных интегралов

Серия примеров (рис. 5.5) иллюстрирует вычисление в символьном виде определенных интегралов.

Следующая группа примеров (рис. 5.6) показывает вычисление определенных интегралов с пределами–функциями.

$$\begin{aligned}
& \int_a^b x^2 dx \\
& -\frac{a^3}{3} + \frac{b^3}{3} \\
& \int_0^a x \sqrt{1+x} dx \\
& \frac{4}{15} + \frac{2}{15} \sqrt{1+a} (-2+a+3a^2) \\
& \int_a^b x \operatorname{Log}[x] dx \\
& (a^2 - b^2 - 2a^2 \operatorname{Log}[a] + 2b^2 \operatorname{Log}[b]) / 4 \\
& \int_1^\infty \frac{1}{x \sqrt{x^2 - 1}} dx \\
& \frac{\pi}{2} \\
& \int_a^\infty x \operatorname{Exp}[-x^2] dx \\
& 1 / (2 \operatorname{Exp}[a^2]) \\
& \int_{-\infty}^\infty \frac{1}{x^2 + 6x + 12} dx \\
& \frac{\pi}{\sqrt{3}} \\
& \operatorname{Integrate}[\{x^n, \operatorname{Exp}[x], \operatorname{Log}[x], \operatorname{Sin}[x]/x\}, \{x, a, b\}] \\
& \left\{ -\frac{a^{1+n}}{1+n} + \frac{b^{1+n}}{1+n}, -E^a + E^b, \right. \\
& \quad \left. -a(-1 + \operatorname{Log}[a]) + b(-1 + \operatorname{Log}[b]), -\operatorname{SinIntegral}[a] + \operatorname{SinIntegral}[b] \right\}
\end{aligned}$$

Рис. 5.5. Примеры вычисления определенных интегралов
в символьном виде

Системы Mathematica имеют самые обширные возможности в вычислении интегралов. Ядро системы вобрало в себя формулы интегрирования из всех известных справочников и даже из древних рукописей. Эффективность интегрирования подтверждена Интернет-интегратором, который действует с 1966 года и позволит любому пользователю сети Интернет в два счета вычислить понадобившийся ему интеграл, даже не имея на своем ПК установленную систему Mathematica.

5.3.3. Примеры на вычисление кратных интегралов

Mathematica способна вычислять даже кратные интегралы с фиксированными и переменными верхним или нижним пределами. Кратный, например двойной, интеграл с фиксированными пределами имеет вид:

$$\int_0^{\sqrt{x}} x^2 dx$$

$$x^{3/2}/3$$

$$\int_{a,b}^x \sqrt{x+1} dx$$

$$\{-2*(-1-x)*\text{Sqrt}[1+x]\}/3 +$$

$$\{2*(-1-b-a*x)*\text{Sqrt}[1+b+a*x]\}/3$$

$$\int_{\text{Log}[x,b]}^{\text{Exp}[a*x]} (a x^b + c) dx$$

$$(E^{a*x}*(c+b*c+a*(E^{a*x})^b))/(1+b) -$$

$$(\text{Log}[b+x]*(c+b*c+a*\text{Log}[b+x]^b))/(1+b)$$

Рис. 5.6. Примеры вычисления определенных интегралов в символьном виде

$$\int_a^b \int_c^d f(x,y) dx dy.$$

Примеры вычисления двойных интегралов представлены на рис. 5.7.

$$\text{Integrate}[x^2 + y^2, \{x, 0, a\}, \{y, 0, a\}]$$

$$(2*a^4)/3$$

$$\int_0^a \int_0^a (x^2 + y^2) dy dx$$

$$(2*a^4)/3$$

$$\int_0^x \int_0^y \text{Sin}[xy^2] dy dx$$

$$x^2*y*\text{Sin}[xy^2]$$

$$\int_a^b \int_c^d xy dy dx$$

$$(-a^2/2 + b^2/2)*(-c^2/2 + d^2/2)$$

Рис. 5.7. Примеры вычисления двойных интегралов

Другая серия примеров (рис. 5.8) показывает, как вычисляются двойные и тройные интегралы, пределы которых сами по себе являются функциями.

Хотя вычисление двойного интеграла предусмотрено одной функцией `Integrate`, она не всегда дает результат. Как правило, вычисление кратных интегралов лучше производить, используя последовательно вычисление однократных интегралов, вложенных друг в друга. Это и показывают приведенные примеры.

Довольно часто интегрирование приводит к результатам, содержащим специальные математические функции. Примеры этого представлены на рис. 5.9.

Эти примеры наглядно показывают, что вычисление первообразных в системе может дать результаты, далекие от тривиального вычисления неопределенных интегралов, имеющих в обычных справочниках по математике. Кстати, и при вычислении тривиальных интегралов результат может оказаться иным, чем в справочниках, из-за различных преобразований, примененных для получения конечных формул. Подчас могут потребоваться определенные усилия для получения результата в заданной форме. Как подынтегральное выражение, так и результаты вычислений, могут содержать как элементарные, так и специальные математические функции.

Ядро систем Mathematica постоянно совершенствуется. Поэтому не исключены случаи, когда результаты интегрирования у разных версий системы Mathematica могут несколько отличаться, в основном, формой представления результата.

5.3.4. Численное интегрирование в Mathematica 5.1/5.2

Для вычисления численных значений определенных интегралов используется функция:

NIntegrate[f, {x, xmin, xmax}] – возвращает численное приближение интеграла от функции *f* по переменной *x* на интервале от *xmin* до *xmax*.

Она имеет ряд опций, вывести которые можно с помощью команды **Options[NIntegrate]**. Поскольку численное интегрирование широко используется на практике, отметим назначение данных опций:

$$\int \left(\int (x^3 + y^3) dx \right) dy$$

$$(x^4 + y^4)/4 + (x*y^4)/4$$

$$\int_0^1 \left(\int_2^{\sqrt{y}} (2 - x - y) dx \right) dy$$

$$11/30$$

$$\int_0^1 \left(\int_0^{\sqrt{1-y^2}} 4y dx \right) dy$$

$$4/3$$

$$\int_0^1 \left(\int_0^{\sqrt{1-x^2}} \frac{1}{x^2 + y^2 + 1} dy \right) dx$$

$$(Pi * ArcSinh[1])/4$$

$$\int_0^1 \left(\int_x^2 \left(\int_{xy}^{x^2 y^3} xy dz \right) dy \right) dx$$

$$2/189$$

$$\int_0^a \left(\int_0^{-z} \left(\int_0^{a-y-z} yz dx \right) dy \right) dz$$

$$a^5/120$$

$$\int_0^1 \left(\int_x^1 \left(\int_0^{1-y} x dz \right) dy \right) dx$$

$$1/12$$

Рис. 5.8. Примеры вычисления двойных интегралов с пределами-функциями

The figure displays three mathematical integrals and their corresponding Mathematica code snippets, illustrating the use of special functions for integration results.

Integral 1: $\int_0^1 \frac{1}{\sqrt{1-x^n}} dx$
Code: `If[Re[n] > 0, (Sqrt[Pi]*Gamma[1 + n^(-1)])/Gamma[1/2 + n^(-1)], Integrate[1/Sqrt[1 - x^n], {x, 0, 1}]]`

Integral 2: $\int_0^2 \frac{1}{\sqrt{1-x^n}} dx$
Code: `If[Re[n] > 0, Hypergeometric2F1[1/2, n^(-1), 1 + n^(-1), 2^n]/(2^(-n))^n^(-1), Integrate[1/Sqrt[1 - x^n], {x, 0, 2}]]`

Integral 3: $\int_0^\pi \frac{1}{\sqrt{1-x^5}} dx$
Code: `Pi*Hypergeometric2F1[1/5, 1/2, 6/5, Pi^5]`

Рис. 5.9. Примеры вычисления интегралов с результатами, представленными специальными функциями

- **AccurateGoal** – задает число цифр, задающих точность промежуточных результатов;
- **Compiled** – задает возможность компиляции функции;
- **DoubleExponential** – является вариантом для опции **Method** функции **NIntegrate**.
- **GaussPoints** – устанавливает количество точек в Гауссовой части квадратуры Гаусса-Кронрода;
- **MaxPoint** – задает максимальное число точек при интегрировании;
- **MaxRecursion** – задает максимальную глубину рекурсии;
- **Method -> DoubleExponential** – назначает для исполнения алгоритм двойной экспоненциальной сходимости;
- **Method -> MultiDimensional** – назначает для исполнения многомерный алгоритм. Имеет смысл только для интегрирования кратных интегралов;
- **Method -> GaussKronrod** – выбирает для исполнения адаптивную квадратуру Гаусса-Кронрода. При многомерном интегрировании **GaussKronrod** обращается к декартову произведению одномерных квадратурных формул Гаусса-Кронрода.
- **Method -> Trapezoidal** – назначает для решения рекурсивный метод трапеции. Он особенно успешен, если подынтегральная функция периодична и интервал интегрирования составляет точно один период. Для многомерного интегрирования данный метод обращается к декартову произведению одномерных правил трапеции;
- **MinRecursion** – задает минимальную глубину рекурсии;

- **PrecisionGoal** – задает погрешность вычислений;
- **SingularityDepth** – указывает, насколько допустима глубокая рекурсия перед тем, как начинается изменение переменной на границах интервала интегрирования.

Приведем примеры на использование функции **NIntegrate** без опций:

<code>NIntegrate[Sqrt[2*x+1], {x, 0, 1}]</code>	1.39872
<code>NIntegrate[1/(x*y), {x, 4, 4.4}, {y, 2, 2.6}]</code>	0.025006
<code>NIntegrate[x*y, {x, 0, 1}, {y, x, x^2}, {z, x*y, x^2*y^3}]</code>	0.010582
<code>NIntegrate[E^-x*Cos[x], {x, 0, Infinity}]</code>	0.5
<code>NIntegrate[1/Sqrt[1-x^6], {x, 0, 1}]</code>	1.21433
<code>N[Sqrt[Pi]*Gamma[1/6]/(6*Gamma[2/3])]</code>	1.21433
<code>NIntegrate[BesselJ[1, x]^3, {x, 0, 1}]</code>	0.0243409

Эти примеры показывают, что функция **NIntegrate** с успехом может применяться как для вычисления однократных, так и многократных определенных интегралов, в том числе с переменными пределами.

В большинстве случаев функция вычисляет определенные интегралы с опциями, установленными по умолчанию. Более того, несогласованная установка опций может вызвать сообщение об ошибке:

```
NIntegrate[Sin[Sin[x]], {x, 0, 1}]
NIntegrate[Sin[Sin[x]], {x, 0, 1}, WorkingPrecision->30]
0.430606103120690604912377
NIntegrate[Sin[Sin[x]], {x, 0, 1}, WorkingPrecision->30, PrecisionGoal
->50, MaxRecursion->20]
```

```
NIntegrate::tmap: NIntegrate is unable to achieve the tolerances
specified by the PrecisionGoal and AccuracyGoal options because the
working precision is insufficient. Try increasing the setting of
the WorkingPrecision option.
```

```
0.43060610312069060491237735248
```

Вообще говоря, умелое владение численным интегрированием невозможно без понимания особенностей определенных интегралов и знания методов их вычислений. Это особенно важно, когда необходима не просто обычная, а высокая точность вычислений.

5.3.5. Численное интегрирование в Mathematica 6

Функция **NIntegrate** в Mathematica 6 существенно улучшена. Лучше организована проверка на сингулярность, увеличено число автоматически выбираемых методов интегрирования. Узаконено вычисление кратных интегралов с применением следующей формы записи:

```
NIntegrate[f, {x, x_min, x_max}, {y, y_min, y_max}, ...]
```

Большинство пользователей устраивает автоматический выбор метода вычислений. С помощью опции `Method`→ можно выбрать метод из следующего списка: "GlobalAdaptive", "LocalAdaptive", "DoubleExponential", "MonteCarlo", "AdaptiveMonteCarlo", "QuasiMonteCarlo", "AdaptiveQuasiMonteCarlo". Возможно также применение методов, основанных на правилах (rules): "CartesianRule", "ClenshawCurtisRule", "GaussKronrodRule", "LobattoKronrodRule", "MultidimensionalRule", "MultipanelRule", "NewtonCotesRule", "TrapezoidalRule". Суть методов прямо вытекает из их названий.

Прямое указание метода нередко позволяет существенно (порой на порядок) уменьшить время вычисления определенных интегралов. Следующий пример наглядно иллюстрирует это:

```
NIntegrate[Cos[x2+y2], {x,0,Pi},{y,0,Pi}]/Timing
{0.844,-0.276982}
NIntegrate[Cos[x2+y2], {x,0,Pi},{y,0,Pi},Method->{"MultiDimensionalRule",
"Generators"->9}]/Timing
{0.094,-0.276982}
```

Время вычисления в этом примере, естественно, зависит от компьютера, на котором установлена система Mathematica. Но в любом случае вариант с указанием лучшего для вычисления заданного интеграла метода дает существенное ускорение вычислений. Все сказанное о численном интегрировании в системах Mathematica 5.1/5.2 полностью относится и к системе Mathematica 6. В справке последней можно найти разделы и учебные курсы, посвященные численному интегрированию и решению других задач в численном виде.

5.4. Вычисление пределов функций

5.4.1. Функция для вычисления пределов *Limit*

Многие функции стремятся к определенному пределу при приближении аргумента к некоторому значению или к некоторой области значений. Так, функция $\sin(x)/x$ при x , стремящемся к нулю (обозначим это как $x \rightarrow 0$), дает предел 1 в виде устранимой неопределенности $0/0$ (рис. 5.10).

Численные математические системы, равно как и большинство программ на обычных языках программирования, не воспринимают выражение $0/0 \rightarrow 1$ как объективную реальность. Их защитный механизм настроен на примитивное правило: ничего нельзя делить на 0. Следовательно, вычисление $\sin(x)/x$ при $x=0$ будет сопровождаться выдачей ошибки типа «Нельзя делить на 0!». Конечно, в данном конкретном случае можно предусмотреть особый результат – выдать 1 при $x=0$. Но это частный случай. В целом, подобные системы «не понимают» понятия предела.

Пределом некоторых функций может быть бесконечность, тогда как многие функции стремятся к конечному пределу при аргументе x , стремящемся к беско-

нечности. Система Mathematica не только вычисляет пределы функций, заданных аналитически, в виде числа, но и позволяет найти предел в виде математического выражения. Примеры применения функции **Limit** представлены на рис. 5.10.

Последние примеры показывают, что возможно вычисление пределов функций, устремляющихся к бесконечности, и вычисление пределов при переменной x , стремящейся в бесконечность. Вычисление пределов функций в аналитическом виде – важное достоинство систем символьной математики.

5.4.2. Опции функции вычисления пределов

При работе с функцией **Limit** используются следующие опции:

- **Analytic** – указывает, следует ли неопознанные функции интерпретировать как аналитические (по умолчанию Automatic).
- **Direction** – указывает направление, в котором происходит приближение к пределу. Опция используется в виде **Direction -> -1** (или +1), по умолчанию выбор остается за системой (Automatic). Значение +1 означает предел слева, а -1 – справа (казалось бы, должно быть наоборот, но заданно именно так).

Их применение поясняют примеры, представленные на рис. 5.11.

```
Limit[ $\frac{1}{\text{Log}[x]} - \frac{1}{x-1}$ , x->1]
1/2

Limit[ $\frac{x^x - a^a}{a^x - x^a}$ , x->a]
(1 + Log[a]) / (-1 + Log[a])

Limit[ $\frac{\text{Sin}[n x]}{x}$ , x->0]
n

Limit[ $\frac{x^2 + x + 2}{x^2 - 2 x - 3}$ , x->3, Direction->-1]
Infinity

Limit[ $\frac{x^2 + x + 2}{x^2 - 2 x - 3}$ , x->3, Direction->+1]
-Infinity

Limit[ $\frac{x^2 + x + 2}{x^2 - 2 x - 3}$ , x->3]
Infinity

Limit[ $\frac{x^2 + 1}{2 x^2 + 3}$ , x->Infinity]
1/2
```

Рис. 5.10. Примеры вычисления пределов некоторых выражений

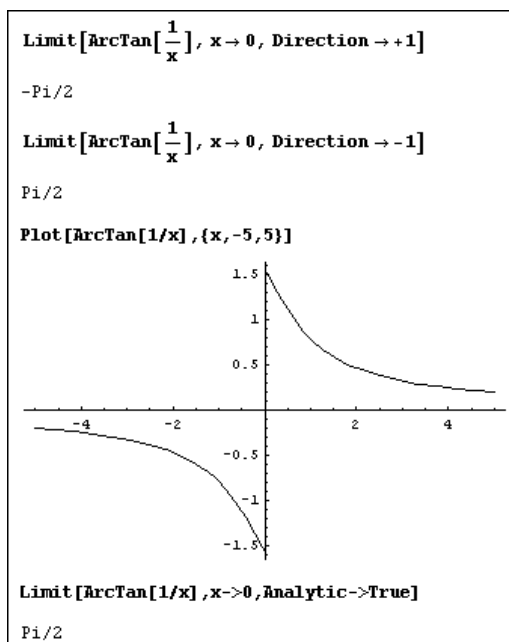


Рис. 5.11. Примеры вычисления пределов и график функции **ArcTan[1/x]**

График функции **ArcTan[1/x]** наглядно иллюстрирует роль опции **Direction**. На рис. 5.11 показано также вычисление пределов этой функции относительно точки разрыва $x=0$. Эта функция имеет разрыв в точке $x=0$, но вблизи точки разрыва значения функции конечны и равны $-\pi/2$ и $\pi/2$. Без применения опции **Direction** остается неясным, с какой стороны от разрыва вычисляется предел.

5.5. Функции решения алгебраических и нелинейных уравнений

5.5.1. Функция **Solve** для решения уравнений

Многие математические задачи сводятся к решению в общем случае нелинейных уравнений вида:

$$f(x)=0 \text{ или } f(x)=expr.$$

Они обозначаются как **eqns** (от слова *equations* – уравнения). Разумеется, могут решаться и системы, состоящие из ряда таких уравнений.

Для решения уравнений (как одиночных, так и систем) в численном и символьном виде **Mathematica** имеет функцию **Solve**:

- **Solve[eqns, vars]** – предпринимает попытку решить уравнение или систему уравнений относительно переменных **vars**.
- **Solve[eqns, vars, elims]** – пытается решать уравнения по переменным **vars**, исключая переменные **elims**.

Входные параметры этой функции могут быть представлены списками или записанными через объединительный знак **&&** выражениями. В **eqns** в качестве знака равенства используется знак **==**. Примеры на применение функции **Solve** представлены на рис. 5.12.

5.5.2. Решение систем нелинейных уравнений в символьном виде

Mathematica обеспечивает решение систем нелинейных уравнений. Примеры этого даны на рис. 5.13.

Весьма характерен второй пример из приведенных на рис. 5.13. Если в нем убрать функцию **N**, то будет получен чрезвычайно громоздкий, хотя и численный результат (проверьте это сами, поскольку размеры результата делают нецелесообразным его приведение в книге). Функция **N** осуществляет выполнение всех промежуточных вычислений, благодаря чему результат получается вполне обозримым.

В последнем примере получен набор из пяти пар корней, определенных через функцию **Root**. В свою очередь она означает вычисление корней полиномиального уравнения пятой степени. Данный пример является наглядной иллюстрацией того, что простота нелинейных уравнений порой оказывается весьма обманчивой, а их решение порой приводит к весьма громоздким и сложным результатам. Тем


```

Solve[x^7 - 1 == 0, x]

{{x -> 1}, {x -> -(-1)^(1/7)}, {x -> (-1)^(2/7)},
 {x -> -(-1)^(3/7)}, {x -> (-1)^(4/7)},
 {x -> -(-1)^(5/7)}, {x -> (-1)^(6/7)}}

Solve[Sqrt[x^3] == 2, x]

{{x -> (-2)^(2/3)}, {x -> 2^(2/3)},
 {x -> -((-1)^(1/3)*2^(2/3))}}

Solve[(x^3 - 1) / (x^2 + 1) == 0, x]

{{x -> 1}, {x -> -(-1)^(1/3)}, {x -> (-1)^(2/3)}}

Solve[Sin[Cos[x]] == .5, x]

Solve::ifun:
  Inverse functions are being used by Solve, so some
  solutions may not be found.

{{x -> -1.0197267436954502}, {x -> 1.0197267436954502}}

Solve[Sqrt[Abs[x]] == 2, x]

Solve::ifun:
  Inverse functions are being used by Solve, so some
  solutions may not be found.

{{x -> -4}, {x -> 4}}

```

Рис. 5.12. Примеры решения уравнений

не менее, возможность решения отдельных нелинейных уравнений и систем с ними в символьном виде трудно переоценить. К сожалению, далеко не все уравнения имеют такие решения, многие можно решать только в численном виде.

Не следует полагать, что Mathematica всегда выдает верное решение систем нелинейных уравнений. На самом деле решение иногда бывает ошибочным. Поэтому в большинстве случаев стоит оформлять решение таким образом, чтобы обеспечить проверку решений. Для этого рекомендуется отдельно задать систему уравнений и результат решения. Тогда проверка легко осуществляется с помощью подстановки (рис. 5.14).

5.5.3. Опции функции Solve

С функцией **Solve** можно использовать ряд опций:

- **InverseFunctions** – указывает, следует ли использовать обратные функции;
- **MakeRules** – указывает, должен ли результат быть представлен (обеспечен) как AlgebraicRulesData объект;
- **Method** – устанавливает алгоритм, используемый для вычисления результата (возможны методы 1, 2 и 3);
- **Mode** – задает характер решения уравнения (возможны Generic, Modular, и Rational);

```

Solve[{y == x^2, x == a + b}, {x, y}]

{{y -> a^2 + 2*a*b + b^2, x -> a + b}}

N[Solve[x^3-y^2==666x^2-y==3,{x,y}]]

{{y -> -1.1042265820067438 - 2.806597506186956*I,
  x -> -1.62521567537979 + 0.8634538629868598*I},
 {y -> -1.1042265820067438 + 2.806597506186956*I,
  x -> -1.62521567537979 - 0.8634538629868598*I},
 {y -> 0.3456042038060736, x -> 1.829099287574645},
 {y -> 2.862848960207415, x -> 2.421332063184935}}

eqns={x==2+3 a x,y==5+2 x};Solve[eqns,{x,y}]

{{y -> (3*(-3 + 5*a))/(-1 + 3*a), x -> -2/(-1 + 3*a)}}

Solve[{x^2 + y == a, y^2 == a + b}, {x, y}]

{{x -> -Sqrt[a - Sqrt[a + b]], y -> Sqrt[a + b]},
 {x -> Sqrt[a - Sqrt[a + b]], y -> Sqrt[a + b]},
 {x -> -Sqrt[a + Sqrt[a + b]], y -> -Sqrt[a + b]},
 {x -> Sqrt[a + Sqrt[a + b]], y -> -Sqrt[a + b]}}

Solve[a*y*x^2==c&&b*y^2+x==d,{x,y}]

{{x -> d - b*Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 1]^2,
  y -> Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 1]},
 {x -> d - b*Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 2]^2,
  y -> Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 2]},
 {x -> d - b*Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 3]^2,
  y -> Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 3]},
 {x -> d - b*Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 4]^2,
  y -> Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 4]},
 {x -> d - b*Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 5]^2,
  y -> Root[-c + a*d^2*#1 - 2*a*b*d*#1^3 + a*b^2*#1^5 &, 5]}}

```

Рис. 5.13. Примеры решения систем нелинейных уравнений

- **Sort** – устанавливает, нужна ли сортировка результатов;
- **Solutions** – устанавливает, следует ли проводить проверку полученных решений и удаление посторонних решений;
- **WorkingPrecision** – устанавливает число цифр промежуточных вычислений (по умолчанию Infinity).

Примеры на применение функции **Solve** с опцией **InverseFunctions** представлены на рис. 5.15. Обратите внимание на то, что заданное уравнение решается с предупреждением при отсутствии опции **InverseFunctions**, вообще не решается, если эта опция задана False, и гладко решается при **InverseFunctions->True**.

О том, насколько может влиять на решение опция **Method**, наглядно показывают примеры, показанные на рис. 5.16. Здесь одно и то же уравнение решается двумя различными методами.

Множество примеров на решение систем нелинейных уравнений в символьном виде можно найти в справочной системе Mathematica.

```
eqns = {x*y == 6, x^2 + y == 7};

result = Solve[eqns, {x, y}]

{{y -> -2, x -> -3}, {y -> 3, x -> 2}, {y -> 6, x -> 1}}
```

eqns /. result

```
{{True, True}, {True, True}, {True, True}}
```

e={x==2+3 a x, y==5+2 x}

```
{x == 2 + 3 a x, y == 5 + 2 x}
```

r=Solve[e,{x,y}]

$$\left\{ \left\{ y \rightarrow \frac{3(-3+5a)}{-1+3a}, x \rightarrow -\frac{2}{-1+3a} \right\} \right\}$$

e /. r

$$\left\{ \left\{ -\frac{2}{-1+3a} == 2 - \frac{6a}{-1+3a}, \frac{3(-3+5a)}{-1+3a} == 5 - \frac{4}{-1+3a} \right\} \right\}$$

Simplify[%]

```
{{True, True}}
```

Рис. 5.14. Проверка решений уравнений с помощью подстановок

5.5.4. Функции численного решения уравнений

Многие нелинейные уравнения и системы нелинейных уравнений в принципе не имеют аналитических решений. Однако их решение вполне возможно численными методами. Для численного решения систем нелинейных уравнений используется следующая функция:

- **NSolve[eqns, vars]** – пытается решать численно одно уравнение или систему уравнений относительно переменных vars.
- **NSolve[eqns, vars, elims]** – пытается решать численно уравнения относительно vars, исключая переменные elims.

С этой функцией используется единственная опция **WorkingPrecision**, задающая число верных цифр результата (по умолчанию 16).

Примеры использования функции **NSolve** для численного решения уравнений представлены на рис. 5.17.

Результаты решения с помощью функции **NSolve** также рекомендуется проверять с помощью подстановки. Пример этого для Mathematica 5.2 представлен ниже:

```
e=2*x^2 - 5*x - 15 == x^3
-15 - 5x + 2x^2 == x^3
```

```

Solve[{x+y+z==a,x*y*z==b,z^2==1},{x,y,z}]

{{x -> -1/2 + a/2 - Sqrt[1 - 2*a + a^2 - 4*b]/2,
  y -> (-1 + a + Sqrt[1 - 2*a + a^2 - 4*b])/2, z -> 1},
 {x -> -1/2 + a/2 + Sqrt[1 - 2*a + a^2 - 4*b]/2,
  y -> (-1 + a - Sqrt[1 - 2*a + a^2 - 4*b])/2, z -> 1},
 {x -> 1/2 + a/2 - Sqrt[1 + 2*a + a^2 + 4*b]/2,
  y -> (1 + a + Sqrt[1 + 2*a + a^2 + 4*b])/2, z -> -1},
 {x -> 1/2 + a/2 + Sqrt[1 + 2*a + a^2 + 4*b]/2,
  y -> (1 + a - Sqrt[1 + 2*a + a^2 + 4*b])/2, z -> -1}}

Solve[{x+y+z==6,x*y*z==6,z^2==9},{x,y,z},InverseFunctions->True]

{{x -> 1, y -> 2, z -> 3}, {x -> 2, y -> 1, z -> 3},
 {x -> 9/2 - Sqrt[89]/2, y -> (9 + Sqrt[89])/2, z -> -3},
 {x -> (9 + Sqrt[89])/2, y -> (9 - Sqrt[89])/2, z -> -3}}

Solve[2*Sin[x/2]*Sec[2*x]^(-1)==0,x]

Solve::ifun: Inverse functions are being used by Solve, so some
solutions may not be found.

{{x -> 0}, {x -> -Pi/4}, {x -> Pi/4}}

Solve[2*Sin[x/2]*Sec[2*x]^(-1)==0,x,InverseFunctions->True]

{{x -> 0}, {x -> -Pi/4}, {x -> Pi/4}}

Solve[2*Sin[x/2]*Sec[2*x]^(-1)==0,x,InverseFunctions->False]

Solve::tdep: The equations appear to involve the variables to be solved
for in an essentially non-algebraic way.

Solve[2*Cos[2*x]*Sin[x/2] == 0, x, InverseFunctions -> False]

```

Рис. 5.15. Примеры решения уравнений с опцией **InverseFunctions**

```

r=NSolve[e,x]
{{x->-1.47871}, {x->1.73935-2.66808 i},
 {x->1.73935+2.66808 i}}

e/.r
{True, True, True}

Options[NSolve]
{WorkingPrecision->Automatic, Sort->True,
 MonomialOrder->Automatic, Method->Automatic}

```

Нетрудно заметить, что в данном случае решение верно. Однако, увы, этот пример в системе Mathematica 5 уже не проходит. Скорее всего, сказывается прокрававшаяся в ядро новой системы ошибка.

```

Solve[x^3-y^2==7&& x^2-y==3,{x,y},Method->1]

{{-6*x^2 - x^3 + x^4 -> -16, y -> -3 + x^2}}

Solve[x^3-y^2==7&& x^2-y==3,{x,y},Method->3]

{{y -> 1, x -> 2}, {y -> -26/9 - (2*(89 - 6*Sqrt[159])^(1/3))/9 +
(89 - 6*Sqrt[159])^(2/3)/9 - (2*(89 + 6*Sqrt[159])^(1/3))/9 +
(89 + 6*Sqrt[159])^(2/3)/9 +
(2*{(89 - 6*Sqrt[159])*(89 + 6*Sqrt[159])^(1/3)})/9,
x -> -1/3 + (89 - 6*Sqrt[159])^(1/3)/3 + (89 + 6*Sqrt[159])^(1/3)/
3}, {y -> -26/9 + (89 - 6*Sqrt[159])^(1/3)/9 +
(I/3*(89 - 6*Sqrt[159])^(1/3))/Sqrt[3] - (89 - 6*Sqrt[159])^(2/3)/
18 + (I/6*(89 - 6*Sqrt[159])^(2/3))/Sqrt[3] +
(89 + 6*Sqrt[159])^(1/3)/9 - (I/3*(89 + 6*Sqrt[159])^(1/3))/
Sqrt[3] - (89 + 6*Sqrt[159])^(2/3)/18 -
(I/6*(89 + 6*Sqrt[159])^(2/3))/Sqrt[3] +
(2*{(89 - 6*Sqrt[159])*(89 + 6*Sqrt[159])^(1/3)})/9,
x -> -1/3 - ((1 + I*Sqrt[3])*(89 - 6*Sqrt[159])^(1/3))/6 -
((1 - I*Sqrt[3])*(89 + 6*Sqrt[159])^(1/3))/6},
{y -> -26/9 + (89 - 6*Sqrt[159])^(1/3)/9 -
(I/3*(89 - 6*Sqrt[159])^(1/3))/Sqrt[3] - (89 - 6*Sqrt[159])^(2/3)/
18 - (I/6*(89 - 6*Sqrt[159])^(2/3))/Sqrt[3] +
(89 + 6*Sqrt[159])^(1/3)/9 + (I/3*(89 + 6*Sqrt[159])^(1/3))/
Sqrt[3] - (89 + 6*Sqrt[159])^(2/3)/18 +
(I/6*(89 + 6*Sqrt[159])^(2/3))/Sqrt[3] +
(2*{(89 - 6*Sqrt[159])*(89 + 6*Sqrt[159])^(1/3)})/9,
x -> -1/3 - ((1 - I*Sqrt[3])*(89 - 6*Sqrt[159])^(1/3))/6 -
((1 + I*Sqrt[3])*(89 + 6*Sqrt[159])^(1/3))/6}}

```

Рис. 5.16. Примеры решения уравнения различными методами

5.5.5. Функции вычисления корней уравнений

Для вычисления корней уравнений, например, многочленов, используется функция **Roots**:

Roots[lhs==rhs, var] – дает дизъюнкцию уравнений, которая представляет корни уравнения.

Примеры применения функции **Roots** даны на рис. 5.18. Формат выдачи результатов у функции **Roots** отличается от такового для функции **Solve**.

При затруднениях в решении уравнений с помощью функции **Roots** можно использовать следующие опции:

- **Cubics** – указывает, следует ли искать явные решения для неприводимых кубических уравнений;
- **EquatedTo** – определяет выражение для замещения переменной в решении;
- **Modulus** – задает промежуточную факторизацию полинома;
- **Multiplicity** – устанавливает кратность каждого из корней в конечном результате;

```

NSolve[2*x^2 + 5*x - 15 == x^3, x]

{{x -> -2.4734}, {x -> 2.2367 - 1.03038 I}, {x -> 2.2367 + 1.03038 I}}

NSolve[x^5+8*x^4+31*x^3+80*x^2+94*x==20,x]

{{x -> -3.73205}, {x -> -2.}, {x -> -1. - 3. I}, {x -> -1. + 3. I}, {x -> -0.267949}}

NSolve[x^3==5,x,15]

{{x -> -0.854988 - 1.48088 I}, {x -> -0.854988 + 1.48088 I}, {x -> 1.70998}}

f[x]:=x^3-6*x^2+21*x
NSolve[f[x]==52,x]

{{x -> 1. - 3.4641 I}, {x -> 1. + 3.4641 I}, {x -> 4.}}

NSolve[{y^x^2==9,x^y^2==3},{x,y}]

{{x -> -1.5 - 2.59808 I, y -> -0.5 - 0.866025 I},
 {x -> -1.5 + 2.59808 I, y -> -0.5 + 0.866025 I}, {x -> 3., y -> 1.}}

NSolve[x^33==1,x]

{{x -> -0.995472 - 0.095056 I}, {x -> -0.995472 + 0.095056 I}, {x -> -0.959493 + 0.281733 I},
 {x -> -0.959493 - 0.281733 I}, {x -> -0.888835 - 0.458227 I}, {x -> -0.888835 + 0.458227 I},
 {x -> -0.786053 - 0.618159 I}, {x -> -0.786053 + 0.618159 I}, {x -> -0.654861 + 0.75575 I},
 {x -> -0.654861 - 0.75575 I}, {x -> -0.5 - 0.866025 I}, {x -> -0.5 + 0.866025 I},
 {x -> -0.327068 + 0.945001 I}, {x -> -0.327068 - 0.945001 I}, {x -> -0.142315 - 0.989821 I},
 {x -> -0.142315 + 0.989821 I}, {x -> 0.0475819 + 0.998867 I}, {x -> 0.0475819 - 0.998867 I},
 {x -> 0.235759 + 0.971812 I}, {x -> 0.235759 - 0.971812 I}, {x -> 0.415415 + 0.909632 I},
 {x -> 0.415415 - 0.909632 I}, {x -> 0.580057 - 0.814576 I}, {x -> 0.580057 + 0.814576 I},
 {x -> 0.723734 - 0.690079 I}, {x -> 0.723734 + 0.690079 I}, {x -> 0.841254 + 0.540641 I},
 {x -> 0.841254 - 0.540641 I}, {x -> 0.928368 - 0.371662 I}, {x -> 0.928368 + 0.371662 I},
 {x -> 0.981929 - 0.189251 I}, {x -> 0.981929 + 0.189251 I}, {x -> 1.}}

```

Рис. 5.17. Примеры применения функции **NSolve**

- **Quartics** – задает точное решение квадратного уравнения и полинома четвертой степени;
- **Using** – указывает какие-либо дополнительные уравнения, которые следует использовать для решения уравнений.

Применение опций нередко позволяет получать решения, которые не удаются с первого раза. Однако такое применение требует определенного опыта и понимания сути решаемой задачи.

5.5.6. Дополнительные функции для решения уравнений

Имеется также ряд дополнительных функций, которые могут использоваться для решения нелинейных уравнений или используются описанными ранее функциями:

Roots[x^2+2*x+15==0,x]

$x == -1 - I \sqrt{14} \mid x == -1 + I \sqrt{14}$

Roots[x^5+8*x^4+31*x^3+80*x^2+94*x+20==0,x]

$x == -2 - \sqrt{3} \mid x == -2 + \sqrt{3} \mid x == -1 - 3 I \mid x == -1 + 3 I \mid x == -2$

Roots[x^33==1,x]

$x == 1 \mid x == (-1)^{2/33} \mid x == (-1)^{4/33} \mid x == (-1)^{2/11} \mid x == (-1)^{8/33} \mid$
 $x == (-1)^{10/33} \mid x == (-1)^{4/11} \mid x == (-1)^{14/33} \mid x == (-1)^{16/33} \mid x == (-1)^{6/11} \mid$
 $x == (-1)^{20/33} \mid x == (-1)^{2/3} \mid x == (-1)^{8/11} \mid x == (-1)^{26/33} \mid x == (-1)^{28/33} \mid$
 $x == (-1)^{10/11} \mid x == (-1)^{32/33} \mid x == (-1)^{1/33} \mid x == (-1)^{1/11} \mid x == (-1)^{5/33} \mid$
 $x == (-1)^{7/33} \mid x == (-1)^{3/11} \mid x == (-1)^{1/3} \mid x == (-1)^{12/33} \mid x == (-1)^{5/11} \mid$
 $x == (-1)^{17/33} \mid x == (-1)^{19/33} \mid x == (-1)^{7/11} \mid x == (-1)^{22/33} \mid$
 $x == (-1)^{25/33} \mid x == (-1)^{9/11} \mid x == (-1)^{29/33} \mid x == (-1)^{31/33}$

N[Roots[x^33==1,x]]

$x == 1. \mid x == 0.981929 + 0.189251 I \mid x == 0.928368 + 0.371662 I \mid$
 $x == 0.841254 + 0.540641 I \mid x == 0.723734 + 0.690079 I \mid x == 0.580057 + 0.814576 I \mid$
 $x == 0.415415 + 0.909632 I \mid x == 0.235759 + 0.971812 I \mid x == 0.0475819 + 0.998867 I \mid$
 $x == -0.142315 + 0.989821 I \mid x == -0.327068 + 0.945001 I \mid x == -0.5 + 0.866025 I \mid$
 $x == -0.654861 + 0.75575 I \mid x == -0.786053 + 0.618159 I \mid x == -0.888835 + 0.458227 I \mid$
 $x == -0.959493 + 0.281733 I \mid x == -0.995472 + 0.095056 I \mid x == -0.995472 - 0.095056 I \mid$
 $x == -0.959493 - 0.281733 I \mid x == -0.888835 - 0.458227 I \mid x == -0.786053 - 0.618159 I \mid$
 $x == -0.654861 - 0.75575 I \mid x == -0.5 - 0.866025 I \mid x == -0.327068 - 0.945001 I \mid$
 $x == -0.142315 - 0.989821 I \mid x == 0.0475819 - 0.998867 I \mid x == 0.235759 - 0.971812 I \mid$
 $x == 0.415415 - 0.909632 I \mid x == 0.580057 - 0.814576 I \mid x == 0.723734 - 0.690079 I \mid$
 $x == 0.841254 - 0.540641 I \mid x == 0.928368 - 0.371662 I \mid x == 0.981929 - 0.189251 I$

Рис. 5.18. Примеры применения функции **Roots**

- **Auxiliary[v]** – применяется модулем **Solve** для указания того, что переменная v должна использоваться функцией **Roots** для результирующих решений, но соответствующие значения v не должны быть включены в окончательный ответ.
- **Eliminate[eqns, vars]** – исключает переменные $vars$ из системы совместных уравнений $eqns$.
- **FindRoot[lhs == rhs, {x, x0}]** – ищет численное решение уравнения $lhs == rhs$, начиная с $x == x0$.
- **MainSolve[eqns]** – основная функция для преобразования системы уравнений. **Solve** и **Eliminate** вызывают ее. Уравнения должны быть представлены в форме $lhs == rhs$. Они могут объединяться с помощью **&&** и **||**. **MainSolve** возвращает **False**, если не существует решения уравнений, и возвращает **True**, если все значения переменных являются решениями. **MainSolve** перестраивает уравнения, применяя определенные директивы.

- **MainSolve[eqns, vars, elim, rest]** – пытается перестраивать уравнения eqns так, чтобы найти решения для переменных vars и исключить переменные elim. Список rest может включаться для указания порядка исключения для любых остальных переменных.
 - **NRroots[lhs==rhs, var]** – возвращает список численных приближений корней полиномиального уравнения.
 - **Residue[expr, {x, x0}]** – ищет вычет expr в точке $x = x_0$.
 - **SolveAlways[eqns, vars]** – возвращает значения параметров, которые превращают уравнения eqns в тождества для всех значений переменных vars.
- Примеры на использование некоторых из этих функций приведены на рис. 5.19.

```

Eliminate[{x==2*y^2+3+y+4,y-1==z},y]

10 + 5*z + 2*z^2 == x

FindRoot[Sin[x],{x,4}]

{x -> 3.1415923871630587}

SolveAlways[a*x^2+b*x+c==0,x]

{{a -> 0, b -> 0, c -> 0}}

Reduce[a*x^2+b*x+c==0,x]

a == 0 && b == -1 || x == (-1 - b)/a && a != 0 || x == 0

Residue[Sin[z]/z^2,{z,0}]

1

FindRoot[x^3==Exp[x],{x,1}]

{x -> 1.8571838605052393}

```

Рис. 5.19. Примеры применения дополнительных функций решения уравнений

В целом, следует отметить, что система Mathematica обладает обширными средствами для решения уравнений и их систем. Умение их применять – залог правильного и эффективного решения сложных математических задач, относящихся к классу решения уравнений.

5.5.7. Графическая иллюстрация и выбор метода решения уравнений

При рассмотрении приведенных выше примеров может сложиться благодушное впечатление о том, что решение нелинейных уравнений может производиться автоматически и без размышлений. Но это далеко не так: представленные выше примеры просто подобраны так, что они имеют решение с помощью соответствующих функций.

На самом деле порой даже простые уравнения могут не иметь решения, как с помощью некоторых функций, так и вообще. В сложных случаях очень полезна графическая визуализация решения. В качестве примера на рис. 5.20 показан пример вычисления корней квадратного уравнения. В данном случае график функции явно показывает на существование двух действительных корней при x близких 0.2 и 2.3. Функция **Nsolve** без труда находит оба корня.

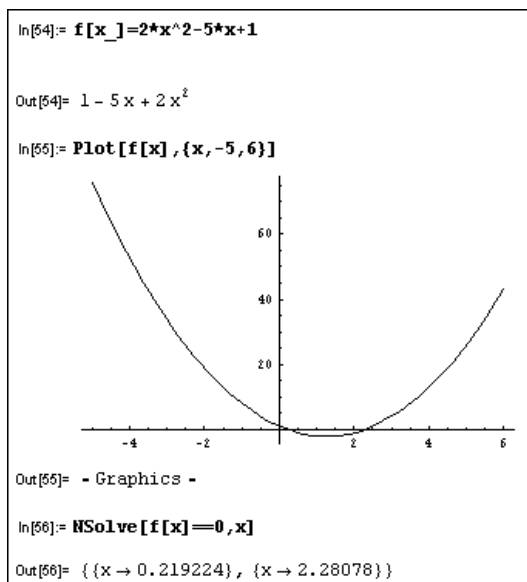


Рис. 5.20. Иллюстрация решения квадратного уравнения для случая двух действительных корней

А вот на рис. 5.21 показан случай, когда из-за изменения последнего члена квадратичной функции график ее уже не пересекает ось x вообще. Это говорит о том, что решения в виде действительных корней нет. И в самом деле, **Nsolve** находит корни как комплексные сопряженные числа. Действительная часть дает координату x для впадины кривой – параболы.

Если требуется решение равенства $f_1(x)=f_2(x)$, то для графической визуализации решения можно построить графики функций $f_1(x)$ и $f_2(x)$, наличие точек их пересечения будет означать существование действительных корней. Этот случай иллюстрирует рис. 5.22. В данном случае проблем с решением нет, поскольку по существу решается квадратное уравнение.

Но вот на рис. 5.23 показан случай решения уравнения $f(x)=\exp(x/2)$. Графики функций ясно показывают, что парабола пересекается экспонентой в двух точках. Однако функция **Nsolve** отказывается решать такое уравнение и выдает сообщение о том, что данное уравнение относится к трансцендентным.

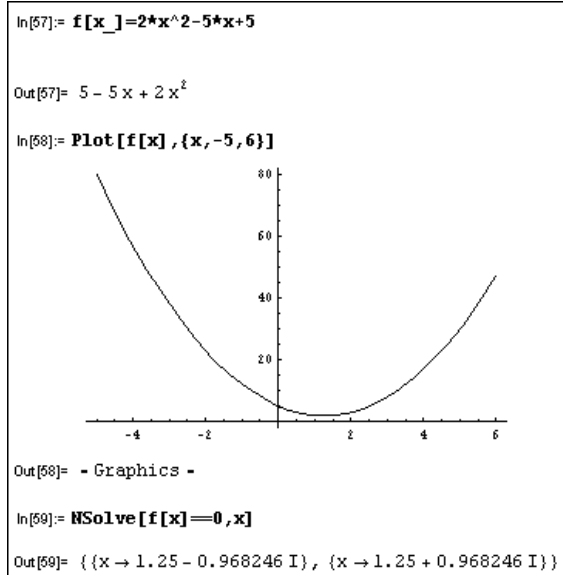


Рис. 5.21. Иллюстрация решения квадратного уравнения для случая двух комплексных корней

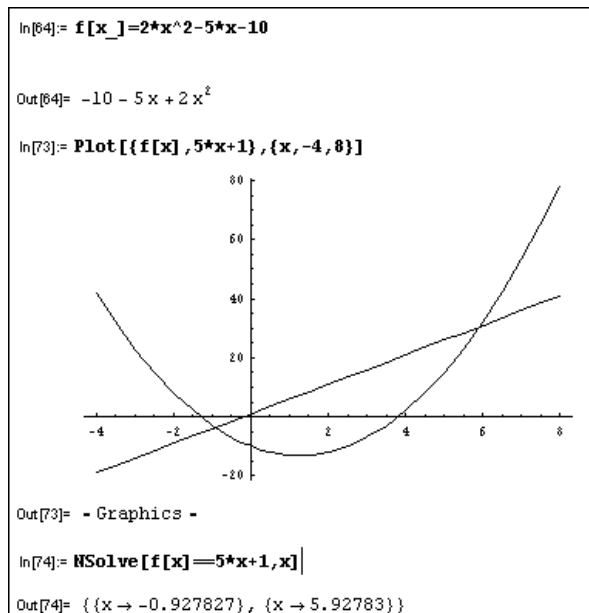


Рис. 5.22. Пример решения уравнения вида $f(x) = 5x + 1$

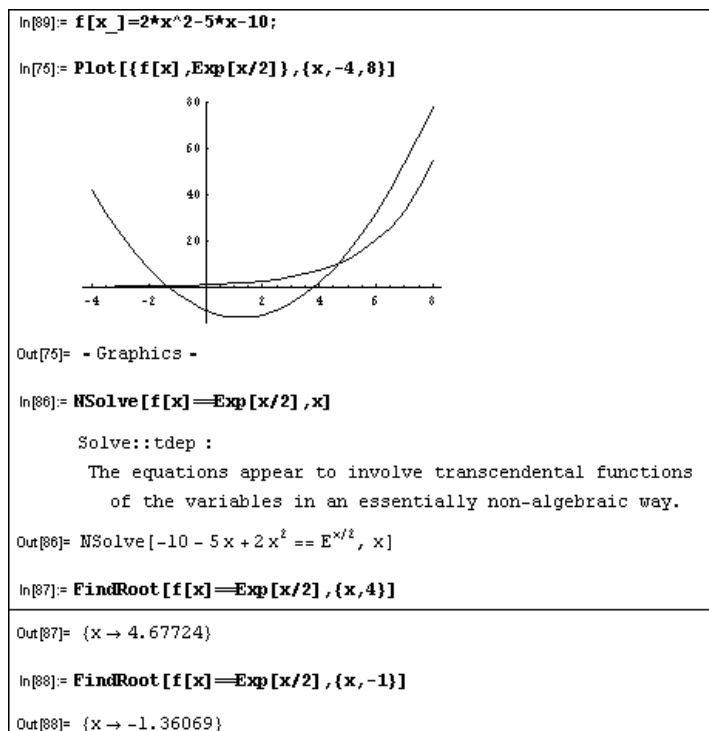


Рис. 5.23. Пример решения уравнения вида $f(x) = \exp(x/2)$

Таким образом, в данном случае наличие графического решения говорит о необходимости смены функции, которой до сих пор решались уравнения. Подходящей в данном случае является функция **FindRoot**, которая отыскивает одно решение вблизи заданной начальной точки. Применив ее дважды, нетрудно получить оба корня данного уравнения.

Приведенные примеры далеко не исчерпывают проблему графической визуализации решения и выбора методов решения. Однако они иллюстрируют возможности системы Mathematica в этой области и заостряют внимание на ее проблемах. Для реализации численных расчетов в систему Mathematica отобраны наилучшие и наиболее эффективные численные методы.

5.5.8. Получение одновременно нескольких корней

Многие уравнения с тригонометрическими функциями могут иметь периодические или близкие к ним решения. К сожалению, функции Mathematica, вычисляющие корни уравнений, не способны в этом случае дать сразу несколько корней.

Однако, ситуация тут далеко не безнадежна – приведенный ниже пример наглядно показывает это.

Пусть требуется в интервале изменения x от 0 до 20 найти все решения уравнения $x \cdot \sin(x) + x/2 - 1 = 0$.

График функции, представляющей левую часть уравнения, показан на рис. 5.24. Хорошо видно, что он пересекает ось x семь раз, т.е. имеет семь корней.

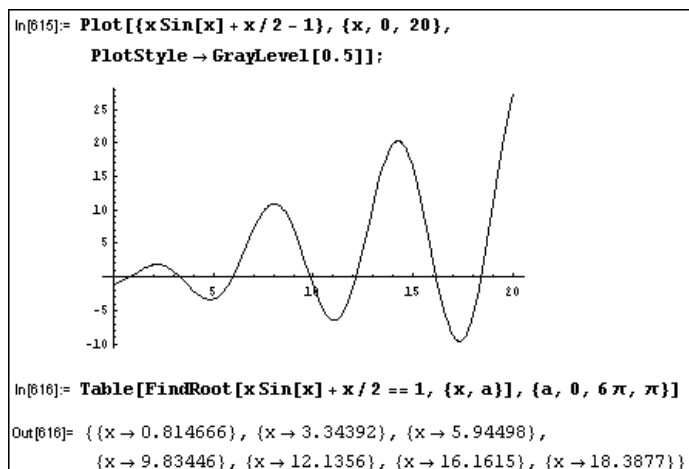


Рис. 5.24. График функции $x \cdot \sin(x) + x/2 - 1$ и пример вычисления всех ее корней в интервале изменения x от 20

Колесательная компонента функции обусловлена функцией $\sin(x)$, которая имеет нули в точках $0, \pi, 2\pi, 3\pi, \dots$. Однако, что видно из рис. 5.24, эти значения лишь приближенные, ввиду влияния других членов уравнения.

Ключевая идея получения всех корней уравнения заключается в получении нужных решений с помощью функции `FindRoot`, которой последовательно представляются приближенные решения. Однако, вместо уже испытанного приема – поиска корней поодиночке, можно воспользоваться таблицей решений, используя функцию `Table`. Приведенное под графиком функции решение на рис. 5.24 наглядно иллюстрирует возможности этого приема: найдены (или, вернее, уточнены) все семь корней исходного уравнения.

5.5.9. Получение неизвестных в явном виде

Читатель, возможно, обратил внимание на то, что решения всех представленных выше примеров выглядят не совсем обычно, в виде списка кажущихся подстановок. Это не позволяет использовать неизвестные в явном виде, например, для проверки решений или передачи найденных неизвестных в последующие вычисли-

тельные блоки. Однако от этого затруднения легко избавиться, если перед конструкцией блока решения использовать выражение следующего вида:

{x,y,z,...}/.

Список переменных в этом выражении должен однозначно соответствовать списку неизвестных системы уравнений. Покажем этот прием в действии. Ниже дано решение системы из трех нелинейных уравнений:

```
FindRoot[{x^2==9,y^2==16,x+y+z==10},{x,1.},{y,1.},{z,1.}]
{x→3.,y→4.,z→3.}
{x,y,z}
{ x, y, z}
```

Обратите внимание, что вывод списка {x,y,z} не дает полученных значений неизвестных. Это связано с тем, что переменные в блоке решения имеют локальный характер, и за пределами блока их значения (в том числе неопределенные) сохраняются такими, как они были до применения в блоке решения.

Теперь зададим решение в виде:

```
{x,y,z}/.FindRoot[{x^2==9,y^2==16,x+y+z==10},{x,1.},{y,1.},{z,1.}]
{3.,4.,3.}
```

Как видно, решение получено в виде списка с числами – явными значениями неизвестных. Можно обозначить их как a, b и c, получить список {a,b,c} и даже использовать их отдельно:

```
{a,b,c}=%
{3.,4.,3.}
a
3.
b
4.
c
3.
```

Можно проверить решение данной системы:

```
{a^2,b^2,a+b+c}
{9.,16.,10.}
```

Полученный вектор правых частей системы совпадает с заданным, что свидетельствует о правильности решения. Разумеется, вместо нового списка {a,b,c} для вектора решения можно было использовать и вектор {x,y,z}.

5.5.10. Решение рекуррентных уравнений

Для решения *рекуррентных разностных уравнений* в ядро Mathematica 4/5 введены функции:

- **RSolve[eqn,a[n],n]** – решает рекуррентное уравнение для a[n].
- **RSolve[eqn,a,n]** – решает рекуррентное уравнение для функции a.
- **RSolve[{eqn1, eqn2,...},{a1, a2,...},n]** – решает систему рекуррентных уравнений, представленных списками.

Ниже представлены примеры применения данных функций:

```
RSolve[a[n+1] == 2 a[n], a[n], n]
{{a[n] -> 2-1+n C[1]}}
```

```
RSolve[{a[n] == a[n-1] + a[n-2],
a[0] == a[1] == 1}, a[n], n]
{{a[n] -> 1/10 (5 (1/2 - sqrt(5)/2)n - sqrt(5) (1/2 - sqrt(5)/2)n +
5 (1/2 + sqrt(5)/2)n + sqrt(5) (1/2 + sqrt(5)/2)n)}}
```

```
RSolve[{a[0] == a[1] == 2, (n+1) (n+2) a[n+2] -
2 (n+1) a[n+1] - 3 a[n] == 0}, a[n], n]
{{a[n] -> (-1)n + 3n / Gamma[1+n]}}
```

```
RSolve[x[1+k] B[4(1-x[k])] x[k], x[k], k]
{{x[k] -> 1/2 - 1/2 Cos[2k C[1]]}}
```

```
RSolve[u[x] == 2 (x-1) u[x-1] / z - u[x-2], u[x], x]
{{u[x] BesselJ[x, z] C[1] + BesselY[x, z] C[2]}}
```

Нетрудно заметить, что решения подчас являются достаточно неожиданными. Например, во втором примере довольно необычно выглядит представление n -го Фибоначчи. Решения могут приводить к элементарным и специальным математическим функциям (см. последние примеры).

5.5.11. Решение уравнения Фробениуса в Mathematica 6

Уравнением Фрабениуса называется следующее уравнение: $a_1 x_1 + \dots + a_n x_n = b$. Оно может иметь множество решений. Для решения уравнения Фробениуса в систему Mathematica 6 введена функция:

FrobeniusSolve[{a₁, ..., a_n}, b] и **FrobeniusSolve[{a₁, ..., a_n}, b, m]**.

Пример:

```
FrobeniusSolve[{10, 15, 20, 25}, 20]
{{0, 0, 1, 0}, {2, 0, 0, 0}}
```

Проверим правильность решения:

```
%.{10, 15, 20, 25}
{20, 20}
```

В данном случае возвращаются два решения. Вторая форма данной функции обеспечивает возврат m решений. Пример:

```
FrobeniusSolve[{10, 15, 20, 25}, 100, 3]
{{1, 6, 0, 0}, {4, 4, 0, 0}, {7, 2, 0, 0}}
```

5.6. Решение дифференциальных уравнений

5.6.1. Решение дифференциальных уравнений в символьном виде

Дифференциальными уравнениями принято называть уравнения, в состав которых входят производные функции $y(x)$, представляющей решение уравнения. Дифференциальные уравнения могут быть представлены в различной форме, например в общеизвестной форме Коши

$$y'(x) = eqn = f(x, y).$$

Несколько дифференциальных уравнений образуют систему дифференциальных уравнений. Решение таких систем также возможно средствами Mathematica и подробно описано в ряде книг по использованию системы [5, 19–24, 84–90].

Дифференциальные уравнения и системы дифференциальных уравнений могут быть линейными и нелинейными. Для линейных уравнений обычно существуют решения в аналитическом виде. Нелинейные дифференциальные уравнения в общем случае аналитических решений не имеют, но могут решаться приближенными численными методами.

Дифференциальные уравнения широко используются в практике математических вычислений. Они являются основой при решении задач моделирования, особенно в динамике. Немногие математические системы имеют реализации численных методов решения систем дифференциальных уравнений. Но система Mathematica имеет средства как для символьного, так и численного решения дифференциальных уравнений и систем дифференциальных уравнений.

Для решения дифференциальных уравнений в символьном виде используются следующие средства:

- **DSolve[eqn, y[x], x]** – решает дифференциальное уравнение относительно функций $y[x]$ с независимой переменной x .
- **DSolve[{eqn1, eqn2, ...}, {y1[x1, ...], ...}, {x1, ...}]** – решает систему дифференциальных уравнений.
- **DSolveConstants** – опция к **DSolve**, определяющая постоянные интегрирования, которые могут быть возвращены.
- **StartingStepSize** – опция к **NDSolve**, определяющая величину начального шага.

Приведем примеры аналитического решения дифференциальных уравнений:

```
DSolve[Derivative[1][y][x] == 2*a*x^3, y[x], x]
```

```
{{ {Y[x] -> 1.5 x^4 + C[1]} }}
```

```
DSolve[{y1'[x] == 2 x^2, y2'[x] == 3 x}, {y1[x], y2[x]}, x]
```

$$\left\{ \left\{ y1[x] \rightarrow \frac{2x^3}{3} + C[1], y2[x] \rightarrow \frac{3x^2}{2} + C[2] \right\} \right\}$$

DSolve[$y''[x] - y'[x] - 6y[x] == 0, y[x], x$]

$$\{ \{ y[x] \rightarrow e^{-2x} C[1] + e^{3x} C[2] \} \}$$

DSolve[$y''[x] + 4y'[x] == 10 \sin[2x], y[x], x$]

$$\left\{ \left\{ y[x] \rightarrow -\frac{1}{4} e^{-4x} C[1] + C[2] - \cos[2x] - \frac{1}{2} \sin[2x] \right\} \right\}$$

DSolve[$y'[x] == \sin[e^x], y[x], x$]

$$\{ \{ y[x] \rightarrow C[1] + \text{SinIntegral}[e^x] \} \}$$

DSolve[$z^2 w'[z] + z w'[z] - (z^2 + 1) w[z] == 0, w[z], z$]

$$\{ \{ w[z] \rightarrow \text{BesselJ}[1, -i z] C[1] + \text{BesselY}[1, -i z] C[2] \} \}$$

DSolve[$y'[x] - (a - 2q \cosh[2x]) y[x] == 0, y[x], x$]

$$\{ \{ y[x] \rightarrow C[1] \text{MathieuC}[a, q, -i x] + C[2] \text{MathieuS}[a, q, -i x] \} \}$$

Как нетрудно заметить, аналитические решения дифференциальных уравнений могут содержать не только элементарные, но и специальные математические функции, что заметно расширяет возможности применения системы Mathematica в решении задач динамического моделирования. Именно они обычно сводятся к решению дифференциальных уравнений.

В решении дифференциальных уравнений встречаются постоянные интегрирования. В общем случае они обозначаются как $C[i]$. Однако с помощью опции **GeneratedParameters** можно сменить обозначения постоянных интегрирования, что иллюстрируют следующие примеры:

DSolve[$y'[x] == y'[x] + y[x] + a, y, x$]

$$\left\{ \left\{ y \rightarrow \text{Function}[\{x\}, -a + e^{\left(\frac{1}{2} - \frac{\sqrt{5}}{2}\right)x} C[1] + e^{\left(\frac{1}{2} + \frac{\sqrt{5}}{2}\right)x} C[2]] \right\} \right\}$$

DSolve[$y'[x] == y'[x] + y[x] + a, y, x, \text{GeneratedParameters} \rightarrow K$]

$$\left\{ \left\{ y \rightarrow \text{Function}[\{x\}, -a + e^{\left(\frac{1}{2} - \frac{\sqrt{5}}{2}\right)x} K[1] + e^{\left(\frac{1}{2} + \frac{\sqrt{5}}{2}\right)x} K[2]] \right\} \right\}$$

В записи дифференциальных уравнений можно ввести граничные условия, которые должны учитываться при решении. Пример этого (и проверки) решения представлен ниже:

DSolve[$\{y'[x] == a y'[x] + y[x], y[0] == 1, y'[0] == 0\}, y, x$]

$$\left\{ \left\{ y \rightarrow \text{Function}[\{x\}, \frac{1}{2\sqrt{4+a^2}} \left(a e^{\frac{1}{2}(a-\sqrt{4+a^2})x} + \sqrt{4+a^2} e^{\frac{1}{2}(a+\sqrt{4+a^2})x} - a e^{\frac{1}{2}(a-\sqrt{4+a^2})x} + \sqrt{4+a^2} e^{\frac{1}{2}(a+\sqrt{4+a^2})x} \right) \right\} \right\}$$


```
{y''[x]==ay'[x]+y[x], y[0]==1, y'[0]==0} /. %//Simplify
{{True, True, True}}
```

В следующем примере решение задается при граничном условии $y(1)=1$:

```
DSolve[{y'[x] == (-1 + x)^-1 + x (1 + Log[-1 + x]), y[1] == 1}, y[x], x]
{{y[x] -> (-1 + x)^-1 + x}}
```

В справке по функции DSolve можно найти символьные решения ряда дифференциальных уравнений специального типа, например Абеля, Риккати, Матье и др. Ниже представлен пример на решение дифференциального уравнения Абеля:

```
DSolve[y'[x] + y[x]^3 == 1, y[x], x]
```

```
Solve::tdep : The equations appear to involve the variables
to be solved for in an essentially non-algebraic way. More...
```

```
{{y[x] -> InverseFunction[
  - ArcTan[ 1 + 2 #1 ] / Sqrt[3] + 1/3 Log[-1 + #1] - 1/6 Log[1 + #1 + #1^2] &] [-x + C[1]] ]}}
```

Mathematica способна также решать системы дифференциально-алгебраических уравнений, например вида $F(t, x, x') = \text{expr}$. Ниже представлен пример решения системы дифференциально-алгебраических уравнений с проверкой решения:

```
eqns = {x'[t] - y[t] == 1, x[t] + y[t] == 2};
sol = DSolve[eqns, {x, y}, t]
```

```
Out[6]= {{x -> Function[{t}, 1/4 (12 + e^-t C[1])], y -> Function[{t}, 2 + 1/4 (-12 - e^-t C[1]) ]}}
```

```
eqns/.sol//Simplify
{{True, True}}
```

Обратите внимание на то, что ответ получен через чистые функции. Они были описаны в Главе 2 и представляют собой функции без конкретного имени.

5.6.2. Решение дифференциальных уравнений в частных производных

Дифференциальные уравнения нередко содержат частные производные и называются *дифференциальными уравнениями в частных производных*. Их определения и некоторые свойства хорошо известны и описаны во многих книгах [28, 29, 84–90]. Для решения таких уравнений в системе Mathematica предусмотрена функция **DSolve**, параметры которой были уже описаны. В справке по системе и во встроенной книге Вольфрама можно найти множество примеров на решение дифференциальных уравнений и систем дифференциальных уравнений в частных производных. В связи с этим ограничимся приведением нескольких примеров на решение таких уравнений:

```

DSolve[D[y[x1, x2], x1] + D[y[x1, x2], x2] == a*x1/x2,
      y[x1, x2], {x1, x2}]
{{y[x1, x2] → a x1 + a x1 Log[x2] - a x2 Log[x2] + C[1] [-x1 + x2]}}
DSolve[D[y[x1, x2], x1] + D[y[x1, x2], x2] == a*x1/x2,
      y, {x1, x2}]
{{y → Function[{x1, x2}, a x1 + a x1 Log[x2] - a x2 Log[x2] + C[1] [-x1 + x2]]}}
(c^2 D[#, x, x] - D[#, t, t]) & [y[x, t]] == 0
-y^(0,2)[x, t] + c^2 y^(2,0)[x, t] == 0
DSolve[x1 D[y[x1, x2], x1] + x2 D[y[x1, x2], x2]
      == Exp[x1/x2], y[x1, x2], {x1, x2}]
{{y[x1, x2] → e^(x1/x2) Log[x1] + C[1] [x2/x1]}}
DSolve[x1 D[y[x1, x2], x1] + x2 D[y[x1, x2], x2]
      == Exp[x1 x2], y[x1, x2], {x1, x2}]
{{y[x1, x2] → 1/2 (ExpIntegralEi[x1 x2] + 2 C[1] [x2/x1])}}
DSolve[D[y[x1, x2], x1] D[y[x1, x2], x2] == w^2, y[x1, x2], {x1, x2}]
DSolve[y^(0,1)[x1, x2] y^(1,0)[x1, x2] = w^2, y[x1, x2], {x1, x2}]

```

Из этих примеров хорошо видны формы задания дифференциальных уравнений в частных производных и формы вывода их решений.

5.6.3. Решение дифференциальных уравнений в численном виде

Многие дифференциальные уравнения не имеют аналитических решений, например, нелинейные. Однако они могут с приемлемой точностью решаться численными методами. Для численного решения систем дифференциальных уравнений используется функция:

- **NDSolve[eqns, y, {x, xmin, xmax}]** – ищет численное решение дифференциальных уравнений eqns относительно функции y независимой переменной x в интервале от xmin до xmax.
- **NDSolve[eqns, {y1, y2, ...}, {x, xmin, xmax}]** – ищет численные решения относительно функций yi.
- **MaxSteps** – опция к **NDSolve**, которая определяет максимальное количество шагов.

Часто весьма желательно выводить результаты решения дифференциальных уравнений в графической форме. Рисунок 5.25 поясняет, как это делается при решении системы нелинейных дифференциальных уравнений, описывающих достаточно сложный колебательный процесс.

Нередко решение предпочитают представить на фазовой плоскости. Рисунок 5.26 иллюстрирует такую возможность. Более того, учитывая, что решается система из трех дифференциальных уравнений, фазовая траектория решения находится в трехмерном пространстве.

Простота задания решения и вывода его результатов в графической форме открывают широкие возможности в применении системы для математического мо-

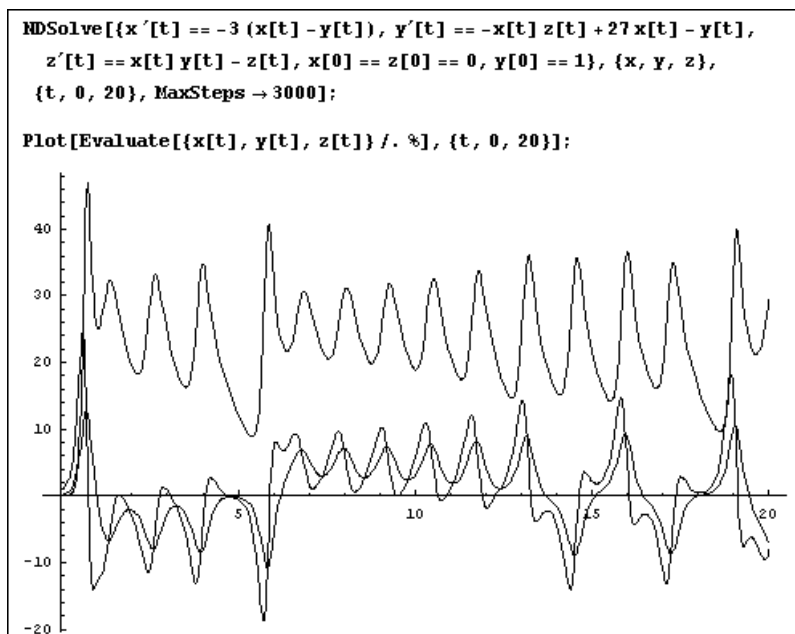


Рис. 5.25. Решение системы дифференциальных уравнений с выводом решения в виде графиков временных зависимостей

делирования сложных явлений. При этом, в отличие от такого решения с помощью обычных языков высокого уровня (например, Фортран, Бейсик, Паскаль или Си), не требуется составления каких-либо программ по реализации численных методов решения систем дифференциальных уравнений, скажем, таких, как метод Рунге-Кутты. Они представлены в виде уже готовых функций.

Несмотря на сказанное, представляется, что степень визуализации решений дифференциальных уравнений даже в последней версии системы Mathematica 5 несколько уступает таковой у конкурирующей версии системы Maple 8. В частности, из-за наличия у последней специальных пакетов расширения, дающих пользователю самую изощренную технику визуализации решения дифференциальных уравнений различного класса.

5.7. Функции минимизации и максимизации

В практике математических прикладных вычислений важная роль принадлежит оптимизационным задачам, например, таким, как поиск минимальных и максимальных значений функций одной или ряда переменных. Mathematica дает разнообразные возможности по решению задач оптимизации: от поиска элементов

```

NDSolve[{x'[t] == -3 (x[t] - y[t]), y'[t] == -x[t] z[t] + 27 x[t] - y[t], z'[t] == x[t] y[t],
  x[0] == z[0] == 0, y[0] == 1}, {x, y, z}, {t, 0, 20}, MaxSteps -> 3000];

ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. %], {t, 0, 20}, PlotPoints -> 1000];

```

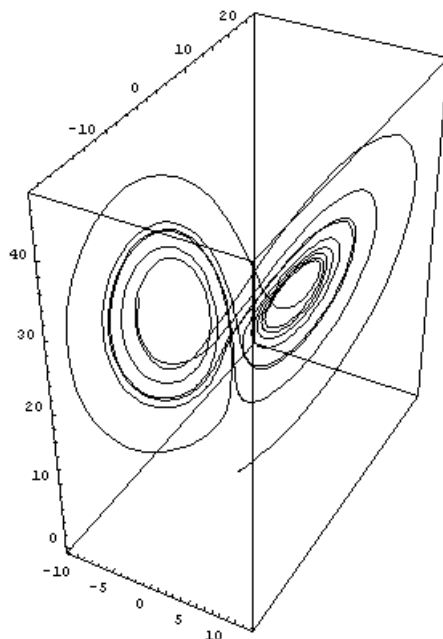


Рис. 5.26. Решение системы дифференциальных уравнений с выводом решения в форме кривых на фазовых плоскостях

списка с минимальным или максимальным значением, до поиска локальных и даже глобальных минимумов функций, заданных аналитически.

5.7.1. Поиск максимального и минимального чисел в списке

Для поиска максимального и минимального значений ряда чисел, входящих в список, система Mathematica предоставляет следующие средства:

- **Max[x1, x2, ...]** – возвращает наибольшее из x_i .
- **Max[{x1, x2, ...}, {y1, ...}, ...]** – возвращает наибольший элемент любого из списков.
- **Min[x1, x2, ...]** – возвращает наименьшее из x_i .
- **Min[{x1, x2, ...}, {y1, ...}, ...]** – возвращает наименьший элемент любого из данных списков.

Следующие примеры показывают действие этих простых функций:

Ввод (In)	Вывод (Out)
Max [1,5,2,6.5,3,4]	6.5
Max [[1,3,2],[4,5,6],[9,8,7]]	9
Min [1,5,2,6.5,-3,4]	-3
Min [[1,3,2],[4 5 6],[9,8,7]]	1

Данные функции выполняются очень быстро, поскольку алгоритм их работы основан просто на сравнении значений элементов списков.

5.7.2. Поиск локального минимума и максимума аналитической функции

Если нужен поиск локального минимума некоторой аналитической функции, используется функция:

- **FindMinimum**[f, {x, x0}] – выполняет поиск локального минимума функции одной переменной f, начиная со значения x=x0, и возвращает его значение.
- **FindMinimum**[f, {x, x0},{y, y0},...] – выполняет поиск локального минимума функции ряда переменных f, начиная со значений x=x0, y=y0 и т.д., и возвращает его значение.

Ниже представлены примеры применения функции **FindMinimum**:

```
FindMinimum[-x Exp[-2 x], {x, 1}]
{-0.18394, {x→0.5}}
```

```
FindMinimum[-x Exp[-2 x], {x, 0.2, 0, 1}]
{-0.18394, {x→0.5}}
```

```
FindMinimum[-5 x Exp[- $\frac{x}{2}$ ] (2 + Sin[3 x]), {x, 1}]
{-7.17833, {x→0.783139}}
```

```
FindMinimum[-5 x Exp[- $\frac{x}{2}$ ] (2 + Sin[3 x]), {x, 3}]
{-10.6299, {x→2.5805}}
```

```
FindMinimum[-5 x Exp[- $\frac{x}{2}$ ] (2 + Sin[3 x]), {x, 4}]
{-6.79134, {x→4.6179}}
```

```
FindMinimum[100 (y - x2)2 + (1 - x)2, {x, 0}, {y, 0},
AccuracyGoal → Automatic]
{0., {x→1., y→1.}}
```

Эти примеры показывают, что выбором начального значения x можно найти ряд минимумов функции f(x), разумеется, если таковые имеют место. Если необходимо разыскивать локальные максимумы, достаточно перед функцией поставить знак минус или умножить ее на -1. В последнем примере вычисляется минимум функции двух переменных Розенброка, которая часто используется как тестовая функция при анализе алгоритмов оптимизации.

5.7.3. Поиск глобального максимума и минимума аналитической функции

Уже в системах Mathematica 4.1/4.2 были следующие две функции для поиска глобального максимума и минимума аналитически заданной функции:

- **ConstrainedMax**[*f*, {*inequalities*}, {*x*, *y*, ...}] – ищет глобальный максимум функции *f* в области, определяемой неравенствами *inequalities*. Полагается, что все переменные *x*, *y*, ... неотрицательны.
- **ConstrainedMin**[*f*, {*inequalities*}, {*x*, *y*, ...}] – ищет глобальный минимум функции *f* в области, определяемой неравенствами *inequalities*. Полагается, что все переменные *x*, *y*, ... неотрицательны.

После имени функции указывается максимизируемая целевая функция, затем указываются все ограничения и, наконец, список искомых переменных. Результатом вычислений является стоимость продукции и количества изделий, представленные списком.

Рассмотрим типичный пример на линейное программирование. Пусть цех малого предприятия должен изготовить 100 изделий трех типов, причем каждое не менее 20 штук. На изготовление этих изделий уходит соответственно 4, 3.4 и 2 кг металла при его общем весе 700 кг. Спрашивается, сколько изделий *x*₁, *x*₂ и *x*₃ каждого типа надо выпустить для обеспечения максимальной стоимости продукции, если цена каждого из изделий равна соответственно 4, 3 и 2 рубля. Ниже представлено решение этой задачи с помощью функции **ConstrainedMax**:

```
ConstrainedMax[
    4*x1+3*x2+2*x3,
    {x1>=20,x2>=20,x3>=20,
    4*x1+3.4*x2+2*x3<=340,
    4.75*x1+11*x2+2*x3<=700,
    x1+x2+x3==100},
    {x1,x2,x3}]
{332.,{x1→56.,x2→20.,x3→24.}}
```

Две последние функции решают типовые задачи линейного программирования. В дополнение к ним может использоваться функция:

- **LinearProgramming**[*c*, *m*, *b*] – ищет вектор *x*, минимизирующий величину *c.x* в соответствии с условиями *m.x* >= *b* и *x* >= 0.
- **LinearProgramming**[*c*, *m*, *aaa*, *aa*, *aa*, *j*, *a*] – ищет вектор *x*, минимизирующий величину *c.x* в соответствии с линейными ограничениями, заданными в матрице *m* и парами *aa*, *aa*. Для каждой строки *m* задается соответствие условию ограничения в виде *a* if *a*, или *a* == *a*, если *a* == 0, или *a* если *a*.
- **LinearProgramming**[*c*, *m*, *b*, *l*] – минимизирует *c.x* с ограничениями, заданными в *m* и *b* и *a*.
- **LinearProgramming**[*c*, *m*, *b*, *aa*, *a*, *j*, *a*] – минимизирует *c.x* с ограничениями, заданными в *m* и *b* и *a*.
- **LinearProgramming**[*c*, *m*, *b*, *aaa*, *aa*, *aa*, *aa*, *j*, *a*] – минимизирует *c.x* с ограничениями, заданными в *m* и *b* и *a*.

Ниже даны примеры решения задачи линейного программирования функциями **ConstrainedMin** и **LinearProgramming**:

ConstrainedMin[$2x - 3y$, $\{x + y < 12, x - y > 1, x + 2y == 14, x > 1\}$,
 $\{x, y\}$]

$\{-\frac{7}{3}, \{x \rightarrow \frac{16}{3}, y \rightarrow \frac{13}{3}\}\}$

LinearProgramming[$\{2, -3\}$, $\begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 2 \\ 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 12 & -1 \\ 1 & 1 \\ 14 & 0 \\ 1 & 1 \end{pmatrix}$]

$\{\frac{16}{3}, \frac{13}{3}\}$

Задачи минимизации традиционно относятся к сложным задачам программирования, особенно при поиске глобального минимума. Наличие в ядре системы Mathematica их реализаций делает систему привлекательной для решения задач этого класса.

5.7.4. Функции оптимизации в Mathematica 5/5.1/5.2

В версиях Mathematica 5/5.1/5.2 средства оптимизации были существенно переработаны и дополнены. К функции **FindMinimum** добавлена новая функция **FindMaximum** с аналогичным **FindMinimum** синтаксисом, которая обеспечивает поиск локального максимума функции f одной или ряда переменных. Рекомендуется просмотреть по справке возможные опции этих функций.

В Mathematica 5/5.1/5.2 включены новые функции для символьного и численного поиска максимумов и минимумов функций ряда переменных:

- **Maximize**[f , $\{x, y, j\}$] – ищет x, y, j для максимума функции f .
- **Maximize**[$\{f, \text{cons}\}$, $\{x, y, j\}$] – ищет максимум функции f с ограничениями cons .
- **Minimize**[f , $\{x, y, j\}$] – ищет x, y, j для минимума функции f .
- **Minimize**[$\{f, \text{cons}\}$, $\{x, y, j\}$] – минимизирует функцию f с ограничениями cons .

Примеры применения этих функций представлены ниже:

Minimize[$\frac{\sqrt{5+x^2}}{2} + \frac{5-x}{4}$, x]

$\{\frac{1}{4} (5 + \sqrt{15}), \{x \rightarrow \sqrt{\frac{5}{3}}\}\}$

Minimize[$x^2 - y^3, x^4 + y^4 \leq 2$, $\{x, y\}$]

$\{\text{Root}[-8 + \#1^4 \&, 1], \{y \rightarrow \text{Root}[-2 + \#1^4 \&, 2], x \rightarrow 0\}\}$

```
Minimize[-2a+7b+c+9d,{6a-  
b+cJ12,a+5bJ3,a+5b+d==5,ai0,bi0,ci0,di0},{a,b,c,d}]
```

```
{474/31,{a->63/31,b->6/31,c->0,d->2}}
```

```
Minimize[ $e^x + \frac{1.2}{x}$ ,1<=x<=2,x]
```

```
{3.91828,{x->1.}}
```

```
Maximize[x*e-x,x]
```

```
{1/e,{x->1}}
```

```
Maximize[Sin[x],0<x<2,x]
```

```
{1,{x->pi/2}}
```

Для численного вычисления максимумов и минимумов служат также функции:

- **NMaximize**[f, {x, y, j}] – ищет x, y, j для максимума функции f.
- **NMaximize**[{f, cons}, {x, y, j}] – ищет максимум функции f с ограничениями cons.
- **NMinimize**[f, {x, y, j}] – ищет x, y, j для минимума функции f.
- **NMinimize**[{f, cons}, {x, y, j}] – минимизирует функцию f с ограничениями cons.

Ниже представлены примеры применения этих функций:

```
NMinimize[ $\frac{\sqrt{5+x^2}}{2} + \frac{5-x}{4}$ , x]//
```

```
{2.21825,{x->1.29099}}
```

```
NMinimize[{x2-y3,x4+y4<=2},{x,y}]
```

```
{-1.68179,{x->0.,y->1.18921}}
```

```
NMinimize[{-2a+7b+c+9d,6a-  
b+cJ12,a+5bJ3,a+5b+d==5,ai0,bi0,ci0,di0},{a,b,c,d}]
```

```
{15.2903,{a->2.03226,b->0.193548,c->0.,d->2.}}
```

```
NMinimize[ $e^x + \frac{1.2}{x}$ ,1<=x<=2,x]
```

```
{3.91828,{x->1.}}
```

```
NMaximize[x*e-x,x]
```

```
{0.367879,{x->1.}}
```

```
NMaximize[{Sin[x],0<x<2},x]
```

```
{1.,{x->1.5708}}
```

5.7.5. Функции оптимизации в Mathematica 6

Функции **ConstrainedMax** и **ConstrainedMax** в Mathematica 5/6 были удалены из списка функций оптимизации. Точнее говоря, они скрыты. Ниже представлено решение рассмотренной выше задачи на максимизацию выпуска продукции малым предприятием с помощью функции **ConstrainedMax** в системе Mathematica 6:


```
ConstrainedMax[
    4*x1+3*x2+2*x3,
    {x1>=20,x2>=20,x3>=20,
    4*x1+3.4*x2+2*x3<=340,
    4.75*x1+11*x2+2*x3<=700,
    x1+x2+x3==100},
    {x1,x2,x3}]
```

ConstrainedMax::deprec : ConstrainedMax is deprecated and will not be supported in future versions of Mathematica. Use NMaximize or Maximize instead.

```
{332., {x1→56., x2→20., x3→24.}}
```

В приведенном примере Mathematica 6 выдает сообщение о том, что функция **ConstrainedMax** отнесена к неперспективным, и в последующих версиях может не поддерживаться. Вместо нее рекомендуется применять функцию NMaximize или Maximize. Тем не менее, в Mathematica 5/5.1/5.2/6 функция ConstrainedMax все еще работает и решает представленную задачу верно.

Примеры решения данной задачи с помощью более предпочтительных для функций Maximize и NMaximize даны ниже:

```
Maximize[
    4*x1+3*x2+2*x3,
    {x1>=20,x2>=20,x3>=20,
    4*x1+3.4*x2+2*x3<=340,
    4.75*x1+11*x2+2*x3<=700,
    x1+x2+x3==100},
    {x1,x2,x3}]
```

```
{332., {x1→56., x2→20., x3→24.}}
```

```
NMaximize[
    {4*x1+3*x2+2*x3,
    x1>=20,x2>=20,x3>=20,
    4*x1+3.4*x2+2*x3<=340,
    4.75*x1+11*x2+2*x3<=700,
    x1+x2+x3==100},
    {x1,x2,x3}]
```

```
{332., {x1→56., x2→20., x3→24.}}
```

Большинство функций оптимизации в Mathematica 6 существенно переработаны, расширены возможности их применения. Например, функции FindMinimum и FindMaximum поддерживают нелинейную оптимизацию с ограничениями. Функции Minimize, Maximize и LinearProgramming теперь поддерживают решение задач целочисленного программирования.

5.7.6. Визуализация оптимизации в Mathematica 6

Mathematica 6 имеет расширенные средства визуализации вычислений, в том числе оптимизации. Рассмотреть их все в книге умеренного объема просто невозможно. Ограничимся парой примеров. Рис. 5.27 показывает применение функции

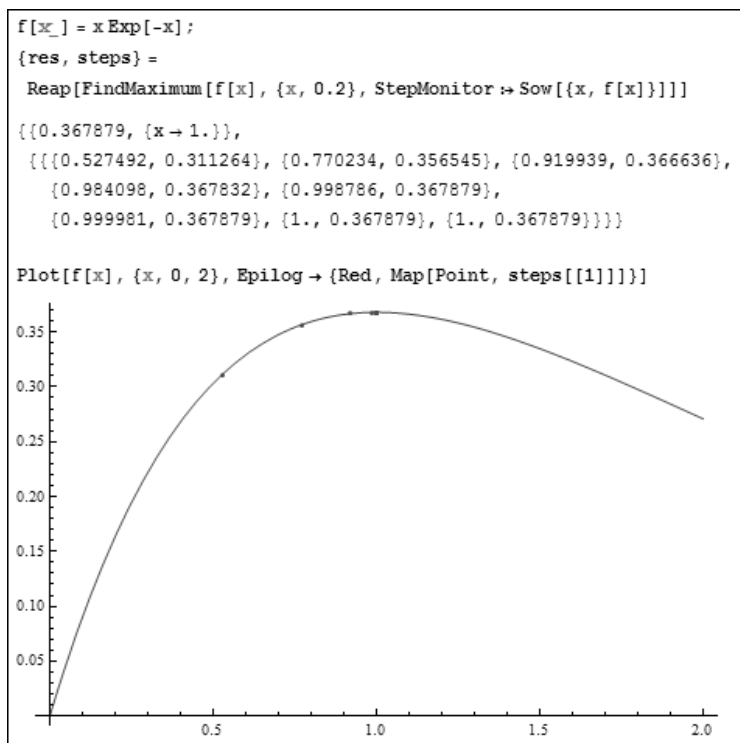


Рис. 5.27. Пример поиска максимума функции с графической визуализацией решения

FindMaximum для поиска максимума функции $x e^{-x}$ одной переменной с графической визуализацией решения путем построения графика функции и точек промежуточных и окончательного решения.

Другой пример (рис. 5.28) иллюстрирует поиск минимума функции двух переменных с графической иллюстрацией путем построения точек промежуточных и окончательного результатов

Многочисленные наглядные примеры применения этих функций можно найти в справке по ним.

5.8. Функции интегральных преобразований

К важному разделу математического анализа относятся *интегральные преобразования*. Большинство из них может выполняться уже описанными функциями интегрирования. Но для наиболее распространенных преобразований Лапласа, Фурье и z-преобразований в ядро Mathematica включены соответствующие функции.

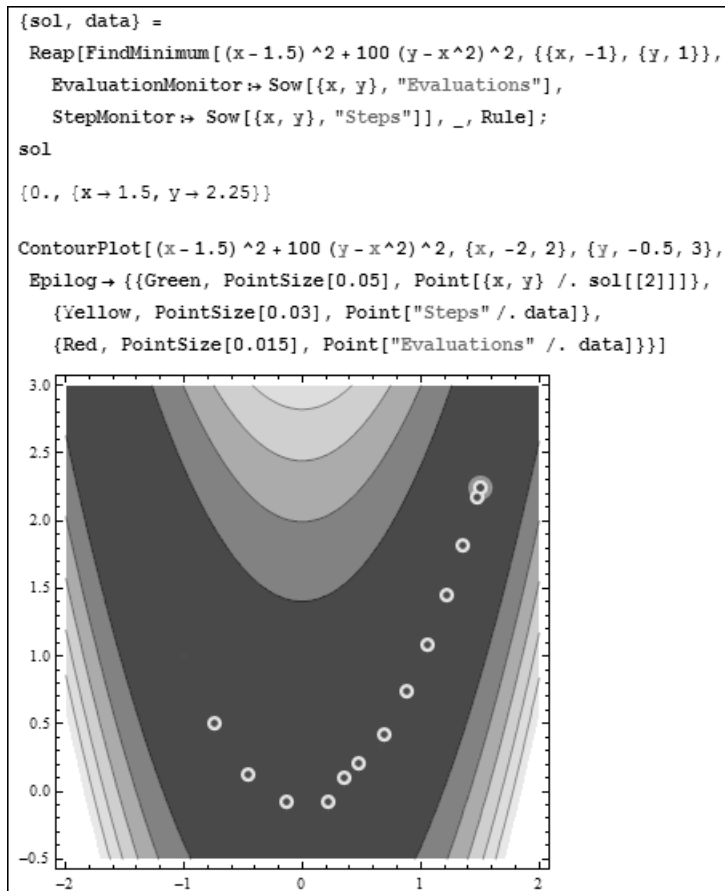


Рис. 5.28. Поиск минимума функции двух переменных с графической иллюстрацией

5.8.1. Функции преобразований Лапласа

Преобразования Лапласа – одни из самых часто применяемых интегральных преобразований. Они широко применяются в электро-радиотехнике, лежат в основе операторного метода и часто используются для решения линейных дифференциальных уравнений.

Прямое преобразование Лапласа заключается в переводе некоторой функции времени $f(t)$ в операторную форму $F(p)$. Это преобразование означает вычисление интеграла

$$F(p) = \int_0^{\infty} f(t) e^{-st} dt.$$

Обратное преобразование Лапласа означает переход от функции $F(p)$ к функции $f(t)$ с помощью формулы

$$F(t) = \int_{s-i\infty}^{s+i\infty} F(p) e^{-pt} dp.$$

Для проведения этих преобразований служат следующие функции:

- **LaplaceTransform[expr,t,s]** — возвращает результат прямого преобразования Лапласа для выражения $\text{expr}[t]$ в виде функции переменной s ;
- **InverseLaplaceTransform[expr,s,t]** — возвращает результат обратного преобразования Лапласа для выражения $\text{expr}[s]$ в виде функции переменной t ;
- **LaplaceTransform[expr,{t1,t2,...},{s1,s2,...}]** — возвращает результат прямого преобразования Лапласа для выражения $\text{expr}[t1,t2,...]$ в виде функции переменных $\{s1,s2,...\}$;
- **InverseLaplaceTransform[expr,{s1,s2,...},{t1,t2,...}]** — возвращает результат обратного преобразования Лапласа для выражения $\text{expr}[s1,s2,...]$ в виде функции переменных $\{t1,t2,...\}$.

Хотя имена переменных t и s можно выбирать произвольно, обычно t означает время, а s — оператор Лапласа. Ниже представлено несколько примеров выполнения преобразования Лапласа:

LaplaceTransform[a*t, t, s]

$$\frac{a}{s^2}$$

InverseLaplaceTransform[% , s, t]

$$a \, t$$

LaplaceTransform[a*Cos[b*t], t, s]

$$\frac{a \, s}{b^2 + s^2}$$

InverseLaplaceTransform[% , s, t]

$$a \, \text{Cos}[b \, t]$$

В этих примерах нет ничего необычного — после прямого и обратного преобразований Лапласа восстанавливается исходная функция. А теперь рассмотрим следующие примеры:

LaplaceTransform[a*Exp[-t]*Sin[t], t, s]

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

InverseLaplaceTransform[% , s, t]

$$-\frac{1}{2} i \, a \, e^{(-1-i) \, t} (-1 + e^{2 i \, t})$$

LaplaceTransform[t^2 Exp[-x], t, x, s, v]

$$\frac{2 \, e^{-x}}{x^3}$$

InverseLaplaceTransform[% , s, t]

$$\frac{2 \, e^{-x} \text{DiracDelta}[t]}{x^3}$$

Здесь уже прямое и обратное преобразования Лапласа в среде Mathematica не восстанавливают исходную функцию сразу в оригинале. В первой паре примеров вместо затухающей по экспоненте синусоиды получилось ее представление в комплексном виде через экспоненциальные функции. Во втором примере исходная функция восстановлена, но с поправкой – умножением на ступенчатую функцию Дирака. Это надо понимать так, что Mathematica 5 уточняет, что результат справедлив только для $t > 0$, а при $t < 0$ он будет нулевым.

Полезно проанализировать и следующие примеры:

```
LaplaceTransform[f[t], t, s]
LaplaceTransform[f[t], t, s]
InverseLaplaceTransform[%, s, t]
f[t]
LaplaceTransform[t^2 Exp[-x], {t, x}, {s, v}]
2
s^3 (1 + v)
InverseLaplaceTransform[%, {s, v}, {t, x}]
{ 2 DiracDelta[t] / (s^3 (1 + v)), 2 DiracDelta[x] / (s^3 (1 + v)) }
Options[LaplaceTransform]
{Assumptions!$Assumptions, GenerateConditions->False, PrincipalValue->False,
Analytic->True}
```

В последнем из них выведен список опций прямого преобразования Лапласа.

5.8.2. Функции Фурье-преобразований

Прямое преобразование Фурье преобразует функцию времени $f(t)$ в функцию частот и заключается в вычислении следующей интегральной функции:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt.$$

Обратное преобразование Фурье задается вычислением интеграла

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega.$$

Оно фактически переводит представление сигнала из частотной области во временную область. При необходимости в процессе этого преобразования можно учесть влияние на сигнал внешних цепей и устройств, умножив комплексный спектр сигнала на комплексный коэффициент передачи внешней цепи.

Системы Mathematica (и особенно Mathematica 5/6) имеют обширные возможности для аналитического проведения преобразований Фурье. Они реализуются следующими функциями:

- **FourierTransform[expr, t, ω]** – возвращает символьное значение прямого преобразования Фурье для выражения **expr**, зависящего от круговой частоты **ω** .

- **FourierTransform[expr, {t1, t2, j}, {ω1, ω2 j}]** – возвращает символьное значение многомерного прямого преобразования Фурье для выражения **expr**.
- **InverseFourierTransform[expr, ω, t]** – возвращает символьное значение обратного преобразования Фурье для выражения **expr**, зависящего от круговой частоты **ω**.
- **InverseFourierTransform[expr, {ω1, ω2 j}, {t1, t2, j}]** – возвращает символьное значение многомерного обратного преобразования Фурье для выражения **expr**.

Приведенные ниже примеры прямого и обратного преобразований Фурье к тривиальным не отнесешь:

```

FourierTransform[t, t, ω]
-i √(2 π) DiracDelta'[ω]
InverseFourierTransform[%, ω, t]
t
FourierTransform[t UnitStep[t], t, ω]
-1/2 i ⎛ -i √(2 π) / ω² + √(2 π) DiracDelta'[ω] ⎞
InverseFourierTransform[%, ω, t]
1/2 t (1 + Sign[t])
FourierTransform[Sin[t], t, ω]
i √(π/2) DiracDelta[-1 + ω] - i √(π/2) DiracDelta[1 + ω]
InverseFourierTransform[%, ω, t]
-1/2 i e^{-it} (-1 + e^{2it})
FullSimplify[%]
Sin[t]
FourierTransform[t^2 Cos[t], t, ω]
-√(π/2) DiracDelta''[-1 + ω] - √(π/2) DiracDelta''[1 + ω]
InverseFourierTransform[%, ω, t]
1/2 e^{-it} (1 + e^{2it}) t^2
FullSimplify[%]
t^2 Cos[t]

```

Уже для первого примера преобразование Фурье имеет смысл в классе обобщенных функций. Тем не менее, прямое и обратное преобразования восстанавливают исходную линейную функцию времени. В ряде приведенных примеров буквального восстановления исходной функции в исходной форме нет, и приходится применять функцию **FullSimplify** для обеспечения идентичности восстановленной и исходной функций.

Бывают случаи, когда после прямого и обратного преобразований Фурье результат получается в виде неявно эквивалентной исходной зависимости, которая так и не приводится к исходной функцией FullSimplify. На рис. 5.29 показан такой пример. Функция после прямого и обратного преобразований Фурье содержит разрывные Тзета-функции Хевисайда. Применение к ним FullSimplify приводит лишь к повторению результатов. Однако график исходной функции f и слегка смещенной функции g (после прямого и обратного преобразований Фурье) оказываются абсолютно идентичными.

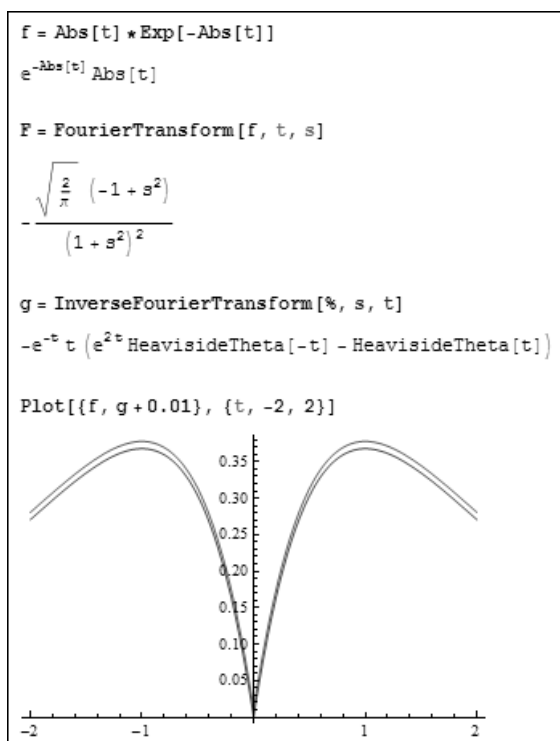


Рис. 5.29. Пример прямого и обратного преобразований Фурье с построением графиков

Интересно отметить, что новейшая версия системы Maple 11 вообще не обеспечивает даже прямого преобразования данной функции. Этот пример, как и ряд примеров, приведенных в [33], показывает, что Mathematica способна решать некоторые задачи, которые не решает конкурирующая система Maple. Впрочем, все-речь относиться к подобным доводам о преимуществах тех или иных систем не стоит – видимо, можно найти отдельные задачи, которые решает система Maple и не решает Mathematica.

5.8.3. Функции косинусного и синусного преобразований Фурье

Разложение функции $f(t)$ в ряд Фурье требует вычисления интегралов следующего вида:

$$F(\omega) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} f(t) \cos(\omega t) dt,$$

$$F(\omega) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} f(t) \sin(\omega t) dt.$$

Они получили название косинусного и синусного интегралов Фурье и фактически задают вычисление коэффициентов ряда Фурье, в который может быть разложена функция $f(t)$. Формулы соответствуют прямому косинусному и синусному преобразованиям Фурье, и для них используются следующие функции:

- **FourierSinTransform[expr, t, ω]** – возвращает символьное значение прямого синусного преобразования Фурье для выражения **expr**, зависящего от круговой частоты ω .
- **FourierSinTransform[expr, {t1, t2, j}, { ω 1, ω 2 j}]** – возвращает символьное значение многомерного прямого синусного преобразования Фурье для выражения **expr**.
- **FourierCosTransform[expr, t, ω]** – возвращает символьное значение прямого косинусного преобразования Фурье для выражения **expr**, зависящего от круговой частоты ω .
- **FourierCosTransform[expr, {t1, t2, j}, { ω 1, ω 2 j}]** – возвращает символьное значение многомерного прямого косинусного преобразования Фурье для выражения **expr**.

Соответственно обратное косинусное и синусное преобразования Фурье реализуются формулами:

$$f(t) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} F(\omega) \cdot \cos(\omega t) d\omega,$$

$$f(t) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} F(\omega) \cdot \sin(\omega t) d\omega.$$

- **InverseFourierSinTransform[expr, ω , t]** – возвращает символьное значение обратного синусного преобразования Фурье для выражения **expr**, зависящего от круговой частоты ω .
- **InverseFourierSinTransform[expr, { ω 1, ω 2 j}, {t1, t2, j}]** – возвращает символьное значение многомерного обратного синусного преобразования Фурье для выражения **expr**.

- **InverseFourierCosTransform[expr, ω, t]** – возвращает символьное значение обратного косинусного преобразования Фурье для выражения **expr**, зависящего от круговой частоты ω.
- **InverseFourierCosTransform[expr, {ω1, ω2 j }, {t1, t2, j }]** – возвращает символьное значение многомерного обратного косинусного преобразования Фурье для выражения **expr**.

Примеры на эти преобразования представлены ниже:

```

FourierSinTransform[t, t, ω]
-√2 π DiracDelta'[ω]
InverseFourierSinTransform[%, ω, t]
t
FourierSinTransform[t^2, t, ω]
2 √(2/π)
- ω^3
InverseFourierSinTransform[%, ω, t]
t^2 Sign[t]
FourierCosTransform[t, t, ω]
2
- √(2 π) ω^2
InverseFourierCosTransform[%, ω, t]
t Sign[t]

```

Данные преобразования требуют повышенного внимания, поскольку большинство пользователей не учитывают временные и частотные ограничения в ходе их выполнения. Это может приводить к «странным» ответам.

5.8.4. Функции z-преобразований

Прямое и обратное z-преобразования функций широко используются при решении задач автоматического управления и обработке дискретных сигналов. Прямое z-преобразование последовательности $f(n)$ в функцию комплексной переменной z задается выражением:

$$f(z) = \sum_{n=-\infty}^{\infty} f(n) \cdot z^{-n}.$$

Обратное z-преобразование сводится к преобразованию комплексной функции $f(z)$ в функцию $f(z)$.

Поэтому в системе Mathematica 4/5 для осуществления z-преобразований в ядро включены следующие функции:

- **ZTransform[expr, n, z]** – возвращает результат прямого z-преобразования для выражения **expr**, представленного как функция целочисленного аргумента **n**;

- **InverseZTransform[expr,n,z]** — возвращает результат обратного z-преобразования для выражения **expr**, представленного как функция целочисленного аргумента **n**.

В системе Mathematica 3 эти функции становятся доступными после исполнения команды

<<DiscreteMath`ZTransform`,

поскольку они входят не в ядро, а в пакет расширения дискретной математики.

Приведем примеры выполнения z-преобразований в системе Mathematica 5:

ZTransform[Cos[n], n, z]

$$-\frac{z(-1 - e^{2i} + 2e^i z)}{2(z + e^{2i}z - e^i(1 + z^2))}$$

InverseZTransform[%, z, n]

$$\frac{1}{2} e^{-in} (1 + e^{2in})$$

FullSimplify[%]

Cos[n]

ZTransform[n^2 a^n, n, z]

$$-\frac{a z (a + z)}{(a - z)^3}$$

InverseZTransform[%, z, n]

$$a^n n^2$$

Обратите внимание на то, что для первой функции (Cos[n]) результат получен через комплексные экспоненты, и лишь упрощение функцией FullSimplify позволило получить его без «фокусов».

Функции обработки данных, функций и сигналов

6.1. Разложение функций в степенные ряды	286
6.2. Средства синтеза сигналов	291
6.3. Функции полиномиальной интерполяции и аппроксимации	294
6.4. Регрессия и метод наименьших квадратов	315
6.5. Функции дискретного преобразования Фурье	323
6.6. Кусочные функции Piecewise	331
6.7. Новые средства Mathematica 6	333
6.8. Функции для работы со звуковыми сигналами	335
6.9. Функции для работы с потоками и файлами	341
6.10. Системные функции	346
6.11. Функции статистической обработки данных и массивов Statistics	350
6.12. Статистические вычисления в Mathematica 6	363

6.1. Разложение функций в степенные ряды

6.1.1. Разложения в ряды Тейлора и Маклорена

Еще одна из широко распространенных математических задач – разложение заданной аналитической функции относительно некоторой узловой точки с абсциссой x_0 в *степенной ряд Тейлора*:

Series[f[x], {x, 0, 6}]

$$f[0] + f'[0] x + \frac{1}{2} f''[0] x^2 + \frac{1}{6} f^{(3)}[0] x^3 + \frac{1}{24} f^{(4)}[0] x^4 + \frac{1}{120} f^{(5)}[0] x^5 + \frac{1}{720} f^{(6)}[0] x^6 + O[x]^7$$

Series[f[x], {x, x0, 6}]

$$f[x_0] + f'[x_0] (x - x_0) + \frac{1}{2} f''[x_0] (x - x_0)^2 + \frac{1}{6} f^{(3)}[x_0] (x - x_0)^3 + \frac{1}{24} f^{(4)}[x_0] (x - x_0)^4 + \frac{1}{120} f^{(5)}[x_0] (x - x_0)^5 + \frac{1}{720} f^{(6)}[x_0] (x - x_0)^6 + O[x - x_0]^7$$

Такой ряд нередко проще самой функции (в том смысле, что не требует вычисления даже элементарных функций, вычисляется с помощью только арифметических операций) и дает единообразное представление для разлагаемых в него различных функций в виде обычного степенного многочлена.

Для разложения в ряд используются следующие функции системы Mathematica:

- **Series[f, {x, x0, n}]** – выполняет разложение в степенной ряд функции f в окрестности точки $x=x_0$ по степеням $(x-x_0)^n$.
- **Series[f, {x, x0, nx}, {y, y0, ny}]** – последовательно ищет разложения в ряд по переменным y , затем x .
- **SeriesCoefficient[s, n]** – возвращает коэффициент при переменной n -й степени ряда s .
- **SeriesData[x, x0, {a0, a1, ...}, nmin, nmax, den]** – представляет степенной ряд от переменной x в окрестности точки x_0 . Величины a_i являются коэффициентами степенного ряда. Показатели степеней $(x-x_0)$ представлены величинами $nmin/den, (nmin+1)/den, ..., nmax/den$.

В данном случае Mathematica выдает формулы разложения, порядок производной указывается показателями степени в круглых скобках. Если разложение идет относительно исходной точки $x_0=0$, что соответствует упрощенному ряду Тейлора, часто называемому рядом Маклорена (первый пример). В ином случае разложение в ряд Тейлора идет относительно исходной точки с x_0 , отличным от нуля (второй пример). Обычно такое разложение сложнее и дает большую остаточную погрешность. Без особой на то необходимости его применять не стоит.

В соответствии с принятой математической символикой погрешность разложения в ряд указывается, как $O[x]^i$ с показателем степени, указывающим на порядок погрешности. Следует отметить, что разложение в ряд использует особый формат вывода, частью которого и является член остаточной погрешности.

6.1.2. Примеры разложения в ряды Тейлора и Маклорена

Рассмотрим разложение в ряд Тейлора ряда функций:

Series[Sin[x], {x, 0, 7}]

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + O[x]^8$$

Series[Cos[x], {x, 0, 7}]

$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + O[x]^8$$

Series[Exp[x], {x, 0, 7}]

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + O[x]^8$$

Series[Sinh[x], {x, 0, 7}]

$$x + \frac{x^3}{6} + \frac{x^5}{120} + \frac{x^7}{5040} + O[x]^8$$

Series[Log[x], {x, 0, 7}]

$$\text{Log}[x] + O[x]^8$$

Series[Log[x], {x, 1, 7}]

$$(x-1) - \frac{1}{2}(x-1)^2 + \frac{1}{3}(x-1)^3 - \frac{1}{4}(x-1)^4 + \frac{1}{5}(x-1)^5 - \frac{1}{6}(x-1)^6 + \frac{1}{7}(x-1)^7 + O[x-1]^8$$

Series[, {x, 0, 7}]

$$\sqrt{x} + O[x]^{15/2}$$

Series[, {x, 0, 7}]

$$1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} - \frac{5x^4}{128} + \frac{7x^5}{256} - \frac{21x^6}{1024} + \frac{33x^7}{2048} + O[x]^8$$

Series[Sin[x*y], {x, 0, 5}, {y, 0, 5}]

$$(y + O[y]^6)x + \left(-\frac{y^3}{6} + O[y]^6\right)x^3 + \left(\frac{y^5}{120} + O[y]^6\right)x^5 + O[x]^6$$

Series[x!, {x, 0, 3}]

$$1 - \text{EulerGamma} x + \frac{1}{2} \left(\text{EulerGamma}^2 + \frac{\pi^2}{6} \right) x^2 + \frac{1}{6} \left(-\text{EulerGamma}^3 - \frac{\text{EulerGamma} \pi^2}{2} + \text{PolyGamma}[2, 1] \right) x^3 + O[x]^4$$

Нетрудно заметить, что не все функции всегда разлагаются в ряд Тейлора системой Mathematica. Например, не имеют разложения относительно точки $x=0$ ло-

гарифм и квадратный корень – они возвращаются в исходном виде. А разложение факториала представлено через гамма- и полигамма-функции. Очень часто (это стоит взять на заметку) разложение получается с дробно-рациональными множителями у членов ряда.

6.1.3. Удаление члена с остаточной погрешностью ряда

Наличие члена с остаточной погрешностью указывает на то, что результат разложения в ряд использует особый формат выдачи данных. Его нельзя явно использовать для расчетов, например, для построения графика функции по данным ее разложения в ряд.

Для устранения остаточного члена и получения приемлемых для расчетов выражений можно использовать функции **Collect** и **Normal**. Ниже показаны примеры на использование этих функций:

```
Series[Sin[x], {x, 0, 7}]

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + O[x]^8$$

Collect[%, x]

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040}$$

Normal[Series[Sin[x*y], {x, 0, 3}, {y, 0, 3}]]

$$xy - \frac{x^3 y^3}{6}$$

f[0.1, 0.2]
0.0199987
```

В данном случае результат представлен в формате стандартного вывода. Его можно использовать для создания функций пользователя, например, путем переноса через буфер промежуточного хранения в правую часть такой функции. Это и показано в конце приведенных выше примеров. Разумеется, можно задать функцию пользователя и напрямую:

```
F[x_, y_] = Normal[Series[Sin[x*y], {x, 0, 3}, {y, 0, 3}]]

$$xy - \frac{x^3 y^3}{6}$$

F[0.1, 0.2]
0.0199987
```

6.1.4. Графическая визуализация разложения в ряд

Для оценки того, насколько разложение в ряд адекватно разлагаемой функции и в какой окрестности, полезно построение на одном рисунке графика исходной функции и графика выражения, дающего разложение в ряд (без остаточной по-

грешности). Другими словами, нужна графическая визуализация разложения в ряд (см. рис. 6.1).

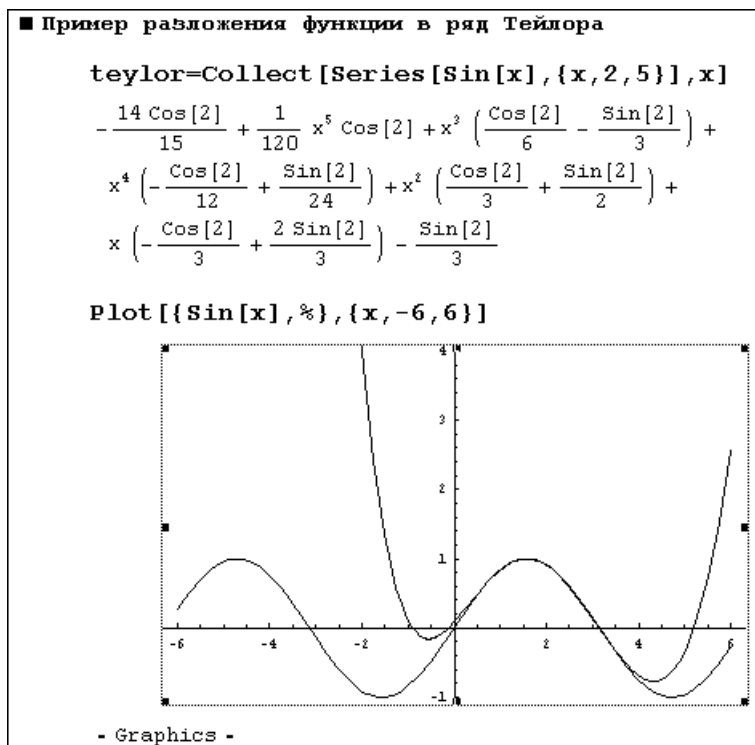


Рис. 6.1. Представление синусоидальной функции рядом Тейлора с графической иллюстрацией его точности

На рис. 6.1 представлены графики синусоиды, построенной по аналитическому выражению, и график разложения ее в ряд Тейлора в окрестности точки $x_0=2$. Хорошо заметно расхождение за пределами области, примыкающей к опорной точке функции. Как отмечалось, погрешность уменьшается, если $x_0=0$ (ряд Маклорена).

6.1.5. О разложении в ряд при большом числе членов

Существует довольно распространенное мнение о том, что удерживать в разложениях в ряд более 6-10 членов нерационально из-за резкого возрастания погрешности приближения функций рядом Маклорена или Тейлора. Это, в корне неверное мнение, базируется на том, что вычисления при разложении в ряд выполняются с числами в формате с плавающей точкой при 6-10 верных числах.

Многие системы компьютерной математики (включая Mathematica) способны вести вычисления с применением рациональных чисел в виде отношения целых чисел. При этом они используют аппарат точной арифметики, что сводит вычислительную погрешность к нулю. В результате погрешность разложения в ряд может быть очень малой и число членов ряда можно резко увеличить.

Рисунок 6.2 показывает разложение функции синуса относительно точки $x=0$ при удержании в разложении 32 членов ряда. Как показывает график синусоиды и график ее разложения в ряд, по крайней мере, 4 периода синусоиды вычисляются «на глаз» практически точно. На это указывает и график абсолютной погрешности приближения, представленный ниже. Этот график позволяет судить о погрешности куда более точно: из него видно, что при двух периодах синусоиды (по периоду слева и справа от точки $x=0$) погрешность действительно очень мала.

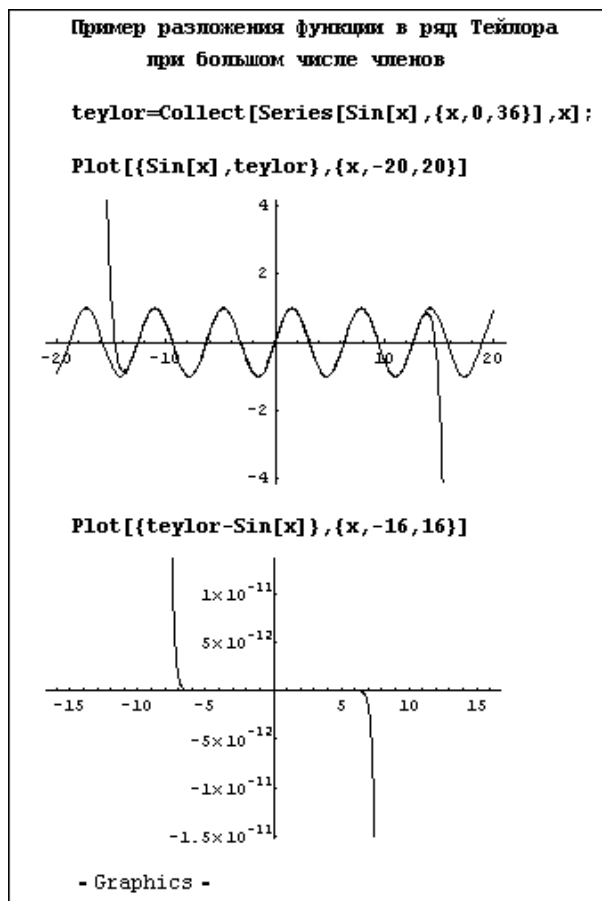


Рис. 6.2. Пример разложения функции синуса в ряд Маклорена при 32 членах ряда

Отмеченная возможность может дать «второе дыхание» методам приближения функций по их разложениям в ряд. Обратите внимание на то, что вывод длинного ряда на рис. 6.2 заблокирован точкой с запятой. Переменная-функция `teylor` обеспечивает вычисление функции синуса по ее разложению в ряд.

6.2. Средства синтеза сигналов

Под сигналами обычно подразумевают временные зависимости, параметры которых могут изменяться (модуляция сигналов) и которые благодаря этому способны переносить информацию. Остановимся на технике *синтеза* таких сигналов (зависимостей).

6.2.1. Синтез сигналов на основе встроенных функций

Довольно часто используют тестовые сигналы в виде синусоидальных функций или прямоугольных, треугольных, пилообразных и других импульсов. Создание синусоидальных сигналов средствами Mathematica тривиально, так что ограничимся парой примеров создания и синтеза импульсных сигналов.

Для задания таких сигналов можно использовать различные функции, например, `Sign[Sin[t]]` позволяет получить симметричные прямоугольные импульсы (меандр), а `Abs[Sin[t]]` моделирует результат двухполупериодного выпрямления синусоидального напряжения. Для получения разрывных сигналов можно использовать функции с условиями сравнения, например функцию `If` (на рис. 6.3 даны примеры имитации с помощью этой функции импульсов прямоугольной и пилообразной формы).

В подпакете `DiracDelta` еще в системе Mathematica 3 было задано определение двух полезных функций Дирака:

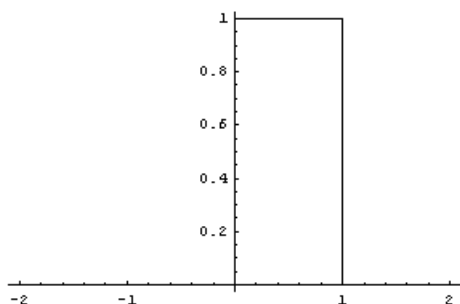
- `DiracDelta[x]` – возвращает дельта-функцию Дирака, которая является импульсом с единичной площадью, бесконечно малой шириной в точке $x=0$ и бесконечно большой амплитудой.
- `UnitStep[x]` – возвращает функцию Хевисайда с единичным скачком при $x=0$ (дает значение 0 при $x<0$ и 1 при $x\geq 0$).

Уже в Mathematica 4 эти широко распространенные функции вошли в ядро системы. Их действие поясняют следующие наглядные примеры:

```
Table[DiracDelta[x], {x, -3, 3}]
{0, 0, 0, DiracDelta[0], 0, 0, 0}
Integrate[DiracDelta[x], {x, -Γ, Γ}]
1
Integrate[f[x]*DiracDelta[x-a], {x, -Γ, Γ}]
f[a]
Table[UnitStep[x], {x, -3, 3}]
{0, 0, 0, 1, 1, 1, 1}
```

Примеры моделирования прямоугольного и пилообразного импульсов

```
f1[t_] := If[{0 > t} || {t > 1}, 0, 1]; Plot[f1[t], {t, -2, 2}]
```



```
f2[t_] := If[{0 > t} || {t > 1.5}, 0, t]; Plot[f2[t], {t, -2, 2}]
```

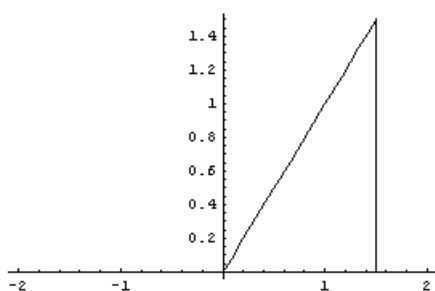


Рис. 6.3. Имитация импульсов прямоугольной и пилообразной формы с помощью функции *If*

Заметим, что прямоугольный импульс с длительностью a , начинающийся с момента $t=0$, легко задать как сумму **UnitStep[t]** и **-UnitStep[t-a]**.

6.2.2. Гармонический синтез сигналов

Для многих частных видов сигналов (а к ним относится большинство тестовых сигналов) разложения в *ряд Фурье* хорошо известны и приводятся в любом математическом справочнике (иногда в несколько разных формах). Это позволяет сразу получить нужное число гармоник сигнала и, что особенно важно, проверить, насколько адекватно синтезируемый сигнал описывает реальный сигнал.

На рис. 6.4 показан пример прямого синтеза разнополярных коротких прямоугольных импульсов. Используется известное разложение их в ряд, причем графики построены для 5 и 20 гармоник. Нетрудно заметить, что даже при двадцати гармониках представление такого сигнала гармоническим рядом не очень точно — отчетливо наблюдаются колебания и выбросы, то есть эффект Гиббса.

Еще один подобный пример — синтез разнополярных треугольных импульсов — представлен на рис. 6.5. Здесь также используется известное выражение

Гармонический синтез коротких двухполярных прямоугольных импульсов

```
sqr[k_, j_, N_, a_] :=  $\frac{4}{\pi}$  Sum[ $\frac{1}{i}$  * (Cos[i * a]) * Sin[2 * i * j *  $\pi$  / N], {i, 1, k, 2}]
```

```
sqr[10, j, N, a]
```

$$\frac{1}{\pi} \left(4 \left(\cos[a] \sin\left[\frac{2j\pi}{N}\right] + \frac{1}{3} \cos[3a] \sin\left[\frac{6j\pi}{N}\right] + \right. \right. \\ \left. \left. \frac{1}{5} \cos[5a] \sin\left[\frac{10j\pi}{N}\right] + \frac{1}{7} \cos[7a] \sin\left[\frac{14j\pi}{N}\right] + \frac{1}{9} \cos[9a] \sin\left[\frac{18j\pi}{N}\right] \right) \right)$$

```
Plot[{sqr[5, j, 200, 1], sqr[20, j, 200, 1]}, {j, 1, 400}, PlotStyle -> {Hue[.75], Hue[.3]},  
PlotRange -> {-1.2, 1.2}]
```

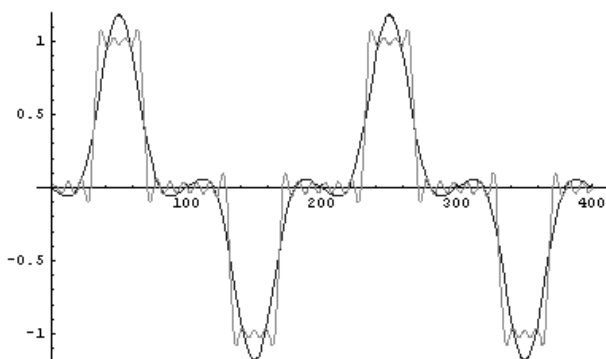


Рис. 6.4. Гармонический синтез коротких разнополярных прямоугольных импульсов

для ряда Фурье. Графики построены для 3 и 20 гармоник. Нетрудно заметить, что гармонический синтез для этого сигнала дает гораздо лучшие результаты — даже три гармоники неплохо воспроизводят сигнал. Это связано с тем, что данный сигнал не имеет разрывов — для него характерны лишь точки резких перегибов временной зависимости.

В целом надо отметить, что чем плавней временная зависимость сигнала, тем меньше проявляются отмеченные выше искажения и слабее заметен эффект Гиббса.

Итак, в результате гармонического анализа сигнала (или его прямого гармонического синтеза) сигнал получается в виде совокупности гармонических сигналов — гармоник. В общем случае каждая гармоника имеет свою амплитуду и фазу, и для их получения в общем случае можно использовать прямое преобразование Фурье (см. ниже). Ряд средств гармонического анализа и синтеза сигналов имеется во встроенном пакете Calculus.

Полученный спектр сигнала можно подвергать различным преобразованиям, например, частотной фильтрации. Полученный после этого измененный спектр гармоник используется (путем гармонического синтеза) для воссоздания искаженного (например, после фильтрации) сигнала.



Рис. 6.5. Гармонический синтез симметричных треугольных импульсов

Простота спектрального подхода обманчива, поскольку он требует довольно громоздких вычислений. Для быстрого их выполнения были созданы различные ускоренные методы спектрального анализа и синтеза, например, метод быстрого преобразования Фурье (БПФ). Но лишь с появлением СКМ класса Mathematica (и ей подобных) спектральный подход превращается в «рабочую лошадку», обеспечивая наглядное и достаточно быстрое решение задач спектрального анализа и синтеза (см. конец данной главы).

6.3. Функции полиномиальной интерполяции и аппроксимации

6.3.1. Функции полиномиальной интерполяции

Нередко исходные данные при решении математических задач представлены рядом точек произвольной зависимости $y(x)$ или функции $f(x)$. Сама по себе эта зависимость может быть неизвестной. Для вычисления промежуточных значений

функции используется аппарат *интерполяции*. При нем истинная зависимость заменяется аппроксимирующей функцией, которая в узловых точках дает точные значения ординат и позволяет вычислить значения интерполируемой функции и в промежуточных точках. Желательно, чтобы аппроксимирующая функция была достаточно простой и легко и быстро вычисляемой.

При *полиномиальной интерполяции (аппроксимации)* в качестве приближающей функции используется степенной многочлен – алгебраический полином степени n :

$$P_n(x) = \sum_{k=0}^n a_k x^{n-k}.$$

Полином – достаточно простая функция, способная представлять множество функций (как правило, без разрывов) единообразным способом. Это является привлекательным свойством полиномиальной интерполяции и аппроксимации.

Интерполяционный полином получим, потребовав совпадения его значений с узловыми точками исходной зависимости:

$$f(x_i) = P_n(x_i), \quad (i=0, 1, \dots, n).$$

Для нахождения этого полинома получаем систему алгебраических уравнений

$$a_0 x_i^n + a_1 x_i^{n-1} + \dots + a_n = f_i, \quad (i=0, 1, \dots, n),$$

и из ее решения можно найти коэффициенты полинома.

В Mathematica для интерполяции используются следующие функции:

- **InterpolatingFunction[range, table]** – возвращает объект – интерполирующую функцию, позволяющую вычислять промежуточные значения в заданном диапазоне **range** для таблицы **table**.
- **InterpolatingPolynomial[data, var]** – возвращает полином по переменной **var**, значения которого в узловых точках точно совпадают с данными из списка **data**. Он может иметь форму: $\{x_1, f_1\}, \{x_2, f_2\}, \dots$ или $\{f_1, f_2, \dots\}$, тогда во втором случае x_i принимают значения 1, 2, Вместо f_i может быть $\{f_i, df_i, ddf_i, \dots\}$, указывая производные в точках x_i .
- **Interpolation[data]** – конструирует объект **InterpolatingFunction**.
- **InterpolationOrder** – опция к **Interpolation**, которая указывает степень подходящего полинома.

Применение основной функции **Interpolation** поясняет следующий пример:

```
data=Table[{x,x^2+1},{x,1,5}]
{{1,2},{2,5},{3,10},{4,17},{5,26}}
funi=Interpolation[data]
InterpolatingFunction[{{1,5}},<>]
{funi[1.5],funi[3],funi[4.5]}
{3.25,10,21.25}
```

Таким образом, на заданном отрезке изменения x функция **Interpolation** позволяет найти любое промежуточное значение функции **funi[x]**, в том числе значения в узловых точках.

6.3.2. Пример полиномиальной аппроксимации

Теперь рассмотрим полиномиальную аппроксимацию, при которой полином ищется в явном виде и используется в качестве приближающей функции для данных, представленных списком координат узловых точек некоторой зависимости. Степень полинома в этом случае всегда на 1 меньше числа узловых точек интерполяции или аппроксимации. Пример на рис. 6.6 иллюстрирует технику проведения полиномиальной аппроксимации с применением интерполирующего степенного многочлена.

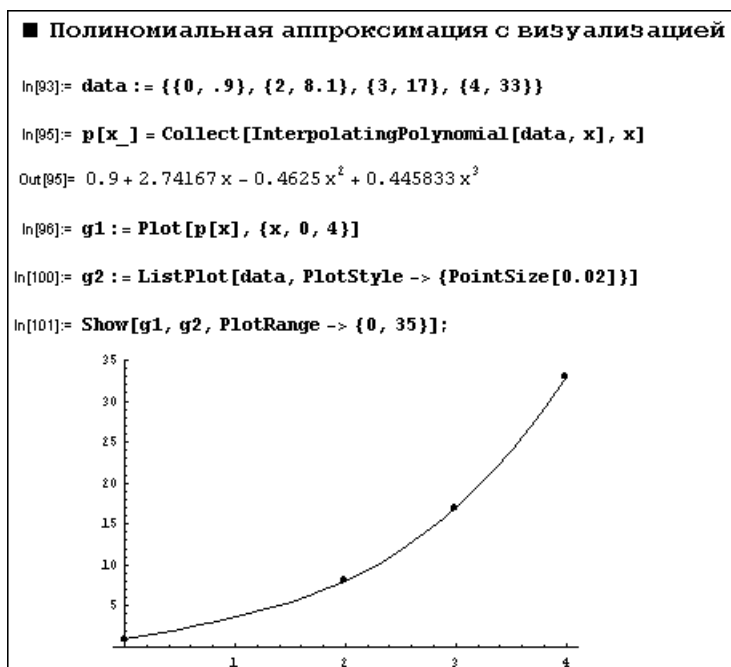


Рис. 6.6. Полиномиальная аппроксимация таблично заданных данных

Как и следовало ожидать, степень аппроксимирующего многочлена равна трем, поскольку задано всего четыре пары данных. На рис.6.6 представлено также сравнение данных полиномиальной аппроксимации с исходными данными. Исходные данные построены на графике в виде точек, а зависимость, представленная аппроксимирующим полиномом, построена на графике сплошной линией.

Полезно обратить внимание на технику построения графика аппроксимирующей функции одновременно с построением узловых точек. На рис. 6.6 для этого используются два графических объекта: g1 – график полинома, g2 – график узло-

вых точек, представленных списком `data`, и строящийся графической функцией **ListPlot** с опцией, задающей размеры точек. Функция **Show[g1, g2]** строит объекты `g1` и `g2` как порознь, так и совместно. Построения отдельно объектов из рис. 6.6 удалены и оставлен только график полинома вместе с узловыми точками. Этот прием мы будем в последующем использовать неоднократно.

6.3.3. Погрешность полиномиальной аппроксимации

При практическом применении полиномиальной аппроксимации особое значение имеет оценка ее погрешности. Теоретически погрешность полиномиальной аппроксимации определяется выражением:

$$|\varepsilon(x)| = |f(x) - P_n(x)| \leq \frac{f^{(n+1)}(x)}{(n+1)!} |\omega_n(x)|,$$

где $\omega_n(x) = \max_{[a,b]} \prod_{i=0}^n (x - x_i)$.

В узлах интерполяции значения интерполирующего многочлена точно совпадают со значениями исходных данных. Однако это не гарантирует малую погрешность за пределами узловых точек или, тем более, погрешность при экстраполяции. С одной стороны, эта погрешность быстро падает с ростом n , но при этом одновременно растут вычислительные погрешности, связанные с ограничением числа точных знаков результата.

Из-за вычислительных погрешностей обычно полиномиальная аппроксимация при степени многочлена выше 5-6 почти не применяется. И это вполне справедливо, когда число верных цифр при вычислениях составляет около 8–10. Однако, для системы *Mathematica*, имеющей средства точных вычислений, эта рекомендация ошибочна, и вполне возможна аппроксимация при n порядка многих десятков.

6.3.4. Полиномиальная аппроксимация специальных функций

Одним из полезных применений аппарата полиномиальной аппроксимации является приближение специальных математических функций. Такие функции системами компьютерной математики вычисляются по довольно сложным алгоритмам с применением интегральных их представлений или рядов. Эти алгоритмы сложны и требуют больших времен реализаций. Особенно сильно это проявляется при построении графиков специальных функций и выполнении с ними циклических вычислений.

При использовании СКМ, имеющих аппарат точной арифметики, применение полиномиальной аппроксимации позволяет резко (порой в десятки, а в отдельных случаях и в сотни раз) уменьшить время вычисления специальных функций при замене их полиномиальным приближением.

Рассмотрим полиномиальную аппроксимацию функции Бесселя, для чего вначале зададим саму функцию в интервале $[0,10]$, построим таблицу ее 11 значений и график функции.

```
f[x_]:=BesselJ[1,x]+1;
data=Table[{x,N[f[x]]},{x,0,10}];
g1:=Plot[f[x],{x,0,10}];g1
```

График функции представлен на рис. 6.7. Сама функция Бесселя колеблется относительно нуля, что заметно усложняет аппроксимацию. Поэтому в данном случае к ее значению прибавлен член 1 (его всегда можно вычесть из аппроксимирующего выражения и получить функцию, приближающую функцию Бесселя).

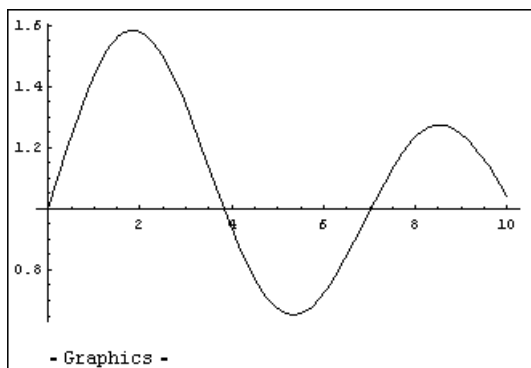


Рис. 6.7. График функции $f(x)$

Теперь выполним полиномиальную аппроксимацию функции $f(x)$ и построим (функцией Show) график исходных точек и график полинома:

```
g2:=ListPlot[data,PlotStyle->{PointSize[0.02]}]
p[x_]:=Collect[InterpolatingPolynomial[data,x],x]
1. + 0.495892x + 0.0128994x2 - 0.0794126x3 + 0.0123736x4 -
0.00303646x5 + 0.00167845x6 - 0.000382565x7 + 0.0000406462x8 -
2.08648x10-6x9 + 4.22174x10-8x10
g3:=Plot[p[x],{x,0,10}]
Show[g3,g2]
```

Как видно из рис. 6.8, график полинома степени 10 очень похож на график аппроксимируемой функции (рис. 6.7) и практически точно проходит через узловые точки.

Судить о точности аппроксимации по графику «на глаз», разумеется, не совсем верно. Гораздо полнее о точности аппроксимации можно судить по графику абсолютной погрешности, которую можно представить разностью $p(x)-f(x)$. Для построения такого графика (рис. 6.9) воспользуемся функцией Plot:

```
Plot[(p[x]-f[x]),{x,0,10},PlotRange->{-0.0008,0.0001}]
```

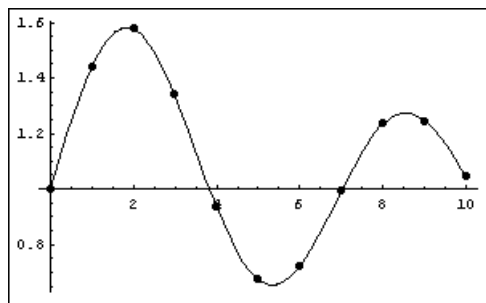



Рис. 6.8. График полинома и узловые точки

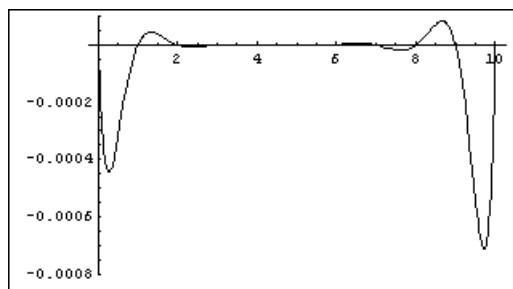


Рис. 6.9. График абсолютной погрешности при аппроксимации функции Бесселя полиномом десятой степени

Результат аппроксимации вполне удовлетворительный. Максимальная погрешность менее 0.001 (или 0,1%). Для ряда практических расчетов (но далеко не для всех) эта погрешность вполне приемлема. В то же время нетрудно заметить, что график погрешности имеет колебания и характерные пики у краев интервала $[0, 10]$ аппроксимации. При этом пики имеют разную амплитуду. Между пиками погрешность оказывается намного меньшей, и это указывает на целесообразность поиска способов ее «выравнивания».

6.3.5. Полиномиальная аппроксимация при большом числе узлов

Просто увеличение числа узлов и степени полинома приводит обычно к неутешительным результатам. Это хорошо видно из рис. 6.10, где показан график аппроксимирующего полинома при степени $n=40$ (41 точка функции $f(x)$, использованной в предшествующем примере).

Mathematica, однако, позволяет задавать вычисления при аппроксимации с достаточно большим числом цифр – порядка степени аппроксимирующего полино-

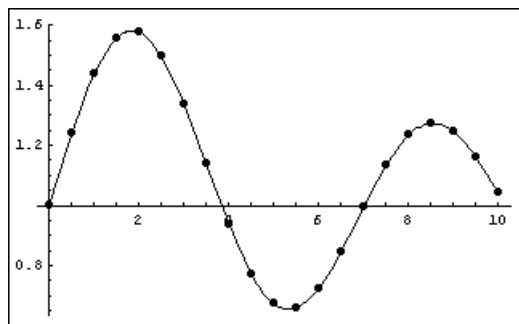


Рис. 6.11. График полинома двадцатой степени

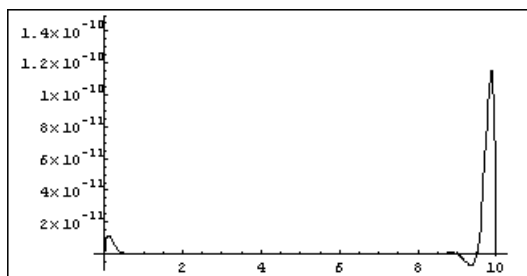


Рис. 6.12. График абсолютной погрешности при аппроксимации, смещенной на 1 функции Бесселя при степени полинома $n=20$

```
Plot[(p[x]-f[x]),{x,0,10},PlotRange->{-10^-11,
1.5*10^-10}]
```

Сравнив этот график с графиком, представленным на рис. 6.8, можно сделать вывод о том, что увеличение порядка полинома всего вдвое ведет к уменьшению погрешности примерно на 7 порядков!

6.3.6. Рациональная интерполяция и аппроксимация

Большую точность приближения, по сравнению с полиномиальным приближением, можно получить, если использовать *рациональную интерполяцию* и *аппроксимацию*. Наиболее важным свойством рациональных функций является то, что ими можно приближать такие зависимости, которые принимают бесконечные значения для конечных значений аргумента и даже внутри интервала его изменения.

При рациональной интерполяции (аппроксимации) приближение к $f(x)$ ищется в виде:

$$R(x) = \frac{a_0 + a_1x + \dots + a_px^p}{b_0 + b_1x + \dots + b_qx^q}, \text{ где } p+q+1=n.$$

Коэффициенты a_p, b_i находятся из совокупности соотношений $R(x_j)=f(x_j)$ ($j=1, \dots, n$), которые можно записать в виде

$$\sum_{j=0}^p a_j x_i^j - f(x_i) \sum_{j=0}^q b_j x_i^j = 0, \quad j=1, \dots, n.$$

Данное уравнение образует систему n линейных уравнений относительно $n+1$ неизвестных. Такая система всегда имеет нетривиальное решение.

Функция $R(x)$ может быть записана в явном виде в случае n нечетное, если $p=q$, и n четное, если $p-q=1$. Для записи функции $R(x)$ в явном виде следует вычислять так называемые обратные разделенные разности, определяемые условиями

$$f^-(x_l, x_k) = \frac{x_l - x_k}{f(x_l) - f(x_k)}$$

и рекуррентным соотношением

$$f^-(x_k, \dots, x_l) = \frac{x_l - x_k}{f^-(x_{k+1}, \dots, x_l) - f^-(x_k, \dots, x_{l-1})}.$$

Интерполирование функций рациональными выражениями обычно рассматривают на основе аппарата цепных дробей. Тогда интерполирующая рациональная функция записывается в виде цепной дроби

$$f(x) = f(x_1) + \frac{x - x_1}{f^-(x_1, x_2) + \frac{x - x_2}{f^-(x_1, x_2, x_3) + \dots + \frac{x - x_{n-1}}{f^-(x_1, \dots, x_n)}}}.$$

Использование рациональной интерполяции часто целесообразнее интерполяции полиномами в случае функций с резкими изменениями характера поведения или особенностями производных в точках.

Подпакет Approximations встроенного пакета расширения NumericalMath содержит ряд функций для рациональной аппроксимации аналитических функций. Для рациональной интерполяции и аппроксимации функций по заданным значениям абсцисс служит функция:

RationalInterpolation[f, {x, m, k}, {x₁, x₂, ..., x_{m+k+1}}] – возвращает аппроксимирующее функцию f выражение в виде отношения полиномов со степенью полинома числителя m и знаменателя k в абсциссах, заданных списком $\{x_1, x_2, \dots, x_{m+k+1}\}$.

Пример на применение этой функции для аппроксимации зависимости $\sin(x)/x+1$ представлен на рис. 6.13. Обратите внимание на первую строку этого примера: в ней задана загрузка подпакета Approximations из встроенного пакета расширения NumericalMath.

Нетрудно заметить, что кривая погрешности явно осциллирует, и погрешность оказывается максимальной вблизи краев интервала аппроксимации. Не-

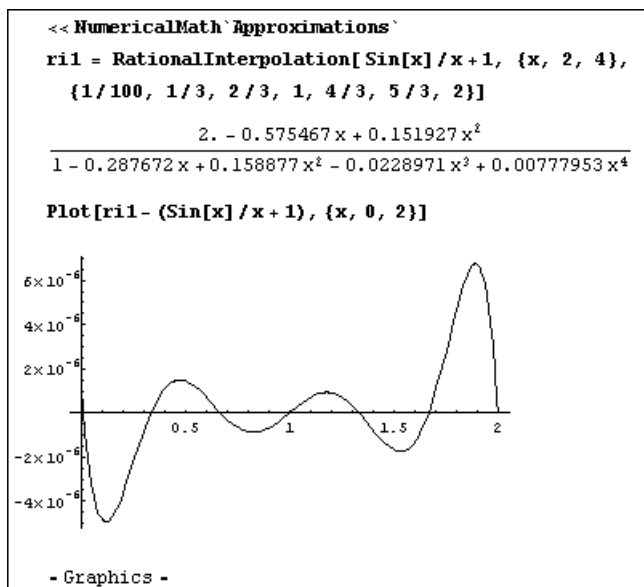


Рис. 6.13. Пример рациональной аппроксимации зависимости $\sin(x)/x+1$

смотря на малое число явно заданных узловых точек (их 7, и точка при $x=0$ с особенностью не используется), погрешность аппроксимации мала и не превышает $8 \cdot 10^{-6}$.

Представленная функция может использоваться и в иной форме:

RationalInterpolation[f,{ x, m, k},{ x, xmin, xmax}]

В данном случае выбор абсцисс осуществляется автоматически в интервале от x_{\min} до x_{\max} . В отличие от первого случая, когда абсциссы могли быть расположены неравномерно, в данном случае расположение их будет равномерным. На рис. 6.14 приведен пример аппроксимации функции синуса в интервале от $-\pi$ до π .

При рациональной аппроксимации можно задать опции **WorkingPrecision** и **Bias** со значениями по умолчанию **\$MachinePrecision** и 0 соответственно. Опция **Bias** обеспечивает автоматическую расстановку узлов интерполяции. При удачном подборе значения **Bias** обеспечивается симметрирование выбросов погрешности, дающее наименьшее ее значение в пиках. На рис. 6.15 приведен пример интерполяции (аппроксимации) экспоненциальной функции в интервале изменения x от 0 до 2. Погрешность аппроксимации в данном случае меньше 10^{-6} .

Из приведенных примеров видна высокая эффективность рациональной аппроксимации даже при умеренной степени полиномов числителя и знаменателя. Имеющиеся в системе Mathematica средства позволяют до предела упростить проведение такой аппроксимации и легко оценивать области ее применения и порядок получаемой погрешности.

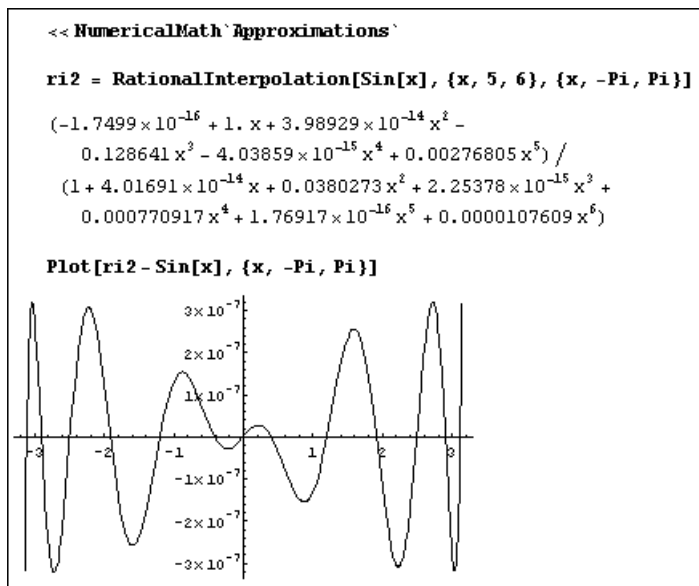


Рис. 6.14. Пример рациональной аппроксимации синусоидальной функции

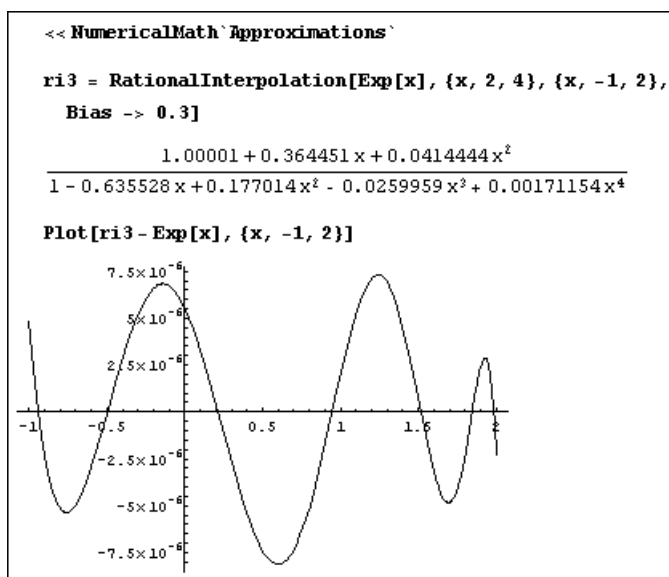


Рис. 6.15. Пример аппроксимации экспоненты при выборе опции Bias->0.3

6.3.7. Функции рациональной Паде-аппроксимация

Рациональная Паде-аппроксимация основана на применении в качестве аппроксимирующей функции рациональной функции в виде отношения двух полиномов, коэффициенты которых определяются коэффициентами разложения исходной зависимости в ряд Тейлора. Она часто используется для аппроксимации аналитических функций.

Пусть имеется степенной ряд

$$f(x) = \sum_{i=0}^{\infty} c_i x^i,$$

который является исходным для использования аппроксимации Паде.

Паде-аппроксимация представляет собой рациональную функцию вида

$$[L/M] = \frac{a_0 + a_1 x + \dots + a_L x^L}{b_0 + b_1 x + \dots + b_M x^M},$$

разложение которой в ряд Тейлора с центром в нуле совпадает с представленным выше рядом. Функция $[L/M]$ имеет $L+1$ коэффициентов в числителе и $M+1$ в знаменателе. Положим, что $b_0 = 1$. Теперь имеем $L+1$ свободных коэффициентов в числителе и M в знаменателе в соотношении (4.2). Всего $L+M+1$ свободных параметров. Это означает, что в общем случае коэффициенты разложения Тейлора функции $[L/M]$ при степенях $1, x, x^2, \dots, x^{L+M}$ должны совпадать с соответствующими коэффициентами ряда. Другими словами, должно выполняться соотношение

$$\sum_{i=0}^{\infty} c_i x^i = \frac{a_0 + a_1 x + \dots + a_L x^L}{b_0 + b_1 x + \dots + b_M x^M} + O(x^{L+M+1}).$$

Существует ряд достаточно эффективных алгоритмов для осуществления аппроксимации Паде, например Кронекера, Бейкера и Ватсона и др. Обобщением аппроксимации Паде может служить аппроксимация Паде – Чебышева, которая часто реализуется в системах компьютерной математики на ряду с аппроксимацией Паде. В системе Mathematica это функция Economized Rational Approximation.

В подпакете Pade встроенного пакета расширения Calculus определены две функции для рациональной аппроксимации Паде:

- **Pade[f,{x,x0,m,k}]** – возвращает выражение для аппроксимации Паде функции $f(x)$ в окрестностях точки x_0 в виде отношения двух полиномов степеней m и k .
- **EconomizedRationalApproximation[f,{x,{xmin,xmax}},m,k]** – возвращает выражение для осуществления *экономичной рациональной аппроксимации* функции $f(x)$ в интервале $\{xmin, xmax\}$ в виде отношения двух полиномов степеней m и k .

На рис. 6.16 представлен пример на аппроксимацию Паде функции синуса с построением графика исходной функции (темная линия) и аппроксимирующей функции (более светлая линия).

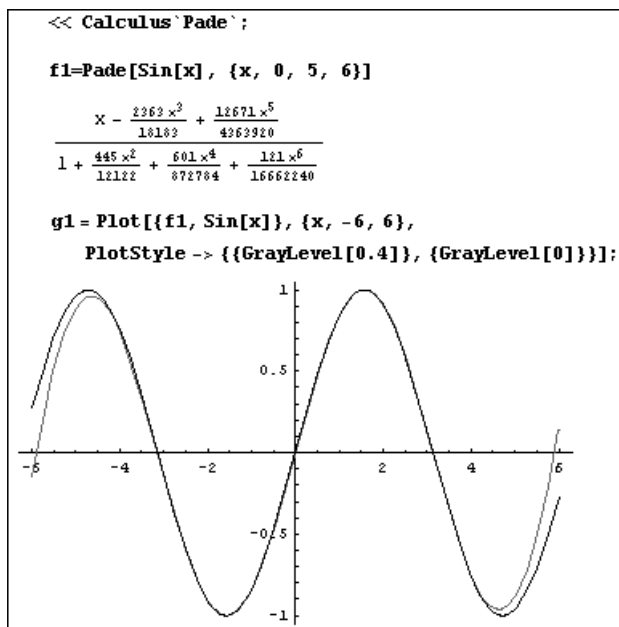


Рис. 6.16. Пример осуществления аппроксимации Паде

Зависимость относительной погрешности от значения независимой переменной x представлена на рис. 6.17.

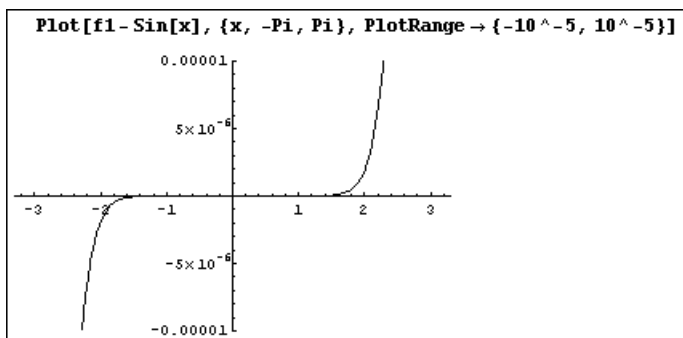


Рис. 6.17. График погрешности при аппроксимации синусоидальной функции

Пример осуществления экономичной рациональной аппроксимации для синусоидальной зависимости показан на рис. 6.18. Здесь также дана графическая визуализация аппроксимации в виде наложенных друг на друга исходной и аппрокси-

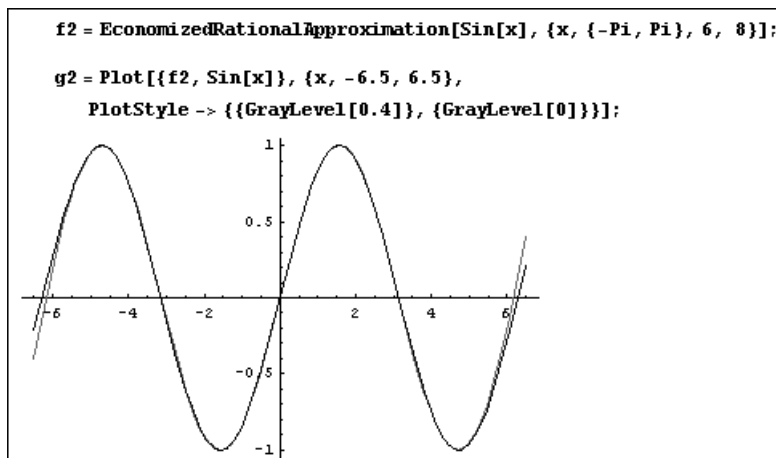


Рис. 6.18. Пример осуществления экономичной рациональной аппроксимации

мирующей функций. Нетрудно заметить, что область видимого на глаз совпадения исходной и аппроксимирующей функций заметно расширилась.

Экономичная рациональная аппроксимация обычно позволяет получить приемлемую погрешность при меньшей степени полиномов числителя и знаменателя аппроксимирующей функции. В ограниченной области $\{x_{\min}, x_{\max}\}$ эта аппроксимация нередко позволяет получить погрешность менее сотых долей процента (см. рис. 6.19). Здесь показан график погрешности в виде разности между значениями аппроксимирующей и аппроксимируемой функций.

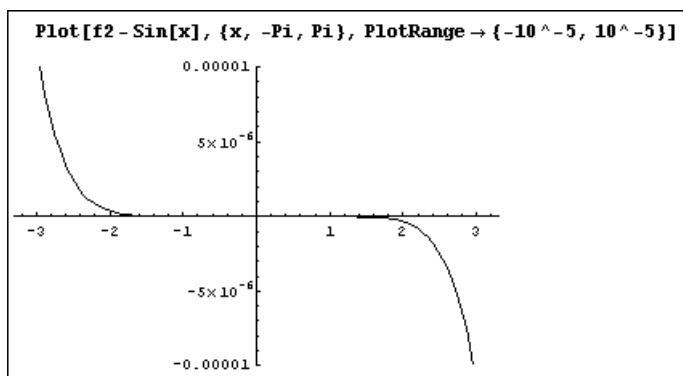


Рис. 6.19. Пример осуществления экономичной рациональной аппроксимации с построением графика погрешности

6.3.8. Оптимизация аппроксимации

Оптимизация аппроксимации обычно осуществляется по двум критериям: уменьшению временных затрат при вычислениях по аппроксимирующему выражению и подбору расположения узлов для обеспечения минимальной погрешности.

Для оптимизации временных затрат при аппроксимации возможно преобразование аппроксимирующих выражений в форму, называемую *схемой Горнера*. Это дает ряд преимуществ перед обычным вычислением полиномов: уменьшается время вычислений, повышается точность вычислений, уменьшается вероятность расхождения численных методов, в которых используются полиномы.

Схема Горнера реализуется следующим выражением:

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-2} + x(a_{n-1} + xa_n)) \dots)).$$

Согласно ему, вычисление $P_n(x)$ сводится к последовательному нахождению следующих величин:

$$b_n = a_n;$$

$$b_{n-1} = a_{n-1} + ab_n;$$

$$b_{n-2} = a_{n-2} + ab_{n-1};$$

...;

$$b_1 = a_1 + ab_2;$$

$$b_0 = a_0 + ab_1 = P_n(a).$$

Схема Горнера реализуется с помощью n умножений и n сложений, то есть за $2n$ арифметических действий. В общем случае не существует способа вычисления алгебраического полинома n -й степени менее чем за $2n$ арифметических действий.

Схема Горнера удобна для программной реализации, благодаря цикличности вычислений и необходимости сохранять кроме коэффициентов полинома и значения аргумента только одной промежуточной величины, а именно b_i или ab_i при текущем значении $i=n, n-1, \dots, 0$.

Следует отметить, что при вычислении полиномов по схеме Горнера с большими коэффициентами может произойти значительная потеря точности за счет вычитания больших округленных чисел. Избежать потери точности иногда удастся применением рекуррентных формул.

Подпакет Horner из встроенного пакета расширения Algebra в системе Mathematica 4/5 (В Mathematica 3 подпакет входил в пакет расширения NumericalMath) реализует схему Горнера для полиномиальных выражений. Для этого в нем есть функция:

- **Horner[poly]** – устанавливает полином poly в форму Горнера;
- **Horner[poly, vars]** – устанавливает полином ряда переменных vars в форму Горнера.

Примеры преобразования полиномов в схему Горнера:

```
<< Algebra`Horner`
Horner[ a x^3 + b x^2 + c x + d, x ]
  (d + x(c + x(b + ax)))
Horner[ x^(1/3) + x + x^(3/2) ]
(1 + (1 + sqrt(x)) x^(2/3)) x^(1/3)
```

Схема Горнера может использоваться и для отношения полиномов:

Horner[poly1/poly2] и **Horner[poly1/poly2,vars1,vars2]**

Эти функции можно использовать для улучшенного представления аппроксимации Паде, что демонстрирует следующий пример:

```
<< Calculus`Pade`
approx = Pade[ Exp[Log[x]-x], x, 0, 3, 2]
```

$$\frac{x - \frac{x^2}{2} + \frac{x^3}{12}}{1 + \frac{x}{2} + \frac{x^2}{12}}$$

```
Horner[ approx ]
x (1 + (-1/2 + x/12) x)
  1 + (1/2 + x/12) x
```

Переход к схеме Горнера дает ряд преимуществ перед обычным вычислением полиномов: уменьшается время вычислений, повышается точность вычислений, уменьшается вероятность расхождения численных методов, в которых используются полиномы.

Нередко оптимальным расположением узлов является такое расположение, которое соответствует *чебышевской интерполяции*. При этом координаты узлов задаются, как корни полинома Чебышева $\bar{T}_{n+1}^{[a,b]}(x)$:

$$x_k = \frac{b+a}{2} + \frac{b-a}{2} \cos\left(\frac{\pi(2k+1)}{2n+1}\right), \quad k=0, \dots, n.$$

Для уменьшения погрешности необходимо минимизировать величину $\max_{[a,b]} |\omega_n(x)|$, которая является полиномом $(n+1)$ -степени со старшим коэффициентом 1. В этом случае

$$\max_{[a,b]} |\omega_n(x)| = \max_{[a,b]} |\bar{T}_{n+1}^{[a,b]}(x)| = 2^{-n} \left(\frac{b-a}{2}\right)^{n+1}.$$

Тогда оценка погрешности

$$\varepsilon \leq \frac{f^{(n+1)}(x)}{(n+1)!} 2^{-n} \left(\frac{b-a}{2}\right)^{n+1}.$$

Эта оценка является наилучшей, так как здесь знак \int меняется на знак $=$, если в качестве $f(x)$ выбрать следующий полином степени $n+1$:

$$f(x) = \frac{f^{(n+1)}(x)}{(n+1)!} x^{n+1} + a_n x^n + \dots$$

в качестве узлов – точки x_k .

Сама по себе чебышевская аппроксимация в системе Mathematica встроенными функциями не задается. Но она используется в более сложных видах минимаксной аппроксимации, описанной ниже.

6.3.9. Методика минимаксной аппроксимации

Из приведенных примеров ясно, что рациональная аппроксимация способна дать существенное уменьшение погрешности при некотором оптимальном расположении узлов аппроксимации и выравнивании относительных погрешностей по абсолютной величине и точкам минимумов и максимумов кривой погрешности. Это лежит в основе так называемой минимаксной аппроксимации.

Пусть имеется функция $y=f(x)$, заданная на отрезке $[a,b]$ значениями y_1, \dots, y_N в точках x_1, \dots, x_N , а также имеется некоторая система непрерывных функций $L=\{\varphi_i(x)\}$, которые линейно независимы на $[a,b]$. Пусть приближение для $f(x)$ ищется в виде линейной комбинации

$$\varphi(x) = \sum_{k=0}^n c_k \varphi_k(x),$$

где c_k – неизвестные коэффициенты. Значения аппроксимирующей функции $\varphi(x)$ должны равняться значениям y_i функции $f(x)$, или быть как можно более близким к ним. Поэтому поиск неизвестных коэффициентов аппроксимирующей функции можно осуществлять путем решения системы линейных алгебраических уравнений

$$\sum c_k \varphi_k(x_i) = y_i, i = 1, \dots, N.$$

При $N > (n+1)$ эта система является несовместной. В этом случае в качестве решения системы можно взять вектор $c=(c_0, c_1, \dots, c_n)$, сводящий к минимуму функцию

$$\Phi(c_0, c_1, \dots, c_n) = \max_{1 \leq i \leq N} \left| \sum_{k=0}^n c_k \varphi_k(x_i) - y_i \right|.$$

Если выбирать коэффициенты аппроксимирующей функции из условия минимума данной функции, то критерий оценивания параметров называется минимаксным, а соответствующая ему задача аппроксимации – минимаксной, или задачей чебышевского приближения.

Заметим, что минимаксный критерий впервые был рассмотрен П. Л. Чебышевым во второй половине XIX века. При решении минимаксных задач часто используется класс алгоритмов Ремеза и Вале-Пуссена в сочетании с итерационным алгоритмом Ремеза. В этих алгоритмах используются множества точек, удовлетворяющих условиям чебышевского альтернанса.

В подпакете `Approximations` встроенного пакета расширения `NumericalMath` минимаксная аппроксимация реализуется следующей функцией:

- **MiniMaxApproximation** [*f*, {*x*, {*xmin*, *xmax*}, *m*, *k*}] – возвращает рациональную функцию аппроксимации *f* при степени полиномов числителя и знаменателя {*m*, *k*} и в интервале изменения *x* от *xmin* до *xmax*.
- **MiniMaxApproximation** [*f*, *approx*, {*x*, {*xmin*, *xmax*}, *m*, *k*}] – возвращает рациональную функцию аппроксимации *f* при степени полиномов числителя и знаменателя {*m*, *k*} и в интервале изменения *x* от *xmin* до *xmax* с возможностью выбора метода аппроксимации *approx*.

Эта аппроксимация использует интерактивный алгоритм вычислений. Они начинаются с первого шага, на котором используется функция **RationalInterpolation**. Затем аппроксимация последовательно улучшается применением алгоритма Ремеза, лежащего в основе этого вида аппроксимации.

Функция **MiniMaxApproximation** возвращает два списка: первый с координатами абсцисс, при которых наблюдается максимальная погрешность, и второй – рациональную функцию аппроксимации. На рис. 6.20 представлен пример на аппроксимацию функции $\exp(x^2)$. Там же построен график погрешности аппроксимации.

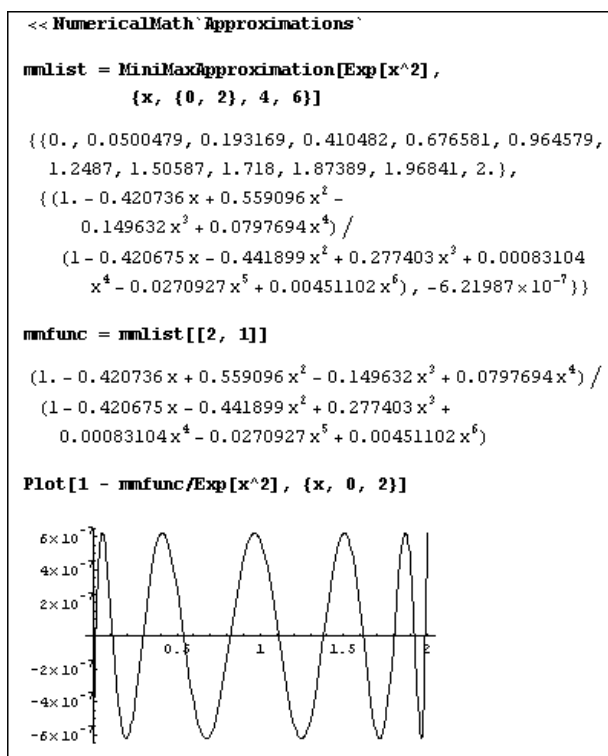


Рис. 6.20. Минимаксная аппроксимация функции $\exp(x^2)$ и график погрешности при ней

График погрешности минимаксной аппроксимации наглядно показывает ее главное свойство: кривая погрешности при наличии колебаний обеспечивает равенство амплитуд колебаний. Погрешность в последнем примере не превышает $6 \cdot 10^{-7}$, что свидетельствует о высокой и в ряде случаев даже избыточной точности.

Следует отметить, что малость абсолютной ошибки для ряда функций (например, тригонометрических) может приводить к большим относительным погрешностям в точках, где функции имеют нулевые значения. Это может привести к отказу от выполнения аппроксимации вследствие исчерпания числа итераций (опция `MaxIteration` 20). Такой случай наблюдается, например, при исполнении такой команды:

```
MiniMaxApproximation[Cos[x], {x, {1, 2}, 2, 4}]
```

Делением функции на $(x - \pi/2)$ можно исключить эту ситуацию и получить нужные приближения (рис. 6.21).

```
<< NumericalMath`Approximations`

MiniMaxApproximation[Cos[x], {x, {1, 2}, 2, 4}]

MiniMaxApproximation::van :
Failed to locate the extrema in 20 iterations. The function
Cos[x] may be vanishing on the interval {1, 2} or the
WorkingPrecision may be insufficient to get convergence.

MiniMaxApproximation[Cos[x], {x, {1, 2}, 2, 4}]

MiniMaxApproximation[Cos[x] / (x - Pi / 2),
  {x, {1, 2}, 2, 4}][[2, 1]]

      -0.636483 - 0.284196 x + 0.0904596 x2
-----
1 - 0.191361 x + 0.0771022 x2 - 0.0103059 x3 + 0.00164047 x4

mmacos = %N[x - Pi / 2]

      (-1.5708 + x) (-0.636483 - 0.284196 x + 0.0904596 x2)
-----
1 - 0.191361 x + 0.0771022 x2 - 0.0103059 x3 + 0.00164047 x4
```

Рис. 6.21. Пример минимаксной аппроксимации функции косинуса

График относительной погрешности для этого примера представлен на рис. 6.22. Обратите внимание на то, что в этом примере погрешность аппроксимации не превышает $7 \cdot 10^{-10}$.

В заключение отметим, что данный пакет имеет функции для *общей рациональной интерполяции*, с помощью которой можно аппроксимировать функции, заданные параметрически:

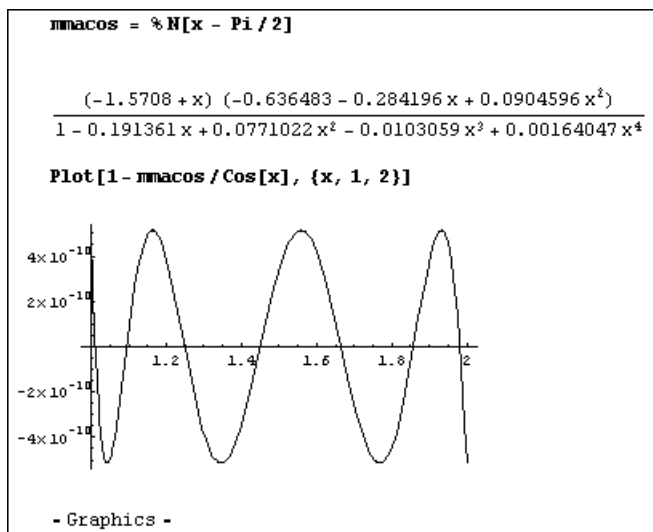


Рис. 6.22. График погрешности
минимаксной аппроксимации косинуса

- **GeneralRationalInterpolation**[{f_x,f_y},{t,m,k},{t₁,t₂,...,t_{n+k+1}}] – дает рациональную интерполяцию для параметрически заданных функций для списка значений параметра t.
- **GeneralRationalInterpolation**[{f_x,f_y},{t,m,k},{t,tmin,tmax}] – дает рациональную интерполяцию для параметрически заданных функций при автоматическом выборе значений параметра t.

С помощью других функций можно осуществить также *общую минимаксную интерполяцию*, обычно обеспечивающую минимальную погрешность:

- **GeneralMiniMaxInterpolation**[{f_x,f_y},{t,(tmin,tmax),m,k},x] – дает рациональную минимаксную интерполяцию для параметрически заданных функций с параметром t.
- **GeneralMiniMaxInterpolation**[{f_x,f_y},approx,{t,(tmin,tmax),m,k},x] – дает рациональную минимаксную интерполяцию для параметрически заданных функций для списка значений параметра t с указанием метода аппроксимации approx.
- **GeneralMiniMaxInterpolation**[{f_x,f_y,g},{t,(tmin,tmax),m,k},x] – дает рациональную минимаксную интерполяцию для параметрически заданных функций при автоматическом выборе значений параметра t и с ошибкой, заданной g(t).

Примеры на применение этого довольно редкого вида аппроксимации можно найти в справке системы Mathematica. Там же можно найти дополнительные соображения по уменьшению погрешности аппроксимации элементарных и специальных математических функций.

6.3.10. Сплайновая интерполяция и аппроксимация

Сплайны представляют собой набор полиномов невысокой степени, последовательно применяемых к наборам точек аппроксимирующей функции. Чаще всего используется *кубическая сплайновая аппроксимация*, которой коэффициенты полиномов выбираются из условий равенства в стыкуемых точках не только значений функции, но и первой и второй производных. Это придает графику сплайна вид плавной кривой, точно проходящей через узловые точки и напоминающей изгибы гибкой линейки.

Для кубических сплайнов $S''''(x)=0$, где $S''''(x)$ – четвертая производная. Из этого следует, что между каждой парой соседних узлов интерполяционная формула записывается в виде полинома третьей степени. Этот полином удобно представить следующим образом:

$$S(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$

$$x_{i-1} \leq x \leq x_i, i = 1, 2, \dots, n.$$

Для построения кубического сплайна необходимо построить n полиномов третьей степени, то есть определить $4n$ неизвестных a_i, b_i, c_i, d_i . Эти коэффициенты ищутся из условия равенства в узлах. В узлах сплайн должен принимать табличные значения функции:

$$S(x_{i-1}) = a_i = f_{i-1},$$

$$S(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_i, \text{ где } h_i = x_i - x_{i-1}.$$

Методика нахождения коэффициентов основана как на равенстве им в узловых точках значений полиномов, так и непрерывности первой и второй производных функций. Этот процесс в Mathematica автоматизирован. Помимо обычных кубических сплайнов, у которых точки стыковки полиномов совпадают с узловыми точками, иногда применяются так называемые В-сплайны, у которых точки стыковки могут иметь и иное расположение.

Подпакет SplineFit встроенного пакета расширения NumericalMath содержит функцию для проведения сплайновой интерполяции и экстраполяции:

SplineFit[data, type] – возвращает сплайн-функцию для данных data и типа сплайн-аппроксимации type, по умолчанию это кубический сплайн Cube (другие типы Bezier (В-сплайны) и CompositeBezier). Сплайн-функция имеет формат: SplineFunction[type, range, interval].

Рисунок 6.23 показывает пример *сплайн-интерполяции* для зависимости $y(x)$, представленной пятью парами точек в прямоугольной системе координат. На нем построены также графики аппроксимирующей функции и исходных точек.

Специфика сплайн-регрессии по функции **SplineFit** заключается в преобразовании значений как x_i , так и y_i . Это позволяет представлять сплайнами в общем виде параметрически заданные функции, что поясняет рис. 6.24.

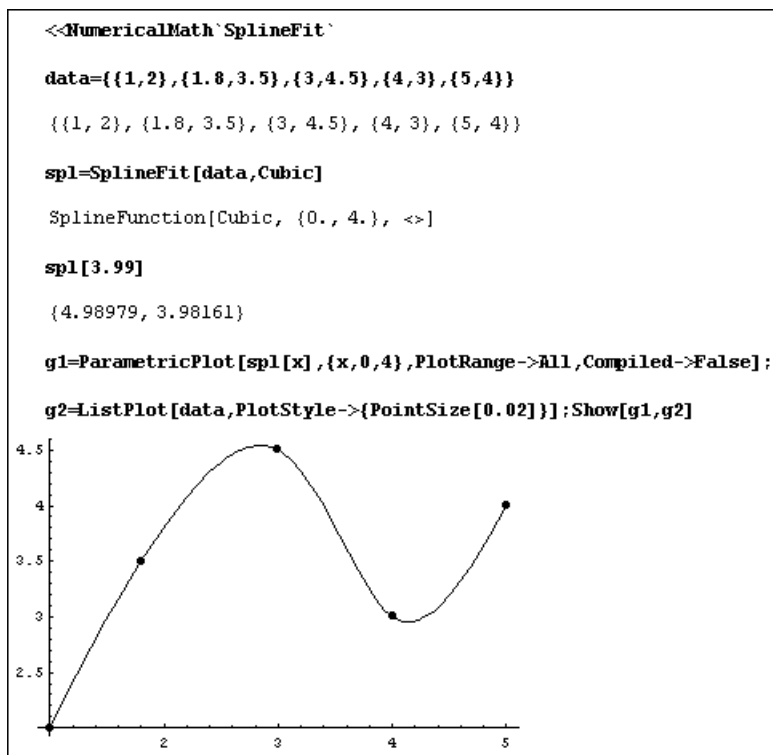


Рис. 6.23. Пример сплайн-интерполяции для зависимости $y(x)$, заданной списком координат своих узловых точек

С помощью функции `InputForm` можно вывести детальные данные о примененных сплайн-функциях, что показано в конце ноутбука (рис. 6.24).

6.4. Регрессия и метод наименьших квадратов

6.4.1. Регрессия и визуализация ее результатов

Зачастую некоторая зависимость задается большим числом отсчетов, и ее данные засорены шумами или значительными случайными погрешностями измерений. В этом случае применение рассмотренных выше видов интерполяции и аппроксимации становится невозможным. Возникает необходимость либо в отдельной статистической обработке данных, либо в ее применении в ходе получения аппрокси-

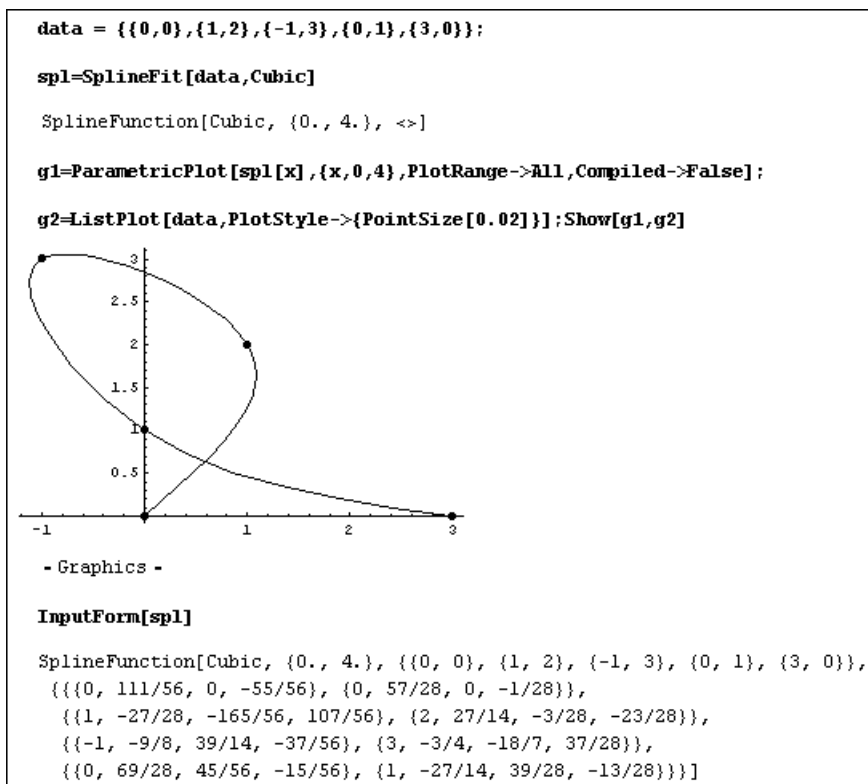


Рис. 6.24. Пример сплайн-интерполяции параметрически заданной функции

мирующего выражения. Последнее лежит в основе широко используемого вида аппроксимации – *регрессии*.

Регрессия заключается в нахождении параметров некоторой функции регрессии, при которой график функции проходит в «облаке» узловых точек, обеспечивая наименьшую среднеквадратическую погрешность вычислений. В этом заключается *метод наименьших квадратов*. В отличие от интерполяции, при регрессии данная функция в узловых точках не дает точного значения ординат – она просто минимизирует погрешности вычислений в этих точках.

Для решения задач регрессии используется функция **Fit**:

Fit[data, funs, vars] – для списка данных **data** ищет приближение методом наименьших квадратов в виде линейной комбинации функций **funs** переменных **vars**. Данные **data** могут иметь форму $\{\{x_1, y_1, \dots, f_1\}, \{x_2, y_2, \dots, f_2\}, \dots\}$, где число координат x, y, \dots равно числу переменных в списке **vars**. Также данные **data** могут быть представлены в форме $\{f_1, f_2, \dots\}$ с одной координатой, принимающей значения 1, 2, Аргумент **funs** может быть любым списком функций, которые зависят только от объектов **vars**.

Следующие примеры показывают приближение исходных данных степенными полиномами второй и третьей степени и линейной комбинацией двух функций:

```
Fit[{{0, 0.9}, {2, 8.1}, {3, 17}, {4, 33}}, {1, x, x^2}, x]
```

```
0.997273 - 1.40864 x + 2.33409 x^2
```

```
Fit[{{0, 0.9}, {2, 8.1}, {3, 17}, {4, 33}}, {1, x, x^2, x^3}, x]
```

```
0.9 + 2.74167 x - 0.4625 x^2 + 0.445833 x^3
```

```
Fit[{{0, 0.9}, {2, 8.1}, {3, 17}}, {x^2, Exp[x], x}, x]
```

```
0.9 e^x + 2.89276 x - 1.08392 x^2
```

Здесь в первом примере выполняется полиномиальная регрессия со степенью многочлена 2. Максимальная степень на 1 меньше числа пар исходной зависимости: при такой степени регрессия вырождается в обычную полиномиальную аппроксимацию. Она рассматривалась выше. В нашем случае число пар равно 4, так что вырождение наступает при степени полинома, равной 3 (второй пример). Третий пример представляет данные линейной комбинации квадратичной, экспоненциальной и линейной функций.

Рисунок 6.25 показывает несколько иной путь проведения полиномиальной регрессии: исходные данные заданы объектом-списком `data`. Представлено два варианта регрессии: для степени полинома 1 (линейная регрессия) и 2 (квадратичная регрессия).

В конце документа рис. 6.25 дано построение графиков аппроксимирующих полиномов и точек исходных данных. Заметно, что при регрессии график полиномов проходит в середине «облака» исходных точек и не укладывается на них точно.

6.4.2. Функции линейной регрессии

В подпакете `LinearRegression` встроенного пакета расширения `Statistics` имеются расширенные функции для проведения *линейной регрессии общего вида*, в дополнение включенной в ядро функции **Fit**. Прежде всего, это функция **Regress**:

- **Regress**[`data`, {`I`, `x`, `x^2`}, `x`] – осуществляет регрессию данных `data`, используя квадратичную модель.
- **Regress**[`data`, {`I`, `x1`, `x2`, `x1x2`}, {`x1`, `x2`}] – осуществляет регрессию, используя в ходе итераций зависимость между переменными `x1` и `x2`.
- **Regress**[`data`, {`f1`, `f2`, ...}, `vars`] – осуществляет регрессию, используя модель линейной регрессии общего вида с уравнением регрессии, представляющим линейную комбинацию функций `fi` от переменных `vars`.

Данные могут быть представлены листом ординат {`y1`, `y2`, ...} или листом {{`x11`, `x12`, ..., `y1`}, {`x21`, `x22`, ..., `y2`}, ...}. Пример применения функции **Regress** представлен на рис. 6.26. Нетрудно заметить, что функция выводит детальный отчет по проведенной регрессии.

На рис. 6.27 показан еще один пример проведения регрессии с графической визуализацией с помощью функции **MultipleListPlot**. Применение этой функции

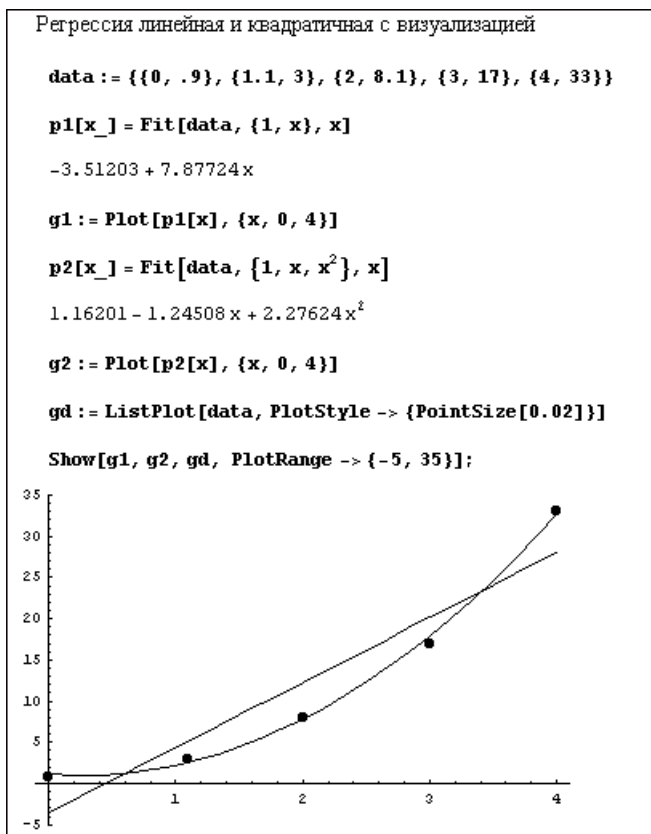


Рис. 6.25. Полиномиальная регрессия с графическим выводом

позволило построить линии интервала разброса данных, в котором располагаются точки данных для регрессии.

Пакет линейной регрессии содержит ряд и иных функций, с которыми можно ознакомиться с помощью справочной базы данных системы Mathematica. Напоминаем еще раз, что сама функция при линейной регрессии может быть нелинейная, она является линейной только относительно искоемых коэффициентов регрессии.

6.4.3. Функции нелинейной регрессии

В подпакете NonlinearFit пакета статистических вычислений Statistica (см. Главу 12) содержатся функции для выполнения *нелинейной регрессии общего вида*:

NonlinearFit[data,model,variables,parameters] – выполняет регрессию по заданной модели (формуле) model с переменными variables и параметрами parameters для заданных данных data.

```

<<Statistics`LinearRegression`

data = {{0.05, 90}, {0.09, 97}, {0.14, 107}, {0.17, 124}, {0.18, 142},
        {0.21, 150}, {0.23, 172}, {0.25, 189}, {0.28, 209}, {0.35, 253}};

(regress = Regress[data, {1, x, x^2}, x];
Chop[regress, 10^(-5)])

      Estimate      SE      TStat      PValue
{ParameterTable → 1      73.2869    11.2362    6.52237    0.000327186
                   x      170.573    120.586    1.41453    0.200111
                   x^2    1034.55    298.131    3.47012    0.0104042

RSquared → 0.982579, AdjustedRSquared → 0.977601, EstimatedVariance → 62.1793,

ANOVA Table →  Model    DF    SumOfSq    MeanSq    FRatio    PValue
                  Error    7    435.255    62.1793    197.404    0
                  Total    9    24984.1
}

func = Fit[data, {1, x, x^2}, x]

73.2869 + 170.573 x + 1034.55 x^2

Options[Regress]

{RegressionReport → SummaryReport,
 IncludeConstant → True, BasisNames → Automatic,
 Weights → Automatic, Tolerance → Automatic, ConfidenceLevel → 0.95}

```

Рис. 6.26. Пример применения функции *Regress*

NonlinearRegress[data,model,variables,parameters] – выполняет регрессию по заданной модели (формуле) *model* с переменными *variables* и параметрами *parameters* для заданных данных *data* с выдачей листа диагностики.

Данные могут быть представлены листом ординат {y1,y2,...} или листом {{x11,x12,...,y1},{x21,x22,...,y2},...}. В ходе регрессии минимизируются заданные параметры, так что заданная модель регрессии приближает данные с минимальной среднеквадратической погрешностью.

На рис. 6.28 показан пример выполнения логарифмической регрессии. При ней модель представлена выражением $a \cdot \text{Log}[b \cdot x]$. Результатом действия функции **NonlinearFit** является уравнение регрессии в виде этой модели с найденными значениями параметров *a* и *b*. Представлена также визуализация регрессии в виде графика функции – модели и исходных точек.

Отчет по нелинейной регрессии, который формирует функция **NonlinearRegression**, представлен на рис. 6.29 для примера, представленного на рис. 6.28.

Более детальные данные об опциях и обозначениях в отчетах нелинейной регрессии можно найти в справочной базе данных.

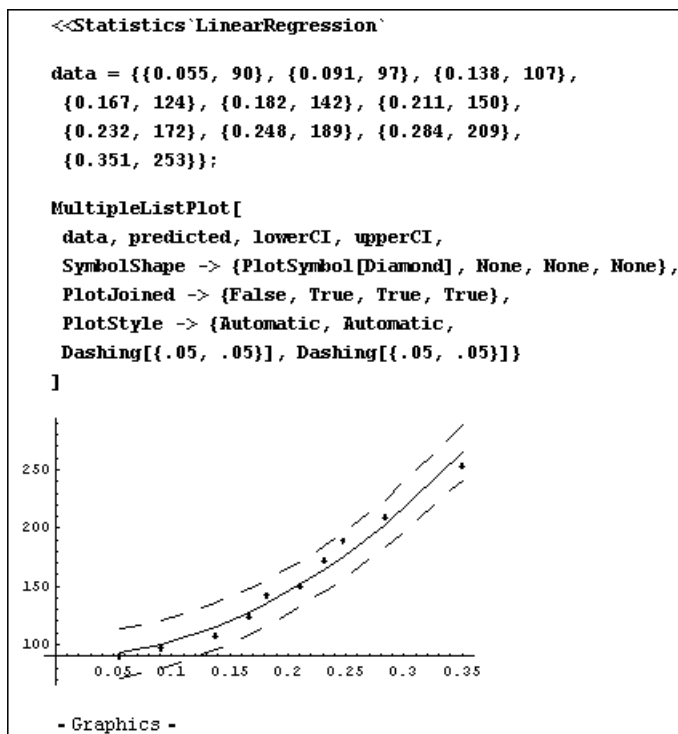


Рис. 6.27. Пример проведения регрессии с графической визуализацией

6.4.4. Функции полиномиальной регрессии

В подпакете `PolynomialFit` пакета `NumericalMath` (см. Главу 11) определена функция для полиномиальной регрессии для полинома степени n без явного указания x^d :

`PolynomialFit[data,n]` – возвращает полином степени n , обеспечивающий наилучшее среднеквадратичное приближение для данных, представленных `data`. Если `data` представлен списком ординат функции, то абсциссы формируются автоматически с шагом 1. Если `data` представлен списком координат $\{x_i, y_i\}$, то полином наилучшим образом приближает зависимость $y_i(x_i)$.

Ниже представлен пример на применение функции полиномиальной аппроксимации:

```
<<NumericalMath`PolynomialFit`
p = PolynomialFit[ { 1,3.9,4.1,8.9,16,24.5,37,50 } ,3]
(FittingPolynomial[<>, 3 ])
p[5.]
25.
Expand[p[x]]

```

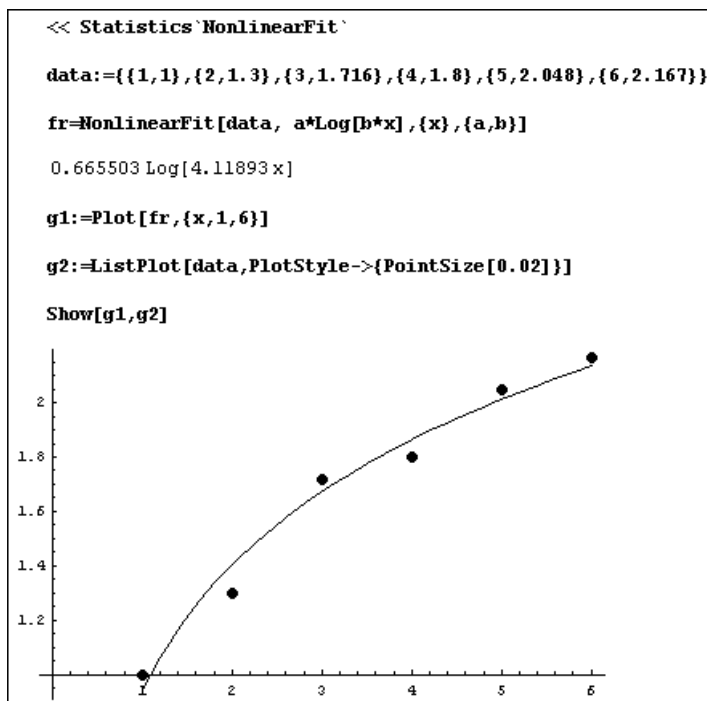


Рис. 6.28. Пример логарифмической регрессии

```
NonlinearRegress[data, a*Log[b*x],{x},{a,b}]
```

```
{BestFitParameters->{a->0.665503, b->4.11893},
```

	Estimate	Asymptotic SE	CI
ParameterCITable-> a	0.665503	0.0504167	{0.525524, 0.805482},
b	4.11893	0.806289	{1.88031, 6.35754}

```
EstimatedVariance->0.00558058,
```

	DF	SumOfSq	MeanSq
Model	2	17.7425	8.87126
ANOVATable-> Error	4	0.0223223	0.00558058,
Uncorrected Total	6	17.7648	
Corrected Total	5	0.994689	

```
AsymptoticCorrelationMatrix->{ 1. -0.972212},
{-0.972212 1.},
```

	Curvature
FitCurvatureTable-> Max Intrinsic	3.94364×10 ⁻¹⁶
Max Parameter-Effects	2.07792
95. % Confidence Region	0.379478

Рис. 6.29. Отчет о проведении нелинейной регрессии

$$0. \times 10^{-11} + 0. \times 10^{-11} x + 1.00000000000 x^2 + 0. \times 10^{-13} x^3$$

Другой пример с построением графиков исходных точек и аппроксимирующего полинома дан на рис. 6.30.

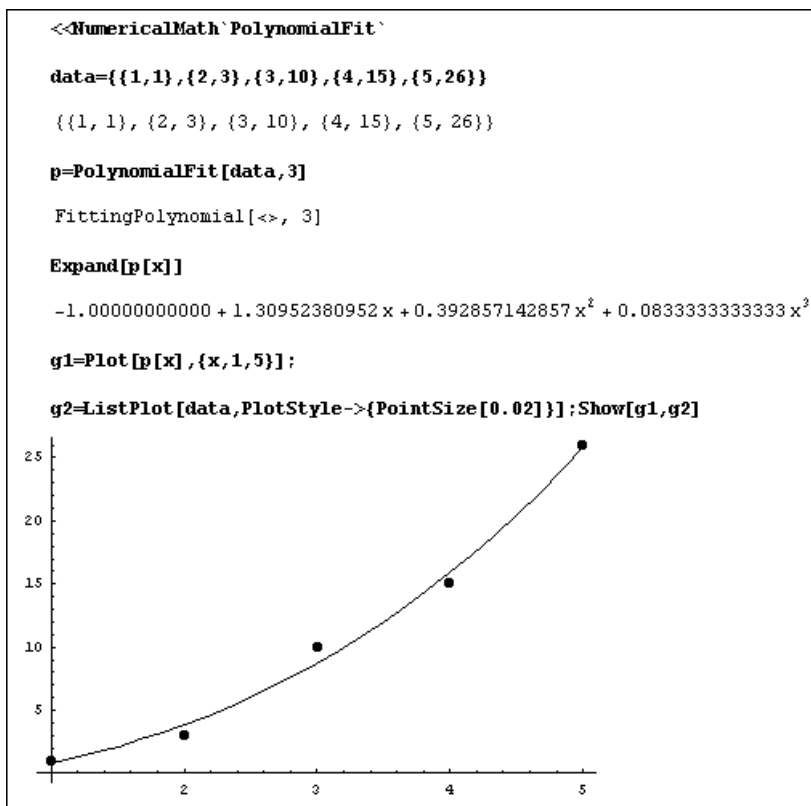


Рис. 6.30. Пример полиномиальной регрессии с графической визуализацией

Нетрудно заметить, что точки исходной зависимости неплохо (но не точно) укладываются на график полинома.

6.4.5. Функции тригонометрической регрессии

Многие выражения содержат периодические тригонометрические функции, например $\sin(x)$ или $\cos(x)$. Помимо обычного спектрального представления выражений, подпакет TrigFit пакета NumericalMath имеет функции для *тригонометрической регрессии*:

- **TrigFit[data, n, x]** – дает тригонометрическую регрессию для данных data с $\cos(n x)$ и $\sin(n x)$ и с периодом 2π .
- **TrigFit[data, n, {x, L}]** – дает тригонометрическую регрессию для данных data с $\cos(2\pi n x/L)$ и $\sin(2\pi n x/L)$ и с периодом L.
- **TrigFit[data, n, {x, x0, x1}]** – дает тригонометрическую регрессию для данных data с $\cos(2\pi n(x-x_0)/(x_1-x_0))$ и $\sin(2\pi n(x-x_0)/(x_1-x_0))$ и с периодом (x_1-x_0) .

Примеры тригонометрической регрессии даны ниже:

```
<<NumericalMath`TrigFit`
data = Table[1+Sin[x]+3Cos[2x]+3 Sin[3x],
  {x, 0, 2Pi-2Pi/7, 2Pi/7}];
TrigFit[data, 0, x]
1.
TrigFit[data, 1, {x, L}]
(1. + 0. Cos[ $\frac{2 \pi x}{L}$ ] + 1. Sin[ $\frac{2 \pi x}{L}$ ])
Fit[Transpose[{Range[0, 2Pi-2Pi/7, 2Pi/7], data}],
  {1, Cos[x], Sin[x]}, x]
1. + 3.4766  $\times 10^{-16}$  Cos[x] + 1. Sin[x]
TrigFit[data, 3, {x, x0, x1}]
1. + 0. Cos[ $\frac{2 \pi (x-x_0)}{-x_0+x_1}$ ] + 3. Cos[ $\frac{4 \pi (x-x_0)}{-x_0+x_1}$ ] -
4.44089  $\times 10^{-16}$  Cos[ $\frac{6 \pi (x-x_0)}{-x_0+x_1}$ ] + 1. Sin[ $\frac{2 \pi (x-x_0)}{-x_0+x_1}$ ] +
4.44089  $\times 10^{-16}$  Sin[ $\frac{4 \pi (x-x_0)}{-x_0+x_1}$ ] + 3. Sin[ $\frac{6 \pi (x-x_0)}{-x_0+x_1}$ ]
```

Этот вид регрессии применяется редко.

6.5. Функции дискретного преобразования Фурье

6.5.1. Прямое и обратное дискретное преобразование Фурье

Многие функции имеют представление в виде тригонометрического ряда, известного под названием ряда Фурье. Задача дискретного преобразования Фурье ставится следующим образом: нужно аппроксимировать на интервале $(0, T)$ тригонометрическим полиномом N -го порядка функцию $y=f(x)$, для которой известны m ее значений $y_k=f(x_k)$ при $x_k=kT/m$, где $k=0, 1, 2, \dots, m-1$. Такой полином имеет вид:

$$Q_n(x) = \frac{a_0}{2} + \sum_{n=1}^N \left(a_n \cos\left(\frac{2\pi}{T} x\right) + b_n \sin\left(n \frac{2\pi}{T} x\right) \right).$$

Коэффициенты a_n и b_n определяются следующими соотношениями:

$$a_n = \frac{2}{T} \int_0^T f(x) \cos\left(n \frac{2\pi}{T} x\right) dx,$$

$$b_n = \frac{2}{T} \int_0^T f(x) \sin\left(n \frac{2\pi}{T} x\right) dx, \quad n = 0, 1, 2, \dots, N.$$

Применяя для вычисления этих интегралов формулу прямоугольников, высота которых определяется по значениям подынтегральных выражений в точках $x_k = kT/m$, где $k=0, 1, 2, \dots, m-1$, имеем:

$$a_n = \frac{2}{m} \sum_{k=0}^{m-1} y_k \cos\left(n \frac{2\pi k}{m}\right),$$

$$b_n = \frac{2}{m} \sum_{k=0}^{m-1} y_k \sin\left(n \frac{2\pi k}{m}\right), \quad n = 0, 1, 2, \dots, N.$$

Таким образом, тригонометрический полином, коэффициенты a_n и b_n находятся по этим формулам, что служит решением поставленной задачи. При этом коэффициенты ряда Фурье минимизируют сумму квадратов отклонений

$$\delta_N^2 = \sum_{k=0}^{m-1} [Q_n(x_k) - y_k]^2.$$

Сам же полином $Q_N(x)$ становится интерполяционным полиномом, так как в этом случае при любом b_N выполняется соотношение $Q_N(x_k) = y_k$ для всех $x_k = kT/m$, где $k=0, 1, 2, \dots, m-1$.

Метод Фурье особенно удобен для представления периодических функций, например, электрических сигналов в радиоэлектронных устройствах. В случае вычисления коэффициентов ряда Фурье методом прямоугольников он дает приближение по методу наименьших квадратов, т.е. реализует тригонометрическую регрессию. Формулы дискретного преобразования Фурье можно найти в справке по системе Mathematica.

Для повышения скорости преобразований используются довольно сложные алгоритмы БПФ, особенно эффективные, если число точек преобразования равно 2^N , где $N=1, 2, 3, \dots$. Именно эти алгоритмы использует система Mathematica в составе следующих функций:

- **Fourier[list]** – осуществляет дискретное преобразование Фурье для списка list комплексных чисел.
- **InverseFourier[list]** – осуществляет дискретное обратное преобразование Фурье списка list комплексных чисел.

Параметром **list** этих функций в общем случае является список, содержащий комплексные числа. Результаты прямого и обратного преобразований Фурье должны приводить к результату, совпадающему с исходными данными (в пределах малой погрешности). Это подтверждает следующий пример:

```
DF := Fourier[{1, 1, 0, 0}]
DF
```

```
{1., +0. i, 0.5 +0.5 i, 0. +0. i, 0.5 -0.5 i}
IF:= InverseFourier[DF]
IF
{1., 1., 0., 0.}
```

Разумеется, этот пример носит исключительно тестовый характер. Но он показывает, что в ходе преобразований могут появляться векторы с комплексными числами. Используя множество возможностей по работе с комплексными числами, можно решать различные задачи спектрального анализа и синтеза сигналов различной формы.

Применение описанных функций имеет некоторые тонкости. Прежде всего, следует отметить, что отсчет элементов векторов начинается не с нуля, а с единицы. Поэтому нулевая гармоника (в электро- и радиотехнике ее называют *постоянной составляющей* разлагаемой в ряд Фурье зависимости) соответствует индексу 1, первая гармоника — индексу 2 и т.д. Таким образом, имеет место смещение нумерации индексов на единицу.

Согласно теореме отсчетов, именуемой также теоремой Котельникова, если функция имеет N отсчетов, то максимальное число гармоник спектрального разложения, достаточное для представления функции, равно $N/2$. Между тем функция `Fourier` в системе `Mathematica` дает все N элементов создаваемого ею вектора. При этом на спектрограмме «лишние» гармоники на деле просто образуют зеркальное отображение реально возможных $N/2$ гармоник. Именно поэтому двойное (прямое и обратное) преобразование Фурье в системе `Mathematica` почти идеально точно восстанавливает исходный вектор.

Еще одна тонкость связана с необычным представлением нулевых мнимых частей элементов векторов, получаемых в ходе преобразований. Они записываются в виде 0. I. Для их устранения может использоваться функция `Chop[V]`.

Для лучшего понимания особенностей спектрального анализа и синтеза рекомендуется внимательно ознакомиться с формулами преобразований Фурье, которые можно найти в справочной системе, благо эти формулы вполне понятны даже тем, кто не силен в английском языке. В литературе подобные формулы встречаются в нескольких различных видах, что порождает некоторые трудности в интерпретации и нормировке результатов спектрального анализа и синтеза. Поэтому полезно познакомиться с дополнительными и вполне конкретными примерами, приведенными ниже.

6.5.2. Спектральный анализ на основе прямого преобразования Фурье

Итак, прямое преобразование Фурье означает перевод *временного* представления сигнала в *частотное*. Другими словами, оно позволяет получить *частотный спектр* сигнала, представленного отсчетами его временной зависимости. Нередко это является конечной целью спектрального анализа.

На рис. 6.31 представлен пример спектрального анализа простого сигнала — треугольного импульса, заданного с помощью функции `If`. Затем, с помощью функ-

Спектральный анализ для функции $f_2[t]$, задающей пилообразный импульс

```
f2[t_] := If[{0 > t} || {t > 1.5}, 0, t];
```

```
data := Table[f2[t / 8], {t, 16}]; ft := Fourier[data]; ft
```

```
{2.4375 + 0. i, -1.21082 + 0.0764104 i, 0.112056 + 0.528109 i, 0.294312 - 0.154429 i,  
-0.1875 - 0.1875 i, -0.131399 + 0.202265 i, 0.200444 + 0.0906092 i, 0.0479075 - 0.191895 i,  
-0.1875 + 0. i, 0.0479075 + 0.191895 i, 0.200444 - 0.0906092 i, -0.131399 - 0.202265 i,  
-0.1875 + 0.1875 i, 0.294312 + 0.154429 i, 0.112056 - 0.528109 i, -1.21082 - 0.0764104 i}
```

Создадим векторы амплитуд и фаз всех 16 гармоник

```
Mg := Abs[ft]; Mg
```

```
{2.4375, 1.21323, 0.539867, 0.332367, 0.265165, 0.241199, 0.219972, 0.197785,  
0.1875, 0.197785, 0.219972, 0.241199, 0.265165, 0.332367, 0.539867, 1.21323}
```

```
Ag := Arg[ft]; Ag
```

```
{0., 3.07857, 1.36171, -0.483222, -2.35619, 2.14692, 0.424551, -1.32614,  
3.14159, 1.32614, -0.424551, -2.14692, 2.35619, 0.483222, -1.36171, -3.07857}
```

Рис. 6.31. Спектральный анализ пилообразного импульса на основе прямого преобразования Фурье

ции **Fourier** прямого преобразования Фурье получены в явном виде векторы амплитуд **Mg** и фаз **Ag** гармоник этого сигнала.

На рис. 6.32 представлено продолжение документа, показанного на рис. 6.31. Здесь с помощью графиков лестничного типа, подчеркивающих дискретность гармоник, построены спектрограммы амплитуд и фаз гармоник пилообразного импульса. Хорошо видно симметричное отражение линий спектра относительно восьмой гармоники — в нашем случае имелось 16 отсчетов сигнала. Это значит, что амплитуда и фаза девятой гармоники те же, что у седьмой гармоники, у десятой — те же, что у шестой и т.д.

6.5.3. Применение преобразования Фурье для получения спектра сигналов

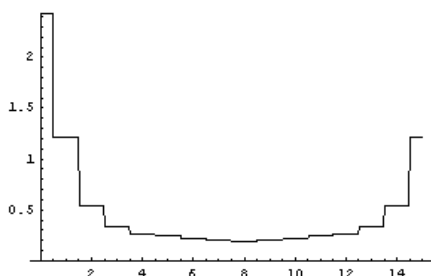
Теперь рассмотрим более сложный случай — получение спектра сложного сигнала (рис. 6.33).

В начале этого рисунка показано формирование синусоидального сигнала с частотой 50 Гц, на который наложена значительная по амплитуде шумовая составляющая. Она создается добавлением к отсчетам сигнала случайных величин, созданных генератором случайных чисел.

Во второй части рисунка показан график частотных отсчетов, полученных после прямого преобразования Фурье. На нем отчетливо виден пик в районе час-

Построим спектрограмму модулей гармоник, обеспечив верное отражение нулевой гармоники.

Plot[Mg[[Round[i] + 1]], {i, 0, 15}, PlotPoints → 16, PlotRange → {0, Mg[[1]}]]



Построим спектрограмму фаз гармоник.

Plot[Ag[[Round[i] + 1]], {i, 0, 15}, PlotPoints → 16, PlotRange → {-π, π}]

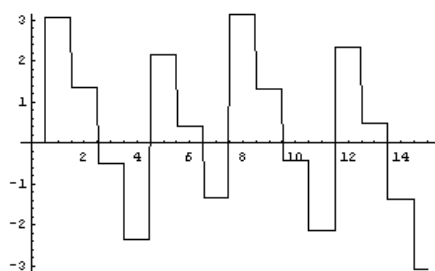


Рис. 6.32. Спектрограммы амплитуд и фаз гармоник пилообразного импульса

тоты 50 Гц (поскольку первый элемент результирующего списка соответствует нулевой частоте, этот пик возникает на 51-м элементе списка). Однако помимо него существует еще один пик на частоте $256 - 50 = 206$ Гц. Он связан с указанным выше свойством симметрии спектра вещественного сигнала.

6.5.4. Фильтрация сигналов с помощью преобразований Фурье

Преобразование Фурье является теоретической основой фильтрации сложных сигналов. Мы рассмотрим комплексный пример на фильтрацию сигнала, представляющего собой функцию Бесселя первого рода третьего порядка. Рисунок 6.34 показывает верхнюю часть документа, демонстрирующую создание исходного сигнала и описание частотного фильтра.

Как и в ранее рассмотренном примере, сигнал формируется как сумма чистого сигнала со случайной компонентой, моделирующей шум. Выбранная форма сигнала напоминает затухающую синусоиду. Уровень шумов выбран достаточно большим, так что форма чистого сигнала с трудом угадывается на фоне шумов (верхний график).

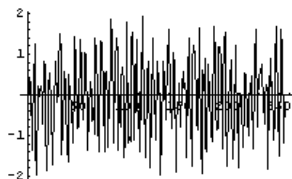
Спектральный анализ сложного сигнала

Создание периодического сигнала, имеющего 256 отсчетов с примесью шума - случайных чисел

```
data = Table[ N[Sin[50 2 Pi n/256] + 2*(Random[ ] - 1/2)], {n, 256} ] ;
```

Построение графика сигнала

```
ListPlot[ data, PlotJoined -> True ]
```



Прямое преобразование Фурье позволяет получить спектрограмму сигнала с двумя отчетливо видимыми пиками.

```
ListPlot[ Abs[Fourier[data]], PlotJoined -> True, PlotRange -> All ]
```

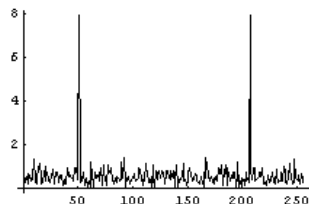


Рис. 6.33. Получение спектра сложного сигнала с помощью прямого преобразования Фурье

Далее показан синтез частотного фильтра и его амплитудно-частотная характеристика (АЧХ). Ее пик должен примерно соответствовать средней частоте спектра сигнала. График АЧХ показан в нижней части документа (рис. 6.34).

На следующем рисунке (рис. 6.35) показан процесс фильтрации. Он сводится к уточнению модели фильтра (ротации АЧХ в область отрицательных частот), а затем к проведению вначале прямого преобразования Фурье, выделению фильтром соответствующих составляющих сигнала, и затем к обратному преобразованию Фурье. Оба преобразования и фильтрация осуществляются в одном выражении – строке `In[67]`. При этом частотные отсчеты сигнала и фильтра перемножаются. Обратное преобразование Фурье переводит результат фильтрации во временную область. Полученный в результате фильтрации сигнал практически очищен от шума. Это подтверждает график сигнала, представленный в нижней части рис. 6.35.

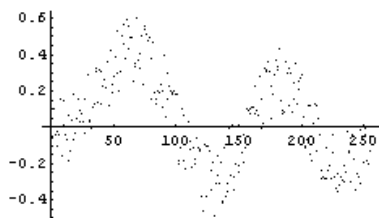
Эти примеры показывают на высокую эффективность средств Mathematica 3/4 в решении задач спектрального анализа, синтеза сигналов и их фильтрации и иных преобразований. Важно отметить, что в новейших версиях Mathematica 4/5 использованы ускоренные алгоритмы быстрого преобразования Фурье, обеспечивающие ускорение описанных операций в несколько раз. Это открывает воз-

Фильтрация сигнала на основе дискретного преобразования Фурье

Создание сложного сигнала с шумовой компонентой

```
data = Table[ N[BesselJ[3, 16*n/256] + 0.4 (Random[ ] - 1/2)], {n, 256} ] ;
```

```
ListPlot[data]
```



Построение фильтрующей функции и ее частотной характеристики

```
kern = Table[Exp[-n^2/200.], {n, -128, 127}] ;ListPlot[kern, PlotRange -> All]
```

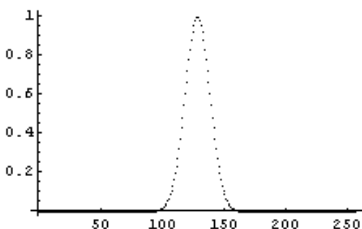


Рис. 6.34. Часть документа,
показывающая создание сигнала и синтез его фильтра

возможность решения серьезных задач обработки сигналов, представленных многими тысячами отсчетов. Другими словами, реально применяемых в технике связи сигналов.

6.5.5. Расширенные функции для преобразования Фурье

Подпакет FourierTransform пакета Calculus имеет ряд *расширенных функций* для более совершенного преобразования Фурье.

Для реализации спектрального анализа и синтеза имеются следующие функции:

- **FourierExpSeries[expr,{x,xmin,xmax},n]** – возвращает разложение $\text{expr}[x]$ в экспоненциальный ряд Фурье с n членами на отрезке $\{xmin, xmax\}$.
- **FourierExpSeriesCoefficient[expr,{x,xmin,xmax},n]** – возвращает коэффициенты разложения $\text{expr}[x]$ в экспоненциальный ряд Фурье с n членами на отрезке $\{xmin, xmax\}$.

Осуществление фильтрации на основе дискретного преобразования Фурье

```
kern = RotateLeft[kern, 128]/Apply[Plus, kern] ;
```

```
conv = InverseFourier[Sqrt[256] Fourier[data] Fourier[kern] ] ;
```

Графическая визуализация результата фильтрации - получен очищенный от шума исходный сигнал

```
ListPlot[ Chop[conv] ]
```

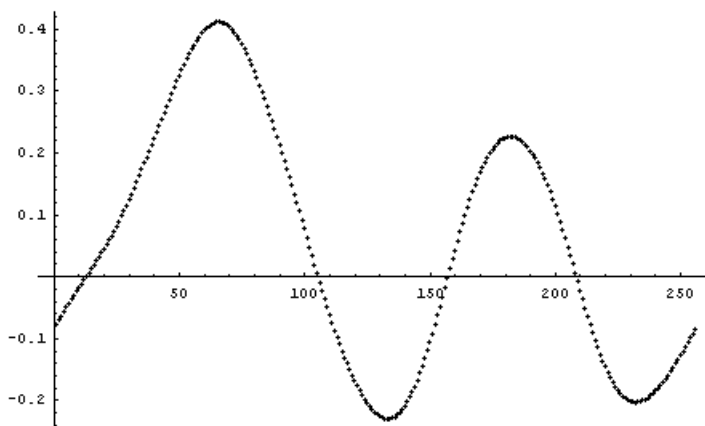


Рис. 6.35. Часть документа, показывающего фильтрацию сигнала и построение графика сигнала, очищенного от шума

- **FourierTrigSeries[expr,{x,xmin,xmax},n]** – возвращает разложение $\text{expr}[x]$ в тригонометрический ряд Фурье с n членами на отрезке $\{xmin, xmax\}$.
- **FourierSinSeriesCoefficient[expr,{x,xmin,xmax},n]** – возвращает синусные коэффициенты разложения $\text{expr}[x]$ в тригонометрический ряд Фурье с n членами на отрезке $\{xmin, xmax\}$.
- **FourierCosSeriesCoefficient[expr,{x,xmin,xmax},n]** – возвращает косинусные коэффициенты разложения $\text{expr}[x]$ в тригонометрический ряд Фурье с n членами на отрезке $\{xmin, xmax\}$.

Рисунок 6.36 иллюстрирует создание пилообразного сигнала, его разложение в тригонометрический ряд Фурье с $n=4$ и графическое воспроизведение сигнала и его представление суммой из четырех гармоник (на рисунке оставлены только совмещенные графики). Таким образом, последняя операция демонстрирует проведение синтеза пилообразного сигнала по четырем гармоникам.

Помимо указанных функций существует целая группа функций для численных операций, связанных с разложением в ряд Фурье. Все они имеют в начале имени букву **N**, например:

NFourierTrigSeries[expr,{x,xmin,xmax},n] – возвращает разложение $\text{expr}[x]$ в тригонометрический ряд Фурье с n членами на отрезке $\{xmin, xmax\}$ и в численном виде.

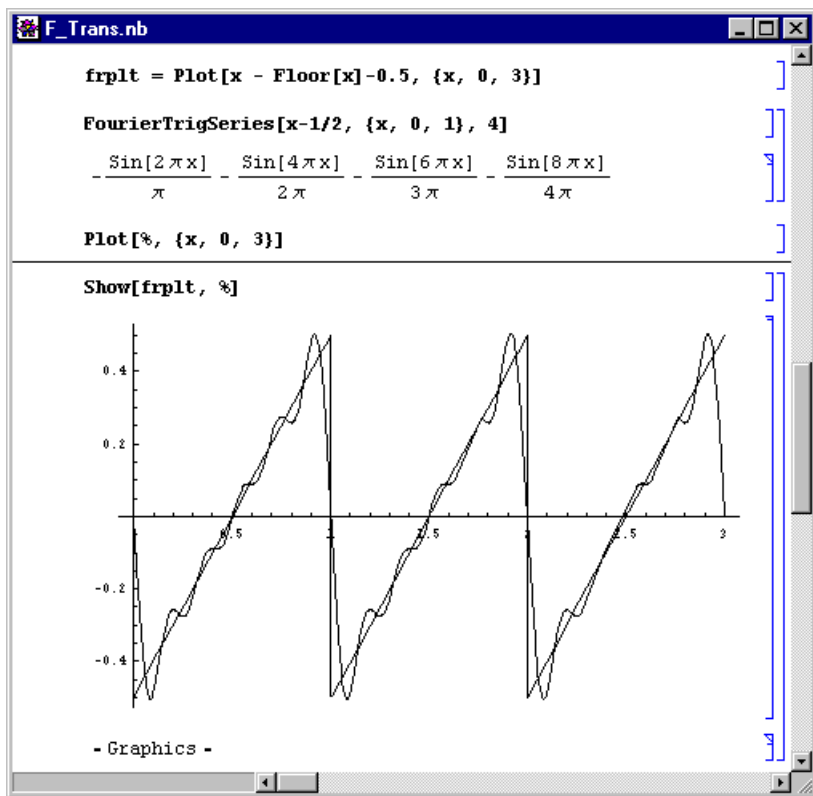


Рис. 6.36. Создание пилообразного сигнала, его разложение в тригонометрический ряд Фурье и синтез сигнала по 4 гармоникам

Предоставляем читателю возможность проверить работу этих функций самостоятельно.

6.6. Кусочные функции Piecewise

6.6.1. Задание кусочных функций

Mathematica позволяет создавать *кусочные функции*, которые имеют несколько участков, в каждом из которых определена своя функциональная зависимость. Такие функции широко применяются, например, для моделирования сложных сигналов. Они задаются следующей функцией системы Mathematica:

```
Piecewise[{{val1,cond1},{val2,cond2},...}]
Piecewise[{{val1,cond1},...},val]
```

Здесь **val_i** – выражение для *i*-го участка функции, **cond_i** – условие для этого участка, определяющая область его определения. Условия обычно задаются в форме неравенств.

Во второй форме параметр **val** используется для задания значения функции, там, где она не определена списком значений и условий. Значение **val** по умолчанию есть 0. Кусочные функции могут быть непрерывными и разрывными.

6.6.2. Работа с кусочными функциями

Работа с кусочными функциями в принципе ничем не отличается от работы с другими видами функций. Так, можно строить их графики, дифференцировать, интегрировать, выполнять различные символьные преобразования, вычислять их числовые значения и т.д. Обычно результат вычислений или преобразований также является кусочной функцией, определенной на тех же участках, что и исходная функция.

Рисунок 6.37 показывает задание кусочной функции **pw** из трех участков, построение графика функции, дифференцирование и затем интегрирование функ-

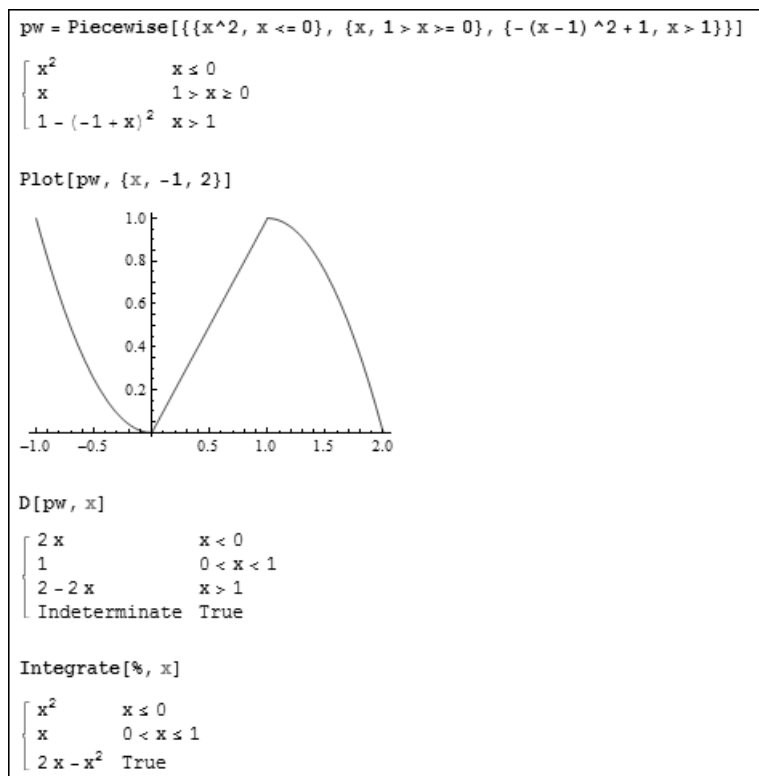


Рис. 6.37. Пример работы с кусочной функцией

ции. Результаты вычислений подтверждают сказанное выше о работе с кусочными функциями.

6.7. Новые средства Mathematica 6

6.7.1. Функции полиномиальной интерполяции

В Mathematica 6 функции интерполяции **Interpolation** и **InterpolatingFunction** и **InterpolatingPolynomial** существенно модифицированы. Могут использоваться четыре формы записи функции **InterpolatingPolynomial**:

```
Interpolation[{f1, f2, ...}] Interpolation[{x1, f1}, {x2, f2}, ...]
Interpolation[{x1, y1, ...}, {f1}, {x2, y2, ...}, {f2}, ...]
Interpolation[{x1, ...}, {f1, df1, ...}, ...]
```

Аналогично, модифицированная функция **InterpolatingPolynomial** также имеет четыре формы записи:

```
InterpolatingPolynomial[{f1, f2, ...}, x]
InterpolatingPolynomial[{x1, f1}, {x2, f2}, ...}, x]
InterpolatingPolynomial[{x1, y1, ...}, {f1}, {x2, y2, ...}, {f2}, ...}, {x, y, ...}]
InterpolatingPolynomial[{x1, ...}, {f1, df1, ...}, ...}, {x, ...}]
```

6.7.2. Пример трехмерной полиномиальной интерполяции

Первые две формы записи функций очевидны, и примеры на их применение уже приводились выше в этой главе. Третья форма задает возможность выполнения многомерной (в частности, трехмерной) интерполяции. Пример ее приведен на рис. 6.38. Поверхность синтезируется с помощью функций **Flatten** и **Table**, после интерполяции строится ее контурный график. Плавность линий графика говорит о достаточно высоком качестве интерполяции.

6.7.3. Полиномиальная интерполяция с заданием значений производной в узлах

Последняя форма функции позволяет задавать в узлах интерполяции не только значения функции f_i , но и значения производных df_i . Это существенно расширяет возможности интерполяции и аппроксимации. На рис. 6.39 показан пример полиномиальной аппроксимации для функции одной переменной x , представленной пятью ординатами и значениями x , по умолчанию равными 1, 2, 3, 4 и 5. При этом в четвертой точке задана не только ордината, но и значение производной, равное -1 .

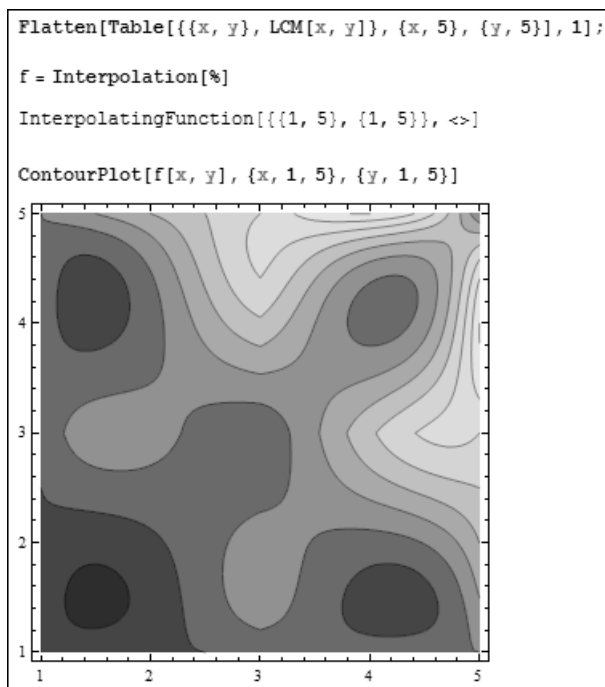


Рис. 6.38. Пример трехмерной интерполяции поверхности

Интересно отметить, что функция **InterpolatingPolynomial** выдает результат в форме Горнера. На рис. 6.39 показано превращение ее в обычную форму результирующего полинома и вычисление производной (оно оказалось равным заданному значению -1).

6.7.4. Функция нелинейной регрессии **FindFit**

Функция *нелинейной регрессии* **FindFit**, которая была и в системе Mathematica 5.2, в Mathematica 6 усовершенствовалась и стала вполне полноценным и простым средством для проведения нелинейной регрессии произвольного вида. Функция задается в виде:

FindFit[data,expr,pars,vars] FindFit[data,{expr,cons},pars,vars]

Здесь: data – список данных, expr – выражение, представляющее функцию регрессии, cons – ограничительные условия, pars – список искомых параметров, переменная или список переменных.

Пример проведения нелинейной регрессии с помощью функции **FindFit** с графической визуализацией регрессии путем построения функции регрессии и точек

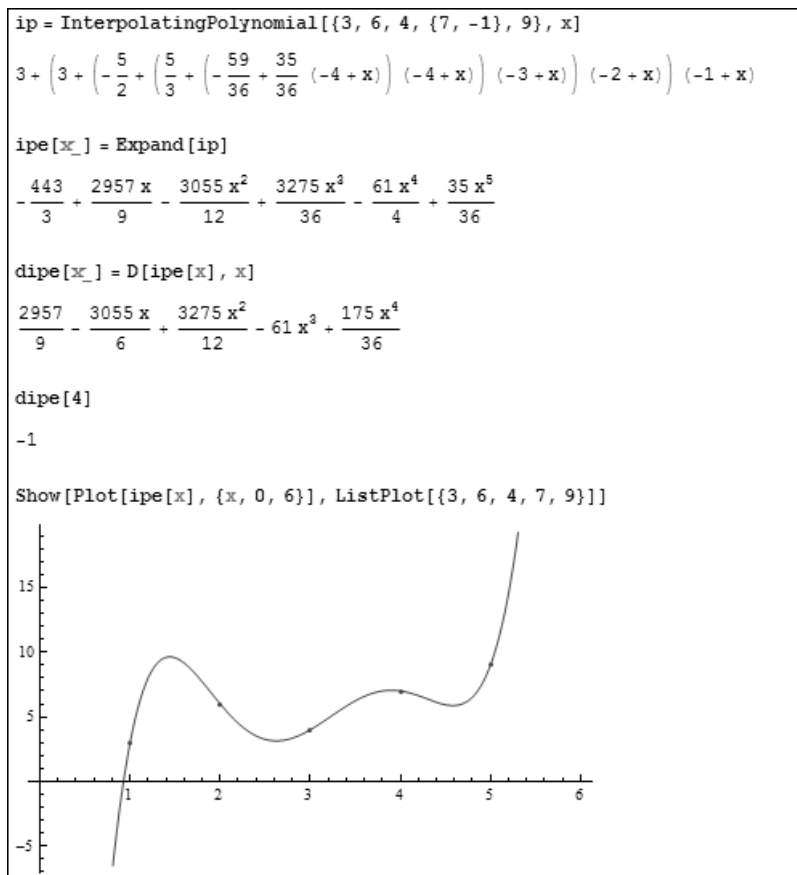


Рис. 6.39. Пример полиномиальной аппроксимации, в одной из узловых точек которой заданы ордината и значение производной

исходных данных показан на рис. 6.40. Хорошо видно, что кривая функции регрессии проходит «в облаке» исходных точек. При этом среднеквадратичное отклонение всех точек равно нулю.

6.8. Функции для работы со звуковыми сигналами

6.8.1. Роль синтеза звука

В системе Mathematica, в отличие от других систем компьютерной математики, имеются особые средства для *синтеза звука*. Сопровождение звуком описания некоторых математических закономерностей (например, биений, развития

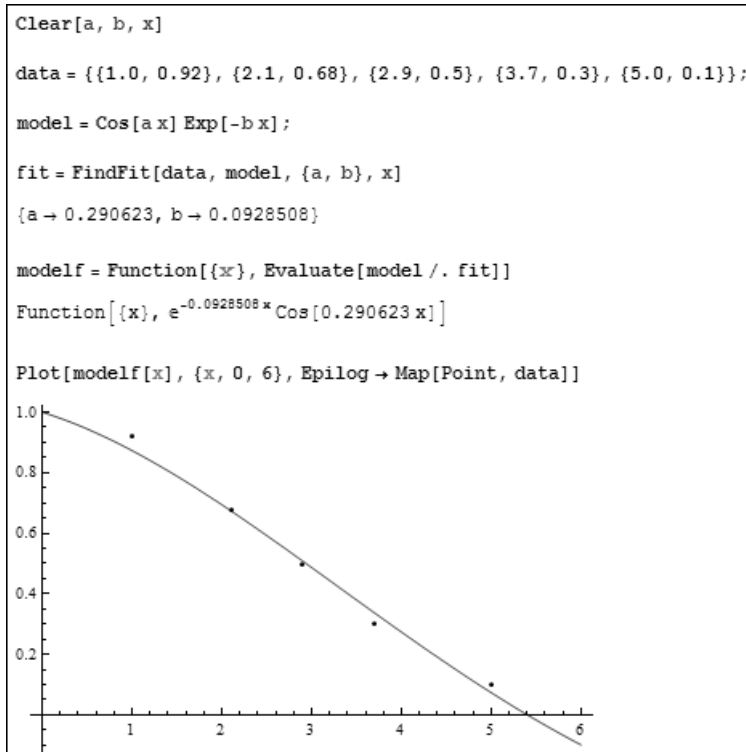


Рис. 6.40. Пример выполнения нелинейной регрессии

взрывных процессов и т.д.) делает это описание более понятным и естественным. Особенно удобна эта возможность в теоретической акустике и в технике аналоговой и цифровой обработки акустических сигналов.

Возможности синтеза звука становятся доступными, если компьютер оборудован звуковой картой класса Sound Blaster фирмы Creative Lab или совместимой с ней. К карте должна быть подключена стереофоническая акустическая система. Возможен синтез как монофонических, так и стереофонических звуков.

С синтезируемым звуком связан некоторый графический образ-ячейка. Он имеет вид осциллограмм звуковых сигналов по обоим стереоканалам. Если такая ячейка выделена, то возможен запуск воспроизведения звука с помощью главного или контекстного меню.

6.8.2. Функции для работы со звуком

Для синтеза звуков в системе Mathematica используются следующие функции:

- **ListPlay[{a1, a2, ...}]** – проигрывает звук с амплитудой, заданной последовательностью уровней a_i .

- **Play[f, {t, tmin, tmax}]** – воспроизводит звук с амплитудой, заданной f как функцией от времени t в секундах между значениями $tmin$ и $tmax$.
- **PlayRange** – опция для Play и родственных функций, указывающая, какой диапазон уровней звуковых амплитуд должен быть включен.
- **SampleDepth** – опция для звуковых примитивов, устанавливающая количество бит для кодирования уровней амплитуды звуковых сигналов.
- **SampledSoundFunction[f, n, r]** – звуковой примитив; воспроизводит звук с амплитудой, квантованной r раз в секунду, которая генерируется применением функции f к последовательным целым от 1 до n .
- **SampledSoundList[{a1, a2, ...}, r]** – звуковой примитив, воспроизводящий звук, амплитуда которого имеет уровни a_i с дискретностью r раз в секунду.
- **SampleRate** – опция для звуковых примитивов, устанавливающая дискретность воспроизведения звука в секунду.
- **Sound[primitives]** – представляет звук нот.
- **\$SoundDisplayFunction** – возвращает установочное значение по умолчанию для опции DisplayFunction в звуковых функциях.

Некоторые из указанных функций напоминают графические функции. И это не случайно. Идеология применения этих функций та же, что при использовании функций графики. Звуковые объекты имеют много схожего с графическими объектами, и их можно наряду с последними включать в различные функции-директивы. Таким образом, единство работы со звуком и графикой обеспечено специально.

6.8.3. Примеры синтеза звуков в Mathematica 5.1/5.2

Рисунок 6.41 показывает использование функции **Sound** для создания звукового объекта в системе Mathematica 5.1/5.2. Графически этот объект представляет собой двоянную «осциллограмму» звука. Слово «осциллограмма» не случайно взято в кавычки, на самом деле речь идет лишь о некотором графическом представлении звуковых сигналов, отдаленно напоминающем осциллограмму. К тому же вид этого отображения сильно зависит от компьютерной платформы, на которой установлена система Mathematica, и даже от применяемых в компьютере видеосредств.

Принятый в системе способ синтеза звуков имеет определенные недостатки. Звуковые средства системы слишком привязаны к математике: для задания звука необходимо описать звуковые колебания математической формулой.

6.8.4. Работа со звуком в Mathematica 6

В принципе, описанные выше функции есть и в новейшей Mathematica 6. Но все они существенно доработаны. На рис. 6.42 показан пример воспроизведения звука функцией **Play**, заданного математическим выражением – формулой. Нетрудно увидеть, что в Mathematica 6 введен виртуальный проигрыватель звуков, работа с которым вполне очевидна.

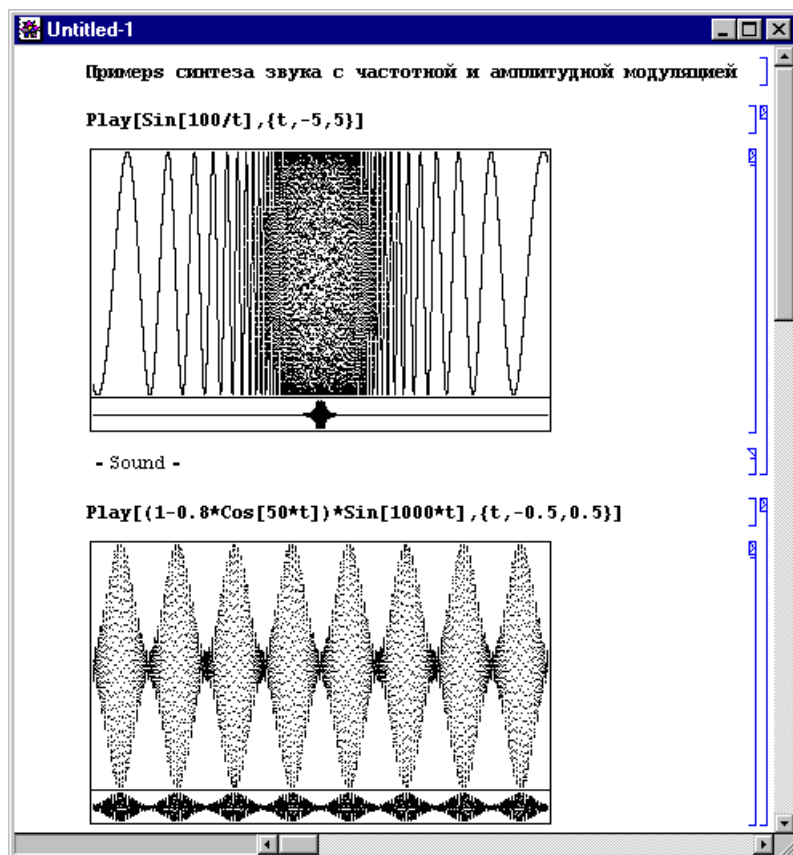


Рис. 6.41. Создание звукового объекта

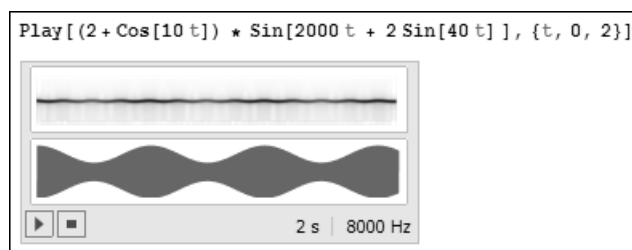


Рис. 6.42. Проигрывание звука, заданного математическим выражением

На рис. 6.43 показано проигрывание синтетического звука с помощью функции **ListPlay**. Задан список из 5000 звуковых отсчетов и скорость воспроизведения в 4096 отсчетов секунду.

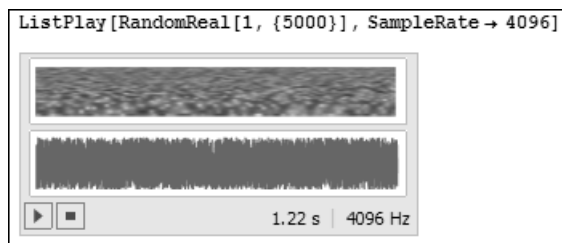


Рис. 6.43. Проигрывание звука, заданного списком отсчетов

С помощью функции **Sound** можно воспроизводить звуки нот. На рис. 6.44 показано воспроизведение целой мелодии, которая ассоциируется с работой клеточного автомата.

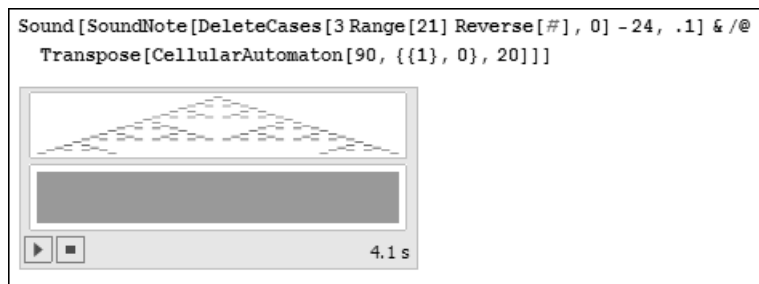


Рис. 6.44. Воспроизведение мелодии с помощью функции *Sound*

С помощью новой функции **SystemDialogInput** с аргументом «Record-Sound» можно вывести цифровой рекордер для записи реальных звуков с помощью микрофона (рис. 6.45). Работа с рекордером вполне очевидна. Отсчеты звука записываются в массив *data*.

Команда **Sound** способна воспроизводить полифонические звуки с помощью нескольких виртуальных инструментов. Пример этого представлен на рис. 6.46.

Новая команда повтора звука **Emit** может использоваться для повторения фрагмента звука, создаваемого функцией **Sound**. Пример ее применения совместно с командой **Sound** представлен на рис. 6.47.

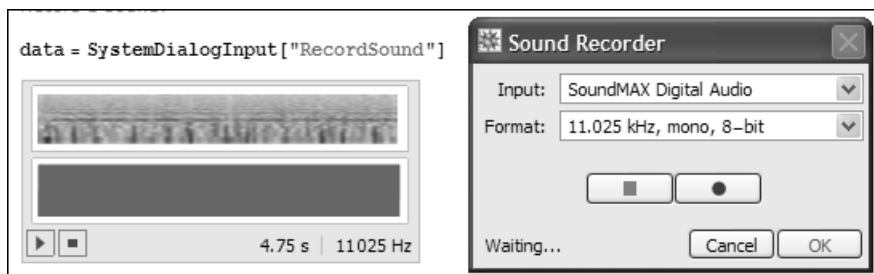


Рис. 6.45. Пример работы с цифровым рекордером звука

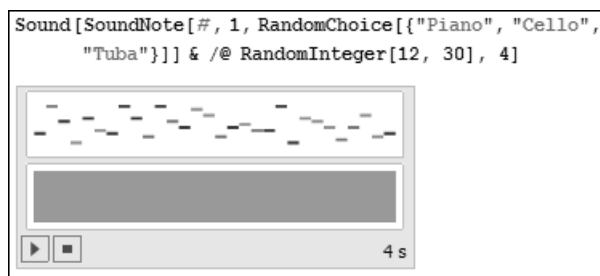


Рис. 6.46. Пример воспроизведения мелодии несколькими инструментами

При необходимости можно вывести несколько проигрывателей, воспроизводящих разные звуки. Так, на рис. 6.48 показан вывод двух проигрывателей с помощью команды-функции **Sound**.

В справке по звуковым средствам Mathematica 6 можно найти многочисленные дополнительные данные по работе со звуками: допустимые форматы файлов, детали их импорта и экспорта, перечень названий виртуальных инструментов, название тестовых файлов с мелодиями и звуками и т.д.

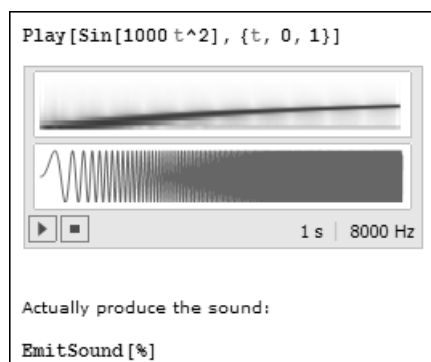


Рис. 6.47. Применение команды **Emit** для повтора звука, созданного командой **Sound**

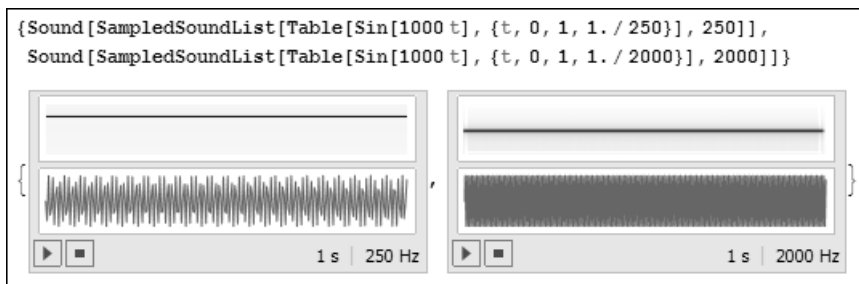


Рис. 6.48. Пример вывода двух проигрывателей звука

6.9. Функции для работы с потоками и файлами

6.9.1. Потоки и файлы

Система Mathematica имеет развитые средства для работы с потоками (streams) и файлами (files). Под *поток*м подразумевается непрерывная последовательность данных, циркулирующих внутри компьютера. Обмен потоками происходит практически непрерывно, например, при вводе поток ввода поступает от клавиатуры в компьютер, при печати поток данных поступает от компьютера в принтер через порт принтера и т.д.

Файлом является упорядоченная структура данных, имеющая имя и хранящаяся на каком-либо носителе, чаще всего на магнитном диске. Файлы могут иметь различные форматы и различный тип доступа к хранимой на них информации. Наиболее распространенные в системе Mathematica файлы документов являются файлами с последовательным доступом и имеют текстовый формат.

Последовательный доступ означает, что информация из открытого файла может быть считана строго последовательно от его начала до конца, отмеченного специальной меткой. Это напоминает считывание с ленты магнитофонной кассеты. *Текстовый формат* означает, что все данные записаны в виде ASCII-кодов. Следовательно, прочесть такой файл можно с помощью любого текстового редактора, работающего с текстами в виде ASCII-кодов.

6.9.2. Упрощенная работа с файлами

Прежде чем рассматривать весьма обширные возможности системы по работе файлами в целом, отметим упрощенный прием вызова файла с помощью двойного символа <<:

<<filename

Эта команда считывает файл с указанным именем *filename* и заносит в память компьютера содержащиеся в нем определения. Имя файла необходимо ука-

зывать полное, то есть вместе с расширением. Исключением является случай, когда файл находится в основном каталоге системы. Эта команда эквивалентна функции

```
Get["filename", key]
```

Для записи объекта (переменной, массива, списка и т.д.) в файл служат упрощенные команды:

- **expr >> filename** — передает значение **expr** в файл с заданным именем;
- **expr >>> filename** — добавляет **expr** в конец файла с заданным именем.

6.9.3. Обычные средства для работы с файлами

Указанные выше команды по существу есть укороченные (и потому более удобные) формы следующих функций:

- **Get["filename", "key"]** — читает файл, который закодирован функцией Encode с использованием ключа "key";
- **GetContext["context"]** — загружает файл с заданным контекстом;
- **Put[expr1, expr2, ..., "filename"]** — записывает последовательность выражений **expr**i в файл с именем **filename**;
- **PutAppend[expr1, expr2, ..., "filename"]** — присоединяет последовательность выражений **expr**i к файлу с именем **filename**.

Еще одна упрощенная функция — **!! filename** — выводит содержимое файла с заданным именем.

Следующие примеры показывают запись выражения в файл C:/myfile, его считывание, затем добавление в файл еще одного выражения и контроль файла после этого:

```
Expand[(x + y)^2] >>> C:/myfile
!!C:/myfile
      a^3 + 3*a^2*b + 3*a*b^2 + b^3*x^2 + 2*x*y + y^2
<<Graphics`Arrow`
Names["Graphics`Arrow`*"]
{Absolute, Arrow, HeadCenter, HeadLength, HeadScaling, HeadShape,
HeadWidth, Relative, ZeroShape}
```

Такая форма вызова особенно удобна для вызова файлов пакетов расширений и применений системы. Имя файла указывается по правилам, принятым в MS-DOS. Необходимо внимательно следить за вводом упрощенных символов. Некоторые символы неравенств очень похожи, но могут отличаться по действию.

Файлы пакетов применений имеют расширение .m. Мы уже приводили примеры использования определений, содержащихся в файлах пакетов расширения системы.

Имеется еще ряд функций для работы с файлами:

- **ReadList["filename"]** — читает все оставшиеся в файле "filename" выражения и возвращает их в виде списка;

- **ReadList["filename", type]** — читает из файла "filename" объекты указанного типа type до конца файла. Возвращает список считанных объектов;
- **ReadList["filename", {type1, type2,...}]** — читает объекты указанных типов type_i до конца файла filename;
- **ReadList["filename", types, n]** — читает только первые n объектов указанных типов types из файла filename;
- **Save["filename", x1, x2,...]** — создает файл с заданным именем filename, содержащий значения переменных x1, x2,...;
- **!command** — исполняет заданную команду операционной системы.

Для загрузки файлов пакетов расширений (AddOn) используются функции, позволяющие задать контекст файлов (подробнее о контекстах см. в Главе 2):

- **Needs["context`", "filename"]** — загружает файл, если указанный контекст отсутствует в списке загруженных;
- **Needs["context`"]** — загружает файл, имя которого определяется с помощью функции ContextToFilename["context`"], если указанный контекст отсутствует в списке загруженных.

Загрузка файлов с указанием их контекстов позволяет избежать конфликтов между разными пакетами расширения, используемыми одновременно.

6.9.4. Использование файлов других языков программирования

Из функций для работы с файлами особо следует отметить следующую функцию-директиву:

- **Splice["file.mx"]** — вставляет в файлы на других языках программирования вычисленные выражения системы Mathematica, которые должны быть записаны в скобках вида <* и *>;
- **Splice["infile", "outfile"]** — читает файл infile, интерпретирует фрагменты, содержащиеся между скобками <* и *> и записывает результат в файл outfile.

Эта возможность особенно существенна при использовании программ на языках программирования C (расширение .mc), Fortran (расширение .mf) и TeX (расширение .mtex), для форматов которых Mathematica имеет средства конвертирования выражений (функции **CForm**, **FortranForm** и **TeXForm** соответственно). Таким образом, имеется возможность экспорта выражений системы Mathematica в программы, составленные на этих языках.

Поясним применение функции-директивы **Splice**. Пусть имеется экспортированная программа на языке C, которая должна рассчитывать численное значение некоторого интеграла, и мы хотим получить формулу для этого интеграла средствами системы Mathematica. Допустим, она представлена файлом demo.mc. Его можно просмотреть следующим образом:

```
!!demo.mc
#include "mdefs.h"
double f(x)
double x;
{
double y;
y = <* Integrate[Sin[x]^5, x] *> ;
return(2*y - 1) ;
}
```

После исполнения функции `Splice["demo.mc"]` программа будет записана в файл `demo.c`, в котором выражение в скобках `<...**>` заменено вычисленным значением интеграла (в форме **CForm**). Файл при этом будет выглядеть так:

```
!!demo.c
#include "mdefs.h"
double f(x)
double x;
{
double y;
y = -5*Cos(x)/8 + 5*Cos(3*x)/48 - Cos(5*x)/80 ;
return(2*y - 1) ;
}
```

6.9.5. Запись в файл определений

Из простых функций, обеспечивающих создание файлов с заданными определениями, следует отметить также функцию `Save`:

Save["filename", symb1, symb2,...]

Она добавляет определения символов `symbi` к файлу `filename` (возможны упрощенные формы `Save`).

Приведем пример ее использования:

```
Clear[x,y,a]
f[x_]=Sin[x]+y
y+Sin[x]
y=a
a
Save["demo1",f]
!!demo1
f[x_] = y + Sin[x]
```

6.9.6. Другие функции для работы с файлами

В целом средства системы `Mathematica` обеспечивают возможности работы с различными файлами, присущие `MS-DOS`, без выхода из среды системы. Важное место занимают функции, дающие информацию о директориях, файлах и потоках. К ним относятся следующие функции:

- **Directory[]** — возвращает текущий рабочий каталог;
- **DirectoryStack[]** — возвращает содержимое стека каталогов, которое представляет последовательность используемых в текущем сеансе каталогов;
- **\$Display** — возвращает список файлов и каналов (*pipes* — канал или абстрактный файл), используемый функцией вывода `$DisplayFunction` по умолчанию;
- **FileByteCount["filename"]** — возвращает количество байтов в файле;
- **FileDate["filename"]** — возвращает дату и время последней модификации файла в виде списка;
- **FileInformation["filename"]** — возвращает информацию о файле;
- **FileNames[]** — приводит список всех файлов в текущем рабочем каталоге;
- **FileNames["form"]** — перечисляет все файлы в текущем рабочем каталоге, чьи имена совпадают с шаблоном `form`;
- **FileNames[{"form1", "form2", ...}]** — перечисляет все файлы, чьи имена соответствуют любому из шаблонов `formi`;
- **FileNames[forms, {"dir1", "dir2", ...}]** — перечисляет файлы с именами, соответствующими шаблонам `forms`, в любом из указанных каталогов `diri`;
- **FileType["filename"]** — возвращает тип файла: `File`, `Directory` или `None` (если указанного файла не существует);
- **\$HomeDirectory** — дает имя «домашней» директории пользователя;
- **\$Output** — дает список файлов и каналов, в которые направляется стандартный вывод системы `Mathematica`.
- **ParentDirectory[]** — возвращает имя родительского каталога для текущего рабочего каталога;
- **ParentDirectory["dir"]** — возвращает имя родительского каталога для каталога `dir`;
- **\$Path** — дает список каталогов для просмотра при попытке поиска внешнего файла;
- **StreamPosition[stream]** — возвращает целое число, которое указывает позицию текущей точки в открытом потоке `stream`;
- **Streams[]** — возвращает список всех потоков, открытых в данный момент;
- **Streams["name"]** — перечисляет только потоки с указанным именем `name`.

Приведенные ниже примеры иллюстрируют использование большинства из этих достаточно простых функций:

```
Directory[]
C:\m3\PC
DirectoryStack[]
{}
$Display
```

```

stdout
FileByteCount["C:.val"]
46
FileDate["C:.val"]
{1999,8,3,16,4,44}
FileInformation["C:.val"]
{File→C:\ma.val,FileType→File,Date→3142685084,ByteCount→46}
FileNames[]
{Examples,FILES,MATHEMATICA.EXE,MATH.EXE,MATHINSTALLER.EXE,MATHKERNEL.EXE}
FileType["C:.val"]
File
HomeDirectory[]
c:\
$Output
{OutputStream[stdout,1]}
ParentDirectory[]
C:\m3
Streams[]
{OutputStream[stdout,1],OutputStream[stderr,2]}

```

Обширный набор функций файловых операций облегчает программирование задач по обмену файлами между системой Mathematica и другими программами и внешними устройствами.

6.10. Системные функции

6.10.1. Функции времени и даты

Для управления системой в процессе вычислений служат системные директивы и функции. Некоторые из них широко используются при программировании решения прикладных задач, другие служат в основном для контроля над системой. Имена многих, вспомогательных с точки зрения конечного пользователя, системных функций начинаются с символа \$. Ниже описаны основные системные функции.

Ряд системных функций служит для получения информации о времени и текущей дате:

- **AbsoluteTime[]** — возвращает полное количество секунд, прошедших с момента 1 января 1900 г.;
- **\$CreationDate** — возвращает дату и время создания используемой версии системного ядра Mathematica;
- **Date[]** — возвращает текущее значение даты и времени в виде {год, месяц, день, час, минута, секунда};
- **FromDate[date]** — превращает дату date вида {год, месяц, день, час, минута, секунда} в число секунд, прошедших с 1 января 1900 г.
- **TimeUsed[]** — возвращает полное количество секунд процессорного времени, использованного на данный момент в текущем сеансе Mathematica;

- **\$TimeUnit** — возвращает минимальный временной интервал в секундах, который можно зарегистрировать в вашей компьютерной системе;
- **TimeZone[]** — возвращает часовой пояс, установленный для вашей компьютерной системы;
- **Timing[expr]** — вычисляет `expr` и возвращает список, состоящий из значения затраченного времени и результата вычислений;
- **ToDate[time]** — преобразует абсолютное время в секундах, прошедшее с 1 января 1900 г., в дату вида {год, месяц, день, час, минута, секунда}.

Следующие примеры иллюстрируют применение некоторых из этих функций.

Ввод (In)	Вывод (Out)
AbsoluteTime[]	3.2747172031900000×10 ⁹
Date[]	{2003, 10, 9, 19, 37, 28.4600000}
FromDate[{2000, 7, 15, 4, 51, 30}]	3172625490
SessionTime[]	856.4000000
TimeUsed[]	0.49

Их действие вполне очевидно и не требует комментариев. Обратите, однако, внимание на то, что многие эти функции у вас будут давать иные результаты, зависящие от даты и скорости вычислений конкретного компьютера.

6.10.2. Общесистемные функции

Ниже представлены некоторые функции общесистемного характера:

- **\$Aborted** — возвращает сообщение о прекращении вычислений при их прерывании функцией **Abort[]**;
- **AbortProtect[expr]** — вычисляет `expr`, запоминая все попытки прерывания, но не выполняя их до тех пор, пока не будет завершено вычисление, либо пока не будет вызвана процедура **CheckAbort**;
- **Accuracy[x]** — указывает число цифр в числе `x` после десятичной точки, которое используется при вычислениях;
- **ByteCount[expr]** — возвращает число байт, которое используется для представления выражения `expr`;
- **Environment["var"]** — возвращает значение переменной окружения операционной системы с именем `"var"`;
- **\$Line** — глобальная переменная, указывающая номер текущей строки ввода;
- **\$MachineEpsilon** — возвращает *машинную точность представления* — наименьшее число, которое, будучи прибавлено к 1.0, даст результат, отличный от 1.0;
- **\$MachineID** — строка, которая возвращает, если возможно, уникальный код идентификации применяемого компьютера;
- **\$MachineName** — строка, возвращающая имя, которое присвоено используемому компьютеру, если такое имя определено;

- **\$MachinePrecision** — возвращает количество десятичных знаков точности представления чисел;
- **\$MachineType** — строка, возвращающая общий тип компьютера, на котором запущена система Mathematica;
- **\$MinMachineNumber** — наибольшее *машинно-представимое* число, которое может применять данная компьютерная система;
- **\$MaxNumber** — возвращает наибольшее из представимых в системе Mathematica чисел;
- **\$MinMachineNumber** — наименьшее положительное *машинно-представимое* число, которое может применять данная компьютерная система;
- **\$MinNumber** — возвращает наименьшее (положительное) представимое в системе Mathematica число;
- **\$OperatingSystem** — строка, дающая тип операционной системы, под управлением которой работает Mathematica;
- **Pause[n]** — выдерживает паузу не менее n секунд;
- **\$ReleaseNumber** — целое число, которое дает младший номер версии ядра данной системы Mathematica;
- **\$Remote** — имеет значение True, если Mathematica применяется в дистанционном режиме или с программным препроцессором, иначе — значение False;
- **\$SessionID** — уникальный номер, который присвоен данному сеансу системы Mathematica;
- **SessionTime[]** — возвращает полное число секунд реального времени, прошедшего с момента начала вашего сеанса работы в системе Mathematica;
- **\$System** — представляет собой строку с указанием типа используемой компьютерной системы;
- **\$Version** — символьная строка, которая представляет используемую версию системы Mathematica;
- **\$VersionNumber** — вещественное число, которое дает полный номер текущей версии системного ядра Mathematica.

Ниже приведены примеры использования ряда общесистемных функций для версии Mathematica 6.

Ввод (In)	Вывод (Out)
Accuracy [12.34]	14.8633
ByteCount [Exp[x]^2/a]	160
\$Version	6.0 for Microsoft Windows (32-bit) (April 28, 2007)
\$System	Microsoft Windows (32-bit)
\$OperatingSystem	Windows
\$MachineEpsilon	2.22045 10^{-16}
\$MaxMachineNumber	1.79769×10^{308}
\$MinMachineNumber	2.22507 10^{-308}
\$MachinePrecision	15.9546
\$Packages	{JLink`, PacletManager`, WebServices`, System`, Global`}

Системная функция **\$Path** выводит лист каталогов файловой системы текущей системы Mathematica.

Приведенные примеры показывают, что благодаря системным функциям можно извлечь достаточно полную информацию о текущих параметрах системы и использовать ее для создания специальных алгоритмов вычислений (например, для генерации последовательности псевдослучайных чисел со случайной базой, заданной системным временем) или организации развитого диалога с системой.

6.10.3. Общесистемные функции в Mathematica 6

Число системных функций в Mathematica 6 заметно возросло. Появились, например, функции для обеспечения параллельных вычислений и многоядерных процессоров. Естественно сохранились описанные выше функции.

У некоторых системных функций заметно улучшен выход и возросло количество информации. Например, функция `SystemInformation` без параметров вызывает появление окна с рядом вкладок, содержащих детальную информацию о компьютере и его системном программном обеспечении (см. рис. 6.49).

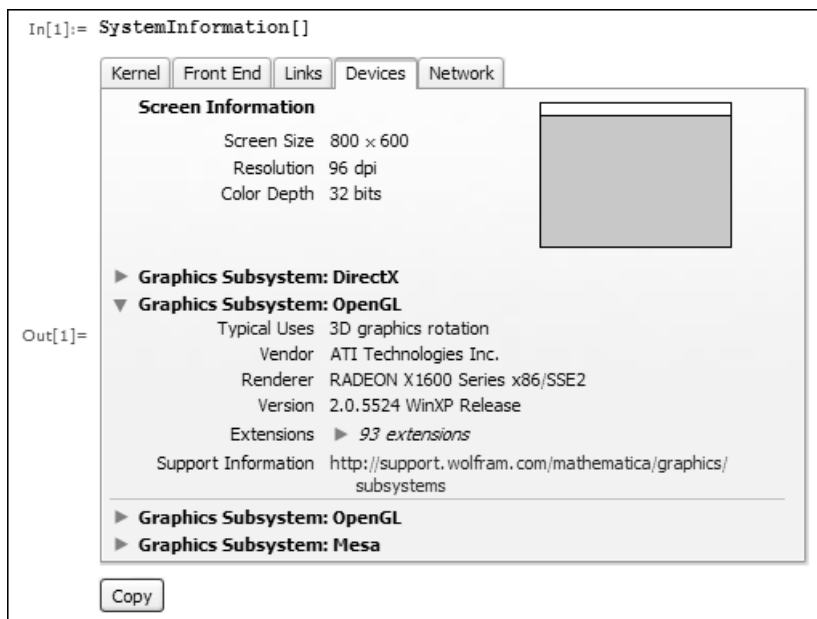


Рис. 6.49. Получение информации о компьютере и его системном программном обеспечении

6.11. Функции статистической обработки данных и массивов **Statistics**

6.11.1. Назначение пакета **Statistics** в **Mathematica 5.1/5.2**

В ядре системы Mathematica 5.1/5.2 практически нет статистических функций. Зато пакет расширения Statistics для систем Mathematica 5.1/5.2 имеет множество функций, охватывающих практически все расчеты теоретической и прикладной статистики.

Пакет расширения Statistics для систем Mathematica 5.1/5.2 содержит следующие подпакеты:

- ConfidenceIntervals – функции доверительных интервалов;
- ContinuousDistributions – функции непрерывных распределений;
- DataManipulation – манипуляции с данными;
- DataSmoothing – сглаживание данных;
- DescriptiveStatistics – статистика распределений;
- DiscreteDistributions – функции дискретных распределений;
- HypothesisTests – проверка статистических гипотез;
- LinearRegression – линейная регрессия (см. Главу 6);
- MultiDescriptiveStatistics – статистика множественных распределений;
- MultinormalDistribution – функции множественных нормальных распределений;
- NonlinearFit – нелинейная регрессия (см. Главу 6);
- NormalDistribution – функции нормального распределения;
- Common – данные общего характера.

Как и ранее, для работы каждого из подпакетов требуется его загрузка в память компьютера с помощью команды

```
>>Statistics`Имя_подпакета`
```

Имена подпакетов расширения статистики приведены выше.

6.11.2. Манипуляции с данными – **DataManipulation**

Статистические данные обычно бывают представлены в виде списков данных – как одномерных, так и двумерных (таблиц и матриц) и даже многомерных. Большая часть функций, обеспечивающих манипуляции с данными, сосредоточена в подпакете DataManipulation.

Данные могут вводиться в строках ввода или считываться из файлов с помощью функции **ReadList**. Для манипуляций с данными могут использоваться многие функции ядра системы, описанные ранее, в частности, все функции обра-

ботки списков. Подпакет **DataManipulation** дает ряд удобных функций. Ниже представлена первая группа таких функций:

- **Column[data,n]** – возвращает n-ый столбец списка data.
- **Column[data,{n1,n2,...}]** – возвращает список из ni столбцов списка данных.
- **ColumnTake[data,spec]** – возвращает столбцы списка data с данной спецификацией spec.
- **ColumnDrop[data,spec]** – удаляет столбцы списка data с данной спецификацией.
- **ColumnJoin[data1,data2,...]** – объединяет столбцы списков datai.
- **RowJoin[data1,data2,...]** – объединяет строки списков datai.
- **DropNonNumeric[data]** – удаляет из списка data нечисловые элементы.
- **DropNonNumericColumn[data]** – удаляет из списка data столбцы с нечисловыми элементами.

Примеры применения этих функций:

```
<<Statistics`DataManipulation`
data = {{a, 7}, {b, 2}, {c, 3}, {d, w},
        {e, 5}, {f, 6}}
{{a,7},{b,2},{c,3},{d,w},{e,5},{f,6}}
col2 = Column[data, 2]
{7,2,3,w,5,6}
newdata = DropNonNumeric[col2]
{7,2,3,5,6}
```

Полезны также следующие функции подпакета:

- **BooleanSelect[list,sel]** – удаляет из list элементы, которые дают True при тестировании sel.
- **TakeWhile[list,pred]** – удаляет из list все элементы, начиная с того, для которого pred дает True.
- **LengthWhile[list,pred]** – возвращает число элементов, которые были удалены после того, как pred дало значение True (отсчет с начала листа).

Примеры на применение этих функций:

```
TakeWhile[col2, NumberQ]
{7 ,2 ,3}
LengthWhile[col2, NumberQ]
3
```

Ряд функций служит для подготовки данных с целью построения гистограмм:

- **Frequencies[list]** – готовит данные для представления частотной гистограммы.
- **QuantileForm[list]** – дает отсортированные данные для представления квантилей.
- **CumulativeSums[list]** – дает кумулятивное суммирование данных листа.

Пример построения гистограммы по данным списка из двойных элементов с помощью функции **Frequencies** дан на рис. 6.50. Для построения графика при этом использована функция **BarChart** из пакета расширения Graphics. В данном

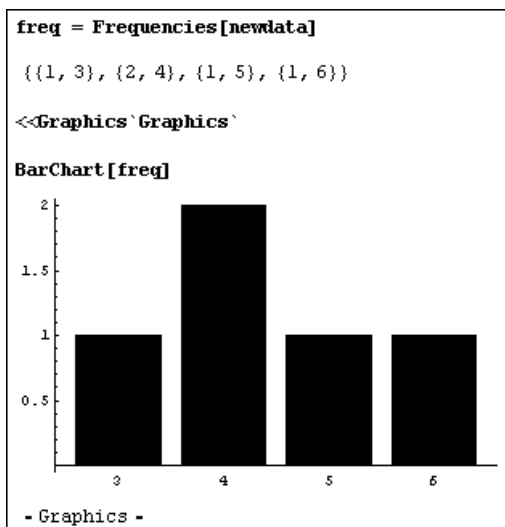


Рис. 6.50. Пример построения гистограммы по данным функции *Frequencies*

случае список задает пары значений высот столбиков гистограммы (число попаданий данных в столбик) и расположение столбиков (или их порядковый номер).

Для подготовки гистограмм могут использоваться и следующие функции:

BinCounts [data,{min,max,dx}]	RangeCounts [data,{c1,c2,...}]
CategoryCounts [data,{e1,e2,...}]	BinLists [data,{min,max,dx}]
RangeLists [data,{c1,c2,...}]	CategoryLists [data,{e1,e2,...}]

С достаточно простыми примерами их работы можно ознакомиться по справочной системе Mathematica с описанием данного подпакета.

6.11.3. Стандартная обработка массива данных

В подпакете *DescriptiveStatistics* сосредоточены наиболее важные и общеизвестные функции по статистической обработке массивов (списков) данных:

- **CentralMoment**(data,r) – возвращает центральный момент данных data порядка r.
- **Mean**[data] – возвращает среднее значение данных data.
- **MeanDeviation**[data] – возвращает среднее отклонение данных.
- **Median**[data] – возвращает центральное значение (медиану) данных.
- **MedianDeviation**[data] – возвращает абсолютное отклонение (от медианы) данных.
- **Skewness**[data] – возвращает коэффициент асимметрии данных (эксцесс).

- **StandardDeviation[data]** – возвращает стандартное отклонение данных.
- **GeometricMean[data]** – возвращает геометрическое среднее данных.
- **HarmonicMean[data]** – возвращает гармоническое среднее данных.
- **RootMeanSquare[data]** – возвращает среднеквадратическое значение.
- **Quantile[data,q]** – возвращает q-ый квантиль.
- **InterpolatingQuantile[data,q]** – возвращает q-ый квантиль, используя при вычислениях интерполяцию данных.
- **VarianceData[data]** – возвращает среднеквадратическое отклонение данных.

Мы не приводим определений этих функций, поскольку при символьных списках **data** их легко получить именно в том виде, который реализован в системе Mathematica:

```
<<Statistics`DescriptiveStatistics`
```

```
ds={x1,x2,x3}
```

```
{ x1, x2, x3 }
```

```
Mean[ds]
```

$$\frac{1}{3} (x1 + x2 + x3)$$

```
MeanDeviation[ds]
```

$$\frac{1}{3} \left(\text{Abs} \left[x1 + \frac{1}{3} (-x1 - x2 - x3) \right] + \right. \\ \left. \text{Abs} \left[x2 + \frac{1}{3} (-x1 - x2 - x3) \right] + \right. \\ \left. \text{Abs} \left[x3 + \frac{1}{3} (-x1 - x2 - x3) \right] \right)$$

```
Variance[ds]
```

$$\frac{1}{2} \left(\left(x1 - \frac{1}{3} (x1 + x2 + x3) \right)^2 + \right. \\ \left. \left(x2 - \frac{1}{3} (x1 + x2 + x3) \right)^2 + \left(x3 - \frac{1}{3} (x1 + x2 + x3) \right)^2 \right)$$

```
Skewness[ds]
```

$$\left(\sqrt{3} \left(\left(x1 + \frac{1}{3} (-x1 - x2 - x3) \right)^3 + \right. \right. \\ \left. \left(x2 + \frac{1}{3} (-x1 - x2 - x3) \right)^3 + \right. \\ \left. \left. \left(x3 + \frac{1}{3} (-x1 - x2 - x3) \right)^3 \right) \right) / \\ \left(\left(x1 + \frac{1}{3} (-x1 - x2 - x3) \right)^2 + \right. \\ \left(x2 + \frac{1}{3} (-x1 - x2 - x3) \right)^2 + \\ \left. \left(x3 + \frac{1}{3} (-x1 - x2 - x3) \right)^2 \right)^{3/2}$$

Следующие примеры поясняют действие этих функций при обработке уже численных данных:

```
data:={10.1,9.6,11,8.2,7.5,12,8.6,9}
CentralMoment[data,2]
1.9525
Mean[data]
9.5
MeanDeviation[data]
1.175
Median[data]
9.3
MedianDeviation[data]
0.95
Skewness[data]
0.374139
StandardDeviation[data]
1.4938
GeometricMean[data]
9.39935
HarmonicMean[data]
9.30131
RootMeanSquare[data]
9.60221
Quantile[data,1]
12.
InterpolatingQuantile[data,1]
InterpolatingQuantile[{10.1,9.6,11,8.2,7.5,12,8.6,9},1]
Variance[data]
2.23143
```

В этом случае результатом действия функций обычно являются числа в формате плавающей точки. С рядом других, менее распространенных, функций этого подпакета можно ознакомиться с помощью справочной системы. Там же даны примеры на их применение.

6.11.4. Линейное сглаживание данных и их фильтрация

В подпакете DataSmoothing определены функции для *сглаживания данных*, имеющих большой случайный разброс. К таким данным обычно относятся результаты ряда физических экспериментов, например, по энергии элементарных частиц, или сигналы, поступающие из космоса. Для того, чтобы выделить полезную информацию из таких данных с большим уровнем шумов, и применяется процедура сглаживания. Она может быть линейной (например, усреднение по ряду точек) или нелинейной (например, экспоненциальной).

В данном подпакете определены следующие функции сглаживания:

- **MovingAverage[data,r]** – сглаживание данных **data** методом усреднения для **r** точек.

- **MovingMedian[data,r]** – сглаживание данных **data** по медиане для **r** точек (опция RepeatedSmoothing->True используется для повторного сглаживания).
- **LinearFilter[data,{c₀,c₁,...,c_{r-1}}]** – линейная фильтрация (**c_j** – весовые множители).

Применение сглаживания усреднением иллюстрирует рис. 6.51. На нем задан массив (таблица) из 500 случайных точек с равномерным распределением и создан графический объект из этих точек в виде кружков малого диаметра. Затем выполнена операция сглаживания (по 12 смежным точкам) и создан графический объект сглаженных точек в виде кружков большего диаметра. Для сопоставления оба объекта построены на одном графике функцией **Show**.

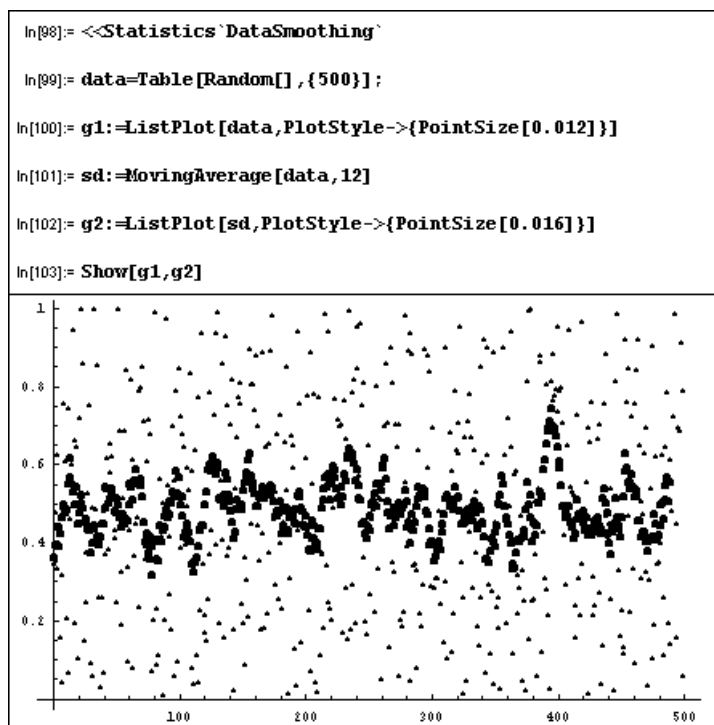


Рис. 6.51. Пример линейного сглаживания данных из 500 точек

Нетрудно заметить, что сглаженные точки группируются вокруг среднего значения, равного 0.5, тогда как исходные точки разбросаны практически равномерно по всему полю рисунка. Остальные функции сглаживания можно использовать аналогичным образом.

6.11.5. Экспоненциальное сглаживание

Для нелинейного экспоненциального сглаживания имеется следующая функция:

ExponentialSmoothing[data,a] – дает экспоненциальное (нелинейное) сглаживание с параметром **a**, задающим степень сглаживания.

Эффективность нелинейного (экспоненциального) сглаживания демонстрирует рис. 6.52, документ которого построен по тому же принципу, что и показанный на рис. 6.51. Нетрудно заметить, что нелинейное сглаживание в данном случае явно более эффективно.

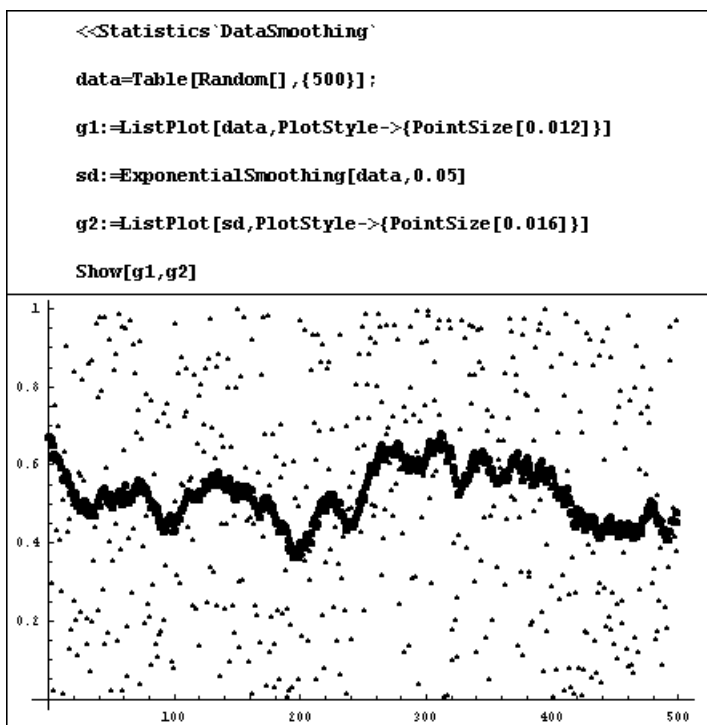


Рис. 6.52. Пример экспоненциального сглаживания

Выбор метода сглаживания зависит от решаемых пользователем задач и остается за ним. Сильное сглаживание ведет к потере быстрых компонент сигнала. А сравнение рис. 6.51 и 6.52 показывает, что для одного и того же массива случайных чисел вид сглаженной зависимости получается разным, что свидетельствует о необходимости очень внимательного отношения к применению этой операции.

6.11.6. Функции непрерывного распределения вероятностей

Подпакет NormalDistribution содержит хорошо известные функции *нормального распределения* вероятностей и родственные им функции следующих распределений:

- **NormalDistribution[*mu*,*sigma*]** – нормальное распределение.
- **StudentT Distribution[*r*]** – Т-распределение Стьюдента.
- **ShiSquareDistribution[*r*]** – хи-квадрат распределение.
- **FRatioDistribution[*r1*,*r2*]** -F-распределение.

Для этих и многих других непрерывных распределений заданы также функции плотности распределения, среднего, среднеквадратического отклонения, стандартного отклонения, вычисления коэффициента асимметрии и др. Рисунок 6.53 иллюстрирует получение выражения для плотности нормального распределения pdf и получения графика плотности этого распределения со смещенной вершиной.

Аналогичным образом легко построить графики других функций распределения.

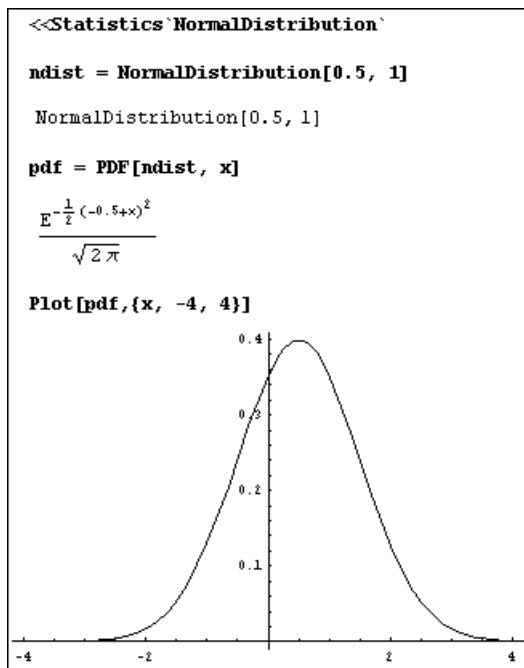


Рис. 6.53. Пример работы с функцией нормального распределения

Целый ряд функций, подобных выше описанным, задан и в пакете Continuius-Distribution для ряда функций непрерывного распределения. Прежде всего, это функции плотности вероятности непрерывного распределения:

- **BetaDistribution**[α, β] – непрерывное бета-распределение;
- **CauchyDistribution**[a, b] – распределение Коши;
- **ChiSquareDistribution**[n] – Хи-квадрат распределение;
- **ExponentialDistribution**[γ] – экспоненциальное распределение;
- **ExtremeValueDistribution**[α, β] – распределение Фишера-Типпета;
- **FratioDistribution**[$n1, n2$] – F-распределение;
- **GammaDistribution**[α, γ] – гамма-распределение;
- **NormalDistribution**[μ, σ] – нормальное распределение (Гауссиана);
- **LaplaceDistribution**[μ, β] – Лапласа (двойное экспоненциальное) распределение;
- **LogNormalDistribution**[μ, σ] – логарифмическое нормальное распределение;
- **LogisticDistribution**[μ, β] – логистическое распределение;
- **RayleighDistribution**[σ] – Релея распределение;
- **StudentDistribution**[n] – распределение Стьюдента;
- **UniformDistribution**[\min, \max] – равномерное распределение;
- **WeibullDistribution**[α, β] – распределение Вейбулла.

Для всех этих законов (**dist**) определен ряд статистических функций. Из них наиболее важными являются следующие функции:

- **PDF**[**dist**, x] – возвращает значение плотности вероятности распределения при аргументе x и заданном законе распределения **dist**.
- **CDF**[**dist**, x] – возвращает значение кумулятивной функции распределения-интеграла от функции плотности распределения.
- **Quantile**[**dist**, q] – возвращает q -ый квантиль для заданного закона распределения.
- **Domain**[**dist**] – возвращает пределы значений переменной.
- **Mean**[**dist**] – возвращает среднее значение данных **data**.
- **Varians**[**dist**] – возвращает дисперсию для заданного закона распределения.
- **StandardDeviation**[**dist**] – возвращает стандартное отклонение для заданного закона распределения.
- **Skewness**[**dist**] – возвращает коэффициент асимметрии для заданного закона распределения.
- **Kurtosis**[**dist**] – возвращает значение коэффициента эксцесса.
- **KurtosisExess**[**dist**] – возвращает эксцесс (остроту пика).
- **CharacteristicFunction**[**dist**, t] – возвращает характеристическую функцию $\phi(t)$ для заданного закона распределения.
- **ExpextedValue**[**f**,**dist**] – возвращает ожидаемое значение чистой функции **f**.
- **ExpextedValue**[**f**,**dist**, x] – возвращает ожидаемое значение чистой функции **f** в точке x .
- **Random**[**dist**] – создает псевдослучайное число с заданным распределением.
- **RandomArray**[**dist**, n] – создает массив с размерностью **dims** псевдослучайных чисел с заданным распределением.

Ниже представлены примеры применения некоторых из этих функций для экспоненциального закона распределения:

```
<<Statistics`ContinuousDistributions`
edist = ExponentialDistribution[1]
ExponentialDistribution[1]
CDF[edist, 10]

$$1 - \frac{1}{e^{10}}$$

cdffunction = CDF[edist, x]

$$1 - e^{-x}$$

Random[edist]
1.75426
Random[edist]
0.499882
RandomArray[gdist, 10]
{0.404935, 0.460706, 2.20041, 0.349605, 0.245138, 0.329001, 0.900709,
0.231028, 0.454624, 0.413024}
```

В этом пакете представлен также ряд менее распространенных статистических функций, например, для нецентрированных законов распределения – Хи-квадрат, Стьюдента и Фишера.

6.11.7. Функции дискретного распределения

Пакет DiskreteDistribution содержит функции для дискретного распределения вероятностей:

- **BernoulliDistribution[p]** – дискретное распределение Бернулли со средним p;
- **BinomialDistribution[n,p]** – биномиальное распределение;
- **DiscreteUniformDistribution[n]** – дискретное равномерное распределение;
- **GeometricDistribution[p]** – геометрическое распределение;
- **HyperGeometricDistribution[n,ns,nt]** – гипергеометрическое распределение;
- **NegativeBinomial Distribution[r,p]** – отрицательное биномиальное распределение;
- **PoissonDistribution[mu]** – распределение Пуассона со средним mu.

В этом подпакете есть также набор статистических функций (PDF, CDF и др.), подобных приведенным для подпакета непрерывных распределений. В связи с этим ограничимся несколькими примерами, приведенными ниже:

```
<<Statistics`DiscreteDistributions`
pdist = PoissonDistribution[0.5]
PoissonDistribution[0.5]
Mean[pdist]
0.5
Quantile[pdist, .85]
1
ExpectedValue[x^3, pdist, x]
5
```

```
RandomArray[bdist, {3, 3}]
{{11, 10, 9}, {11, 16, 8}, {15, 13, 10}}
```

Подпакеты непрерывных и дискретных распределений охватывают наиболее распространенные законы распределения.

6.11.8. Графика пакета *Statistica*

В подпакете `StatisticsPlots` есть ряд функций для построения статистических данных:

- **BoxWhiskerPlot[data]** – построение «ящика с усами».
- **ParetoPlot[data]** – построение диаграммы Парето.
- **QuantilePlot[list1, list2]** – построение графиков квантилей.
- **PairwiseScatterPlot[matrix]** – построение точечной диаграммы для многовариантных данных.

Следующий пример демонстрирует построение пяти «ящиков с усами» (рис. 6.54):

```
<<Statistics`StatisticsPlots`
datm = Table[Random[], {100}, {5}];
BoxWhiskerPlot[datm]
```

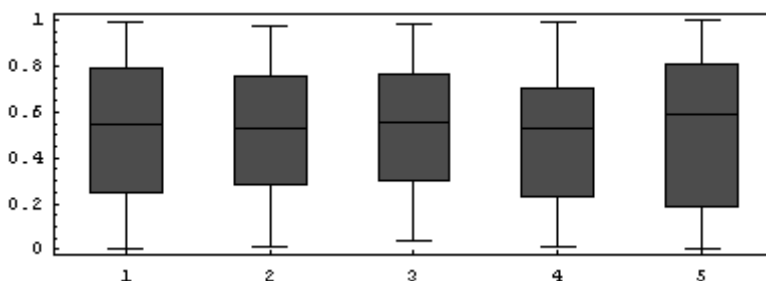


Рис. 6.54. Построение пяти «ящиков с усами»

Параметры «ящиков с усами» можно менять с помощью опций. Так, обычно «ящики с усами» строятся для квантилей, отстоящих от медианы на величину 0,25. Опция `BoxQuantile` позволяет изменить это значение. Опция `BoxOrientation` с начальной установкой `Vertical` задает вертикальное расположение ящиков, тогда как эта опция с установкой `Horizontal` позволяет строить горизонтально расположенные «ящики с усами». С другими опциями можно познакомиться по справке.

Пример построения диаграммы Парето представлен на рис. 6.55. Диаграмма представляет собой одновременное построение гистограммы и интегральной (кумулятивной) кривой для данных. В этом примере интересно представление данных в виде массива символов.

```
ParetoPlot[{a, b, c, d, d, d, e, d, e, e, f, a, b, c}]
```

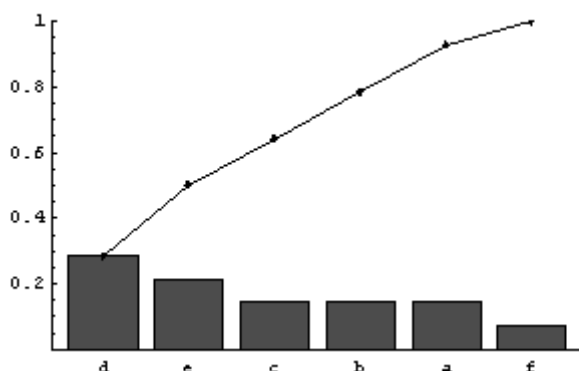


Рис. 6.55. Пример построения диаграммы Парето

6.11.9. Другие функции статистики

Для выполнения дисперсионного анализа служит подпакет ANOVA *дисперсионного анализа*. Он содержит основную функцию

- **ANOVA[data]** – односторонний дисперсионный анализ.
- **ANOVA[data,model,factor]** – общий дисперсионный анализ со спецификацией model и factors.

Вывод результатов этого анализа довольно громоздок, и его можно найти в примерах справки по данному подпакету.

Функции для оценки доверительных интервалов сосредоточены в подпакете ConfidenceInterval. Приведем примеры применения некоторых функций этого подпакета:

```
<<Statistics`ConfidenceIntervals`
data1 = {2.0, 1.25, 0.7, 1.0, 1.1, 3.2, 3.2,
3.3, 2.1, 0.3};
data2 = {1.8, 0.2, 1.5, 1.9, 1.1, 3.0, 2.3,
0.9, 2.4, 1.0};
MeanCI[data2]
MeanCI[{1.8,0.2,1.5,1.9,1.1,3.,2.3,0.9,2.4,1.}]
MeanDifferenceCI[data1, data2,
  EqualVariances -> True]
{-0.72087,1.13087}
MeanDifferenceCI[data1, data2]
{-0.726116,1.13612}
VarianceCI[data1]
{0.588256,4.14394}
VarianceRatioCI[data1, data2,
  ConfidenceLevel -> .9]
{0.559735,5.65632}
mean = Mean[data1]
```

```

1.815
se = StandardErrorOfSampleMean[data1]
0.352613
StudentTCI[mean, se, Length[data1] - 1,
ConfidenceLevel -> 0.9]
{1.16862, 2.46138}

```

В подпакете HypothesisTests сосредоточено сравнительно небольшое число хорошо известных функций для выполнения тестов *проверки статистических гипотез*. Загрузка пакета и проведения теста на среднее значение показана ниже:

```

<<Statistics`HypothesisTests`
data1 = 34, 37, 44, 31, 41, 42, 38, 45, 42, 38;
MeanTest[data1, 34, KnownVariance -> 8]
OneSidedPValue->3.05394×10-9

```

Пакеты LinearRegression и NonlinearFit содержат средства для выполнения линейной регрессии общего вида и нелинейной регрессии.

У специалистов в области статистики интерес вызовут подпакеты MultiDescriptiveStatistics и MultinormalDistribution с многочисленными функциями множественных распределений. Они позволяют оценивать статистические характеристики объектов, описываемых функциями многих переменных. Рисунок 6.56

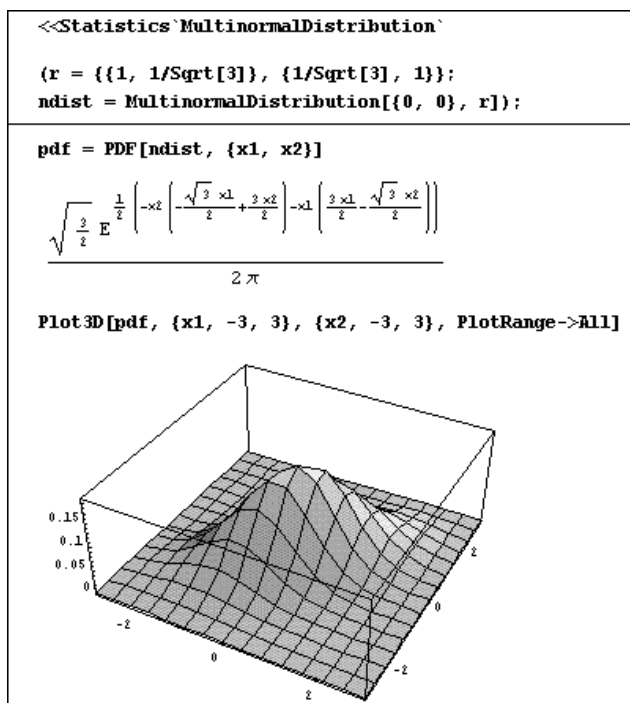


Рис. 6.56. Получение аналитического выражения и графика нормального распределения по двум переменным

поясняет загрузку подпакета `MultinormalDistribution`, получение выражения для плотности нормального распределения по двум переменным `x1` и `x2` и получение трехмерного графика для плотности такого распределения.

Подпакет `Common` содержит перечисление всех определений, вошедших в расширение `Statistics` с указанием директорий, в которые они входят.

6.12. Статистические вычисления в Mathematica 6

6.12.1. О пакете расширения *Statistics* в системе Mathematica 6

Пакет расширения `Statistics` существует в Mathematica 6 в качестве наследства от предшествующих версий Mathematica. Однако, если попытаться загрузить его (или его часть), то будет выдано следующее сообщение:

```
<<Statistics`DataManipulation`  
General::obspkg : Statistics`DataManipulation` is now obsolete. The legacy  
version being loaded may conflict with current Mathematica functionality.  
See the Compatibility Guide for updating information.
```

Из этого сообщения вытекает, что данный пакет для Mathematica 6 устарел и не рекомендуется к применению. Его применение возможно, но может вызвать конфликт со статистическими функциями, включенными в ядро системы Mathematica 6. Тут пора раскрыть секрет – функции данного пакета в системе Mathematica 6 включены в ядро системы, так что необходимости вызывать из наследственного пакета `Statistics`, как правило, просто нет.

На рис. 6.57 показана страница справки системы Mathematica 6, в которой представлены все основные статистические функции, в том числе входящие в состав пакета `Statistics`.

В справке можно найти также самоучители `Continuous Distributions` и `Discrete Distributions` с описанием функций непрерывных и дискретных распределений.

6.12.2. Аналитические статистические расчеты

Mathematica 6 сохранила возможность проведения статистических вычислений в аналитическом (символьном) виде. Это приоткрывает завесу перед такими вычислениями и демонстрирует формулы, по которым такие вычисления производятся в ядре системы. Примеры подобных вычислений представлены ниже:

```
data:={a,b,c,d}  
Mean[data]  
1/4 (a+b+c+d)  
data^2*Mean[data]^2
```

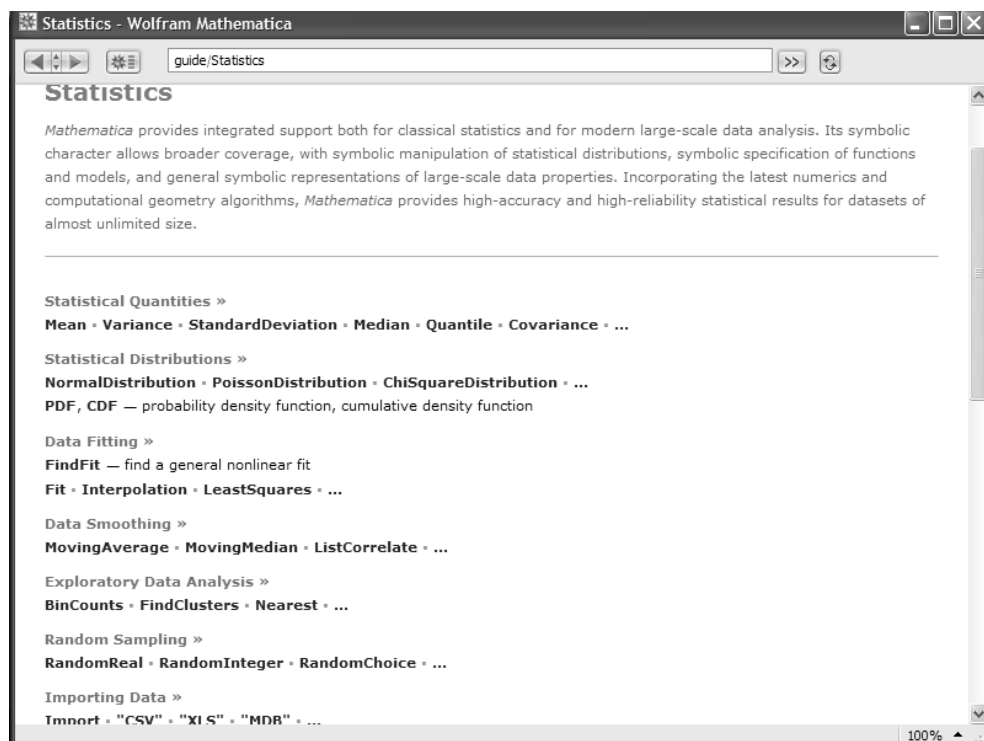


Рис. 6.57. Страница справки системы Mathematica 6 по статистическим расчетам

```
{1/16 a^2 (a+b+c+d)^2, 1/16 b^2 (a+b+c+d)^2, 1/16 c^2 (a+b+c+d)^2, 1/16 d^2 (a+b+c+d)^2}
```

Variance[data]

```
1/12 ((3 a-b-c-d) Conjugate[a]+(-a+3 b-c-d) Conjugate[b]+(-a-b+3 c-d) Conjugate[c]+(-a-b-c+3 d) Conjugate[d])
```

Quantile[NormalDistribution[μ,σ],q]

```
μ+√2σ InverseErf[-1+2 q]
```

StandardDeviation[LogNormalDistribution[0,1]]

```
√(-1+e) e
```

Covariance[data]

```
1/12 ((3 a-b-c-d) Conjugate[a]+(-a+3 b-c-d) Conjugate[b]+(-a-b+3 c-d) Conjugate[c]+(-a-b-c+3 d) Conjugate[d])
```

MovingAverage[{a,b,c,d,e,f},3]

```
{1/3 (a+b+c), 1/3 (b+c+d), 1/3 (c+d+e), 1/3 (d+e+f)}
```

MovingAverage[{a,b,c,d,e,f},{1,3}]

```
{a/4+(3 b)/4, b/4+(3 c)/4, c/4+(3 d)/4, d/4+(3 e)/4, e/4+(3 f)/4}
```

6.12.3. Численные статистические расчеты в Mathematica 6

Численные статистические расчеты выполняются столь же просто, как и аналитические. Их результаты представлены числами или списками с числами, они более компактны. Ограничимся приведением нескольких наглядных примеров:

```
Clear[data]
data={1.1, 1.95, 3.13, 4.05, 4.96}
{1.1,1.95,3.13,4.05,4.96}
Mean[data]
3.038
Median[data]
3.13
Covariance[data]
2.41657
MovingAverage[data,3]
{2.06,3.04333,4.04667}
```

Заинтересованный в подобных расчетах читатель может сам опробовать действие и других статистических функций системы Mathematica 6.

6.12.4. Статистические расчеты с графической визуализацией

Средства статистических расчетов в Mathematica 6 можно объединить с графической визуализацией их результатов. В качестве примера на рис. 6.58 представлен пример, обеспечивающий задание вектора из 20 чисел с плавающей точкой, их экспоненциальное сглаживание с помощью функции `ExponentialMovingAverage` и построение графиков с линейной интерполяцией для исходных и сглаженных точек.

Как видно из рис. 6.58, исходная кривая, построенная по исходным точкам, характеризуется сильной неравномерностью, а кривая, построенная по сглаженным точкам, является, естественно, гораздо более плавной. Однако характерно, что она (за исключением начала) заметно смещена вниз. Это говорит о несовершенстве алгоритма сглаживания.

В другом примере (рис. 6.59) задана синусоидальная функция, зашумленная с помощью генератора случайных чисел, реализованного функцией `RandomReal`. Сглаживание производится с помощью экспоненциального окна, после чего строится график зашумленной синусоиды, а внутри него белым цветом строится график уже сглаженной функции.

В этих примерах использована обычная и достаточно эффективная техника графической визуализации статистических расчетов. Но, что делает статистические расчеты в среде Mathematica 6 уникальными по наглядности, так это применение средств динамической интерактивной визуализации. Эти средства основа-

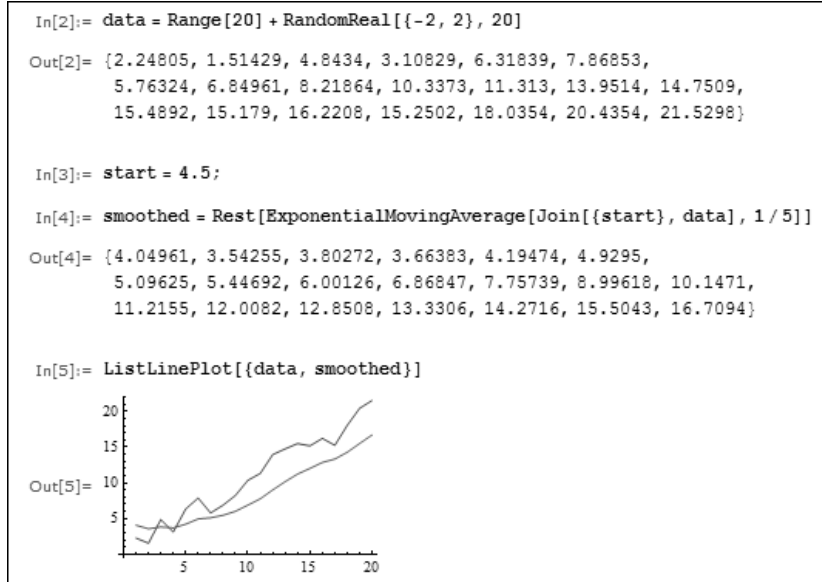


Рис. 6.58. Пример сглаживания 20 точек с графической визуализацией

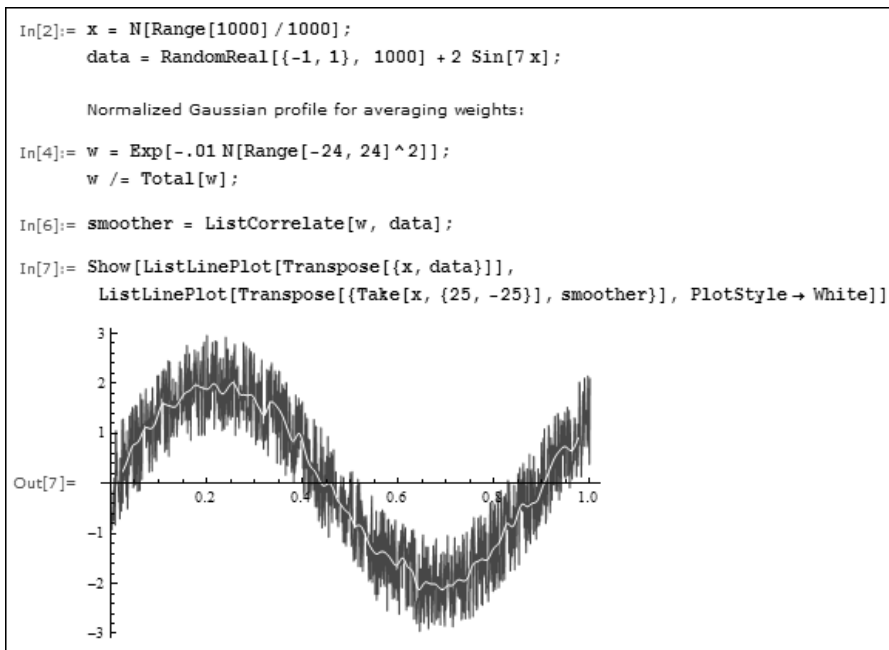


Рис. 6.59. Пример оконного экспоненциального сглаживания зашумленной синусоидальной функции

ны на использовании средств графического интерфейса пользователя в сочетании со средствами динамической визуализации.

На рис. 6.60 показан пример осуществления полиномиальной регрессии некоторой зависимости, заданной пятью точками. Точки заданы локаторами (кружками с крестиками), которые можно перемещать мышью, динамически имитируя самые разные зависимости. С помощью слайдера можно менять степень полинома, который используется для проведения регрессии (если степень полинома меньше числа точек минус 1) или просто полиномиальной аппроксимации. В последнем случае кривая аппроксимирующего полинома будет проходить точно по исходным точкам, но возможны большие выбросы в промежутках между ними и за пределами исходных точек.

Рисунок 6.60 представляет случай, когда степень полинома $n=3 < (5-1)$. При этом кривая полинома проходит вблизи исходных точек, обеспечивая наимень-

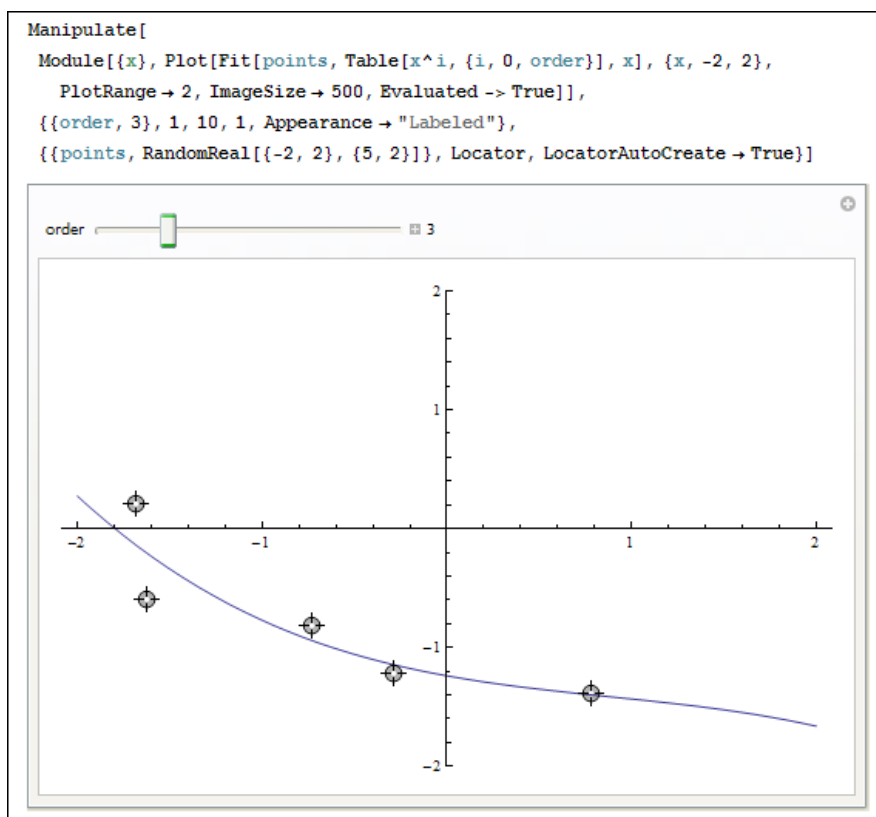


Рис. 6.60. Пример полиномиальной регрессии и аппроксимации, произвольно расположенных и перемещаемых мышью исходных точек (степень полинома 3)

шую среднеквадратическую погрешность приближения. Другой случай (для $n=7$) представлен на рис. 6.61. Как уже отмечалось, уже при $n=4$ регрессия вырождается в полиномиальную аппроксимацию, при которой кривая полинома строго проходит через исходные точки. Дальнейшее повышение степени полинома (как на рис. 6.61) приводит лишь к большим выбегам кривой приближающего полинома между точками и за их пределами.

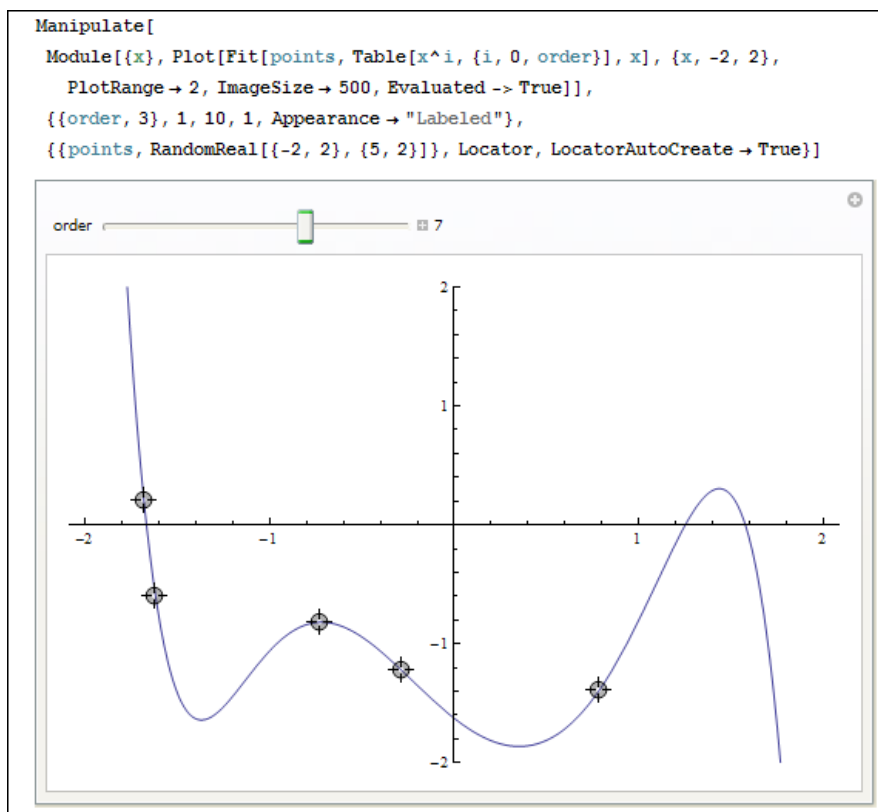


Рис. 6.61. Пример полиномиальной регрессии и аппроксимации, произвольно расположенных и перемещаемых мышью исходных точек (степень полинома 7)

Возможность перемещения исходных точек мышью и оперативной смены степени приближающего полинома делает работу с примерами из рис. 6.60 и 6.61 весьма наглядной, удобной и очень поучительной. Разумеется, подобные примеры можно привести и к другим разделам статистических вычислений.

Функции символьных преобразований

7.1. Работа с выражениями	370
7.2. Работа с функциями	375
7.3. Задание математических отношений	379
7.4. Функции упрощения выражений	381
7.5. Раскрытие и расширение выражений	384
7.6. Функции и директивы для работы с полиномами	389
7.7. Расширенные операции с выражениями	394

7.1. Работа с выражениями

7.1.1. Полная форма выражений

Одним из важнейших понятий системы Mathematica является математическое выражение, или просто *выражение* – **expr** (от английского слова expression). Работа с математическими выражениями в символьном виде – основа основ символьной математики.

Выражение может быть представлено в общепринятом виде (как математическая формула или ее часть) с помощью операторов, например $a*(x+y+z)$ или x^u , оно может задавать и некоторую функцию $f[x,y,...]$ или их комбинацию. Наряду с такой формой, существует так называемая *полная форма* представления выражений, при которой основные арифметические операции задаются не операторами, а только соответствующими функциями. Фактически именно с выражениями, представленными в полной форме, оперирует символьное ядро системы Mathematica. Эта форма соответствует реализации вычислений в функциональном программировании.

Приведем примеры полной формы основных математических выражений:

Выражение expr	Полная форма expr	Комментарий
$x+y+z$	Plus [x,y,z]	Сложение
$x\ y\ z$	Times [x,y,z]	Умножение
x^n	Power [x,n]	Возведение в степень
$\{a,b,c\}$	List [a,b,c]	Создание списка
$a \rightarrow b$	Rule [a,b]	Подстановка
$a=b$	Set [a,b]	Установка

Для вывода выражения expr в полной форме используется функция

FullForm[expr]

Примеры перевода выражений в его полную форму:

$1 + x^2 + (y+z)^2 + 2$

$3 + x^2 + (y + z)^2$

FullForm[%]

Plus[3,Power[x,2],Power[Plus[y,z],2]]

Integrate[a*Sin[b*x]*Exp[-c*x],x]

$$-\frac{aE^{-cx}(b\cos[bx] + c\sin[bx])}{(-Ib+c)(Ib+c)}$$

FullForm[%]

Times[-1,a,Power[Plus[Times[Complex[0,-1],b],c],-1],Power[Plus[Times[Complex[0,1],b],c],-1],Power[E,Times[-1,c,x]],Plus[Times[b,Cos[Times[b,x]]],Times[c,Sin[Times[b,x]]]]]

Для определения типа выражения служит функция

Head[expr]

Применительно к числовым выражениям, она возвращает тип результата, например:

1 + 2 + 3

6

Head[%]

Integer

Head[123/12345]

Rational

Head[2*0.25]

Real

Следующие примеры поясняют действие функции **Head** при символьных выражениях:

Head[f[x,y,z]]	возвращает	f
Head[a+b+c]	возвращает	Plus
Head[x^n]	возвращает	Power
Head[{a,b,c}]	возвращает	List

и т. д. Другая пара примеров показывает применение **Head** в списках с разнородными выражениями:

```
{Head[1+2], Head[a*b], Head[5/7], Head[1+3*I], Head[Exp[2]]}
{Integer, Times, Rational, Complex, Power}
{Integer, Rational, Real, Complex, Symbol, f, List, Plus, Times}
```

Обратите внимание на второй пример: при нем функция **Head** используется к каждому выражению списка, что дает более компактную запись.

7.1.2. Основные формы выражений

Помимо полной, возможны еще четыре основные формы записи выражений:

f[x,y]	Стандартная форма для f[x,y].
f@x	Префиксная форма для f[x].
x//f	Постфиксная форма для f[x].
x~f~y	Инфиксная форма для f[x,y].

Примеры на применение этих форм (ввод слева, вывод справа):

F[x_]=2*x^2	$2x^2$
F[a]	$2a^2$
a//F	$2a^2$
f[x_,y_]=x^2+y^2	$a^2 + x^2$
f[a,b]	$2a^2$
a~f~b	$2a^2$

Можно использовать ту или иную форму выражений в зависимости от класса решаемых математических задач.

7.1.3. Части выражений и работа с ними

Сложные выражения состоят из *частей*, которые могут интерпретироваться различным образом:

Тип части	Зависимость	Пример
Function	От аргументов или параметров	Exp[x], f[x,y]
Command	От аргументов или параметров	Expand[(x-1)^2]
Operator	От операндов	x+y+z, a=b
Head	От элементов	{a,b,c}
Object type	От контекста	RGBColor[r,g,b]

Работа с частями выражений напоминает работу со списками. Для выделения любой заданной части выражений используются функция **Part** или двойные квадратные скобки:

Part[expr,n] или expr[[n]]	Выделяет n-ую часть выражения с начала.
expr[[-n]]	Выделяет n-ую часть выражения с конца.
expr[[n1,n2,...]]	Выделяет части выражения и показывает их в форме дерева.
expr[{{n1,n2,...}}]	Дает комбинацию нескольких частей выражения.

Приведем примеры на использование этих средств:

```
f:=a+b*x^2+c*x^3
Part[f,2]          bx^2
Part[f,3]          cx^3
f[[1]]             a
f[[3]]             cx^3
f[[-1]]            cx^3
```

Нередко выражения рассматриваются как возможные значения переменных. В этом случае используются операторы *присваивания* переменным заданных значений. Mathematica имеет два типа присваивания: с помощью символов **:=** и **=**. Они отличаются временем (моментом) исполнения выражения, следующего за этими символами. Знак **:=** используется для *задержки присваивания* до вычисления правой части, например:

```
f[x_] := % + 2 x
```

Вывода здесь нет. Продолжим наш эксперимент:

```
1 + y^2
1 + y^2
g[x_] = % + 2 x
1 + 2x + y^2
```

Теперь вывод есть, так как % определено в виде выражения $1 + y^2$ и при задании **g[x_]** использован оператор присваивания *немедленного исполнения*. Далее:

```
2+z
2+z
{f[a],g[a]}
{2 + 2a + z, 1 + 2a + y^2}
```

Следующие функции возвращают особые части выражения:

- **Denominator[expr]** – возвращает знаменатель выражения **expr**;
- **Numerator[expr]** – возвращает числитель выражения **expr**;
- **First[expr]** – возвращает первый элемент в **expr**;

- **Last[expr]** – возвращает последний элемент в expr;
- **Rest[expr]** – возвращает expr с удаленным первым элементом.

Примеры применения этих функций:

Denominator [(x+1)/(x ² +2*x+3)]	3 + 2x + x ²
Numerator [(x+1)/(x ² +2*x+3)]	1+x
expr =a*b+c-d	a b+c-d
First [expr]	a b
Last [expr]	-d
Rest [expr]	c-d

Работа с выражениями, умение их преобразовывать и выделять нужные их фрагменты является важнейшей частью культуры символьных преобразований.

7.1.4. Удаление элементов выражения

Иногда возникает необходимость в **удалении** части выражения. Для этого используются следующие функции:

- **Delete[expr, n]** – удаляет элемент на позиции n в выражении expr. Если n отрицательно, позиция отсчитывается с конца;
- **Delete[expr, {i, j, ...}]** – стирает часть выражения на позиции {i, j, ...};
- **Delete[expr, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – удаляет части выражения в нескольких указанных позициях;
- **DeleteCases[expr, pattern]** – исключает все элементы выражения expr, которые совпадают с образцом pattern;
- **DeleteCases[expr, pattern, levspec]** – исключает все части выражения expr на уровнях, указанных levspec, и соответствующих образцов pattern.

Следующие примеры иллюстрируют применение этих функций:

expr =a*b+c-d	a b+c-d
Delete [expr, 1]	c-d
Delete [expr, 3]	a b+c
Delete [expr, {{1}, {3}}]	c
DeleteCases [expr, a*b]	c-d
DeleteCases [expr, c, 1]	a b-d

Обратите внимание на то, что в общем случае выражения могут быть многоуровневыми. Уровень задается спецификацией **levspec**.

7.1.5. Другие манипуляции с выражениями

В процессе преобразований выражений с ними возможны и иные манипуляции. Наиболее важные из них выполняются следующими функциями:

- **Append[expr, elem]** – возвращает expr с дополнением elem;
- **AppendTo[s, elem]** – добавляет elem к значению s и переустанавливает s в новое значение;
- **Apply[f, expr, levspec]** – возвращает expr, заменяя заголовки в тех частях expr, которые указаны спецификацией уровня levspec;

- **Cancel[expr]** – возвращает expr с сокращением общих множителей числителя и знаменателя;
- **Cases[expr, pattern, levelspec]** – возвращает список всех частей выражения expr на уровнях, указанных спецификацией levelspec, и которые соответствуют шаблону pattern;
- **Chop[expr]** – в комплексном выражении expr задает нулевой малую мнимую часть;
- **Chop[expr, tol]** – присваивает значение 0 приближенным вещественным числам в выражении expr, абсолютные величины которых меньше tol;
- **Replace[expr, rules]** – возвращает expr с подстановкой заданной правилом или списком правил rules;
- **ReplaceAll** – используется в виде: **expr /. rules** – возвращает expr с подстановками, заданными правилом или списком правил;
- **ReplaceHeldPart[expr, Hold[new], n]** – возвращает выражение, в котором n-ная часть expr заменена на new;
- **ReplaceHeldPart[expr, Hold[new], {i, j, ...}]** – заменяет часть на позиции {i, j, ...};
- **ReplaceHeldPart[expr, Hold[new], {{i1, j1, ...}, {i2, j2, ...}, ...}]** – замещает части в нескольких позициях на new;
- **ReplacePart[expr, new, n]** – возвращает выражение, в котором n-ная часть expr замещена на new;
- **ReplacePart[expr, new, {i, j, ...}]** – замещает часть на позиции {i, j, ...};
- **ReplacePart[expr, new, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – замещает части в нескольких позициях на new;
- **ReplaceRepeated** – применяется в виде: **expr //. rules** – неоднократно выполняет замещения до тех пор, пока expr не перестанет изменяться.

Действие некоторых этих функций достаточно очевидно и поясняется следующими примерами:

Append[a+c, b]	a+b+c
x={a, b, c}	{a, b, c}
AppendTo[x, 15]	{a, b, c, 15}
x	{a, b, c, 15}
Apply[f, a^2+b^2, 2]	f[a, 2]+f[b, 2]
Cancel[(z-1)^2/(z-1)]	-1+z
Cases[{a, 3.5, 2, 5, "HELLO"}, _Integer]	{2, 5}
Exp[N[- π I]]	-1. - 1.22461×10 ⁻¹⁶ i
Chop[%]	-1.
Exp[N[- π I]]	-1. - 1.22461×10 ⁻¹⁶ i
Exp[N[- π I]]	-1. - 1.22461×10 ⁻¹⁶ i
Chop[%, 1*10⁻¹⁰]	-1.
Replace[s^2, s^2->a]	a
s^2/. s->a	a ²

Заинтересованному в таких манипуляциях читателю рекомендуется просмотреть множество примеров в справочной системе Mathematica и, разумеется, попробовать свои примеры.

7.1.6. Контроль выражений

При создании программного обеспечения на языке Mathematica, а иногда и в ходе диалоговой работы с системой, необходим контроль над некоторыми *свойствами выражений*. Следующие функции обеспечивают такой контроль:

- **AtomQ[expr]** – возвращает True, если выражение expr не может быть разложено на подвыражения и является атомарным, и возвращает False в противном случае;
- **ByteCount[expr]** – возвращает количество байтов, используемых системой Mathematica для внутреннего хранения выражения expr;
- **FreeQ[expr, form]** – вырабатывает значение True, если в выражении expr отсутствует подвыражение, совпадающее с form, в противном случае дает False;
- **FreeQ[expr, form, levelspec]** – тестирует только части выражения на уровнях, указанных levelspec.

Следующие примеры показывают действие этих функций:

AtomQ[{a}]	False
AtomQ[2+3/4]	True
ByteCount[1+3/4]	64
ByteCount[1.+3./4]	16
FreeQ[a*x^b, a]	False
FreeQ[a*x^b+c, 1]	False
FreeQ[a*x^b+c, 1, 1]	True
FreeQ[a*x^b+c, b, 2]	True

7.2. Работа с функциями

Функции в системе Mathematica характеризуются именем (обобщенно f) и выражением expr, задающим функциональную зависимость, и списком параметров (возможно пустым) в квадратных скобках. Обычно функция возвращает значение выражения – численное или символьное – в ответ на обращение к ней. Однако в системе Mathematica понятие функции значительно расширено, и она может возвращать также и любой объект, например графический или звуковой. Можно сказать, что входной язык общения с системой Mathematica основан на принципах *функционального программирования* с применением полных форм представления выражений.

7.2.1. Приложение имени функции к выражению или его части

Следующие функции позволяют прикладывать имя функции к выражению или к частям выражения:

- **Apply[f, expr]** – замещает заголовок выражения expr на f;

- **Nest[f, expr, n]** – возвращает выражение, полученное n-кратным применением f к expr.
- **Map[f, expr]** – прикладывает f к каждому элементу на первом уровне в expr.
- **Map[f, expr, levelspec]** – применяет f к частям expr, указанным с помощью levelspec.
- **MapAll[f, expr]** – прикладывает f ко всем частям выражения expr.

Приведем примеры на действие этих функций:

Apply[f, {a,b,x}]	$f[a,b,x]$
Nest[f,x,3]	$f[f[f[x]]]$
s[x_,y_,z_]:=x+y+b	
N[Apply[s,{1,2,a}]]	$3. + b$
Map[f,{a,b,c}]	$\{f[a], f[b], f[c]\}$
MapAll[f,a*x+b]	$f[f[b]+f[f[a] f[x]]]$
MapAll[f,{a,b,c}]	$f[\{f[a], f[b], f[c]\}]$

7.2.2. Укороченная форма функций

Из описания указанных функций вытекает, что они наряду с полной формой могут задаваться *укороченной формой*:

f @ expr	f[expr]
f @@ expr	Apply[f, expr]
f /@ expr	Map[f, expr]
f //@ expr	MapAll[f, expr]

Смысл укороченных выражений очевиден:

f@{a,b,c}	$f[\{a,b,c\}]$
f@@{a,b,c}	$f[a,b,c]$
f/@{a,b,c}	$\{f[a], f[b], f[c]\}$
f//@{a,b,x}	$f[\{f[a], f[b], f[x]\}]$

Укороченная форма функций может оказаться полезной для укорочения записи алгоритмов и программ.

7.2.3. Выделение заданного аргумента в функциях

Функция **Slot[n]**, или в укороченной форме **#n**, представляет n-й аргумент функции. Это иллюстрируют следующие примеры:

(5*Slot[1]+Slot[2]*Slot[3]^2)&[a,b,c]	$5a + bc^2$
#1^#2&[a,b]	a^b

Объект **#** эквивалентен **#1**, а **#0** – заголовку абстрактной функции. Так что $F[\#.2] \& F[a,b]$ эквивалентно $F[a,b]$.

Функция **SlotSequence[n]** или в укороченной форме **##n**, где $n=1,2,\dots$, представляет порядок применения формальных аргументов к абстрактной функции. Так что объект **##n** определяет последовательность аргументов, начиная с n-го:

```
(Times[5,##2]+Times[##2,##3^2])&[a,b,c]      5bc + bc^3
```

Представленные средства обеспечивают работу с функциями на абстрактном уровне.

7.2.4. Подстановки в функциях

Интересные возможности связаны с использованием *подстановок* при определении функций. Система допускает использование таких подстановок:

f[x] = value и **f[x_] = value**

Поясним это примерами:

```
f[x]=u           u
f[x]+f[y]        u+f[y]
f[x_]=x^2        x^2
f[x]+f[y]        u = y^2
Clear[f]
f[x]+f[y]        f[x]+f[y]
```

Как видно из этих примеров, подстановки в функциях могут существенно изменить исходную функциональную зависимость. А потому важной областью их применения является модификация функций.

7.2.5. Рекурсивные функции

Использование подстановок при определении функции позволяет легко реализовать *рекурсивные* алгоритмы, т.е. алгоритмы, при которых очередной шаг вычислений основан на определенном преобразовании предшествующих шагов. Примером может служить задание функции вычисления факториала **fact[n]**, представленное ниже:

Операция	Комментарий
fact[n_] := n*fact[n - 1]	Задана рекурсивная функция факториала
fact[1] := 1	Выполнена инициализация функции
fact[3]	Вычислено значение 3!
6	
fact[10]	Вычислено значение 10!
3628800	
?fact	Выполнена проверка определения функции
Global`fact	
fact[1] := 1	
fact[1] := 1	
fact[_n] := nfact[n - 1]	
fact[n_] := nfact[n - 1]	

Обратите внимание на использование знака вопроса перед именем функции в конце документа, показанного выше. Оно позволяет вывести текст декларации (определения) функции. После объявления функции таким образом она может быть использована в последующих ячейках документа.

7.2.6. Дополнительные примеры на работу с функциями

Приведем еще ряд примеров на действие функций **Apply**, **Map** и **Nest**:

Nest [f,x,3]	f[f[f[x]]]
Apply [f,{a,b,c}]	f[a, b, c]
s [x_,y_,z_] := x+y+b	
N [Apply [s,{1,2,a}]]	3. + b
Map [f,{a,b,c}]	{f[a], f[b], f[c]}
N [Map [Exp ,{1,2,3}]]	{2.71828, 7.38906, 20.0855}
Map [f,1+2+c]	f[3] + f[c]
m ={{a,b},{c,d}}	{{a, b}, {c, d}}
Map [f,m]	{f[{a, b}], f[{c, d}]}
take2 [list_] := Take [list,2]	
Map [take2 ,{{a,b,c},{c,a,b},{c,c,a}}]	{{a, b}, {c, a}, {c, c}}

Большинство из описанных операций для работы с функциями могут использоваться и для операций со списками. Порою это резко упрощает запись алгоритмов вычислений для данных, представленных списками, поскольку дает общее определение функций для произвольного числа их параметров. Примерами могут служить определения следующих статистических функций.

Вычисление среднего для элементов списка:

```
Mean[list_] := Apply[Plus, list] / Length[list]  /; VectorQ[list]
&& Length[list] > 0
```

Вычисление среднего геометрического для списка:

```
GeometricMean[list_] := Apply[Times, list^(1/Length[list])]  /;
VectorQ[list] && Length[list] > 0
```

Вычисление гармонического среднего для списка:

```
HarmonicMean[list_] := Length[list] / Apply[Plus, 1/list]  /;
VectorQ[list] && Length[list] > 0
```

Обратите внимание, что при задании первой функции **Mathematica** предупреждает о том, что введенный идентификатор **list** опасно напоминает зарезервированный идентификатор **List**. Все приведенные выше функции не имеют смысла, если список пустой. Поэтому в них введен контроль над такой ситуацией.

Теперь можно выполнить расчеты по этим формулам:

data ={1,2,3,4}	{1,2,3,4}
Mean [data]	$\frac{5}{2}$
GeometricMean [data]	$2^{3/4} 3^{1/4}$
HarmonicMean [data]	$\frac{48}{25}$

Большое число операций для работы с функциями полезно при организации функционального программирования, а также при создании пакетов расширения системы для выполнения символьных преобразований и расчетов. Разумеется,

это разумно делать профессионалам-математикам, а не обычным пользователям. Последних, скорее всего, более чем удовлетворит уже имеющийся в системе набор таких операций и функций.

7.2.7. Инверсные функции

Инверсными функциями называют функции, полученные в результате обращения заданных функций. Например, для функции **Sin[x]** инверсной будет **ArcSin[x]** и т.д. Следующие функции обеспечивают представление инверсных функций:

- **InverseFunction[f]** – представляет функцию, обратную f , определенную таким образом, что **InverseFunction[f][y]** – возвращает значение x , для которого $f[x]$ равно y . Для функции нескольких переменных **InverseFunction[f]** представляет обращение по первому аргументу;
- **InverseFunction[f, n]** – представляет обращение по n -му аргументу;
- **InverseFunction[f, n, tot]** – представляет обращение по n -му аргументу, когда имеется всего tot аргументов.

Следующие примеры иллюстрируют работу с этими функциями:

```
InverseFunction[Sin]
ArcSin
%[x]
ArcSin[x]
Composition[f, g, h]
Composition[f, g, h]
InverseFunction[Composition[%, q]]
Composition[q(-1), h(-1), g(-1), f(-1)]
```

Обратите внимание на то, что в этих примерах фигурируют заголовки функций: например, для получения инверсной функции от **Sin[x]** следует использовать в качестве аргумента функции **InverseFunction[f]** вместо f соответственно **Sin**.

7.3. Задание математических отношений

7.3.1. Для чего нужно задание новых отношений

Символьные преобразования при всей их кажущейся таинственности осуществляются по определенным, хотя и весьма многочисленным, а потому для многих из нас запутанным, правилам [2, 3]. Основные из них давно известны из математики и описаны в многочисленных справочниках и монографиях [5, 28–37]. Они записаны в ядре системы и вызываются из него при создании условий, необходимых для выполнения того или иного преобразования. Если этих условий нет, исходное выражение просто повторяется. А если обнаружена явная ошибка в преобразова-

ниях, то о ее сути выводится соответствующее сообщение. При ситуациях лишь близких к ошибочным выводится предупреждающее сообщение, и вычисления продолжаются.

Однако математика и использующие ее науки непрерывно развиваются. Появляются все новые и новые правила преобразований. Пользователь-математик может пожелать изменить встроенные правила преобразований, например, для создания новых разделов математики, базирующихся на каких-либо новых представлениях. Блестящий пример этого – теория относительности Эйнштейна.

Таким образом, возникает необходимость *расширения* математических символьных систем и обучения их новым правилам математических преобразований. Система Mathematica имеет и такие возможности. Поясним на простых примерах, как это делается.

7.3.2. Примеры задания математических отношений

В математике можно найти множество примеров *математических отношений*. Например, хорошо известно такое отношение для логарифма и экспоненциальной функции:

$$\log(\exp(x)) = x.$$

Не обременяя себя поиском действительно новых закономерностей (порою на это может не хватить жизни, да и везет не каждому ученому), зададим приведенную закономерность для введенных по-новому функций \log и \exp . Центральным моментом тут является введение новых имен функций, которые начинаются с малых букв, а не с больших, как у встроенных функций Log и Exp . Поэтому система воспринимает \log и \exp как новые функции.

Итак, «новую» закономерность вводим следующим образом:

```
log[exp[x_]]:=x
```

```
General ::spell1 :
```

```
Possible spelling error : new symbol name "log" is
similar to existing symbol "Log". More...
```

```
General ::spell1 :
```

```
Possible spelling error : new symbol name "exp" is
similar to existing symbol "Exp". More...
```

Система на всякий случай сообщает о рискованности эксперимента: символы \log и \exp похожи на зарезервированные имена функций **Log** и **Exp**. Проигнорировав это предупреждение, проверим данную закономерность в работе:

```
log[exp[15]]
```

```
15
```

```
log[exp[y^2+1]]
```

```
1 + y^2
```

Итак, наша «новая» закономерность работает. Можно ввести, скажем, и такое известное отношение:

```
log[x_^n_] := n*log[x]
```

Проверим, какие отношения нами заданы для функции log:

```
?log
Global`log
log[exp[x]] := x
log[x_^n_] := nlog[x]
```

Проверим введенные правила, например:

```
log[(1+x)^5]
5 log[1+x]
```

Рассмотрим еще пару примеров на задание «новых» математических правил. В первом примере задано правило: логарифм произведения равен сумме логарифмов сомножителей:

```
log[x_*y_] := log[x] + log[y]
```

Любопытно, что эта закономерность действует при любом числе сомножителей:

```
log[a*b*c*d*e]
log[a] + log[b] + log[c] + log[d] + log[e]
```

Второй пример иллюстрирует задание объекта, ассоциированного со списком.

```
a/:a[x_] + a[y_] := a[x+y]
a[x] + a[y] + a[z]
a[x+y+z]
```

Введенные здесь обозначения $x_$, $y_$ и $n_$ означают *образцы*, на место которых могут подставляться произвольные выражения. Позже мы обсудим применение образцов более детально. Описанные выше примеры наглядно демонстрируют возможности выполнения так называемого *математического программирования*, в основе которого лежит задание определенных математических соотношений между математическими понятиями, прежде всего, такими, как функции.

7.4. Функции упрощения выражений

7.4.1. Роль упрощения выражений

Упрощение математических выражений – одна из самых важных задач символической математики. Качество выполнения операции упрощения во многом определяется мощностью ядра математической системы, поскольку зависит от числа функций и правил преобразования выражений, заложенных в ее символическое ядро.

Выражения делятся на недостаточно простые и достаточно простые выражения. Недостаточно простые выражения таят в себе всевозможные «излишества»: сокращаемые общие члены, лишние переменные и функции, полиномы со степенями, допускающими понижение, и т.д. Это затрудняет качественный анализ выражений и может даже привести к неоднозначным и даже неверным результатам.

Mathematica всегда старается упростить то или иное выражение, если для этого не требуется каких-либо особых средств. Например, сложные выражения, содержащие элементарные или специальные функции, превращаются в более простые выражения, в том лишь смысле, что они состоят из более простых функций. Следующие примеры иллюстрируют это:

(Csc[x] Tan[w]) / (Cot[x] Sec[w])

Sec[x] Sin[w]

BesselY[1/2, z]

$$-\frac{\sqrt{\frac{2}{\pi}} \cos[\xi]}{\sqrt{\xi}}$$

Однако так бывает далеко не всегда, и для проведения необходимых преобразований используются различные функции, описанные ниже.

7.4.2. Основная функция *Simplify*

Для упрощения выражений используется функция **Simplify**.

Simplify[expr] – исполняет последовательность алгебраических преобразований над выражением *expr* и возвращает простейшую из найденных форм (обычно это бывает нормальная форма выражений).

Приведем наиболее характерные результаты действия функции **Simplify**:

- комбинирование числовых подвыражений, например:

Simplify[6 x 2] возвращает 12 x,

- приведение подобных множителей у произведений, например:

Simplify[x^3 y x^5] возвращает x y,

- приведение подобных членов суммы, например:

Simplify[x + 12 + 4 x] возвращает 5 x + 12,

- упрощение тождеств, содержащих 0 или 1, например:

Simplify[2+0] возвращает 2

Simplify[1*x] возвращает x,

- распределение целочисленных показателей степени в произведениях, например:

Simplify[(5 x^2 y^3)^2] возвращает $5^4 x^4 y^6$,

- сокращение на наибольший полиномиальный делитель, например:

Simplify[(x^2 - 2 x y + y^2)/(x^2 - y^2)] возвращает $\frac{x - y}{x + y}$,

- разложение полиномов и понижение степени выражений, например:

Simplify[(x + 1)^2 - x^2] возвращает 2 x + 1,

- приведение общих знаменателей к выражениям с пониженной степенью или с исключением сокращаемых переменных, например:

Simplify[2 x / (x^2 - 1) - 1/(x + 1)] возвращает 1/(x + 1).

7.4.3. Примеры упрощения выражений функцией *Simplify*

Следующие примеры дополнительно поясняют применение функции **Simplify**:

Simplify[$a^2 - 2ab + b^2$]

$(a - b)^2$

Simplify[$\text{Exp}[x]^2/x$]

e^{2x}

Simplify[$\text{Sin}[x-y] + \text{Sin}[x+y]$]

$2 \cos[y] \sin[x]$

Simplify[$\text{Exp}[x] \cdot \text{Exp}[y] / \text{Exp}[z]$]

e^{x+y-z}

Simplify[$\text{Exp}[z \cdot \text{Log}[b]]$]

b^z

Simplify[$\text{Log}[x/y]$]

$\text{Log}\left[\frac{x}{y}\right]$

Simplify[A]

A

$A := (\cos[4x] - 4\cos[2x] + 3) / (4\cos[2x] + \cos[4x] + 3)$

Simplify[A]

$\tan[x]^4$

Simplify[$6 \cdot \text{Log}[10]$]

$6 \text{Log}[10]$

Simplify[$6 \text{Log}[10]$, ComplexityFunction -> LeafCount]

$\text{Log}[1000000]$

Операция **Simplify** часто выполняется по умолчанию. Например, это обычно происходит при вычислении выражений, примеры чего приводились выше. Несомненно, это одна из наиболее важных и часто применяемых операций компьютерной алгебры.

7.4.4. Функция полного упрощения *FullSimplify*

Функция **FullSimplify**, области применения которой в Mathematica заметно расширены, обладает заметно большими возможностями, чем функция **Simplify**. В частности она обеспечивает упрощение выражений, содержащих специальные математические функции:

FullSimplify[$\text{Gamma}[x] \cdot x \cdot (x+1) \cdot (x+2) \cdot (x+n)$]

$(n+x) \text{Gamma}[3+x]$

Simplify[$\text{Tan}[x]$, ComplexityFunction -> (Count[{#1}, _Tan, ∞] &)]

$\text{Tan}[x]$

FullSimplify[$\text{Tan}[x]$, ComplexityFunction -> (Count[{#1}, _Tan, ∞] &)]

$$- \frac{i (-1 + e^{ix}) (1 + e^{ix})}{1 + e^{2ix}}.$$

Как видно из этих примеров, функция **FullSimplify** обеспечивает упрощение даже в том случае, когда функция **Simplify** пасует. Неплохо упрощаются тригонометрические функции, особенно при использовании опции **ComplexityFunction**, подсказывающей путь упрощения.

7.5. Раскрытие и расширение выражений

7.5.1. Функции раскрытия и расширения выражений

Расширение или *раскрытие* выражений – еще одна типовая операция компьютерной алгебры. По смыслу она противоположна упрощению выражений. Часто компактная форма представления выражений обусловлена определенными операциями по упрощению выражений. Существует множество выражений, для которых эти правила известны. Например, мы знаем, что выражение

$$(a - b)^2 = (a - b) * (a - b)$$

можно представить как $a^2 - 2*a*b + b^2$.

Разумеется, такое соответствие существует далеко не всегда. К примеру, выражение в виде числа 1 вовсе не является представлением только выражения $\sin(x)^2 + \cos(x)^2$.

Ниже представлены основные функции, возвращающие результаты с раскрытием и расширением выражений:

- **ComplexExpand[expr]** – раскрывает expr, полагая все переменные вещественными;
- **ComplexExpand[expr, {x1, x2, ...}]** – раскрывает expr, полагая переменные, соответствующими какому-либо действительному;
- **FunctionExpand[expr]** – раскрывает выражения expr, содержащие специальные функции;
- **Expand[expr]** – раскрывает произведения и положительные целые степени в expr;
- **Expand[expr, patt]** – не выполняет расширение для тех элементов expr, которые не содержат соответствующие шаблону patt члены;
- **ExpandAll[expr]** – раскрывает все произведения и целочисленные степени в любой части expr;
- **ExpandAll[expr, patt]** – исключает из операции расширения те части expr, которые не содержат соответствующие шаблону patt члены;
- **ExpandDenominator[expr]** – раскрывает произведение и степени, которые присутствуют в выражении expr в роли знаменателей;
- **ExpandNumerator[expr]** – раскрывает произведения и степени в числителе выражения expr;

- **PowerExpand[expr]** – раскрывает вложенные степени, степени произведений, логарифмы от степеней и логарифмы от произведений. Осторожно используйте **PowerExpand**, так как эта функция не реагирует на разрывный характер выражения expr.

7.5.2. Примеры расширения и раскрытия выражений

Приведем примеры на операции расширения выражений **Expand**:

```
Expand[(x-a)*(x-b)*(x-c)]
-abc + abx + acx +
bcx - ax2 - bx2 - cx2 + x3
Simplify[%]
-(a-x)(-b+x)(-c+x)
Expand[(Sin[x]+Cos[x])/(Cos[x]*Sin[x])]
Csc[x]+Sec[x]
Simplify[%]
Csc[x]+Sec[x]
Expand[2*Cos[x]^2,Trig->True]
2Cos[x]2
Simplify[%]
2Cos[x]2
Expand[Sin[x]^2+Cos[y]^2]
Cos[y]2 + Sin[x]2
Expand[Sin[x]^2+Cos[y]^2,Trig->True]
1 -  $\frac{\cos^2[x]}{2} + \frac{\cos^2[y]}{2}$ 
 $\frac{\sin^2[x]}{2} - \frac{\sin^2[y]}{2}$ 
Simplify[%]
 $\frac{1}{2} (2 - \cos[2x] + \cos[2y])$ 
```

В этих примерах полезно обратить внимание на то, что далеко не всегда последовательное применение функций **Expand** и **Simplify** дают исходное выражение. Гораздо чаще получается новое выражение, порою представляющее ценность. При операциях с тригонометрическими выражениями нередко нужно использовать опцию **Trig->True**, намечая тригонометрический путь решения. В противном случае может просто наступить отказ от выполнения операции **Expand** с заданным выражением, и оно будет просто повторено в строке вывода.

Приведем примеры на другие функции расширения выражений:

```
ExpandAll[Sin[2*Cos[x]],Trig->True]
Cos[Cos[x]+i Sin[x]] Sin[Cos[x]-i Sin[x]]+Cos[Cos[x]-i Sin[x]]
Sin[Cos[x]+i Sin[x]]
Simplify[%]
Sin[2 Cos[x]]
```

ExpandNumerator[(1+x)^2/x]

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

ExpandDenominator[(1-x)^2/(1+x)^2]

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

ComplexExpand[Sin[a+I*b]]

$$\text{Cosh}[b] \text{Sin}[a] + i \text{Cos}[a] \text{Sinh}[b]$$

ComplexExpand[(a+b I)/(x+d I)]

$$\frac{bd}{d^2 + x^2} + \frac{ax}{d^2 + x^2} + i \left(-\frac{ad}{d^2 + x^2} + \frac{bx}{d^2 + x^2} \right)$$

Simplify[%]

$$\frac{-i a + b}{d - i x}$$

PowerExpand[Sqrt[a^2*b*c]]

$$a \sqrt{b} \sqrt{c}$$

FunctionExpand[Gamma[4,x]]

$$e^{-x} x^3 + 3 (e^{-x} x^2 + 2 (e^{-x} + e^{-x} x))$$

FunctionExpand[Beta[4,2+x]]

$$\frac{6}{(2+x)(3+x)(4+x)(5+x)}$$

FunctionExpand[Zeta[3,2+x]]

$$\frac{1}{2} \left(-2 \left(\frac{1}{x^3} + \frac{1}{(1+x)^3} \right) - \text{PolyGamma}[2, x] \right)$$

Разумеется, этими примерами далеко не исчерпываются возможности данной группы функций. Рекомендуется опробовать примеры из справочной системы данных Mathematica и свои собственные примеры.

7.5.3. Функция *Collect*

К операциям, расширяющим выражения, относятся также функции:

- **Collect**[expr, x] – выполняет приведение общих членов выражения по степеням переменной x;
- **Collect**[expr, {x1, x2, ...}] – приведение общих членов выражения по степеням переменных x1, x2,

Эта операция особенно полезна, если результат можно представить в виде степенных многочленов. Проиллюстрируем это следующими примерами:

Collect[(x-1)*(x-2)*(x^2-9), x]

$$-18 + 27x - 7x^2 - 3x^3 + x^4$$

Collect[a*x^2+b*x*y+c*y+d*y^2, x]

$$ax^2 + cy + bxy + dy^2$$

Collect[a*x^2+b*x*y+c*y+d*y^2, y]

$$ax^2 + (c + bx)y + dy^2$$

(5+x^2)*(x-1)*x


```
(-1 + x) x (5 + x^2)
Collect[%,x]
-5x + 5x^2 - x^3 + x^4
```

Другой пример показывает применение функции **Collect** к выражению с двумя переменными:

```
Collect[(x-1)*(y-3)*(x-2)*(y-2)*(x-1),y,x]
yx[-5(-2+x)(-1+x)^2] +
y^2x[(-2+x)(-1+x)^2] + x[6(-2+x)(-1+x)^2]
```

Разумеется, как и в случае упрощения выражений, их расширение не является однозначной операцией и предполагает наличие определенных условностей. Опытный пользователь, используя опции функций, обычно без труда может получить результат в нужной форме.

7.5.4. Функции преобразования тригонометрических выражений

Хотя представленные выше функции иногда применимы для тригонометрических выражений, для последних есть ряд специальных функций, дающих более надежные результаты в ходе преобразований тригонометрических функций. В названии этой группы функций имеется слово **Trig**. Начнем с функции **TrigExpand**.

TrigExpand[expr] – обеспечивает расширение выражения *expr*, содержащего тригонометрические и гиперболические функции.

Представленные ниже примеры иллюстрируют работу этой функции:

```
TrigExpand[Sin[a+b]]
Cos[b] Sin[a]+Cos[a] Sin[b]
TrigExpand[Cos[3*x]]
Cos[x]^3 - 3Cos[x]Sin[x]^2
TrigExpand[Sinh[2*x]]
2 Cosh[x] Sinh[x]
TrigExpand[Sin[Cos[Tan[x]^2]]]
Cos[1/2 Cos[Tan[x]^2] + 1/2 I Sin[Tan[x]^2]]
Sin[1/2 Cos[Tan[x]^2] - 1/2 I Sin[Tan[x]^2]] +
Cos[1/2 Cos[Tan[x]^2] - 1/2 I Sin[Tan[x]^2]]
Sin[1/2 Cos[Tan[x]^2] + 1/2 I Sin[Tan[x]^2]]
- 1/2 - Cos[x]^6/2 + 2 Cos[x] Sin[x] +
15/2 Cos[x]^4 Sin[x]^2 - 15/2 Cos[x]^2 Sin[x]^4 + Sin[x]^6/2
TrigExpand[Sin[2 ArcCoth[t]]]
2 Cos[ArcCoth[t]] Sin[ArcCoth[t]]
```

Следующая пара функций:

- **TrigToExp[expr]** – преобразует тригонометрические выражения к экспоненциальному виду.
- **ExpToTrig[expr]** – преобразует экспоненциальные выражения в тригонометрические.

Примеры на применение этих функций:

TrigToExp[Cos[z]]

$$\frac{e^{-i\xi}}{2} + \frac{e^{i\xi}}{2}$$

ExpToTrig[%]

Cos[z]

f:=Sinh[z]+Cosh[z]

TrigToExp[f]

$$e^{\xi}$$

ExpToTrig[%]

Cosh[z]+Sinh[z]

TrigToExp[Sin[x]/Cos[y]]

$$i \left(\frac{e^{-i\xi} - e^{i\xi}}{e^{-i\psi} + e^{i\psi}} \right)$$

ExpToTrig[%]

Sec[y] Sin[x]

Приведем еще две функции:

- **TrigFactor[expr]** – раскладывает на простые множители тригонометрическое выражение expr;
- **TrigFactorList[expr]** – раскладывает тригонометрическое выражение expr на листы с термами выражения.

Следующие примеры показывают применение этих функций:

expr=TrigExpand[Sin[a+b]^3]

$$\begin{aligned} & \frac{3}{4} \cos[b] \sin[a] - \frac{3}{4} \cos[a]^2 \cos[b]^3 \sin[a] + \\ & \frac{1}{4} \cos[b]^3 \sin[a]^3 + \frac{3}{4} \cos[a] \sin[b] - \\ & \frac{3}{4} \cos[a]^3 \cos[b]^2 \sin[b] + \\ & \frac{9}{4} \cos[a] \cos[b]^2 \sin[a]^2 \sin[b] + \\ & \frac{9}{4} \cos[a]^2 \cos[b] \sin[a] \sin[b]^2 - \\ & \frac{3}{4} \cos[b] \sin[a]^3 \sin[b]^2 + \\ & \frac{1}{4} \cos[a]^3 \sin[b]^3 - \frac{3}{4} \cos[a] \sin[a]^2 \sin[b]^3 \end{aligned}$$

TrigFactor[expr]

Sin[a + b]^3

TrigFactorList[expr]

{ {1,1}, {Sin[a+b],3} }

TrigExpand[Cosh[Sin[x*y]]]

$$\begin{aligned} & \cos\left[\frac{1}{2}\cos[xy] - \frac{1}{2}i\sin[xy]\right] \\ & \cos\left[\frac{1}{2}\cos[xy] + \frac{1}{2}i\sin[xy]\right] + \\ & \sin\left[\frac{1}{2}\cos[xy] - \frac{1}{2}i\sin[xy]\right] \\ & \sin\left[\frac{1}{2}\cos[xy] + \frac{1}{2}i\sin[xy]\right] \end{aligned}$$

TrigFactorList[%]

{ {1,1}, {Cosh[Sin[x y]],1} }

TrigReduce[expr] – упрощает выражения с произведениями тригонометрических функций.

Примеры на применение этой функции:

TrigReduce[2*Sin[x]*Cos[y]]

Sin[x-y]+Sin[x+y]

TrigReduce[Cosh[x]*Tanh[x]]

Sinh[x]

TrigReduce[Sin[x]^2+Cos[x]^2]

1

TrigReduce[Sin[x]*Cos[x]]

$\frac{1}{2} \sin[2x]$

TrigReduce[Sinh[x/y]^3]

$\frac{1}{4} \left(-3 \sinh\left[\frac{x}{y}\right] + \sinh\left[\frac{3x}{y}\right] \right)$

Применение этих функций расширяет круг задач, решаемых с применением символьных преобразований.

7.6. Функции и директивы для работы с полиномами

7.6.1. Определение полинома (степенного многочлена)

Полиномом называют выражение, состоящее из нескольких частей одного вида. В западной математической литературе к ним часто относят *степенной многочлен* вида

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_n x^n.$$

Хотя термин «полином» не очень прижился в отечественной математической литературе, мы оставляем его ввиду краткости и ради лучшего понимания син-

таксиса функций системы, поскольку слова `poly` и `Polynomial` входят в параметры и имена многих функций. При этом полиномы мы будем кратко обозначать как `poly` или `pi` (*i* – индекс или порядковый номер полинома).

7.6.2. Основные операции над полиномами

Над полиномами можно выполнять обычные арифметические операции: сложение, вычитание, умножение и деление. Это иллюстрируют следующие примеры (`p1` и `p2` – полиномы от одной переменной *x*):

```
p1:=x^3+2*x^2+3*x+4
p2:=x^2-1
p1+p2
      3   3x   + 3x2 + x3
p1-p2
      5   3x   + x2 + x3
Expand[p1*p2]
      -4 - 3x + 2x2 + 2x3 + 2x4 + x5
p1/p2
      4 + 3x + 2x2 + x3
      -----
      -1 + x2
```

Если ситуация со сложением и вычитанием полиномов достаточно очевидна, то с умножением и делением результат часто повторяет задание. Для получения результирующего полинома в обычной форме при умножении полиномов следует использовать функцию расширения символьных выражений **Expand**.

Если полином делится на другой (это бывает далеко не всегда), то для получения результата необходимо использовать функцию **Simplify**. В общем случае при делении полиномов может оставаться остаток. Функция, обеспечивающая деление полиномов и вычисляющая остаток, описана ниже.

7.6.3. Разложение полиномов – функции класса *Factor*

Разложение чисел, математических выражений и особенно полиномов на простые множители является столь же распространенной операцией, что и функции **Simplify**, **Collect** и **Expand**. Имеется целый ряд функций, в названии которых есть слово **Factor**, и которые решают указанные задачи:

- **Factor[poly]** – выполняет разложение полинома над целыми числами;
- **Factor[poly, Modulus->p]** – выполняет разложение полинома по модулю простого *p*;
- **FactorInteger[n]** – возвращает список простых множителей целого числа *n* вместе с их показателями степеней. Опция **FactorComplete** позволяет указать, следует ли выполнять полное разложение;

- **FactorList[poly]** – возвращает список множителей полинома с их показателями степени. Опция **Modulus->p** позволяет представить множители полинома по модулю простого p.
- **FactorSquareFree[poly]** – записывает полином в виде произведения множителей, свободных от квадратов. Опция **Modulus->p** позволяет представить разложение полинома по модулю простого p.
- **FactorSquareFreeList[poly]** – возвращает список множителей полинома, свободных от квадратов, вместе с показателями степени. Может использоваться опция **Modulus->p**.
- **FactorTerms[poly]** – извлекает полный (общий) числовой множитель в poly.
- **FactorTermsList[poly]** – возвращает лист всех общих числовых множителей полинома poly.

Ниже представлен ряд примеров на применение этих функций:

```
Factor[x^3-6*x^2+11*x-6]
(-3+x) (-2+x) (-1+x)
Factor[x^3-6*x^2+21*x-52]
(-4 + x) (13 - 2x + x^2)
Factor[x^5+8*x^4+31*x^3+80*x^2+94*x+20,Modulus->3]
(1 + x) / (2 + x)
FactorList[x^4-1,Modulus->2]
{{1,1},{1+x,4}}
FactorSquareFree[(x^2+1)*(x^4-1)]
(-1 + x^2) (1 + x^2)
FactorSquareFree[(x^2+1)*(x^4-1),Modulus->2]
(1+x)
FactorSquareFreeList[(x^2+1)*(x^4-1),Modulus->2]
{{1,1},{1+x,6}}
FactorTerms[2*x^2+4*x+6]
2(3 + 2x + x^2)
FactorTermsList[2*x^2+4*x+6]
{2, 3 + 3x + x^2}
FactorInteger[123456789]
{{3,2},{3607,1},{3803,1}}
FactorList[x^4-1]
{{1,1},{-1 + x, 1}, {1 + x, 1}, {1 + x^2, 1}}
FactorSquareFreeList[(x^2+1)*(x^4-1)]
{{1,1},{-1 + x^2, 1}, {1 + x^2, 2}}
```

Обычно функция **Factor** выявляет внутреннюю суть полинома, раскладывая его множители, содержащие корни полинома. Однако в ряде случаев корни полинома удобнее получать в явном виде с помощью уже рассмотренной функции **Roots**.

Функция **Factor** может работать и с тригонометрическими выражениями, поскольку многие из них подчиняются правилам преобразований, присущим полиномам. При этом тригонометрический путь решения задается опцией **Trig->True**. Это иллюстрируют следующие примеры:

```
Factor[Csc[x]+Sec[x],Trig->True]
Csc[x] Sec[x] (Cos[x]+Sin[x])
Factor[Sin[3*x],Trig->True]
(1+2 Cos[2 x]) Sin[x]
```

7.6.4. Функции для работы с полиномами

Имеется множество функций, большей частью достаточно очевидных для знакомого с математикой пользователя, – функций для работы с полиномами:

- **Decompose[poly, x]** – выполняет разложение полинома, если это возможно, на более простые полиномиальные множители.
- **GroebnerBasis[{poly1, poly2, ...}, {x1, x2, ...}]** – возвращает список полиномов, которые образуют базис Гробнера для идеала, порожденного полиномами polyi.
- **PolynomialDivision[p, q, x]** – возвращает список частного и остатка, полученных делением полиномов p и q от x.
- **PolynomialGCD[poly1, poly2, ...]** – возвращает наибольший общий делитель ряда полиномов poly1, poly2, С опцией **Modulus->p** функция возвращает GCD по модулю простого p.
- **PolynomialLCM[poly1, poly2, ...]** – возвращает наименьшее общее кратное полиномов poly1, poly2, С опцией **Modulus->p** функция возвращает LCM по модулю простого p.
- **PolynomialMod[poly, m]** – возвращает полином poly, приведенный по модулю m.
- **PolynomialMod[poly, {m1, m2, ...}]** – выполняет приведение по модулю всех mi.
- **PolynomialQ[expr, var]** – выдает значение True, если expr является полиномом от var, иначе дает False.
- **PolynomialQ[expr, {var1, ...}]** – проверяет, является ли expr полиномом от vari.
- **PolynomialQuotient[p, q, x]** – возвращает частное от деления p и q, как полиномов от x, игнорируя какой-либо остаток.
- **PolynomialRemainder[p, q, x]** – возвращает остаток от деления p на q, как полиномов от x.
- **Resultant[poly1, poly2, var]** – вычисляет результат полиномов poly1 и poly2 по переменной var. С опцией **Modulus->p** функция вычисляет результат по модулю простого p.

7.6.5. Примеры работы с полиномами

Итак, работа с этими функциями по существу сводит операции с таким сложным видом символьных данных, как многочлены, к типовым алгебраическим операциям над обычными символьными переменными. Следующие примеры поясняют работу с полиномами:

```

P[x] := a*x^3+b*x^2+c*x+d
Q[x] := e*x^2-f*x-1
Null2
Collect[P[x]+Q[x],x]
-1 + d + (b + e)x2 + ax3 + x(c - Cosh[z] - Sinh[z])
Collect[P[x]*Q[x],x]
-d + aex5 + x4(be - a(Cosh[z] + Sinh[z])) +
x3(-a + ce - b(Cosh[z] + Sinh[z])) +
x2(-b + de - c(Cosh[z] + Sinh[z])) +
x(-c - d(Cosh[z] + Sinh[z]))
{PolynomialQ[P[x]],PolynomialQ[Q[x]]}
{True,True}
PolynomialQ[Sin[x],x]
False
PolynomialQ[P[x]+Q[x]]
True
Decompose[P[x],x]
{d + cx + bx2 + ax3}
PolynomialQuotient[P[x],Q[x],x]

$$\frac{b}{e} + \frac{ax}{e} + \frac{a \cosh[z]}{e^2} + \frac{a \sinh[z]}{e^2}$$

PolynomialRemainder[Q[x],P[x],x]
-1 + ex2 - x(Cosh[z] + Sinh[z])
CoefficientList[P[x],x]
{d,c,b,a}
Decompose[x^6+x+1-x^3+2*x^5,x]
{1 + x - x3 + 2x5 + x6}
PolynomialGCD[P[x],Q[x]]
1
PolynomialLCM[P[x],Q[x]]
(d + cx + bx2 + ax3)(-1 + ex2 - xCosh[z] - xSinh[z])
PolynomialQuotient[3*x^3-2*x^2+x,x^2-x+1,x]
1+3 x
PolynomialRemainder[3*x^3-2*x^2+x,x^2-x+1,x]
-1-x
Reduce[a*x^2+b*x+c==0,x]

$$a \neq 0 \&\& \left( x = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \mid \mid x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right) \mid \mid$$


$$a = 0 \&\& b \neq 0 \&\& x = -\frac{c}{b} \mid \mid c = 0 \&\& b = 0 \&\& a = 0$$


```

Полиномы широко используются в математических расчетах. Поэтому обилие функций по работе с ними облегчает проведение сложных математических расчетов и позволяет представлять в достаточно простой и удобной форме. Если бы системы компьютерной алгебры работали бы только с одними полиномами, то и в этом случае они вполне оправдали бы себя в глазах многих математиков.

7.7. Расширенные операции с выражениями

7.7.1. Функции для расширенных операций с выражениями

Выше была описана сравнительно немногочисленная группа функций для работы с выражениями: их упрощения, расширения, выделения множителей и т.д. Эти функции способны решать большинство повседневных задач, связанных с аналитическими преобразованиями выражений. Однако система Mathematica имеет гораздо более полный набор функций для работы с выражениями. В этом параграфе сосредоточено описание основных функций для расширенных операций с выражениями.

Для расширенной работы с выражениями служат следующие функции:

- **AlgebraicRules[eqns, {x1, x2, ...}]** – формирует множество алгебраических правил, которые замещают переменные, ранее находившиеся в списке x_i , на более поздние в списке, соответствующем уравнению (или уравнениям) eqns.
- **Apart[expr]** – возвращает expr, записывая заново рациональное выражение как сумму членов с минимальными знаменателями.
- **Apart[expr, var]** – аналогична **Apart[expr]**, но все переменные, кроме var, интерпретируются как константы.
- **ApartSquareFree[expr, var]** – возвращает expr как сумму членов со знаменателями, свободными от квадратов, по переменной var.
- **Catch[expr]** – возвращает аргумент первого Throw, генерируемого при вычислении expr.
- **Check[expr, failexpr]** – вычисляет expr и возвращает его результат, если только не вырабатывались сообщения, иначе вычисляет и возвращает failexpr.
- **Check[expr, failexpr, s1::t1, s2::t2, ...]** – выполняет контроль только для указанных сообщений.
- **CheckAbort[expr, failexpr]** – вычисляет expr, возвращая failexpr в случае прерывания.
- **Coefficient[expr, form]** – возвращает коэффициент перед form в полиномиальном выражении expr.
- **Coefficient[expr, form, n]** – возвращает коэффициент перед form^n в выражении expr.
- **CompoundExpression** – применяется в виде: **expr1; expr2; ...** – вычисляет expr1 по очереди, возвращая последнее как результат.
- **Depth[expr]** – возвращает максимальное число индексов, требуемых для указания любой части выражения expr, плюс единица.
- **Dimensions[expr]** – возвращает список размерностей выражения expr.

- **Dimensions[expr, n]** – возвращает список размерностей expr, пониженного до уровня n.
- **Edit[expr_____]** – предоставляет на редактирование выражения expr.
- **Evaluate[expr]** – вычисляет выражение expr безусловно, т.е. даже если оно оказывается аргументом функции, чьи атрибуты определяют его невычислимым.
- **Exponent[expr, form]** – возвращает максимальную степень, с которой form присутствует в expr.
- **Exponent[expr, form, h]** – применяет h к множеству показателей степеней (экспонент), с которыми form обнаруживается в выражении expr.
- **FlattenAt[expr, {i, j, ...}]** – выравнивает часть выражения expr на позиции {i, j, ...}.
- **FlattenAt[expr, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – выравнивает части выражения expr в нескольких позициях.
- **HeldPart[expr, pos]** – извлекает (удаляет) часть или несколько частей, указанных с помощью pos, и помещает их в Hold.
- **Hold[expr]** – содержит expr в невычисленном виде.
- **HoldForm[expr]** – выводит выражение expr, сохраняя при этом его в невычисленной форме.
- **LeafCount[expr]** – возвращает общее (полное) число неделимых подвыражений в expr.
- **Length[expr]** – возвращает число элементов в expr.
- **Level[expr, levelspec]** – возвращает список всех подвыражений выражения expr на уровнях, указанных параметром levelspec.
- **Level[expr, levelspec, f]** – относит f к списку подвыражений.
- **Literal[expr]** – является эквивалентом expr в смысле совпадения формы, но содержит expr в непреобразованном виде.
- **LogicalExpand[expr]** – выполняет расширение выражений, содержащих логические связи, как, например, && и ||.
- **MapAt[f, expr, n]** – применяет f к элементу на позиции n в expr. Если n отрицательно, позиция отсчитывается с конца.
- **MapAt[f, expr, {i, j, ...}]** – применяет f к частям expr на позиции {i, j, ...}.
- **MapAt[f, expr, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – применяет f к частям expr в ряде позиций.
- **MapIndexed[f, expr]** – применяет f к элементам expr, возвращая спецификацию части каждого элемента в качестве второго аргумента в f.
- **MapIndexed[f, expr, levspec]** – применяет f ко всем частям expr на уровнях, указанных с помощью levspec.
- **Order[expr1, expr2]** – возвращает 1, если expr1 предшествует expr2 в канонической последовательности, и дает -1, если expr1 стоит после expr2 в каноническом порядке. Результатом будет 0, если expr1 тождественно expr2.
- **Postfix[f[expr]]** – выводит с f[expr], заданной по умолчанию в постфиксной форме: expr // f.

- **Postfix[f[expr], h]** – выводит в виде exprh.
- **Prepend[expr, elem]** – возвращает expr, к которому предварительно добавлен elem.
- **Print[expr1, expr2, ...]** – выводит на экран дисплея выражения expr1 и затем дает перевод строки. Может использоваться для создания диалога.
- **Return[expr]** – возвращает величину expr из функции.
- **Run[expr1, expr2, ...]** – создает выводимую форму выражений expr1, разделенных пробелами, и выполняет ее как внешнюю команду операционной системы.
- **RunThrough["command", expr]** – выполняет внешнюю команду и возвращает результат вычисления expr.
- **Scan[f, expr]** – вычисляет f, применяемую к каждому элементу expr по очереди.
- **Scan[f, expr, levelspec]** – применяет f к частям выражения expr, указанным с помощью levelspec.
- **SequenceForm[expr1, expr2, ...]** – печатает в виде текстовой конкатенации (объединения) печатных форм выражений expr1.
- **SetAccuracy[expr, n]** – дает вариант expr, в котором все числа должны быть установлены с точностью n цифр.
- **SetPrecision[expr, n]** – вырабатывает вариант expr, в котором все числа установлены с точностью представления n цифр.
- **Share[expr]** – меняет способ внутреннего хранения выражения expr, что минимизирует объем используемой памяти.
- **Through[expr, h]** – выполняет преобразование всюду, где h появляется в заголовке выражения expr.
- **Together[expr]** – приводит члены суммы к общему знаменателю и сокращает множители в полученном результате.
- **Variables[expr]** – возвращает список всех независимых переменных в выражении.
- **With[{x = x0, y = y0, ...}, expr]** – указывает, что в случае обнаружения в выражении expr символов x, y, ... они должны быть заменены на x0, y0....
- **Write[channel, expr1, expr2, ...]** – записывает в указанный выходной канал channel последовательно (один за другим) выражения expr1, завершаемые newline (переводом строки).
- **WriteString[channel, expr1, expr2, ...]** – превращает expr1 в символьные цепочки, а затем последовательно записывает их в указанный выходной канал channel.

7.7.2. Примеры расширенной работы с выражениями

К сожалению, объем книги не позволяет привести примеры на все эти функции, да и вряд ли они будут интересны всем читателям. Поэтому приведем лишь отдельные примеры работы с некоторыми из этих функций:

```

Apart[(x^4+1)/(x^2-1)]
1 +  $\frac{1}{-1+x}$  + x^2 -  $\frac{1}{1+x}$ 
Apart[(x^3-y^3-1)/(x^2-y), y]
x^4 + x^2 y + y^2 +  $\frac{1-x^3+x^6}{-x^2+y}$ 
Cancel[(x^2-1)/(x-1)]
1+x
Denominator[(x^2-x-1)/(x-1)]
-1+x
Numerator[(x^2-x-1)/(x-1)]
-1 - x + x^2
Depth[x^3+x^2+x+1]
3
Dimensions[x^3-2*x^2+1]
{3}
Evaluate[1+1+Sin[1]]
2+Sin[1]
Head[Sin[x]]
Sin

```

Обилие функций для работы с математическими выражениями позволяет с помощью системы Mathematica решать самые серьезные задачи символьной математики (компьютерной алгебры). Разумеется, для этого требуется время на полное освоение системы и серьезный опыт практической работы с ней. Он приходит лишь спустя год-два постоянной и интенсивной работы с системой.

7.7.3. Средства работы с выражениями в Mathematica 6

Mathematica 6 для работы с выражениями использует, в основном, описанные выше средства. Добавлено сравнительно немного новых функций. Например, это функция вычисления дискриминанта полинома poly переменной var:

Discriminant[poly, var] **Discriminant[poly, var, Modulus->p]**

Приведенные ниже примеры демонстрируют применение этой функции:

```

Discriminant[a x^2+b x+c,x]
b^2-4 a c
Discriminant[2 x^2+3 x+4,x]
-23
Discriminant[2 x^2+3 x+4,x, Modulus?2]
1

```

Функция **CountRoots** находит число корней полинома, в том числе в заданном интервале изменения x:

CountRoots[poly,x] **CountRoots[poly,{x,a,b}]**

Примеры:

```
CountRoots[(x-1) (x-2) (x-3), {x, 0, 10}]
```

3

```
CountRoots[x^21-1, {x, 0, 1+I}]
```

6

```
CountRoots[(x^3-2) (x^2-4), {x, -Infinity, Infinity}]
```

3

```
CountRoots[(x^4-2) (x^4-3), {x, -2-2 I, 2+2 I}]
```

8

Для ускорения вычислений часто используется схема Горнера, минимизирующая число арифметических операций. Для представления полиномов по схеме Горнера используется следующая функция:

```
HornerForm[poly] HornerForm[poly,vars]
HornerForm[poly1/poly2] HornerForm[poly1/poly2,vars1,vars2]
```

Примеры ее применения даны ниже:

```
HornerForm[a x^3-b x^2+c x+d]
```

d+c x+x² (-b+a x)

```
Expand[%]
```

d+c x-b x²+a x³

```
HornerForm[(9 x^3-3 x^2+2 x+1)/(x^2-2 x+1)]
```

(1+x (2+x (-3+9 x)))/(1+(-2+x) x)

Функция **SymmetricPolynomial** позволяет создавать симметрические полиномы:

```
SymmetricPolynomial[k,{x1,[],xn}]
```

Пример:

```
SymmetricPolynomial[2,{x1,x2,x3,x4}]
```

x¹ x²+x¹ x³+x² x³+x¹ x⁴+x² x⁴+x³ x⁴

Улучшена работа функции полного упрощения FullSimplify. Она стала применима для большего числа классов выражений, в том числе представимых через специальные функции. Например:

```
FullSimplify[x^2 Gamma[x]]
```

x Gamma[1+x]

С другими, куда более редкими, функциями можно познакомиться по справке системы Mathematica 6.

Средства программирования графики

8.1. Построение графиков функций одной переменной	400
8.2. Перестройка и комбинирование графиков ...	409
8.3. Примитивы двумерной графики	410
8.4. Построение графиков в полярной системе координат	412
8.5. Построение контурных графиков	415
8.6. Построение графиков плотности	419
8.7. Построение графиков поверхностей	421
8.8. Примитивы трехмерной графики и их применение	434
8.9. Дополнительные средства графики Mathematica 5.1/5.2 ...	438
8.10. Новые средства графики в Mathematica 6	442
8.11. Функции пакета расширения Graphics	461
8.12. Идеология применения пакета Graphics в Mathematica 6	488

8.1. Построение графиков функций одной переменной

8.1.1. Графическая функция *Plot*

Концептуально графики в системе Mathematica являются *графическими объектами*, которые создаются (возвращаются) соответствующими *графическими функциями*. Их немного – всего около десяти, и они охватывают построение практически всех типов математических графиков. Как отмечалось, достигается это за счет применения опций и директив графики.

Программирование графики в Mathematica относится к функциональному типу и сводится, в основном, к выбору вложенных друг в друга или одиночных функций. Поскольку графики являются объектами, то они могут быть значениями переменных. Mathematica допускает такие конструкции, как:

- `Plot[Sin[x], {x, 0, 20}]` – построение графика синусоиды;
- `g:=Plot[Sin[x], {x, 0, 20}]` – задание объекта – графика синусоиды с отложенным выводом;
- `g=Plot[Sin[x], {x, 0, 20}]` – задание объекта – графика синусоиды с немедленным выводом.

Для построения двумерных графиков функций вида $f(x)$ используется встроенная в ядро функция **Plot**:

- `Plot[f, {x, xmin, xmax}]` – возвращает объект – график функции f аргумента x в интервале от $xmin$ до $xmax$.
- `Plot[{f1, f2, ...}, {x, xmin, xmax}]` – возвращает объект в виде графиков ряда функций f_i .

Функция **Plot** используется для построения одной или ряда линий, дающих графическое представление для указанных функций f, f_1, f_2 и т.д. На рис. 8.1 показано построение графика функции $\sin(x)/x$ без использования каких-либо опций (или точнее с набором опций по умолчанию).

Тут виден как раз тот редкий случай, когда масштаб графика по вертикали оказался выбранным системой по умолчанию неудачно – часть графика сверху просто отсекается. В большинстве же случаев применение функции **Plot** позволяет получить вполне «удобоваримый» график.

8.1.2. Опции функции *Plot*

По мере усложнения задач, решаемых пользователем, его рано или поздно не устроят графики, полученные при автоматическом выборе их стиля и иных параметров. Для точной настройки графиков Mathematica использует специальные **опции** графических функций. Так, с функцией **Plot** используются следующие опции (даны для Mathematica 5.2):

Options[Plot]

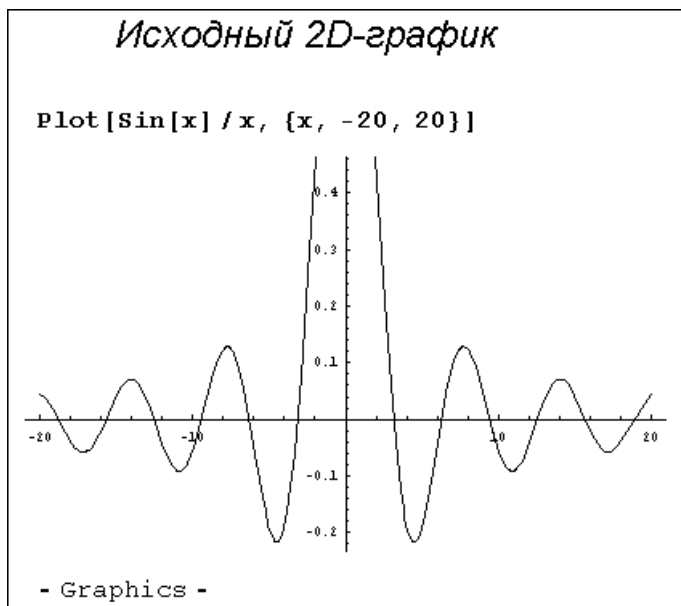


Рис. 8.1. Построение 2D графика

```
{AspectRatio →  $\frac{1}{\text{GoldenRatio}}$ , Axes → Automatic, AxesLabel → None,
  AxesOrigin → Automatic, AxesStyle → Automatic, Background → Automatic,
  ColorOutput → Automatic, Compiled → True, DefaultColor → Automatic,
  DefaultFont → $DefaultFont, DisplayFunction → $DisplayFunction,
  Epilog → {}, FormatType → $FormatType, Frame → False, FrameLabel → None,
  FrameStyle → Automatic, FrameTicks → Automatic, GridLines → None,
  ImageSize → Automatic, MaxBend → 10., PlotDivision → 30.,
  PlotLabel → None, PlotPoints → 25, PlotRange → Automatic,
  PlotRegion → Automatic, PlotStyle → Automatic, Prolog → {},
  RotateLabel → True, TextStyle → $TextStyle, Ticks → Automatic}
```

В приведенном обширном списке опций помимо их названий даны значения опций по умолчанию. Рассмотрим более подробно наиболее важные из опций (знаком * отмечены опции, применяемые как для 2D, так и для 3D графики):

- ***AspectRatio** – задает пропорцию графика – отношение высоты к ширине ($1/\text{GoldenRatio}$ задает отношение по правилу золотого сечения – около 1.618...).
- ***Axes** – задает прорисовку координатных осей (False – осей нет, True – строятся обе оси и {Boolean, Boolean} – задает построение осей отдельно).
- ***AxesLabel** – задает вывод меток для осей в виде {«stringX», «stringY»}.

- **AxesOrigin** – задает начало отсчета для осей (указывает точку пересечения осей).
- ***AxesStyle** – задает стиль вывода осей с помощью ряда директив.
- ***BackGround** – задает цвет фона в одной из трех цветовых систем.
- ***ColorOutput** – задает цвет построений в одной из трех цветовых систем.
- ***DefaultFont** – задает шрифт для текста в графиках.
- **Frame** – задает прорисовку рамки вокруг графика при True и ее отсутствие при False.
- **FrameLabel** – задает надписи на гранях рамки ($\text{FrameLabel} \Rightarrow \{\text{"Text1"}, \text{"Text2"}, \text{"Text3"}, \text{"Text4"}\}$, причем построение идет по часовой стрелке, начиная с нижней надписи).
- **FrameStyle** – задает стиль граней рамки с помощью ряда директив.
- **FrameTicks** – задает прорисовку штриховых меток для граней рамки.
- **GridLines** – задает прорисовку линий сетки.
- ***PlotLabel** – задает вывод титульной надписи ($\text{PlotLabel} \Rightarrow \text{"Text"}$).
- ***PlotRange** – задает масштаб построения в относительных единицах.
- ***PlotRegion** – задает область построения в относительных единицах.
- **RotateLabel** – задает разворот символьных меток на вертикальных осях фрейма с тем, чтобы они стали вертикальными.
- ***Ticks** – устанавливает штриховые метки для осей.

Кроме того, имеется ряд характерных для этой функции дополнительных функций:

- **Compiled** – задает компиляцию функции перед выводом.
- **MaxBend** – задает максимальную кривизну между сегментами кривой.
- **PlotDivision** – задает количество делений при построении гладкой кривой.
- **PlotPoints** – задает число точек выборки, участвующих в построении.
- **PlotStyle** – задает стиль линий или точек графика.

Опции внутри графических функций задаются своим именем name и значением value в виде:

name -> value

Наиболее распространенные символьные значения опций:

- **Automatic** – используется автоматический выбор.
- **None** – опция не используется.
- **All** – используется в любом случае.
- **True** – используется.
- **False** – не используется.

Многие опции могут иметь числовые значения. В сомнительных случаях рекомендуется уточнять форму записи опций и их значений по оперативной справочной системе.

8.1.3. Применение опций функции *Plot*

Мы уже отметили неудачный выбор масштаба в случае, представленном на рис. 8.1. Очевидно, этот недостаток графика легко исправить, введя коррекцию масштаба по оси *y*. Это и сделано в примере, показанном на рис. 8.2. Для изменения масштаба использована опция **PlotRange**.

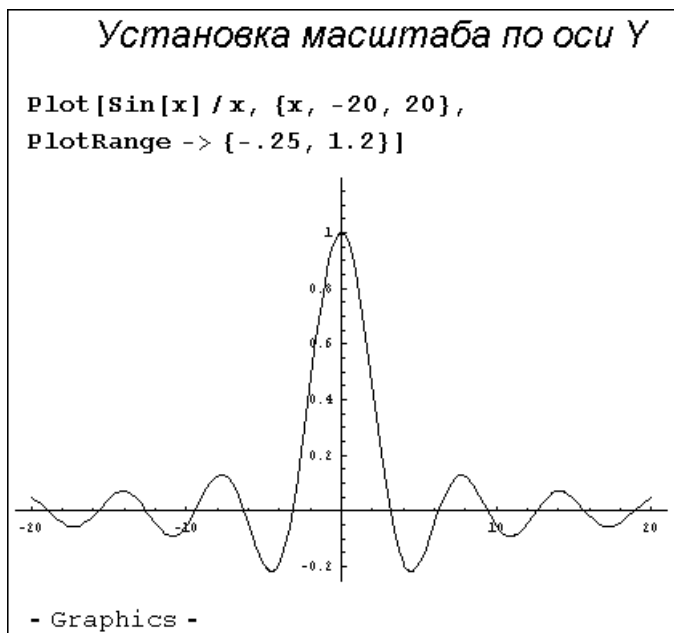


Рис. 8.2. График функции $\sin(x)/x$ с масштабом, дающим его отображение в полном виде

По умолчанию система строит графики, не указывая надписей ни по осям координат (кроме букв x и y), ни в верхней части графика. Такая надпись на графике по центру сверху называется титульной.

Рисунок 8.3 показывает построение графика с надписями у координатных осей. Для создания таких надписей используется опция **AxesLabel**. После нее указывается список, содержащий две надписи: одну для оси x и другую для оси y . Надписи указываются в кавычках.

С помощью опции **Axes** с параметром **None** можно убрать с графика отображение осей. Вид графика при этом показан на рис. 8.4. При его построении, кроме этого, использована опция **PlotLabel** для вывода указанной в качестве ее параметра титульной надписи.

Часто возникает необходимость построения на одном рисунке нескольких графиков одной и той же функции, но при разных значениях какого-либо параметра — например, порядка специальных математических функций. В этом случае они могут быть заданы в табличной форме. Рисунок 8.5 дает пример построения пяти графиков функций Бесселя порядка от 1 до 4.

Рисунок 8.5 иллюстрирует недостаток простого представления на одном рисунке нескольких графиков: все они построены одинаковыми линиями, и потому неясно, какой график к какой функции относится. Рисунок 8.6 показывает возможности управления стилем линий графиков с помощью опции **PlotStyle**.

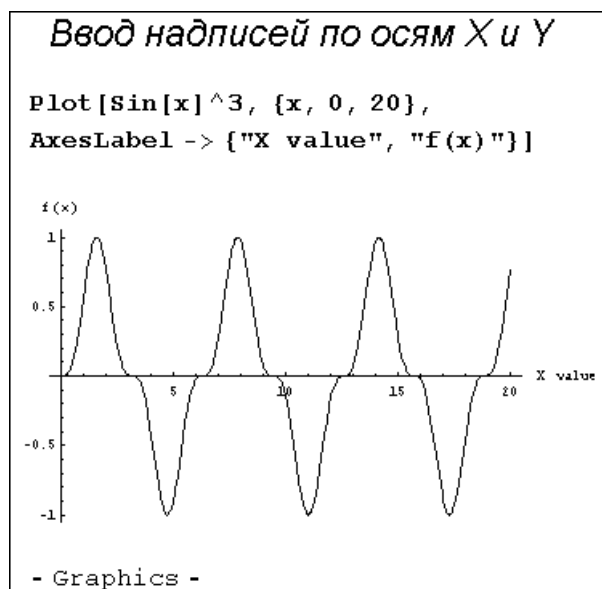


Рис. 8.3. График с надписями по координатным осям

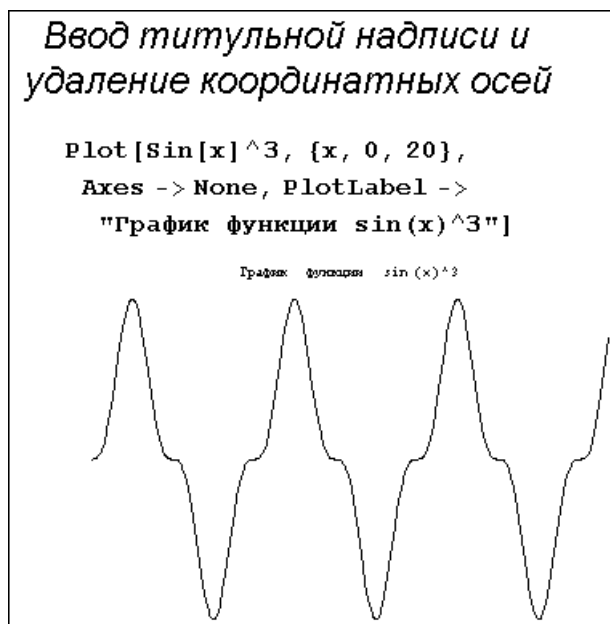


Рис. 8.4. График без координатных осей, но с титульной надписью

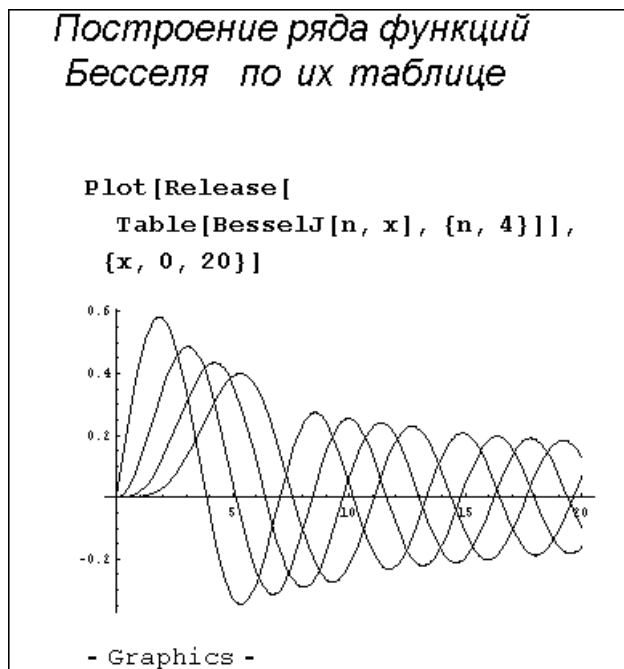


Рис. 8.5. Семейство функций Бесселя на одном графике

Применение других опций позволяет задавать массу других свойств графиков, например, цвет линий и фона, вывод различных надписей и так далее. Помимо приведенных примеров полезно просмотреть и множество примеров на построение двумерных графиков, приведенных в справке по системе Mathematica.

8.1.4. Директивы двумерной графики и их применение

Еще одним важным средством настройки графиков являются *графические директивы*. Синтаксис их подобен синтаксису функций. Однако опции не возвращают объектов, а лишь влияют на их характеристики. Используются также следующие директивы двумерной графики:

- **AbsoluteDashing[{d1, d2, ...}]** – задает построение последующих линий пунктиром со смежными (последовательными) сегментами, имеющими абсолютные длины d1, d2, ... (повторяемые циклически). Значения длины d_i задаются в пикселах.
- **AbsolutePointSize[d]** – задает построение последующих точек графика в виде кружков с диаметром d (в пикселах).
- **AbsoluteThickness[d]** – задает абсолютное значение толщины для последующих рисуемых линий (в пикселах).

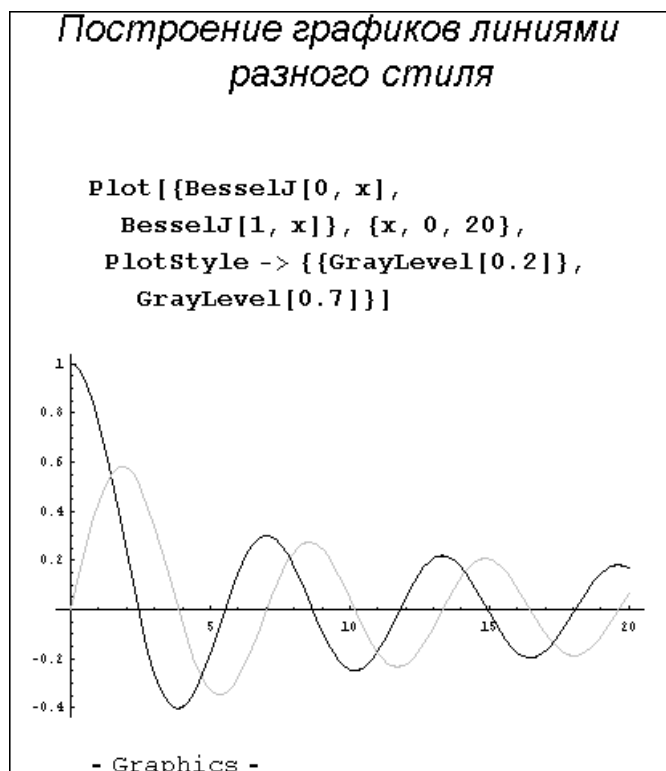


Рис. 8.6. Построение графиков линиями разного стиля

- **Dashing[{r1, r2,...}]** – задает построение последующих линий пунктиром с последовательными сегментами длиной r1, r2, ..., повторяемыми циклически, причем ri задается как дробная часть от полной ширины графика.
- **PointSize[d]** – задает вывод последующих точек графика в виде кружков с относительным диаметром d, заданным как дробная часть от общей ширины графика.
- **Thickness[r]** – устанавливает толщину r для всех последующих линий, заданную как дробная часть от полной ширины графика.

Рисунок 8.7 показывает построение графика функции Бесселя в виде пунктирной линии. Она задается применением графической директивы **Dashing**.

Применение графических директив совместно с опциями позволяет создавать графики самого различного вида, вполне удовлетворяющие как строгим требованиям, так и различным «извращениям» в их оформлении.



Рис. 8.7. Построение графика функции Бесселя с применением графической директивы **Dashing**

8.1.5. Построение графика по точкам – функция **ListPlot**

Часто возникает необходимость построения графика по точкам. Это обеспечивает встроенная в ядро графическая функция **ListPlot**:

- **ListPlot[{y1, y2, ...}]** – выводит график списка величин. Координаты x для каждой точки принимают значения 1, 2, ...
- **ListPlot[{x1, y1}, {x2, y2}, ...]** – выводит график списка величин с указанными x - и y -координатами.

В простейшем случае (рис. 8.8) она задает сама значения координаты $x=0,1,2,3,\dots$ и строит точки на графике с координатами (x,y) , выбирая y последовательно из списка координат.

Можно подметить характерный недостаток построений: точки (особенно при небольшом размере) имеют вид, заметно отличающийся от закрашенной идеальной окружности – круга. Эта функция, особенно в ее второй форме (с заданными координатами x и y), удобна для вывода на график экспериментальных точек.

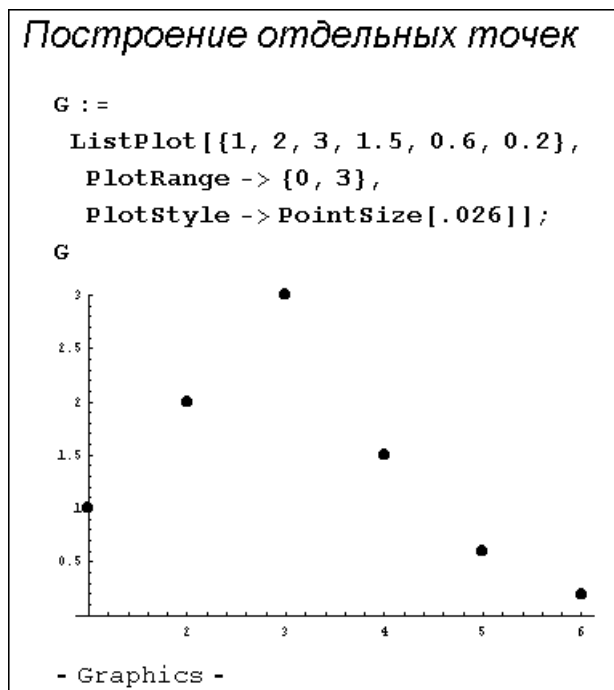


Рис. 8.8. Построение ряда точек графика

8.1.6. Получение информации о графических объектах

Порой некоторые детали построения графиков оказываются для пользователя неожиданными и не вполне понятными. Причина этого кроется во множестве опций, которые могут использоваться в графиках, причем в самых различных сочетаниях. Поэтому полезно знать, как можно получить информацию о свойствах графических объектов. Порой небольшая модификация опций (например, замена цвета линий или фона) делает график полностью удовлетворяющим требованиям пользователя.

Следующие функции дают информацию об опциях графического объекта g:

- **FullAxes[g]** – возвращает список опций координатных осей;
- **Options[g]** – возвращает упрощенный список опций;
- **FullOptions[g]** – возвращает полный список опций;
- **InputForm[g]** – возвращает информацию о графике (включая таблицу точек).

В списке функции **FullOptions** имеются численные данные обо всех параметрах графика. Аналогично можно получить и иные данные, они не приводятся вви-

ду громоздкости выводимой информации. Анализ графиков с применением этих функций может оказаться весьма полезным при задании построения сложных графиков и их редактировании.

Рекомендуется просмотреть с помощью этих функций опции графического объекта, например

```
g:=Plot[Sin[x],{x,-10,10}];
```

Ввиду громоздкости списков опций они не приводятся. Функции **OptionsFull** и **Options** можно также использовать в виде:

- **Options[g, option]** – возвращает значение указанной опции;
- **FullOptions[g, option]** – возвращает значение указанной опции;

В этом случае можно получить информацию по отдельной опции.

8.2. Перестройка и комбинирование графиков

8.2.1. Директива Show

При построении графиков требуется изменение их вида и тех или иных параметров и опций. Этого можно достичь повторением вычислений, но при этом скорость работы с системой заметно снижается. Для ее повышения удобно использовать специальные функции перестройки и вывода графиков, учитывающие, что их узловые точки уже рассчитаны и большая часть опций уже задана.

В этом случае удобно использовать следующую функцию-директиву:

- **Show[plot]** – построение графика;
- **Show[plot, option -> value]** – построение графика с заданной опцией;
- **Show[plot1, plot2,...]** – построение нескольких графиков с наложением их друг на друга.

Директива **Show** полезна также и в том случае, когда желательно, не трогая исходные графики, просмотреть их при иных параметрах. Соответствующие опции, меняющие параметры графиков, можно включить в состав директивы **Show**. Другое полезное применение директивы – объединение на одном графике нескольких графиков различных функций или экспериментальных точек и графика теоретической зависимости.

8.2.2. Примеры применения функции Show

Рисунок 8.9 показывает создание двух графических объектов g1 и g2 с отложенным построением, а затем уже построение графиков их функций и применение директивы **Show** для создания объединенного графика. В этом случае директива **Show** строит вначале исходные графики отдельно, а затем объединенный график. В приведенных ниже примерах оставлен только объединенный график, другие удалены командой **Clear** в позиции **Edit** главного меню.

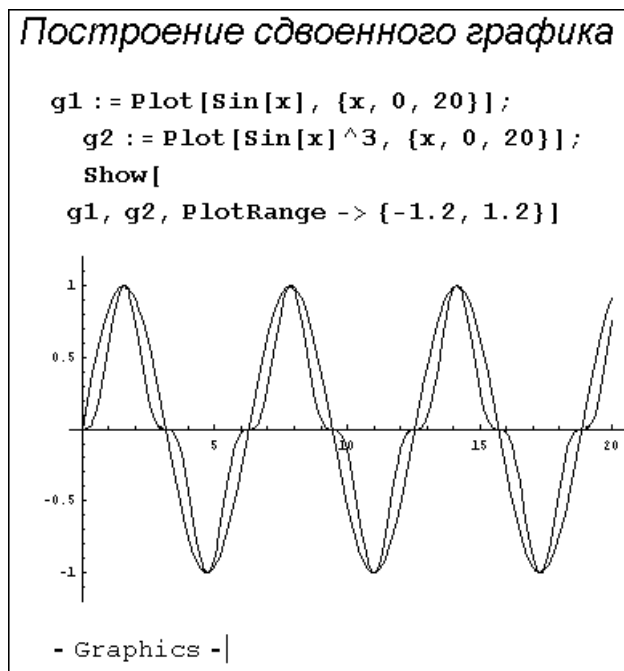


Рис. 8.9. Построение двух графических объектов и их объединение

Разумеется, при использовании директивы **Show** следует побеспокоиться о выравнивании масштабов графиков, налагаемых друг на друга. Полезно особо обратить внимание на возможность присвоения переменным (в нашем примере $g1$ и $g2$) в качестве значений графиков функций. Такие переменные становятся графическими объектами, используемыми директивой **Show** для вывода на экран дисплея.

Директива **Show** часто применяется, когда необходимо построить на одном графике кривую некоторой функции и представляющие ее узловые точки (например, при построении кривых регрессии «в облаке» точек исходных данных). Примеры такого построения даны на рис. 5.12 и 5.13.

8.3. Примитивы двумерной графики

Примитивами двумерной графики называют дополнительные указания, вводимые в функцию **Graphics** для построения некоторых заданных геометрических фигур. Функция **Graphics** задается в виде:

Graphics[primitives, options] – представляет двумерное графическое изображение.

Применение примитивов в составе функции **Graphics** избавляет пользователя от задания математических выражений, описывающих эти фигуры. Примитивы могут выполнять и иные действия. Они заметно увеличивают число типов графиков, которые способна строить система Mathematica.

Используются следующие примитивы двумерной графики:

- **Circle[{x, y}, r]** – строит окружность с радиусом r и центром в точке $\{x, y\}$.
- **Circle[{x, y}, {rx, ry}]** – строит эллипс с центром $\{x, y\}$ и полуосями rx и ry .
- **Circle[{x, y}, r, {theta1, theta2}]** – представляет дугу окружности радиуса r с центром $\{x, y\}$ и углами концевых точек $theta1$ и $theta2$.
- **Disk[{x, y}, r]** – является примитивом двумерной графики, представляющим закрашенный круг радиусом r с центром в точке $\{x, y\}$.
- **Disk[{x, y}, {rx, ry}]** – строит закрашенный овал с полуосями rx и ry и центром $\{x, y\}$.
- **Disk[{x, y}, r, {theta1, theta2}]** – строит сегмент круга радиуса r с центром $\{x, y\}$ и углами концевых точек $theta1$ и $theta2$.
- **Line[{pt1, pt2, ...}]** – строит линию, соединяющую последовательность точек.
- **Point[{x, y}]** – строит точку с координатами x и y .
- **Polygon[{x1, y1}, {x2, y2}, ...]** – построение полигона с закраской.
- **PostScript["string"]** – построение объекта, заданного на языке PostScript.
- **Rectangle[{xmin, ymin}, {xmax, ymax}]** – строит закрашенный прямоугольник, ориентированный параллельно осям, намеченный координатами точек противоположащих углов.
- **Rectangle[{xmin, ymin}, {xmax, ymax}, graphics]** – строит закрашенный прямоугольник, заполненный в соответствии с указаниями в функции **graphics** и заданным координатами противоположащих углов.
- **Raster[{{a11, a12, ...}, ...}]** – строит прямоугольный массив ячеек яркости.
- **RasterArray[{{g11, g12, ...}, ...}]** – строит прямоугольный массив ячеек, окрашенных в соответствии с графическими директивами g_{ij} .
- **Text[expr, coords]** – выводит текст, соответствующий печатной форме выражения $expr$, центрированный в точке с указанными координатами $coords$.

Рисунок 8.10 показывает применение функции **Graphics** для построения одновременно четырех графических объектов: отрезка прямой, заданного координатами его концевых точек, окружности с центром $(0,0)$ и радиусом 0.8 , текстовой надписи «Hello!» и жирной точки. Каждый объект задан своим примитивом.

На другом рисунке (рис. 8.11) представлено построение пятиугольника, заданного координатами его вершин.

Приведенные примеры поясняют технику применения графических примитивов. Но они, разумеется, не исчерпывают всех возможностей этого метода построения геометрических фигур и объектов. Все указанные примитивы используются как при построении двумерных, так и трехмерных графиков.

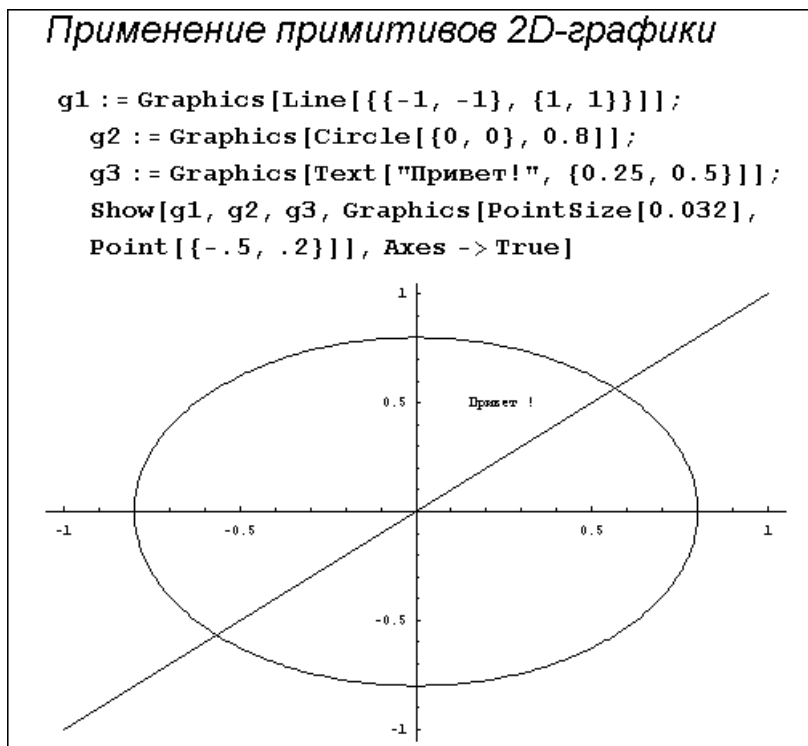


Рис. 8.10. Построение четырех графических объектов с помощью примитивов двумерной графики

8.4. Построение графиков в полярной системе координат

8.4.1. Задание функции в параметрической форме

Возможно построение графиков в полярной системе координат двумя способами. Первый способ основан на использовании обычной Декартовой системы координат. Координаты каждой точки при этом задаются в параметрическом виде:

$$x = f_x(t) \quad \text{и} \quad y = f_y(t),$$

где независимая переменная t меняется от минимального значения t_{\min} до максимального t_{\max} с шагом dt . Особенно удобно применение таких функций для построения замкнутых линий, таких как окружности, эллипсы, циклоиды и др. Например, окружность радиуса R может быть задана в следующей параметрической форме:

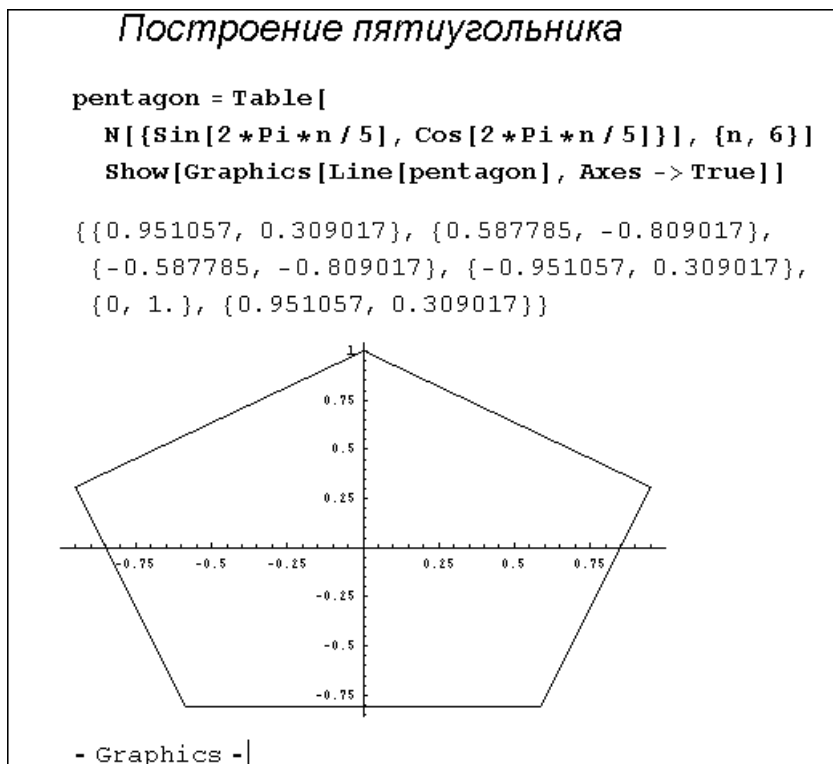


Рис. 8.11. Построение пятиугольника

$$x = R \cos(t) \quad \text{и} \quad y = R \sin(t),$$

если t меняется от 0 до 2π . В общем случае радиус также может быть функцией параметра t .

8.4.2. Функции для построения параметрически заданных графиков

Для построения параметрически заданных функций используются следующие графические средства:

- **ParametricPlot[{fx, fy}, {t, tmin, tmax}]** – строит параметрический график с координатами fx и fy (соответствующими x и y), получаемыми как функции от t .
- **ParametricPlot[{fx, fy}, {gx, gy}, ..., {t, tmin, tmax}]** – строит графики нескольких параметрических кривых.

Функции fx , fy и т.д. могут быть как непосредственно вписаны в список параметров, так и определены как функции пользователя.

8.4.3. Примеры построения графиков в полярной системе координат

Рисунок 8.12 показывает построение параметрически заданной фигуры Лиссажу. Она задается функциями синуса и косинуса с постоянным параметром R и аргументами, кратными t . Эти фигуры наблюдаются на экране электронного осциллографа, когда на его входы X и Y подаются синусоидальные сигналы с кратными частотами.

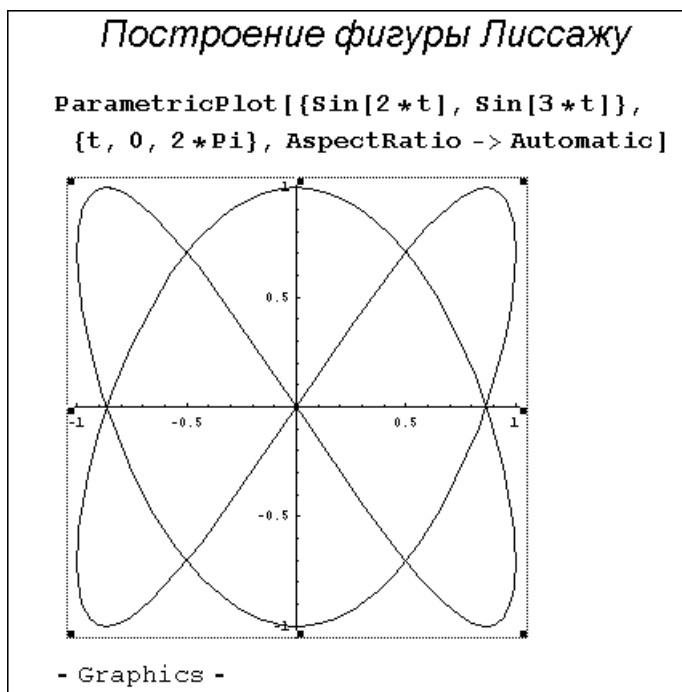


Рис. 8.12. Построение фигуры Лиссажу

На одном графике можно строить две и более фигуры с параметрически заданными уравнениями. На рис. 8.13 показан пример такого построения: строятся две фигуры Лиссажу, причем одна из них окружность. Больше двух фигур строить нерационально, так как на черно-белом графике их трудно различить.

Теперь рассмотрим второй способ построения графиков в полярной системе координат (рис. 8.14). Здесь каждая точка является концом радиус-вектора $R(t)$, причем угол t меняется от 0 до 2π . На рис. 8.14 функция $R(t)$ задана как функция пользователя $R[t_]$ с использованием образца для задания локальной переменной t в теле функции.

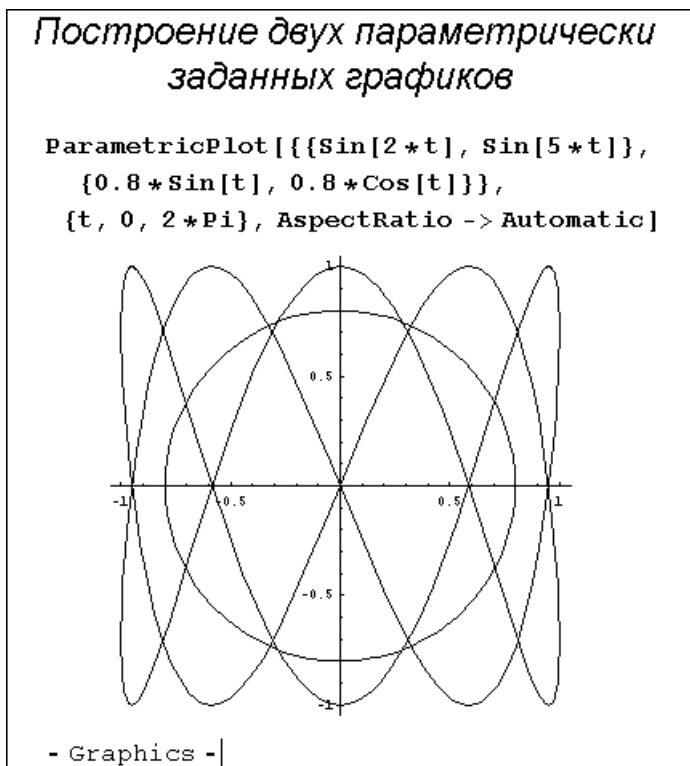


Рис. 8.13. Построение на одном графике двух фигур Лиссажу

Изменение параметра R позволяет заметно увеличить число отображаемых функций, фактически их бесконечно много. Помимо описанной фигуры на рис. 8.14 дополнительно построена линия окружности единичного радиуса. Чтобы она имела вид окружности, задана опция **AspectRatio->1**.

8.5. Построение контурных графиков

8.5.1. Функции для построения контурных графиков

Контурные графики, или графики линий равных высот, используются для отображения на плоскости трехмерных поверхностей. Они удобны для выявления всех экстремумов функций в пределах области графика. Такие графики являются линиями пересечения поверхности с секущими горизонтальными плоскостями, расположенными параллельно друг под другом. Они часто используются в картографии.

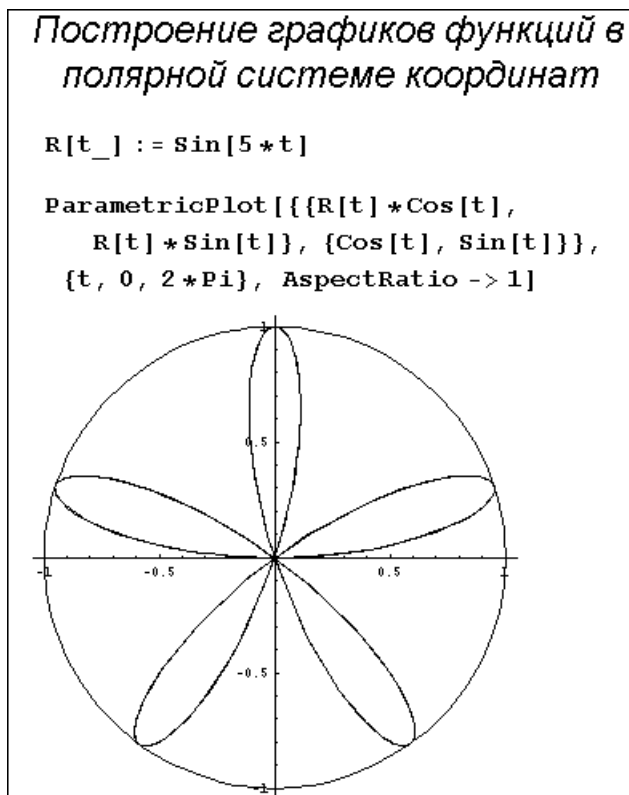


Рис. 8.14. Построение графика функции в полярной системе координат

Основными функциями и директивами для построения контурных графиков являются следующие:

- **ContourPlot[f,{x, xmin, xmax}, {y, ymin, ymax}]** – порождает контурный график f как функции от x и y .
- **ContourGraphics[array]** – представляет контурный график массива `array`.
- **ListContourPlot[array]** – формирует контурный график из массива величин высот.

Этих функций достаточно для построения практически любых монохромных графиков такого типа.

8.5.2. Опции для функций контурной графики

Для управления возможностями графической функции **ContourPlot** используются опции, список которых можно вывести командой:

```
Options[ContourGraphics]
```

Аналогично можно получить данные об опциях других функций этого раздела. Помимо уже рассмотренных ранее опций используются также следующие:

- **ColorFunction** – задает окраску областей между линиями.
- **Contours** – задает число контурных линий.
- **ContourLines** – задает прорисовку явных (explicit) контурных линий.
- **ContourShading** – задает затенение областей между контурными линиями.
- **ContourSmoothing** – задает сглаживание контурных линий.
- **ContourStyle** – задает стиль рисуемых линий для контурных графиков.
- **MeshRange** – задает области изменения X и Y координат.

Как видно из приведенного выше перечня опций, помимо указанных возможно применение и множества других опций.

8.5.3. Примеры построения контурных графиков

Рисунок 8.15 показывает построение контурного графика с окраской промежуточных областей между линиями. Окраска обеспечивается опцией `ColorFunction -> Hue`. Опция `ContourSmoothing -> True` задает сглаживание контурных линий.

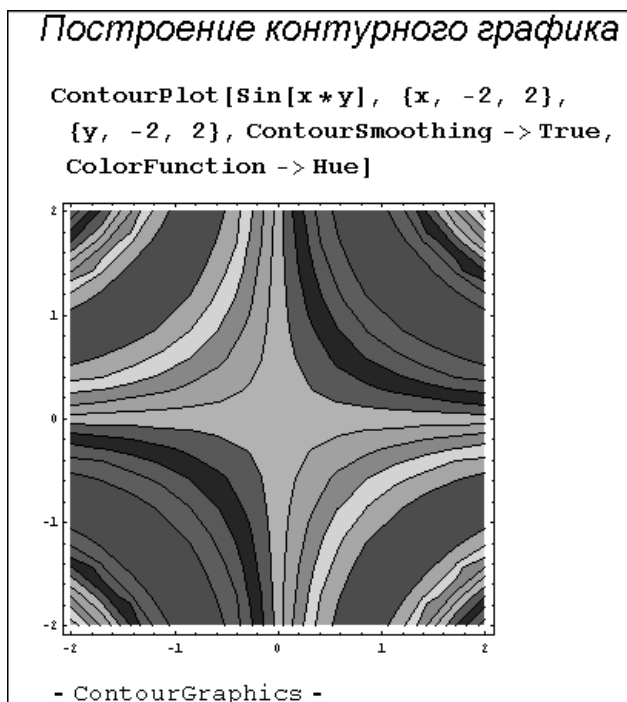


Рис. 8.15. Контурный график поверхности $\sin(x*y)$ с закраской областей между линиями равного уровня оттенками серого цвета

Следующий пример (рис. 8.16) иллюстрирует эффективность применения опции **ContourShading**. Если задать ее значение **False**, то заполнение пространства между линиями будет отсутствовать. Таким образом, в данном случае строятся только линии равного уровня.

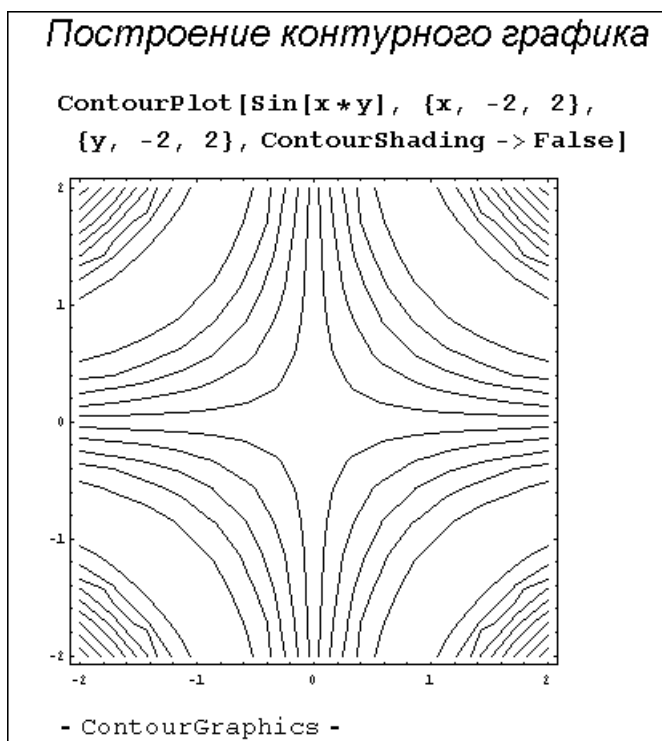


Рис. 8.16. Контурный график,
представленный только линиями равного уровня

Иногда график оказывается более наглядным, если убрать построение контурных линий, но оставить закраску областей между линиями. Такой вариант графика более предпочтителен, если нужно наблюдать качественную картину. Для построения такого графика необходимо использовать опцию **ContourLine** со значением **False** (рис. 8.17).

В данном случае используется вариант монохромной окраски областей между линиями (PostScript). Он может оказаться предпочтителен, например если график размещается в книге или предназначен для печати монохромным лазерным принтером.

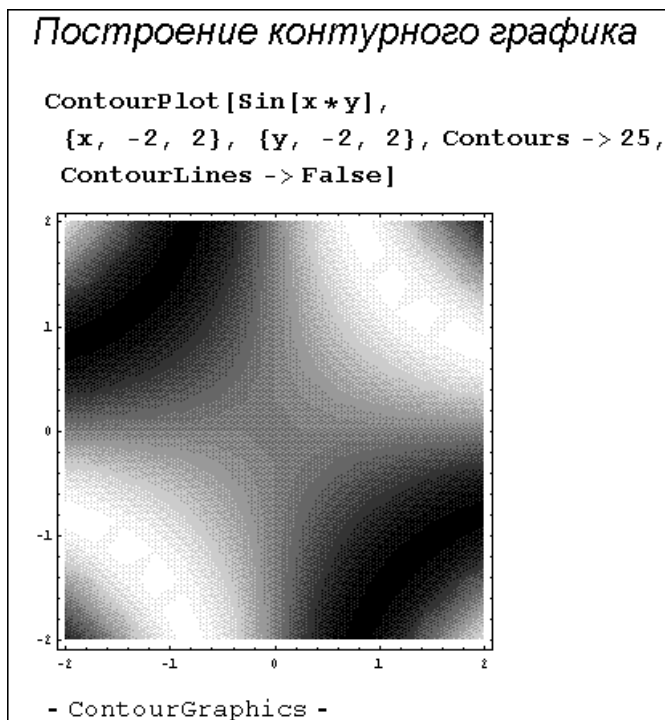


Рис. 8.17. Контурный график без линий равного уровня

8.6. Построение графиков плотности

8.6.1. Функции графиков плотности

Функцией двух переменных $f(x, y)$ может описываться плотность некоторой среды. Для построения графиков плотности используются следующие графические функции:

- **DensityGraphics[array]** – является представлением графика плотности.
- **DensityPlot[f, {x, xmin, xmax}, {y, ymin, ymax}]** – строит график плотности f , как функции от x и y .
- **ListDensityPlot[array]** – формирует график плотности из массива величин высот.

С этими функциями используется множество, в основном, уже рассмотренных опций. Их перечень можно получить с помощью функции **Options**.

8.6.2. Примеры построения графиков плотности

Внешне график плотности похож на контурный график. Однако для него характерно выделение элементарных участков (с равной плотностью) в форме квадратов (рис. 8.18).

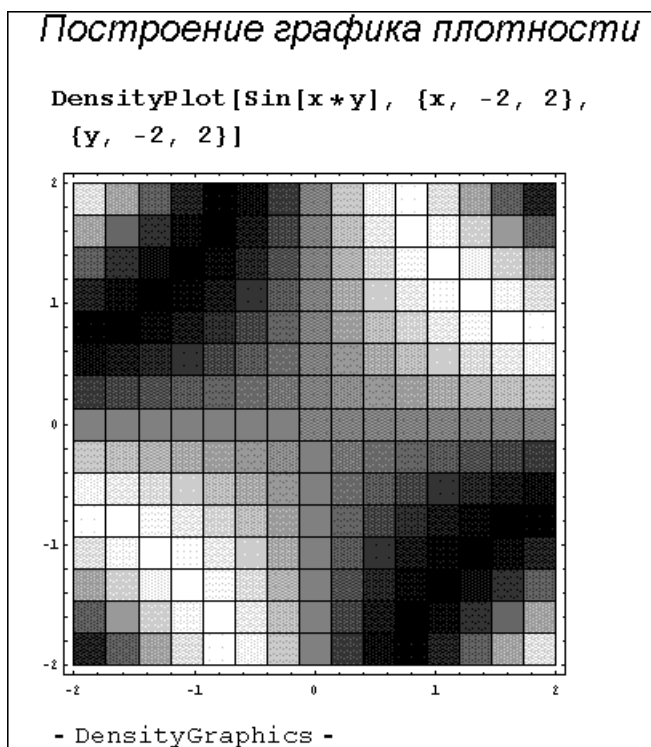


Рис. 8.18. График плотности

График плотности (рис. 8.18) также дан в режиме PostScript. Цветная функциональная раскраска таких графиков также возможна (см. опции, указанные выше для контурных графиков).

8.7. Построение графиков поверхностей

8.7.1. Принципы построения поверхностей и фигур

Функция двух переменных $z = f(x, y)$ в пространстве образует некоторую трехмерную фигуру или поверхность. Для их построения приходится использовать координатную систему с тремя осями координат: x , y и z . Поскольку экран дисплея ПК плоский, то на самом деле объемность фигур лишь имитируется: используется хорошо известный способ наглядного представления 3D фигур в виде аксонометрического графика.

Вместо построения всех точек фигуры обычно строится ее каркасная модель, содержащая линии разреза фигуры по взаимно перпендикулярным плоскостям. В результате фигура представляется в виде совокупности из множества криволинейных четырехугольников. Для придания фигуре большей естественности используются алгоритм удаления невидимых линий каркаса и функциональная закраска четырехугольников по правилу бокового освещения фигуры.

8.7.2. Основные функции для построения 3D графиков

Для построения графиков трехмерных поверхностей используется основная графическая функция **Plot3D**:

- **Plot3D**[f , { x , $xmin$, $xmax$ }, { y , $ymin$, $ymax$ }] – строит трехмерный график функции f переменных x и y .
- **Plot3D**[{ f , s }, { x , $xmin$, $xmax$ }, { y , $ymin$, $ymax$ }] – строит трехмерный график, в котором высоту поверхности определяет параметр f , а затенение – s .

На рис. 8.19 показан пример построения поверхности, описываемой функцией двух переменных $\cos(xy)$ при x и y меняющихся от -3 до 3 . Поверхность строится в виде каркаса с прямоугольными ячейками с использованием функциональной окраски. Все опции заданы по умолчанию.

Этот график будем считать исходным для демонстрации его модификации изменением опций.

8.7.3. Опции 3D графики

Для модификации трехмерных графиков могут использоваться следующие опции:

- **AmbientLight** – задает функциональную засветку от постоянного источника света с заданными координатами.
- **AxesEdge** – определяет, на каких гранях ограничительного параллелепипеда (ящика) должны выводиться оси.

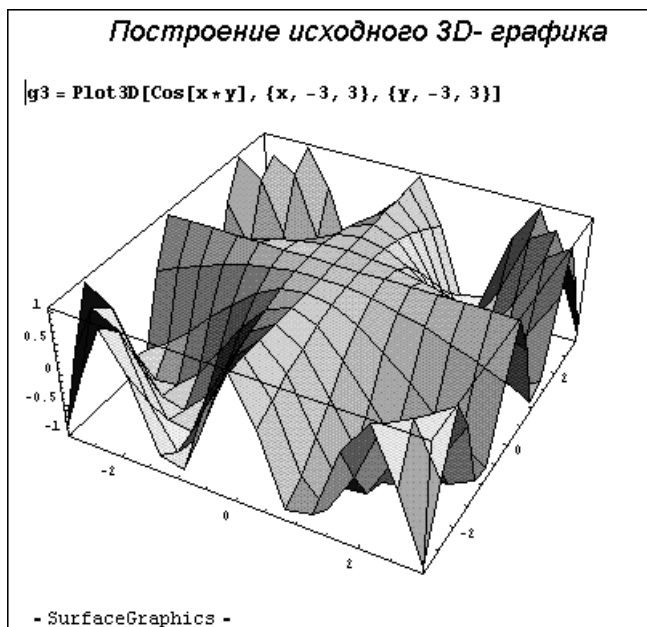


Рис. 8. 19. Пример построения поверхности $\cos(x+y)$ функцией `Plot3D` с опциями по умолчанию

- **Boxed** – указывает, следует ли рисовать контуры (ребра, грани) ограничительного параллелепипеда в трехмерном изображении.
- **BoxRatios** – задает значение отношений длин сторон для ограничительной рамки трехмерного изображения.
- **BoxStyle** – задает прорисовку ограничительной рамки.
- **Background** – задает цвет фона.
- **ClipFill** – определяет, как отсекаемые части поверхности должны выводиться.
- **ColorFunction** – определяет функцию, используемую для функциональной окраски.
- **ColorOutput** – задает тип цветового выхода для вывода.
- **DefaultFont** – возвращает шрифт по умолчанию для текста в графике.
- **DefaultColor** – задает цвет по умолчанию для линий, точек и т.д.
- **\$DisplayFunction** – задает установочное значение по умолчанию для опции `DisplayFunction` в графических функциях.
- **DisplayFunction** – определяет функцию, которая применяется к графическим и звуковым примитивам для их отображения.
- **Epilog** – опция для графических функций, дающая список графических примитивов, которые должны воспроизводиться после воспроизведения главной части графики.

- **FaceGrids** – опция для функций трехмерной графики; устанавливает вывод линий сетки на гранях (лицевых сторонах) ограничительной рамки.
- **HiddenSurface** – определяет, необходимо или нет удалять невидимые линии каркаса.
- **Lighting** – указывает, следует ли использовать моделируемую освещенность (simulated illumination) в трехмерных изображениях.
- **LightSources** – опция к Graphics3D и родственным функциям, которая устанавливает возможности (свойства) точечных источников света для моделируемого освещения.
- **Mesh** – указывает, следует ли вырисовывать явно заданную x-y сетку.
- **MeshRange** – устанавливает диапазон (область изменения) x и y координат, которые соответствуют массиву заданных величин z.
- **MeshStyle** – задает стиль вывода линий сетки.
- **SphericalRegion** – указывает, следует ли конечный образ масштабировать так, чтобы сфера рисовалась вокруг трехмерного отображения.
- **Plot3Matrix** – опция к Graphics3D и родственным функциям, которая может использоваться для определения матрицы преобразования явной гомогенной (однородной) перспективы.
- **PolygonIntersections** – опция для Graphics3D, которая определяет, следует ли пересекающиеся многоугольники оставлять без изменения.
- **Prolog** – опция для графических функций, дающая список графических примитивов, которые предоставляются, до главной части графики.
- **RenderAll** – опция к Graphics3D, которая указывает, должен или нет генерироваться PostScript для всех многоугольников.
- **Shading** – опция для SurfaceGraphics, указывающая, следует ли выполнять затенение поверхностей.
- **ToColor[color, form]** – превращает color в form; если form представляет собой функцию GrayLevel, RGBColor или CMYKColor, то color превращается в нее. В противном случае вычисляется **form[color]**, и как результат ожидается правильная цветовая директива.
- **ViewCenter** – задает масштабные координаты точки, оказывающейся в центре области отображения в окончательном (последнем, целевом) графике.
- **ViewPoint** – меняет точку пространства, из которой рассматривается объект.
- **ViewVertical** – устанавливает, какое направление в относительных координатах должно быть вертикальным в окончательном образе.

8.7.4. Директивы трехмерной графики

Помимо опций для трехмерной графики используется ряд графических директив и функций:

- **CMYKColor[cyan, magenta, yellow, black]** – устанавливает составляющие цвета.
- **EdgeForm[g]** – указывает, что грани многоугольников должны быть нарисованы с применением графической директивы или списка директив.

- **FaceForm[*gf*, *gb*]** – указывает, что передние грани (лицевые поверхности) многоугольников должны выводиться с применением графического примитива *gf*, а задние грани (невидимые поверхности) – посредством *gb*.
- **FullAxes[*graphics*]** – возвращает опции осей графического объекта.
- **FullGraphics[*g*]** – берет графический объект и производит новый, в котором объекты, определяемые графическими опциями, даются как явные (точные) списки графических примитивов.
- **FullOptions[*expr*]** – возвращает полные установки опций, которые явно определены в выражении типа графического объекта.
- **FullOptions[*expr*, *name*]** – возвращает полное установочное значение для указанного именем графического объекта..
- **Hue[*h*]** – указывает, что графические объекты, которые последуют, должны будут отображаться по возможности в цвете *h*.
- **Hue[*h*, *s*, *b*]** – определяет цвета в значениях оттенка *h*, насыщенности *s* и яркости *b*.
- **LineForm[*g*]** – устанавливает, что вывод линий следует выполнять с применением графической директивы *g* или списка графических директив *g*.
- **PointForm[*g*]** – указывает форму точек объекта *g*.
- **PointSize[*r*]** – указывает, что точки при последующем выводе должны изображаться по возможности в виде кругов с радиусом *r* (дробь от общей ширины графика).
- **RGBColor[*red*, *green*, *blue*]** – указывает, что последующие графические объекты должны отображаться заданной совокупностью цветов. Значения *red* (красный), *green* (зеленый) и *blue* (синий) указываются в относительных единицах – от 0 до 1.
- **SurfaceColor[*dcol*]** – устанавливает, что последующие многоугольники должны действовать как рассеивающие (диффузные) отражатели света с заданным цветом *dcol*.
- **SurfaceColor[*dcol*, *scol*]** – указывает, что должен содержаться компонент зеркального отражения с цветом, заданным *scol*.
- **SurfaceColor[*dcol*, *scol*, *n*]** – указывает, что отражение должно происходить с показателем зеркального отражения *n*.

Применение указанных функций и опций позволяет строить большое число графиков различных типов даже при задании одной и той же поверхности. В качестве примера рассмотрим отдельные кадры документа, демонстрирующего влияние опций на вид 3D математической поверхности.

8.7.5. Примеры модификации 3D графиков с помощью опций

На рис. 8.20 показана исходная поверхность (рис. 8.19), построенная с применением опции **PlotPoint->50**. Это означает, что поверхность по каждой оси делится на 50 частей (в исходном графике по умолчанию используется деление в 10 раз).

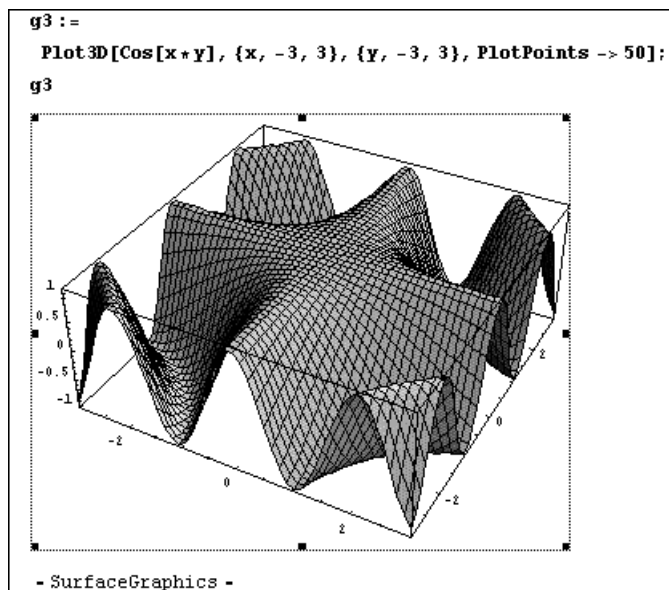


Рис. 8.20. Исходная математическая поверхность

Масштаб по вертикали задается автоматически, с тем, чтобы все высоты поверхности не ограничивались.

На рис. 8.21 та же поверхность показана с применением при построении опции `PlotRange`, срезающей верхнюю часть поверхности. График поверхности при этом существенно меняется (сравните с рис. 8.20).

Опция **Boxed->False** удаляет ограничивающие рамки, образующие ящик, в который вписывается строящаяся трехмерная поверхность (рис. 8.22). Осталось лишь построение осей.

Опция **ViewPoint** – позволяет включить при построении отображение перспективы и изменять углы, под которыми рассматривается фигура. Рис. 8.23 иллюстрирует применение этой опции.

Опция **Mesh->False** позволяет удалить линии каркаса фигуры. Нередко это придает фигуре более естественный вид (рис. 8.24), обычно мы наблюдаем такие фигуры без линий каркаса.

В ряде случаев, напротив, именно линии каркаса несут важную информацию. Система строит каркас 3D поверхности для двух типов построений: с использованием алгоритма удаления невидимых линий и без использования этого алгоритма.

Рисунок 8.25 показывает построения при использовании алгоритма удаления невидимых линий. Нетрудно заметить, что в этом случае поверхность выглядит достаточно эстетично даже без применения функциональной закрашки.

А на рис. 8.26 показано построение каркаса без удаления невидимых линий. Такой вид математическая поверхность имеет, если представить ее построенной

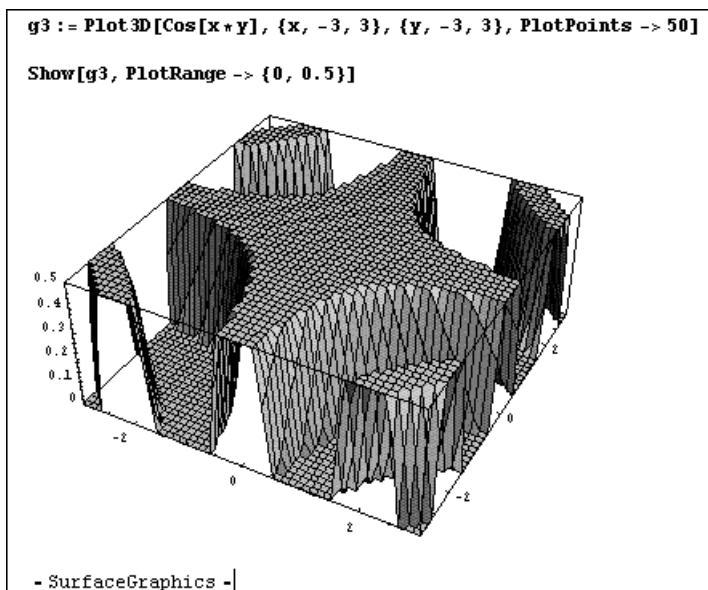


Рис. 8.21. Математическая поверхность
с отсеченной верхней частью

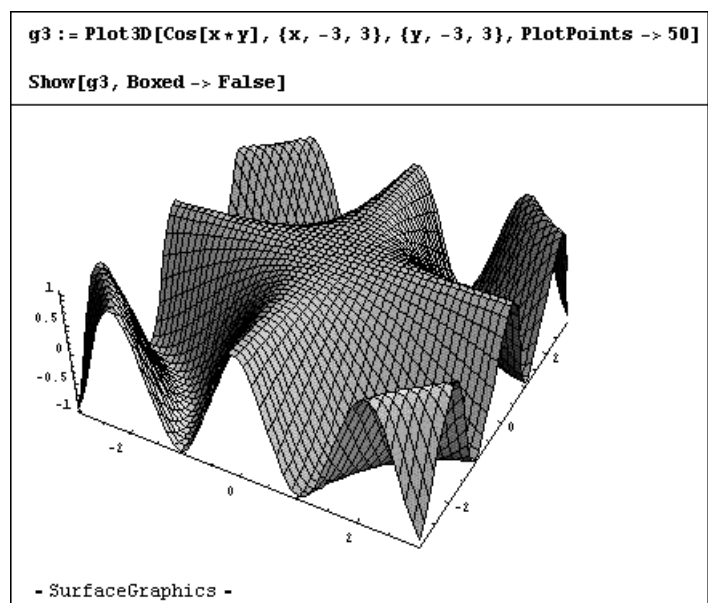


Рис. 8.22. Построение 3D поверхности
без ограничительного ящика

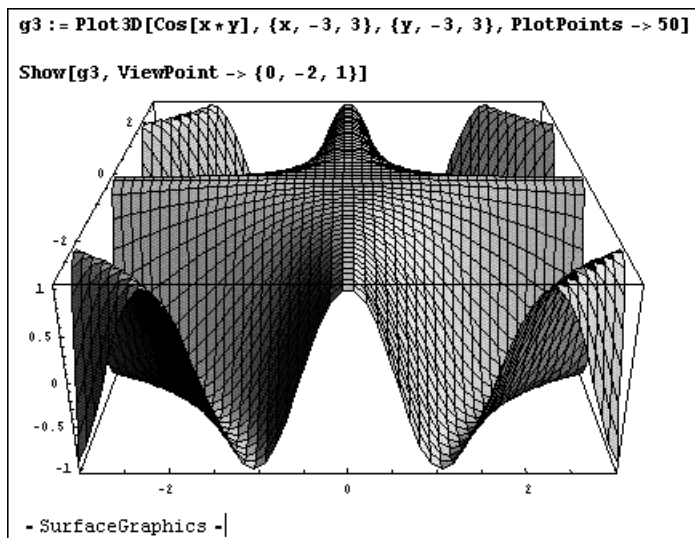


Рис. 8.23. Математическая поверхность,
построенная в перспективе

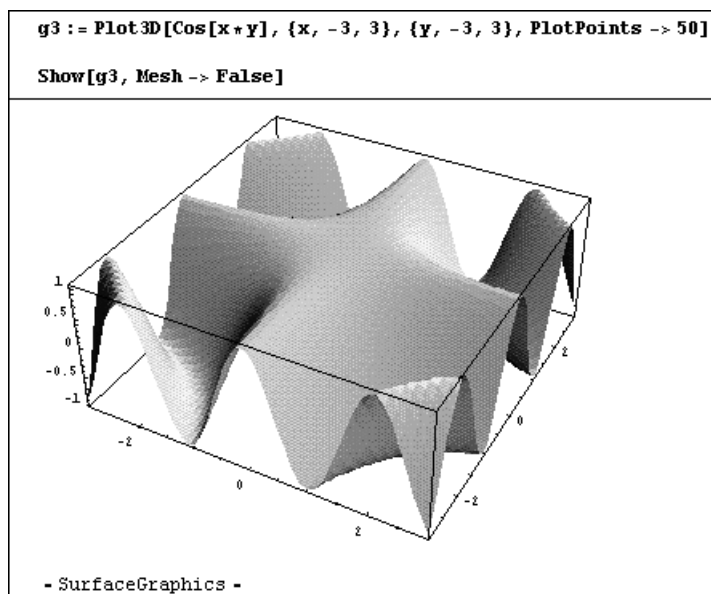


Рис. 8.24. Математическая поверхность
с удаленными линиями каркаса

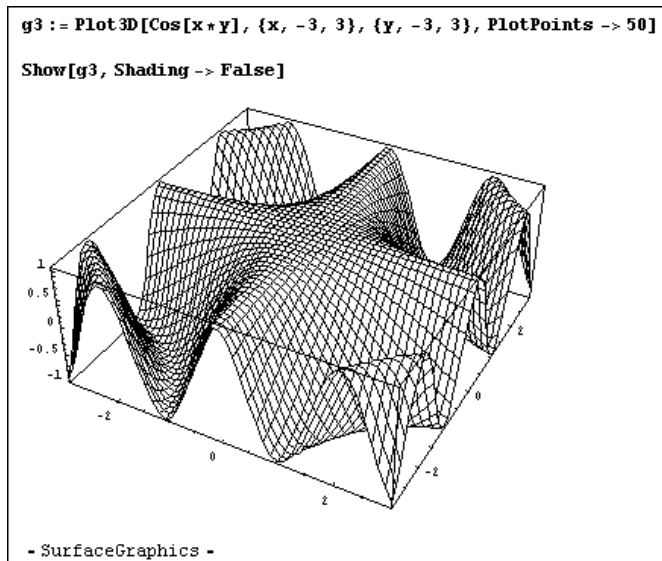


Рис. 8.25. Построение каркаса математической поверхности с использованием алгоритма удаления невидимых линий

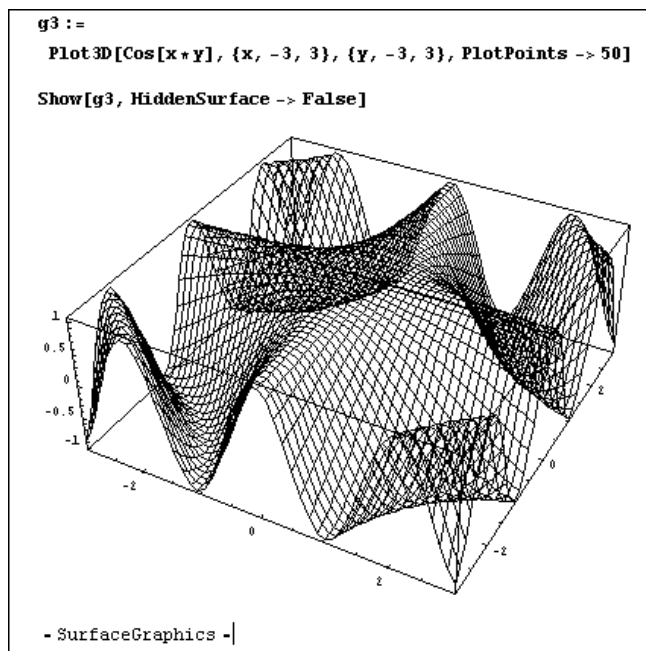


Рис. 8.26. Построение каркаса математической поверхности без использования алгоритма удаления невидимых линий

из тонких проволочек, висящих в пространстве. Это дает дополнительную информацию о пространственной фигуре, но эстетически она выглядит хуже, чем фигура, построенная с применением алгоритма удаления невидимых линий каркаса.

Таким образом, как и ранее, применение опций позволяет легко варьировать характером и типом графиков, придавая им вид, удобный для заданного применения. На рис. 8.27 показан пример построения 3D графика с применением одновременно нескольких опций.

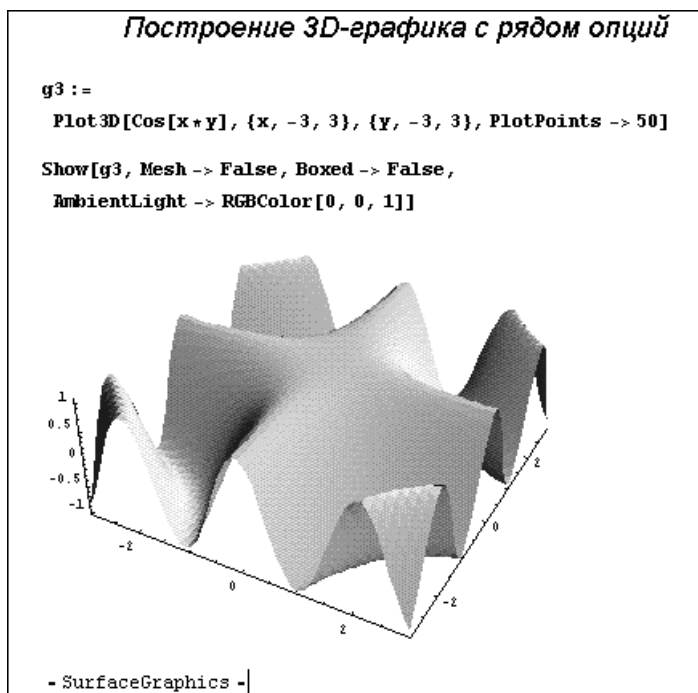


Рис. 8.27. Пример построения 3D графика с несколькими опциями

Таким образом, приведенные примеры самым наглядным образом показывают, насколько легко модифицируются графики с применением тех или иных опций. Разумеется, есть множество возможностей для иных модификаций, которые пользователь может опробовать самостоятельно.

8.7.6. Графическая функция *ListPlot3D*

Часто 3D поверхность задается массивом своих высот (аппликат). Для построения графика в этом случае используется графическая функция **ListPlot3D**:

- **ListPlot3D[array]** – строит трехмерный график поверхности, представленной массивом значений высот;

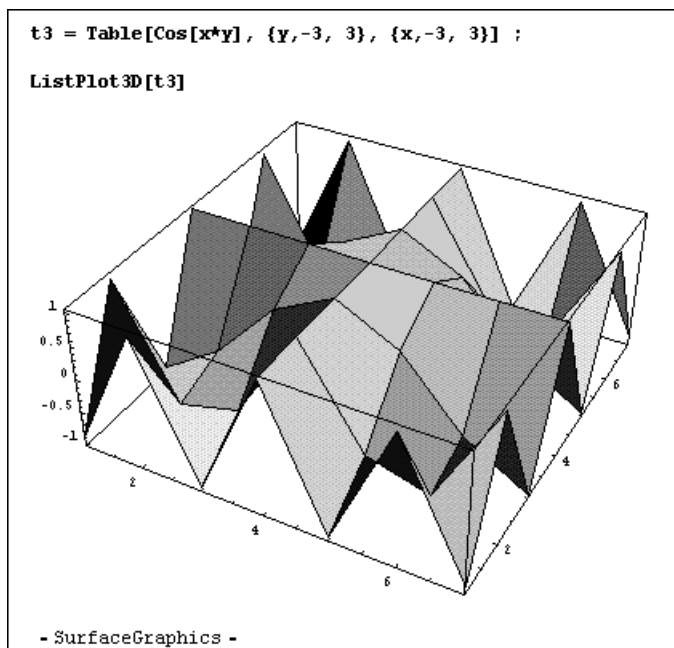


Рис. 8.28. Пример применения функции **ListPlot3D**

- **ListPlot3D[array, shades]** – строит график так, что каждый элемент поверхности штрихуется (затеняется) согласно спецификации в shades;
- **PlotJoined** – дополнительная опция для **ListPlot**, указывающая, следует ли точки, нанесенные на график, соединять линией (рис. 8.28).

Командой **Options[ListPlot3D]** можно вывести полный список опций данной функции и использовать их для модификации графиков, которые строит эта функция.

8.7.7. Параметрическая 3D графика

Особый шик построениям 3D фигур и поверхностей придает функция **ParametricPlot3D**, в которой предусмотрено параметрическое задание всех трех функций, описывающих координаты каждой точки. Данная функция используется в следующих видах:

- **ParametricPlot3D[{fx, fy, fz}, {t, tmin, tmax}, {u, umin, umax}]** – строит трехмерную поверхность, параметризованную по t и u.
- **ParametricPlot3D[{fx, fy, fz}, {t, tmin, tmax}]** – выполняет трехмерную пространственную кривую, параметризованную переменной t, которая изменяется от tmin до tmax.
- **ParametricPlot3D[{fx, fy, fz, s},...]** – выполняет затенение графика в соответствии с цветовой спецификацией s.

- **ParametricPlot3D**[{{fx, fy, fz}, {gx, gy, gz}, ...}, ...] – строит несколько объектов вместе.

Эта функция имеет множество опций, список которых выводит команда

Options[ParametricPlot3D]

Большая часть из них уже рассматривалась ранее. При этом даже при использовании только опций, заданных по умолчанию, можно получить любопытные построения. Так, на рис. 8.29 показан простой пример применения функции **ParametricPlot3D** для построения замкнутой линии, расположенной в пространстве. Это, так сказать, объемный вариант фигур Лиссажу, построение которых было описано выше.

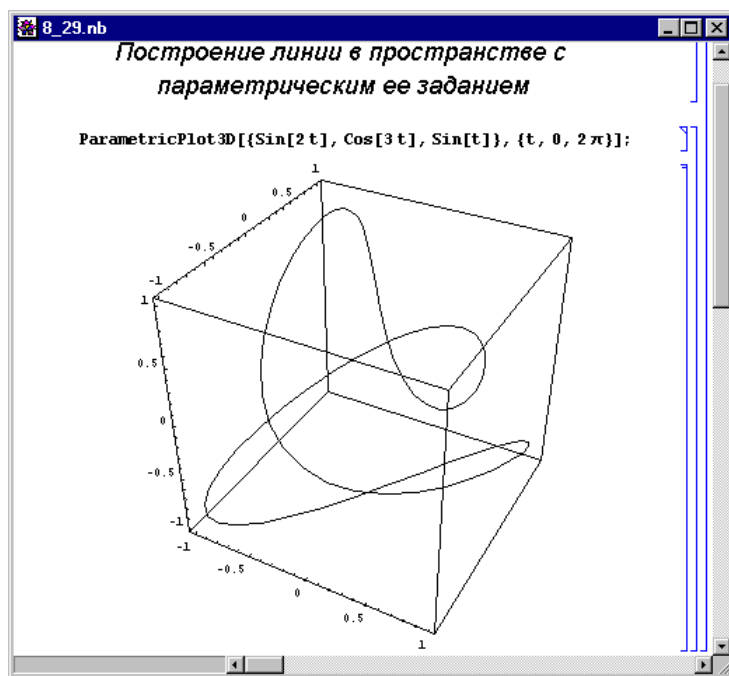


Рис. 8.29. Построение кривой в пространстве, заданной в параметрической форме

Параметрическое задание функций позволяет легко строить сложные пространственные фигуры, визуально весьма напоминающие реальные объекты. Покажем это на трех примерах.

Первым примером может служить фигура «рог изобилия», показанная на рис. 8.30. По существу это раскручивающаяся объемная спираль, диаметр которой постепенно нарастает.

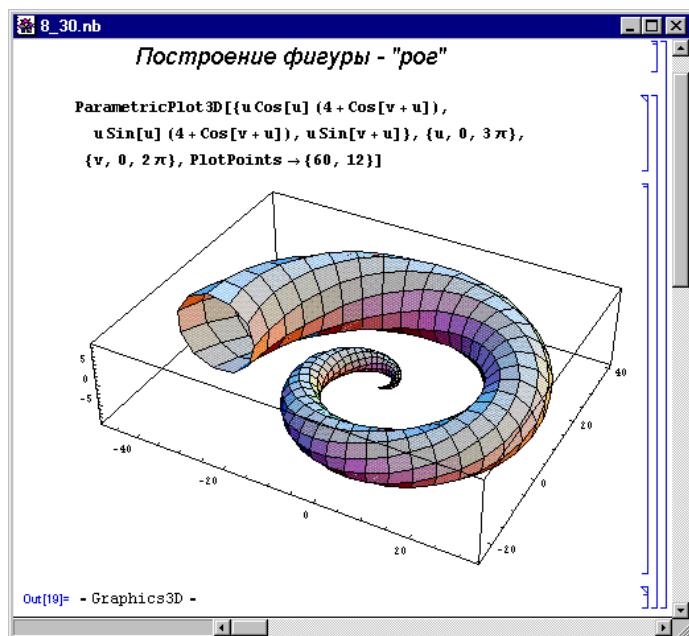


Рис. 8.30. Построение фигуры «рог»

Другой пример – объемное кольцо с сечением, напоминающим знак бесконечности – ∞ . Его построение дано на рис. 8.31. Обратите внимание на интересный эффект: из кольца удален сектор, что сразу выделяет его внутреннее строение. Все, что потребовалось для создания этого эффекта, так это задать верхний предел изменения переменной t , как $2\pi - 0.6$. Если сделать этот предел равным 2π , то кольцо станет непрерывным.

Третий пример такого рода – построение объемной сферы. Этот пример показан на рис. 8.32. Здесь также использован прием изменения значений переменной t для получения выреза сегмента сферы. Опять-таки, задав изменение t от 0 до 2π , можно получить построение всей сферы без выреза.

8.7.8. Построение фигур, пересекающихся в пространстве

Пожалуй, наиболее впечатляющими являются построения 3D фигур, пересекающихся в пространстве. Для этого достаточно каждую фигуру представить в виде графического объекта, а затем с помощью директивы **Show** вывести их на одном графике. При этом Mathematica автоматически рассчитывает линии пересечения фигур и строит график так, чтобы заслоненные ячейки фигур не были видны.

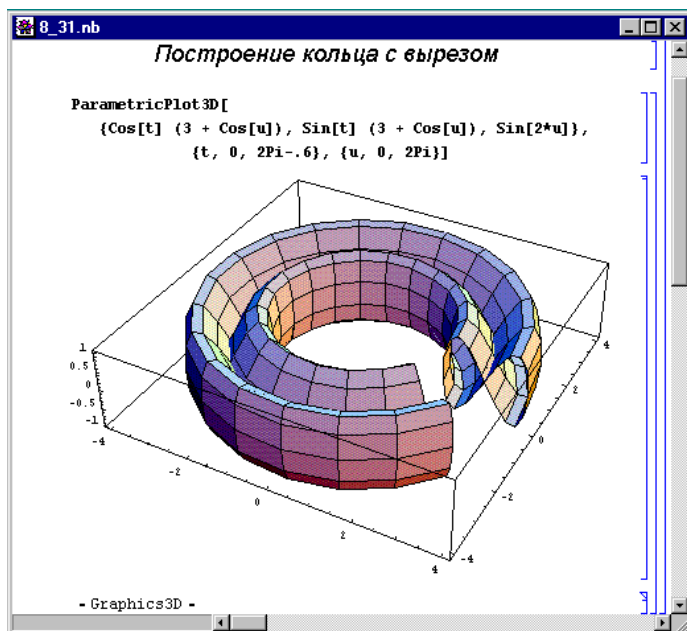


Рис. 8.31. Построение кольца с удаленным сегментом

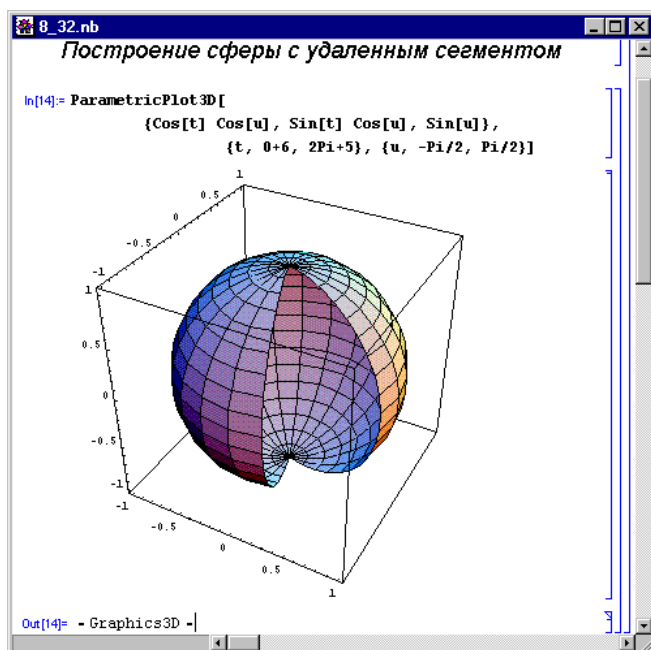


Рис. 8.32. Построение сферы с удаленным сегментом

Проиллюстрируем это с помощью рис. 8.33. На нем показано задание и построение одного графического объекта *g1* – объемной спирали, полученной сворачиванием ленты.

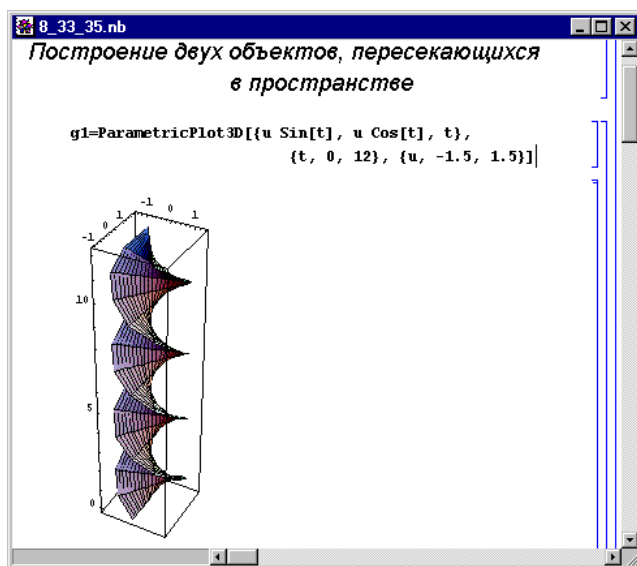


Рис. 8.33. Построение объекта *g1* – объемной спирали

Второй объект, построение которого представлено на рис. 8.34, – это объемное кольцо. Его построение было описано выше. В конце части документа, показанного на рис. 8.34, задана функция *Show* вывода объектов на одном графике.

Рисунок 8.35 демонстрирует комбинированный график, построенный функцией **Show**. Он показывает кольцо, через отверстие которого проходит объемная спираль. Вырез в кольце показывает, как проходит спираль внутри кольца.

Графики такого типа дают высокую степень визуализации трехмерных поверхностей и фигур.

8.8. Прimitives трехмерной графики и их применение

8.8.1. Функция *Graphics3D* и ее опции и примитивы

Наряду с построением графиков поверхностей, заданных аналитическими выражениями, имеется возможность создания графиков из различных элементарных геометрических объектов, называемых примитивами. Они включаются в список параметров функции **Graphics3D**:

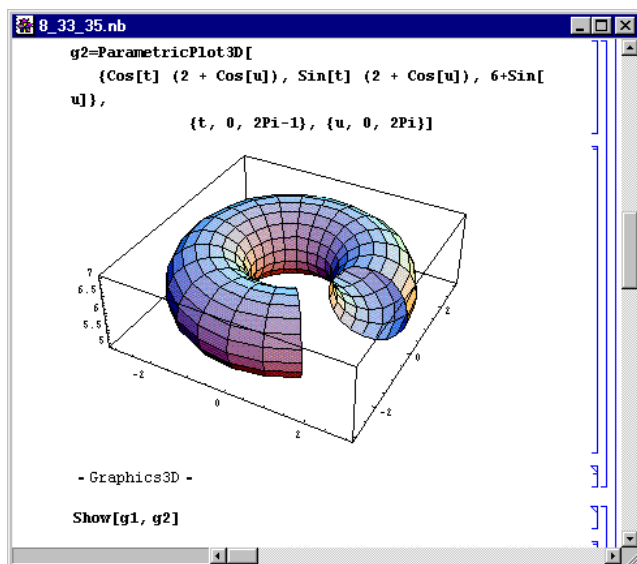


Рис. 8.34. Построение объекта g_2 – объемного кольца с удаленным сегментом

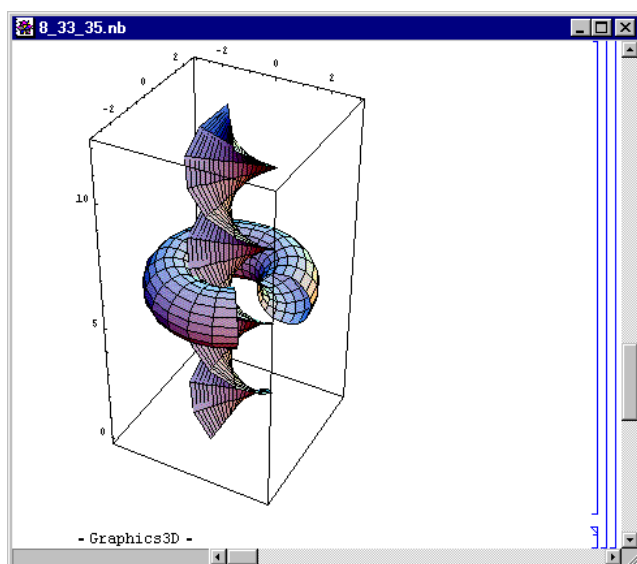


Рис. 8.35. Построение комбинированного объекта – спирали внутри кольца

Graphics3D[primitives, options] – представляет трехмерное графическое изображение.

С ней, помимо примитивов 2D графики, могут использоваться следующие графические примитивы:

- **Cuboid[{xmin, ymin, zmin}]** – представляет единичный куб, ориентированный параллельно осям.
- **CellArray[{{a11, a12, ...}, ...}]** – представляет прямоугольный массив элементов яркости.
- **Cuboid[{xmin, ymin, zmin}, {xmax, ymax, zmax}]** – представляет прямоугольный параллелепипед, заданный координатами противоположных вершин.
- **PostScript["string1", "string2", ...]** – графический примитив, задающий построение графика по кодам языка PostScript.
- **SurfaceGraphics[array]** – представляет трехмерный график поверхности, для которого значения высоты каждой точки на сетке заданы элементами массива.
- **SurfaceGraphics[array, shades]** – представляет поверхность, части которой затеняются согласно массиву shades.

8.8.2. Примеры применения функции Graphics3D с примитивами

Функция **Graphics3D** со своими примитивами может использоваться для построения в пространстве различных объектов, например точек, кубиков или многоугольников.

Рисунок 8.36 показывает два варианта случайных точек в пространстве. Для генерации координат точек используется функция **Random[]**, возвращающая случайные числа, распределенные по равномерному закону распределения.

Поскольку ограничительный ящик не удален, создается впечатление о построении точек внутри куба.

На рис. 8.37 показано построение в пространстве ряда небольших кубиков. Для этого используется примитив **Cuboid**, повторенный 7 раз. Для воспроизведения набора кубиков, перечисленных в функции **Graphics3D**, применяется функция-директива **Show**.

Нетрудно заметить, что и здесь неплохо работают встроенные алгоритмы удаления невидимых частей объектов. Это дает довольно реалистическое изображение их в пространстве.

Еще более наглядное представление об этом алгоритме дает рис. 8.38. На нем показано построение в пространстве ряда плоских многоугольников, частично проникающих друг в друга. Нетрудно заметить, что и здесь алгоритм удаления невидимых поверхностей работает превосходно.

Здесь каждый из многоугольников формируется с помощью функции пользователя **randpoly[n_]**, в теле которой используется примитив **Polygon**. Эта

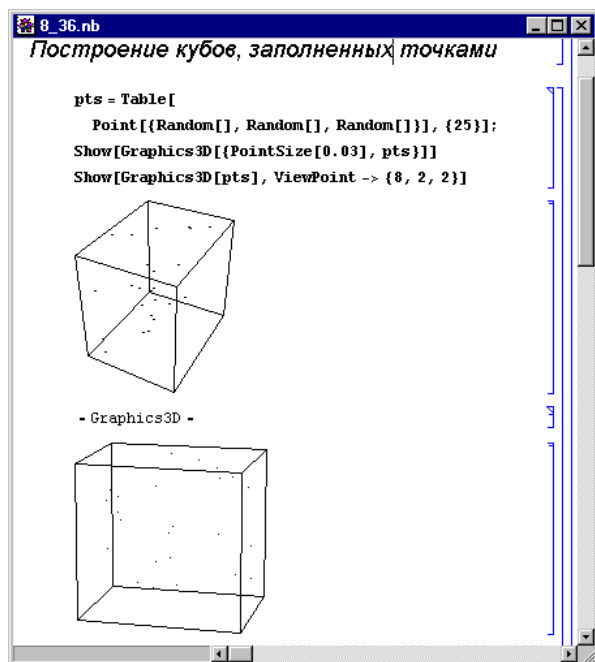


Рис. 8.36. Построение случайных точек в пространстве

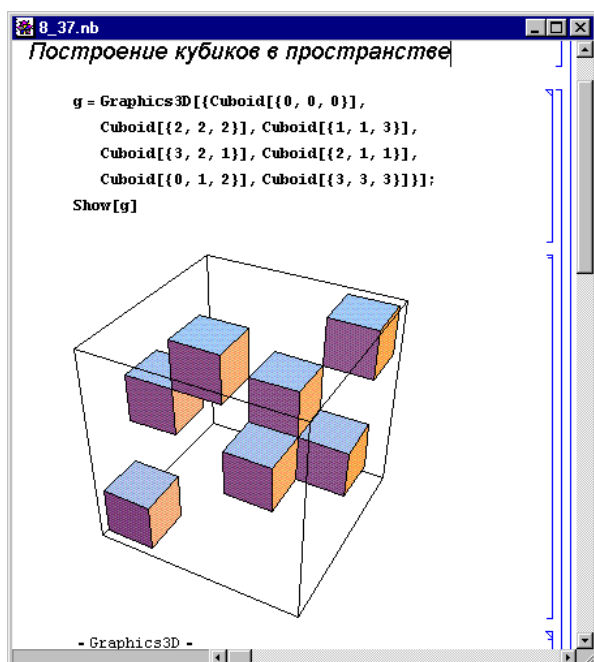


Рис. 8.37. Построение в пространстве ряда кубиков



Рис. 8.38. Построение в пространстве
взаимно пересекающихся плоских многоугольников

функция формирует случайные многоугольники, выводимые затем функцией-директивой **Show**.

8.9. Дополнительные средства графики Mathematica 5.1/5.2

8.9.1. Импорт графических изображений

Несмотря на обширные возможности встроенных в ядро системы Mathematica графических функций, примитивов и опций, они не способны охватить все многообразие графических приложений в математике. Поэтому предусмотрен *импорт рисунков*, созданных в различных графических системах или в документах самой системы Mathematica.

В Mathematica 5.2 импорт файлов графических форматов осуществляет функция:

Import["file.ext"] или **Import["file", "format"]**

Функция возвращает объект типа Graphics. Для его просмотра можно использовать функцию **Show**. Пример загрузки файла формата .rsx и его просмотр показаны на рис. 8.39. Если импортируемый файл находится не в основной директо-

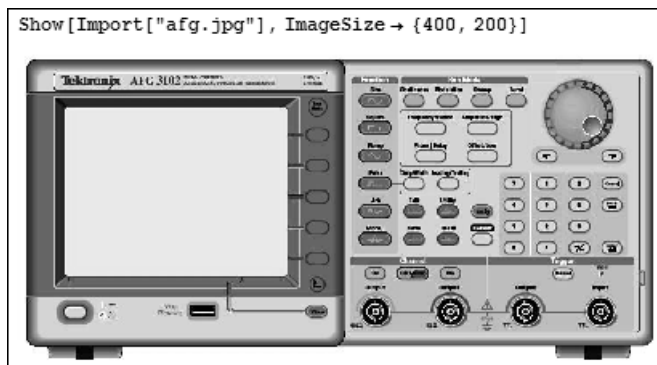


Рис. 8.39. Пример импорта графического файла и его просмотра

рии Mathematica, то его имя необходимо указывать с учетом директории, в которой находится файл.

С функцией `Import` применяются опции: `ImageSize` для задания размера изображения (пример ее применения см. на рис. 8.39), `ImageResolution` (установка разрешения) и `ImageRotated` для поворота изображения. Имеется также функция `Display`, импортирующая потоки из каналов и конвертирующая их в формат PostScript.

8.9.2. Экспорт графических изображений

Для экспорта графических файлов используется функция:

Export["file.ext", expr] или **Export["file", expr, "format"]**

На рис. 8.40 сверху показан пример экспорта графика функции $\sin(x)/x$. В результате создается файл `gf.jpg`. Пример снизу показывает его считывание и просмотр с помощью функции `Show`. Созданное и считанное из файла изображения идентичны.

Возможен также импорт изображения с использованием *буфера* промежуточного хранения. Нужно изображение в каком-либо приложении (например, в графическом редакторе) необходимо выделить и поместить в буфер командой **Copy** или **Cut**. Затем следует перейти к работе с системой Mathematica. Установив маркер мыши на место ввода, достаточно выполнить команду **Paste** в позиции **Edit** главного меню системы Mathematica. Если при импорте изображения необходимо очистить буфер, то следует использовать команду **Paste and Discard**.

Импортированное изображение размещается в ячейке вывода и с ним возможны все манипуляции, характерные для рисунков в ячейках вывода. Так, их можно растягивать или сжимать, а также перемещать в пределах ячейки. Этот способ импорта изображений полезен для создания электронных книг, уроков и статей средствами системы Mathematica.

Хотя функции импорта/экспорта поддерживают большое число типов файлов, их работа с реальными изображениями была проверена только при формате файлов с расширением `.jpg`.

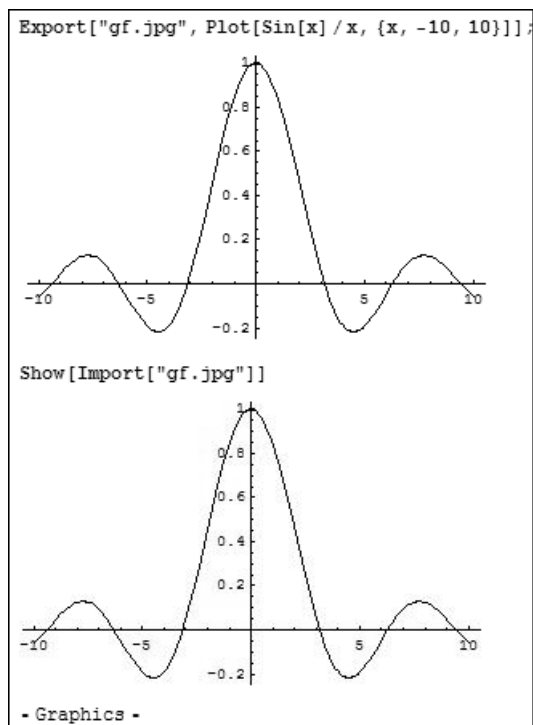


Рис. 8.40. Пример экспорта файла графика функции и его импорта и просмотра

8.9.3. Вставка графических и иных объектов

Более широкие возможности предоставляет вставка объектов (Insert Object). Как уже отмечалось в главе 2, она реализуется командой **Insert Object...** в позиции Edit главного меню. Эта команда открывает окно со списком возможных приложений, которые могут выступать в роли экспортера объектов для системы Mathematica.

Если, к примеру, выбрать в качестве объекта рисунок графического редактора Paint, то на экране появится окно редактора (рис. 8.40). Теперь в этом редакторе можно создавать любые изображения, например, вроде рожицы, квадрата и эллипса, представленные в окне редактора на рис. 8.41.

Если теперь закрыть окно редактора, то созданный рисунок появится в строке вывода документа системы Mathematica (рис. 8.42). Его можно выделять, растягивать в разных направлениях, перемещать и т.д.

Вставка объекта отличается от импорта рисунков (или текстов) одним принципиально важным обстоятельством – объекты могут редактироваться с автоматическим вызовом для этого приложения, являющегося экспортером объектов.

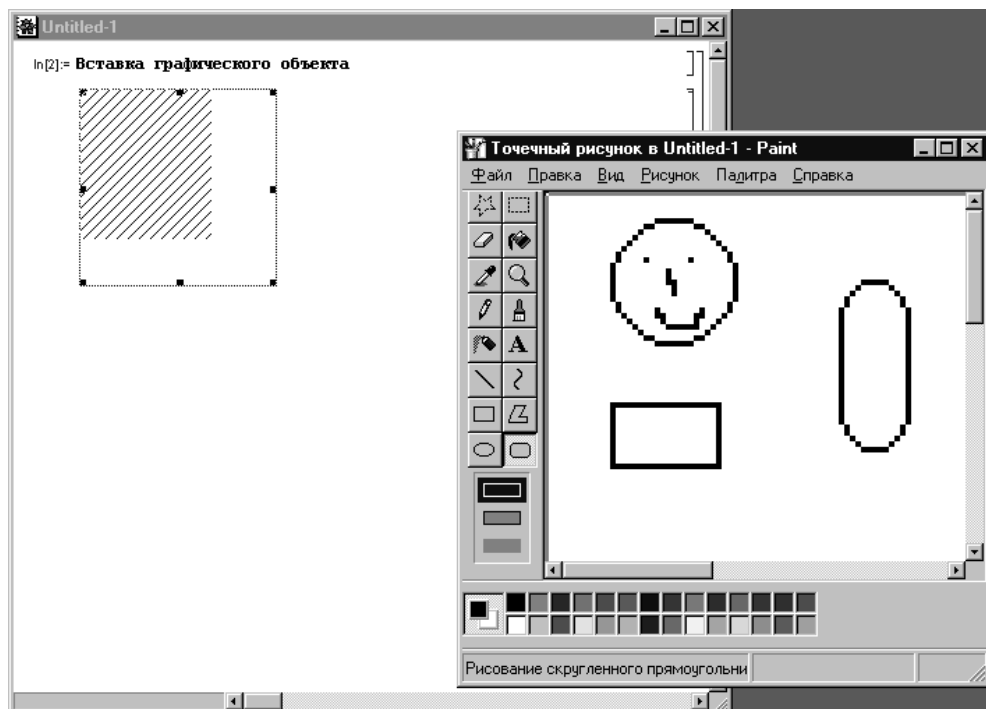


Рис. 8.41. Подготовка объекта в среде графического редактора Paint, вызванного из документа системы Mathematica

Для редактирования объекта, например, нашего рисунка, достаточно навести на него курсор мыши и дважды быстро щелкнуть левой клавишей мыши. Произойдет загрузка графического редактора, и мы увидим картину, подобную приведенной на рис. 8.41. На экране появится окно редактора с рисунком, и его можно произвольно изменять. После закрытия окна редактора новый рисунок появится в месте вставки.

Разумеется, объектами вставки могут быть не только рисунки, но и тексты, и документы других систем. Интересно оценить, насколько Mathematica восприимчива к другим математическим системам. Увы, эта «высокопоставленная мадам» очень критична к своим возможным партнерам или соперницам. Так, она не воспринимает системы Maple и MATLAB, которые способны соперничать с ней по своим возможностям и скорости работы. Не понимает система и такую «мелочь», как системы начального уровня Derive и MuPAD.

Зато благосклонно относится к системе Mathcad, известной своим бесподобным интерфейсом и, главное, возможностями задания в документах сложных формул в их вполне естественном виде. Рисунок 8.43 показывает подготовку в Mathcad графика трех функций и вычисления определенного интеграла.

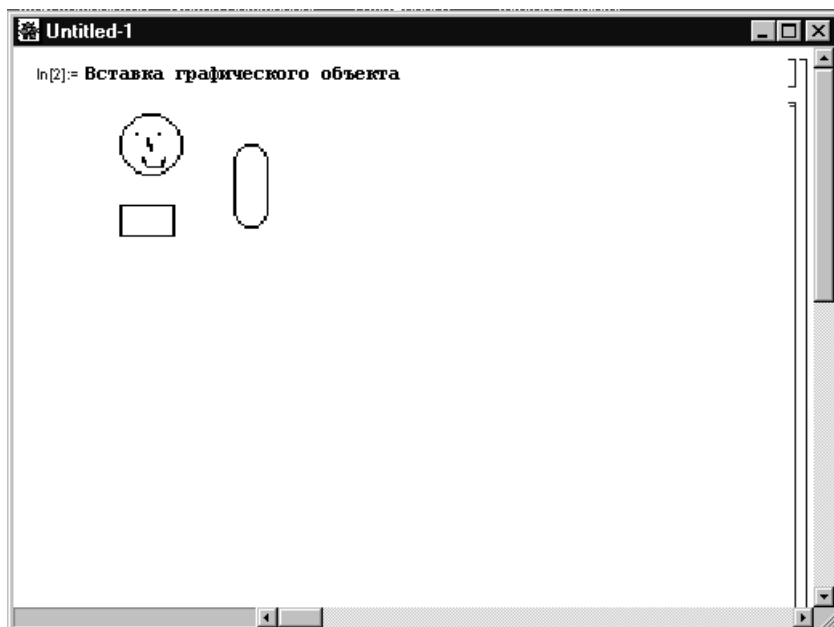


Рис. 8.42. Пример вставленного объекта, созданного в среде графического редактора Paint

Увы, Mathematica не способна воспринимать документ Mathcad целиком, если в нем больше одного блока, ибо каждый блок воспринимается как отдельный объект. Поэтому приходится располагать блоки Mathcad (выделяя их перед выходом) в отдельных ячейках системы Mathematica, что и демонстрирует рис. 8.44.

Из этого следует, что пока полноценной объектной связи Mathematica не реализует. И, по всей видимости, это сделано разработчиками намеренно. Неслучайно пары Mathematica-Word и Mathematica-Excel поставляются фирмой Wolfram как самостоятельные программные продукты.

8.10. Новые средства графики в Mathematica 6

8.10.1. Позиция *Graphics* меню и графический редактор

При создании сложных ноутбуков в прежних версиях Mathematica явно не хватало средств для подготовки хотя бы простых рисунков и диаграмм, которыми часто сопровождаются математические и научно-технические расчеты. Подобные рисунки и диаграммы, разумеется, можно создавать средствами программирова-

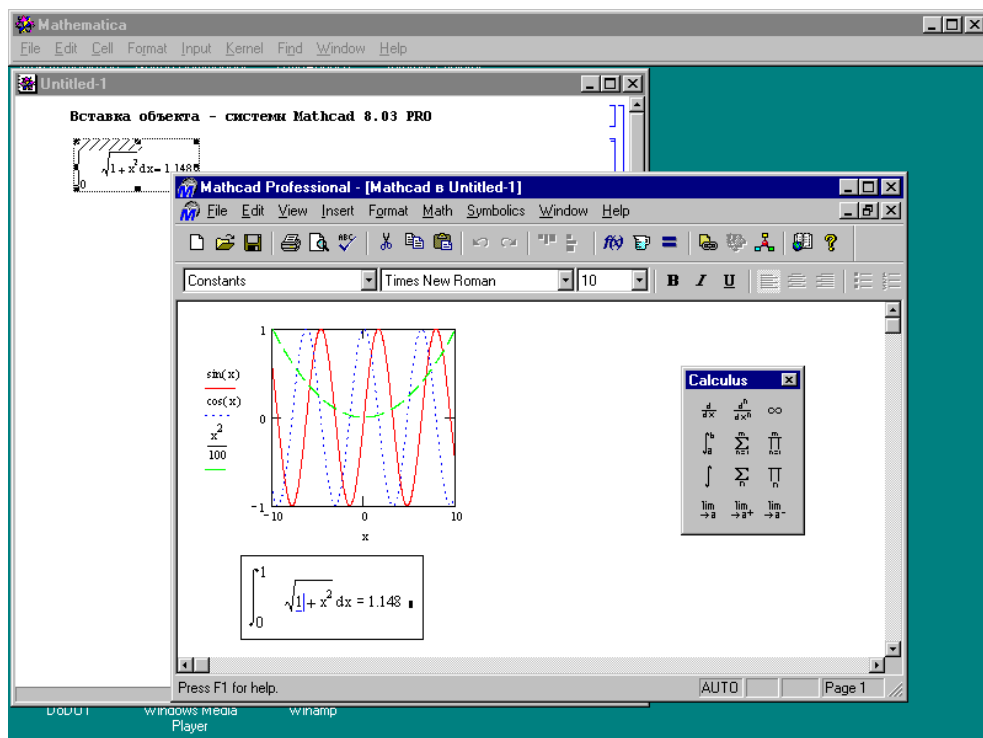


Рис. 8.43. Подготовка в Mathcad графика функций и вычисления определенного интеграла

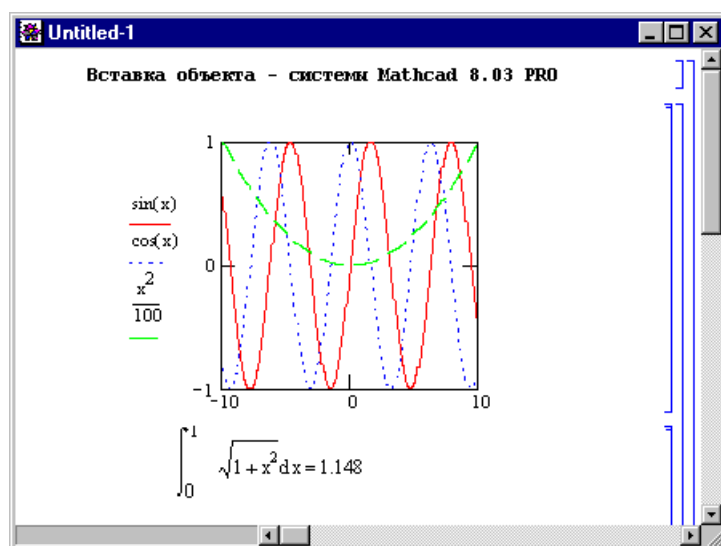


Рис. 8.44. Документ системы Mathematica с двумя объектами из документа Mathcad

ния систем Mathematica, но это требует много времени и умения хорошо программировать графические задачи.

Учтя это, разработчики Mathematica 6 ввели средство построения с помощью мыши простых рисунков по типу хорошо известного графического редактора Paint. Доступ к нему обеспечен с новой позиции Graphics меню. Она содержит следующие команды:

- **New Graphic** – вывод окна для построения графика;
- **Drawing Tool** – вывод окна графического редактора;
- **Graphics Inspector** – вывод окна инспектора графики;
- **Rendering** – вывод подменю операций рендеринга;
- **Opetations** – вывод подменю дополнительных операций.

Работа с указанными графическими средствами проста и очевидна. Ее иллюстрирует рис. 8.45. На нем показаны окна рисунка, графического редактора и инспектора графики. Отметим, что у графического редактора нет средств для построения незакрашенного эллипса, прямоугольника и полигона. Однако установкой цветов эти фигуры несложно получить.

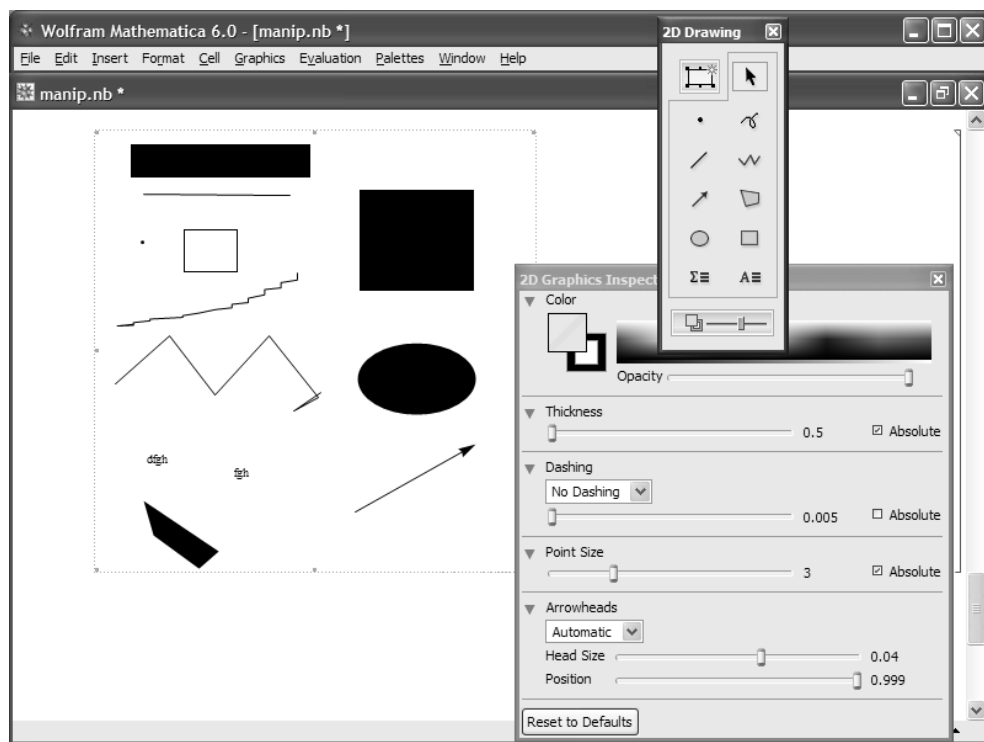


Рис. 8.45. Работа с графическим редактором и инспектором графики в системе Mathematica 6

8.10.2. Расширение возможностей функции Plot

В Mathematica 6 существенно расширены возможности функции построения двумерных графиков Plot. Это видно из перечня ее опций, который можно вывести, исполнив команду:

Options[Plot]

```
{AlignmentPoint→Center, AspectRatio→1/
GoldenRatio, Axes→True, AxesLabel→None, AxesOrigin→Automatic,
AxesStyle→{}, Background→None, BaselinePosition→Automatic, BaseStyle→{},
ClippingStyle→None, ColorFunction→Automatic, ColorFunctionScaling→True,
ColorOutput→Automatic, ContentSelectable→Automatic, DisplayFunction→
$DisplayFunction, Epilog→{}, Evaluated→Automatic, EvaluationMonitor→None,
Exclusions→Automatic, ExclusionsStyle→None, Filling→None, FillingStyle→
Automatic, FormatType→TraditionalForm, Frame→False, FrameLabel→None,
FrameStyle→{}, FrameTicks→Automatic, FrameTicksStyle→{}, GridLines→None,
GridLinesStyle→{}, ImageMargins→0., ImagePadding→All, ImageSize→Automatic,
LabelStyle→{}, MaxRecursion→Automatic, Mesh→None, MeshFunctions→{#1&},
MeshShading→None, MeshStyle→Automatic, Method→Automatic, PerformanceGoal→
$PerformanceGoal, PlotLabel→None, PlotPoints→Automatic, PlotRange→
{Full, Automatic}, PlotRangeClipping→True, PlotRangePadding→Automatic,
PlotRegion→Automatic, PlotStyle→Automatic, PreserveImageOptions→Automatic,
Prolog→{}, RegionFunction→(True&), RotateLabel→True, Ticks→Automatic,
TicksStyle→{}, WorkingPrecision→MachinePrecision}
```

Применение этих опций позволяет легко строить самые разнообразные, а порою просто невероятные по своей выразительности графики. Примеры построения таких графиков можно найти в разделе Options справки по функции Plot, а также в самоучителе Options for Graphics.

8.10.3. Использование опций закрашки областей двумерных графиков

Из новых опций функции **Plot** в Mathematica 6 наиболее эффектно выглядят опции закрашки областей двумерных графиков **Filling** (Закраска) и **FillingStyle** (Стиль закрашки). Первая по умолчанию отключена, вторая имеет значение **Auto** (Автоматический выбор стиля).

Прежде всего продемонстрируем действие опции **Filling** (рис. 8.46). На нем функцией **Plot** строится график функции **Sin[x]/x** в интервале изменения x от -4π до $+4\pi$ с 4 типами закрашки. Они представлены значениями опции **Filling**: **Axis** (окраска идет от каждой точки кривой до оси абсцисс), **Top** (окраска области от вершины окна графика до его кривой), **Bottom** (окраска от кривой до низа окна графика) и **0.5** (окраска от линии графика до горизонтали с вертикальной координатой, равной 0.5).

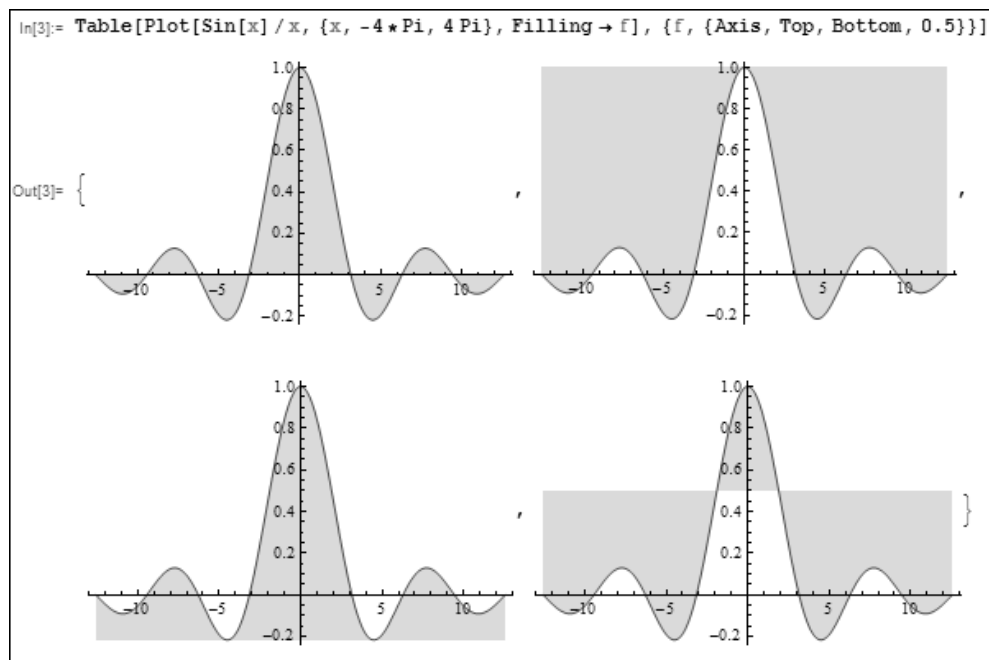


Рис. 8.46. Построение графика $\sin(x)/x$ с различными значениями опции окраски

Эта опция может использоваться и с функцией **ListPlot[list]**, строящей точки с ординатами, взятыми из списка **list**. Она дает возможность построения вертикалей, соединяющих точки графика с осью абсцисс (рис. 8.47). На этом рисунке показано, как меняется значение простых чисел от их номера. Любопытно, что эта зависимость близка к линейной.

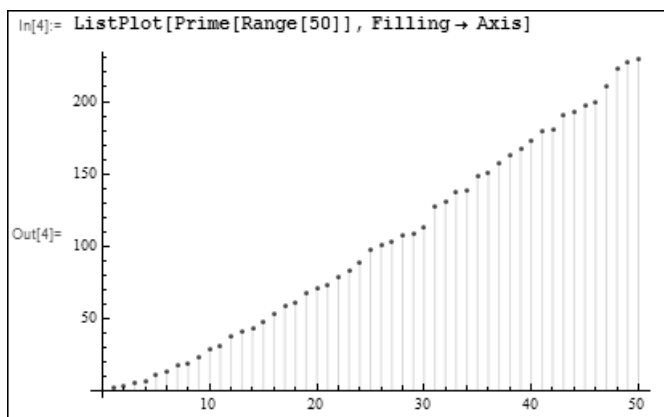


Рис. 8.47. Зависимость значений первых 50 простых чисел от их номера

Подобный вид графиков используется для представления амплитуды гармоник спектров при спектральном Фурье-анализе. Рисунок 8.48 дает пример такого рода. Здесь строится график синусоидального сигнала с интерполяцией между его узловыми точками. Сигнал выглядит как построенный сплошной линией, но на самом деле он задан как дискретный сигнал. После этого строится график спектра, причем знак гармоник, задающий их фазу, не учитывается. Спектр сигнала симметричный и представлен вертикальными отрезками прямых с точкой над каждым отрезком.

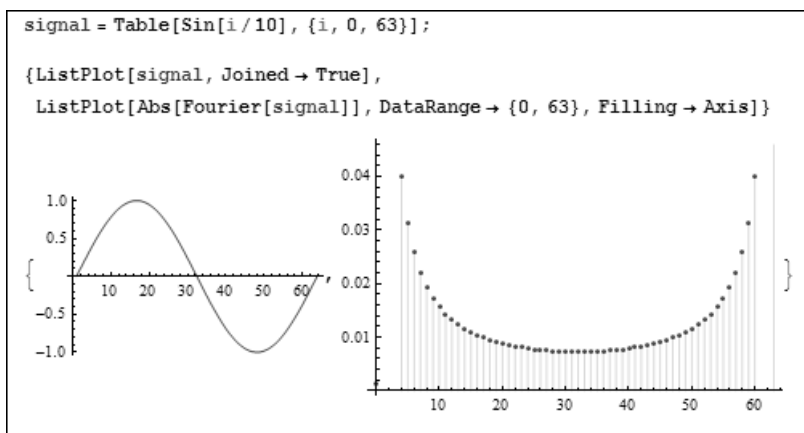


Рис. 8.48. Построение графика синусоидального сигнала и его дискретного прямого преобразования Фурье

Рисунок 8.49 демонстрирует закраску областей между двумя кривыми – синусоидой и косинусоидой. Разумеется, что такая окраска возможна как для кривых периодических, так и непериодических функций.

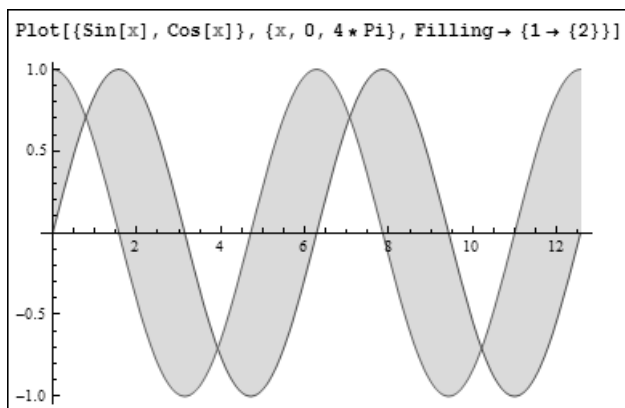


Рис. 8.49. Пример закраски областей между двумя кривыми

Приведенные примеры составляют лишь малую часть примеров применения опций закрашки. В справке можно найти многие десятки примеров на построение рисунков с использованием различных видов закрашки, в том числе различных областей разными цветами, разными стилями закрашки и т.д.

8.10.4. Графические динамические модули в Mathematica 6

Возможность в динамике (в режиме реального времени) изменять те или иные параметры графических объектов и тут же наблюдать вызванные этим их изменения – еще одна замечательная особенность системы Mathematica 6. Она реализуется с помощью функций-модулей **DynamicModule** и **Manipulate**. Подробные сведения о них даны в Главе 10. Ниже представлены характерные примеры их применения в графике.

На рис. 8.50 показан модуль на основе функции **Manipulate**, который реализует основные методы закрашки графика функции одной переменной. Функция является суммой двух гармонических колебаний с кратными частотами. Кратность может меняться с помощью двух слайдеров, при этом можно наблюдать множество кривых. Опция **ControlType>SetterBar** задает меню-бар, позволяющее задать и наблюдать 10 вариантов закрашки различных областей графика.

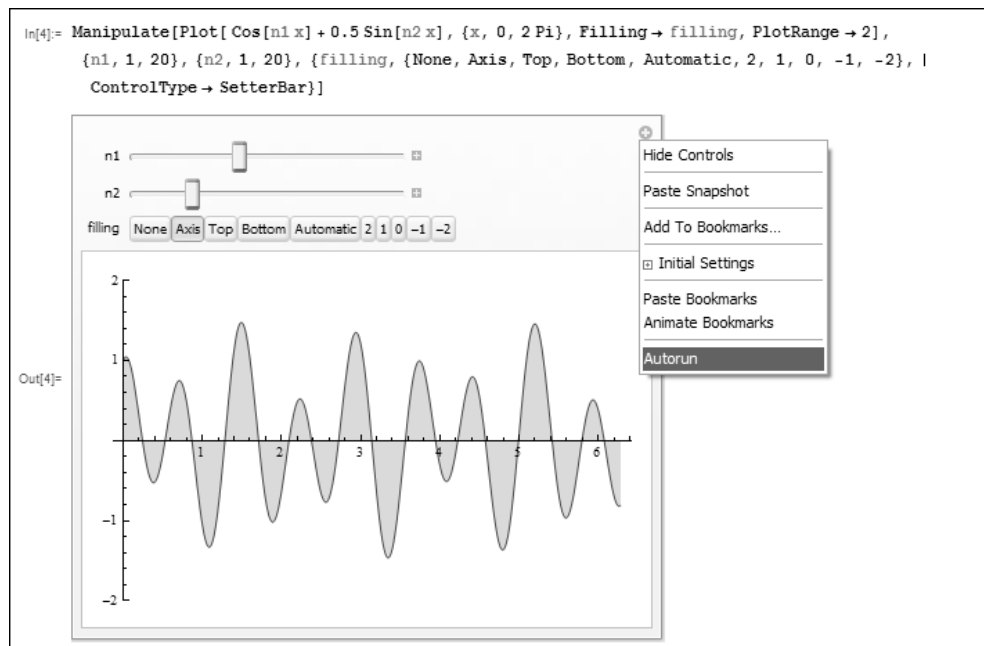


Рис. 8.50. Модуль, иллюстрирующий построение графика суммы двух гармонических колебаний с изменяемой кратностью частот и различные виды закрашки графика с помощью меню

Рисунок 8.51 прекрасно иллюстрирует идею получения любого цвета смешением трех базовых цветов: red – красного, green – зеленого и blue – синего. Этот простой модуль строит круг, покрашенный смесью указанных цветов, причем интенсивность каждого цвета задается одним из трех слайдеров. Данный модуль хорошо иллюстрирует суть *RGB-метода*. К сожалению, в данной книге цвета на рисунках (в том числе рис. 8.50) не воспроизводятся, и вместо них наблюдаются лишь оттенки серого цвета (тип окраски grayscale).

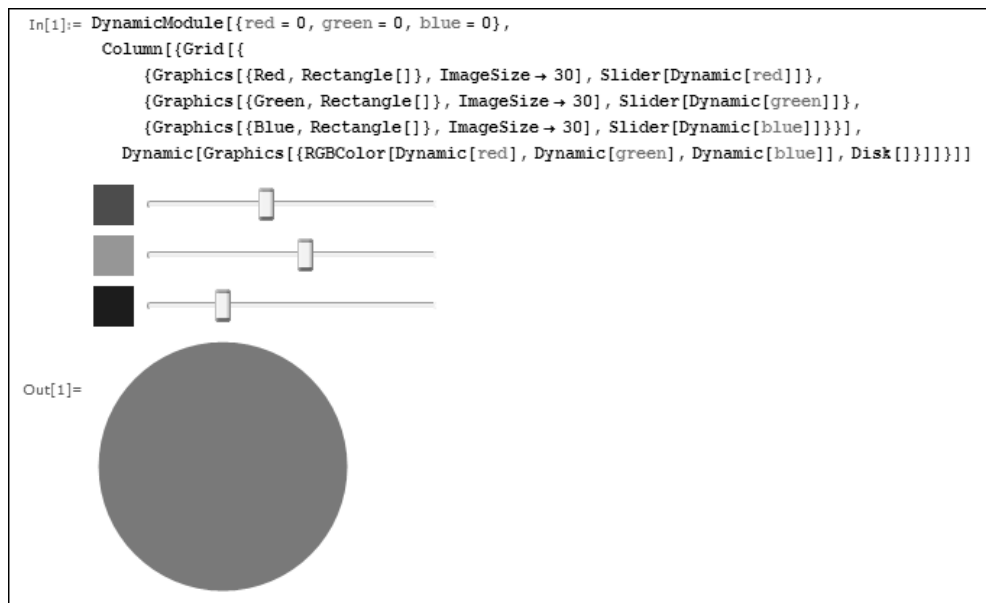


Рис. 8.51. Демонстрация сложения трех цветов: красного, зеленого и синего

Данный модуль построен на основе динамического модуля **DynamicModule**, наполнение которого довольно очевидно и может быть взято пользователем за основу разработки своих подобных модулей.

Следующий модуль, показанный на рис. 8.52, является прекрасной иллюстрацией полиномиальной аппроксимации. Он, с помощью локаторов, задает 4 точки и осуществляет их классическую аппроксимацию полиномом третьей степени. Сплошной линией строится график полинома, который всегда точно проходит через точки – локаторы. Замечательным является то, что мышью можно смещать точки с их начального состояния, заданного в модуле, и тут же наблюдать изменение графика полинома. Нетрудно убедиться в том, что точное прохождение графика полинома через узловые точки отнюдь не гарантирует его близость к ним в промежутках между ними. В частности, в этих промежутках возможен значительный выбег графика полинома.

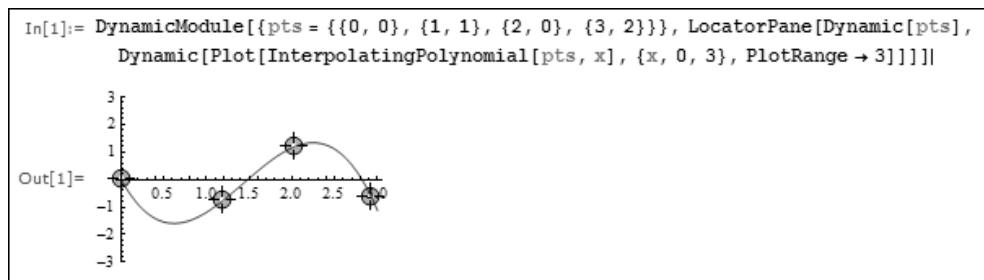


Рис. 8.52. Иллюстрация полиномиальной аппроксимации для четырех точек

Динамический модуль, показанный на рис. 8.53, иллюстрирует построение графика в полярной системе координат с помощью задатчика угла, заданного функцией **angularSlider**. Задатчик угла выглядит как окружность, внутри которой помещен радиус-вектор, угловое положение которого задает изменяемый угол. Изменение положения радиус-вектора осуществляется мышью.

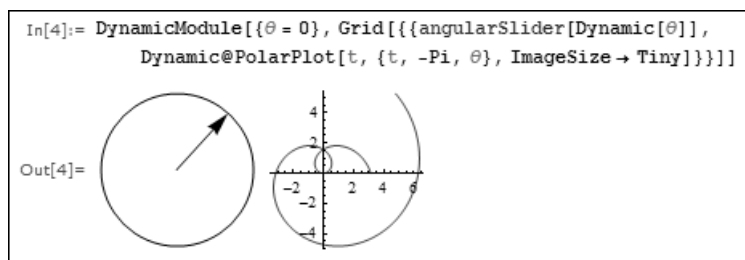


Рис. 8.53. Пример, иллюстрирующий построение графика функции в полярной системе координат при изменении угла поворота радиус-вектора

Наконец, на рис. 8.54 показан модуль, строящий объемную фигуру, которую можно вращать в пространстве с помощью двухкоординатного слайдера. При этом двухкоординатный слайдер изменяет два угла поворота фигуры.

8.10.5. Визуализация данных из списков

В Mathematica 6 существенно расширены и обновлены графические функции для визуализации данных, хранящихся в списках. Так, есть следующие модифицированные функции для построения точек в декартовой системе координат:

- **ListPlot**[{ y_1, y_2, j }] – построение точек списка, заданных ординатами;
- **ListPlot**[{ $\{x_1, y_1\}, \{x_2, y_2\}, j$ }] – построение точек списка, заданных координатами;
- **ListPlot**[{ $list_1, list_2, \dots$ }] – построение нескольких групп точек, заданных списками.

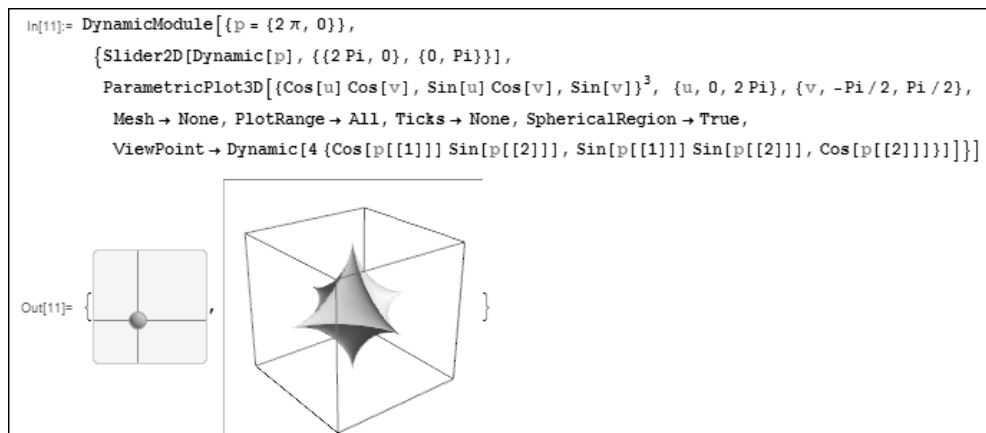


Рис. 8.54. Пример, иллюстрирующий вращение трехмерной фигуры в пространстве с помощью двухкоординатного слайдера

Аналогичная по синтаксису записи новая функция **ListLinePlot** строит графики, соединяя точки отрезками линий. На рис. 8.55 показано построение с помощью данной функции звезды.

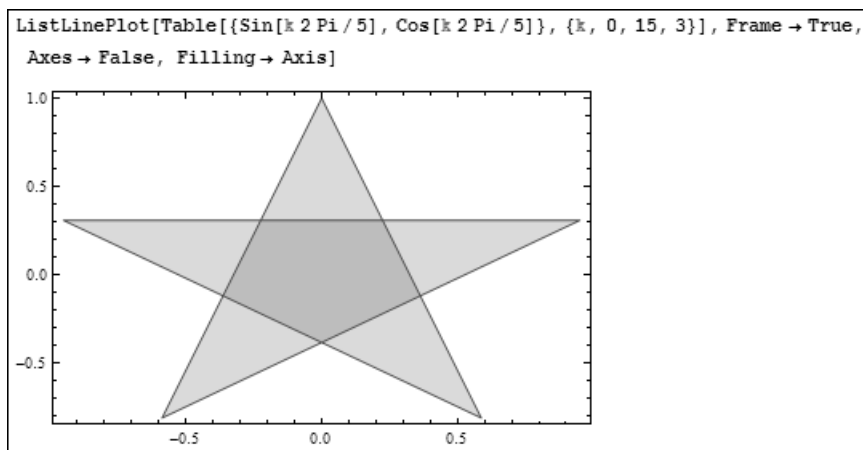


Рис. 8.55. Построение звезды

Еще один пример, показанный на рис. 8.56, показывает график роста 500 случайных чисел благодаря аккумулярованию их значений, которые лежат в интервале от -1 до $+1$. Кривая роста носит случайный характер и меняется при каждом пуске заданной графической функции. Кроме того, тут задается случайный цвет закрашки.

В справке можно найти десятки примеров на применение этих функций с разными опциями. Для визуализации функций двух переменных (поверхностей и

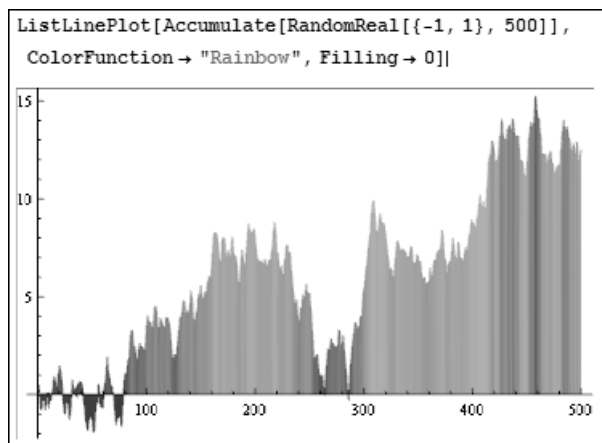


Рис. 8.56. График роста аккумулярованного значения 500 случайных чисел со случайным цветом закрашки

трехмерных фигур) в виде контурных графиков служат следующие графические функции:

ListContourPlot [array] **ListContourPlot** [{ { x_1, y_1, f_1 }, { x_2, y_2, f_2 }, ... }]
ListContourPlot3D [array] **ListContourPlot3D** [{ { x_1, y_1, z_1, f_1 }, { x_2, y_2, z_2, f_2 }, ... }]

Ограничимся примером применения функции **ListContourPlot** для построения контурного графика среза головы человека, данные которого импортируются из файла (см. рис. 8.57).

Для построения трехмерных графиков по данным точек служат функции:

ListPlot3D [array] **ListPlot3D** [{ { x_1, y_1, z_1 }, { x_2, y_2, z_2 }, ... }]
ListPlot3D [{ $data_1, data_2, \dots$ }] **ListSurfacePlot3D** [{ { x_1, y_1, z_1 }, { x_2, y_2, z_2 }, ... }]
RegionPlot3D [$pred, \{x, x_{min}, x_{max}\}, \{y, y_{min}, y_{max}\}, \{z, z_{min}, z_{max}\}$]

Рисунок 8.58 дает пример построения поверхности по точкам, заданным таблицей для функции $\cos(j^2+i)$. Верхний рисунок получен при отсутствии интерполяции (порядок интерполяции 0), а второй при применении квадратичной интерполяции (порядок интерполяции 2). Нетрудно заметить, что применение интерполяции позволяет получить весьма реалистичное изображение поверхности. Однако представление поверхности без интерполяции нередко преследует достижение специальных целей – например, наглядного сравнения представлений поверхности в отсутствии и при наличии интерполяции.

Mathematica имеет ряд красочных фигур, оформленных в виде примеров данных (ExampleData). Их можно использовать для проверки работы графических функций. На рис. 8.59 показано построение объемной фигуры из числа таких примеров с помощью функции **ListSurfacePlot**. При этом используется только опция, задающая число фрагментов фигуры. Остальные опции заданы по умолчанию.

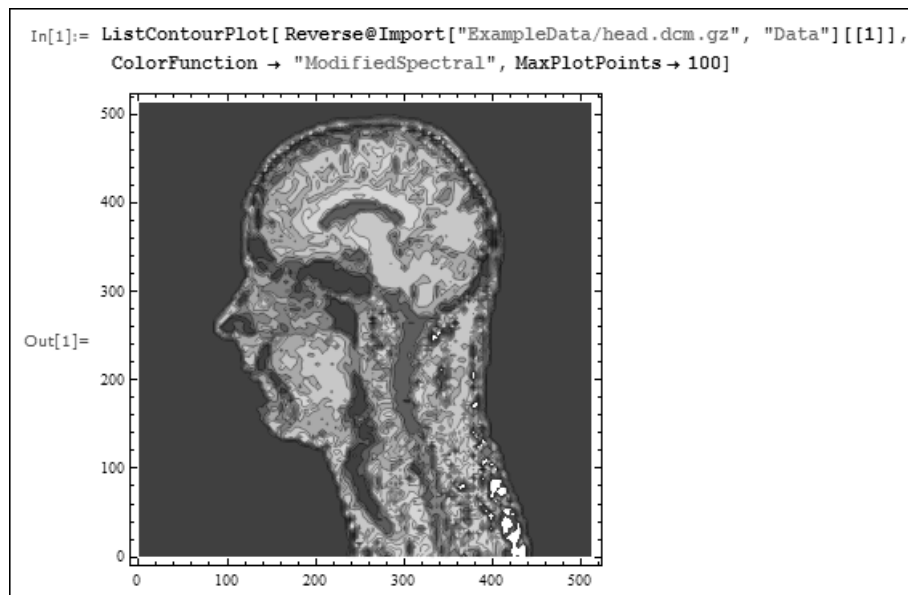


Рис. 8.57. Пример построения контурного графика для среза головы человека

А на рис. 8.60 дано подобное построение при использовании других опций функции **ListSurfacePlot**. Нетрудно заметить, что изображение на рис. 8.60 сильно отличается от показанного на рис. 8.59.

8.10.6. Рельефная графика

Новая функция системы Mathematica 6 **ReliefPlot[array]** служит для построения реалистических графиков рельефа, который задается ординатами точек массива. Примером применения этой функции служит рис. 8.61, на котором построен рельеф поверхности, заданной математической формулой $i + \cos(i^3 + j^3)$, где i и j меняются с дискретностью 0,03 в интервале от -4 до 4 . Вид рельефа зависит от значения опции **ColorFunction**.

Еще один пример применения функции **ReliefPlot** представлен на рис. 8.62. Здесь задаются три массива случайных результатов ряда арифметических операций, включающих нормы матриц. Полученные три рельефа являются в значительной мере случайными и меняются от пуска к пуску представленного модуля.

Рисунок 8.63 строит рельеф мнимой части функции $\sec(i + I*j)^2$ для двух значений опции **PlotRange**, равных **All** и **Automatic**. Нетрудно заметить серьезное изменение характера выявления деталей одного и того же рельефа.

Разумеется, массив для этой функции может создаваться не только математическими выражениями, но и любыми другими способами – например, загрузкой массивов рисунков. Множество опций функции **ReliefPlot** позволяет создавать рельефы с разным разрешением, разной окраской и другими особенностями.

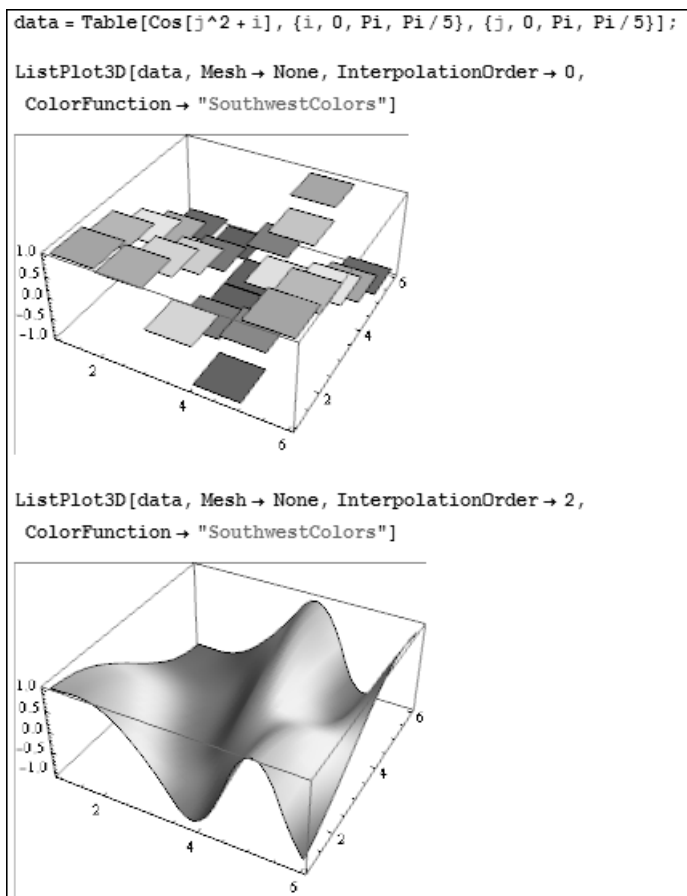


Рис. 8.58. Построение поверхности по точкам без интерполяции (верхний рисунок) и с квадратичной интерполяцией (нижний рисунок)

8.10.7. Трехмерные объекты, полученные вращением кривых

Довольно широко распространенными являются трехмерные графические объекты, полученные вращением кривых относительно некоторой оси. Например, поворачивая окружность на угол π , можно получить поверхность шара. Меняя пределы изменения угла поворота, можно строить замкнутые или незамкнутые фигуры. Для построения таких поверхностей (фигур) в Mathematica 6 служит функция:

```
RevolutionPlot3D[fz, {t, tmin, tmaxx, fy, fz}, {t, tmin, tmax

```

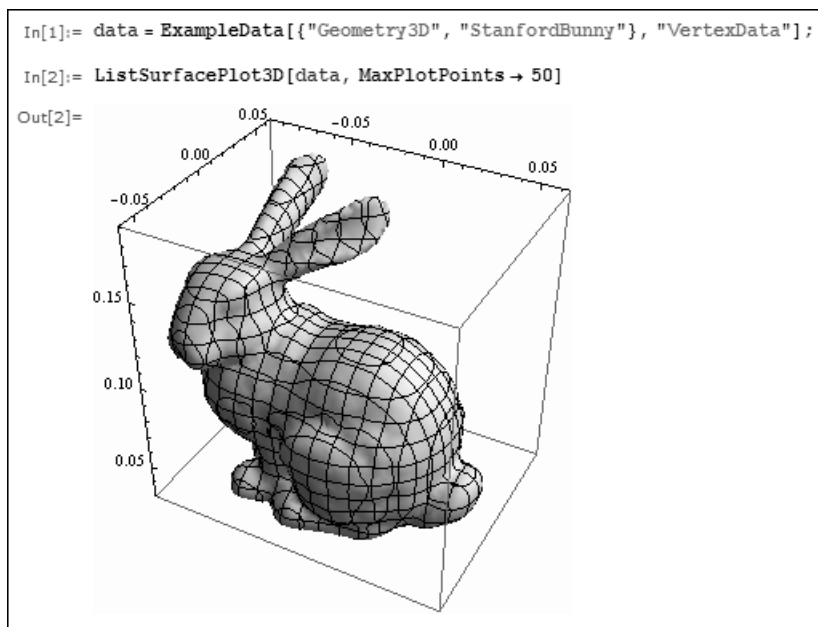


Рис. 8.59. Построение объемной фигуры функцией **ListSurfacePlot**

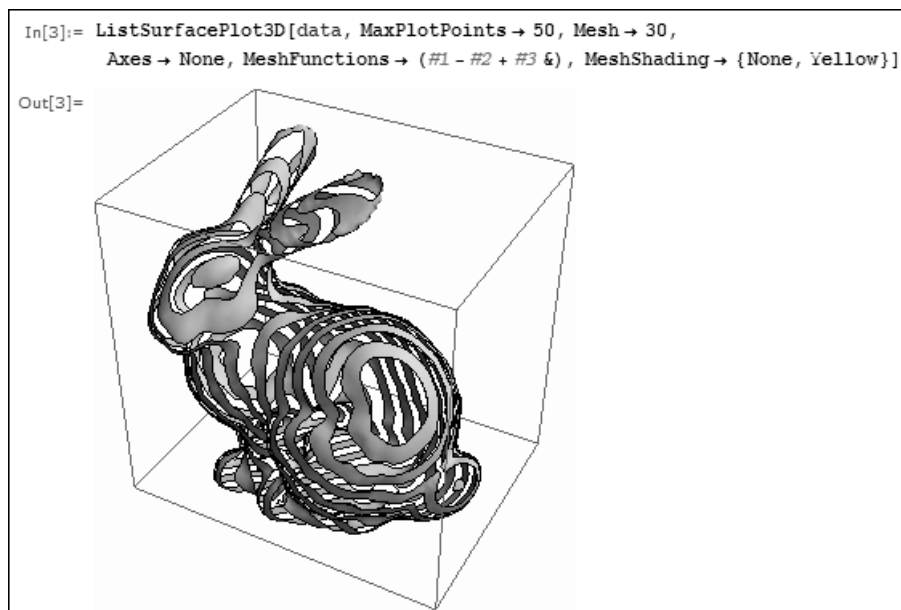


Рис. 8.60. Построение объемной фигуры функцией **ListSurfacePlot** с иным, чем на рис. 8.59, набором опций

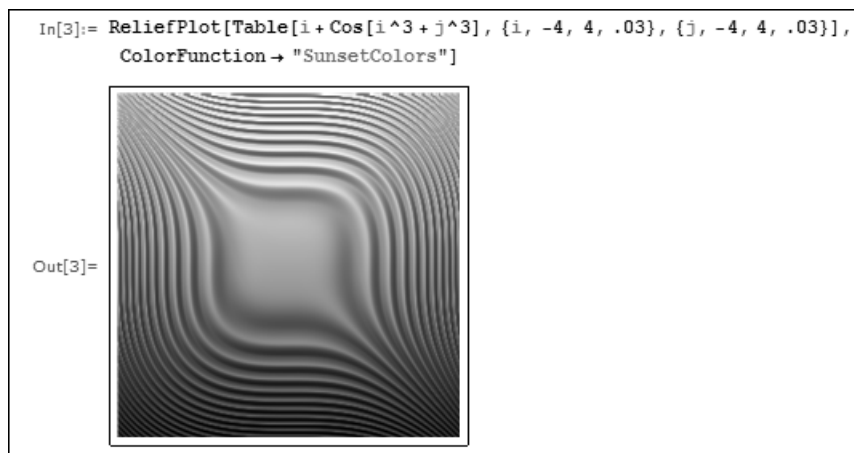


Рис. 8.61. Построение графика рельефа поверхности, заданной формулой $i + \cos(i^3 + j^3)$

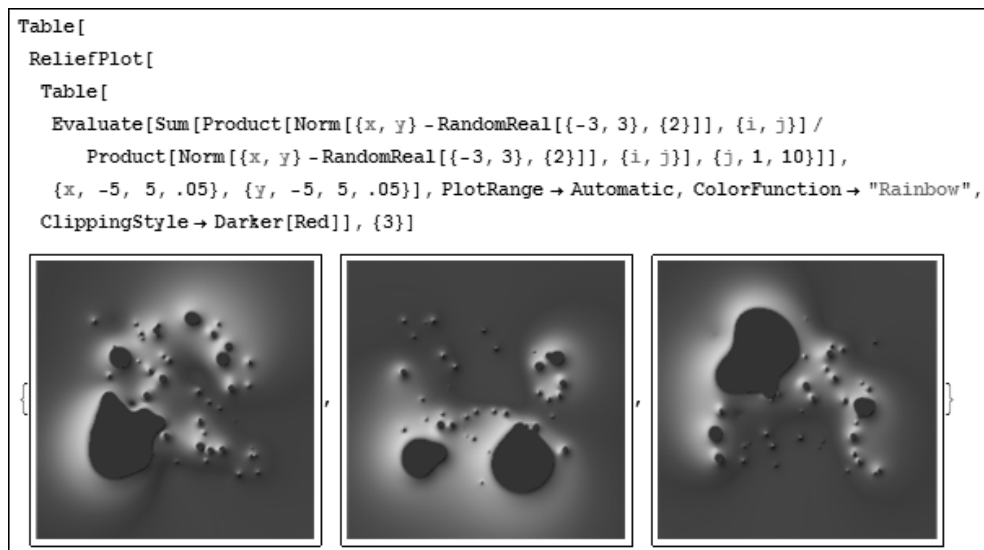


Рис. 8.62. Создание трех случайных рельефов

На рис. 8.64 показано применение этой функции для построения поверхности половинки бублика. Фигура выглядит достаточно реалистично.

Более забавная фигура строится применением этой функции, показанным на рис. 8.65. Здесь параметрами являются два изменяющихся угла, и для вращения используется параметрическая кривая.

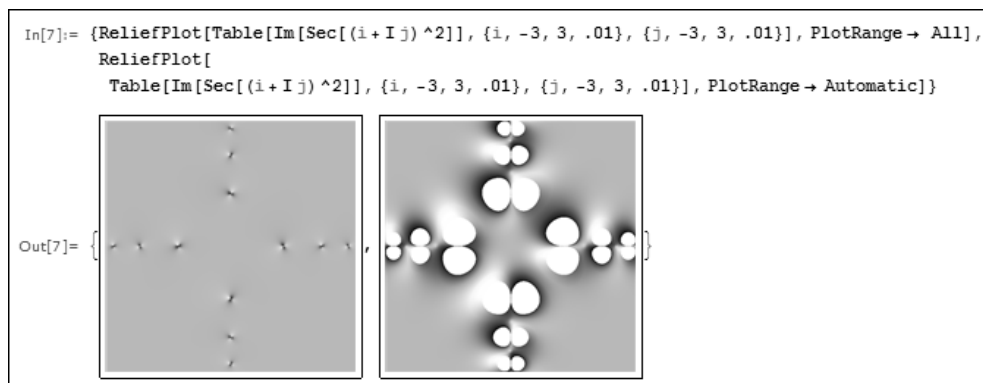


Рис. 8.63. Рельеф мнимой части функции $\sec(i + I j)^2$ для двух значений опции *PlotRange*, равных *All* и *Automatic*

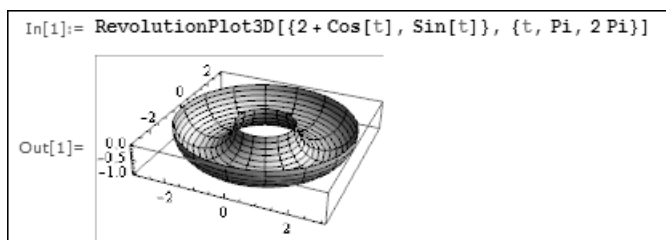


Рис. 8.64. Построение фигуры «половинка бублика»

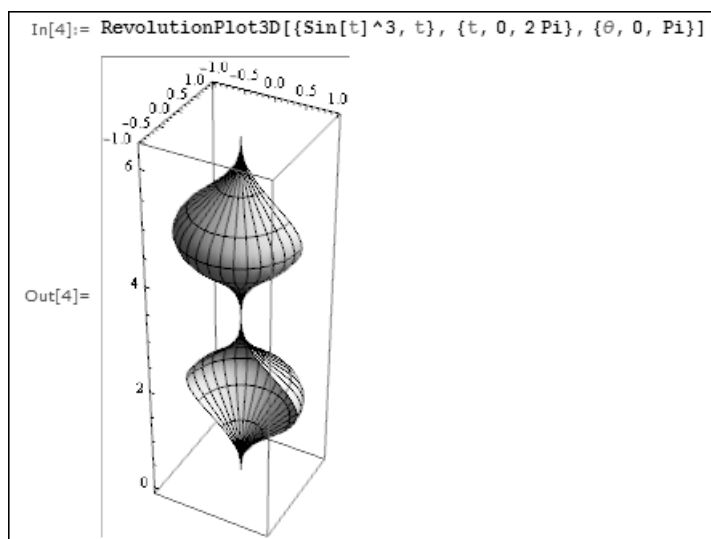


Рис. 8.65. Построение трехмерной фигуры

Пример применения опций функции **RevolutionPlot3D** представлен на рис. 8.66. Здесь кривая вращения задается с помощью шести неравенств, с учетом которых строятся 6 фигур. К сожалению, как и ранее, цветовая окраска фигур воспроизводится лишь оттенками серого цвета.

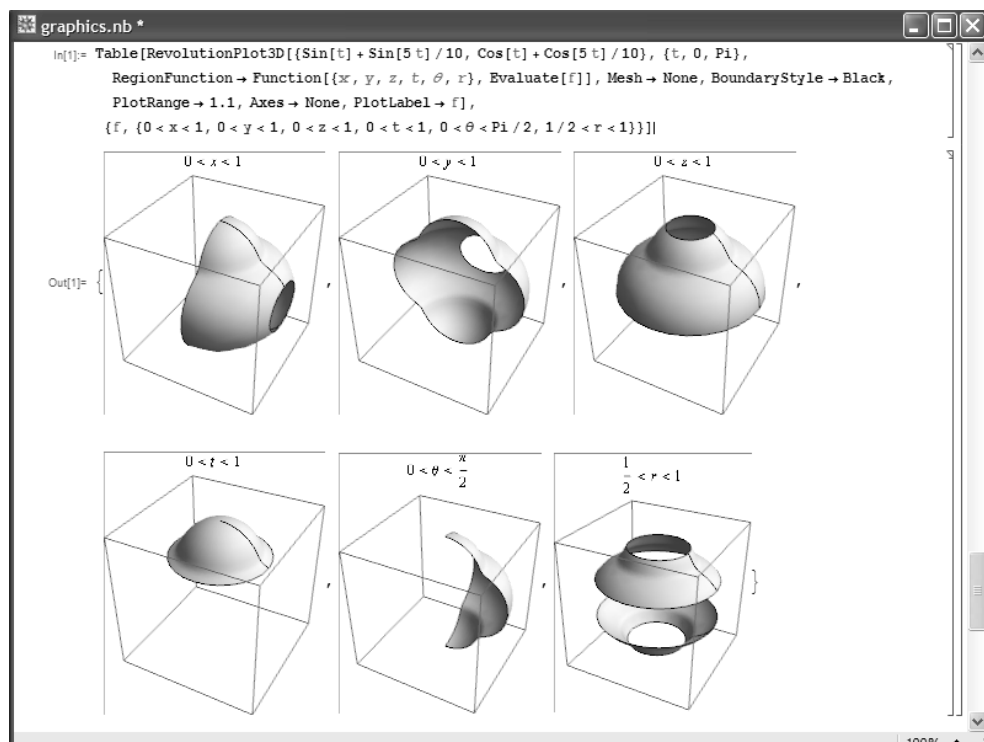


Рис. 8.66. Построение шести вариантов пространственных фигур с применением опций

Как и другие графические функции, функция **RevolutionPlot3D** может строить большое число интересных фигур.

8.10.8. Визуализация работы клеточных автоматов

Комбинируя по определенным алгоритмам на клеточной доске темные и светлые квадраты, можно получить порою очень неожиданные фигуры, напоминающие очереди, фракталы и иные графические объекты с весьма неожиданными математическими и художественными свойствами [56]. Одним из таких свойств является самоподобие фигур и возможность их бесконечного дробления.

Mathematica имеет функцию реализации клеточных автоматов **CellularAutomaton**, общая форма записи которой следующая:

CellularAutomaton[rule,init,{t,All,...}]

Возможны и упрощенные формы записи. Функция задается спецификацией rule и начальными условиями init. Алгоритм работы функции описан в разделе MORE INFORMATION справки по этой функции. Приведем простой пример ее работы (rule=30, задано два шага):

```
CellularAutomaton[30,{0,0,0,1,0,0,0},2]
{{0,0,0,1,0,0,0},{0,0,1,1,1,0,0},{0,1,1,0,0,1,0}}
```

А на рис. 8.67 с помощью функции **ArrayPlot** демонстрируется построение созданной из темных квадратиков фигуры при 60 шагах работы клеточного автомата. Тут уже начинают появляться первые признаки самоподобия (треугольники в треугольниках) фигуры.

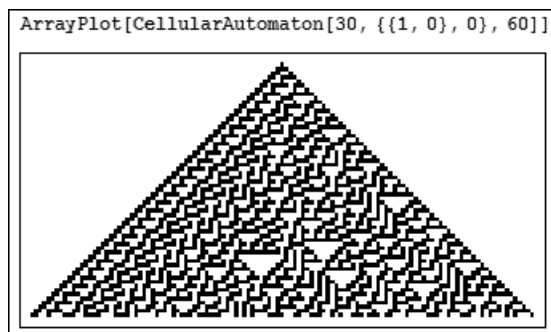


Рис. 8.67. Фигуры, построенные клеточным автоматом при rule=30 и 60 шагах работы

О том, насколько разнообразные фигуры может создавать клеточный автомат при небольшом изменении параметров функции **CellularAutomaton**, свидетельствует рис. 8.68 с множеством примеров применения данной функции.

Еще один рисунок – рис. 8.69 – наглядно показывает, что графика клеточных автоматов не лишена художественной ценности. Некоторые рисунки напоминают художественный орнамент витражей и мозаик.

Функция **CellularAutomaton** позволяет получать и объемные фигуры. Рисунок 8.70 демонстрирует построение кубической фигуры с помощью функции **Graphics3D** с примитивом **Cuboid**, внутри которой размещены трехмерные фигуры, построенные с помощью функции **CellularAutomaton**. Рисунок 8.70 дает прекрасное представление о комбинированных графиках, которые способна строить система Mathematica 6.

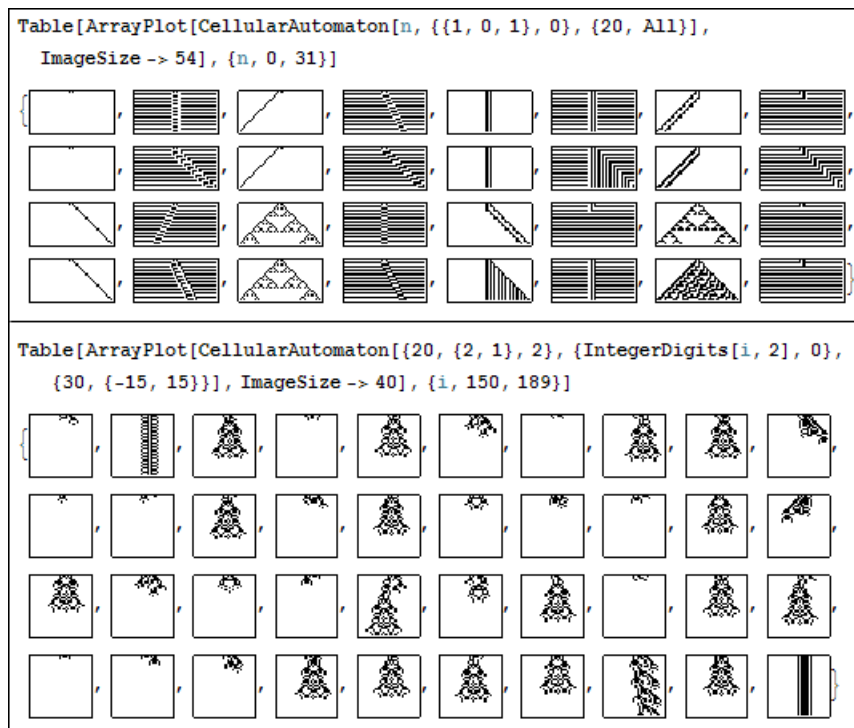


Рис. 8.68. Примеры построения функций **CellularAutomaton** множества фигур на плоскости

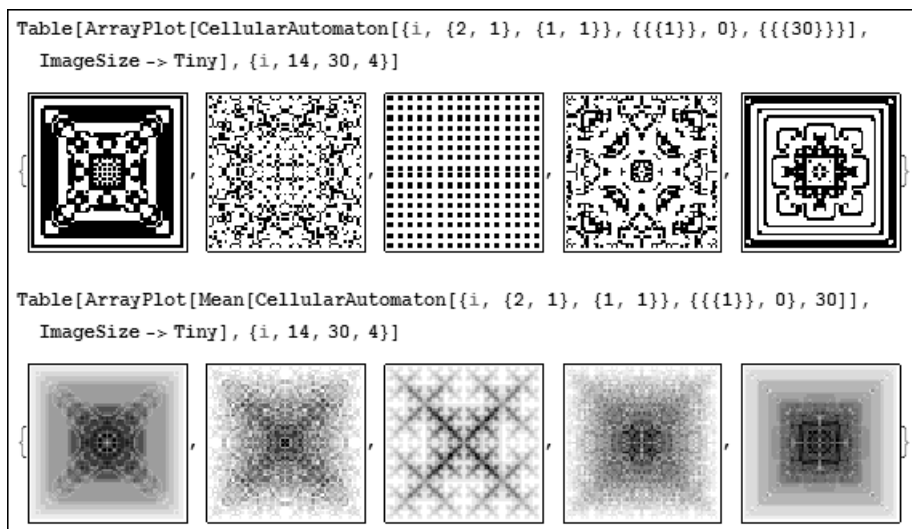


Рис. 8.69. Дополнительные примеры творчества клеточных автоматов

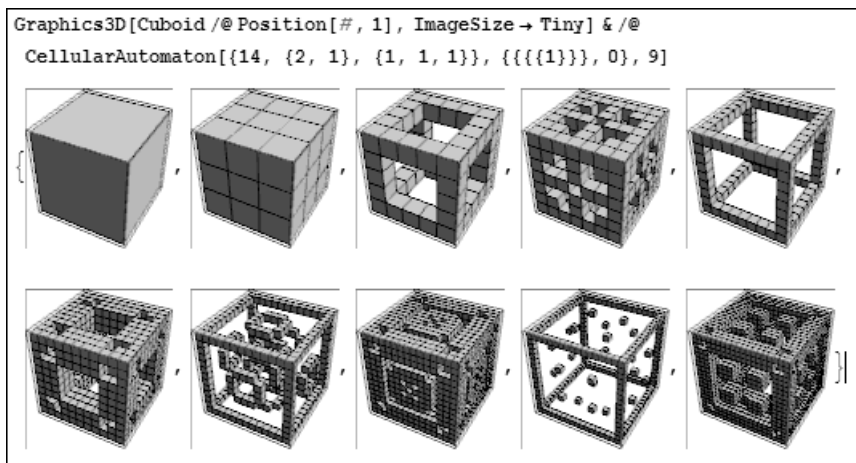


Рис. 8.70. Комбинированные графики – кубы, заполненные фигурами, которые строит клеточный автомат

8.10.9. Графы, деревья и прочее

Графы и цепи являются объектами, хорошо известными в математике и в научно-технических расчетах. Их полноценная поддержка также обеспечена в системе Mathematica. На рис. 8.71 приведены примеры применения функций **GraphPlot** графической поддержки построения двумерных графов и **GraphPlot3D** для поддержки трехмерных графов. С помощью опций этих функций можно обеспечить построение практически любых графов.

Для построения множества видов деревьев служит функция **TreePlot**. Примеры ее применения для построения обычных деревьев показаны на рис. 8.72.

Из деревьев могут быть построены довольно занятные и сложные фигуры. Примером этого служат фигуры, показанные на рис. 8.73, и построенные с помощью функции **TreePlot**. В справке по этой функции можно найти много и других примеров ее применения.

Как уже отмечалось, в задачи данной книги не входит полное описание всех функций системы Mathematica 6 и примеров их применения. Описаны лишь основные и наиболее важные функции.

8.11. Функции пакета расширения Graphics

8.11.1. Функции анимационной графики

Пакет расширения Graphics в системах Mathematica 5.1/5.2 дает множество мощных дополнительных средств для построения графиков самого изысканного вида. Он является прекрасным инструментом для визуализации задач, допускающих

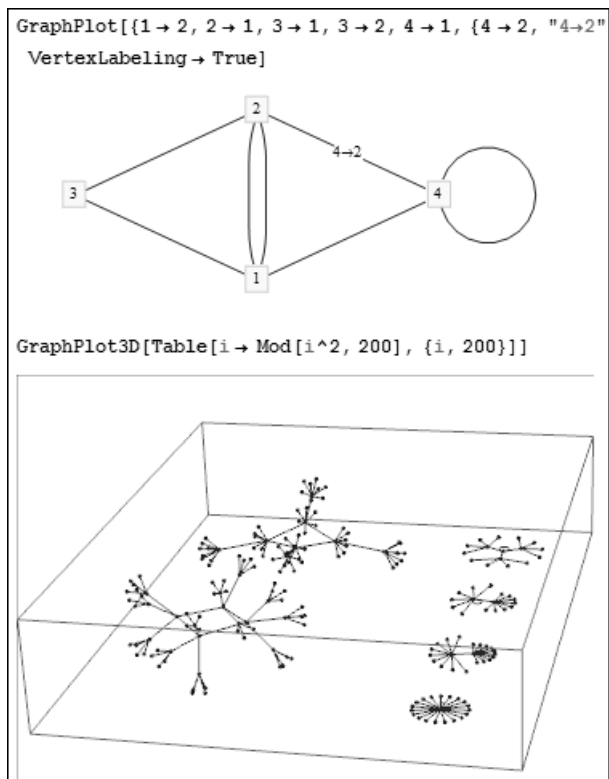


Рис. 8.71. Примеры построения двумерных и трехмерных графов

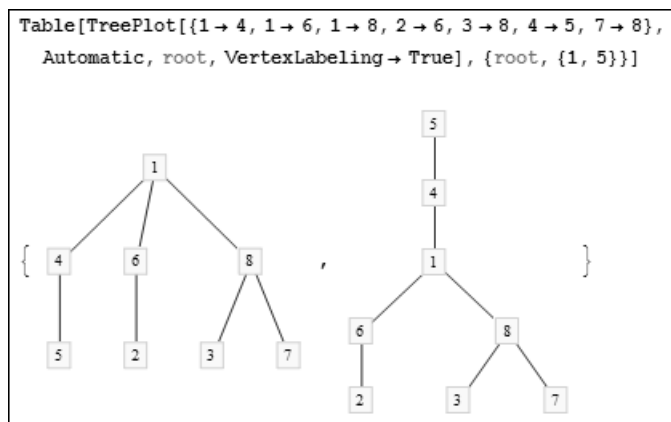


Рис. 8.72. Примеры построения простых деревьев

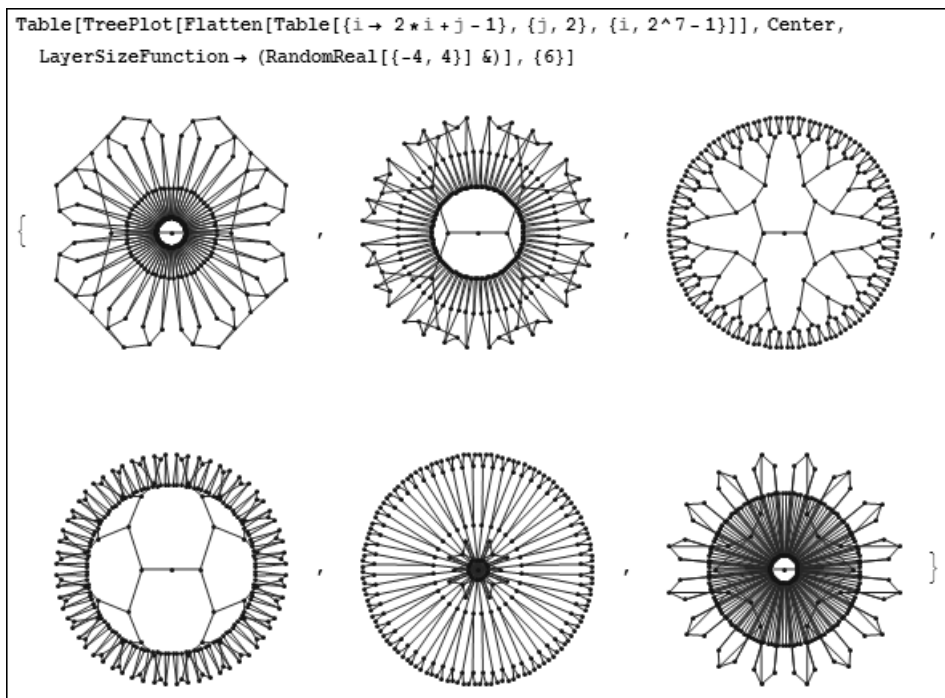


Рис. 8.73. Сложные фигуры, построенные с применением функции *TreePlot*

представление результатов в графической форме. Начнем их обсуждение с техники анимации.

Техника *анимации* (оживления) графиков сводится к подготовке отдельных кадров анимационного рисунка, которые специфицируются особой изменяющейся переменной t . Это не обязательно время, возможно, что t задает размеры изображения, его место или иную характеристику. Естественно, что имя переменной можно выбрать произвольно.

Подпакет *Animation* задает две важнейшие функции:

- **Animate[grcom,{t,tmin,tmax,dt}]** – задает построение серии графических объектов *grcom* при изменении параметра t от $tmin$ до $tmax$ с шагом dt .
- **ShowAnimation[{p1,p2,p3,...}]** – дает анимацию последовательным воспроизведением ранее подготовленных объектов $p1,p2,p3,...$

Рисунок 8.74 показывает пример подготовки к анимации простого графика – функции $n \cdot \sin[x]/x$ при n , меняющемся от 0.1 до 1 с шагом 0.1. Таким образом, демонстрируется изменение данной функции по высоте (амплитуде). Опцией *PlotRange* задан фиксированный масштаб для всех кадров анимации. При выполнении анимации внизу окна документа появляются кнопки анимационного проигрывателя (рис. 8.74).

Следующий пример иллюстрирует задание анимации графика с параметрическим заданием функции:

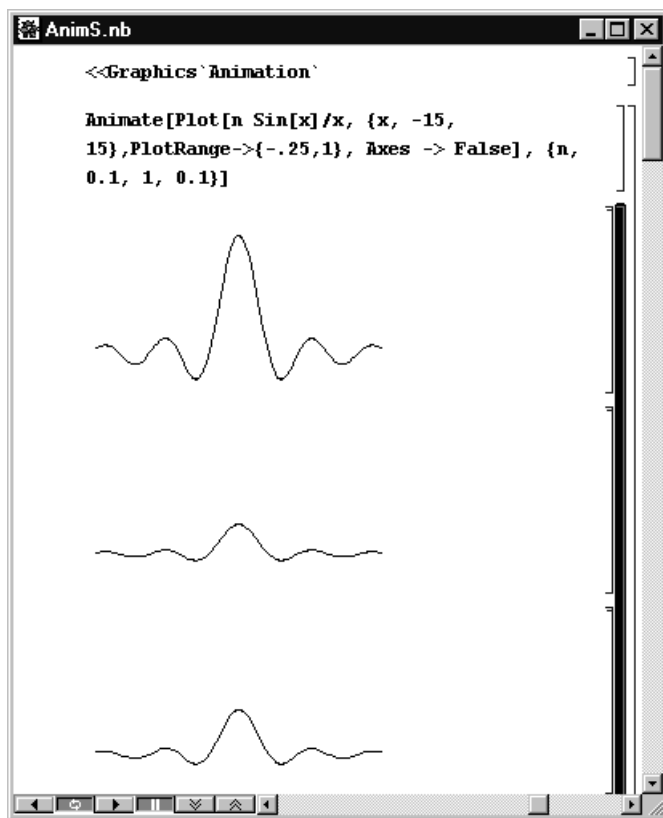


Рис. 8.74. Стоп-кадр анимации графика функции $n \sin[x]/x$

```
ShowAnimation[Table[ Graphics[Line[0, 0, Cos[ t], Sin[t]],
PlotRange -> -1, 1, -1, 1], t, 0, 2Pi, Pi/8]]
```

Запустив этот фрагмент программы, вы увидите построение отрезка прямой, вращающегося вокруг одного неподвижного конца. Здесь для анимации вначале строится набор кадров, а затем используется функция **ShowAnimation**.

Аналогичным образом осуществляется анимация трехмерных графиков поверхностей или фигур. Рисунок 8.75 показывает начало подготовки к анимации сложной поверхности, описываемой функцией Бесселя, аргумент которой меняется от кадра к кадру.

Обратите внимание на применение функций **Show** и **GraphicsArray** для построения на одном графике сразу всех кадров (фаз) анимации. Порой этот набор кадров даже важнее, чем анимация, длящаяся доли секунд или несколько секунд. Теперь необходимо выполнить команду

```
ShowAnimation[%%]
```

Для упрощения анимации сложных графиков в подпакете Animations заданы функции:

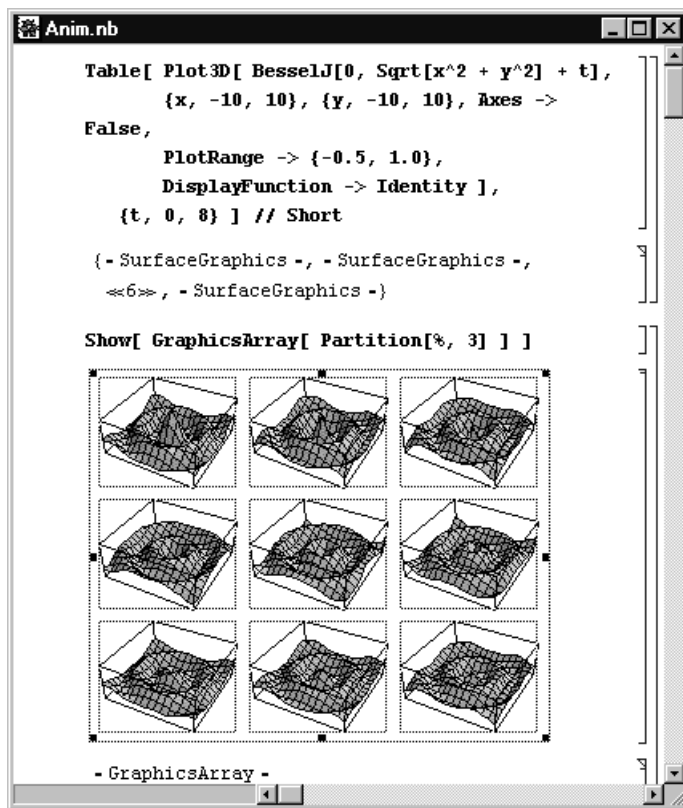


Рис. 8.75. Подготовка к анимации сложной 3D поверхности

- **MovieParametricPlot**[[{f[s,t],{g[s,t]}},{s,smin,smax},{t,tmin,tmax}] – дает анимацию параметрического графика.
- **SpinShow**[graphics] – дает вращение графического объекта. Функция имеет ряд опций, которые можно просмотреть командой **Options**[SpinShow].

Пример построения сложной вращающейся в пространстве фигуры, напоминающей гантели, показан на рис. 8.76. В данном случае трехмерная фигура задана в параметрической форме, а для последующей ее анимации используется функция **SpinShow**.

8.11.2. Управление цветом графиков

При построении графиков в полярной системе координат полезно указывать цвет комплексной величины. Для этого в подпакете ArgColor служат такие функции:

- **ArgColor**[z] – дает цвет, определяемый комплексным аргументом z.
- **ArgShade**[z] – дает уровень серого цвета, определяемый комплексным аргументом z.

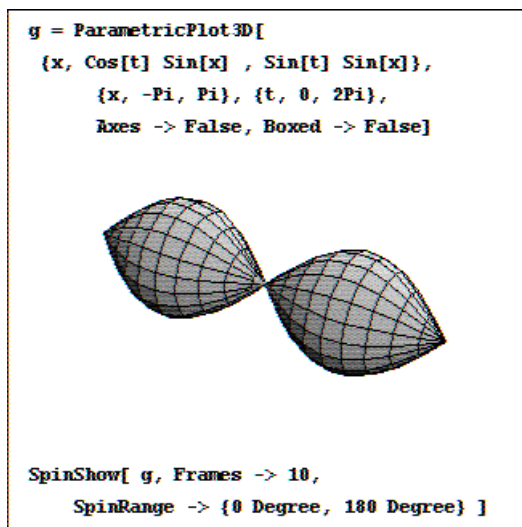


Рис. 8.76. Построение вращающейся в пространстве фигуры – «гантели»

Действие функции **ArgShade** иллюстрирует показанный на рис. 8.77 пример. Он строит 12 расположенных по окружности кругов с разной степенью окраски (от белого до черного) с помощью функции **ArgShade**.

Заменяв в этом программном модуле функцию **ArgShade** на **ArgColor**, вы сможете наблюдать окраску кругов разными цветами.

Обычно цвета задаются в RGB (Red, Green, Blue) цветовой системе. При этом указывается относительный уровень интенсивности каждого цвета – в виде вещественных чисел с плавающей точкой в интервале от 0 до 1. В подпакете **Colors** заданы функции установки цвета, заданного в других известных цветовых системах:

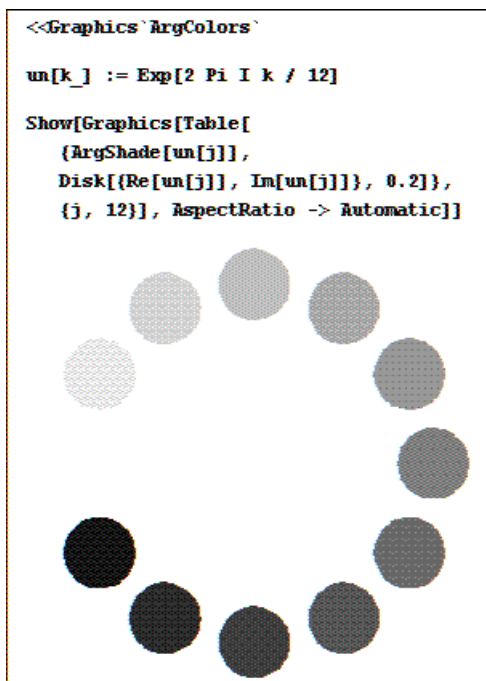


Рис. 8.77. Построение кругов, расположенных по окружности, с разной степенью окраски серыми полутонами

- **CMYColor[c,m,y]** – установка цвета по системе CMY (Cyan-Magenta-Yellow).
- **YIQColor[y,i,q]** – установка цвета по системе YIQ (NTSC телевизионная система).
- **HLSColor[h,l,s]** – установка цвета по системе HLS (Hue-Lightness-Saturation).
- **AllColors** – переменная-функция, выводящая список установленных цветов.

Примеры применения функций даны ниже:

```
<<Graphics`Colors`
YIQColor[0.5, -0.1, 0.2]
RGBColor[0.53,0.4,0.957]
Blue
RGBColor[0.,0.,1.]
Red
RGBColor[1.,0.,0.]
```

Кроме этого, в подпакете имеется внушительная таблица англоязычных наименований разных цветов и цветовых оттенков; она выводится функцией **AllColors**. Их можно использовать для задания в качестве аргумента у функций, управляющих цветами. Например, шоколадный цвет можно задать следующим образом:

```
Chocolate
RGBColor[0.823496,0.411802,0.117603]
```

Поскольку система Mathematica постоянно совершенствуется, то данные по компонентам цветов в разных версиях систем могут отличаться.

8.11.3. Построение стрелок

Подпакет Arrow служит для построения стрелок на двумерных графиках (или самих по себе). Для этого предназначена функция:

Arrow[start,finish,opts] – строит стрелку по координатам ее начала start и конца finish. Рекомендуется посмотреть список опций этой функции.

Рисунок 8.78 показывает построение двухсторонних стрелок, посаженных на иглу, стоящую на кресте, – своеобразная модель стрелок компаса.

Построение стрелок оживляет многие типы графиков. Их можно использовать, к примеру, для указания особых точек на графиках.

8.11.4. Задание картографических систем

Подпакет ComplexMap задает функции для построения графиков в наиболее важных **картографических** системах:

- **CartesianMap[f,{xmin,xmax},{ymin,ymax}]** – строит рисунок f в Картезианской системе координат.
- **PolarMap[f,{rmin,rmax},{thetamin,thetamax}]** – строит рисунок по данным f в полярной системе координат.

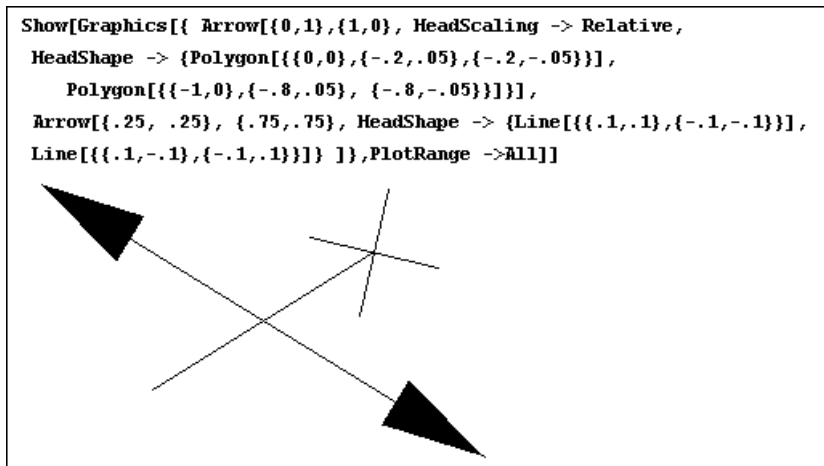


Рис. 8.78. Построение двухсторонних стрелок, посаженных на иглу

Действие этих довольно простых функций очевидно. Эти функции полезны тем пользователям, которые работают в области географии и картографии.

8.11.5. Построение объемных контурных графиков – *ContourPlot3D*

В подпакете *ContourPlot3D* заданы две функции, которые строят контурные объемные графики. Напоминаем, что функции ядра *ContourPlot* и *ListContourPlot* строят графики этого типа только двумерные. Для построения **объемных контурных графиков** необходимо использовать следующие функции:

- **ContourPlot3D**[*f*, {*x*, *xmin*, *xmax*}, {*y*, *ymin*, *ymax*}, {*z*, *zmin*, *zmax*}] – строит трехмерный контурный график функции *f* трех переменных *x*, *y* и *z*.
- **ListContourPlot3D**[*f*, {*f*₁₁₁, *f*₁₁₂, ...}, {*f*₁₂₁, *f*₁₂₂, ...}, ..., ...}] – строит контурный график по данным трехмерного массива значений *f*_{xyz}.

На рис. 8.79 показано построение сферы с отверстием с помощью первой из этих функций.

Обратите внимание на то, что никаких усилий по созданию в сфере отверстия не требуется. Оно получено просто усечением ограничительного ящика, в котором размещается сфера. Для этого пределы по оси *y* заданы как {−1.2, 2}, тогда как по остальным осям {−2, 2}.

Интересные возможности открывает опция *Contours*, которая позволяет как бы раздвинуть в пространстве части трехмерной поверхности. Рисунок 8.80 демонстрирует ее действие.

Вторая функция – **ListContourPlot3D** позволяет строить ряд фигур или поверхностей в пространстве. Пример такого построения дан на рис. 8.81. Масшта-

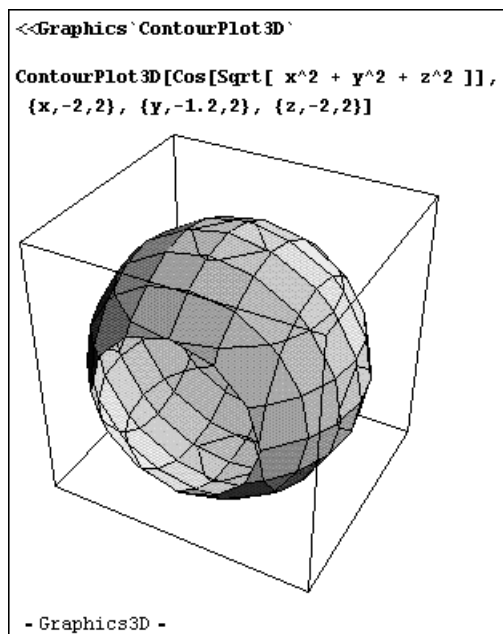


Рис. 8.79. Построение сферы с отверстием

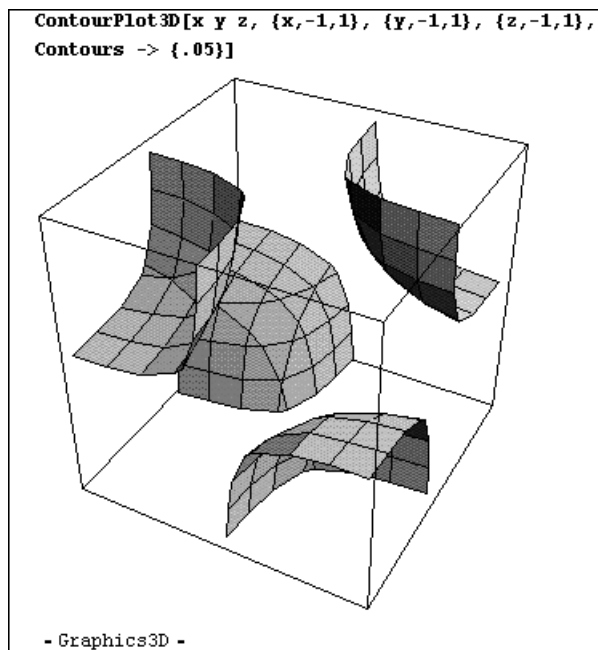


Рис. 8.80. Построение частей сферы в пространстве

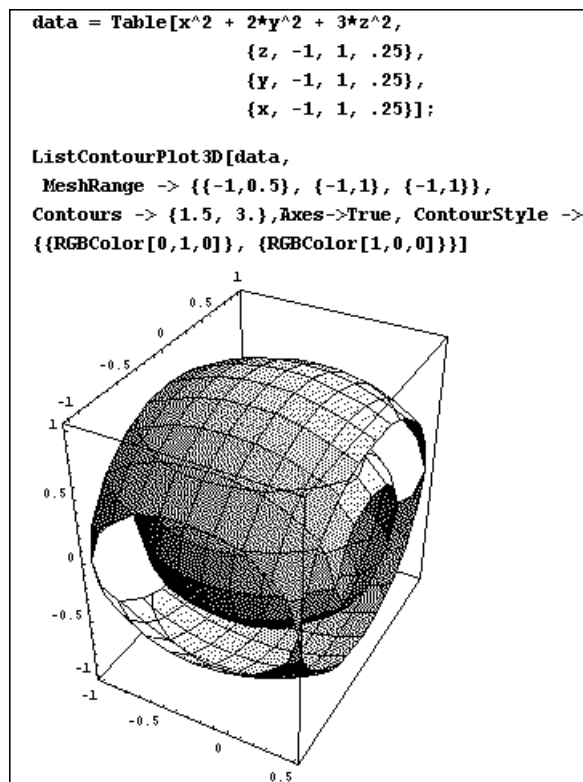


Рис. 8.81. Построение яйца, вложенного в параболы

бы осей подобраны так, чтобы справа фигура была несколько обрезана, что создаст изображение отверстия во внутренней яйцеобразной фигуре.

Как видно из этих примеров, применение описанных выше функций позволяет упростить построение трехмерных поверхностей и добиться интересных эффектов в их построении.

8.11.6. Построение графиков с окраской внутренних областей

Для построения графиков с окраской их замкнутых фрагментов в подпакете FilledPlot имеется ряд полезных функций. Подпакет загружается командой

```
<< Graphics`FilledPlot`
```

и имеет следующие функции:

- **FilledPlot[f,{x,xmin,xmax}]** – строит график функции $f(x)$ с окраской площадей, образованных линией функции и горизонтальной осью x . По умол-

чанию действуют опции `Fills->Automatic` и `Curves->Back` (т.е. кривые строятся на заднем плане, при значении `Front` построение фигур задано на переднем плане).

- **FilledPlot[{f1,f2,...},{x,xmin,xmax}]** – строит графики функций с выделением областей между ними разной окраской (цвет задается автоматически).
- **FilledListPlot[{y1,y2,...}]** – строит графики с окраской, меняющейся между кривыми $\{1,y_1\}$, $\{2,y_2\}$ и осью абсцисс x .
- **FilledListPlot[{x1,y1},{x2,y2},...]** – строит графики ряда кривых с окраской, заданной $\{x_i,y_i\}$ и осью абсцисс x .
- **FilledListPlot[data1, data2,...]** – строит графики ряда кривых с закраской областей, специфицированных данными `datai`.

Пример применения функции `FieldPlot` для различной закраски областей между тремя кривыми и координатными осями дан на рис. 8.82.

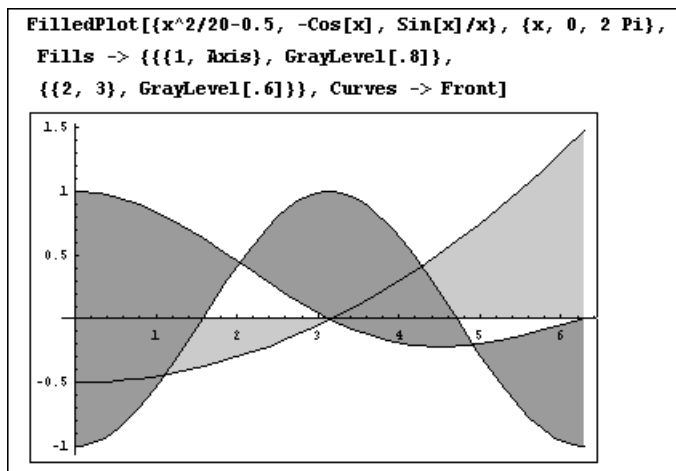


Рис. 8.82. Пример использования функций закраски

Иногда важное значение может иметь опция **AxesFront->Значение**. При значении этой опции **False** область закраски закрывает соответствующую часть осей, а при значении **True** оси выводятся поверх закраски. Сам по себе общий вывод осей задается опцией **Axes->True** – оси выводятся, и **Axes->False** – они вообще не выводятся. Рисунок 8.83 поясняет вывод осей и их построение поверх закрашенной области.

В данном случае область окраски ограничена треугольником, который строится как полигон. Если установить опцию **AxesFront->False**, то часть осей (внутри треугольника) будет не видна.

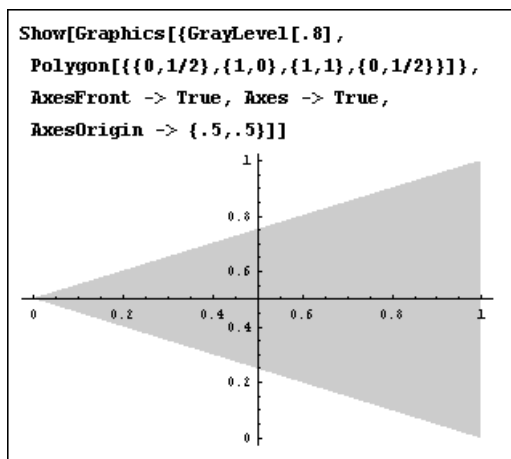


Рис. 8.83. Пример вывода осей поверх закрашенной области

8.11.7. Графики логарифмические и полулогарифмические

Подпакет Graphics задает ряд функций для построения специальных графиков, например, с логарифмическими и полулогарифмическими масштабами, с нанесенными на кривые точки, графики в виде гистограмм и т.д. Такие графики широко применяются для визуализации не только математических и физических, но и финансовых и экономических расчетов.

Для построения логарифмических и полулогарифмических графиков служат следующие функции:

- **LogPlot[f,{x,xmin,xmax}]** – строит линейно-логарифмический график $f(x)$ при изменении x от x_{\min} до x_{\max} .
- **LogLinearPlot[f,{x,xmin,xmax}]** – строит логарифмически-линейный график $f(x)$.
- **LogLogrPlot[f,{x,xmin,xmax}]** – строит логарифмический (по обеим осям) график $f(x)$.
- **LogListPlot[{{x1,y1},{x2,y2},...}]** – строит линейно-логарифмический график точек.
- **LogLinearListPlot[{{x1,y1},{x2,y2},...}]** – строит логарифмически-линейный график точек.
- **LogLogListPlot[{{x1,y1},{x2,y2},...}]** – строит логарифмический (по обеим осям) график точек.

Группа функций

LogListPlot[{y1,y2,...}]

LogLinearListPlot[{y1,y2,...}]

LogLogListPlot[{y1,y2,...}]

дает те же построения, что предшествующие три точки с той разницей, что ординаты абсцисс точек x равны 1,2,3,... и так далее. Это иногда упрощает задание графиков. Применение этих функций очевидно.

8.11.8. Графики в полярной системе координат

Для построения графиков функций в полярной системе координат заданы следующие функции:

- **PolarPlot**[$f, \{t, tmin, tmax\}$] – строит график функции в полярной системе координат как положение конца радиус-вектора f при изменении угла от $tmin$ до $tmax$.
- **PolarPlot**[$\{f1, f2, \dots\}$] – строит графики ряда функций $f1, f2, \dots$ в полярной системе координат.
- **PolarListPlot**[$\{r1, r2, \dots\}$] – строит графики ряда функций по их радиус-векторам.

Применение этих функций также очевидно. Отметим лишь, что они позволяют построить несколько графиков в одном окне.

8.11.9. Построение столбиковых и круговых диаграмм

В ряде случаев удобно представление данных в виде столбиковых и круговых диаграмм. Для построения *столбиковых диаграмм* служат функции, описанные ниже.

- **BarChar**[$datakist1, datalist2, \dots$] – строит столбиковую диаграмму по данным листов, располагая столбики рядом и обеспечивая их автоматическую закраску цветом.
Здесь любопытно отметить, что списки данных могут иметь разную длину. Число столбцов задается большим списком. Отсутствующие данные у списков меньшей длины считаются нулевыми, и для них столбцы не строятся. Данные, представленные отрицательными числами, строятся как столбцы, обращенные вниз.
- **StackedBarChar**[$datakist1, datalist2, \dots$] – строит столбиковую диаграмму, располагая столбцы одних данных над столбцами других данных с автоматическим выбором цветов для каждого набора данных.
- **PercentileBarChar**[$datakist1, datalist2, \dots$] – строит столбиковую диаграмму, отображающую нормированные данные в процентах.
- **GeneralizeBarChar**[$datakist1, datalist2, \dots$] – строит столбиковую диаграмму с заданной высотой и шириной столбцов.

Все указанные функции имеют опции, существенно влияющие на вид диаграмм. Рекомендуется просмотреть их с помощью функции **Options**. На рис. 8.84 показан пример построения столбиковых диаграмм. Наряду с цветовыми эффектами задается обвод пунктирной линией столбцов одного из комплектов данных и

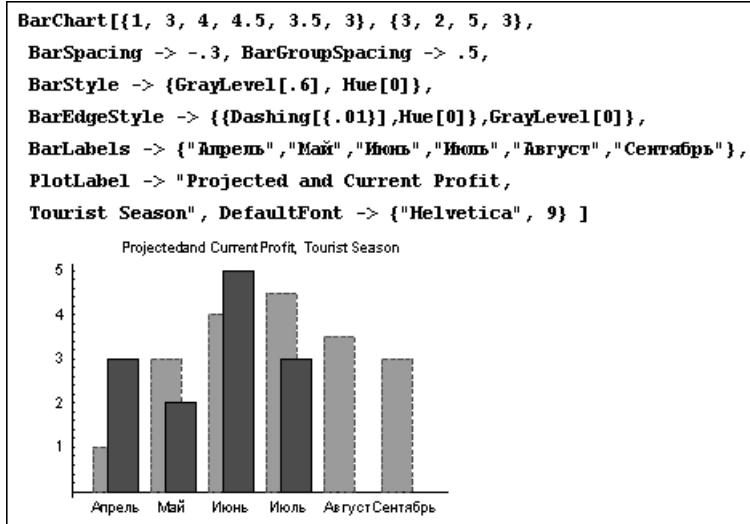


Рис. 8.84. Комплексное построение столбиковых диаграмм

главное – вывод надписей (названия месяцев) под наборами столбцов. Обратите внимание на то, что надписи могут быть на русском языке, несмотря на выбор набора шрифтов по умолчанию.

Круговые диаграммы наиболее удобны, когда оцениваются относительные величины, при этом сумма данных соответствует площади круга. Следующая функция позволяет строить такие диаграммы:

PieChart[data] – строит круговые диаграммы по данным data (рис. 8.85). Тип диаграммы задается опциями, список которых и значения по умолчанию можно

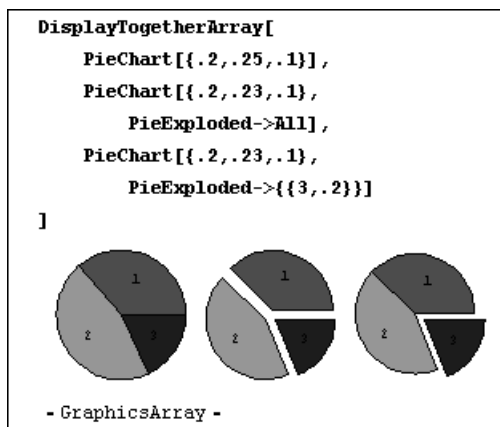


Рис. 8.85. Построение набора круговых диаграмм

получить командой **Options[PieChart]**. Одна из опций PieExploded позволяет отделить заданный сектор от диаграммы, что порою повышает наглядность представления данных.

8.11.10. Объединение графиков различного типа

В ноутбуке на рис. 8.85 использованы полезные в ряде случаев функции построения комбинированных графиков:

- **DisplayTogether[plot1,plot2,...,opts]** – строит комбинированный графический объект.
- **DisplayTogetherArray[plot1,plot2,...,opts]** – строит комбинированный графический объект.

Функция **DisplayTogether** позволяет объединять графики разного типа. На рис. 8.86 показано построение графика экспериментальных точек и линий параболической и кубической регрессий.

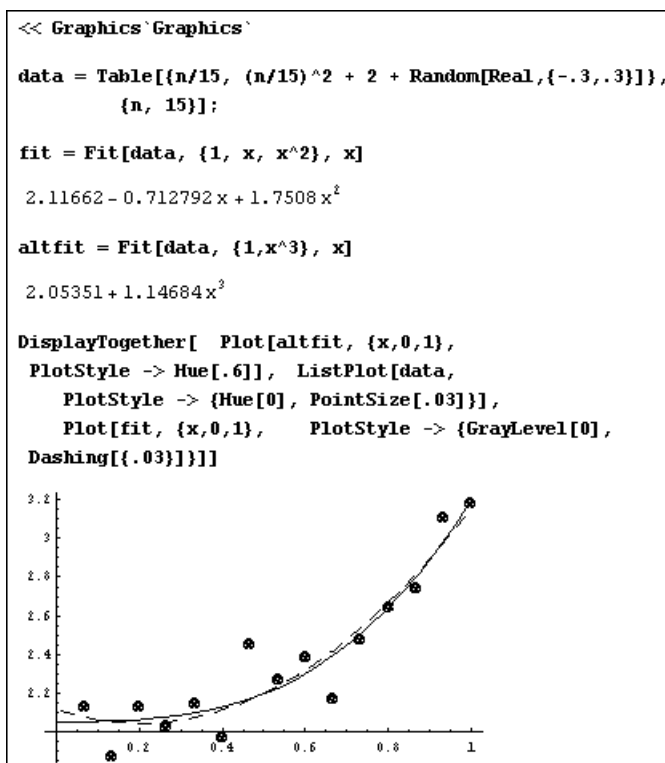


Рис. 8.86. Совместное построение исходных точек данных и линий параболической и кубической регрессий

Иногда нужно использовать в качестве точек графика цифры. Это реализует следующая функция, имеющая три формы:

- **TextListPlot**[[{y1,y2,...}]] – построение точек с ординатами y_i и абсциссами 1,2,... с представлением их числами 1,2,....
- **TextListPlot**[[{x1,y1},{x2,y2},...]] – построение точек с координатами $\{x_i,y_i\}$ и представлением их числами 1,2,....
- **TextListPlot**[[{x1,y1,expr1},{x2,y2,expr2},...]] – построение точек с координатами $\{x_i,y_i\}$ и представлением их числами, заданными выражениями $expr_i$.

Еще одна функция, также имеющая три формы

LabelListPlot[[y1,y2,...]] **LabelListPlot**[[{x1,y1},{x2,y2},...]]
LabelListPlot[[{x1,y1,expr1},{x2,y2,expr2},...]]

дает тот же результат, но дополнительно строит и сами точки.

Наконец, есть еще одна функция

ErrorListPlot[[{y1,d1},{y2,d2},...]] – построение графика точек y_i с зонами ошибок d_i .

Таким образом, подпакет Graphics обеспечивает построение наиболее распространенных типов графиков, используемых в научно-технической и финансово-экономической литературе.

Функция **MultipleListPlot** может использовать в списках указания на построение точки с зоной ошибок (ErrorBar).

8.11.11. Трехмерные столбиковые диаграммы

В подпакете Graphics3D, загружаемом командой

`<<Graphics`Graphics3D``,

имеется ряд программ для простого построения трехмерных графиков. Они описаны ниже с примерами.

- **BarChart3D**[[{z₁₁,z₁₂,...},{z₂₁,z₂₂,...},...]] – строит трехмерную столбиковую диаграмму по наборам данных высот столбцов z_{11}, z_{12}, \dots (рис. 8.87).
- **BarChart3D**[[{z₁₁,stile₁₁},{z₂₁,stile₂₁},...]] – строит трехмерную столбиковую диаграмму по наборам данных высот столбцов z_{11}, z_{12}, \dots с указанием спецификации стиля для каждого столбца.

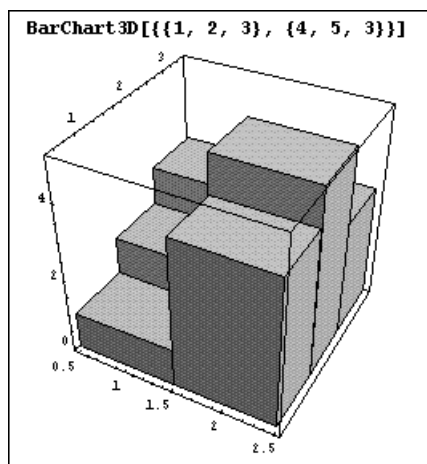


Рис. 8.87. Построение трехмерной столбиковой диаграммы

Нетрудно заметить, что функция **BarChart3D** автоматически задает стиль и цвет построения столбцов диаграммы. Эта функция имеет массу опций, с помощью которых можно менять вид диаграммы. Как обычно, перечень опций можно вывести с помощью команды **Options[BarChart3D]**.

8.11.12. Построение точек и кривых в пространстве

Иногда возникает необходимость в построении точек и кривых в пространстве. Для этого служит функция:

ScatterPlot3D[[$\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \dots\}$] – строит точки в пространстве по их заданным координатам.

При использовании опции **PlotJoined->True** точки соединяются отрезками прямых, и строится линия в пространстве. Обратите внимание, что список точек формируется с помощью функции **Table**. Это возможно, когда построение делается для аналитически заданной функции, описывающей 3D поверхность.

8.11.13. Построение графиков поверхности и ее проекций

Для построения трехмерного графика поверхности служат следующие функции:

- **ListSurfacePlot3D**[[$\{x_{11}, y_{11}, z_{11}\}, \{x_{12}, y_{12}, z_{13}\}, \dots\}$]] – строит поверхность по координатам ее точек.
- **ShadowPlot3D**[f, {x, xmin, xmax}, {y, ymin, ymax}] – строит график поверхности f(x,y) с ее проекцией на опорную плоскость.
- **ListShadowPlot3D**[[$\{x_{11}, y_{11}, z_{11}\}, \{x_{12}, y_{12}, z_{13}\}, \dots\}$]] – строит график поверхности z(x,y) с ее проекцией на опорную плоскость по координатам точек поверхности.

С помощью функции **Shadow[go]**, где go – графический объект, представляющий трехмерную фигуру, можно построить и более сложные рисунки, например, график объемной фигуры, и сразу всех трех ее проекций на взаимно перпендикулярные плоскости. Такое построение иллюстрируется документом, показанным на рис. 8.88.

С функцией **Shadow** можно использовать различные опции. Отметим наиболее существенные X, Y и Zshadow. Например, задав **Zshadow->False**, можно удалить одну из проекций, плоскость которой перпендикулярна оси z.

Для получения проекций на заданную плоскость, расположенную в пространстве, служат функции:

- **Project[g,pt]** – дает проекцию объекта g на диагональную плоскость, заданную списком из трех элементов pt. Например, список {1,1,0} даст проекцию на диагональную плоскость.
- **Project[g,{e1,e2},pt]** – дает проекцию объекта g в плоскости, определенной списком векторов {e1,e2} и списком pt.

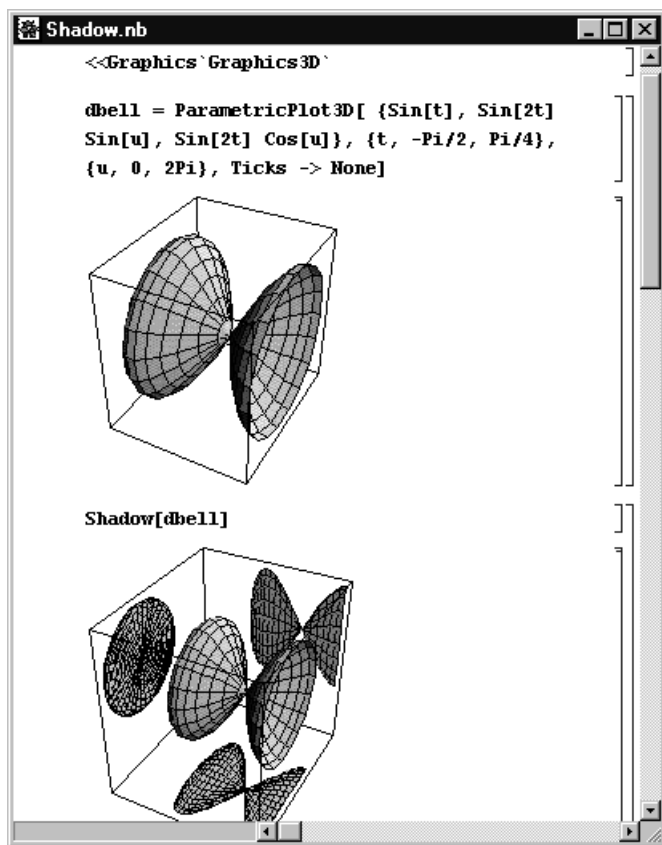


Рис. 8.88. Построение объемной фигуры и всех трех ее объектов

- **Project[g,{e1,e2},pt,origin]** – дает проекцию объекта g в плоскости, определенной списком векторов $\{e1,e2\}$ и списками pt и $origin$.

В конце подпакета определена следующая функция:

StackGraphics[{g1,g2,...}] – строит в пространстве графические объекты, располагая их каскадно.

8.11.14. Построение графиков неявных функций

Пакет **ImplicitPlot** задает три варианта функции для построения графиков неявно заданных функций:

- **ImplicitPlot[eqn,{x,xmin,xmax}]** – построение неявно заданной уравнением eqn функции при x , меняющемся от $xmin$ до $xmax$.

- **ImplicitPlot[eqn,{x,xmin,m1,m2,...,xmax}]** – построение неявно заданной уравнением eqn функции при x, меняющемся от xmin до xmax с исключением точек m1, m2,...
- **ImplicitPlot[{eqn1,eqn2,...},ranges,options]** – построение неявно заданных уравнениями eqn функции при x, меняющемся в пределах ranges и при задании опций options.

Примером может быть функция $x^2+k*y^2=r^2$, задающая построение эллипса.

8.11.15. Вывод обозначений кривых – легенд

Наглядность графиков, особенно имеющих несколько кривых, повышается при выводе обозначений кривых, так называемой легенды. В подпакете Legend для этого заданы следующие средства:

- **PlotLegend->{text1,text2,...}** – опция для функции **Plot**, устанавливающая легенду в виде последовательных текстовых надписей.
- **ShowLegend[graphic,legend1,legend2,...]** – устанавливает легенду в имеющийся график graphic.
- **{{{box1,text1},...},opts}** – спецификация легенды с цветными примитивами или графиками для рамок boxi с текстами texti.
- **{colorfunction,n,ninstring,maxstring,opts}** – спецификация легенды с рамками и указанием цветовой спецификации с помощью строк, размещаемых в рамках.

Обратите внимание на то, что среди многочисленных опций функции **Plot** имеется ряд, относящихся к параметрам легенды: **LegendPosition->{-1,1}** – установка позиции легенды, **LegendSize->Automatic** – установка размера легенды, **LegendShadow->Automatic** – установка, **LegendOrientation->Vertical** – ориентация рамки легенды, **LegendLabel->None** – отметка легенды и **LegendTextDirection->Automatic** – направление текста. С помощью этих опций можно существенно влиять на вид легенды.

На рис. 8.89 показано построение графиков двух функций с легендой и с применением различных опций.

В заключении отметим еще две функции подпакета Legend:

- **Legend[legendargs,opts]** – создает графический примитив для задания индивидуальной легенды.
- **ShadowBox[pos,size,opts]** – создает графический примитив в виде рамки для легенды.

К примеру, функция

ShadowBox[{0, 0}, {1, 1},ShadowBackground->GrayLevel[.8]]

создает графический примитив в виде пустого ящика для легенды с тенью. Для его просмотра можно использовать команду:

Show[Graphics[%]]

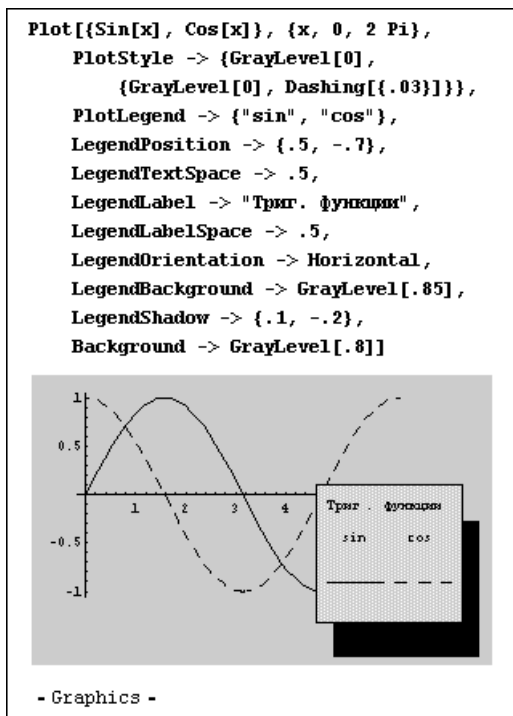


Рис. 8.89. Пример построения графика двух функций с легендой и установкой ряда ее опций

Применение функции **Graphics** здесь связано с тем, что **ShadowBox** порождает графический примитив, а не законченный графический объект.

8.11.16. Построение графиков с примитивами

Следующие функции служат для вывода в качестве точек символов:

- **PlotSymbol[type]** – задает тип символа (type: Box, Diamond, Star или Triangle). Возможно применение опции **Filled->False**.
- **PlotSymbol[type,size]** – задает тип символа и его размер size.
- **MakeSymbol[ptimitives]** – задает вывод символа, создаваемого графическим примитивом.

Для создания примитивов в виде регулярных многоугольников (полигонов) задана директива:

- **RegularPolygon[n]** – регулярный n-угольный полигон.
- **RegularPolygon[n,rad]** – регулярный n-угольный полигон с заданным радиусом описанной окружности rad.
- **RegularPolygon[n,rad,ctr]** – то же с заданным центром ctr.

- **RegularPolygon[n,rad,ctr,tilt]** – то же с углом поворота фигуры на tilt градусов.
- **RegularPolygon[n,rad,ctr,titl,k]** – соединение линиями через k вершин.

8.11.17. Построение трехмерных заданных параметрически графиков

Трехмерные графики параметрически заданных функций относятся к числу наиболее сложных, но в то же время весьма эффектных. В подпакете ParametricPlot3D определены функции, упрощающие подготовку таких графиков:

- **ParametricPlot3D[{fx,fy,fz},{u,u0,u1,du},{v,c0,v1,dv}]** – строит 3D поверхность, заданную параметрически функциями fx, fy и fz от переменных u и v с заданными диапазонами изменения и приращениями du и dv.
- **PointParametricPlot3D[{fx,fy,fz},{u,u0,u1,du}]** – строит точками 3D поверхность, заданную параметрически функциями fx, fy и fz от одной переменной u с заданным диапазоном изменения и приращением du.
- **PointParametricPlot3D[{fx,fy,fz},{u,u0,u1,du},{v,c0,v1,dv}]** – строит точками 3D-поверхность, заданную параметрически функциями fx, fy и fz от переменных u и v с заданными диапазонами изменения и приращениями du и dv.

8.11.18. Трехмерные графики в сферической и цилиндрической системах координат

Для построения трехмерных поверхностей в сферической и в цилиндрической системах координат служат функции:

- **SphericalPlot3D[r,{t,tmin,tmax},{p,pmin,pmax}]** – построение графика в сферической системе координат.
- **CylindricalPlot3D[z,{t,tmin,tmax},{p,pmin,pmax}]** – построение графика в цилиндрической системе координат.

С помощью опции ViewPoint можно изменять положение точки, с которой рассматривается фигура.

8.11.19. Построение графиков полей

В подпакете PlotField имеются функции, позволяющие строить стрелками графики полей на плоскости:

- **PlotVectorField[{fx,fy},{x,xmin,xmax},{y,ymin,ymax}]** – строит плоскость из векторов (стрелок), ограниченную пределами изменения x и y.
- **PlotGradientField[f,{x,xmin,xmax},{y,ymin,ymax}]** – строит плоскость из векторов (стрелок) градиента, ограниченной пределами изменения x и y.

- **PlotHamiltonianField**[f,{x,xmin,xmax},{y,ymin,ymax}] – строит плоскость полей Гамильтона, ограниченную пределами изменения x и y.
- **PlotPolyFueled**[f,{x,xmin,xmax},{y,ymin,ymax}] – представляет график комплексной функции f(x,y).

Указанные функции имеют множество опций. Отметим основные из них:

- **ScaleFactor->Automatic** – устанавливает размер векторов (стрелок);
- **ScaleFunction->None** – устанавливает функцию, вычисляющую размер стрелок;
- **MaxArrowLength->None** – устанавливает ограничение по длине стрелок;
- **ColorFunction->None** – задает функцию цвета;
- **PlotPoints->15** – задает число точек по координатам для построения стрелок.

Применение опций позволяет строить самые разнообразные графики различных полей – тепловых, гравитационных, электрических и других.

Есть еще одна функция, представляемая в двух формах:

- **ListPlotVectorField**[{{vect11,vect12,...},{vect21,vect22,...},...}] – строит график векторного поля прямоугольного массива векторов (**vect**)_{xy}.
- **ListPlotVectorField**[{{pt1,vect1,...},{pt2,vect2,...},...}] – строит график векторного поля прямоугольного массива векторов (**vect**)_{xy}.

Для представления векторных полей в пространстве служат функции подпакета **PlotField3D**:

- **PlotVectorField3D**[{fx,fy,fz},{x,xmin,xmax},{y,ymin,ymax},{z,zmin,zmax}] – строит график векторного поля параметрически заданной 3D фигуры.
- **PlotGradientField3D**[{fx,fy,fz},{x,xmin,xmax},{y,ymin,ymax},{z,zmin,zmax}] – строит график градиента векторного поля параметрически заданной 3D фигуры.

Обычно векторное поле строится отрезками прямых, а не стрелками. Последнее связано с тем, что по умолчанию задана опция **VectorHeads->False**. Изменив ее на **VectorHeads->True**, можно получить представление векторного поля направленными стрелками. Кроме того, используя опцию **PlotPoints->n**, можно получить заданное число стрелок n по всем направлениям графика. Все это учтено на графике, представленном на рис. 8.90.

Имеется еще одна функция:

ListPlotVectorField3D[{{vect₁,pt₁,...},{vect₂,pt₂,...},...}] – строит график векторного поля в пространстве по данным векторов **vect_i**.

К сожалению, при большом числе векторов в пространстве графики этого типа теряют наглядность. Рекомендуется тщательно отлаживать их, используя весь набор опций (как его получить, описывалось неоднократно).

8.11.20. Построение пространственных фигур стереометрии

Подпакет **Polyhedra** служит для создания регулярных пространственных фигур – полиэдров. Они задаются как графические примитивы и выводятся функцией:

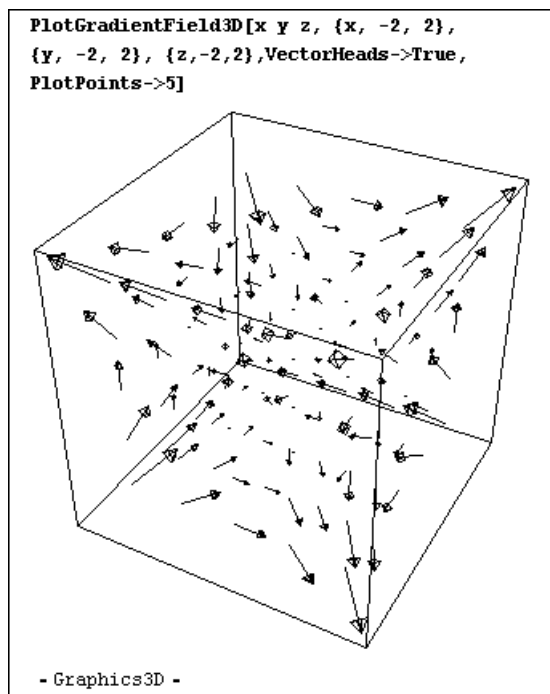


Рис. 8.90. Пример построения графика градиента поля направленными стрелками

- **Show[Polyhedron[polyname]]** – строит полиэдр с именем polyname в центре графика.
- **Show[Polyhedron[polyname,{x,y,z},scale]]** – строит полиэдр с именем polyname с центром в точке {x,y,z} и параметром масштаба scale.

Возможно задание следующих имен полиэдров: Tetrahedron, Cube, Octahedron, Dodecahedron, Icosahedron, Hexahedron, GreatDodecahedron, SmallStellatedDodecahedron, GreateStellatedDodecahedron и GreatIcosahedron. Пример построения двух полиэдров показан на рис. 8.91.

Для вывода полиэдров служит также ряд описанных ниже функций. Так, для построения звездообразных полиэдров предназначена функция:

- **Show[Stellate[Polyhedron[polyname]]** – построение звездообразных полиэдров.
- **Show[Stellate[Polyhedron[polyname],ratio]]** – построение звездообразных полиэдров с заданным отношением описанной и вписанной сфер ratio.

Полиэдры, применяемые в геодезии, можно получить с помощью следующих функций:

- **Show[Geodesate[Polyhedron[polyname],n]]** – построение полиэдров, состоящих из регулярных n-угольных многоугольников, образующих сферу.

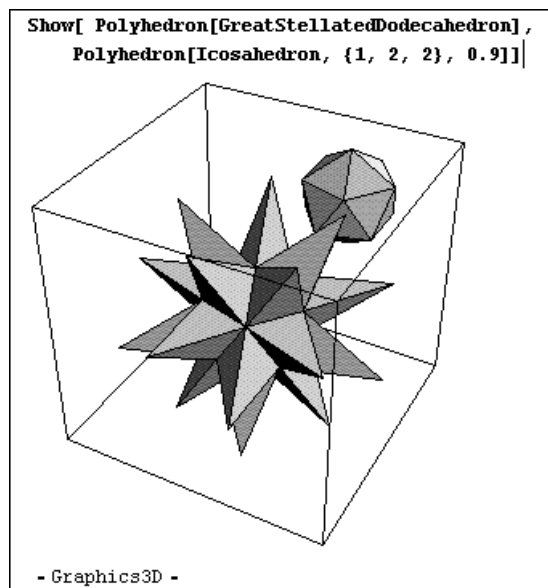


Рис. 8.91. Вывод функцией Show двух полиэдров

- **Show[Geodesate[Polyhedron[polynome], n,{x,y,z},radius]** – построение полиэдров, состоящих из регулярных n-угольных многоугольников, образующих сферу с заданным положением центра {x,y,z} и радиуса radius.

Для построения усеченных полиэдров предназначены функции:

- **Show[Truncate[Polyhedron[polynome]]]** – построение усеченных полиэдров.
- **Show[Truncate[Polyhedron[polynome], ratio]** – построение усеченных полиэдров с заданным радиусом.
- **Show[OpenTruncate[Polyhedron[polynome]]]** – построение полиэдров с открытым усечением.
- **Show[OpenTruncate[Polyhedron[polynome], ratio]** – построение полиэдров с открытым усечением и заданным отношением ratio (до 0.5).

В заключение этого раздела отметим следующие функции:

- **First[Polyhedron[polynome]]** – возвращает список полигонов для указанного полиэдра.
- **Vertices[polynome]** – возвращает список координат вершин полиэдра.
- **Faces[polynome]** – возвращает список вершин, ассоциированных с каждой поверхностью.

Они ничего не строят, а лишь возвращают специфические параметры полиэдров. Приведенные выше функции можно использовать на занятиях стереометрии, где полученные с их помощью фигуры могут прекрасно иллюстрировать теоретические положения курса и заменить сделанные из пресс-папье неказистые наглядные пособия.

8.11.21. Создание графических форм

Нередко желательно придать трехмерным объектам определенную форму, например кольца или бублика. Некоторые возможности для этого дают функции подпакета Shapes. Основной из них является функция

Show[Graphics3D[shape]] – отображение формы со спецификацией shape.

С ней могут использоваться графические примитивы:

- **Cone[r,h,n]** – конус с основанием радиуса r и высотой h на основе n -стороннего полигона в основании.
- **Cylinder[r,h,n]** – цилиндр радиуса r и высотой h на основе n -стороннего полигона.
- **Torus[r1,r2,n,m]** – объемное кольцо с внешним и внутренним радиусами r_1 и r_2 и числом сторон каркаса n и m .
- **Sphere[r,n,m]** – сфера радиуса r , составленная из многоугольников с параметрами n и m и числом сторон $n(m-2)+2$.
- **MoebiusStrip[r1,r2,n]** – кольцо Мебиуса с радиусами r_1 и r_2 , построенное на основе полигона с $2n$ сторонами.
- **Helix[r,h,m,n]** – плоская спираль радиуса r и высоты h на основе поверхности, разбитой на n и m четырехугольников.
- **DoubleHelix[r,h,m,n]** – плоская двойная спираль радиуса r и высоты h на основе поверхности, разбитой на n и m четырехугольников.

Возможно указание фигур без параметров. Это означает, что они выбираются по умолчанию следующими:

Cone[1, 1, 20]	Cylinder[1, 1, 20]	Helix[1, 0.5, 2, 20]
DoubleHelix[1, 0.5, 2, 20]	MoebiusStrip[1, 0.5, 20]	Sphere[1, 20, 15]
Torus[1, 0.5, 20, 10]		

Имеются еще и следующие функции:

RotateShape[g,phi,theta,psi] – поворот графического объекта на углы ϕ , θ и ψ .

TranslateShape[g,{x,y,z}] – преобразование графического объекта в представляющий его вектор.

AffineShape[g,{scale1,scale2,scale3}] – умножение всех координат объекта g на указанные множители.

8.11.22. Построение фигур, пересекающихся в пространстве

Функции **Show** и **Graphics** позволяют строить трехмерные фигуры, которые пересекаются в пространстве. Нетрудно заметить, что линии пересечения строятся с точностью до одной ячейки-полигона. Поэтому для получения качественных фигур необходимо увеличивать число полигонов, из которых фигуры синтезируются. Это, однако, увеличивает время построения фигур.

8.11.23. Применение сплайнов

Подпакет Spline обеспечивает представление данных с помощью сплайна. В подпакете Spline определена единственная функция:

Spline[points,type] – создает графический примитив, представляющий сплайн-кривую типа type (Cubic, Bezier или CompositeBezier – см. описание подпакета NumericalMath`SplineFit`).

Среди ее опций важно отметить следующие (значения, как и ранее, даны по умолчанию): **SplineDots->None**, **SplinePoints->25**, **MaxBend->10.0** и **SplineDivision->20.0**.

Рисунок 8.92 показывает задание массива из 5 точек на плоскости и соединение их отрезками прямых и кубическими сплайн-функциями. Хорошо видна аналогия сплайна с гибкой линейкой.

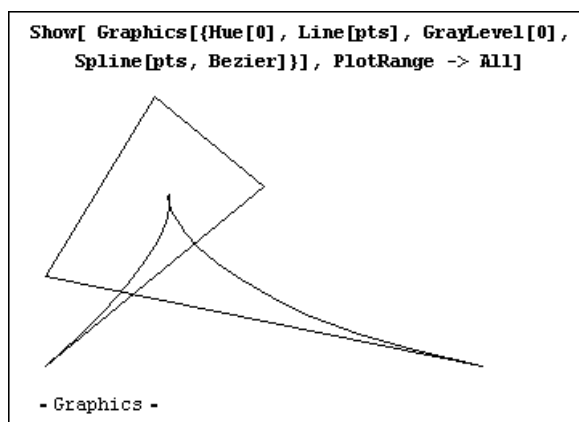


Рис. 8.92. Пример интерполяции пяти точек отрезками прямой и сплайнами

Сплайны функции в данном случае применяются в порядке задания точек в списке pts. В этом случае возможно создание замкнутых линий (рис. 8.93 является наглядным примером этого).

Следует отметить, что хотя сплайн-аппроксимация дает хорошие результаты при умеренном числе точек, при малом их числе и неудачном выборе типа сплайнов результат может оказаться неудовлетворительным.

8.11.24. Функции построения фигур вращения

Одна из задач компьютерной графики – создание поверхностей вращения. Средства для этого дает подпакет SurfaceOfRevolution. Они представлены следующими функциями:

- **SurfaceOfRevolution[f,{x,xmin,xmax}]** – строит поверхность, образованную вращением кривой, описанной функцией f при изменении x от $xmin$ и $xmax$, в плоскости x - z .
- **SurfaceOfRevolution[{f_x,f_y},{t,tmin,tmax}]** – строит поверхность, образованную вращением кривой, описываемой параметрически заданной на плоскости функцией $\{f_x, f_y\}$, в плоскости x - z при изменении параметра t от $tmin$ и $tmax$.
- **SurfaceOfRevolution[{f_x,f_y,f_z},{t,tmin,tmax}]** – строит поверхность, образованную вращением кривой, описываемой параметрически заданной в пространстве функцией $\{f_x, f_y, f_z\}$, в плоскости x - z между $xmin$ и $xmax$.
- **SurfaceOfRevolution[f,{x,xmin,xmax},{theta,thetamin,thetamax}]** – строит поверхность вращения кривой, описываемой функцией f , при угле $theta$, меняющемся от $thetamin$ и $thetamax$.

Рисунок 8.93 демонстрирует возможность построения объемной фигуры с вырезами. Все, что для этого необходимо, – удачно выбрать диапазон изменения угла вращения. Если он будет от 0 до 2π , то фигура будет сплошной и не содержать вырезов.

Для поворота фигуры вращения служат опции:

- **RevolutionAxes->{x,y}** – задает поворот в плоскости x - z .
- **RevolutionAxes->{x,y,z}** – задает поворот в пространстве.

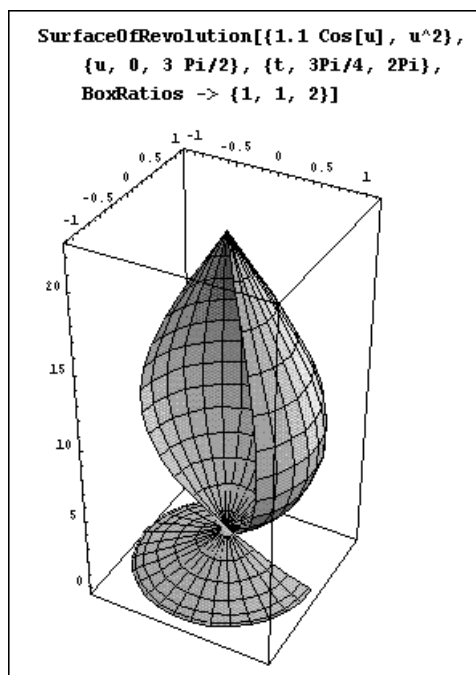


Рис. 8.93. Построение яйца с вырезом

Следующая функция позволяет построить фигуру вращения, образующая линия которой задается массивом точек:

- **ListSurfaceOfRevolution[{point1,point2,...}]** – создает поверхность вращения, заданную массивом точек point1, point2,....
- **ListSurfaceOfRevolution[{point1,point2,...},{theta,thetamin,thetamax}]** – создает поверхность вращения, заданную массивом точек и углом вращения theta от thetamin до thetamax.

8.12. Идеология применения пакета Graphics в Mathematica 6

8.12.1. Роль пакета Graphics в Mathematica 6

Пакет расширения Graphics в Mathematica 6 отнесен к классу наследственных пакетов, т.е. поневоле доставшихся в наследство от предшествующих версий Mathematica 5.1/5.2. Поэтому его применение нежелательно, хотя и возможно. Большая часть функций этого пакета перешла в ядро системы Mathematica 6 и существенно модернизирована. Для применения таких функций загрузка пакета Graphics уже не нужна.

Такие функции Mathematica 6, как **Animate**, **ListContourPlot**, **ListLogPlot**, **ListPolarPlot**, **ListSurfacePlot3D**, **GraphPlot** и др. подобны уже описанным в этой главе. Рекомендуется внимательно изучить их многочисленные опции, порою открывающие интересные возможности этих графических функций.

8.12.2. Представление точек графиков произвольными объектами

В практике представления научно-технических расчетов большое значение имеет наглядное представление точек графиков. Большие возможности тут открывает опция **PlotMarker**. Например, если задать **PlotMarker->Automatic**, то точки линий графика будут автоматически помечаться простыми геометрическими фигурами – кружками, квадратиками, треугольниками и т.д. Задав **PlotMarker->{"a","b","c",...}**, можно строить графики буквами и т.д. На рис. 8.94 показан интересный пример, когда точки графика обозначены трехмерными фигурами.

Ниже мы рассмотрим несколько функций системы Mathematica, которые дают новые возможности.

8.12.3. Функция PolyhedronData

Функция **PolyhedronData** – это, по существу, база данных для построения трехмерных фигур – полиэдров. Возможно огромное число примеров и форм использования этой функции. Рассмотрим только самые важные из них.

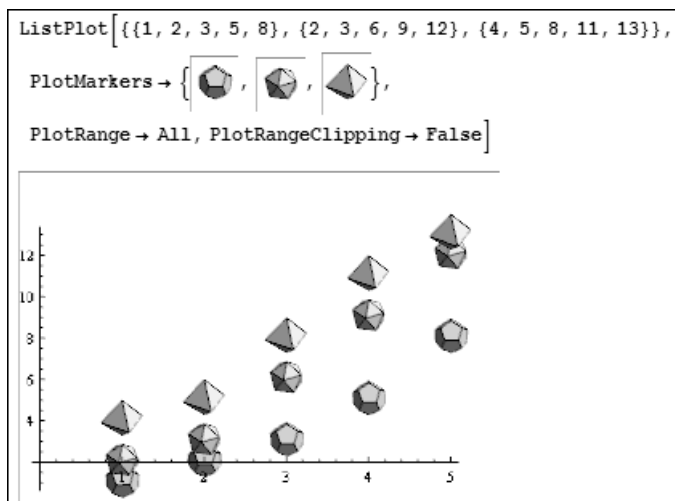


Рис. 8.94. Пример обозначения точек графиков
трехмерными фигурами

Следующая команда возвращает список классов полиэдров:

PolyhedronData["Classes"]

```

{Antiprism, Archimedean, ArchimedeanDual, Chiral, Concave, Convex,
Cuboid, Deltahedron, Dipyramid, Equilateral, Hypercube, Johnson,
KeplerPoinsot, Orthotope, Platonic, Prism, Pyramid, Quasiregular,
RectangularParallelepiped, Rhombohedron, Rigid, SelfDual, Shaky,
Simplex, SpaceFilling, Uniform, Zonohedron}

```

А так можно получить, к примеру, список возможных полиэдров класса Chiral:

PolyhedronData["Chiral"]

```

{GyroelongatedPentagonalBicupola, GyroelongatedPentagonalBirotonda,
GyroelongatedPentagonalCupolarotunda, GyroelongatedSquareBicupola,
GyroelongatedTriangularBicupola, PentagonalHexecontahedron,
PentagonalIcositetrahedron, SnubCube, SnubDodecahedron}

```

В справке можно найти десятки интересных примеров применения этой функции как для выявления свойств полиэдров, так и для построения их графических образов. Один из самых интересных и простых примеров представлен на рис. 8.96. Этот модуль на основе функции Manipulate строит окно с графическим интерфейсом, которое позволяет из списка задать любой тип фигуры *g* и вывести параметр из списка *p*.

Списки в окне на рис. 8.95 содержат полные наборы полиэдров и их параметров, так что окно дает полную информацию о них. Выводится изображение полиэдра и численные или символьные значения выбранного параметра.

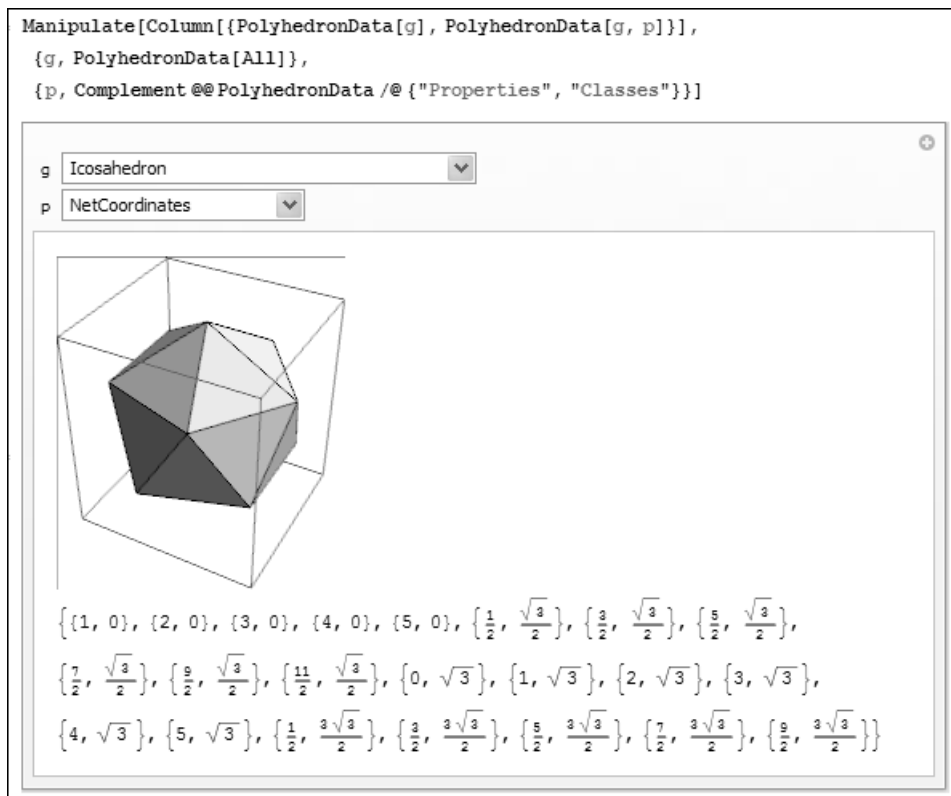


Рис. 8.95. Окно данных о полиэдрах и их параметрах

8.12.4. Функция **GraphData**

Функция **GraphData** дает доступ к базе данных по графам. Ее использование подобно применению функции **PolyhedronData**. На рис. 8.96 представлен модуль, подобный описанному выше, который позволяет по заданному типу графа *g* построить его график и вывести значение его заданного параметра *p*.

Модуль на рис. 8.96 дает возможность детального знакомства с графами различного типа и их параметрами. Такая возможность незаменима при серьезной работе с графами и их изучении, например, в образовательных целях. Много интересного о графах, их свойствах и применениях можно найти в книге [57], хотя она описывает графы в системе Maple. Там же можно найти детальный список литературы по этим объектам.

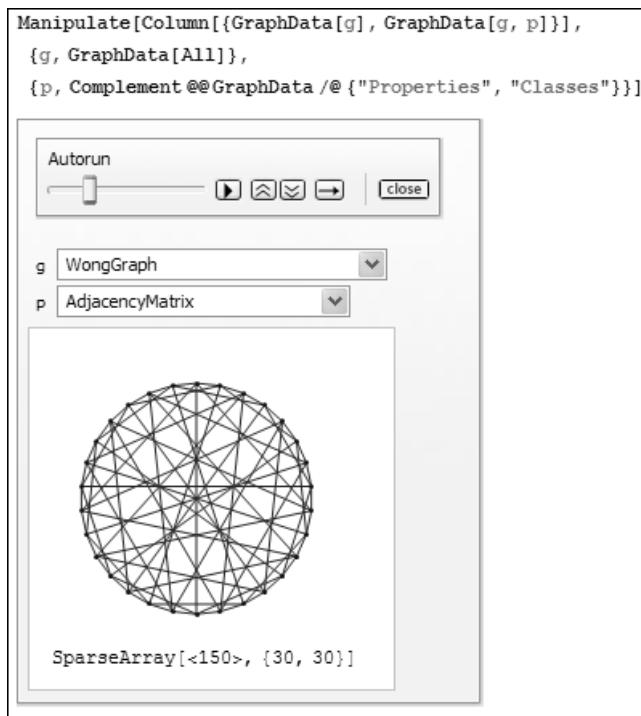


Рис. 8.96. Окно данных о графах и их параметрах

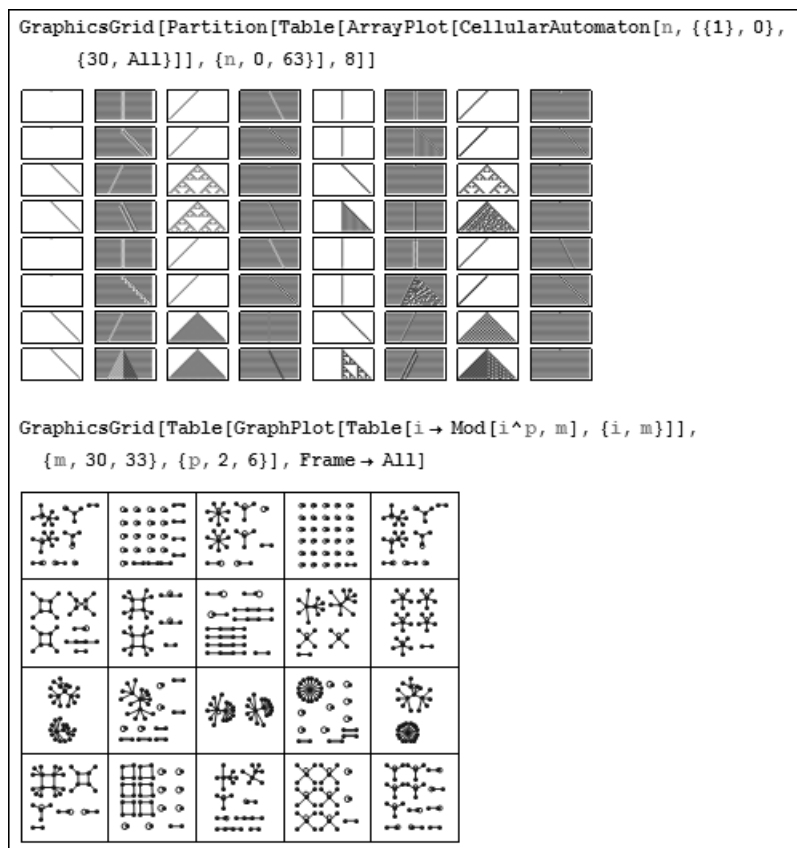
8.12.5. Функция **GraphicsGrid**

Из функций, позволяющих представить обширную информацию о графических средствах Mathematica 6, нельзя не отметить функцию

GraphicsGrid[{{ g_1, g_2, \dots }, ...}]

Эта функция генерирует ряд графических объектов двумерной графики. На рис. 8.97 показаны два наглядных примера применения данной функции: один строит 64 объекта, иллюстрирующих работу клеточных автоматов, а другой 20 графов. Оба примера дают наглядное представление о разнообразии и необычности этих графических средств.

В Mathematica 6 есть еще пара подобных функций: **GraphicsRow** – построение строки из ряда графиков и **GraphicsColumn** – построение столбца из ряда графиков.

Рис. 8.97. Примеры применения функции **GraphicsGrid**

8.12.6. Директива вставки *Inset*

Директива *Inset* обеспечивает вставку в графический объект другого объекта, например другого графика, текста и т.д. Самая полная форма директивы следующая:

Inset[*obj, pos, opos, size, dirs*]

Но она может быть сокращена до **Inset**[*obj*]. Примеры применения директивы представлены на рис. 8.98. В первом примере в закрашенный диск вставляется график функции одной переменной. Во втором примере в окружность вставляется надпись – формула, которая задает уравнение окружности.

Интересно отметить, что директива **Inset** интерактивная. Клик мышью выделяет основной объект, а двойной клик – вставленный объект. При этом выделенный объект можно перемещать по полю основного объекта и растягивать в том или ином направлении. Момент этого действия показан в нижнем примере: в нем надпись перемещена с центра окружности и увеличена путем растягивания по диагонали.

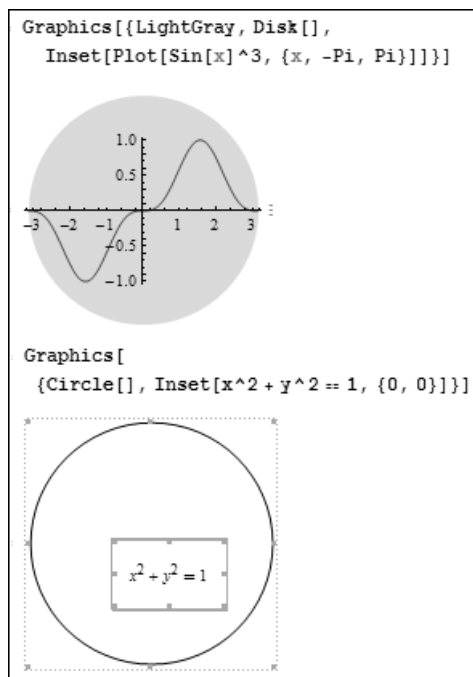


Рис. 8.98. Примеры вставки объектов
в графический объект

8.12.7. Директива непрозрачности *Opacity*

Из ряда полезных директив графики отметим директиву непрозрачности

Opacity[*a*] **Opacity[*a,color*]**

Она служит для установки степени непрозрачности *a* двумерных и трехмерных графических объектов. В ряде случаев это позволяет делать объекты полупрозрачными, и сквозь них наблюдать другие объекты. На рис. 8.99 показан пример применения этой директивы для построения случайно заданных 20 цилиндров.

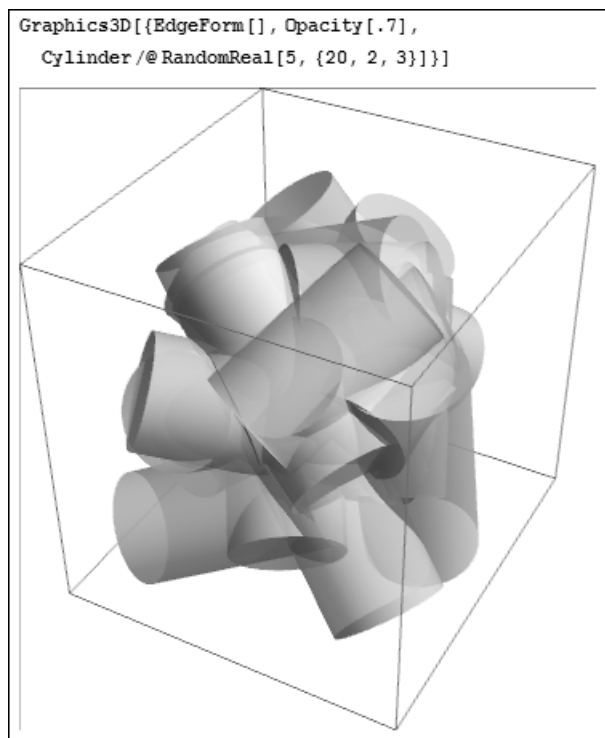


Рис. 8.99. Построение 20 случайных цилиндров с заданной степенью непрозрачности

Специальные средства программирования

9.1. Функциональное программирование специальной графики	496
9.2. Подготовка пакетов расширений системы Mathematica	499
9.3. Отладка и трассировка программ	505
9.4. Новые средства программирования в Mathematica 6	510
9.5. Обзор пакетов расширения Add-On	518
9.6. Данные о других средствах расширения	566

9.1. Функциональное программирование специальной графики

9.1.1. Пример программирования графической задачи

Графические задачи составляют значительную часть задач, решаемых системой Mathematica. С точки зрения программирования эти задачи не имеют особой специфики. Большая часть из них сводится к заданию функции, описывающей график, и применению одной из многочисленных графических функций системы с соответствующими опциями и директивами.

Примером такого подхода является задание функции **GrayCode** (Большие Коды) и ее графическое представление, полученное с помощью встроенной функции **ListPlot**. Это показано на рис. 9.1.

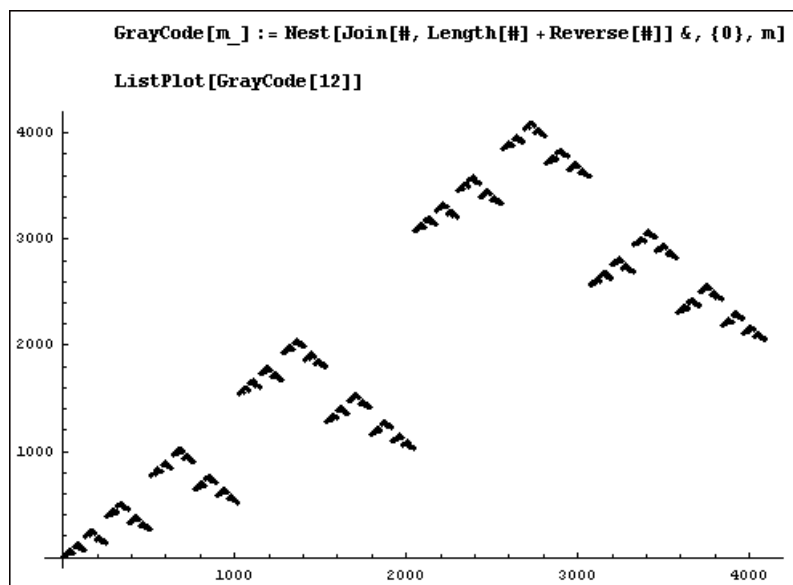


Рис. 9.1. Задание функции **GrayCode** и ее графическое представление на плоскости

9.1.2. Задание функции для построения фрактала Манделброта

В качестве следующего примера рассмотрим задачу на построение сложного графика фрактала Mandelbrot. Пример задания соответствующей функции **MandelbrotFunction** и применения графической функции **DensityPlot** для наглядного визуального представления функции **MandelbrotFunction** на комплексной плоскости представлен на рис. 9.2. Данная функция строит фрактал.

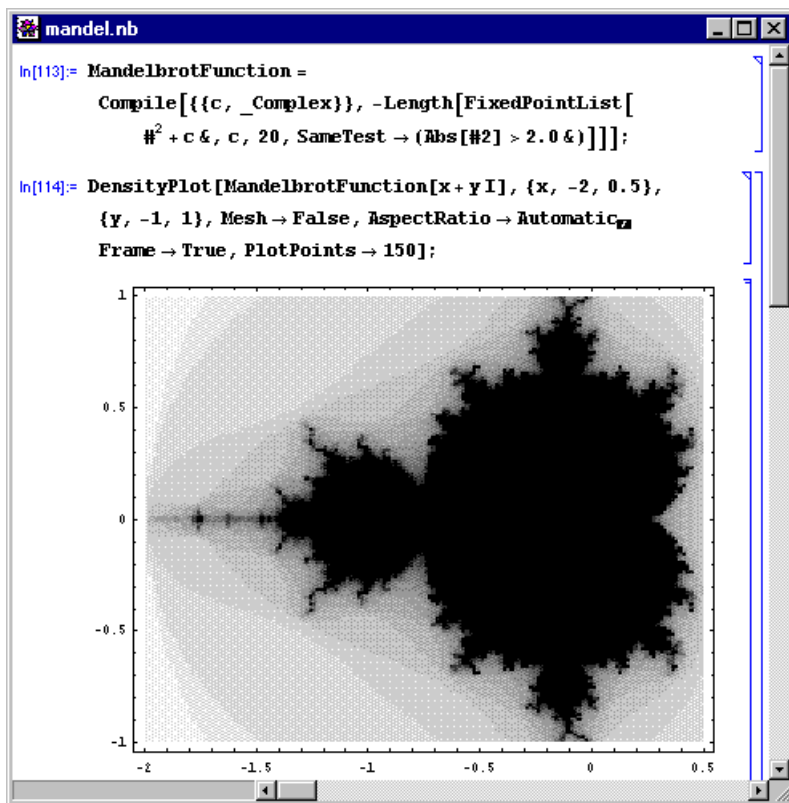


Рис.9.2. Пример задания функции **MandelbrotFunction** и построения ее контурного графика

9.1.3. Задание функции для построения модели деления клеток

Еще более сложную и любопытную задачу демонстрирует рис. 9.3. Здесь задана функция **JuliaFunction**, которая представляет одну из моделей деления клеток. На этом же рисунке дано построение множества графиков, дающих прекрасное визуальное представление данной функции.

Разумеется, приведенные примеры далеко не исчерпывают всего многообразия графических возможностей языка программирования систем Mathematica. Прекрасные примеры программирования графики можно найти в пакетах расширения AddOn систем Mathematica.

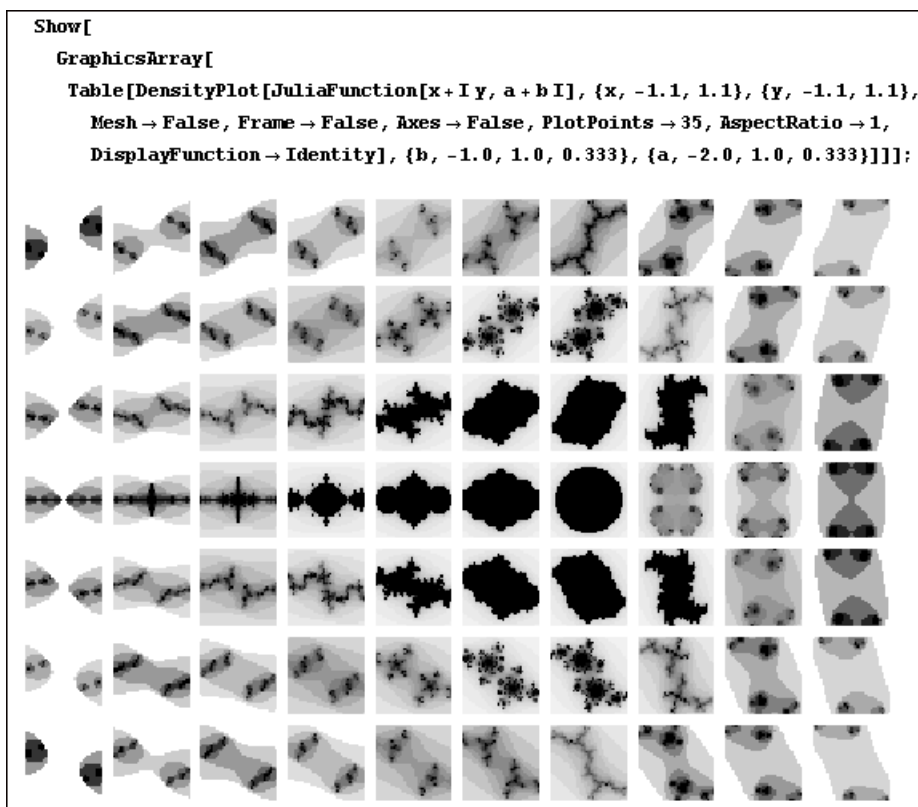


Рис. 9.3. Задание функции **JuliaFunction**
и ее графическое представление

9.2. Подготовка пакетов расширений системы Mathematica

9.2.1. Типовая структура пакетов расширения

Мощным средством расширения возможностей системы Mathematica является подготовка пакетов ее расширений. Пакеты расширений позволяют создавать новые процедуры и функции и хранить их на диске в виде файлов с расширением .m. После считывания такого пакета с диска все входящие в него определения функций становятся доступными для использования в соответствии с правилами, принятыми для встроенных функций. Текст пакета расширения не выводится после его вызова, чтобы не загромождать документ вспомогательными описаниями.

Структура пакета расширений в минимальном виде выглядит следующим образом:

```
(*Вводный комментарий*)
BeginPackage["Имя_пакета`"]
Mean::usage = "Имя функции[Параметры] Текстовый комментарий"
.....
Begin["`Private`"]
Unprotected[Список_имен]
Определения новых функций
End[ ]
Установка атрибутов защиты
+EndPackage[ ]
(* Завершающий комментарий *)
```

Особая структура пакетов расширений связана с реализацией описанной выше идеологии контекстов. Пакет открывается необязательным текстовым комментарием, который обрамляется двойными символами (* и *). Он может быть как однострочным, так и многострочным. Обычно вводный комментарий включает в себя имя пакета, наименование фирмы и автора – создателей пакета, историю развития, дату создания и т.д. Если вы программируете для себя, можете на первых порах опустить все эти комментарии. Но не забудьте их ввести после отладки пакета, как того требует культура и дисциплина программирования.

Затем пакет открывается словом **BeginPackage**. Это слово дается с квадратными скобками, в которых указывается контекст (см. выше) пакета. Обратите внимание на то, что после имени пакета должен стоять апостроф или цепочка символов, обрамленная апострофами. Имя пакета не должно совпадать ни с одним из известных, т.е. быть уникальным.

Эта команда изменяет контекстную дорожку, и она принимает вид {Имя_пакета`,System`}. Таким образом, на первом месте контекстной дорожки оказывается имя пакета, а на втором – контекст System`. Теперь любой вводимый и невстроенный символ приобретает контекстную приставку с именем данного пакета.

Обратите внимание на то, что контекст System` сохранился на новой контекстной дорожке, но вторым. Это значит, что если вы вводите слова и символы, встроенные

енные в систему, то они будут замещены новыми определениями. К примеру, если вы решили вычислять функцию $\text{Sin}[x]$ по новому и ценному для вас алгоритму, то ему будет отдаваться предпочтение при каждом использовании этой функции, до тех пор, пока вы работаете с данным пакетом расширения. Однако, как только вы перестаете работать с пакетом, восстановится роль встроенной функции $\text{Sin}[x]$.

Следующий блок пакета – сообщения о назначении функций. Эти сообщения выводятся, если после загрузки пакета задать вопросительный знак с последующим именем функции. Эти сообщения не обязательны, но они обеспечивают единство диалога с системой и, безусловно, нужны при профессиональной подготовке пакета. Обычно в этих сообщениях кратко указываются синтаксические правила использования функций и назначение их параметров, указываемых в квадратных скобках.

Затем следует главная часть пакета – определения новых функций. Она открывается определением **Begin["`Private`"]**. Оно, не меняя контекстную дорожку, устанавливает новый текущий контекст `Имя_пакета`Private``. Он присваивается всем ранее не встречавшимся символам. Имя `Private` принято в пакетах расширения системы *Mathematica*, хотя в принципе может быть любым другим именем. После него следуют сами определения, в которых могут использоваться любые средства, включенные в ядро системы.

В некоторых случаях имена функций могут повторять ранее определенные в ядре системы. Это полезно, если пользователь считает, что введенное им определение уже известной функции более точно или более универсально, чем использованное в системе. В таких случаях необходимо позаботиться о снятии с них защиты перед новым применением с помощью функций **Unprotected**. Именно эта часть и определяет существо пакета и его ценность.

Завершается эта часть определением **End[]**. При этом восстанавливается контекст, который был до определения **Begin["`Private`"]**, т.е. контекст с именем пакета. После этого идет необязательная часть с указанием атрибутов защиты. Пакет завершается определением **EndPackage[]**, которое восстанавливает бывший до загрузки пакета контекст (например `Global``), а контекст `Имя_пакета`` помещает в начало прежней контекстной дорожки.

Необязательный заключительный комментарий чаще всего дает список тестовых примеров. Он особенно желателен, если пакет содержит определения не вполне очевидных функций. Не забывайте, что этот комментарий не выводится и не исполняется, он нужен лишь на этапе знакомства с пакетом. Разумеется, такое знакомство необходимо при каждой серьезной попытке применения того или иного пакета расширения или применения системы.

В принципе, текстовые комментарии могут вводиться на русском языке. Однако при этом возникают определенные трудности. Такие пакеты не читаются вьюерами (программами просмотра файлов), ориентированными на просмотр текстов в стандарте ASCII. Да и при выводе их на экран дисплея при работе с оболочкой системы *Mathematica* также наблюдаются несоответствия между шрифтами, установленными при вводе комментариев и при их выводе. Поэтому лучше использовать комментарии на английском языке, тем более что коммента-

рии ко всем встроенным функциям и к поставляемым расширениям системы даны, естественно, на английском языке.

9.2.2. Средства создания пакетов расширений

Для создания пакетов расширений в общем случае используются следующие средства системы:

- **Begin["context"]** – устанавливает текущий контекст.
- **BeginPackage["context"]** – делает context единственным активным контекстом. Возможна также форма: **BeginPackage["context", {"need1", "need2", ...}]**.
- **CallProcess["command", f, {x1, x2, ...}]** – вызывает функцию f во внешнем процессе с аргументами xi. (CallProcess заменяется операциями MathLink).
- **Do[]** – возвращает Null или аргумент для первого Return, если происходила эволюция.
- **End[]** – возвращает текущий контекст и переходит к предыдущему.
- **EndAdd[]** – возвращает настоящий контекст и переходит к предыдущему, предварительно добавляя настоящий контекст к \$ContextPath.
- **EndPackage[]** – восстанавливает \$Context и \$ContextPath в их значениях до предшествующего BeginPackage и дополняет текущий контекст к списку \$ContextPath.
- **EndProcess["command"]** – завершает внешний процесс, в котором функции, возможно, вызывались из системы Mathematica. (EndProcess заменен командами MathLink).
- **Exit[]** – завершает сеанс работы Mathematica.
- **Goto[tag]** – просматривает текущее составное выражение в поиске Label[tag] и передает управление в эту точку.
- **Interrupt[]** – производит прерывание в теле расширения.
- **Label[tag]** – представляет точку в составном выражении, в которую передается управление директивой Goto.
- **Quit[]** – завершает сеанс работы системы Mathematica.
- **StartProcess["command"]** – запускает внешний процесс, в котором функции могут вызываться из системы Mathematica. StartProcess должен поддерживаться операциями MathLink.

Мы вернемся к рассмотрению построения пакетов расширений после более детального рассмотрения некоторых деталей этого процесса.

9.2.3. Текстовые сообщения и комментарии

Ценность многих программ на любом языке программирования нередко сводится к нулю из-за отсутствия подробных текстовых комментариев. Из-за этого даже сами разработчики программ через месяц-другой перестают понимать собствен-

ные творения. А что говорить о пользователях, рискующих применить такие программы?

Для создания текстовых комментариев различного назначения (как выводимых, так и не выводимых в ходе исполнения пакета) в языке программирования системы Mathematica 2 используются следующие средства:

- **(* Comment *)** – задание невыводимого текстового комментария в любом месте пакета, как однострочного, так и многострочного.
- **Message[symbol::tag]** – вывод сообщения symbol::tag, если только не выполнено отключение.
- **Message[symbol::tag, e1, e2, ...]** – выводит сообщение, вставляя значения e1 по мере необходимости.
- **\$MessageList** – глобальная переменная, возвращающая список имен сообщений, вырабатываемых во время вычисления текущей входной строки. Имя каждого сообщения заключено в HoldForm[]. \$MessageList сохраняется в MessageList[n] и переустанавливается в {} после того, как произведена n-ная выходная строка.
- **MessageList[n]** – глобальный объект, который является списком имен (сообщений), вырабатываемых в процессе обработки n-ной входной линии.
- **MessageName** – применяется в виде: **symbol::tag** или
- **MessageName[symbol, \"tag\"]** – имя для сообщения.
- **\$MessagePrePrint** – глобальная переменная, чье значение, если установлено, применяется к выражениям перед тем, как они помещаются в тексте сообщений.
- **\$Messages** – возвращает список файлов и каналов, в которые направляется вывод сообщений.
- **Messages[symbol]** – возвращает все сообщения, присвоенные данному символу symbol.

Следует отметить, что широкое применение комментариев обычно является признаком культуры программирования. Это особенно важно для математических систем, реализующих вычисления по сложным и подчас малопонятным для неспециалистов алгоритмам. Без подробных комментариев пакеты расширений и применений теряют свою практическую полезность и превращаются в ребусы – увы, куда менее интересные, чем те, которые публикуются в газетах и журналах.

9.2.4. Примеры подготовки пакетов расширений

Приведем пример простого пакета расширения, дающего определение новой функции **ExpandBoth** с помощью некоторых из представленных средств:

```
(* :Title: ExpandBoth *)
(* :Context: ProgrammingInMathematica`ExpandBoth` *)
(* :Author: Roman E. Maeder *)
ExpandBoth::usage = "ExpandBoth[e] expands all numerators and
denominators in e."
```

```

Begin["`Private`"]
ExpandBoth[x_Plus] := ExpandBoth /@ x
ExpandBoth[x_] := Expand[ Numerator[x] ] / Expand[ Denominator[x] ]
End[]
Null

```

Этот пример настолько прост, что читателю будет нетрудно разобраться с его сутью – расширением выражения по числителю и знаменателю. Ниже представлен сеанс работы с этим пакетом, файл которого `expboth.m` размещен в каталоге `myrpack`, включенном в общий каталог пакетов расширений:

```

<<myrpack/expboth.m
?ExpandBoth
ExpandBoth[e] expands all numerators and denominators in e.
ExpandBoth[124/12]
31
 3
ExpandBoth[1234/12]
617
 6

```

В файловой системе Mathematica можно найти множество уже подготовленных пакетов расширения. Поэтому ограничимся еще парой простых примеров. Первый из них содержит определение функции `AlgExpQ[expr]`, позволяющей выяснить, является ли выражение `expr` алгебраическим:

```

(* :Title: AlgExp *)
(* :Context: ProgrammingInMathematica`AlgExp` *)
BeginPackage["ProgrammingInMathematica`AlgExp`"]
AlgExpQ::usage = "AlgExpQ[expr] returns true if expr is an
algebraic expression."
Begin["`Private`"]
SetAttributes[AlgExpQ, Listable]
AlgExpQ[ _Integer ] = True
AlgExpQ[ _Rational ] = True
AlgExpQ[ c_Complex ] := AlgExpQ[Re[c]] && AlgExpQ[Im[c]]
AlgExpQ[ _Symbol ] = True
AlgExpQ[ a_ + b_ ] := AlgExpQ[a] && AlgExpQ[b]
AlgExpQ[ a_ * b_ ] := AlgExpQ[a] && AlgExpQ[b]
AlgExpQ[ a_ ^ b_Integer ] := AlgExpQ[a]
AlgExpQ[ a_ ^ b_Rational ] := AlgExpQ[a]
AlgExpQ[_] = False
End[]
EndPackage[]

```

Если выражение является алгебраическим, то функция `AlgExpQ` возвращает логическое значение `True`, иначе она возвращает значение `False`:

```

<<myrpack\algexp.m
?AlgExpQ
AlgExpQ[expr] returns true if expr is an algebraic expression.

```

```
AlgExpQ[a*x^2+b*x+c]
True
AlgExpQ[Sqrt[x]]
True
AlgExpQ[«x^2+1»]
False
AlgExpQ[1]
True
AlgExpQ[1.0]
False
```

Второй пример иллюстрирует переопределение графической функции в системе Mathematica 6, которая была в ее прежних версиях в составе пакета расширения Graphics:

```
(* ::Package:: *)
(*:Name: Graphics`ContourPlot3D` *)
(*:Mathematica Version: 6.0 *)
(*:Copyright: Copyright 1990-2007, Wolfram Research, Inc.*)
(*:Summary:
Three-dimensional contour plotting is now handled in the kernel.
The package Graphics`ContourPlot3D` is obsolete.
*)
Message[General::obspkg,"Graphics`ContourPlot3D`"];
BeginPackage["Graphics`ContourPlot3D`"]
EndPackage[]
```

На рис. 9.4 показан пример вызова функции **ContourPlot3D** из пакета **Graphics**. Затем с ее помощью построена поверхность с контурными линиями. В среде Mathematica 6 это ведет к сообщению о том, что такой вызов относится к устаревшим и в Mathematica 6 уже не нужен.

9.2.5. Подготовка пакетов применений

Пакеты применений – это группы документов, предназначенные для решения определенного класса математических или научно-технических проблем и задач. В отличие от пакетов расширения, в документах пакетов применений обычно дается подробно комментируемое описание всех основных алгоритмов решения задач. При этом комментарий обычно выводится на экран дисплея.

Довольно часто в пакетах применений используется описанный в главе 1 прием – объединение ряда ячеек в одну с общим текстовым заголовком. Это особенно полезно для организации вспомогательных и промежуточных вычислений, ячейки которых загромождают экран и делают текст документа мало наглядным. Данный прием скрывает такие вычисления, но позволяет в любой момент вывести их на экран дисплея при активизации маленького прямоугольника, отмечающего такие совмещенные ячейки. Тексты Notebooks, поставляемые с системой, являются прекрасными образцами использования этого приема.

Документы пакетов применения – это конечный продукт практического применения системы Mathematica. Поэтому они могут включать в себя все ранее опи-

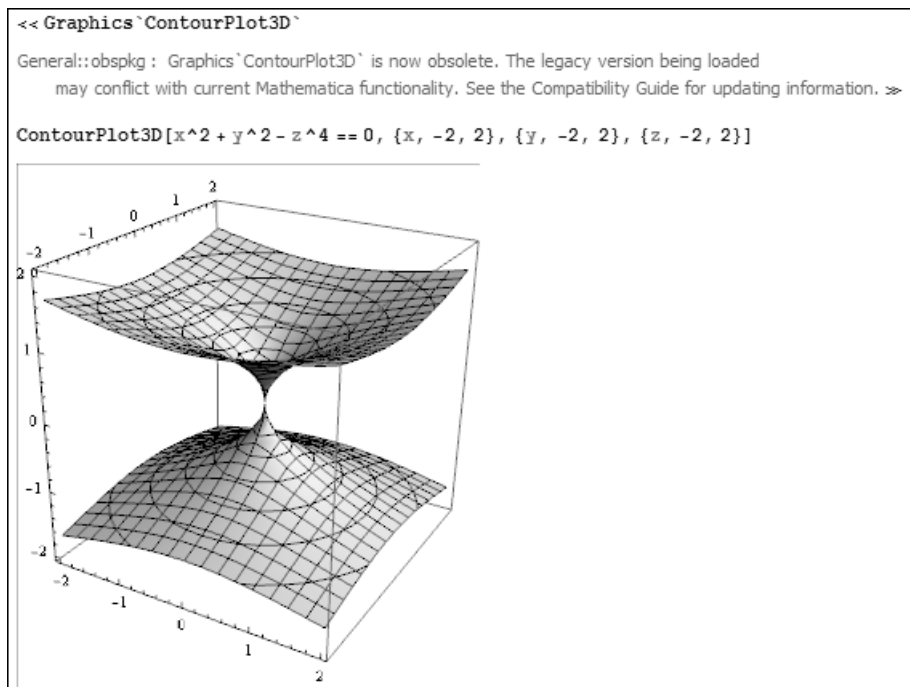


Рис. 9.4. Пример применения функции **ContourPlot3D**

санные средства системы. Как уже неоднократно отмечалось, документы записываются на диск в виде файлов с расширением .m (в ранних версиях Mathematica – .ma), а их полный битовый образ (включающий рисунки) сохраняется во вспомогательных файлах с расширением .mb. При большом числе сложных рисунков в документе эти файлы могут быть весьма большими – сотни Кбайтов и даже единицы Мбайтов.

9.3. Отладка и трассировка программ

Отладка программ, за исключением самых простых, – дело далеко не простое. Начальный опыт программирования на любом языке приходит спустя годы практической работы с ним. Эти сроки намного сокращаются, если пользователь всерьез знаком с одним или лучше с несколькими языками программирования.

Но даже такой пользователь нуждается в специальных средствах диагностики и контроля программ. Чем их больше, тем совершеннее система программирования. При этом пользователь-программист должен заботиться и о том, что бы такие средства входили в программные модули, которые он создает.

9.3.1. Некоторые правила культурного программирования

Выше мы описали множество методов программирования на языке системы Mathematica. Попробуем сформулировать некоторые общие правила так называемого *культурного программирования* с учетом специфики систем Mathematica, позволяющие создавать надежные и эффективные программные средства.

- Тщательно продумайте алгоритм решения задачи. Порой выбор лучшего алгоритма позволяет кардинально повысить скорость вычислений и упростить программу (впрочем, одновременно это бывает далеко не всегда).
- Используйте, прежде всего, возможности функционального программирования, из него родились основы языка программирования систем Mathematica.
- Разделяйте задачу на малые части и оформляйте их в виде законченных программных модулей, прежде всего, функций.
- Не скупитесь на программные комментарии: чем их больше, тем понятнее программа и тем больше шансов, что она заинтересует пользователей и будет долго жить. Учтите, что ясность программы в большинстве случаев важнее скорости ее работы.
- Тщательно готовьте сообщения об ошибках и диагностические сообщения, а также наименования программных модулей и описания их назначения.
- Тщательно проводите диагностику программных модулей, в том числе с самыми безумными значениями и типами параметров; хорошо спроектированный модуль должен диагностировать любые виды ошибочных ситуаций при работе с ним и реагировать на них адекватным образом.
- Используйте имена переменных и констант в стиле, принятом в Mathematica, и обязательно с использованием понятных по смыслу обозначений. Как можно реже используйте в именах зарегистрированные идентификаторы команд и функций.
- Заменяйте циклы функциями обработки списков, например, функциями суммирования и произведения. Применяйте эффективные варианты упрощенных операторов и функций.
- В максимальной степени используйте функции ядра системы. Обращайтесь к пакетам расширений только в том случае, когда это действительно необходимо.
- Проводите тщательное тестирование своих модулей, в том числе с выполнением их трассировки. Помните, что нет программы, которую нельзя хоть чуть-чуть, но улучшить и сократить. Но цените затраченное на это время.
- По мере возможности используйте готовые апробированные программные модули – изобретать велосипед и делать то, что уже сделано, неразумно.
- Обращайте особое внимание на реализацию механизма контекстов, позволяющего избежать грубых ошибок при модернизации различных объектов программ, прежде всего, наборов функций.

- Не слишком оригинальничайте! Не применяйте программные трюки и недокументированные приемы программирования. Такие программы могут выглядеть в момент создания удивительно эффектными и потрясающе оригинальными, но вполне возможно, что в следующей версии системы они перестанут работать вообще, поскольку разработчики обычно стараются исключить любые недокументированные трюки в своих программах.

Применение этих рекомендаций на практике позволит вам создавать программы, которые нужны не только вам, но и многим пользователям системы Mathematica. Только такие программы могут быть размещены в Internet и, вполне возможно, войти в пакеты расширения и электронные книги системы Mathematica.

9.3.2. Трассировка программных модулей

В практике подготовки и отладки программ важное значение имеет наличие специальных средств отладки программ по шагам – средств **трассировки**. Mathematica имеет ряд функций для осуществления трассировки своих программных конструкций.

Функция **Trace[expr]** позволяет выполнить трассировку выражения expr. Возьмем простой пример – вычисление выражений $2*(3+4)^2/5$:

Trace[2(3 + 4)²/5]

$$\{\{\{\{3 + 4, 7\}, 7^2, 49\}, \{\frac{1}{5}, \frac{1}{5}\}, \frac{49}{5}, \frac{49}{5}\}, \frac{2 \cdot 49}{5}, \frac{98}{5}\}$$

Результат трассировки представлен вложенными списками, имеющими два элемента: вычисляемое выражение и результат вычислений. В частности, для приведенного примера отчетливо видно, что вначале вычисляется выражение в круглых скобках (3+4) и получается результат 7, который затем возводится в квадрат – получается число 49. Затем вызывается явно не записанная единица для деления на 5, потом 49 умножается на 1/5 и, наконец, 49/5 умножаются на 2, и получается конечный результат. Отсюда ясно, что даже равноценные операции умножения и деления Mathematica разделяет по приоритету – деление выполняется перед умножением!

Символьные операции также могут трассироваться (см. пример ниже):

Trace[a*a/(b*b)]

$$\{\{\{\{b \cdot b, b^2\}, \frac{1}{b^2}, \frac{1}{b^2}\}, \frac{a}{b^2}\}, \frac{a \cdot a}{b^2}, \frac{a \cdot a}{b^2}, \frac{a^2}{b^2}\}$$

Можно выполнить и трассировку рекуррентных вычислений. Функция **TracePrint[expr]** дает распечатку последовательности действий по вычислению выражения expr:

TracePrint[a*b/c]

```

ab
c
Times
a
```

$$\frac{b}{\frac{1}{c} \cdot \text{Power} \cdot c \cdot -1} = \frac{ab}{c}$$

9.3.3. Основные функции трассировки и отладки

Помимо указанных примеров выполнения трассировки и отладки возможны и иные их варианты с помощью ряда функций. Они представлены ниже:

- **Off[s]** – отключает сообщения трассировки, связанные с символом *s*.
- **Off[m1, m2, ...]** – отключает несколько сообщений.
- **Off[]** – отключает все сообщения трассировки.
- **On[s]** – включает трассировку для символа *s*.
- **On[m1, m2, ...]** – включает ряд сообщений.
- **On[]** – включает трассировку для всех символов.
- **Trace[expr]** – генерирует список всех выражений, используемых при вычислении *expr*.
- **Trace[expr, form]** – включает только те выражения, которые сопоставимы с *form*.
- **Trace[expr, s]** – включает все вычисления, которые используют правила преобразования, связанные с символом *s*.
- **TraceDialog[expr]** – инициирует диалог для каждого выражения, используемого при вычислении *expr* (на каждом шаге продолжение диалога осуществляется исполнением команды **Return[]**).
- **TraceDialog[expr, form]** – инициирует диалог только для выражений, сопоставимых с *form*.
- **TraceDialog[expr, s]** – инициирует диалоги только для выражений, при вычислении которых используются правила преобразований, связанные с символом *s*.
- **TraceLevel[]** – возвращает тот уровень ее выхода, который в данный момент заполняется.
- **TracePrint[expr]** – выводит (печатает) все выражения, используемые в процессе вычисления *expr*.
- **TracePrint[expr, form]** – включает в операцию только те выражения, которые совпадают с *form*.
- **TracePrint[expr, s]** – включает все вычисления, которые применяют правила преобразования, связанные с указанным символом *s*.
- **TraceScan[f, expr]** – применяет *f* ко всем выражениям, используемым при вычислении *expr*.

- **TraceScan[f, expr, form]** – включает только те выражения, которые сопоставимы с form.
- **TraceScan[f, expr, s]** – включает все вычисления, которые применяют правила преобразования, связанные с символом s.
- **TraceScan[f, expr, form, fp]** – применяет f до вычисления, а fp после вычисления к выражениям, используемым при вычислении expr.

С этими функциями могут использоваться следующие основные опции и относящиеся к ним значения:

- **TraceForward** – указывает, следует ли включать в вычислительную цепочку более поздние (последующие) выражения, которые содержат искомую форму шаблона.
- **TraceInternal** – имея значения True или False, указывает, следует ли трассировать вычисления выражений, генерируемые внутри Mathematica. Вспомогательная установка Automatic трассирует выбранное множество внутренних вычислений, включая Messages и установки или отмены установок видимых символов.
- **\$TraceOff** – является значением активной в данный момент опции TraceOff, относящейся к Trace и родственным функциям. Она может быть восстановлена (сброшена в исходное состояние – reset) в течение трассировки для изменения множества выражений, в которых трассировка заблокирована.
- **TraceOff** – отключает трассировку.
- **\$TraceOn** – является значением активной в данный момент опции TraceOn, относящейся к функции Trace и родственным функциям. Она может быть восстановлена в процессе трассировки для изменения множества выражений, в которых произойдет трассировка.
- **TraceOn** – включает трассировку.
- **TraceOriginal** – указывает, следует ли проверять форму каждого выражения перед вычислением его заголовка и аргументов.
- **\$TracePattern** – активный в данный момент аргумент (параметр) конфигурации (pattern argument), относящийся к Trace и родственным функциям. Он может быть сброшен (восстановлен) в процессе исполнения trace для изменения множества выражений, записываемых или выводимых.
- **\$TracePostAction** – активный в данный момент четвертый параметр функции TraceScan (или эквивалент в родственных функциях). Он может быть установлен во время трассировки для изменения операции, применяемой после того, как перехваченные выражения вычислены.
- **\$TracePreAction** – активный в данный момент первый аргумент функции TraceScan (или эквивалент в родственных функциях). Он может быть установлен во время трассировки для изменения действия, предпринимаемого перед тем, как перехваченные выражения будут вычислены.

Помимо приведенных выше иллюстраций применения функций трассировки, множество примеров трассировки имеется в базе данных помощи систем Mathematica. Необходимо, однако, отметить, что применение этих функций на совре-

менном уровне программирования ограничено; в подобной трассировке особой необходимости нет, поскольку система выдачи диагностических сообщений позволяет выполнять более удобными средствами.

9.4. Новые средства программирования в Mathematica 6

9.4.1. Динамическое изменение переменных и функция *Dynamic*

Как уже отмечалось в конце Главы 1, в Mathematica 6 введена новая концепция динамического изменения значений переменных. Рассмотрим ее более подробно.

Обычно глобальные переменные в ноутбуке доступны в любом его месте. Например, мы можем задать где-то значение переменной *x*, равное 1, и проверить ее значение:

```
x=1
1
x
1
```

(1)

Затем мы можем изменить это значение на 0.5 и также проверить его:

```
x=0.5
0.5
```

Принципиально важно, что строке вывода (1) остается прежнее значение переменной *x*, поскольку никакой динамической связи между новым и старым значениями переменных нет.

Совсем иное дело, если мы объявим динамическую связь с помощью функции **Dynamic**:

```
Dynamic[x]
0.5
```

(2)

В строке вывода (2) появится значение переменной *x*, равное 0.5, т.е. последнее присвоенное переменной значение. Но теперь изменим значение переменной *x* на иное, например 123:

```
x=123
123
```

Вы увидите, что выход в строке вывода (2), расположенной выше, тут же изменится с 0.5 на 123. В конце Главы 1 было показано, что это наблюдается и в том случае, когда значения переменных задаются слайдерами, при этом перемещения движка последнего слайдера будут вести к перемещению движков предыдущих слайдеров. Тем самым реализуется динамическая связь между конечными и предыдущими значениями переменной, включенной в список параметров функции **Dynamic**.

Описанная возможность применима к любому выражению, т.е. основная форма функции имеет вид **Dynamic[expr]**. В форме **Dynamic[expr,None]** устанавливается запрет на динамическую связь выражения. Например, команда

{Slider[Dynamic[x,None]],Dynamic[x]}

выводит слайдер для изменения x , но перемещение его движка件不可能.

В следующих формах записи

Dynamic[expr,f] **Dynamic[expr,{f,fend}]** **Dynamic[expr,{fstart,f,fend}]**

динамическая связь устанавливается на множественные значения переменных в соответствии со значениями переменной f .

Большинству пользователей описанная возможность может показаться экзотикой. Но это революционное изменение техники программирования, и надо полагать, что должно пройти время, прежде чем это изменение будет оценено должным образом и всерьез применено на практике.

9.4.2. Динамический модуль *DynamicModule*

Часто желательно локализовать динамическую связь пределами одного модуля. Для этого служит функция создания модуля с динамической связью:

DynamicModule[{x,y,...},expr] **DynamicModule[{x=x₀,y=y₀,...},expr]**

Работу этого модуля хорошо поясняет рис. 9.5. Из него видно, что заданное изначально значение переменной $x=123$ сохраняется и после изменения переменной x в модуле. Речь, стало быть, идет о разных переменных с одним именем: переменная x в модуле локальная, тогда как за пределами модуля (до него и после) она глобальная. Примеры применения модуля *DynamicModule* уже приводились неоднократно.

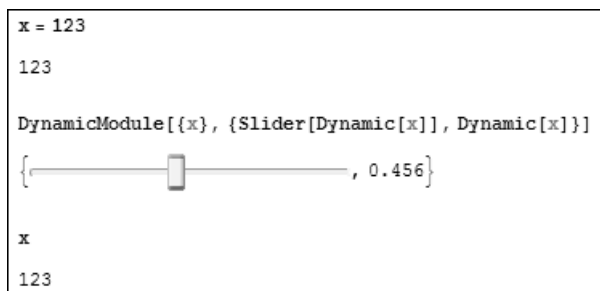


Рис. 9.5. Иллюстрация локализации переменной с динамическим изменением ее значения в модуле *DynamicModule*

9.4.3. Функция сброса интерактивных изменений *Deploy*

Функция **Deploy**[*expr*] сбрасывает введенные изменения выражения. Например, конструкция

```
{Graphics[{Disk[], Inset[Slider2D[]]], Deploy[Graphics[{Disk[], Inset[Slider2D[]]}]]}
```

выводит два двумерных слайдера, движки которых можно устанавливать в любое положение. Однако стоит выполнить эту команду, как установки станут нулевыми и движки двумерных слайдеров установятся в центральное положение.

9.4.4. Модуль манипуляций *Manipulate*

Модуль манипуляций *Manipulate* позволяет создавать различные интерактивные средства по заданному выражению *expr* и заданным изменениям его параметров:

```
Manipulate [expr, {u, umin, umax}]
```

```
Manipulate [expr, {u, umin, umax, du}]
```

```
Manipulate [expr, {{u, uinit}, umin, umax, ...}]
```

```
Manipulate [expr, {{u, uinit, ulbl}, ...}]
```

```
Manipulate [expr, {u, {u1, u2, ...}}]
```

```
Manipulate [expr, {u, ...}, {v, ...}, ...]
```

```
Manipulate [expr, "cu"->{u, ...}, "cv"->{v, ...}, ...]
```

Здесь выражение трактуется в самом общем виде. Это может быть математическое выражение, графическая функция и т.д., что открывает неисчерпаемые возможности создания ноутбуков и просто программных модулей с превосходными средствами динамической интерактивности. Можно задавать стандартное, плавное, дискретное, целочисленное и т.д. изменение параметров с помощью элементов динамической интерактивности.

На рис. 9.6 показан пример создания модуля, создающего фигурку человека, состоящую из диска и ряда отрезков прямых. Содержание модуля совершенно очевидно. Отрезки прямых – ручки и отрезки прямых – ножки могут попарно поворачиваться с помощью верхнего и нижнего слайдеров, которые автоматически создаются модулем *Manipulate* с помощью динамического объекта *Slider*. Тем самым реализуется метод объектно-ориентированного программирования.

Щелкнув мышью на кнопке в виде серого кружка со знаком «+», можно вывести контекстное меню управления созданным в строке вывода графическим объектом. Последняя позиция **Autorun** меню запускает слайдеры на поочередное перемещение движка взад-вперед, т.е. создает анимацию объекта – человек поочередно начинает синхронно вращать тута-сюда ручки и ножки. Кадр анимации показан на рис. 9.7. Там же виден анимационный проигрыватель и обычное контекстное меню правой клавиши мыши.

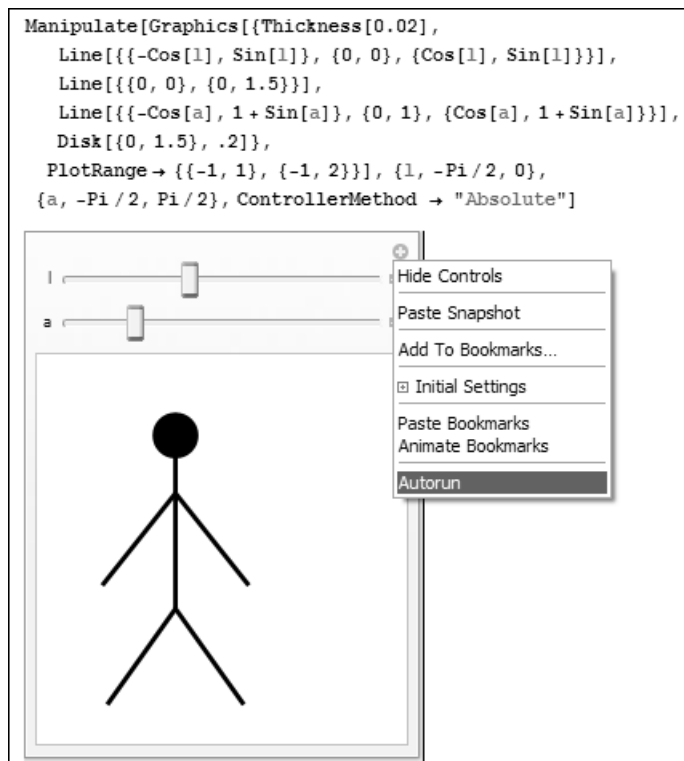


Рис. 9.6. Модуль, строящий фигуру человека

Модуль Manipulate позволяет создавать интерактивно управляемые графики самого различного вида. Так, на рис. 9.8 показан модуль, который строит контурный график функции двух переменных p и q , значения которых можно менять слайдерами. Кроме того, фокусы графиков могут с помощью локаторов перемещаться. Все это позволяет наглядно интерпретировать вид графика при изменении всех возможных его параметров и тут же наблюдать различные варианты его.

Еще один модуль построения оригинального графического объекта показан на рис. 9.9. Для построения используется функция построения трехмерного контурного графика с опцией, запрещающей построение контурных линий. При изменении параметра a в уравнении $x^2 + y^2 + a \cdot z^3 = 1$ поверхность как бы выворачивается наизнанку (рис. 9.10).

9.4.5. Средства отладки программ и ноутбуков

Описанные выше средства отладки и трассировки программ и отладки ноутбуков остались и в системе Mathematica 6. Но они дополнились некоторыми новыми или существенно модернизированными возможностями. Прежде всего, отметим

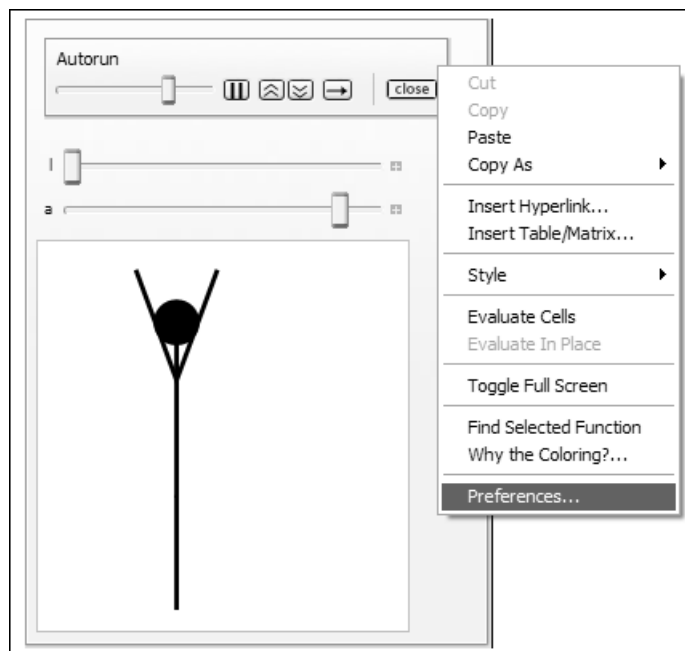


Рис. 9.7. Один из кадров анимации человечка, который строит модуль из рис. 9.6

возможность установки некоторых параметров отладчика программ в окне предпочтений **Preferences**. Это окно вызывается из контекстного меню (рис. 9.7) или из меню **Edit**.

Окно предпочтений с открытой вкладкой **Debugger** (Отладчик) показано на рис. 9.11. На вкладке можно изменить цветовые выделения, которые используются при отображении записи кодов программ и ноутбуков в ходе их ввода на входном языке **Mathematica 6** и на языке ее программирования. Окно имеет и ряд других вкладок, которые можно при необходимости просмотреть.

В выводе ошибок **Mathematica 6** имеет некоторые особенности. Как видно из рис. 9.12, при синтаксической ошибке во вводимом выражении сообщения об ошибках изначально не выводятся и не загромождают экран. Вместо этого ошибочные символы выделяются красным фоном, и в конце строки ввода появляется красный прямоугольник со знаком «+». Если его активизировать, то появляются ячейки с сообщениями об ошибках тоже красного цвета.

Mathematica 6 имеет также средства для полноценной профессиональной подготовки программ, например пакетов расширения. Позиция **New** меню **File** открывает подменю, которое имеет следующие команды:

- **Notebook(.nb)** – открывает окно подготовки ноутбуков;
- **Slide Show** – открывает окно подготовки слайд-шоу;

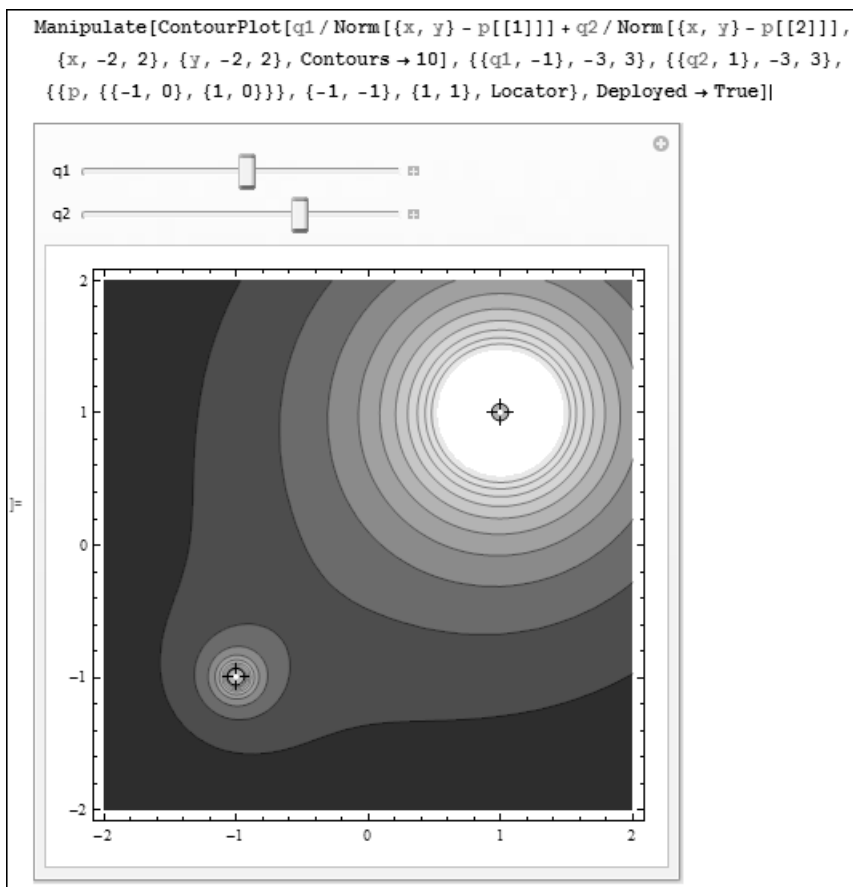


Рис. 9.8. Модуль построения графика функции в полярных координатах с интерактивным управлением

- **Demonstration** – открывает окно подготовки демонстраций;
- **Package (.m)** – открывает окно подготовки пакета расширения;
- **Text Document (.txt)** – открывает окно подготовки текстового документа.

С помощью команды **Open** можно загружать готовые ноутбуки, пакеты расширения и прочие документы. Рисунок 9.13 иллюстрирует средства Mathematica 6, которые служат для редактирования пакетов расширения и их отладки.

Из этих средств главным является редактор-отладчик, окно которого показано на рис. 9.13 в левом нижнем углу. Под титульной строкой редактора расположены кнопки вывода списков примененных в пакете расширения функций и секций, кнопка обновления **Update**, окно опции вывода панели отладчика и кнопка запуска пакета расширения. Соответствующие средства есть и позиции **Debugger Control** позиции **Evaluation** основного меню системы.

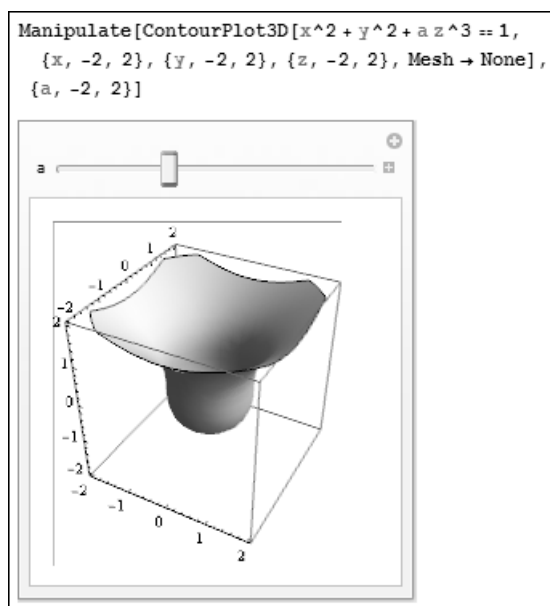


Рис. 9.9. Построение поверхности по уравнению $x^2 + y^2 + a z^3 = 1$ (первый случай)

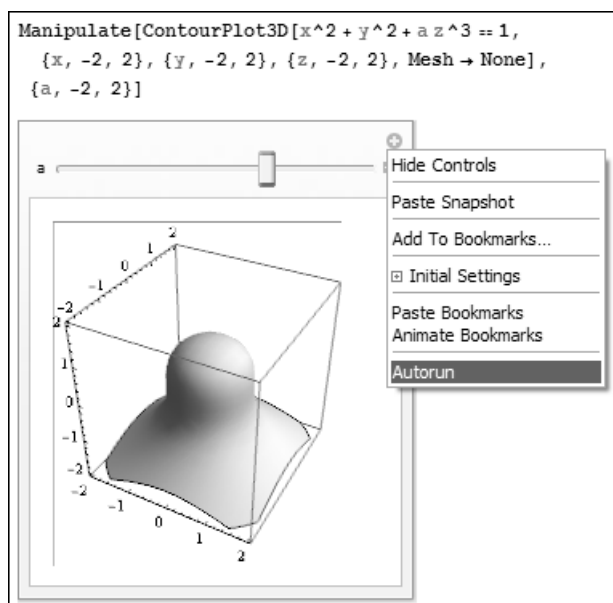


Рис. 9.10. Построение поверхности по уравнению $x^2 + y^2 + a z^3 = 1$ (второй случай)

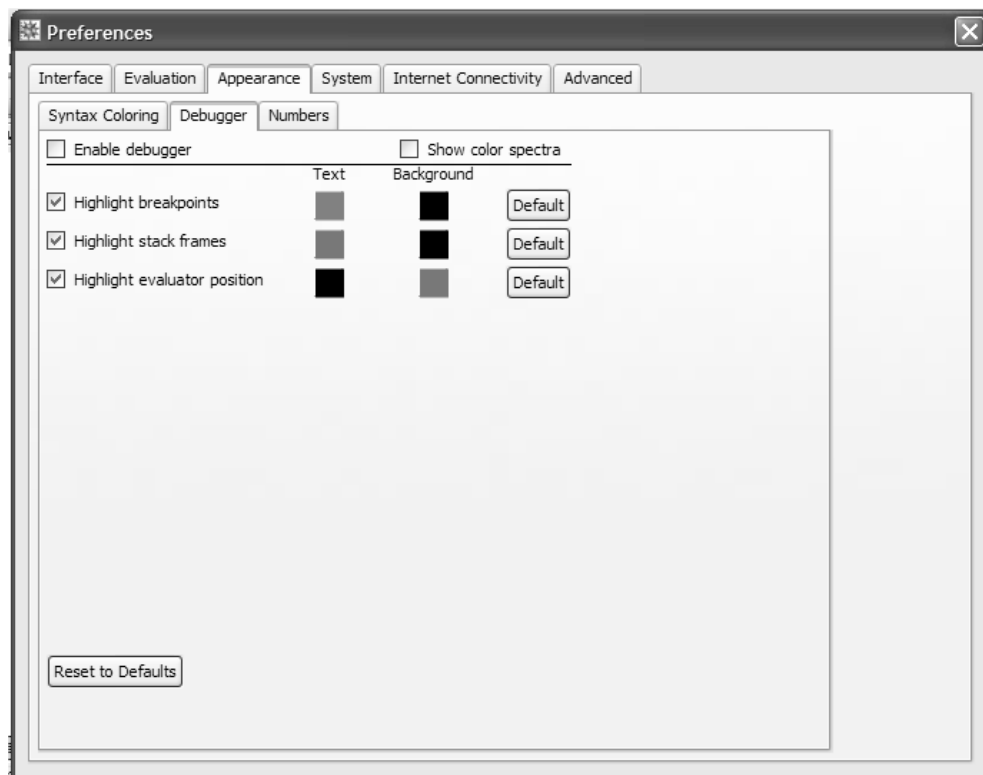


Рис. 9.11 Окно предпочтений с открытой вкладкой Debugger

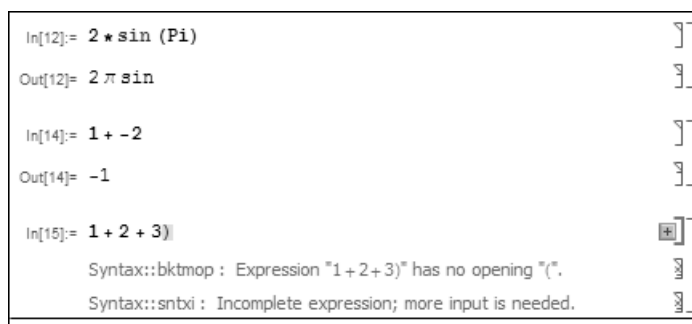


Рис. 9.12. Пример вывода сообщений об ошибках в Mathematica 6

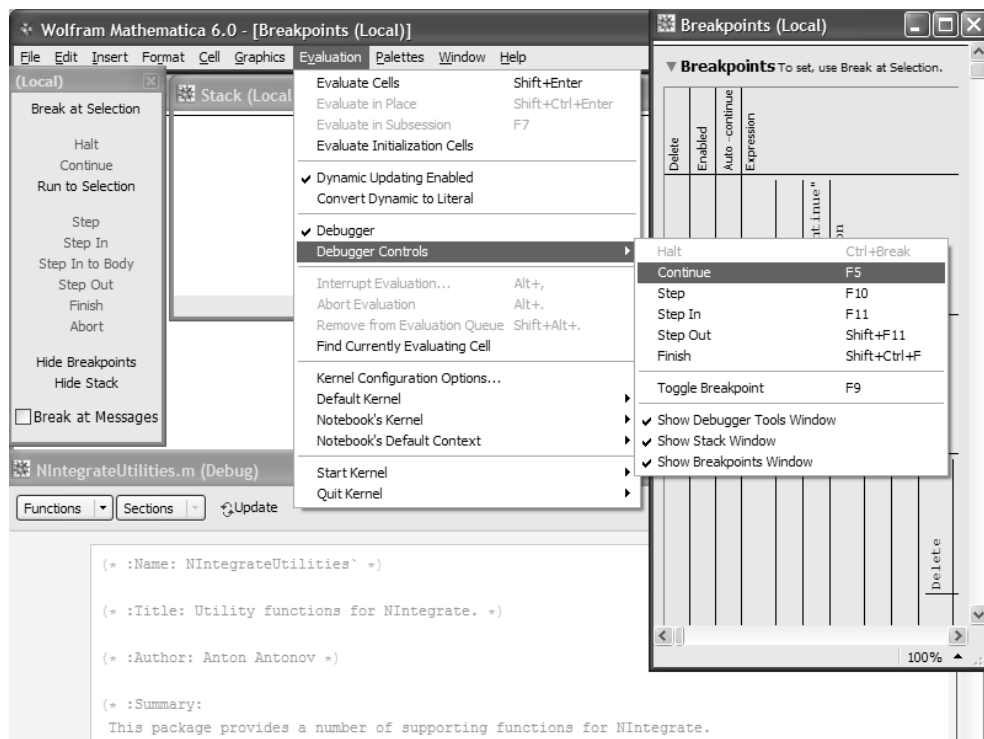


Рис. 9.13. Средства работы с пакетами расширения

9.5. Обзор пакетов расширения Add-On

9.5.1. Состав пакетов расширения Add-On систем Mathematica 5.1/5.2

Как уже отмечалось, для расширения возможностей языка программирования систем Mathematica в их состав входит библиотека расширений, называемых пакетами расширения Add-On. Набор этих пакетов и их состав меняется от одной версии системы Mathematica к другой. Состав пакетов указан в справке по каждой версии в разделе Add-On. В задачу данного раздела не входит детальный анализ возможностей пакетов расширений в конкретные версии систем класса Mathematica. Ниже дается лишь общая оценка наиболее важных пакетов расширения и приводятся примеры их применения.

Состав пакетов Add-On систем Mathematica 5.1/5.2 представлен ниже.

- **Algebra** – работа с полиномами, алгебраическими неравенствами, Гамильтоновой алгеброй и др.

- **Calculus** – символьные вычисления производных, интегралов и пределов функций, прямое и обратное преобразования Фурье и Лапласа, решение систем нелинейных уравнений, реализация инвариантных методов, решение дифференциальных уравнений в частных производных, нахождение полных интегралов и дифференциальных инвариантов нелинейных уравнений, аппроксимация Паде, вычисление эллиптических интегралов и работа с векторами.
- **DiscreteMath** – вычисления из области дискретной математики, комбинаторики, вычислительной геометрии и теории графов, решение рекуррентных и разностных уравнений, операции с целыми числами и т.д.
- **Geometry** – функции для выполнения геометрических расчетов, задания правильных прямоугольников и многогранников, вращения геометрических фигур в плоскости и в пространстве.
- **Graphics** – построение графиков специального вида, геометрических фигур и поверхностей, графиков параметрически и неявно заданных функций, представления функций комплексного переменного, отображение ортогональных проекций трехмерных фигур, имитация теней, функции оформления графиков.
- **LinearAlgebra** – решение задач линейной алгебры, дополнительные векторные и матричные операции, задание ортогональных векторных базисов и др.
- **Miscellaneous** – задание единиц измерения физических величин, данные о химических элементах, физические константы, географические данные и т.д.
- **NumberTheory** – функции теории чисел.
- **NumericalMath** – реализация важнейших численных методов, аппроксимация данных и аналитических функций полиномами, сплайнами и тригонометрическими рядами, численное интегрирование и дифференцирование, решение дифференциальных уравнений, вычисление корней нелинейных уравнений, нахождение вычетов и разложений в комплексной плоскости и т.д.
- **Statistics** – статистические функции для непрерывных и дискретных распределений, реализация линейной и нелинейной регрессий, вычисление параметров ряда распределений (особенно нормального), функции сглаживания и подгонки данных и т.д.
- **Utilities** – дополнительные утилиты для работы с бинарными файлами, с памятью ПК, поддержки языков, для работы с системами класса AutoCAD и т.д.

В системе Mathematica 5/5.1/5.2 эти пакеты размещены в директории Add-On\StandardPackages и написаны на языке программирования системы. В Mathematica 6 эти пакеты расширения сохранились, но размещены в папке Add-On\LegacyPackades – приставка Legacy (Наследство) подчеркивает наследственный характер этих пакетов расширения. Их наличие обеспечивает почти полную совместимость ноутбуков, написанных в среде предшествующих версий со средой Mathematica 6.

Функции ряда пакетов расширения Add-On включены в ядро Mathematica 6 и их применение предпочтительно. В отдельных случаях могут быть конфликты между одноименными функциями ядра Mathematica 6 и функциями в «наследных» пакетах расширения.

Наиболее важные функции ряда пакетов расширения были описаны выше, например, это функции пакетов Graphics, LinearAlgebra и Statistica. Ниже выборочно отмечаются возможности и других пакетов расширения Add-On.

9.5.2. Пакет алгебраических функций Algebra

Пакет расширения Algebra содержит ряд новых функций для работы с неравенствами, ограниченными полями и полиномами. Для доступа сразу ко всем функциям пакета используется команда:

```
<<Algebra`
```

До сих пор мы сталкивались с решениями уравнений, представленных равенствами. Пакет Algebra дает важное дополнение в виде функций, обеспечивающих работу с *неравенствами*. Так, это реализует следующая функция:

SemialgebraicComponents[ineqs, vars] – определяет комплект решений неравенств ineqs по переменной vars.

Приведенные ниже примеры иллюстрируют работу данной функции:

```
<<Algebra`AlgebraicInequalities`
```

```
SemialgebraicComponents[{x (x^3 - 2) (x^2 - 1) > 10}, x]
{-3, 3}
```

```
SemialgebraicComponents[{x (x^3 - 2) (x^2 - 1) > -10}, x]
{0}
```

```
SemialgebraicComponents[{x (x^3 - 2) (x^2 - 1) > 0}, x]
{-2,  $\frac{27}{32}$ , 2}
```

```
SemialgebraicComponents[{x^2 + y^2 < 3, x y > 0}, {x, y}]
{{- $\frac{1}{8}$ , - $\frac{1}{8}$ }, { $\frac{1}{8}$ ,  $\frac{1}{8}$ }}
```

```
SemialgebraicComponents[{x^2 + y^2 + z^2 > 1,
x^2 + (y - 1)^3 + (z - 2)^4 < -2}, {x, y, z}]
```

```
{-2, -2, 2}, {- $\frac{31}{32}$ , - $\frac{1}{2}$ , 2}, {- $\frac{1}{2}$ , -1, 2}, {- $\frac{1}{2}$ , - $\frac{1}{2}$ , 2},
```

```
{0, -2, 2}, {0, - $\frac{47}{48}$ , 2}, {0, - $\frac{2}{3}$ , 2}, {0, - $\frac{1}{2}$ , 2},
```

```
{ $\frac{1}{2}$ , -1, 2}, { $\frac{1}{2}$ , - $\frac{1}{2}$ , 2}, { $\frac{31}{32}$ , - $\frac{1}{2}$ , 2}, {2, -2, 2}
```

Для решения неравенств служит функция:

- **InequalitySolve[expr, var]** – решает неравенство expr относительно переменной var.

- **InequalitySolve[expr,{var1, var2,...}]** – решает неравенство expr относительно нескольких переменных.

Следующие примеры иллюстрируют применение данной функции:

```
<<Algebra`InequalitySolve`
InequalitySolve[x (x^2 - 5) (x^2 - 6) > 0, x]
(-√6 < x < -√5 || 0 < x < √5 || x > √6)
InequalitySolve[x^2/Abs[x - 2] >= 0 && 1/x < x + 1, x]
(1/2 (-1 - √5) < x < 0 || 1/2 (-1 + √5) < x < 2 || x > 2)
InequalitySolve[Abs[x - 2] <= 3 && E^x <= 3, x]
InequalitySolve::npi: A nonpolynomial equation or inequality
encountered. The solution set may be incorrect.
-1 ≤ x ≤ Log[3]
```

Встроенные функции для представления комплексных данных не всегда работают корректно. Подпакет ReIm обеспечивает переименование этих функций:

```
<<Algebra`ReIm`
Re[1/x+1/y]
( Re[x] / (Im[x]^2 + Re[x]^2) + Re[y] / (Im[y]^2 + Re[y]^2) )
Re[(z + I)^3 + Exp[I z]]
(e^(-Im[z]) Cos[Re[z]] - 2 (1 + Im[z])^2 Re[z] +
Re[z] (- (1 + Im[z])^2 + Re[z]^2) )
Im[x] ^= 0; RealValued[f, g]
{ f, g }
Im[1/(1 - I f[x] g[x])]
f[x] g[x] / (1 + f[x]^2 g[x]^2)
Im[Sin[a]]
Cos[Re[a]] Sinh[Im[a]]
```

Пакет обеспечивает также работу с полями. *Поле* является алгебраической структурой, подчиняющейся правилам обычной арифметики. Оно может быть определено как множество, имеющее не менее двух элементов, над которыми определены двоичные перестановочные и ассоциативные операции дополнения и умножения. Кроме того, для существования поля нужны два особых элемента: нуль 0, задающий правило сложения $a+0=a$, и 1 – для задания правила умножения $a \cdot 1=1$. Определено также понятие противоположного элемента $-a$, такого, что $a+(-a)=0$ и обратного элемента a^{-1} , такого, что $a^{-1}a=1$.

Поле характеризуется размером p и целым положительным d , называемым степенью расширения. Этот пакет представляет элементы поля как полиномиальные с единственной переменной.

Пакет задает набор функций **GF[p][{k}]**, **GF[p, 1][{k}]**, **GF[p, {0, 1}][{k}]**, **GF[p, d]** и **GF[p, ilist][elist]**, действие которых иллюстрируют следующие примеры:

```
<<Algebra`FiniteFields`
GF[7][4] + GF[7][6]
```

```
GF[7,{0,1}][4]+ GF[7,{0,1}][6]
GF[3,4][1,2,1] GF[3,4][2,2,2,0]
GF[3,{2,1,0,0,1}][1,2,1] GF[3,{2,1,0,0,1}][2,2,2,0]
GF[5,1][1] + GF[3,4][1,1,1]
GF[3,{2,1,0,0,1}][1,1,1]+GF[5,{0,1}][1]
```

Вряд ли подробное описание их заинтересует большинство читателей. Специалисты по полям не затруднит более детальное знакомство с этими функциями по справочной базе данных в разделе Add-on. Там же можно найти описание ряда других функций, относящихся к теории ограниченного поля.

Следующие функции подпакета RootIsolation позволяют оценивать *интервалы изоляции* для действительных и комплексных корней полиномов:

- **CountRoots[poly,{x,m1,m2}]** – возвращает число корней полинома poly от переменной x в комплексном интервале {m1, m2}.
- **RealRootsIntervals[poly]** – возвращает разделенный интервал изоляции для вещественных корней полинома poly.
- **RealRootsIntervals[poly1, poly2,...]** – возвращает разделенные интервалы изоляции для вещественных корней полиномов.
- **ComplexRootsIntervals[poly]** – возвращает разделенный интервал изоляции для комплексных корней полинома.
- **ComplexRootsIntervals[poly1, poly2,...]** – возвращает разделенные интервалы изоляции для комплексных корней полиномов.
- **ContractInterval[a,n]** – возвращает интервал изоляции для числа a с точностью, задаваемой числом знаков n-результата.

Если ограниченные поля применяются относительно редко, то полиномы встречаются сплошь и рядом во многих математических и научно-технических расчетах. В пакете расширения Algebra определен ряд новых операций над полиномами. Начнем их рассмотрение со следующей функции:

- **PolynomialExtendedGCD[poly1,poly2]** – возвращает наибольший общий делитель двух полиномов.
- **PolynomialExtendedGCD[poly1,poly2, Modulus->p]** – возвращает наибольший общий делитель двух полиномов по модулю p.

Примеры на применение этой функции даны ниже:

```
<<Algebra`PolynomialExtendedGCD`
PolynomialExtendedGCD[x^2 + 3 x + 2,
Expand[(x + 1)(x + 2)], Modulus->7]
{(2 + 3x + x^2), {0, {} }
PolynomialExtendedGCD[
Expand[(12+I) z^2 + 5 z + I] (I z + 3)],
Expand[(9+I) z + (3+I)] (3I z + 9)]
{ (-3 i + z), { (- 2261 / 3341 + 710 i / 3341) ,
(( 35 / 3341 - 395 i / 10023 ) + ( 5959 / 20046 - 2053 i / 20046 ) z) } }
```

Другой является следующая функция:

PolynomialPowerMod [poly1,n,{poly2,p}] – существенно ускоренная функция **PolynomialMod**.

Еще одна функция в трех ее модификациях работает с симметричными полиномами:

- **SymmetricReduction[{x₁,...,x_n},k]** – возвращает симметричный полином степени k по переменным {x₁,...,x_n}.
- **SymmetricReduction[f,{x₁,...,x_n}]** – возвращает часть полинома {p,q} по переменным {x₁,...,x_n}, где f=p+q, причем p есть симметричная часть, q – остаток.
- **SymmetricReduction[f,{x₁,...,x_n},{s₁,...,s_n}]** – возвращает часть полинома {p,q} по переменным {x₁,...,x_n}, где элементарный симметричный полином представляет список {s₁,...,s_n}.

Следующий пример поясняет создание симметричного полинома степени 4 по переменным {x,y,z,w,t}:

```
<<Algebra`SymmetricPolynomials`
SymmetricPolynomial[{x, y, z, w, t}, 4]
twxy + twxz + twyz + txyz + wxyz
```

Действие других функций поясняют следующие примеры:

```
SymmetricReduction[(x + y)^2 + (x + z)^2 + (z + y)^2,
{ x, y, z } ]
{2 (x + y + z)^2 - 2 (xy + xz + yz), 0}
SymmetricReduction[x^5 + y^5 + z^4, { x, y, z } ,
{ s1, s2, s3 } ]
{ s1^5 - 5 s1^3 s2 + 5 s1 s2^2 + 5 s1^2 s3 - 5 s2 s3, z^4 - z^5 }
```

Раздел Quaternion пакета дает средства для работы с Гамильтоновой алгеброй кватернионов. С квантернионами можно выполнять операции, как с комплексными числами. Алгебра кватернионов вряд ли интересует большинство читателей, поэтому ее функции не описываются.

9.5.3. Пакет вычислительных функций Calculus

Пакет расширения Calculus вводит в систему Mathematica ряд улучшенных вычислительных функций.

Так, многие нелинейные дифференциальные уравнения не имеют общих решений. В пакете определена функция, позволяющая найти решения в форме *полного интеграла*:

CompleteIntegral[eqn,u[x,y,...],{x,y,...}] – создает полный интеграл для дифференциального уравнения, касательного к u[x,y,...].

Применение ее поясняют следующие примеры:

```
<<Calculus`
CompleteIntegral[Derivative[0,1][u][x,y]==(u[x,y]+x^2*
Derivative[1,0][u][x,y]^2)/y,u[x,y],{x,y}]
```

$$\left\{ \left\{ u[x, y] \rightarrow \frac{1}{4} \left((4y - B[1]) B[1] - 2 B[1] \log[x] - \log[x]^2 \right) \right\}, \right. \\ \left. \left\{ u[x, y] \rightarrow \frac{1}{4} \left((4y - B[1]) B[1] + 2 B[1] \log[x] - \log[x]^2 \right) \right\} \right\}$$

CompleteIntegral[-

[x,y]+(2+y)*Derivative[0,1][u][x,y]+x*Derivative[1,0][u][x,y]+3*Derivative[1,0][u][x,y]^2==0,u[x,y],{x,y},
IntegralConstants->F]

$$\left\{ \left\{ u[x, y] \rightarrow \right. \right. \\ 12 - \frac{e^{C[3]}}{8} - \frac{\sqrt{-e^{C[3]} (-12 + x)^2}}{2\sqrt{6}} - 2x + (2 + y) C[3] \left. \right\}, \\ \left\{ u[x, y] \rightarrow 12 - \frac{e^{C[3]}}{8} + \frac{\sqrt{-e^{C[3]} (-12 + x)^2}}{2\sqrt{6}} - \right. \\ 2x + (2 + y) C[3] \left. \right\}, \left\{ u[x, y] \rightarrow \right. \\ 12 + 8 e^{C[3]} - 2 \sqrt{\frac{2}{3}} \sqrt{e^{C[3]} (-12 + x)^2} - 2x + (2 + y) C[3] \left. \right\}, \\ \left\{ u[x, y] \rightarrow 12 + 8 e^{C[3]} + 2 \sqrt{\frac{2}{3}} \sqrt{e^{C[3]} (-12 + x)^2} - \right. \\ 2x + (2 + y) C[3] \left. \right\} \left. \right\}$$

Ряд функций данного подпакета относится к заданию и преобразованию координат. Это достаточно простые функции, и читатель может познакомиться с ними самостоятельно. Подпакет **VectorAnalysis** содержит множество функций, используемых при выполнении *векторного анализа*. Здесь необходимо иметь в виду, что речь идет не только о векторах, как представителях одномерных массивов, которые рассматривались ранее. В ряде случаев *вектор* – это направленный отрезок прямой в пространстве, заданном той или иной системой координат.

Некоторые из функций векторного анализа стоит отметить:

- **DotProduct[v1,v2]** – возвращает точечное произведение векторов v1 и v2, заданных в текущей системе координат.
- **CrossProduct[v1,v2]** – возвращает кросс-произведение векторов v1 и v2, заданных в текущей системе координат.
- **ScalarTripleProduct[v1,v2,v3]** – возвращает скалярный триплет для векторов v1, v2 и v3, заданных в текущей системе координат.
- **DotProduct[v1,v1,coordsys]** – возвращает точечное произведение векторов v1 и v2, заданных в системе координат coordsys.
- **CrossProduct[v1,v2,coordsys]** – возвращает кросс-произведение векторов v1 и v2, заданных в системе координат coordsys.

Примеры на применение этих операций представлены ниже:

SetCoordinates[ParabolicCylindrical[]]

ParabolicCylindrical[Uu,Vv,Zz]

DotProduct[{a, b, c},{d, e, f}]

$$\left(a b d e + \frac{1}{4} (a^2 - b^2) (d^2 - e^2) + c f \right)$$

DotProduct[[1, 2, 3], {4, 5, 6}]

$\frac{259}{4}$

CrossProduct[[1., 2, 3], {4, 5, 6}]

{-0.458777, 9.80869, -21. }

ScalarTripleProduct[[1, 0, 1], {1, 1, 0}, {0, 1, 1},
Cartesian]

2

Ряд функций служит для создания *матрицы Якоби* (частных производных) и относящихся к ней понятий:

- **JacobianMatrix**[] – возвращает матрицу Якоби, определенную в текущих координатах.
- **JacobianMatrix**[pt] – возвращает матрицу Якоби в точке pt и в текущих координатах.
- **JacobianMatrix**[coordsys] – возвращает матрицу Якоби, определенную в системе координат coordsys.
- **JacobianMatrix**[pt,coordsys] – возвращает матрицу Якоби, определенную в системе координат coordsys.
- **JaacobianDeteminant**[pt], **JaacobianDeteminant**[pt] и т.д. – вычисление детерминанта матрицы Якоби при указанных выше определениях.
- **ScaleFactor**[], **ScaleFactor**[pt] и т.д. – вычисление масштабного фактора при указанных выше определениях.

Отметим еще несколько функций:

- **Curl**[f] – возвращает вихрь векторного поля f в текущей системе координат.
- **Grad**[f] – возвращает градиент векторного поля f в текущей системе координат.
- **Laplacian**[f] – возвращает Лапласиан векторного поля f в текущей системе координат.
- **Biharmonic**[f] – возвращает Лапласиан Лаплассиана векторного поля f в текущей системе координат.
- **Div**[f,coordsys], **Curl**[f,coordsys] и т.д. – указанные выше функции в системе координат coordsys.

Приведем примеры на использование этих функций:

Laplacian[x*y^2*z^3, **ProlateSpheroidal**[x, y, z]]

$$\frac{1}{\sin[y]^2 + \sinh[x]^2} \left(\csc[y] \operatorname{Csch}[x] (y^2 z^3 \cosh[x] \sin[y] + 2xy z^3 \cos[y] \sinh[x] + 2xz^3 \sin[y] \sinh[x] + 6xy^2 z \csc[y] \operatorname{Csch}[x] (\sin[y]^2 + \sinh[x]^2)) \right)$$

Grad[5 x^2 y^3 z^4, **Cartesian**[x, y, z]]

{10xy^3z^4, 15x^2y^2z^4, 20x^2y^3z^3}

Grad[5 x^2 y^3 z^4, **ProlateSpheroidal**[x, y, z]]

$$\left\{ \frac{10xy^3z^4}{\sqrt{\sin[y]^2 + \sinh[x]^2}}, \frac{15x^2y^2z^4}{\sqrt{\sin[y]^2 + \sinh[x]^2}}, 20x^2y^3z^3 \csc[y] \operatorname{Csch}[x] \right\}$$

Curl[[f[r, theta, phi], g[r, theta, phi], h[r, theta, phi]],
Cylindrical[r, theta, phi]]

$$\left\{ \frac{-r g^{(0,0,1)}[r, \theta, \phi] + h^{(0,1,0)}[r, \theta, \phi]}{r}, \right. \\ \left. \frac{r^2 \theta^2 - h^{(1,0,0)}[r, \theta, \phi],}{-2 \phi r^2 \theta + g[r, \theta, \phi] + r g^{(1,0,0)}[r, \theta, \phi]} \right\}$$

Подпакет **VariationalMethods** содержит функции для реализации *вариационных методов*. Напомним, что вариационные методы заменяют минимизацию функционала, заданного на некотором бесконечномерном линейном пространстве, задачами его минимизации на последовательности конечномерных подпространств. *Функционал* в системе Mathematica задается следующим образом:

$$F = \int_{x \min}^{x \max} f[u[x], u'(x), x] dx.$$

В данный подпакет включены функции:

- **VaritionalD**[f,u[x],x] – дает первую вариационную производную для функционала F одной переменной x;
- **VaritionalD**[f,u[x,y,...],{x,y,...}] – дает первую вариационную производную для функционала ряда переменных;
- **VaritionalD**[f,{u[x,y,...],v[x,y,...],...},{x,y,...}] – дает лист первых вариационных производных для функционала ряда переменных;
- **EulerEquations**[f,u[x],x] – дает равенство Эйлера при f с одной переменной;
- **EulerEquations**[f,u[x,y,...],{x,y,...}] – дает равенство Эйлера при f с рядом переменных;
- **EulerEquations**[f,{u[x,y,...],v[x,y,...],...},{x,y,...}] – дает лист с равенствами Эйлера при f с рядом переменных;
- **FirstIntegral**[f,u[x],x] – дает первый интеграл с f, определенной для одной переменной x;
- **FirstIntegral**[f,{u[x,y,...],v[x,y,...],...},{x,y,...}] – дает первый интеграл при f с рядом переменных;
- **FirstIntegral**[u] – дает первый интеграл, ассоциированный с переменной u.

Применение данных функций поясняют следующие примеры:

```
<<Calculus`VariationalMethods`
VariationalD[y[x] Sin[1+y'[x]], y[x], x]
(-Cos[1+y'[x]] y'[x] + Sin[1+y'[x]] (1+y[x] y''[x]))
EulerEquations[m 1^2 theta'[t]^2/2+m g 1 Cos[theta[t]], theta[t], t]
(-1 m (g Sin[theta[t]] + 1 theta''[t]) == 0)
FirstIntegrals[m(r'[t]^2+r[t]^2 phi'[t]^2)/
2-U[r], r[t], phi[t], t]
FirstIntegrals[
-U[r] + 1/2 m (r[t]^2 phi'[t]^2 + r'[t]^2), r[t], phi[t], t]
```

Помимо указанных функций подпакет содержит функции **VariationalBound** и **NVariationalBound** для представления границ и значений функционала. Ввиду громоздкости записи параметров этой функции ограничимся примерами ее применения:

```
VariationalBound[{-u[r] D[r^2 u'[r], r]/r^2 -
2u[r]^2/r r^2, u[r]^2 r^2}, u[r], {r, 0, Infinity}, (a - r) E^(-
b r), {a}, {b}]
{-0.25, {a -> 2., b -> 0.5}}
VariationalBound[-u[x, y] (D[u[x, y], {x, 2}] +
D[u[x, y], {y, 2}]) - 2u[x, y], u[x, y], {{x, -a, a}, {y, -a, a}},
(x^2 - a^2) (y^2 - a^2) (a1 + a2 (x^2 + y^2)), {a1}, {a2}]
{-0.561572 a^4, {a1 -> 0.292193 / a^2, a2 -> 0.0592283 / a^4}}
NVariationalBound[
{u'[x]^2 + (x^2 + x^4) u[x]^2/4, u[x]^2},
u[x], {x, -Infinity, Infinity},
E^(-a x^2) (1 + b x^2), {a, 0.5}, {b, 0.1}]
{0.804175, {a -> 0.74136, b -> 0.365556}}
```

С полными возможностями этих функций можно ознакомиться по справочной базе данных (раздел Add-on).

9.5.4. Функции дискретной математики – пакет **DiscreteMath**

Пакет **DiscreteMath** задает набор функций *дискретной математики*. Это, прежде всего, функции *комбинаторики* и работы с *графами* (около 450 функций!).

Подпакет **Combinatorica** задает определение ряда функций комбинаторики и теории графов. Следует отметить, что ввиду обилия функций даже в справочной системе даны примеры на лишь избранные функции. Для ознакомления с назначением каждой функции достаточно исполнить команду **?Имя_функции**. Примеры применения некоторых функций комбинаторики представлены ниже:

```
<<DiscreteMath`Combinatorica`
?Permute
Permute[l, p] permutes list
l according to permutation p. More...
MinimumChangePermutations[{a, b, c}]
{{a, b, c}, {b, a, c}, {c, a, b}, {a, c, b}, {b, c, a}, {c, b, a}}
Map[RankPermutation, Permutations[{1, 2, 3, 4}]]
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23}
```

В рамках наиболее характерных задач данной книги подпакет комбинаторики содержит явно избыточное число функций. Большинство из них просты, и заинтересованный читатель может легко воспользоваться ими по мере необходимости.

Mathematica имеет самые обширные возможности в решении задач, связанных с графами. Задание графов и манипуляции с ними также включены в пакет комбинаторики. Они представлены четырьмя группами функций:

- Представление графов;
- Создание графов;
- Свойства графов;
- Алгоритмическая теория графов.

Ввиду большого числа этих функций рассмотрим применение лишь наиболее характерных из функций.

Одной из самых важных функций группы представления графов является функция **ShowGraph** (Показать Граф). Она обеспечивает визуальное представление графа, заданного аргументом функции. Покажем работу избранных функций этой группы на нескольких примерах.

На рис. 9.14 показано построение полного графа и его таблицы. Параметром графа является число 6, характеризующее число узловых точек графа, соединенных друг с другом.

Изменяя значение параметра графа, можно получить множество других графов.

Набор функций данного пакета позволяет строить практически любые виды графов и обеспечивает высокую степень их визуализации. Рисунок 9.15 (сверху)

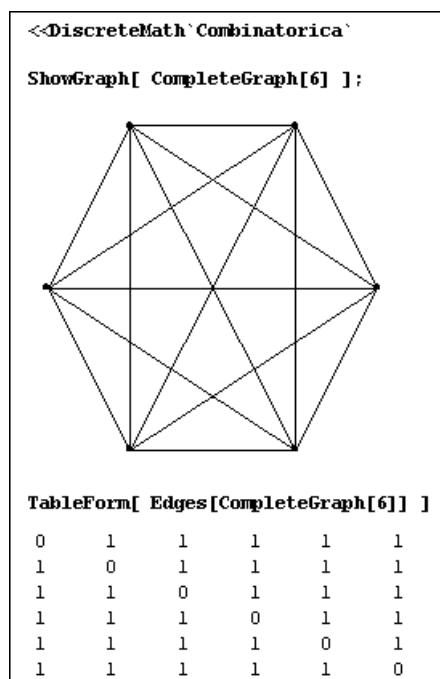


Рис. 9.14. Пример построения полного графа и его таблицы

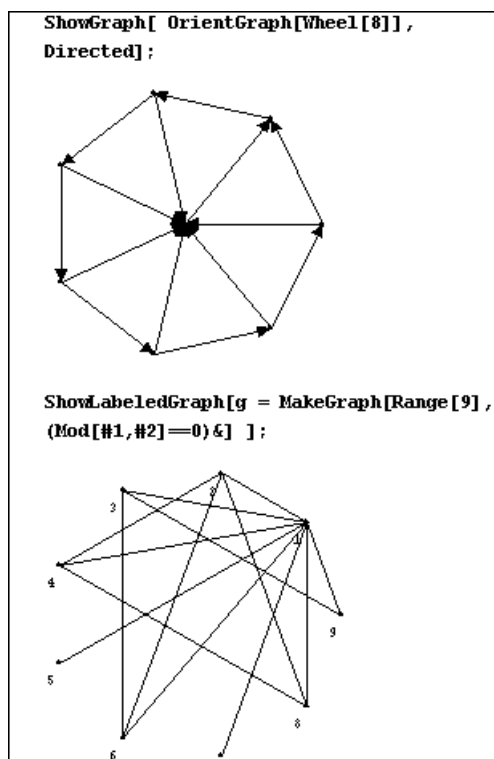


Рис. 9.15. Построение графов:
ориентированного (сверху)
и с маркированными вершинами (снизу)

показывает применение функции **OrientGraph** для построения ориентированного графа, который представляется стрелками. Там же показано применение функции **ShowLabelGraph** (снизу) для построения графа с маркированными числами вершинами. Напомним, что функция **ShowGraph** позволяет наблюдать графы без маркировки вершин.

Построение широко используемой в теории графов диаграммы Хассе (Hasse) иллюстрирует рис. 9.16.

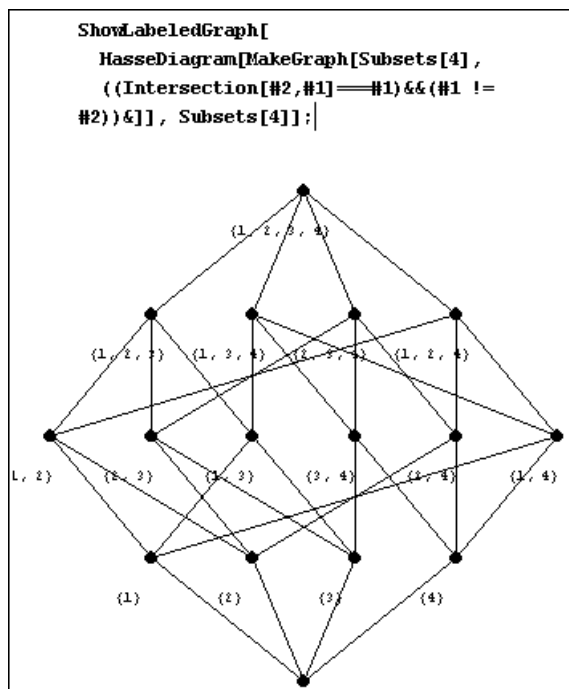


Рис. 9.16. Построение диаграммы Хассе

В целом следует отметить, что набор функций в области создания, визуализации и теории графов весьма представительен, и специалисты в области графов могут найти в этом наборе как типовые, так и уникальные средства.

9.5.5. Функции вычислительной геометрии

В подпакете ComputationalGeometry заданы следующие функции, относящиеся к геометрическим поверхностям:

- **ConvexHull[{x1,y1...},{x2,y2,...},...]** – вычисляет выпуклость оболочки в точках плоскости;

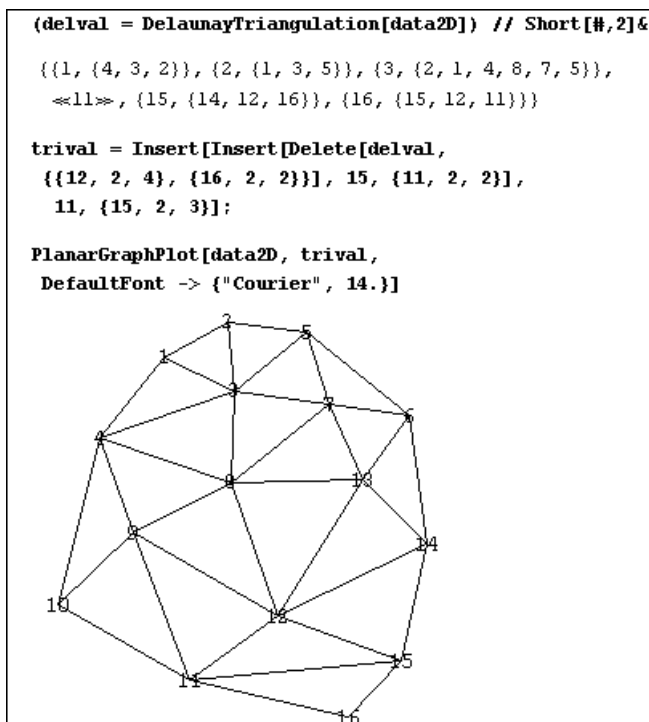


Рис. 9.17. Выполнение триангуляции Делоне

Действие этой функции демонстрирует следующий пример:

```
<<DiscreteMath`DiscreteStep`
DiscreteStep[n - 1] - DiscreteStep[n - 2] // SimplifyDiscreteStep
DiscreteDelta[-1+n]
```

Подпакет Tree задает функции задания и применения жрезовидных структур, именуемых **деревьями**. Вот эти функции:

- **MakeTree[list]** – создает дерево по информации, представленной в списке list;
- **TreeFind[tree,x]** – возвращает позицию наименьшего элемента, превосходящего x в списке list, представляющем дерево.

Действие этих функций поясняют следующие примеры:

```
<<DiscreteMath`Tree`
MakeTree[{e1, e2, e3, e4}]
{{e2,2},{e1,1},{},{},{e3,3},{},{e4,4},{},{}}
tree = MakeTree[{6.05, 4.48, 6.40, 2.46, 3.43, 5.4, 2.0}]
{{4.48,4},{2.46,2},{2.1},{},{},{3.43,3},{},{},{6.05,6},
 {5.4,5},{},{},{6.4,7},{},{}}
TreeFind[tree, 5.1]
```

```
TreeFind[tree, 1]
```

```
0
```

Для визуализации деревьев служат функции:

- **TreePlot[tree]** – строит график дерева tree;
- **ExprPlot[expr]** – строит график, представляющий expr в виде дерева.

Примеры построения графиков деревьев представлены на рис. 9.18. Верхний график построен по данным дерева tree, определенного в приведенных выше примерах, а нижний – по данным случайного дерева.

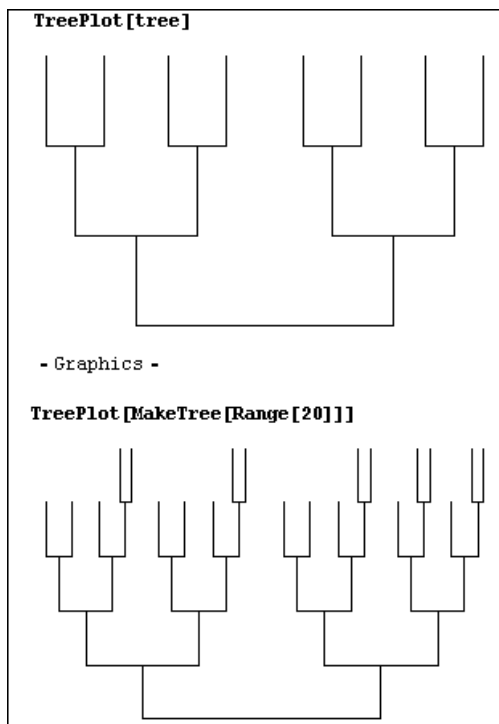


Рис. 9.18. Примеры визуализации деревьев

9.5.6. Функции геометрических расчетов – пакет Geometry

Подпакет Polytopes содержит ряд функций для *регулярных полигонов* (многоугольников). Эти функции позволяют вычислить следующие параметры фигур:

- **NumberOfVertices[p]** – число вершин углов у полигона;
- **NumberOfEdges[p]** – число сторон у полигона;
- **NumberOfFaces[p]** – число граней у полигона;

- **Vertices[p]** – список координат вершин углов у полигона;
- **Area[p]** – площадь полигона при длине каждой стороны, равной 1;
- **InscribedRadius[p]** – радиус вписанной в полигон окружности;
- **CircumscribedRadius[p]** – радиус описывающей полигон окружности.

В функциях наименование полигона p может быть следующим (в скобках дано число сторон):

Digon (2) Triangle (3) Square (4) Pentagon (5) Heagon (6)

Heptagon (7)

Octagon (8) Nonagon(9) Decagon (10) Undecagon (11) Dodecagon (12)

На рис. 9.19 показаны примеры применения некоторых из этих функций и построение крупными точками вершин полигона – пентагона (пятиугольника).

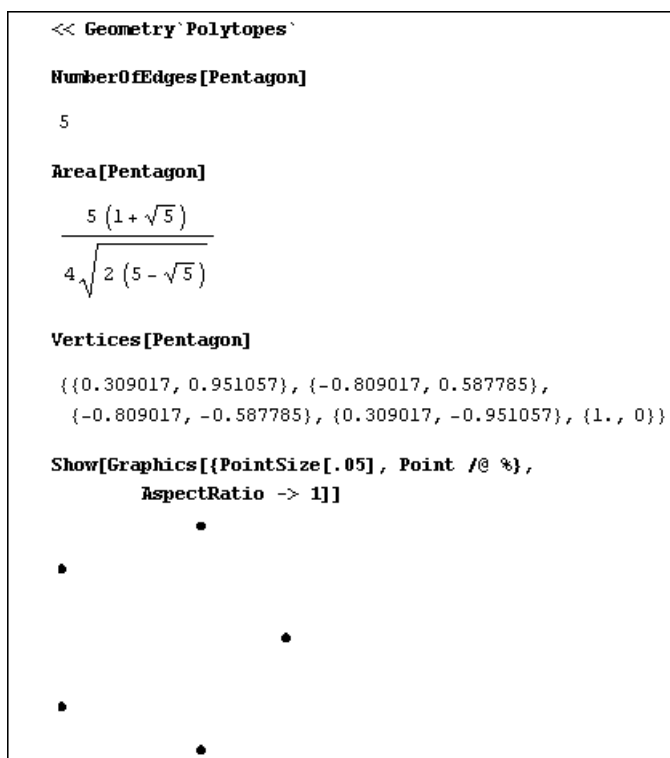


Рис. 9.19. Примеры работы с функциями полигонов

Для объемных фигур – полиэдров имеются следующие функции:

- **NumberOfVertices[p]** – число вершин углов у полиэдра;
- **NumberOfEdges[p]** – число сторон у полиэдра;
- **NumberOfFaces[p]** – число граней у полиэдра;

- **Vertices[p]** – список координат вершин углов у полиэдра;
- **Area[p]** – площадь полиэдра при длине каждой стороны, равной 1;
- **InscribedRadius[p]** – радиус вписанной в полиэдр окружности;
- **CircumscribedRadius[p]** – радиус окружности, описывающей полиэдр;
- **Volume[p]** – объем полиэдра;
- **Dual[p]** – дуальный полиэдр;
- **Schlaflf[p]** – символ полиэдра.

Здесь наименование полиэдра может быть следующим:

Tetrahedron (4) Cube (6) Octahedron (8)
Dodecahedron (12) Icosahedron (20)

Примеры на применение функций полиэдров представлены ниже:

```
<< Geometry`Polytopes`
Volume[Octahedron]

$$\frac{\sqrt{2}}{3}$$

Vertices[Octahedron]
{{0, 0,  $\sqrt{2}$  }, { $\sqrt{2}$  , 0, 0}, {0,  $\sqrt{2}$  , 0},
{0, 0,  $-\sqrt{2}$  }, { $-\sqrt{2}$  , 0, 0}, {0,  $-\sqrt{2}$  , 0}}
Dual[Octahedron]
Cube
InscribedRadius[Octahedron]

$$\frac{1}{\sqrt{6}}$$

CircumscribedRadius[Cube]

$$\frac{\sqrt{3}}{2}$$

```

Для задания *поворота* плоских фигур на заданный угол в подпакете Rotations заданы функции:

- **RotationMatrix2D[theta]** – дает данные поворота матрицы на угол theta;
- **Rotate2D[vec,theta]** – дает данные поворота вектора на угол theta;
- **Rotate2D[vec,theta,{x,y}]** – дает данные поворота вектора на угол theta относительно точки с координатами {x,y}.

Аналогичные функции заданы для получения данных поворота у трехмерных фигур:

- **RotationMatrix3D[psi,theta,phi]** – дает данные поворота матрицы 3D на заданные углы;
- **Rotate3D[vec,psi,theta,phi]** – дает данные поворота вектора в 3D пространстве на заданные углы;
- **Rotate3D[vec,psi,theta,phi,{x,y,z}]** – дает данные поворота вектора в 3D пространстве на заданные углы theta относительно точки с координатами {x,y,z}.

Приведем пример на вращение матрицы:

```
<<Geometry`Rotations`
RotationMatrix3D[Pi, Pi/2, Pi/6]
```

$$\left\{ \left\{ -\frac{\sqrt{3}}{2}, 0, \frac{1}{2} \right\}, \left\{ \frac{1}{2}, 0, \frac{\sqrt{3}}{2} \right\}, \{0, 1, 0\} \right\}$$

MatrixForm[%]

$$\begin{pmatrix} -\frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \end{pmatrix}$$

9.5.7. Расширение в теории чисел – пакет *NumberTheory*

Следующие функции подпакета ContinuedFractions служат для представления чисел в виде **цепных дробей** или для формирования цепной дроби из списков:

- **ContinuedFractionForm[{a0,a1,...}]** – создает цепную дробь из списка {a0,a1,...};
- **Normal[ContinuedFractionForm[{a0,a1,...}]]** – представляет цепную дробь в нормальном виде;
- **Normal[PeriodicForm[{a0,...},{am,...}]]** – преобразование в нормальную форму;
- **Normal[PeriodicForm[{a0,...},{am,...},b]]** – преобразование в нормальную форму с основанием b.

Примеры разложения чисел на цепные дроби:

```
<<NumberTheory`ContinuedFractions`
Table[ Normal[ContinuedFractionForm[Table[1, {n}]]], {n, 9}]
{1, 2, 3/2, 5/3, 8/5, 13/8, 21/13, 34/21, 55/34}
% - N[GoldenRatio]
{-0.618034, 0.381966, -0.118034, 0.0486327, -0.018034, 0.00696601,
-0.00264937, 0.00101363, -0.00038693}
cf = ContinuedFraction[r, 10]
{2, 1, 2, 1, 1, 4, 1, 1, 6, 1}
ContinuedFractionForm[cf]
2 + 1 / (1 + 1 / (2 + 1 / (1 + 1 / (1 + 1 / (4 + 1 / (1 + 1 / (1 + 1 / (6 + 1 / 1))))))))
Normal[%]
1457 / 536
```

Имеются также следующие функции:

- **PeriodicForm**[{a0,...},{am,...}] – дает периодическую форму представления списков;
- **PeriodicForm**[{a0,...},{am,...},b] – дает периодическую форму представления списков с основанием b.

Ниже представлены примеры с применением этих функций:

```
PeriodicForm[RealDigits[ 1/50 ]]
0.02
PeriodicForm[RealDigits[ 1/50, 2 ], 2]
0.000001010001111101011100002
Normal[%]

$$\frac{1}{50}$$

```

Алгоритм *разложения чисел на простые множители* реализуется следующими функциями:

FactorIntegerECM[n] – возвращает наибольший целочисленный простой множитель. Возможны опции FactorSize->q, CurveNumber->b и CurveCountLimit->c.

Примеры на применение этой функции:

```
<<NumberTheory`FactorIntegerECM`
FactorIntegerECM[123456789]
34227
3*5*7*9
945
FactorIntegerECM[945]
189
```

В подпакете NumberTheoryFunctions имеется ряд функций, относящихся к *теории чисел*:

- **SquareFreeQ**[n] – дает True, если n не имеет квадратичного фактора, и False в ином случае;
- **NextPrime**[n] – дает наименьшее простое число, превосходящее n;
- **ChineseRemainderTheorem**[list₁,list₂] – дает наименьшее неотрицательное целое r с Mod[r,list₂]==list₁;
- **SqrtMod**[d,n] – дает квадратный корень d mod_n для нечетного n;
- **PrimitiveRoot**[n] – дает примитивный корень n;
- **QuadraticRepresentation**[d,n] – дает решение {x,y} для уравнения x²+(d y)²=n для нечетного n и положительного d;
- **ClassList**[d] – дает лист неравенств квадратичных форм дискриминанта d для отрицательного и свободного от квадратов целого d в форме 4n+1;
- **ClassNumber**[d] – дает лист неравенств квадратичных форм дискриминанта d;
- **SumOfSquares**[d,n] – дает число представлений для целого числа n как суммы d квадратов;
- **SumOfSquaresRepresentations**[d,n] – дает лист чисел представлений для целого числа n как суммы d квадратов, игнорируя порядок и знак.

Примеры на применение данных чисел даны ниже:

```
<<NumberTheory`NumberTheoryFunctions`
SquareFreeQ[2*3*5*7]
True
SquareFreeQ[50]
False
NextPrime[1000000]
1000003
ChineseRemainderTheorem[{0, 1, 2}, {4, 9, 121}]
244
ChineseRemainderTheorem[Range[16], Prime[Range[16]]]
20037783573808880093
SqrtMod[3, 11]
5
SqrtMod[2, 10^64 + 57]
876504467496681643735926111996546100401033611976777074909122865
PrimitiveRoot[7]
3
QuadraticRepresentation[1, 13]
{3, 2}
ClassList[-19]
{{1, 1, 5}}
ClassNumber[-10099]
25
SumOfSquaresRepresentations[3, 20]
{{-4, -2, 0}, {-4, 0, -2}, {-4, 0, 2}, {-4, 2, 0}, {-2, -4, 0}, {-2, 0, -4},
{-2, 0, 4}, {-2, 4, 0}, {0, -4, -2}, {0, -4, 2}, {0, -2, -4}, {0, -2, 4}, {0, 2, -4},
{0, 2, 4}, {0, 4, -2}, {0, 4, 2}, {2, -4, 0}, {2, 0, -4}, {2, 0, 4}, {2, 4, 0}, {4, -2, 0},
{4, 0, -2}, {4, 0, 2}, {4, 2, 0}}
```

В подпакете PrimeQ вдобавок к функции оценки простых чисел **PrimeQ[n]** ядра имеется ряд функций для работы с **простыми числами**:

- **ProvablePrimeQ[n]** – возвращает True, если n проверено на простоту, и False в ином случае;
- **PrimeQCertificate[n]** – возвращает сертификат о том, что n простое или композитное число;
- **ProvablePrimeQ[n, Certificate->True]** – возвращает сертификат, который может использоваться для проверки чисел на простоту;
- **PrimeQCertificateCheck[check, n]** – возвращает сертификат проверки check простоты или композитности n.

Следующие примеры показывают работу с простыми числами:

```
<<NumberTheory`PrimeQ`
PrimeQ[127]
True
ProvablePrimeQ[127]
True
PrimeQCertificate[127]
{127, 3, {2, {3, 2, {2}}}, {7, 3, {2, {3, 2, {2}}}}}
```

```

ProvablePrimeQ[127, Certificate→True]
{True, {127, 3, {2, {3, 2, {2}}, {7, 3, {2, {3, 2, {2}}}}}}}
PrimeQCertificate[3511, SmallPrime → 1000]
{{CertificatePrime→3511, CertificatePoint→PointEC[2, 1044, 1447, 2135, 3511],
CertificateK→32, CertificateM→3424, CertificateNextPrime→107,
CertificateDiscriminant→-7}, 107, 2, {2, {53, 2, {2, {13, 2, {2, {3, 2, {2}}}}}}}}}

```

Подпакет **PrimitiveElement** содержит всего одну функцию для вычисления *примитивных элементов* множественного алгебраического выражения:

PrimitiveElement[*z*, {*a*₁, *a*₂, ...}] – возвращает список {*b*, {*f*₁, *f*₂, ...}}, где *b* – примитивный элемент расширения рациональных алгебраических чисел *a*₁, *a*₂, ... и *f*₁, *f*₂, ... – полином переменной *z*, представляющей *a*₁, *a*₂, ... как термы примитивного элемента.

Ее действие видно из примера:

```

<<NumberTheory`PrimitiveElement`
PrimitiveElement[z, {Sqrt[2], Sqrt[3]}]
{Root[1 - 10 #12 + #14 &, 4], {- 9 z / 2 + z3 / 2, 11 z / 2 - z3 / 2}}
RootReduce[%[[2]] /. z → %[[1]]]
{√2, √3}

```

В подпакете **Ramanujan** определены следующие функции:

- **RamanujanTau**[*n*] – *n*-ый коэффициент Рамануджана τ -Дирихле серии τ_n ;
- **RamanujanTauGeneratingFunction**[*z*] – генерирует Рамануджана τ -Дирихле серии;
- **RamanujanTauDirichletSeries**[*s*] – Рамануджана τ -Дирихле серия *f*(*s*);
- **RamanujanTauTheta**[*t*] – Рамануджана τ -Дирихле функция *q*(*t*);
- **RamanujanTauZ**[*t*] – Рамануджана τ -Дирихле функция *z*(*t*).

Это довольно редкие функции, представляющие интерес для специалистов в теории чисел. Достаточно подробные определения их даны в справочной базе данных. Ограничимся приведением примеров на их применение:

```

<<NumberTheory`Ramanujan`
RamanujanTau[5]
4830
Sum[RamanujanTau[n] zn, n, 5]
z - 24z2 + 252z3 - 1472z4 + 4830z5
RamanujanTauGeneratingFunction[.1]
0.00610209
RamanujanTauGeneratingFunction[.99]
4.10287803703×10-1673
RamanujanTauDirichletSeries[6 + 9.22I]
0.00040309- 0.00239013 I
z = RamanujanTauZ[9.22]
0.00242388
theta = RamanujanTauTheta[9.22]
1.40372043366323
z Exp[-I theta]
0.00040309- 0.00239013 I

```


Подпакет Rationalize расширяет возможности представления чисел в рациональном виде. Он содержит определения следующих функций:

- **ProjectiveRationalize[{x0,x1,...,xn}]** – возвращает лист целых чисел, дающих рационализацию для чисел заданного списка;
- **ProjectiveRationalize[{x0,x1,...,xn},prec]** – возвращает лист целых чисел, дающих рационализацию с погрешностью, не более 10^{prec} ;
- **AffineRationalize[{x0,x1,...,xn}]** – возвращает список рациональных приближений для чисел заданного списка;
- **AffineRationalize[{x0,x1,...,xn},prec]** – возвращает список рациональных приближений для чисел заданного списка, вычисленных с погрешностью не более 10^{prec} .

Встроенная в ядро функция дает рациональное представление для одиночных вещественных чисел. Приведенные функции выполняют рационализацию для списков чисел. Примеры на их применение представлены ниже:

```
<<NumberTheory`Rationalize`
Rationalize[N[3 Pi], 6]/ Rationalize[N[11 Pi], 6]

$$\frac{9}{34}$$

AffineRationalize[{N[3 Pi], N[11 Pi]}, 6]

$$\left\{ \frac{1065}{113}, \frac{3905}{113} \right\}$$

ProjectiveRationalize[{N[3 Pi], N[11 Pi]}]
{3, 11}
```

Подпакет Recognize содержит определение одноименной с ним функции в двух формах:

- **Recognize[x,n,t]** – находит полином степени большей n по t, такой, что x дает его корень.
- **Recognize[x,n,t,k]** – находит полином степени большей n по t, такой, что x дает его корень и с предельной степенью полинома k.

Действие этой функции поясняют примеры:

```
<< NumberTheory`Recognize`
NSolve[2 x^3 - x + 5 == 0]
{{x→-1.4797},{x→0.739852 -1.06871 i},{x→0.739852 +1.06871 i}}
sol = First[x /. %]
-1.4797
Recognize[sol, 3, t]
-5 + t - 2t^3
Recognize[sol, 2, t]
-225599 - 146496t + 4032t^2
Recognize[N[Sqrt[3^(2/5)]], 5, t]
-3 + t^5
Recognize[N[Sqrt[3^(2/5)]], 5, t, 10]
-14625 + 11193t + 328t^2 + 88t^3 + t^4
```

Подпакет SiegelTheta содержит еще одну редкую функцию:

SiegelTheta[z,s] – возвращает значение Тета-функции Зигеля.

Примеры вычисления этой функции даны ниже:

```
<< NumberTheory`SiegelTheta`
SiegelTheta[1+I,2+I, 2+I,-1+4I, 1.2, 2.3+.3I]
0.973715- 0.000297048 I
Sum[E^(Pi I t1,t2.1+I,2+I, 2+I,-1+4I.t1,t2 + 2 Pi I t1,t2 . 1.2,
2.3+.3I), t1, -10, 10, t2, -10, 10]
0.973715- 0.000297048 I
```

В заключительной части этого примера дано вычисление SiegelTheta-функции по ее исходному определению.

Mathematica представляет алгебраические численные поля через Root-объекты. При этом для любых f и k , для которых $\theta = \text{Root}[f, k]$, можно задать алгебраическое число $C_0, C_1\theta, \dots, C_l\theta^l$. Для этого служит функция:

Algebraic[$f, \{c_0, C_1, \dots, C_l\}, k$] – представляет алгебраическое число $c_0, C_1\theta, \dots, C_l\theta^l$, где $\theta = \text{Root}[f, k]$.

Пакет AlgebraicNumberFields содержит около трех десятков функций для работы с алгебраическими числовыми полями. Ввиду редкости применения специфических математических понятий алгебраических числовых полей рассмотрение этих функций нецелесообразно. С ними и примерами их применения можно познакомиться по справке.

9.5.8. Функции численных расчетов – расширение NumberMath

Расширение NumberMath содержит множество полезных функций для тех, кто имеет дело с численными расчетами. В их числе функции для выполнения высокоточных аппроксимаций рациональными функциями (уже были рассмотрены в Главе 6), численного интегрирования и дифференцирования, вычисления пределов функций, решения уравнений, разложения в ряд и т.д. Ниже описано подавляющее большинство функций этого расширения. Исключены лишь отдельные функции, представляющие ограниченный интерес и несложные для самостоятельного изучения (в подпакетах Butcher, Microscope и ComputerArithmetic).

В подпакете BesselZeros определены функции, дающие список аргументов функций Бесселя в их первых n нулевых точках: **BesselJZeros**[μ, n], **BesselYZeros**[μ, n], **BesselJPrimeZeros**[μ, n], **BesselYPrimeZeros**[μ, n] и др. Ввиду очевидности использования функций этого класса, ограничимся парой примеров их применения:

```
<< NumericalMath`BesselZeros`
BesselJZeros[0, 5]
{2.40483, 5.52008, 8.65373, 11.7915, 14.9309}
BesselJYZeros[2, 6/5, 3, WorkingPrecision -> 30]
{15.8066224441765790253647015575 , 31.4655600915368446462653148738
, 47.1570167108650317281338334229 }
```

Подпакет InterpolateRoot дает средства для ускоренного и более точного поиска корней уравнений по сравнению с соответствующими функциями ядра. Дости-

гается это за счет применения интерполяции функции, корни которой ищутся. Подпакет задает функцию:

InterpolateRoot[f,{x,a,b}] – находит корень функции f в интервале x от a до b .

Вместо функции f можно задавать уравнение eqn. Возможны опции AccuracyGoal->Automatic, MaxIterations->15, WorkingPrecision->\$MachinePrecision и ShowProgress->False (указаны принятые по умолчанию значения).

Примеры применения данной функции (n-число итераций):

```
<<NumericalMath`InterpolateRoot`
<<NumericalMath`InterpolateRoot`
n = 0; FindRoot[n++; Exp[x] == 2, {x, 0, 1}]
{ (x->0.693147) }
n
n
InterpolateRoot[Exp[x] == 2, {x, 0, 1},
  ShowProgress -> True,
  WorkingPrecision -> 30]
{0,0.581976706869326424385}
{20.9546,0,-0.12246396352039524100}
{1,0.70193530378827640144433707647233867351054}
{20.9546,20.0666,0.0130121629575404389120930392357}
{3,0.69320657720652631652899857937263679885792}
{20.9546,20.0666,0.0000624807887477135488047731126364}
{6,0.69314719326039338416187260582372987023825}
{21,20.0666,1.26443483693584888038460387601×10-8}
{12,0.69314718055994511945782244695590259222303570504860}
{x->0.69314718056}
n
20
```

Иногда важно найти не приближенное значение корня, а уточнить интервал, в котором он находится. В подпакете IntervalRoots для этого используется ряд известных методов, реализованных следующими функциями:

- **IntervalBisection[f,x,int,eps]** – находит корень функции $f(x)$ путем уточнения исходного интервала int с заданной погрешностью eps методом бисекции;
- **IntervalSecant[f,x,int,eps]** – находит корень функции $f(x)$ путем уточнения исходного интервала int с заданной погрешностью eps методом секущей;
- **IntervalBisection[f,x,int,eps]** – находит корень функции $f(x)$ путем уточнения исходного интервала int с заданной погрешностью eps методом Ньютона (касательной).

Во всех функциях можно опциями задать максимальное число рекурсий (MaxRecursion) и погрешность (WorkingPrecision). Примеры применения этих функций даны ниже:

```
<<NumericalMath`IntervalRoots`
IntervalBisection[Sin[x], x, Interval[{2., 8.}], .1]
Interval[{3.125,3.21875},{6.21875,6.3125}]
```

```
IntervalBisection[Sin[x], x, Interval[{2., 8.}], .01,
  MaxRecursion -> 10]
Interval[{3.13672, 3.14258}, {6.27734, 6.2832}]
IntervalSecant[Sin[x], x, Interval[{2., 8.}], .01]
Interval[{3.14159, 3.1416}, {6.28316, 6.28321}]
IntervalSecant[Sin[x], x, Interval[{2., 8.}], .01]
Interval[{3.14159, 3.1416}, {6.28316, 6.28321}]
IntervalBisection[Sin[x], x, Interval[{2, 8}], .1,
  WorkingPrecision -> Infinity]
Interval[{ $\frac{25}{8}$ ,  $\frac{103}{32}$ }, { $\frac{199}{32}$ ,  $\frac{101}{16}$ }]
```

Встроенная в ядро функция **NIntegrate** вычисляет определенные интегралы при известной подынтегральной функции. Однако нередко, например, при экспериментах, такая функция задается таблицей или списком значений. В подпакете ListIntegrate имеются функции для решения этой задачи – *табличного интегрирования*:

- **ListIntegrate[{y₁, y₂, ..., y_n}, h]** – возвращает численное значение интеграла для функции, заданной списком ординат y_i при заданном шаге h по x;
- **ListIntegrate[{y₁, y₂, ..., y_n}, h, k]** – возвращает численное значение интеграла для функции, заданной списком ординат y_i при заданном шаге h по x, используя k точек каждого подинтервала;
- **ListIntegrate[{x₁, y₁}, {x₂, y₂}, ..., {x_n, y_n}, k]** – возвращает численное значение интеграла для функции, заданной списком координат {x_i, y_i}, используя k точек для каждого подинтервала.

Примеры применения данной функции:

```
<<NumericalMath`ListIntegrate`
data = Table[ n^2, {n, 0, 7}]
{0, 1, 4, 9, 16, 25, 36, 49}
ListIntegrate[data, 1]
 $\frac{343}{3}$ 
ListIntegrate[{{0, 0}, {1, 1}, {2, 4}, {5, 25}, {7, 49}}, 2]
 $\frac{241}{2}$ 
```

При проведении интегрирования для данных, заданных таблично, можно использовать интерполяцию:

```
app = ListInterpolation[data, 0, 7]
Integrate[app[x], x, 0, 7]
 $\frac{343}{3}$ 
Integrate[Interpolation[0, 0, 1, 1, 2, 4, 5, 25, 7, 49,
  InterpolationOrder -> 1][x], x, 0, 7]
 $\frac{241}{2}$ 
```

В подпакете Nlimit определена также функция

Nlimit[expr,x->x0]

для численного вычисления пределов выражений expr (см. примеры ниже):

```
<<NumericalMath`NLimit`
NLimit[Zeta[s] - 1/(s-1), s->1]
0.577216
N[EulerGamma]
0.577216
```

В этом субпакете задано также вычисление бесконечных сумм Эйлера **EulerSum[f,{i,imin,Infinity}]**. Например:

```
EulerSum[(-1)^k/(2k + 1), k, 0, Infinity]
0.785398
EulerSum[(-1)^k/(2k + 1), k, 0, Infinity,
WorkingPrecision->40, Terms->30, ExtraTerms->30]
0.78539816339744830961566084579130322540
% - N[Pi/4, 40]
-2.857249565×10-29
```

Имеется также функция вычисления в численном виде производной:

- **ND[f,x,x0]** – вычисляет первую производную $f(x)$ в точке x_0 ;
- **ND[f,{x,n},x0]** – вычисляет n -ую производную $f(x)$ в точке x_0 .

Пример вычисления производной:

```
ND[Exp[Sin[x]], x, 2]
-1.03312
```

В некоторых случаях вычисления могут быть ошибочными. Тогда следует использовать опции – особенно выбора метода Method. Помимо метода по умолчанию EulerSum можно использовать **NIntegrate** (метод интегрирования Cauchy).

В подпакете NResidue имеется функция вычисления остатка **Residue[expr,{x,x0}]** в точке $x=x_0$:

```
<<NumericalMath`NResidue`
NResidue[1/z, {z, 0}]
1.
f = 1/Expand[(z-1.7) (z+.2+.5 I) (z+.2-.5 I)]
1/((-0.493 - 7.20994×10-18 i) -
(0.39+0. i) z - (1.3+0. i) z2 + z3)
Residue[f, {z, 1.7}]
0
NResidue[f, {z, 1.7}]
0.259067 - 2.71051×10-17 i
1/((z+.2+.5 I) (z+.2-.5 I)) /. z -> 1.7
1
((0.2 - 0.5 i) + 1.7) ((0.2 + 0.5 i) + 1.7)
Options[NResidue]
{ (Radius -> 1/100), (WorkingPrecision -> 15.9546),
```

```
(PrecisionGoal -> Automatic) }
```

Обратите внимание на возможные опции для этой функции в последнем примере.

Подпакет **NSeries** вводит функцию:

NSeries[f,{x,x0,n}] – дает численный ряд, аппроксимирующий функцию $f(x)$ в окрестности точки $x=x_0$, включая термы от $(x-x_0)^{-n}$ до $(x-x_0)^n$.

Примеры применения данной функции:

```
<< NumericalMath`NSeries`
NSeries[Sin[x], {x, -2, 2}]


$$\frac{0.00126291 + 6.81123 \times 10^{-17} i}{(x+2)^2} + \frac{0.0000825688 - 1.4817 \times 10^{-17} i}{x+2} -$$


$$(0.90932 + 4.291 \times 10^{-17} i) - (0.416148 - 1.51544 \times 10^{-17} i) (x+2) +$$


$$(0.454649 - 9.84211 \times 10^{-17} i) (x+2)^2 + O[x+2]^3$$

Rationalize[Chop[%]]

$$\frac{0.00126291}{(x+2)^2} + \frac{0.0000825688}{x+2} - 0.90932 -$$


$$0.416148 (x+2) + 0.454649 (x+2)^2 + O[x+2]^3$$

Rationalize[Chop[NSeries[Log[x], {x, 1, 5},
Radius -> 1/8]]]

$$(x-1) - \frac{1}{2} (x-1)^2 + \frac{1}{3} (x-1)^3 - \frac{1}{4} (x-1)^4 + \frac{1}{5} (x-1)^5 + O[x-1]^6$$

Rationalize[Chop[NSeries[Exp[x], {x, 0, 5},
WorkingPrecision -> 40, Radius -> 1/8]]]

$$+ x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O[x]^6$$

Rationalize[Chop[NSeries[Exp[x], {x, 0, 5},
Radius -> 4]]]

$$\frac{1.693 \times 10^{-9}}{x^5} + + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O[x]^6$$

Chop[NSeries[Zeta[s], {s, 1, 3}]]

$$\frac{1.}{s-1} + 0.577216 + 0.0728158 (s-1) -$$


$$0.00484518 (s-1)^2 - 0.000342306 (s-1)^3 + O[s-1]^4$$

```

Функция **NIntegrate**, имеющаяся в ядре системы Mathematica, реализует метод интегрирования Гаусса-Кронрода. Еще одним известным методом интегрирования является *метод Ньютона-Котесса*, сводящий интегрирование к вычислению ординат функции, умноженных на весовые множители.

- **NewtonCotesWeights[n,a,b]** – возвращает список весовых коэффициентов и абсцисс узловых точек $\{w_i, x_i\}$ для квадратуры на интервале от a до b ;
- **NewtonCotesError[n,f,a,b]** – возвращает погрешность формулы Ньютона-Котесса для заданной функции f .

Примеры применения этих функций представлены ниже:

```
<<NumericalMath`NewtonCotes`
NewtonCotesWeights[5, 0, 10]
{{0, 7/9}, {5/2, 32/9}, {5, 4/3}, {15/2, 32/9}, {10, 7/9}}
NewtonCotesError[5, f, 0, 10]
15625 f(6)
3024
NewtonCotesWeights[5, -0, 10, QuadratureType -> Open]
{{1375/576}, {3, 125/144},
{5, 335/96}, {7, 125/144}, {9, 1375/576}}
NewtonCotesError[5, f, 0, 10, QuadratureType -> Open]
5575 f(6)
1512
```

Обратите внимание на то, что приведенные формулы готовят данные для численного интегрирования методом Ньютона-Котесса, но не выполняют самого интегрирования.

9.5.9. Функции работы со звуком пакета *Miscellaneous*

Расширение *Miscellaneous* в переводе на русский язык означает «всякая всячина». И впрямь, большинство функций этого пакета на первый взгляд не имеет прямого отношения к математическим расчетам. Однако, как сказать! Этот пакет представляет систему *Mathematica* в особом свете – как систему, имеющую далеко нестандартные средства синтеза звука и представления информации самого общего вида, относящейся к датам, времени, геодезии, картографии и т.д. Физики, химики, географы и даже музыканты могут найти в этом пакете средства, полезные им при обработке различной жизненно важной информации вида на ПК.

Подпакет *Audio* встроенного пакета расширения *Miscellaneous* служит для генерации стандартных звуковых сигналов разной формы, частоты и длительности, модуляции сигналов по амплитуде и по частоте и считывания звуковых файлов с дисков. Для создания звуковых объектов служит функция:

- **Waveform[type,freq,dur]** – создает звуковой сигнал формы **type** с частотой **freq** (в Герцах) и длительностью **dur** (в секундах). Возможны следующие формы сигнала: *Sinusoid* – синусоидальный, *Triangle* – треугольный, *Square* – прямоугольный и *Sawtooth* – пилообразный;
- **Waveform[type,freq,dur,Overtones->n]** – создает звуковой сигнал формы **type** с частотой **freq** (в Герцах) и длительностью **dur** (в секундах), имеющий **n** гармоник.

Приведенный на рис. 9.20 пример дает генерацию прямоугольного сигнала частотой 1000 Гц и длительностью 0.5 с. Следует обратить внимание на то, что созданный звуковой объект проигрывается и показывается после команды **Show**.

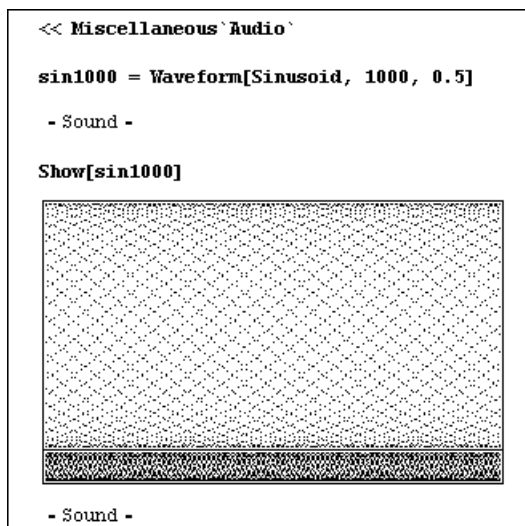


Рис. 9.20. Создание и просмотр звукового объекта – синусоидального сигнала

Звуковой объект, как отмечалось, ассоциируется с графическим объектом. К сожалению, явной связи между осциллограммой звукового сигнала и его графическим образом нет. Более того, вид графического объекта сильно зависит от компьютерной платформы, на которой установлена система Mathematica. Так что графический звуковой объект – это просто некий условный графический образ звукового сигнала.

Рисунок 9.21 показывает генерацию прямоугольного сигнала с двумя гармониками. Здесь используется опция `Overtones->2`. Ее нельзя применять к синусоидальному сигналу, поскольку он принципиально не имеет гармоник.

Когда используется опция `Overtones`, функция **Waveform** использует ряд Фурье для создания высших гармоник, обогащающих тембр звука. При этом возможно изменение числа гармоник. Возможно также создание сигнала с заданными номерами гармоник и заданной их амплитудой. Для этого служит функция:

ListWaveform[`{{n1,a1},{n2,a2},...},freq,dir`] – создает звуковой объект с основной частотой гармоники `freq` и длительностью `dir`, содержащий дополнительные частоты с кратностями `ni` и амплитудами `ai`.

На рис. 9.22 представлен пример создания звукового объекта сложного типа, содержащего ряд частотных составляющих. Данные представлены списком

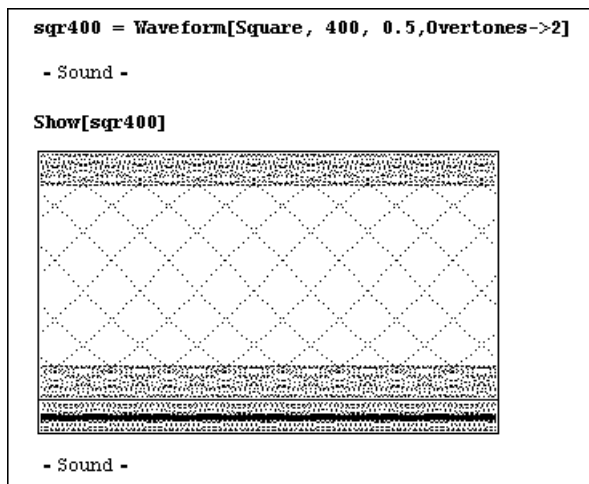


Рис. 9.21. Генерация прямоугольного сигнала с двумя гармониками

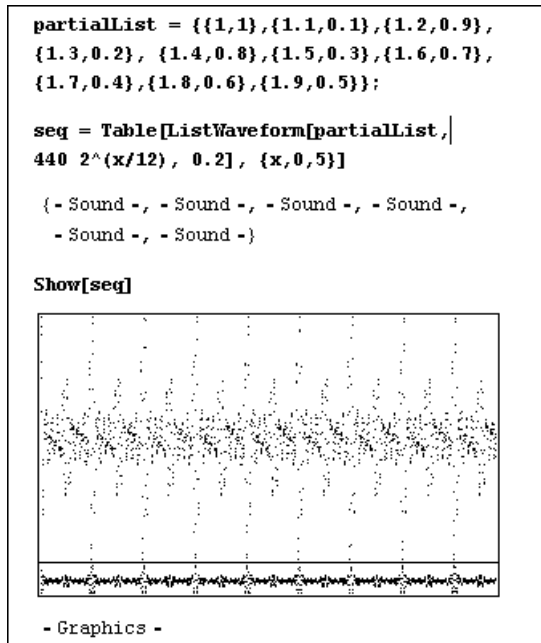


Рис. 9.22. Генерация многочастотного сигнала

patrialList. С помощью функции **Table** подготовлен объект, содержащий шесть звуковых подобъектов.

Для создания звуковых объектов, порождающих звук с амплитудной и частотной модуляцией, служат следующие функции:

- **AmplitudeModulation**[f_c, f_m, m, dur] – создает амплитудно-модулированный синусоидальный сигнал с основной частотой f_c , частотой модуляции f_m , коэффициентом модуляции m и длительностью dur . Опция `RingModulation -> True` позволяет получить ring-модуляцию;
- **FrequencyModulation**[$f_c, \{f_m, pd\}, dur$] – создает частотно модулированный (FM) синусоидальный сигнал с основной частотой f_c , модулированный по частоте сигналом с частотой модуляции f_m с пиком девиации pd , с длительностью dur .

Рисунок 9.23 демонстрирует создание звукового объекта с амплитудной и частотной модуляцией. Обратите внимание, что объект показывается сразу, поскольку в состав его выражения включена команда **//Show**.

Для создания сложных сигналов с частотной модуляцией служит функция:

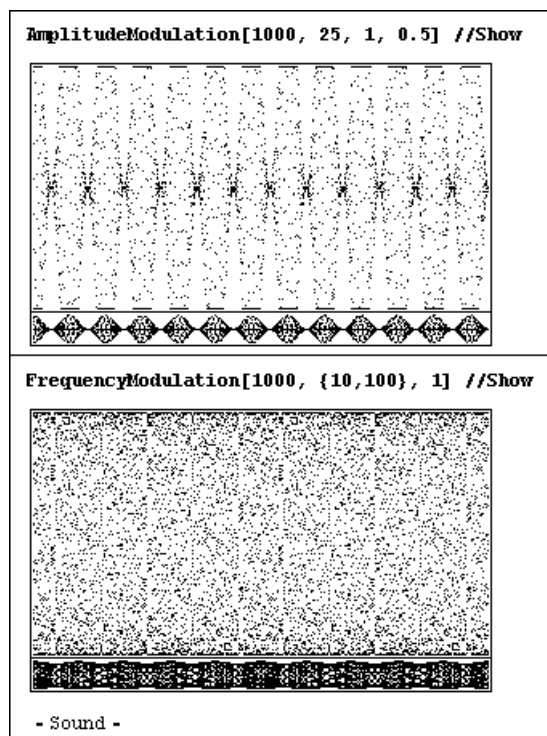


Рис. 9.23. Генерация звуковых объектов с амплитудной (сверху) и частотной (снизу) модуляцией

FrequencyModulation[*fc*,{{*f1*,*pd1*},{*f2*,*pd2*},...},*dur*] – создает частотно модулированный (FM) синусоидальный сигнал с основной частотой *fc* и каскадом частот модуляции *fmi* с пиками девиации *pd_i* и длительностью *dur*. Опция *ModulationType*->*Parallel* создает звуковой сигнал с параллельными частотами модуляции, а опция *ModulationType*->*Cascade* – последовательными частотами модуляции.

Рисунок 9.24 показывает создание и воспроизведение композитного звукового сигнала. Он представлен списком объектов {*sq*,*s2*,*s3*}.

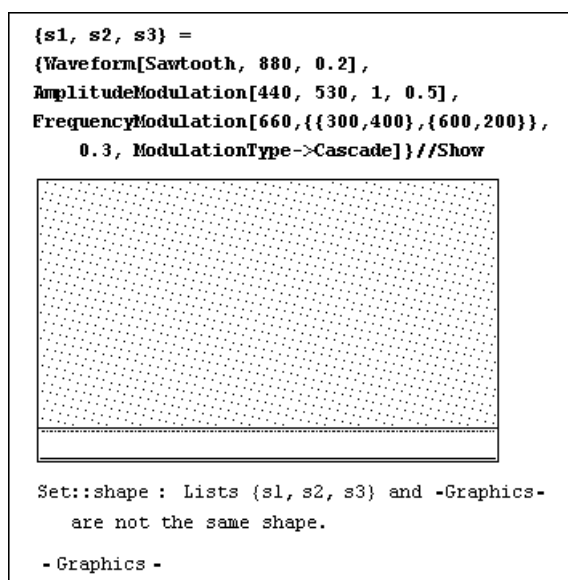


Рис. 9.24. Генерация композитного звукового сигнала

Для считывания звуковых файлов с магнитного диска служит функция:

ReadSoundfile[«*soundfile*»] – опознает файлы разного формата и конвертирует их в список с целыми числами от -32768 до +32767. Опция *PrintHeader*->*True* позволяет вывести отчет о звуковом файле. Поддерживаются следующие форматы звуковых файлов: NeXT/Sun, WAVE и AIFF.

Разумеется, для считывания звукового файла он должен быть в текущей директории или его имя должно точно указывать местоположение файла. Файлы могут содержать звуки речи и музыки и отображаются соответствующим графическим образом (как приводилось выше).

Подпакет *Music* как бы продолжает рассмотренный выше подпакет поддержки звуковых возможностей системы *Mathematica*. Он задает функцию последовательного воспроизведения листа, содержащего отдельные звуки:

Scale[ihist,freq,dur] – генерирует звуковой объект, представленный данными списка *ihist* музыкальных интервалов с частотой *freq* и длительностью *dur*.

Пример воспроизведения отрывка мелодии Just Major представлен на рис. 9.25. Помимо фрагмента JustMajor имеется еще с десяток фрагментов, которые поставлены с системой Mathematica (QuarterTone, MeanMajor, MeamMinor, SixthTone, JustMinor и др.). Все они могут воспроизводиться функцией Scale.

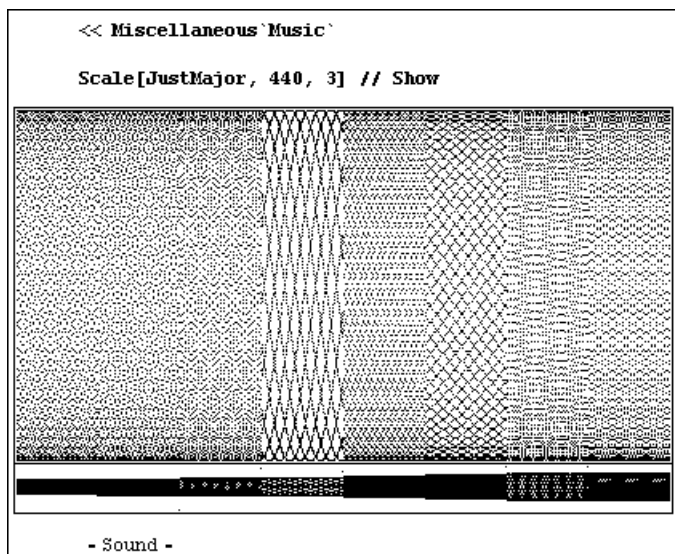


Рис. 9.25. Пример воспроизведения музыкального фрагмента

Кроме того, есть ряд функций преобразования:

- **HertzToCents[flist]** – преобразует лист частот в лист музыкальных интервалов (cents);
- **CentsToHertz[ihist]** – преобразует лист музыкальных интервалов в лист частот (Гц), начиная с частоты 440 Гц;
- **CentsToHertz[ihist,f]** – преобразует лист музыкальных интервалов в лист частот (Гц), начиная с заданной частоты *f*.

Примеры преобразования даны ниже:

```
<< Miscellaneous`Music`
HertzToCents[{200, 400, 1000}]
{1200., 1586.31}
alist = Table[ N[440 2^(i/12)], {i, 0, 12, 2}]
{440., 493.883, 554.365, 622.254, 698.456, 783.991, 880.}
HertzToCents[alist]
{200., 200., 200., 200., 200., 200.}
CentsToHertz[{0, 1000}]
{440., 783.991}
```

```
CentsToHertz[Range[0, 1600, 200], 880]
{880., 987.767, 1108.73, 1244.51, 1396.91, 1567.98, 1760., 1975.53, 2217.46}
HertzToCents[{Aflat4, Eflat5}]
{700.}
```

Описанные возможности синтеза музыки являются, скорее, отдающими дань моде на мультимедиа, чем нужными на практике. Так, время подготовки музыкального объекта довольно значительно (до нескольких секунд на ПК с процессором Pentium III с частотой 600 МГц). Так что они годятся только для создания простейших музыкальных звуков, которыми можно сопровождать некоторые учебные программы.

9.5.10. Функции для работы с географическими объектами

В пакете расширений Miscellaneous есть ряд подпакетов, содержащих *функции времени и даты*. Так, в подпакете Calendar сосредоточены вычисления, относящиеся к календарным датам:

- **Data0Week**[**{year, month, day}**] – вычисляет день недели по заданному году, месяцу и числу;
- **DaysBetween**[**{year1, month1, day1}, {year2, month2, day2}**] – вычисляет число суток между двумя датами (второй и первой);
- **DaysPlus**[**{year, month, day}, n**] – дает дату **n**-го дня после заданной начальной даты.

Во всех этих функциях возможна опция Calendar->cal, где cal задает тип календаря, например Григорианский (Gregorian) или Юлианский (Julian). Имеется также функция смены календаря:

CalendarChange[**{year, month, day}, cal1, cal2**] – преобразует заданную дату из одного календаря в другой.

Примеры на эти вычисления:

```
<<Miscellaneous`Calendar`
DayOfWeek[{2003, 2, 7}]
Friday
DaysBetween[{2002, 1, 1}, {2003, 1, 1}]
365
DaysPlus[{2003, 1, 1}, 500]
{2004, 5, 15}
DaysBetween[{2002, 1, 1}, {2003, 1, 1}, Calendar -> Julian]
365
CalendarChange[{1992, 2, 29}, Gregorian, Julian]
{1992, 2, 16}
CalendarChange[{1992, 2, 29}, Gregorian, Islamic]
{1412, 8, 25}
CalendarChange[{1, 1, 1}, Islamic, Julian]
{622, 7, 16}
```

В подпакете `CityData` можно найти функции, позволяющие найти местное время для большинства крупных городов мира: **`CityData[city,datatype]`**, **`CityData[city]`** и **`CityData[datatype]`**. Весьма обширный список городов можно найти в справке по системе. Приведем данные по Москве:

```
<< Miscellaneous`CityData`
CityData["Moscow", CityPosition]
{{55 , 45 }, {37 , 35 }}
CityData["Moscow"]
{{ CityPosition, {{55 , 45 }, {37 , 35 }}}}
```

Разумеется, большинство стран представлено своими столицами и наиболее крупными городами. Однако (см. далее) есть возможность пополнять базу данных по городам.

В этом же подпакете есть функции для вычисления расстояния между городами:

- **`CityDistance["city1", "city"]`** – возвращает расстояние между двумя указанными городами;
- **`CityDistance["city1", "city", CityDistanceMethod->Method]`** – возвращает расстояние между двумя указанными городами со спецификацией метода вычислений (`SphericalDistance` или из подпакета `Geodesy`).

Примеры на вычисление расстояний между городами даны ниже:

```
<< Miscellaneous`CityData`
CityData["Moscow", CityPosition]
{{55 , 45 }, {37 , 35 }}
CityData["Moscow"]
{{ CityPosition, {{55 , 45 }, {37 , 35 }}}}
```

```
CityDistance["Washington", "Moscow"]//N
N [7820.84]
```

База данных по городам может пополняться. В приведенных ниже примерах из справки показано пополнение базы данных координатами города Шампейн (`Champaign`), в котором расположена корпорация `Wolfram Research Inc.`:

```
CityPosition[{"Champaign", "USA", "IL"}] =
{{40, 7, 5}, {-88, -14, -48}};
CityData["Champaign", CityPosition]
{{40 , 7 , 5 }, {-88 , -14 , -48 }}
```

В подпакете `Geodesy` есть функции, вычисляющие расстояние между двумя точками с учетом выпуклости Земли:

- **`SphericalDistance[pos1,pos2]`** – вычисляет расстояние между двумя позициями, в предположении, что Земля – идеальный шар (сфера);
- **`SpheroidalDistance[pos1,pos2]`** – вычисляет расстояние между двумя позициями, в предположении, что Земля – приплюснутый шар (сфероид).

Примеры на вычисления по этим функциям представлены ниже:

```
SphericalDistance[0, 0, 45, 45] //N
6671.7
SpheroidalDistance[0, 0, 45, 45] //N
6662.47
```

% – %%

–9.23014

Для пополнения базы данных городов можно использовать функцию

AppendTo[\$CityFields, CityPopulation]

Например, для Вашингтона

```
AppendTo[$CityFields, CityPopulation];
```

```
CityPopulation[{"Washington", "USA", "DC"}] = 638000;
```

```
CityData["Washington"]
```

```
{{CityPosition, {{38, 53, 42}, {-77, -2, -12}}}, {CityPopulation, 638000}}
```

В пакете Miscellaneous имеется также база данных по странам мира. Доступ к ней открывает подпакет WordData. Для этого имеется функция

WordData["Страна"].

возвращающая список координат конечных отрезков прямых, которые задают контурный график – карту заданной страны. Например, данные по Азербайджану можно получить следующим образом:

```
<<Miscellaneous`WorldData`
```

```
WorldData["Azerbaijan"]
```

```
{{{2378, 2689}, {2374, 2770}, {2344, 2806}, {2330, 2770}, {2378, 2689}}, {{2361, 2849},  
{2419, 2781}, {2472, 2701}, {2445, 2823}, {2509, 2778}, {2473, 2871},  
{2510, 2915}, {2414, 3024}, {2307, 2933}, {2376, 2888}, {2361, 2849}}}
```

Попробуйте сами найти данные по России (Russia); мы их не приводим ввиду громоздкости списка, что вполне естественно, поскольку Россия – крупнейшая страна мира и имеет самую длинную границу (контурную линию) с многочисленными изломами.

В подпакете WorldNames имеется список континентов: Africa, Asia, Europa, MiddleEast, NorthAmerica, Oceania, SouthAmerica и World (весь мир). Например, так можно узнать, какие страны расположены в Океании:

```
<<Miscellaneous`WorldNames`
```

```
Oceania // InputForm
```

```
"Indonesia", "Papua New Guinea", "Fiji", "Australia", "New Zealand"
```

Наиболее эффектными являются возможности подпакета WorldPlot, функции которого позволяют строить карты любой страны или всего мира:

- **WorldPlot[countrylist]** – построение карты страны по списку ее данных;
- **WorldPlot[countrylist, RandomColors]** – построение карты страны по списку ее данных с раскраской случайными цветами;
- **WorldPlot[countrylist, RandomGrays]** – построение карты страны по списку ее данных с раскраской случайными оттенками серого цвета;
- **WorldPlot[{countrylist, colorfunc}]** – построение карты страны по списку ее данных с раскраской по списку colorfunc;
- **WorldPlot[{countrylist, colorlist}]** – построение карты страны по списку ее данных с раскраской по списку colorlist.

На рис. 9.26 показано построение контурной карты России (верхний рисунок) и цветной карты мира (World). Окраска достигается применением директивы RandomColors.

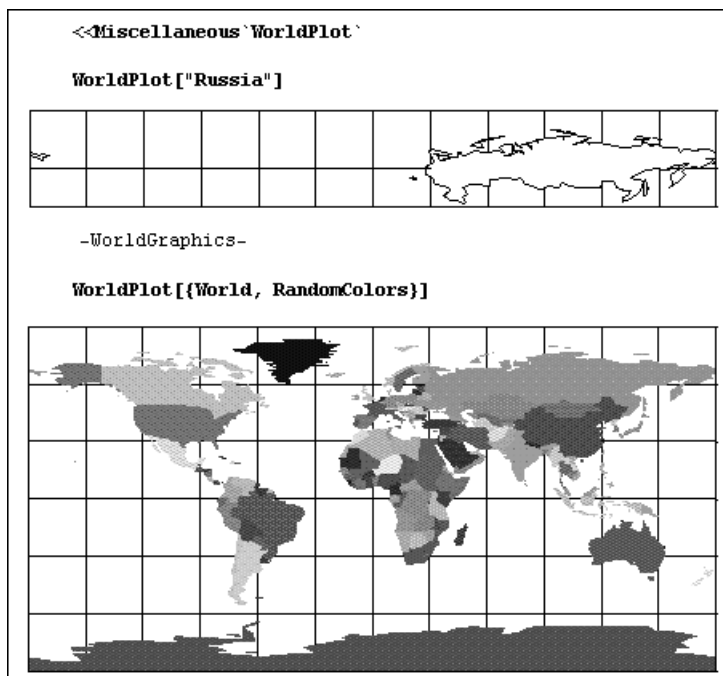


Рис. 9.26. Контурная карта России и цветная карта мира

Следующий рисунок (рис. 9.27) показывает возможность композиционного изображения картографических изображений. На нем построена карта Америки и особо выделена (темным цветом) карта Канады. Здесь для цветовых выделений использована директива GrayLevel, позволяющая задавать заданную степень густоты серого цвета.

Карты могут строиться в различной проекции: Albers, Equirectangular, LambertAzimuthal, LambertCylindrical, Mercator, Mollweide и Sinusoidal. Для этого используется опция Projection->Имя_проекции.

На рис. 9.28 представлена цветная карта всего мира, заданная в синусоидальной проекции. Такая проекция удобна для общего обозрения всей поверхности Земного шара при взгляде с экватора.

Выбор вида проекции способен преобразовать вид изображения. Для иллюстрации этого на рис. 9.29 представлена карта мира в иной проекции – азимутальной проекции Ламберта (LambertAzimuthal). В таком виде получается прекрасный вид на Землю со стороны северного полюса.

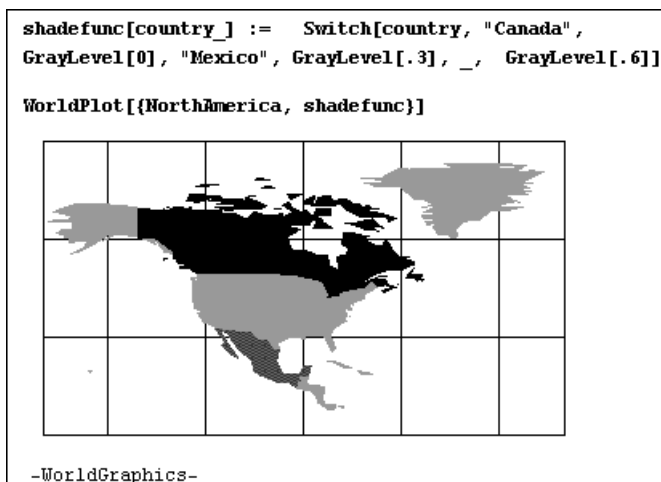


Рис. 9.27. Карта Америки с выделенной Канадой

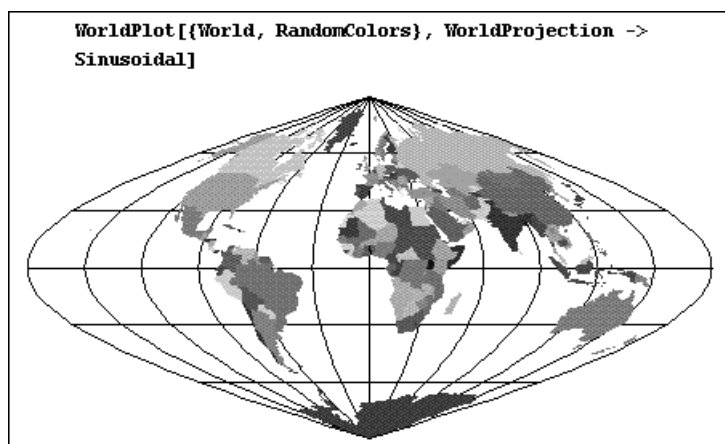


Рис. 9.28. Вид на Земной шар при синусоидальной проекции

Еще один пример (с цилиндрической проекцией Ламберта) представлен на рис. 9.30. Здесь показана карта Африки. Цилиндрическая проекция позволяет представить карту без геометрических искажений границ, обусловленных сферической поверхностью Земли. Обратите также внимание на технику окраски самого континента, фона и рамки.

В пакет WorldPlot включены функции преобразования углов:

- **ToMinutes[deg]** – преобразует градусы в минуты;
- **ToMinutes[{deg,min}]** – преобразует градусы и минуты в минуты с долями;

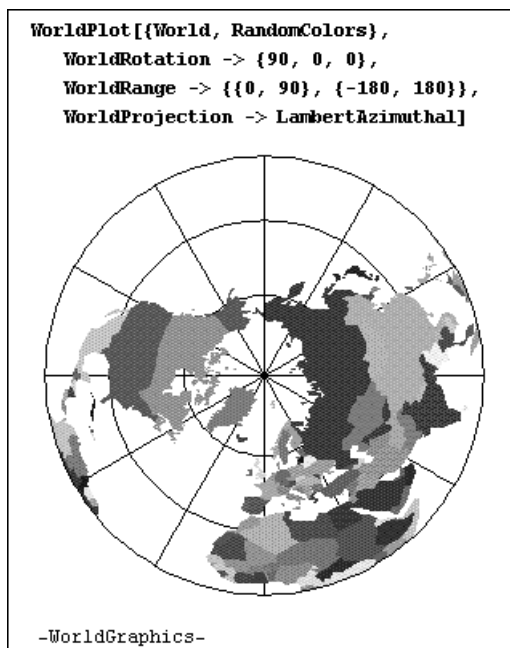


Рис. 9.29. Вид на Земной шар со стороны северного полюса при азимутальной проекции Ламберта

- **ToMinutes[{deg,min,sec}]** – преобразует градусы, минуты и секунды в минуты с долями.

Примеры преобразований представлены ниже:

```
ToMinutes[1]
```

```
60
```

```
ToMinutes[1,20]
```

```
80
```

```
ToMinutes[1,20,10]
```

```
481
```

```
6
```

```
N[%]
```

```
80.1667
```

9.5.11. Физические и химические данные

В подпакете `PhysicalConstants` определено несколько десятков наиболее употребительных *физических констант* [89]. Они представлены как размерные величины, т.е. помимо своего численного значения имеют единицы измерения. Физические константы вводятся своими полными символьными именами, например:

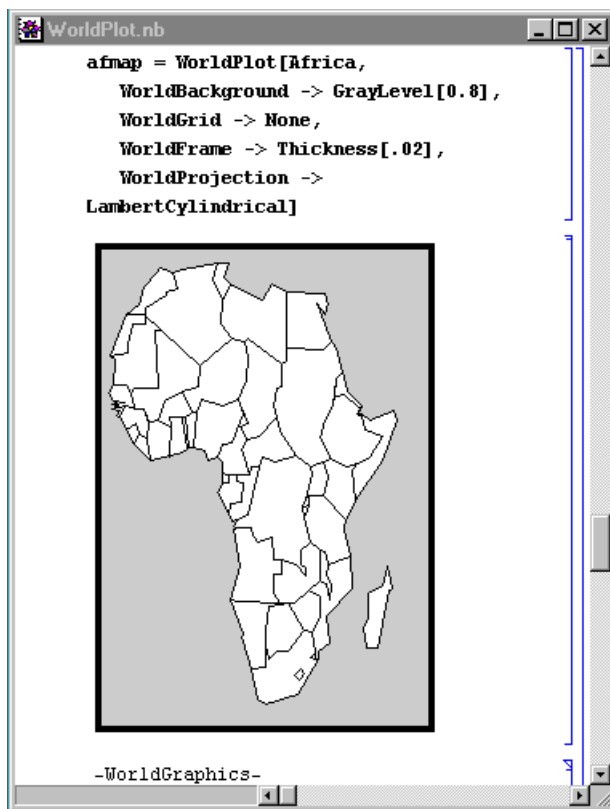


Рис. 9.30. Контурная карта Африки с цилиндрической поверхностью

Ввод и вывод

<<Miscellaneous`PhysicalConstants`

SpeedOfLight

$2.99792 \times 10^8 \text{ Meter}$

Second

SpeedOfLight AgeOfUniverse

$1.40902 \times 10^{26} \text{ Meter}$

ElectronMass

$9.10939 \times 10^{-21} \text{ Kilogram}$

AccelerationDueToGravity

9.80665 Meter

Second^2

Комментарий

Загрузка подпакета

Скорость света

Выражение с константами

Масса электрона

Ускорение свободного падения

Полные списки физических констант приведены в справочной базе данных по подпакету PhysicalConstants.

Для выполнения физических, химических и иных расчетов в Mathematica предусмотрена возможность работы с *размерными переменными*. Для этого база данных системы содержит символьные имена практически для всех единиц измерения (времени, массы, расстояния, температуры и т.д.). Данные о них можно найти в справочной базе данных подпакета Units. Там же имеются функции для перевода единиц измерений из одной системы размерных единиц в другую.

Начнем с функции:

Convert[old, newunits] – преобразование из старой формы в новую. Например:

```
<<Miscellaneous`Units`
Convert[12 Meter/Second, Mile/Hour]
26.8432 Mile
Hour
Convert[3 Kilo Meter / Hour, Inch / Minute]
1968.5 Inch
Minute
```

Для преобразования температуры служит функция:

ConvertTemperature[temp, oldunits, newunits] – преобразование температуры из старой формы в новую. Возможные единицы измерения температуры следующие: Celsius (Цельсия), Centigrade, Fahrenheit (Фаренгейта), Kelvin (Кельвина) и Rankine (Ренкина).

Пример преобразования температуры:

```
ConvertTemperature[20, Fahrenheit, Centigrade]
-6.66667
```

Наконец, имеется три широкопрофильных функции преобразования в различные *системы единиц*:

- **SI[expr]** – преобразует expr в Международную систему единиц SI;
- **MKS[expr]** – преобразует expr в систему единиц MKS (метр/килограмм/секунда);
- **CGS[expr]** – преобразует expr в систему единиц CGS (сантиметр/грамм/секунда).

Пример преобразования дан ниже:

```
SI[3 Atmosphere]
303975.Pascal
?Pascal
Pascal is the derived SI unit of pressure.
```

Помимо задания физических констант в пакете расширения Miscellaneous имеется три дополнительных подпакета: StandardAtmosphere (данные об атмосфере), ResonanceAbsorptionLines (построение резонансных линий поглощения) и BlackBodyRadiation. Ввиду узкой направленности входящих в них функций эти подпакеты подробно не описываются.

В подпакете ChemicalElements имеется ряд функций, позволяющих выявить *свойства химических элементов*. Начнем с функции без параметров **Elements**, выводящей список всех химических элементов (он полезен для знания англоязычных наименований элементов):

<<Miscellaneous`ChemicalElements`**Elements**

{Hydrogen, Helium, Lithium, Beryllium, Boron, Carbon, Nitrogen, Oxygen, Fluorine, Neon, Sodium, Magnesium, Aluminium, Silicon, Phosphorus, Sulfur, Chlorine, Argon, Potassium, Calcium, Scandium, Titanium, Vanadium, Chromium, Manganese, Iron, Cobalt, Nickel, Copper, Zinc, Gallium, Germanium, Arsenic, Selenium, Bromine, Krypton, Rubidium, Strontium, Yttrium, Zirconium, Niobium, Molybdenum, Technetium, Ruthenium, Rhodium, Palladium, Silver, Cadmium, Indium, Tin, Antimony, Tellurium, Iodine, Xenon, Caesium, Barium, Lanthanum, Cerium, Praseodymium, Neodymium, Promethium, Samarium, Europium, Gadolinium, Terbium, Dysprosium, Holmium, Erbium, Thulium, Ytterbium, Lutetium, Hafnium, Tantalum, Tungsten, Rhenium, Osmium, Iridium, Platinum, Gold, Mercury, Thallium, Lead, Bismuth, Polonium, Astatine, Radon, Francium, Radium, Actinium, Thorium, Protactinium, Uranium, Neptunium, Plutonium, Americium, Curium, Berkelium, Californium, Einsteinium, Fermium, Mendelevium, Nobelium, Lawrencium, Rutherfordium, Dubnium, Seaborgium, Bohrium, Hassium, Meitnerium, Ununnilium, Unununium, Ununbium}

Для выявления свойств заданного элемента **element** служат следующие функции:

- **Abbreviation[element]** – возвращает стандартную аббревиатуру элемента;
- **AtomicNumber[element]** – возвращает атомный номер элемента;
- **AtomicWeight[element]** – возвращает атомный вес элемента;
- **StableIsotopes[element]** – дает лист стабильных изотопов элемента;
- **Elements** – возвращает лист с химическими элементами.

Примеры применения этих функций:

Abbreviation[Wolfram]

W

AtomicNumber[Wolfram]

74

AtomicWeight[Wolfram]

183.84

StableIsotopes[Wolfram]

{180 , 182 , 183 , 184 , 186 }

Off[AtomicWeight::unstable]

Рисунок 9.31 показывает графическую зависимость отношения атомного веса к атомному номеру элементов.

Полезны также функции, возвращающие значения величин, определяющих физические свойства элементов:

- **MeltingPoint[element]** – температура в точке плавления (здесь и далее в градусах Кельвина);
- **BoilingPoint[element]** – температура в точке кипения;
- **HeatOfFusion[element]** – теплота плавления (килоджоуль/моль);
- **HeatOfVaporization[element]** – теплота парообразования (килоджоуль/моль);
- **Density[element]** – плотность в килограммах на кубический метр (при 298 градусах Кельвина);
- **ThermalConductivity[element]** – теплопроводность элемента;

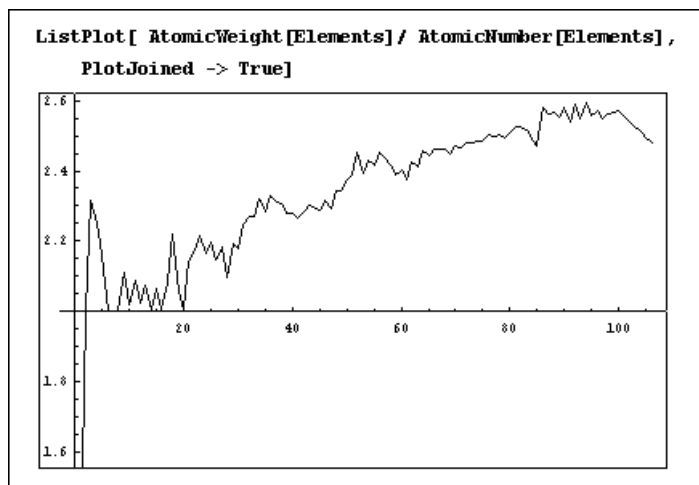


Рис. 9.31. Зависимость отношения атомного веса к атомному номеру элементов

- **ElectronConfiguration[element]** – конфигурация электронов в виде списка;
- **ElectronConfigurationFormat[element]** – конфигурация электронов в стандартной форме записи.

Используемая в размерных значениях этих функций величина моль является мерой количества вещества, численно равной $6.0221367 \cdot 10^{23}$ молекул вещества.

Определим свойства элемента – вольфрама (чувствуете намек на фамилию создателя системы Mathematica – S. Wolfram):

MeltingPoint[Wolfram]

3680. Kelvin

BoilingPoint[Wolfram]

5930. Kelvin

HeatOfFusion[Wolfram]

35.2 JouleKilo

Mole

HeatOfVaporization[Wolfram]

HeatOfVaporization[Tungsten]

Density[Wolfram]

19300. Kilogram

Meter³

ThermalConductivity[Wolfram]

174. Watt

KelvinMeter

ElectronConfiguration[Wolfram]

{{2 }, {2 , 6 }, {2 , 6 , 10 }, {2 , 6 , 10 , 14 }, {2 , 6 , 4 }, {2 }}

ElectronConfigurationFormat[Wolfram]

(1s² 2s²2p⁶ 3s²3p⁶3d¹⁰ 4s²4p⁶4d¹⁰4f¹⁴ 5s²5p⁶5d⁴ 6s²)

Вольфрам – один из самых тугоплавких элементов в природе. Недаром из него делают нити для ламп накаливания.

9.5.12. Задание данных только вещественного типа – *RealOnly*

В ряде случаев (как при вычислениях, так и при построении графиков) Mathematica сообщает о наличии у функций особых значений. Это хорошо иллюстрирует рис. 9.32, на котором предпринята попытка построения графика, казалось бы, простой функции $x^{1/3}$. Нетрудно заметить, что график в отрицательной области значений x не построен и перед построением неполного графика выдан целый букет предупреждающих сообщений.

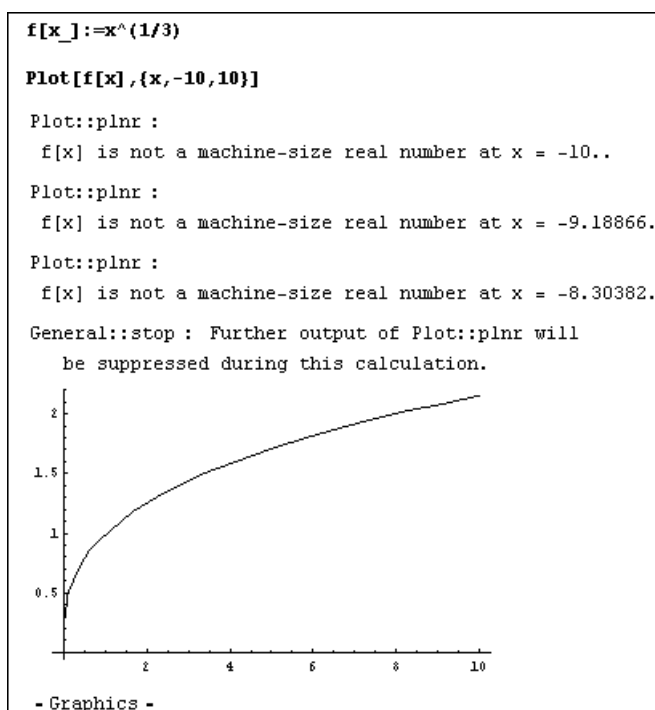


Рис. 9.32. Попытка построения графика функции $x^{1/3}$

Причина этой частичной неудачи в том, что в некоторых точках данная функция дает комплексные значения. Например:

```

(-8.0) ^ (1/3)
1. + 1.73205 I
  
```

Подпакет `RealOnly` не вводит никаких новых функций. Он просто превращает данные последующих вычислений в только вещественные. Так что после его загрузки построение графика указанной функции строится без каких-либо проблем (рис. 9.33).

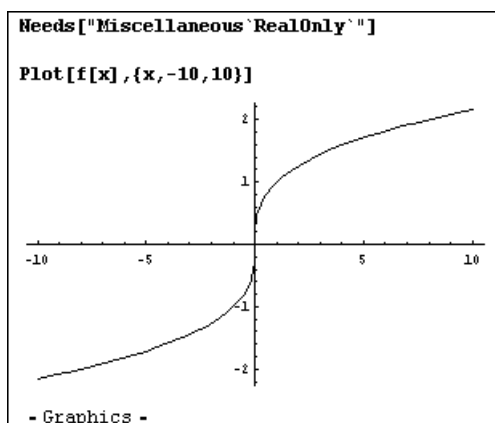


Рис. 9.33. Построение графика функции $x^{1/3}$ после загрузки подпакета `RealOnly`

Разумеется, подобное свойство нужно далеко не всегда, и при неумелом его применении способно привести к ошибочным результатам. Тем не менее, есть случаи (см. приведенный пример), когда оно полезно.

9.5.13. Пакет расширения с утилитами – *Utilities*

Пакет `Utilities` содержит ряд полезных утилит. Так, в подпакете `BinaryFiles` имеются типовые функции для работы с **бинарными файлами**:

- **OpenReadBinary["filename"]** – открытие файла для считывания бинарных данных;
- **OpenWriteBinary["filename"]** – открытие файла для записи бинарных данных;
- **OpenAppendBinary["filename"]** – открытие файла для добавления данных в конец;
- **ReadBinary[stream,type]** – считывает заданный бинарный файл в поток;
- **ReadBinary[stream,expr]** – считывает бинарный файл, представляющий выражение `expr` в поток.

Функции

ReadListBinary[filename,type] **ReadListBinary[stream,type,n]**
ReadListBinary[stream,type]

оперируют с данными в виде списков, а функция **WriteBinary[stream,data]** записывает данные в поток в бинарной форме. Примеры применения этих функций представлены ниже:

```
<< Utilities`BinaryFiles`
data = N[Table[10^n, {n, -10, 10}]]
{1.×10-10, 1.×10-9, 1.×10-8, 1.×10-7, 1.×10-6, 0.00001,
 0.0001, 0.001, 0.01, 0.1, 1., 10., 100., 1000., 10000.,
 100000., 1.×106, 1.×107, 1.×108, 1.×109, 1.×1010}
stream = OpenWriteBinary["binarytest"]
OutputStream[binarytest,12]
WriteBinary[stream, data]
Close[stream]
binarytest
ReadListBinary["binarytest", Double]
{1.×10-10, 1.×10-9, 1.×10-8, 1.×10-7, 1.×10-6, 0.00001,
 0.0001, 0.001, 0.01, 0.1, 1., 10., 100., 1000., 10000.,
 100000., 1.×106, 1.×107, 1.×108, 1.×109, 1.×1010}
ReadListBinary["binarytest", SignedInt16, 5]
{15835,31967,-9769,-16965,15889}
ReadListBinary["binarytest", Double + Double]
{1.1×10-9, 1.1×10-7, 0.000011, 0.0011, 0.11, 11., 1100.,
 110000., 1.1×107, 1.1×109, 1.×1010 + EndOfFile}
ToBytes[-34.3421435]
{192,65,43,203,91,179,132,253}
ToBytes[-34.3421435, CString]
ToBytes::cast: Warning: converting object of type Real to type
CString.
{45,51,52,46,51,52,50,49,0}
stream = OpenWriteBinary["binarytest2"]
OutputStream[binarytest2,17]
WriteBinary[stream, data, ByteConversion ->
  (ToBytes[#, RealConvert -> Single]&)]
Close[stream]
binarytest2
ReadListBinary["binarytest2", Double]
{5.74515×10-83, 5.15555×10-67, 7.42285×10-51,
 5.33097×10-35, 5.37764×10-19, 0.0078125, 5.27766×1013,
 5.59549×1029, 7.18642×1045, 4.9389×1061}
ReadListBinary["binarytest", SignedInt16, 5]
{15835,31967,-9769,-16965,15889}
```

Для конвертирования выражений **expr** в байтовый формат служат функции

ToBytes[expr] и **ToBytes[expr,type]**.

Например:

```
ToBytes[-34.3421435]
{192 ,65 ,43 ,203 ,91 ,179 ,132 ,253 }
```

```
ToBytes[-34.3421435, CString]
```

```
ToBytes::cast: Warning: converting object of type Real to type CString.
```

```
{45,51,52,46,51,52,50,49,0}
```

В широко распространенных графических системах AutoCAD используется формат файлов DXF. Подпакет DXF позволяет записывать графические объекты Mathematica в этом формате с помощью функции

```
WriteDXF["filename",graphics]
```

Здесь filename – имя файла и graphics – имя предварительно созданного графического объекта. Применение данной функции вполне очевидно.

В ряде случаев возникает необходимость в фильтрации типа интегрирования. В подпакете FilterOptions имеется директива, позволяющая задать ряд опций фильтрации:

FilterOptions[symbol,opt1,opt2,...] – возвращает последовательность опций, действующих для symbol.

Рисунок 9.34 поясняет применение опции фильтрации для создания графической функции **PlotIntegrate**, строящей график интеграла от заданной функции. Показан также пример построения графика интеграла от функции $\cos[x]$ в интервале от 0 до 2π . Как и следовало ожидать, график функции очень близок к синусоиде.

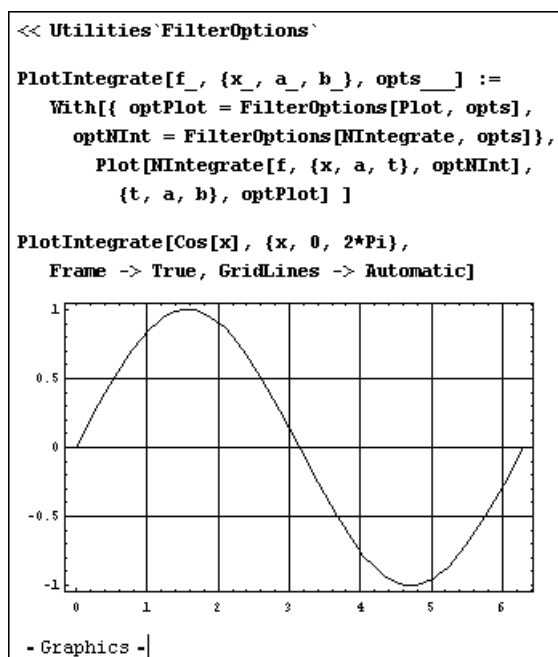


Рис. 9.34. Пример применения подпакета FilterOptions

Вы можете опробовать действие этой графической функции и на других примерах (желательно, чтобы интегрируемая функция не имела особенностей в пределах области построения графика).

Подпакет `MemoryConserve` в дополнение к утилите высвобождения памяти `Share[]` ядра содержит две директивы управления памятью:

- **On[MemoryConserve]** – включает автоматическое сжатие занимаемой системой памяти;
- **Off[MemoryConserve]** – отключает автоматическое сжатие памяти.

Данная утилита полезна лишь при использовании системы Mathematica на ПК с малым объемом оперативной памяти.

В подпакете `Package` имеется несколько функций, полезных при работе с пакетами расширения:

- **FindPackages[path]** – возвращает лист файлов с расширением `.m`, имеющих в каталоге `path`. Опция `FullPath->True` дает возврат имен всех каталогов;
- **FindPackages[path,pattern]** – возвращает лист файлов с расширением `.m`, имеющих в каталоге `path` и удовлетворяющих заданному образцу `pattern`;
- **Annotation[package]** – возвращает список слов – аннотацию пакета;
- **Annotation[package,keyword]** – возвращает аннотацию пакета, связанную с `keyword`.

В подпакете `ShowTime` собраны средства для осуществления контроля над временем выполнения различных операций:

- **ShowTime[expr]** – выводит время выполнения операции `expr`;
- **On[ShowTime]** – включает вывод времени выполнения последовательности операций;
- **Off[ShowTime]** – выключает вывод времени выполнения последовательности операций.

Следующие примеры иллюстрируют применение этих средств:

```
<< Utilities`ShowTime`
NIntegrate[{x Exp[-x] Sin[x]}, {x, 0, Infinity}]
0. Second
{0.5}
Off[ShowTime]
0. Second
Null
ShowTime[Sum[1/n, {n, 1, 9999}]];
0.49 Second
```

Обратите внимание на то, что времена выполнения использованных в примерах выражений относятся к компьютеру, на котором примеры выполнялись. Для других компьютеров времена будут другими. Контроль над временем исполнения операций – важная часть отладки высокоэффективных программ и программных модулей.

9.6. Данные о других средствах расширения

Читателю, желающему лучше изучить возможности систем Mathematica, полезно ознакомиться с некоторыми примерами применения этих систем. В Интернете по адресу <http://www.exponenta.ru/soft/Mathemat/dyakonov/nb1/nb1.asp> можно найти 10 ноутбуков автора для Mathematica 4/5. Эти ноутбуки посвящены следующим темам:

1. Интегралы и Mathematica.
2. Интеграл Дюамеля (расчет переходных процессов).
3. Метод 5-ти ординат (спектральный анализ).
4. Расчет коэффициентов Берга (спектральный анализ).
5. Имитация, практический спектральный анализ и синтез сигналов.
6. Анализ и синтез сигналов с линейной интерполяцией между узлами.
7. Вычисление амплитудно-частотной и фазо-частотной характеристик линейной системы по ее переходной характеристике.

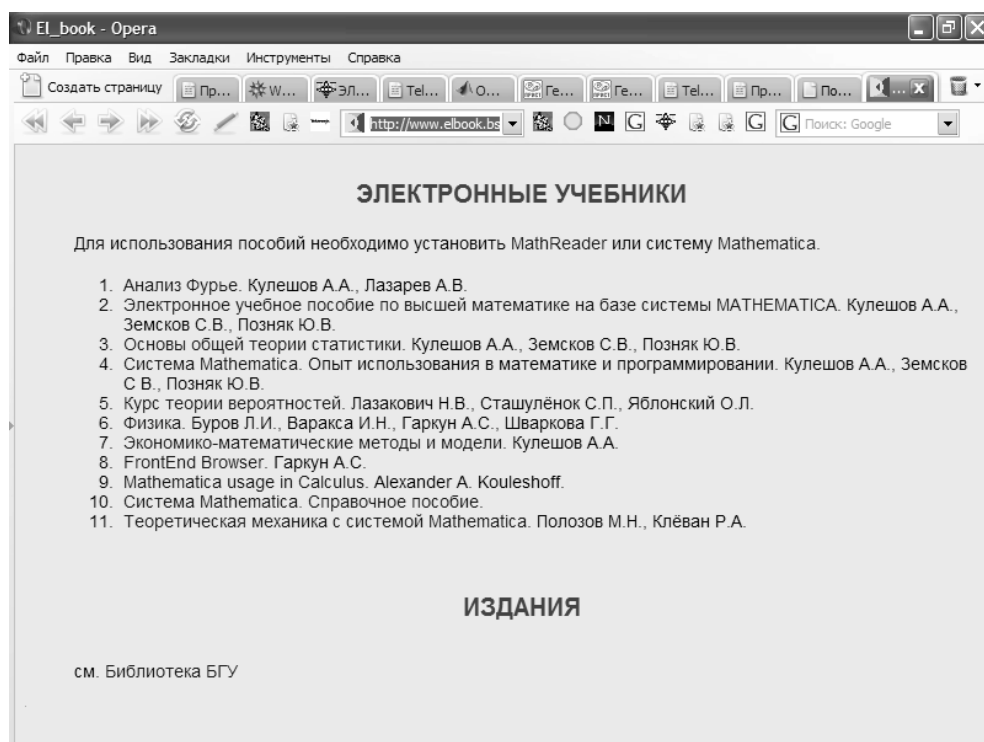


Рис. 9.35. Интернет-страница с перечнем учебников, созданных в Белорусском государственном университете

8. Анализ статических вольтамперных характеристик диодов и транзисторов с учетом лавинного пробоя.
9. Расчет импульсов релаксатора на лавинном транзисторе.
10. Моделирование цепи на туннельном диоде.

К сожалению, при использовании этих ноутбуков в среде Mathematica 5.1/5.2/6 могут возникнуть проблемы с русскоязычными комментариями из-за выбора по умолчанию различных наборов шрифтов. Полное описание этих ноутбуков есть в книге [29].

По адресу http://www.elbook.bsu.by/PRODUCTS/elbook_list.html можно найти серию электронных учебников в среде системы Mathematica, созданных в Белорусском государственном университете. Их состав демонстрирует Интернет-страница, показанная на рис. 9.35.

Во время визита в Россию представителя фирмы Wolfram Research, Inc. Александра Павлыка на семинаре по системе Mathematica был продемонстрирован ноутбук с описанием основных возможностей системы Mathematica 5.2 (рис. 9.36). С помощью этого ноутбука (его файл был предоставлен всем желающим, включая автора данной книги) можно познакомиться с общими возможностями системы Mathematica 5. Ноутбук является типичным примером проекта интерактивного

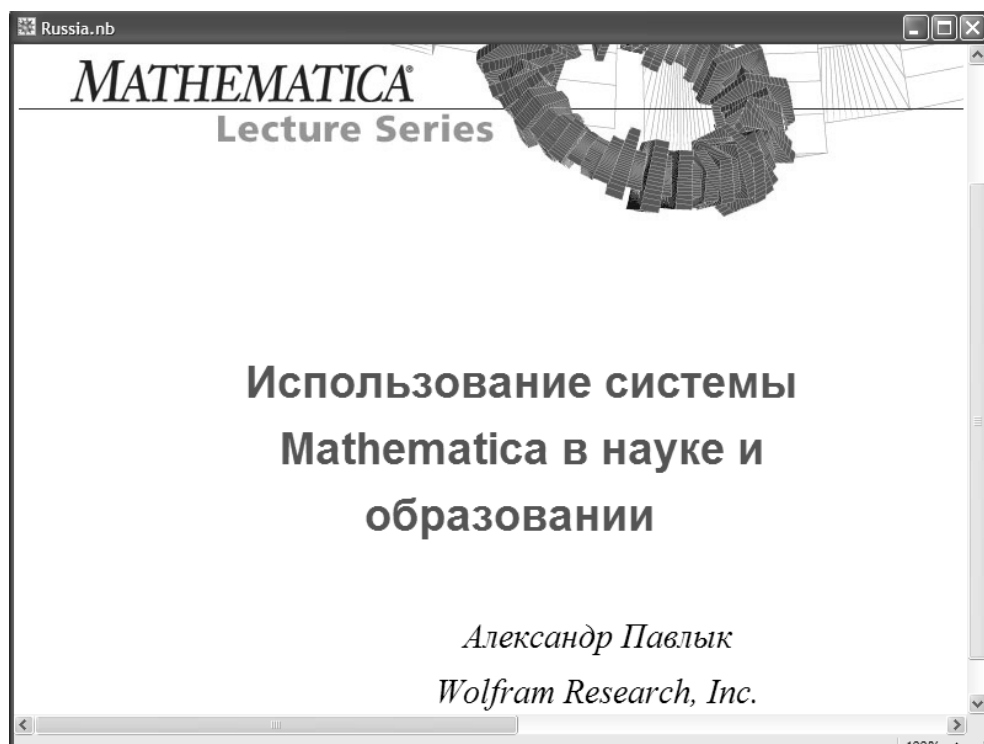


Рис. 9.36. Заглавная страница ноутбука Александра Павлыка

учебного пособия среднего размера, выполненного полностью в среде системы Mathematica 6.

В заключении стоит отметить, что ныне на сайте разработчика системы Mathematica (www.wolfram.com) размещено огромное число пакетов расширения, библиотек и ноутбуков, демонстрирующих поистине неисчерпаемые возможности этой мощной и популярной системы. Они дают прекрасные и вполне реальные примеры программирования в ее среде.

Стоит особо отметить внешние расширения системы Mathematica, называемые пакетами применения (Applications Packages), которые реализуются коммерческим путем корпорацией Wolfram Research, Inc. Их возможности у нас плохо известны. Информацию о пакетах применения (рис. 9.37), реализуемых фирмой, можно найти по адресу http://www.wolfram.com/products/field_specific.html.

Все пакеты применения англоязычные. В [29] описаны на русском языке следующие внешние пакеты расширения:

- **Digital Image Processing** – цифровая обработка изображений (загрузка, представление и запись изображений в различных форматах, точечные, геометрические и морфологические операции, вычисление площадей отдельных участков изображений, сегментация, математические преобразования и цифровая фильтрация изображений);

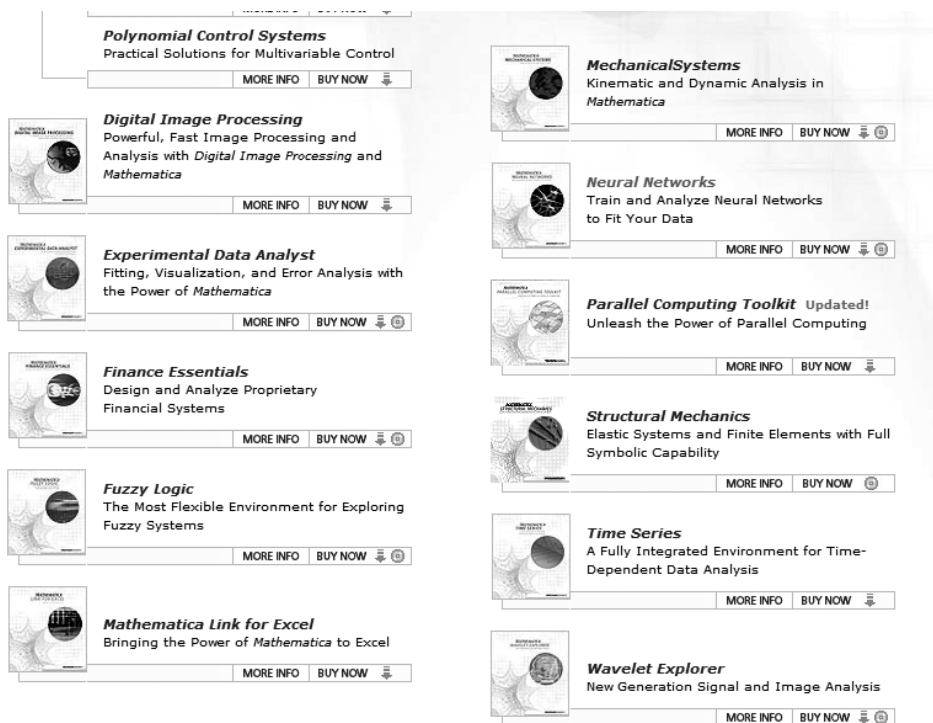


Рис. 9.37. Страница с указанием доступных для приобретения пакетов применения

- **Signal of Systems** – сигналы и системы (представление и обработка сигналов, интегральные преобразования сигналов, спектральный анализ, в том числе оконный, сигналов, фильтрация сигналов и проектирование фильтров, свертка и корреляция сигналов, различные утилиты, анализ и идентификация систем);
- **Wavelets Explorer** – работа с вейвлетами (основы теории вейвлетов, задание вейвлетов и техника их визуализации, вейвлет-преобразования, вейвлет-обработка и фильтрация сигналов и изображение, приближение функций вейвлетами, компрессия и декомпрессия сигналов, включая звуковые, и изображений с помощью вейвлетов).

Список литературы

1. Глушков В. М., Бондарчук В. Г., Гривченко Т. А. Аналитик — алгоритмический язык для описания процессов с использованием аналитических преобразований // Кибернетика. — 1971. — № 3.
2. Дьяконов В. П. Справочник по расчетам на микрокалькуляторах. Издание третье, дополненное и переработанное. — М.: Наука, Физматлит, 1989.
3. Дьяконов В. П. Справочник по алгоритмам и программам на языке бейсик для персональных ЭВМ. — М.: Наука, Физматлит, 1987/1989.
4. Дэвенпорт Дж., Сирэ И., Турнье Э. Компьютерная алгебра / пер. с франц. Е. В. Панкратьева; под ред. А. В. Михалева. — М.: Мир, 1991.
5. Акритас А. Основы компьютерной алгебры / пер. с англ. Е. В. Панкратьева. — М.: Мир, 1994.
6. Грэхэм Р., Кнут Д., Поташник О. Конкретная математика. Основание информатики / пер. с англ.; под ред. А. В. Ходулева. — М.: Мир, 1998.
7. Дьяконов В. П. Современные зарубежные микрокалькуляторы. — М.: СОЛОН-Р, 2002.
8. Дьяконов В. П. Компьютерная математика. Теория и практика. — М.: Нолидж, 2001.
9. Дьяконов В. П. Справочник по применению системы Eureka. — М.: Наука, Физматлит, 1993.
10. Дьяконов В. П. Система MathCAD: Справочник. — М.: Радио и связь, 1993.
11. Дьяконов В. П. Mathcad 11/12/13 в математике: Справочник. — М.: Горячая линия — Телеком, 2007.
12. Дьяконов В. П. Справочник по применению системы PC MatLAB. — М.: Наука, Физматлит, 1993.
13. Дьяконов В. П. MATLAB 6.5 SP1/7.0 + Simulink 5/6. Основы применения. — М.: СОЛОН-Пресс, 2005.
14. Дьяконов В. П. MATLAB 6.5 SP1/7.0 + Simulink 5/6 в математике и моделировании. — М.: СОЛОН-Пресс, 2005.
15. Дьяконов В. П. MATLAB 6.5 SP1/7.0 + Simulink 5/6. Обработка сигналов и проектирование фильтров. — М.: СОЛОН-Пресс, 2005.
16. Дьяконов В. П. MATLAB 6.5 SP1/7.0 + Simulink 5/6. Работа с изображениями и видеопотоками. — М.: СОЛОН-Пресс, 2005.
17. Дьяконов В. П., Круглов В. В. MATLAB 6.5 SP1/7/7 SP1/7 SP2 + Simulink 5/6. Инструменты искусственного интеллекта и биоинформатики. — М.: СОЛОН-Пресс, 2006.
18. Дьяконов В. П. Справочник по применению системы Derive. — М.: Наука, Физматлит, 1996.
19. Дьяконов В. П. Системы компьютерной алгебры DERIVE: Самоучитель. — М.: СОЛОН-Р, 2002.

20. Дьяконов В. П. Математическая система Maple V R3/R4/R4. – М.: Солон, 1998.
21. Дьяконов В. П. Maple 9.5/10 в математике, физике и образовании. – М.: Солон, 2006.
22. Дьяконов В. П. Как выбрать математическую систему? // Монитор-Аспект. – 1993. – № 2.
23. Дьяконов В. П. Mathematica 2.0 под MS-DOS и под Windows // Монитор-Аспект. – 1993. – № 2.
24. Дьяконов В. П. Mathematica 2.1 для Windows: от слов к делу! // Монитор-Аспект. – 1994. – № 2.
25. Дьяконов В. П. Mathematica 2.2 – на пути к совершенству // Монитор-Аспект. – 1995. – № 6.
26. Дьяконов В. П. Системы символьной математики Mathematica 2 и Mathematica 3. – М.: СК Пресс/PC Week, 1998.
27. Дьяконов В. П. Mathematica 3/4 с пакетами расширений. – М.: Нолидж, 2000.
28. Дьяконов В. П. Mathematica 4: Учебный курс. – СПб.: Питер, 2001.
29. Дьяконов В. П. Mathematica 4.1/4.2/5.0 в математических и научно-технических расчетах. – М.: СОЛОН-Пресс, 2004.
30. Половко А. М. Mathematica для студентов. – СПб.: БХВ Санкт-Петербург, 2007.
31. Шмидский Я. К. Mathematica 5% Самоучитель. – М.: Издательский дом «Вильямс», 2004.
32. Капустина Т. В. Компьютерная система Mathematica 3.0 для пользователей. – М.: Солон-Р, 1999.
33. Кристалинский Р. Е., Кристалинский В. Р. Преобразования Фурье и Лапласа в системах компьютерной математики. – М.: Горячая линия – Телеком, 2006.
34. Справочник по специальным функциям / под ред. М. Абрамовича и И. Стиган. – М.: Наука, Физматлит, 1979.
35. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. – М.: Наука, Физматлит, 1973.
36. Гандмахер Ф. Р. Теория матриц. 4-е изд. – М.: Наука, Физматлит, 1988.
37. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы. – М.: Наука, Физматлит, 1987.
38. Иванов В. В. Методы вычислений на ЭВМ. – Киев: Наукова думка, 1984.
39. Толстой Г. П. Ряды Фурье. – М.: Физматлит, 1980.
40. Жуков А. И. Метод Фурье в вычислительной математике. – М.: Физматлит, 1992.
41. Брейсуэлл Б. Преобразование Хартли. – М.: Мир, 1990.
42. Ланцош К. Практические методы прикладного анализа. – М.: Физматлит, 1961.
43. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. – М.: Мир, 1989.
44. Прокис Дж. Цифровая связь. – М.: Радио и связь, 2000.

45. Зернов Н. В., Карпов В. Г. Теория радиотехнических цепей. – Л.: Энергия, 1972.
46. Баскаков С. И. Радиотехнические цепи и сигналы: Учебник для вузов по специальности «Радиотехника». – М.: Высшая школа, 2000.
47. Добеши И. Десять лекций по вейвлетам / пер. с англ. Е. В. Мищенко; под ред. А. П. Петухова. – М.: РХД, 2001.
48. Чуи К. Введение в вейвлеты / пер. с англ.; под ред. Я. М. Жилейкина. – М.: Мир, 2001.
49. Воробьев В. И., Грибунин В. Г. Теория и практика вейвлет-преобразований. – СПб.: ВУС, 1999.
50. Новиков И. Я., Стечкин С. Б. Основные конструкции всплесков. Фундаментальная и прикладная математика. – Т. 3. – Вып. 4. – 1997.
51. Дьяконов В. П., Абраменкова И. В. MATLAB. Обработка сигналов и изображений. – СПб.: Питер, 2002.
52. Дьяконов В. П. Вейвлеты: От теории к практике. – М.: Солон-Р, 2002.
53. Дьяконов В. П., Абраменкова И. В., Пеньков А. А. Новые информационные технологии. Ч. 3. Основы математики и математическое моделирование. – Смоленск: СГПУ, 2003.
54. Дьяконов В. П. Мой Word 95/97. – М.: АСТ, 1998.
55. Дьяконов В. П. Internet: Настольная книга пользователя. 4-е изд. – М.: Солон-Р, 2002.
56. Шредер М. Фракталы, хаос, степенные законы: Миниатюры из бесконечного ряда. – Ижевск: НИЦ «Регулярная и хаотическая динамика», 2001.
57. Кирсанов М. Н. Графы в Maple. – М.: Физматлит, 2007.
58. Wolfram S. The Mathematica book. 4th ed. – Addison-Wesley, 1999.
59. Mathematica 4.0 Standard Add-on Packages. – Wolfram Research, Inc. Wolfram Media, Inc., 1999.
60. Maeder R. Programming for Mathematica. 3rd ed. – Addison-Wesley, 1996.
61. Blachman N., Williams C. P. Mathematica: A Practical Approach. 2nd ed. – Prentice Hall, 1999.
62. Wagon S. Mathematica in Action. 2nd ed. – TELOS/Springer-Verlag, 1999.
63. Gray J. W. Mastering Mathematica: Programming Methods and Applications. 2nd ed. – Academic Press, 1997.
64. Graphica 1. The Imaginary Made Real: The Art of Michael Trott. – Wolfram Media, 1999.
65. Graphica 2. The Pattern of Beauty: The Art of Igor Bakshee. – Wolfram Media, 1999.
66. Gass R. Mathematica for Scientists and Engineers: Using Mathematica to Do Science. – Prentice Hall, 1998.
67. Tam P. T. Physicist's Guide to Mathematica. – AP Professional, 1997.
68. Hibbard A. C., Levasseur K. M. Exploring Abstract Algebra with Mathematica. – TELOS/Springer-Verlag, 1999.
69. Abell M. L., Braselton J. P., Rafter J. A. Statistics with Mathematica. – Academic Press, 1999.

-
70. Denker M., Woyczynski W. A. Introductory Statistics and Random Phenomena: Uncertainty, Complexity, and Chaotic Behavior in Engineering and Science. – Birkhäuser, 1998.
 71. Gaylord R. J., Nishidate K. Modeling Nature: Cellular Automata Simulations with Mathematica. – TELOS/Springer-Verlag, 1996.
 72. DeJong M. L. Mathematica for Calculus-Based Physics. – Addison-Wesley, 1999.
 73. Trott M. The Mathematica GuideBook for Programming. – Springer-Verlag, 2004.
 74. Trott M. The Mathematica GuideBook for Graphics. – Springer-Verlag, 2004.
 75. Trott M. The Mathematica GuideBook for Symbolics. – Springer-Verlag, 2006.
 76. Trott M. The Mathematica GuideBook for Numerics. – Springer-Verlag, 2006.

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЪЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в Internet-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(495) 258-91-94, 258-91-95**; электронный адрес **books@alians-kniga.ru**.

Дьяконов Владимир Павлович

Математика 5.1/5.2/6

Программирование и математические вычисления

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru

Корректор *Галушкина А. В.*

Верстка *Чаннова А. А.*

Дизайн обложки *Мовчан А. Г.*

Подписано в печать 28.01.2007. Формат 70×100 ¹/₁₆.

Гарнитура «Петербург». Печать офсетная.

Усл. печ. л. 36. Тираж 1500 экз.

№

Web-сайт издательства: www.dmk-press.ru

Internet-магазин: www.abook.ru

Электронный адрес издательства: books@dmk-press.ru